MATHEMATICS AND PHYSICS ENGINEERING FINAL PROJECT

Hierarchical Risk Parity: portfolio optimization

Author:
Mikel Mercader Pérez

Supervisor:
Dr. Josep Masdemont

May 2021

Contents

Abstract

Striking the optimal balance between risk and return is at the core of financial investment theory. This paper analyzes whether Hierarchical Risk Parity algorithm, a novel portfolio optimization method, could outperform conventional models in certain dimensions. In a series of 3 tests run with a custom written code, the performance of HRP is compared to that of classical methods like Minimum Variance and Risk Parity in terms of expected return, volatility, Sharpe ratio, maximum drawdown and diversification ratio. In many of the cases analyzed, HRP showed superior results in expected return and in minimizing drawdown, but often at the expense of higher volatility and higher transaction costs. The paper finishes with a review of the alternative methods being currently used by the emerging financial robo-advisory industry, and a qualitative discussion about the future opportunities for HRP application in this context.

keywords: Finance, HRP, Markowitz, robo-advisory, portfolio optimization. 91-08

Acknowledgements

I would like to thank Gerard Alba for all the ideas, information and tools he has provided me during this work. This project would not have been possible without his help.

I would also like to express my most sincere gratitude to my good friend Aina, whose support has accompanied me during all this journey.

Introduction

Investing in corporate securities may be profitable, but one should not underestimate the risk of investing on a single security. Risk arises when there is a possibility of variation of the return. Holding more than one security at a time enables investors to spread their risks: even if some of the securities incur losses, the investor hopes that a better performance of the rest might protect them from an extreme loss. This is the underlying rationale behind the concept of a financial portfolio. We are defining the idea of a *portfolio*, that is, the composite set of ownership rights to financial assets in which the investor wishes to invest.

Although this idea was formalized in 1952 by Harry Markowitz, it was not an strange concept previously in history. In the 16-th century, in the play *The Merchant of Venice*, Shakespeare proves to have some intuition about the idea of diversifying one's investments. As spoken by merchant Antonio in Act I, Scene I:

My venturies are not in one bottom trusted,
Not to one place, nor is my whole estate
Upon the fortune of this present year;
Therefore, my merchandise makes me not sad.

It seems that Shakespeare understood not only the concept of diversification, but also, to some extent, the effect of covariance.

A more recent example is the series of the Wiesenberger's annual reports in *Investment Companies*, beginning in 1941. These reports analyzed firms that held a large number of securities to invest and provided some diversification for their customers. At the same time, those were modeled after the investments trusts of Scotland and England from the 19-th century.

In 1938, John Burr Williams wrote a book called *The theory of Investment value* that captured the most advanced thinking of the time: the *dividend discount model*, which predicts the price of a company's stock based on the theory that its present-day price is equivalent to the expected value of the sum of all of its future dividend payments when discounted back to their present value. During those times, financial information from the companies was difficult to obtain and the loose ways of the market, even after the tightened regulations following the Great Depression, generated the impression that investing was a form of gambling for the wealthy. Despite of that, professional investment managers like Benjamin Graham made huge progress by first getting accurate information and then analyzing correctly before making any investments.

What was lacking prior to 1952 was a formalized theory that covered the effects of diversification when the assets are correlated, distinguished efficient and inefficient portfolios, and analyzed risk-return tradeoffs.

Harry Markowitz, at the time a graduate student in operations research, devoted his doctoral thesis to the study of the investment market. Upon reading John Burr William's book, he was struck by the fact that no consideration was given to the risk of a particular investment. This inspired him to write *Portfolio Selection*, an article published in 1952 in the *Journal of Finance*. However, the work languished for a decade, mostly because only a minor part of the article contained text explanations, while the vast majority was dominated by graphs and mathematical proofs, which made it difficult to understand for a non academic audience.

In this work, we will in first place review the fundamentals of the Markowitz model, providing the basic definitions and the tool set needed to better understand his work. We will then aim to go beyond the conventional methods of portfolio optimization, and go further by exploiting the power of Data Analysis. For that purpose, we will also study a new method of optimization, namely *Hierarchical Risk Parity*, and we will also review some of the methods used by the emerging *Robo-advisory* services industry. In order to do that, our work is structured as follows:

- Chapter 1 provides with the basic definition for the Markowitz model and goes through the standard methods of portfolio optimization.
- Chapter 2 gives an in-depth explanation of Hierarchical Risk Parity, including code and practical examples comparing HRP with the standard methods.
- Chapter 3 explains the new subject of the Robo-Advisory, focusing on the case-study of Scalable Capital.

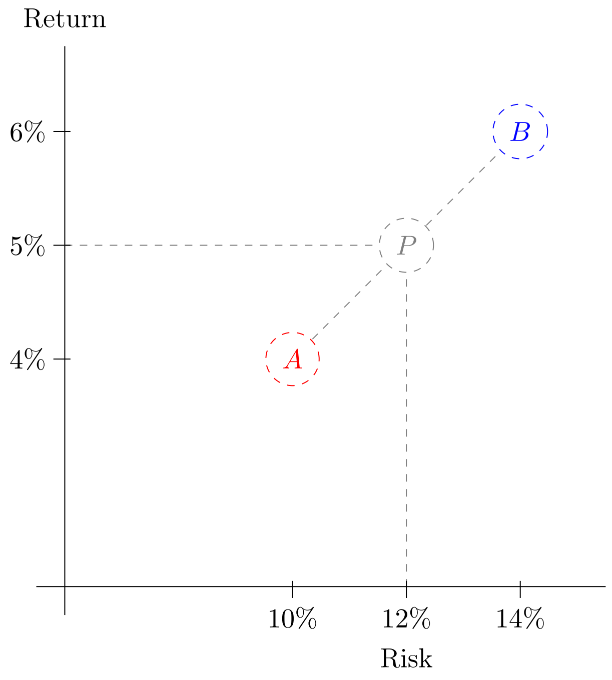
Chapter 1

Portfolio optimization

1.1 Motivation

Why should one construct portfolios instead of looking for individual assets to invest in? We can give a simple example that will show the main reason why constructing portfolios is useful. In this example we will use two properties of assets that we will properly define later: the expected return, which is essentially the expected value that the asset will have at the end of a time period compared to its current one, and the risk or volatility, which has to do with the variance in this return. Suppose now that we have two different assets, asset A with an expected return of 4% and a volatility of 10% and asset B with an expected return of 6% and a volatility of 14%.

Now we consider all the possible portfolios that can be made combining asset A and B. It is clear that the properties of the whole portfolio will resemble those of an individual asset more as we increase the weight of given asset, but how exactly can they be computed? Lets say our portfolio consists of 50% of asset A and 50% of asset B, then one might guess that its expected return is $\frac{4+6}{2} = 5\%$ and its volatility is $\frac{10+14}{2} = 12\%$.



Even though this seems like the most intuitive approach, here we have not taken into account an important concept, the correlation between the two assets. The expected return is actually correct and can be computed with a simple weighted mean of the assets' returns, but it turns out that when we properly compute the risk of our portfolio it will generally be lower than the one we expected looking at the mean of the assets'. This is directly related to the correlation between them: if the assets are largely uncorrelated we will be able to reduce significantly the volatility of their combinations, while the more correlated they are the more reality resembles our previous naive approach. Figure 1 shows the actual behaviour of all portfolios that result from combining A and B in different proportions, assuming a correlation of 0.4. Every point corresponds to a certain composition, 100% A and 0% B, 90% A and 10% B etc... The horizontal axis displays volatility while the vertical axis displays return.

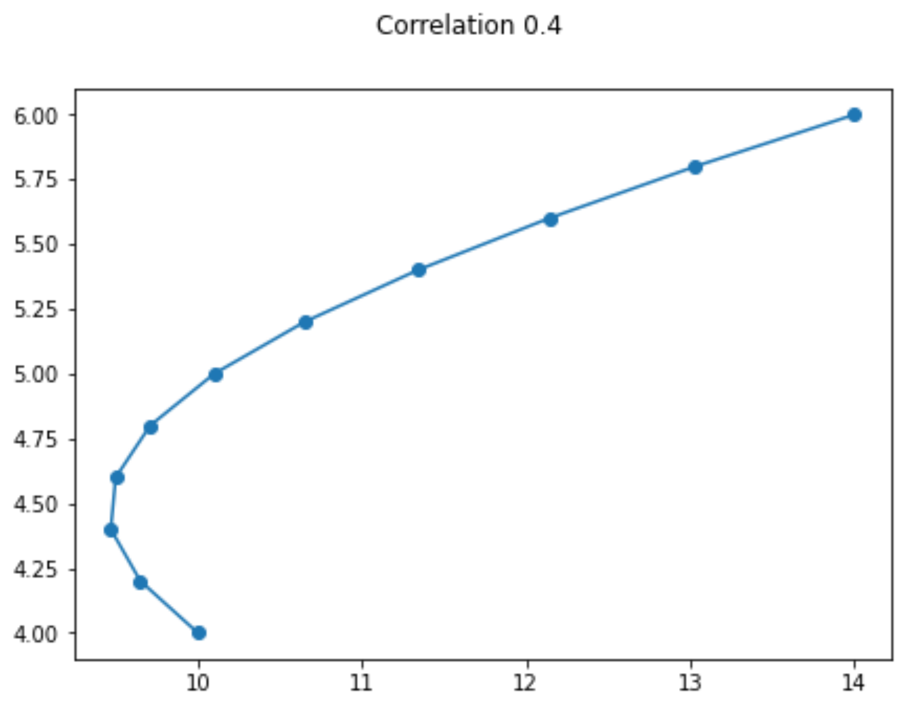


FIGURE 1.1: ghj

We can observe how the curve is on the left of the straight line that goes from A to B. If the correlation was lower we would see a more pronounced curvature, while if it was closer to 1 the curve would approach the straight line. The intuition behind this is that if an asset behaves in an opposite way of the other and we invest in both, when one of them presents losses the other is likely to present gains and cover for those losses, therefore reducing the total risk of the investment.

We have seen this basic example for two assets, but how would this graphic look for three or more? We will discover that for an arbitrary number of assets instead of a curve we have a region that represents all the possible portfolios. Out of this region, we will only be interested on its edge on the left, which we will call efficient frontier. This is because for a given return we will only be interested in the portfolio that offers the lowest risk. This way, we obtain a curve similar to the one from our simple example. This phenomenon will be explained in a more detailed manner once we have formally presented Markowitz's model.

1.2 Markowitz model: definitions

Suppose we have a certain asset for which we can have a historical price information. For example we can look at an asset's historical price on a daily, weekly or monthly basis. We can construct a vector P of all the prices, being P_t the price at time t. We can look at the monthly price of an index like IBEX 35 (IBEX 35 is the benchmark stockmarket index of the Bolsa de Madrid, Spain's principal stock exchange) to obtain a vector which we can represent in a graphic:

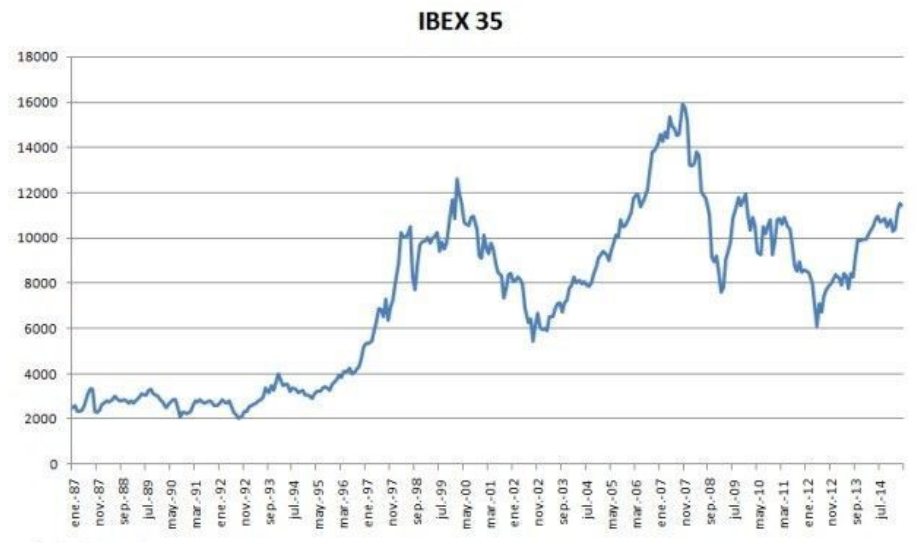


FIGURE 1.2: Price of ibex35 in time

Definition 1.2.1. We define the *return* from time period t to time period $t+1$ as

$$R_{t,t+1} = \frac{P_{t+1}}{P_t}.$$

The return represents the change in the value of an asset and tells us how much it grows or decreases. An asset the value of which grows from 10 to 12 has a return of 1.2 or 120%. We could also say that the return is 0.2 or 20%, but for the purpose of making computation easier it is sometimes more practical to define returns in the previous manner.

With this definition if an asset has a return $R_{1,2}$ from time period 1 to time period 2 and a return $R_{2,3}$ from time period 2 to time period 3 we can calculate the total return from time period 1 to time period 3 simply as

$$R_{1,3} = R_{1,2} \cdot R_{2,3}.$$

From the original vector of prices of size N we can obtain a vector of returns of size N-1.

When doing financial analysis returns have much more interest than the price itself, it is the relative growth of an investment that will generate profit or losses, independently of the particular price of the assets its composed of. That is why from now on we will work with returns instead of prices to build our model, and in particular we will look at the average return, $\bar{R}$, of every asset.

Definition 1.2.2. The *volatility* of an asset, denoted by σ, is the standard deviation of the returns, given by the following formula

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(R_i - \bar{R})^2.$$

Where $\bar{R}$ is the arithmetic mean of the returns or the average return.

The price of an asset with low volatility presents a steady and consistent growth through the different time periods, while the price of an asset with high volatility will present large variations in its evolution. One of the main goals of constructing a portfolio is to reduce this volatility to the minimum possible.

Definition 1.2.3. A *portfolio* is a financial product that results of the combination of multiple assets. If we have N assets to choose from, a portfolio consisting of those can be expressed as a vector of weights $W = (w_1, w_2, ...w_N)$ where each w_i represents what fraction of the portfolio consists of the asset i. Therefore, we have that $\sum_{i=1}^{N} w_i = 1$.

Up until now, for a portfolio consisting of N assets we have the following magnitudes:

- A vector $W = (w_1, w_2, ...w_N)$ of weights.
- A vector $r = (r_1, r_2, ...r_N)$ of the average returns of each asset. Notice that until now we had used $\bar{R}$ to refer to these average returns, but for simplicity we will just call them r_i from now on.
- A vector $\sigma = (\sigma_1, \sigma_2, ...\sigma_N)$ of the volatilities of each asset.

Just as for every individual asset, we can compute an expected return and a volatility for the whole portfolio.

Definition 1.2.4. We compute the *return of our portfolio* as

$$r_p = r^T w = \sum_{i=1}^{N} r_i w_i.$$

The return of the portfolio is simply a weighted average of the individual returns, however computing its volatility requires some extra steps. This is because we can not look only at the variance or volatility of each asset individually, but we also have to look at the correlations between them. To do this we will first need to store all the information regarding the variances and correlations in a covariance matrix.

Definition 1.2.5. The *covariance matrix* Σ is a $N \times N$ matrix computed as

$$\Sigma = (\sigma_{ij})_{N\times N} = \sigma_i \sigma_j \rho_{ij}$$

where ρ_{ij} is the correlation between the returns of asset i and asset j. It can be computed as

$$\rho_{ij} = \frac{\sum^k (r_{i,k} - r_i)(r_{j,k} - r_j)}{\sqrt{\sum^k (r_{i,k} - r_i)^2 \sum^k (r_{j,k} - r_j)^2}}$$

where r_i and r_j are the average returns for assets i and j and $r_{i,k}$ is the return of asset i at time period k.

Definition 1.2.6. The *volatility of the portfolio* is

$$\sigma_p^2 = Var(r_p) = w^T \Sigma w = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij}$$

This is the volatility of our portfolio as a whole, which, taking into account the correlations between assets, is different than the average of their volatilities.

To illustrate how we use all of this in a practical way, lets compute all these values in the initial example we used as motivation. Lets recall our case:

$$\text{Asset A:} \quad r_a = 4\% \quad \sigma_a = 10\%$$

$$\text{Asset B:} \quad r_b = 6\% \quad \sigma_b = 14\%$$

$$\text{Correlation:} \quad \rho_{ij} = 0.4$$

How would a portfolio consisting of 50% A and 50% B look?

$$w = (w_1, w_2) = (0.5, 0.5)$$

$$r_p = r^T w = 0.5 \cdot 10\% + 0.5 \cdot 14\% = 12\%$$

$$\Sigma = \begin{pmatrix} 0.1 \cdot 0.1 \cdot 1 & 0.1 \cdot 0.14 \cdot 0.4 \\ 0.1 \cdot 0.14 \cdot 0.4 & 0.14 \cdot 0.14 \cdot 1 \end{pmatrix} = \begin{pmatrix} 0.01 & 0.0056 \\ 0.0056 & 0.0196 \end{pmatrix}$$

And its actual volatility is

$$\sigma_p^2 = w^T \Sigma w = 0.5 \cdot 0.5 \cdot 0.01 + 0.5 \cdot 0.5 \cdot 0.0056 + 0.5 \cdot 0.5 \cdot 0.0056 + 0.5 \cdot 0.5 \cdot 0.0196 = 0.102$$

$$\sigma_p = 0.101 = 10.1\%$$

This is lower than the average of the assets' volatilities, almost as low as A's. In fact, it is possible to make portfolio from these assets that has a volatility lower than any of the two. If we make a portfolio with weights 0.8 of A and 0.2 of B, we obtain a volatility $\sigma_p = 9.47$. In figure 1.1 we can see the volatilities for portfolios with weights $w = (1, 0)$ $w = (0.9, 0.1)$ $\ldots$ $w = (0.1, 0.9)$ $w = (0, 1)$. This example illustrates the usefulness of portfolio creation when it comes to reducing volatility.

Definition 1.2.7. The *sharpe ratio* is a magnitude used to evaluate the performance of a portfolio. It measures the return it provides per unit of risk and is defined as

$$\frac{R_p - R_f}{\sigma_p}.$$

R_f is the risk free return, meaning the return we can obtain from an investment that has virtually no risk, for example a U.S. Treasury bond. Subtracting it allows us to focus on the amount of return that does come with the assumption of some risk.

Definition 1.2.8. The *maximum drawdown* is a risk measure that represents the greatest loss that a financial product undergoes in a certain period of time. It does not need the Markowitz model to be defined, it can be computed using only the temporal price series of the product.

The drawdown in any point of time is defined as the difference between the current value and the previous maximum (divided my the maximum, to express it in relative terms). If we search for the largest of these values across the period, we obtain the maximum drawdown, a measure of the largest value downfall that has taken place.

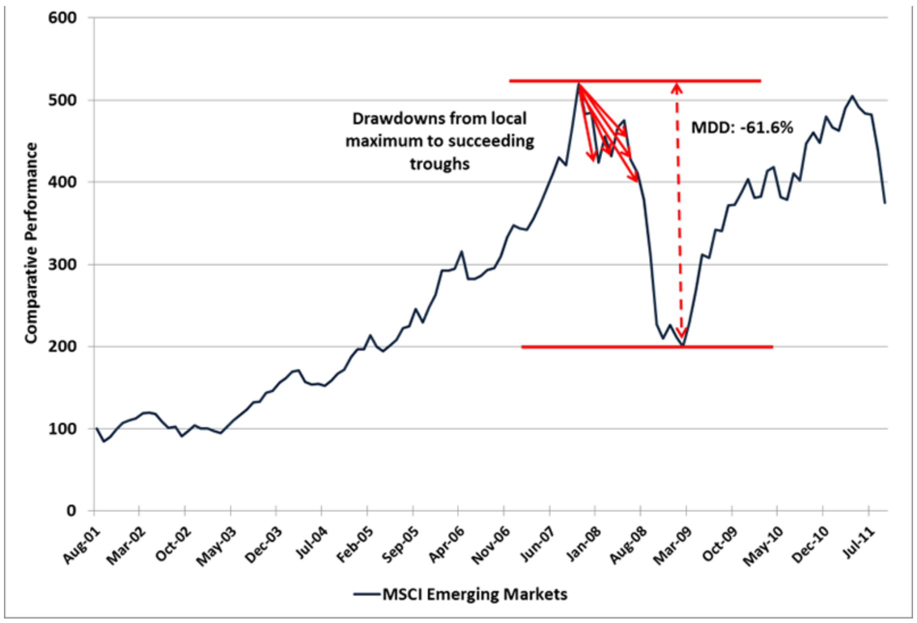


FIGURE 1.3: https://blogs.cfainstitute.org/investor/2013/02/12/sculpting-investment-portfolios-maximum-drawdown-and-optimal-portfolio-strategy/

An important advantage this kind of risk measure has over volatility is that it only measures downside risk, focusing only on possible losses, while volatility is affected by both unexpected gains and unexpected losses.

1.3 Efficient frontier

Now we know how all the possible portfolios made out of two assets look, but how is this generalized to N assets? When we have several assets we have more than one degree of freedom when choosing the weights and what was previously a line in the risk-return plane becomes a whole region. For example, we can illustrate the region of possible portfolios made by 3 assets by generating many portfolios with random weights and computing the return and volatility of each one.

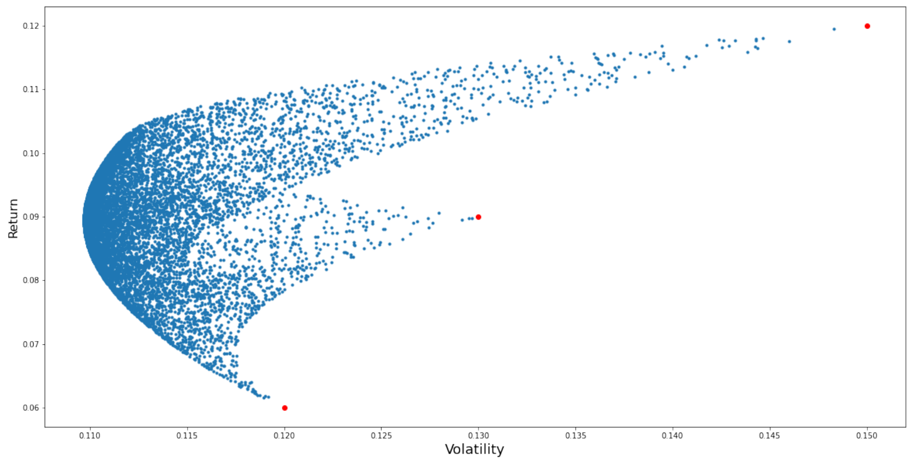


FIGURE 1.4: Possible portfolios generated by 3 assets.

In the figure 1.4 the original assets are represented by the red dots. However, of the infinite number of possible portfolios represented by this region, which ones are of our interest? It is not always easy to compare a pair of portfolios: take the ones in our initial example, portfolio A with a 4% return and 10% volatility and portfolio B with a return of 6% and 14% volatility. Different investors could choose either one and there are arguments to be made for both, depending on what your priority is, maximizing return or avoiding risk. However, if we had two assets both with 5% return and one with 10% volatility while the other had 15% volatility, everyone would agree that the first one is objectively superior since it offers the same return for less volatility. A way to see this is to think of **risk** as something that **has a cost**, meaning that more risk reduces the value of an asset. The same applies to portfolios: in the region we have generated there are several portfolios with the same expected return and different volatilities, of which only the one with the lowest volatility is of interest. This means that in the bigger picture we will only be looking at the left edge of the region of possible portfolios. The curve that this edge defines is called the *efficient frontier*.

Computing this efficient frontier is an optimization problem: for a given return we have to minimize the risk or volatility of the portfolio. The formulation of the problem is

$$\begin{aligned}
\textbf{Minimize:} \quad & \sigma_p^2 = w^T \Sigma w \\
\textbf{Subject to:} \quad & r^T w = r_p \text{ (a determined return)} \\
& \sum_{i=0}^{N} w_i = 1 \\
& w_i \geq 0 \ \forall i
\end{aligned}$$

We can solve this problem for returns ranging from the lowest one of all the assets to the highest one to plot the efficient frontier. Observe that this is a quadratic programming problem and therefore it can be solved efficiently. This way we can find the optimal portfolios for every return level according to Markowitz's model.

Once we have found this efficient frontier, however, it is not immediate to choose which particular portfolio is the best one. That is why many classical methods have been created that work inside Markowitz's model and to try to find a single most optimal portfolio. Here we will explain and discuss some of them.

1.4 Mean-variance optimization

In the Markowitz universe, Mean Variance optimization is based on the previously mentioned idea that risk has a cost. It uses a simple way of translating risk into units of return to find a portfolio that has a good trade-off between risk and return. One of the advantages of this method is that its easily customizable: we define a constant λ called risk aversion factor. This constant is positive and it will be greater the more we prioritize avoiding risk, even if it implies sacrificing return. The optimization problem looks like this:

$$\begin{aligned}
&\textbf{Maximize:} && w^T r - \tfrac{\lambda}{2} w^T \Sigma w = r_p - \tfrac{\lambda}{2}\sigma_p \\
&\textbf{Subject to:} && \sum_{i=0}^{N} w_i = 1 \\
& && w_i \geq 0 \ \forall i
\end{aligned}$$

The function that we are trying to maximize depends on both return and risk, return increases it while risk decreases it. The greater λ is the more relevant the risk term will be and the more emphasis the method will make in avoiding risk.

1.5 Risk parity

This method is based only on the volatilities described in Markowitz's model, it does not take returns into account. Its goal is to create a portfolio in which every asset contributes equally to the total risk. To do this we define the individual risk contribution of every asset in a manner that makes sense conceptually. As we know,

$$\sigma_p = \sigma_p(w) = w^T \Sigma w$$

is a homogeneous function and therefore we can apply Euler's theorem for homogeneous functions, obtaining

$$\begin{aligned}
&\sigma(w) = \sum_{i=1}^{N} \sigma_i(w), \text{ where} \\
&\sigma_i(w) = w_i \cdot \partial_{w_i} \sigma(w) = \frac{w_i (\Sigma w)_i}{\sqrt{w' \Sigma w}} \text{ is the risk contribution of asset } i.
\end{aligned}$$

If we want each asset's contribution to be equal, we must impose $\sigma_i(w) = \sigma_j(w) \ \forall i, j$ or equally $\sigma_i(w) = \frac{\sigma(w)}{N} \ \forall i$ where N is the total number of assets. This problem can be solved by looking at the fixed point problem

$$w_i = \frac{\sigma(w)^2}{(\Sigma w)_i N}$$

or solving the optimization problem given by

$$\min_{w} \sum_{i=1}^{N} \left[w_i - \frac{\sigma(w)^2}{(\Sigma w)_i N} \right]^2 .$$

1.6 Most diversified portfolio

This method is based on the assumption that for every asset the expected return should be proportional to its volatility, meaning that the sharpe ratio of all portfolios is equal. Within this assumption we forget about the actual return and the previous portfolio sharpe ratio we calculated is changed to the diversification ratio, DR:

$$\mathrm{DR}(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{\sigma}}{\sigma_p} = \frac{\sum_{i=1}^{N} (w_i \sigma_i)}{\sqrt{w' \Sigma w}}.$$

Following this concept, the method consists on maximizing the function from the mean-variance method but replacing the returns by the individual volatilities since we have assumed that $r \propto \sigma$:

$$\max_{w} \left[\boldsymbol{w}^T \boldsymbol{r} - \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w} \right] = \max_{\boldsymbol{w}} \left[\boldsymbol{w}^T \boldsymbol{\sigma} - \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w} \right].$$

Maximizing this function is equivalent to maximizing the diversification ratio. Conceptually, it can be interpreted as an attempt to maximize the reduction in risk that the portfolio has when taking into account the effect of the correlations, in comparison to the naive sum of weighted individual volatilities.

In the next part of this work we will introduce a new method, hierarchical risk parity (HRP) and we will evaluate the performance of its performance in comparison to the previously mentioned classical methods.

Chapter 2

Hierarchical Risk Parity

2.1 Tree clustering

The main difference between this method and the classical ones is the first step of its process. Its name is hierarchical risk parity because the first thing we are going to do is organize the financial assets in clusters. Subsequently, every cluster will be subdivided in smaller clusters too, until we get down to the individual assets. This process will therefore define a hierarchy, a tree that represents the recursive process in which we divide the universe of assets in clusters that will then be divided too. We will explain in detail how this clustering is performed.

The idea is that clusters will contain assets that are "similar". The key here is how to define the similarity of two assets. To do this it is natural to look at the correlation between them, the higher the similarity is the higher the correlation will be. This way we can define a distance between pairs of assets in relation to their correlation:

$$d_{i,j} = \sqrt{0.5 \cdot (1 - \rho_{i,j})}.$$

Perfectly correlated assets will have the minimum distance of 0 while the most uncorrelated, or negatively correlated assets will have a distance close to 1.

This way we can obtain a distance matrix from the correlation matrix by simply applying this definition to each of its elements. However this distance only looks at assets as pairs, it would be much more useful to define a distance that takes into account the role that each one of the two assets plays in the context of the universe of assets. For example, two assets A and B might not be particularly correlated, but maybe the relationship they have with the rest of the assets is similar, in the sense that if we take a random asset C it is likely that A and C are correlated in a similar way that B and C are. In this case we would want to consider A and B more similar than we initially thought. Using the initial distance d we can define a new distance $\bar{d}$ that applies this concept in the following way:

$$\bar{d}_{i,j} = \sqrt{\sum_{n=1}^{N} (d_{n,i} - d_{n,j})^2}.$$

This basically means that the new distance $\bar{d}_{i,j}$ between assets i and j is the euclidean distance between columns i and j of the original distance d matrix.

Once this distance is defined, the next step is to recursively form clusters. This way we will end up with a tree in which every cluster is made up of sub-clusters. When all our assets are still split, we look for the minimum distance in the distance matrix and combine its two assets into a cluster. For example, suppose our distance matrix is the following:

	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

TABLE 2.1: aaaa

Here the two closest nodes are a and b with a distance of 17. Therefore we would combine a and b into a cluster (a, b). We update the distance matrix by removing the two previous nodes and adding one that represents the recently formed cluster. In this step we need to define a method to compute the distance between the new cluster and the other nodes (or, in the future, other clusters). One way to do this is with the nearest point algorithm.

$$\bar{d}_{A,B} = \min\left[\left\{\bar{d}_{i,j}\right\}_{i\in A, j\in B}\right]$$

where A and B are clusters (an asset can be seen as a cluster with only 1 element). Although this is the most commonly used method one could define a different one and examine if the results of the algorithm change significantly. After this first step our distance matrix would transform into the following:

	(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

TABLE 2.2: bbbb

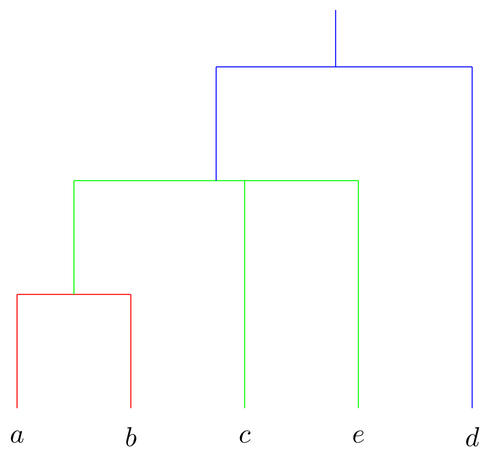
The new distances have been calculated with the method described above, for example $\bar{d}(c,(a,b)) = \min(\bar{d}_{c,a}, \bar{d}_{c,b}) = \min(21, 30) = 21$. In our next step the minimum distance we can find is 21, which is shared by the new cluster, c and e. Therefore we will merge these three into a new cluster *without forgetting the hierarchy*, meaning that we will keep in mind that a and b were combined together before any of the others. Our distance matrix will end up looking like this:

	$((a,b),\mathbf{c},\mathbf{e})$	**d**
$((\mathbf{a},\mathbf{b}),\mathbf{c},\mathbf{e})$	0	28
d	28	0

TABLE 2.3: ccc

In a last step we would combine these 2 clusters together.

We can make a dendrogram to visually represent how we have formed the clusters in every step.



A way to store the whole clustering process is with a linkage matrix. This matrix has $(N-1)\times 4$ size, where N is the number of clusters, and every row represents a step in our clustering algorithm. If row m is $(y_{m,1}, y_{m,2}, y_{m,3}, y_{m,4})$ this means that in step m we have fused the $y_{m,1}$ and $y_{m,2}$ clusters that had a distance of $y_{m,3}$.The last number $y_{m,4}$ is the number of original, individual components that the resultant cluster will have. We label our initial n elements with integers 0 to $n-1$ and then every new cluster is labeled with the next integer. For example if our clustering method ended up with the following result:

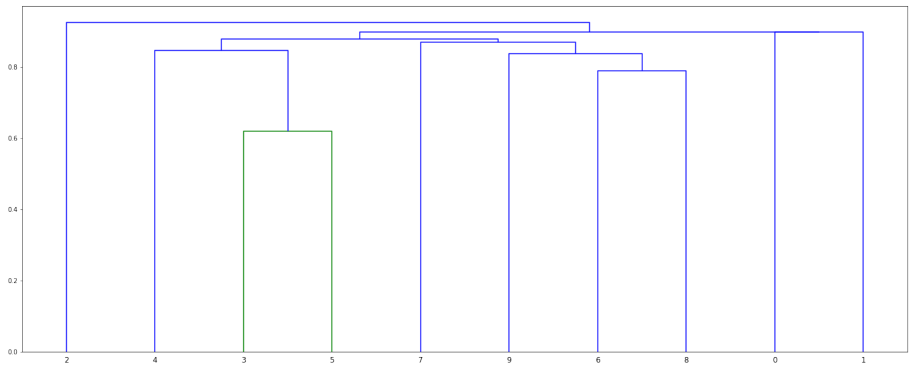


FIGURE 2.1

It would lead to a linkage matrix like this:

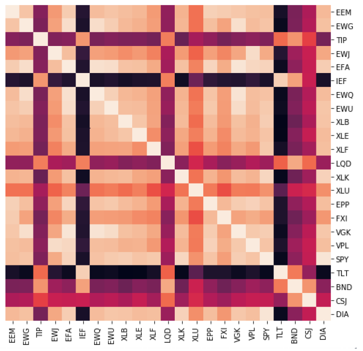
	0	**1**	**2**	**3**
0	3.0	5.0	0.618707	2.0
1	6.0	8.0	0.788512	2.0
2	9.0	11.0	0.837068	3.0
3	4.0	10.0	0.845906	3.0
4	7.0	12.0	0.869022	4.0
5	13.0	14.0	0.878178	7.0
6	0.0	1.0	0.897254	2.0
7	15.0	16.0	0.898109	9.0
8	2.0	17.0	0.924699	10.0

TABLE 2.4: ddd

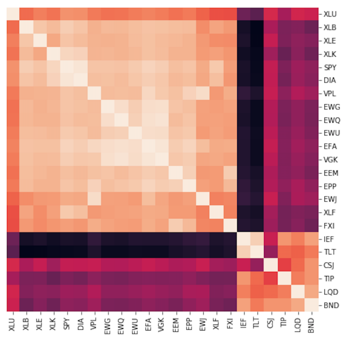
As we can observe in the dendrogram, this would mean that in the first step we combined assets 3 and 5 into a new cluster labeled 10. In the second step we would combine assets 6 and 8 into a cluster labeled 11 and in the next step we would combine this cluster 11 with asset 9 to create a cluster of 3 assets.

2.2 Quasi-diagonalization

In this step we reorganize the assets in the correlation matrix so that the largest correlations lie around the diagonal. This way, assets will end up close to those similar to them and far apart from very different ones and we will be able to visualize the clusters in a correlation matrix. To follow the order given by our clustering method we just have to look at every row of the linkage matrix, starting by the last one, and recursively replace the clusters $(y_{m-1,1}, y_{m-1,2})$ by their components.



(A) Unclustered Correlations



(B) Clustered Correlations

FIGURE 2.2: https://hudsonthames.org/an-introduction-to-the-hierarchical-risk-parity-algorithm/

In figure —- we can see how a correlation matrix could look before and after undergoing this quasi-diagonalization. The colors represent higher and lower correlations and we can observe that greater correlations lie near the diagonal forming clusters.

2.3 Recursive bisection

This is the final step where we will decide how to assign the weights in our portfolio. We will initialize all the weights with the same value for all n assets, $w = (1, 1, ...1)$. Then we will explore our clustering tree from top to bottom and every time it is divided in two clusters these will compete for the weights. That is, we will compute an individual volatility for each one of the clusters and we will update their weights in a manner inversely proportional to the volatilities. We compute these volatilities keeping in mind that for a diagonal correlation matrix the inverse-variance allocations are optimal (). This is, for each cluster the weights we will use to compute its volatility are:

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

Where V is the covariance matrix of the constituents of the cluster. This way we can compute the variance of both clusters as

$$\sigma_1 = w_1^T V_1 w_1$$
$$\sigma_2 = w_2^T V_2 w_2$$

.

Now we compute two factors we will use to re-escalate the weights of the clusters, α_1 and α_2.

$$\alpha_1 = 1 - \frac{\sigma_1}{\sigma_1 + \sigma_2}; \alpha_2 = 1 - \alpha_1$$

.

And we will have that $w_1 = w_1\alpha_1$, $w_2 = w_2\alpha_2$. Observe that the total sum of the weights of the final result is equal to 1. Suppose we have 2^n assets, then our initial vector w will be a vector of 2^n ones, with a sum of 2^n. In the first step we multiply half the vector by α_1 and the other half by $\alpha_2 = 1 - \alpha_1$. If we make pairs with an updated weight from the first half and another one from the second, each pair has a sum of 1 and there are $\frac{n}{2}$ pairs, meaning that the total sum has been halved in this step. Recursively, in every step we take a vectors of identical weights and multiply half of it by α_i and the other half by $1 - \alpha_i$, halving their sum. If we consider that we visit the entirety of our assets every step, the whole process will take $\log_2 2^n = n$ steps and the final sum will be $2^n \cdot (\frac{1}{2})^n = 1$. For example:

$$(1,1,1,1) \longrightarrow (0.7, 0.7, 0.3, 0.3) \longrightarrow (0.7 \cdot 0.6,\ 0.7 \cdot 0.4,\ 0.3 \cdot 0.8,\ 0.3 \cdot 0.2)$$

2.4 Code

We prepared a snippet of code that allows us to perform HRP on a predefined set of stocks and then assess its performance in comparison to the classic methods.

These will be the imports that we will need. Most of the work is made using the functions of `Numpy` and taking advantage of the properties that `Pandas`' structures offer. We will only need `Scipy` to allow us to make dendrograms. The visualizations will be made through the `Matplotlib` and `Seaborn` libraries.

```python
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
import seaborn as sns
```

We prepared two ways of reading the input data:

1. In our first tests, we obtained the data from `Yahoo Finances`. In this website, you can select a given stock and obtain historical data within some parameters like time period and frequency. You will be able to download a file `csv` that has 7 columns: Date, Open, High, Low, Close, Adj Close and Volume.

   In order to perform our simulations, we needed to download one file for each of the stocks we wanted to analyze. Then, we had to perform some computations to compute the returns of each stock (returns are computed as explained in the previous chapter). Finally, we needed to obtain a single data structure that encapsulated the information for all the stocks, as the HRP algorithm works in a matrix manner like. This is what the next code does: from reading the files, to computing returns, to create a single data structure for all the stocks. Note that we need to give the name of the stocks beforehand to the code, and that depends on the test that we are running.

```
#TEST 3 DIVERSIFIED DATASET
stocks = ["BNDX","EMB","IEFA","IEMG",
"ITOT","IWN", "IWS","JPST", "MUB","SCHV","TFI",
"VBR","VEA", "VOE","VTI", "VTIP","VTV", "VWO"]
stocks_compl = ["AGG"] + stocks


#Start by reading the file of the first stock
aapl = pd.read_csv(stocks_compl[0]+'.csv', index_col=0)
#Get the column with the information of the closing prices
#The function pct_change does the percentage with respect
#to the previous value in the column
#Thus with the pct_change we go from prices to returns
pct_aapl = aapl["Close"].pct_change()
#We also save the prices so they may be used later on to compute
#different information
prices_aapl = aapl["Close"]
returns = pct_aapl.to_frame()
prices = prices_aapl.to_frame()

#Rename the column from "Close" to the stock name that it came from
returns  = returns.rename(columns={"Close": stocks_compl[0]})
prices  = prices.rename(columns={"Close": stocks_compl[0]})

#Do exactly the same for all the stocks in the list, and join all
#the dataframes together
for stock in stocks:
df = pd.read_csv(stock+'.csv', index_col=0)
df_pct = df["Close"].pct_change()
df_price = df["Close"]
returns = returns.join(df_pct).rename(columns={"Close": stock})
prices = prices.join(df_price).rename(columns={"Close": stock})

#The pct_change leaves one NaN at the start of the dataframe
#we must take that out.
returns = returns.dropna()
returns.head(), prices.head()
```

2. Later, we wanted to test our code with data extracted from JPM's paper, which we will discuss later. The data was given to us in a different format: a single file contained all the stocks. Such file had a column for the date and one column with the prices for each of the stocks. This kind of format was easier to deal with, but we had to code some lines for it, as we still needed to compute returns.

   Additionally, the data extracted from the paper was not clean. Some of the stocks couldn't be found because they were either not publicly disclosed or no longer existed. From the stocks that we could obtain, many values from certain dates were missing. To this point, we had two options: either input the missing data or just skip the rows. We decided to simply skip the rows because inputing the missing data would implicate to dedicate a special analysis on how the inputation affects the result of the HRP algorithm. We wanted to focus this work on the algorithm specifically, so we did not want to introduce new variables with the inputation.

```
#----------------------------------------------------------------------
#TEST 5 JPMORGAN PAG 16 VALUES
stocks = ["JMABDJSE INDEX","JPFCVA01 INDEX",
"JPFCVA05 INDEX","JPMZVEL1 INDEX", "JPMZVP4G INDEX"]
stocks_compl = ["ERJPMFGU INDEX"] + stocks
variable_read = "vals_16.csv"
#----------------------------------------------------------------------

prices = pd.read_csv(variable_read, decimal=',', index_col=0)
prices = prices.dropna()
returns = prices.pct_change().dropna()
prices.head(), returns.head()
```

Once we reach this part, we have a single data structure using the library `Pandas`: it is indexed by the date and has a column containing the returns information for each of the stocks that we are dealing with. In the code, this structure has the variable name `returns`. We can start the HRP algorithm.

The first step is the **tree clustering**. We need to compute a distance matrix from the return prices. Note that here relies the importance of having all the information in the same data structure: we deal with it as if it were a matrix. We will make use of the advantages of computing with `Numpy` and then use `Scipy` to compute the dendrogram, as it makes it a lot easier than to just do it manually.

Notice that we use the libraries `Matplotlib` and `Seaborn` for visualization purposes.

```
#Build the correlation matrix and print it for visualization purposes.
corr = returns.corr()
#Vislualization purposes
ax = sns.heatmap(corr, cmap="coolwarm")
#Following the instructions of the HRP, build the distance matrix
#based on the correlation matrix.
d_corr = np.sqrt(0.5*(1-corr))
#The built in function linkage identifies the clusters and returns
#a matrix that describe the cluster formation
link = linkage(d_corr, 'single')
Z = pd.DataFrame(link)

#A dendrogram is built for visualization purposes.
fig = plt.figure(figsize=(25, 10))
dn = dendrogram(Z, labels = stocks_compl)
plt.show()
```

Our next goal is to work with the **quasi diagonalization**. The function `linkage` from the previous stack of code provided us with a table that described the order in which the clusters were formed. Using this table, we will rearrange the indexes of the matrix in the proper order defined by the clustering, thus providing a quasi diagonalization.

```
def get_quasi_diag(link):

    # sort clustered items by distance

    link = link.astype(int)

    # get the first and the second item of the last tuple
    sort_ix = pd.Series([link[-1,0], link[-1,1]])

    # the total num of items is the third item of the last list
    num_items = link[-1, 3]

    # if the max of sort_ix is bigger than or equal to the max_items
    while sort_ix.max() >= num_items:
        # assign sort_ix index with 24 x 24
        sort_ix.index = range(0, sort_ix.shape[0]*2, 2) #odd numers as index

        df0 = sort_ix[sort_ix >= num_items] # find clusters

        # df0 contain even index and cluster index
        i = df0.index
        j = df0.values - num_items #

        sort_ix[i] = link[j,0] # item 1

        df0  = pd.Series(link[j, 1], index=i+1)

        sort_ix = sort_ix.append(df0)
        sort_ix = sort_ix.sort_index()

        sort_ix.index = range(sort_ix.shape[0])


    return sort_ix.tolist()
```

To make that effective, we have to remember that our initial data structure is indexed by names, more specifically, the names of the stocks. It is key that we have provided a list containing all the stocks, in the code is referenced as `stocks compl`. Then, we use the indexes properties of `Numpy` to obtain our quasi diagonalized matrix. Using `Seaborn` again we produce another visualization, so we can compare the structure of the before and after.

```
sort_ix = get_quasi_diag(link)
stocks_compl = np.array(stocks_compl)
df_vis = returns[stocks_compl[sort_ix]]
corr2 = df_vis.corr()
ax = sns.heatmap(corr2, cmap="coolwarm")
```

The last step is computing the **recursive bisection**. We start with a vector of weights that initially has all weights to 1. Then, for each cluster that we bisect, we will multiply those weights by a variation of the volatility of each cluster.

Since the volatility of the cluster is used several times in this function, we built a function to compute it. This code essentially computes the formula adapting to the format of our data structure.

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

```python
def get_cluster_var(cov, c_items):
    cov_ = cov.iloc[c_items, c_items] # matrix slice
    # calculate the inversev-variance portfolio
    ivp = 1./np.diag(cov_)
    ivp/=ivp.sum()
    w_ = ivp.reshape(-1,1)
    c_var = np.dot(np.dot(w_.T, cov_), w_)[0,0]
    return c_var
```

And then we proceed with the recursive bisection.

```python
def get_rec_bipart(cov, sort_ix):
    # compute HRP allocation
    # intialize weights of 1
    w = pd.Series(1, index=sort_ix)

    # intialize all items in one cluster
    c_items = [sort_ix]
    while len(c_items) > 0:
        # bisection
        """
        [[3, 6, 0, 9, 2, 4, 13], [5, 12, 8, 10, 7, 1, 11]]
        [[3, 6, 0], [9, 2, 4, 13], [5, 12, 8], [10, 7, 1, 11]]
        [[3], [6, 0], [9, 2], [4, 13], [5], [12, 8], [10, 7], [1, 11]]
        [[6], [0], [9], [2], [4], [13], [12], [8], [10], [7], [1], [11]]
        """
        c_items = [i[int(j):int(k)] for i in c_items for j,k in
                    ((0,len(i)/2),(len(i)/2,len(i))) if len(i)>1]

        # now it has 2
        for i in range(0, len(c_items), 2):

            c_items0 = c_items[i] # cluster 1
            c_items1 = c_items[i+1] # cluter 2

            c_var0 = get_cluster_var(cov, c_items0)
            c_var1 = get_cluster_var(cov, c_items1)

            alpha = 1 - c_var0/(c_var0+c_var1)

            w[c_items0] *= alpha
            w[c_items1] *=1-alpha
    return w
```

Since our goal was to analyze the HRP algorithm, we need some parameters to compare, namely, the results of the other classic methods. We computed the minimum variance, the risk parity and the uniform weights using the formulas provided in the previous chapter.

```
def compute_MV_weights(covariances):
    inv_covar = np.linalg.inv(covariances)
    u = np.ones(len(covariances))

    x = np.dot(inv_covar, u) / np.dot(u, np.dot(inv_covar, u))
    return pd.Series(x, index = stocks_compl, name="MV")


def compute_RP_weights(covariances):
    weights = (1 / np.diag(covariances))
    x = weights / sum(weights)
    return pd.Series(x, index = stocks_compl, name="RP")


def compute_unif_weights(covariances):
    x = [1 / len(covariances) for i in range(len(covariances))]
    return pd.Series(x, index = stocks_compl, name="unif")
```

We proceed to show the final results for a given set of stocks. We compute all the weights and join them in the same data structure so we can print the result and have better readability.

```
cov = returns.cov()

weights_HRP = get_rec_bipart(cov, sort_ix)
new_index = [returns.columns[i] for i in weights_HRP.index]
weights_HRP.index = new_index
weights_HRP.name = "HRP"

weights_MV = compute_MV_weights(cov)
weights_RP = compute_RP_weights(cov)
weights_unif = compute_unif_weights(cov)

results = weights_HRP.to_frame()
results = results.join(weights_MV.to_frame())
results = results.join(weights_RP.to_frame())
results = results.join(weights_unif.to_frame())
```

Finally, we compute some parameters that will allow us to better compare and analyze the result of our algorithms.

We start by computing the expected return. The function provided in this code computed the expected return for a given set of stocks, provided with some weights. The rest of the code is just dealing the `Pandas` requirements to put everything together in a single data structure so it can be easily read.

```
def compute_ER(weights):
  mean = returns.mean(0)
  return weights.values * mean

er_hrp = compute_ER(weights_HRP)
er_hrp.name = "HRP"
er_mv = compute_ER(weights_MV)
er_mv.name = "MV"
er_rp = compute_ER(weights_RP)
er_rp.name = "RP"
er_unif = compute_ER(weights_unif)
er_unif.name = "unif"

ers = er_hrp.to_frame()
ers = ers.join(er_mv.to_frame())
ers = ers.join(er_rp.to_frame())
ers = ers.join(er_unif.to_frame())
ers = ers.sum()
ers.name = "Expected Return"
ers = ers.to_frame()
```

Next, we compute the portfolio volatility. Again, only the code encapsulated in the function is really necessary to compute the portfolio volatility, the rest depends of the structure of `Pandas`.

```
def portfolio_volatility(weights, cov):
  return np.sqrt(np.dot(np.dot(weights.values, cov.values), weights.values))

data = [portfolio_volatility(weights_HRP, cov)]
data.append(portfolio_volatility(weights_MV, cov))
data.append(portfolio_volatility(weights_RP, cov))
data.append(portfolio_volatility(weights_unif, cov))
volatility = pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns=["Volatility"])
```

Our next parameter is the sharpe ratio. In order to compute it, we need a function that gives us the risk free return. This is the rate that one can obtain with virtually no risk, for example with treasury bonds. This value would depend on the country the investor is on, for now we have left it at 0.

```
def risk_free():
  return 0

def sharpe_ratio(weights, cov):
  ret_portfolio = compute_ER(weights).sum()
  ret_free = risk_free()
  volatility = portfolio_volatility(weights, cov)

  return (ret_portfolio - ret_free)/volatility

data = [sharpe_ratio(weights_HRP, cov)]
data.append(sharpe_ratio(weights_MV, cov))
data.append(sharpe_ratio(weights_RP, cov))
data.append(sharpe_ratio(weights_unif, cov))
```

```
sharpe_R= pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns=["Sharpe Ratio"])
```

Another important parameter is the maximum drawdown. There are several ways of computing this with python, including built-in functions from some libraries. However, we decided to code this by hand as it was not very difficult to do so.

```
def compute_mdd(weights):
  df = weights * prices
  df = df.sum(1)
  roll_max = df.cummax()
  daily_drawdown = df/roll_max - 1.0

data = [compute_mdd(weights_HRP)]
data.append(compute_mdd(weights_MV))
data.append(compute_mdd(weights_RP))
data.append(compute_mdd(weights_unif))
dd = pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns = ["Max DD"])
```

The last parameter that we will compute will be the diversification ratio. Notice that, in order to compute this one, we need to use the portfolio volatility function.

```
def diversification_ratio(weights, cov):
  p_volatility = portfolio_volatility(weights, cov)
  return np.dot(np.sqrt(np.diag(cov.values)), weights) / p_volatility

data = [diversification_ratio(weights_HRP, cov)]
data.append(diversification_ratio(weights_MV, cov))
data.append(diversification_ratio(weights_RP, cov))
data.append(diversification_ratio(weights_unif, cov))
dr = pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns = ["Div Ratio"])
```

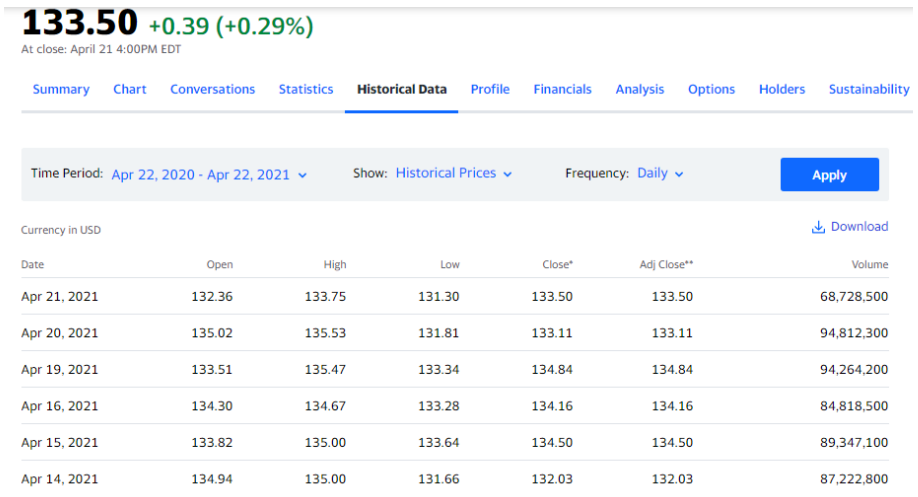
The last lines of our code just put all the parameters together so we can print a nice table containing all the results.

```
final_results = ers.join(volatility)
final_results = final_results.join(sharpe_R)
final_results = final_results.join(dd)
final_results = final_results.join(dr)
```

2.5 Tests with real world values

2.5.1 First test

In order to test how the HRP is performing, we need to find a list of stocks or financial assets with which it can create portfolios. We decided to test it with some widely known stocks with high trading volumes, namely AAPL (Apple), GOOG (Google), AMZN (Amazon), MSFT (Microsoft), GME (Gamestop), REP.MC (Repsol), ACS (Actividades de Construcción y Servicios), TRE.MC (Técnicas Reunidas), IAG.MC (International Consolidated Airlines Group), REE.MC (Red Eléctrica Corporación), OCO.V (Oroco Resources). We obtained this data from the website Yahoo Finance, that allows us to select the period of time that we want to analyze among other things. In image 2.3 we can see the options that we are given and the structure that the data has.



Time Period: Apr 22, 2020 - Apr 22, 2021 Show: Historical Prices Frequency: Daily

Currency in USD

Date	Open	High	Low	Close*	Adj Close**	Volume
Apr 21, 2021	132.36	133.75	131.30	133.50	133.50	68,728,500
Apr 20, 2021	135.02	135.53	131.81	133.11	133.11	94,812,300
Apr 19, 2021	133.51	135.47	133.34	134.84	134.84	94,264,200
Apr 16, 2021	134.30	134.67	133.28	134.16	134.16	84,818,500
Apr 15, 2021	133.82	135.00	133.64	134.50	134.50	89,347,100
Apr 14, 2021	134.94	135.00	131.66	132.03	132.03	87,222,800

FIGURE 2.3: Information shown by Yahoo Finance from the APPL stock.

To perform our tests, we looked at the closing price of every day from February 2020 to February 2021, for a total of 254 prices (since asset markets only open on working days). As commented in the previous section, the first step is to load the data for all the assets and compute all the returns. We obtain a $11x253$ table that we can use to compute the average return of every asset and the covariance and correlation matrix. Initially, with our assets in a random order, this is a representation of our correlation matrix:

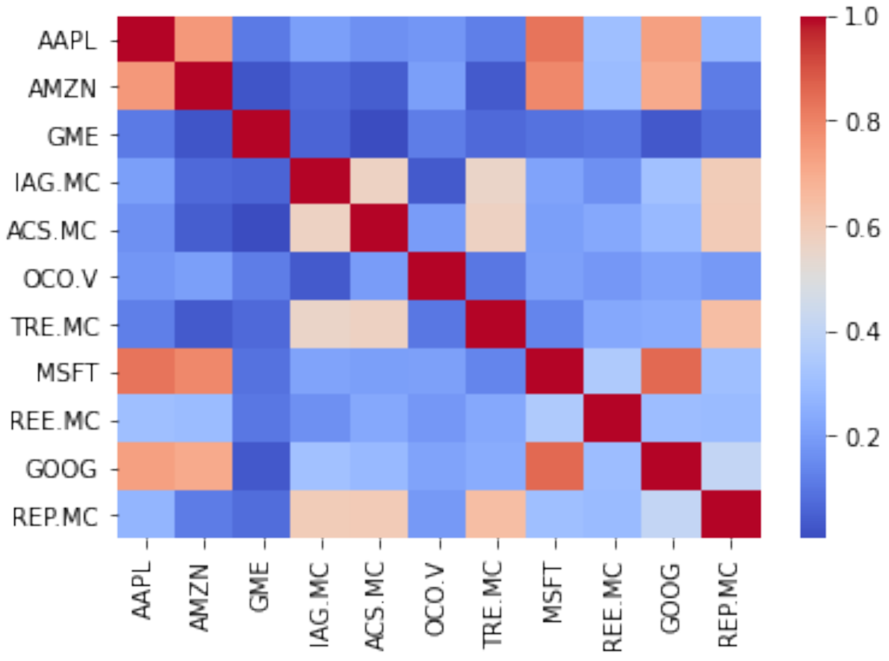


FIGURE 2.4: Correlation matrix

Now we run the first part of HRP, the clustering. Based on the correlation between the assets, the algorithm organizes them in a clustering tree.

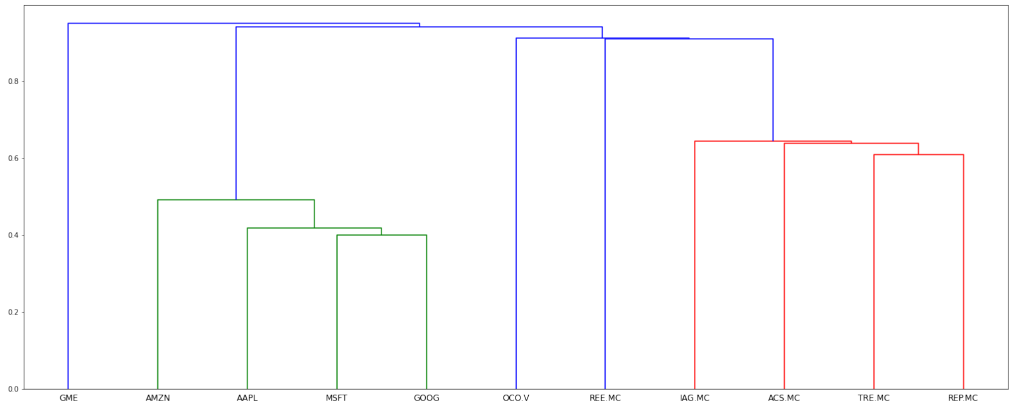


FIGURE 2.5: Hierarchical clustering tree

Unsurprisingly, the technological start-up Gamestop, that had some artificial irregularities in its behaviour lately, is isolated in its own cluster. Inside the other one, the four largest companies MSFT, GOOG, AAPL and AMZN make their own cluster, as well as the four Spanish assets IAG, ACS, TRE and REP. OCO and REE seem to be pretty uncorrelated to these clusters, even though not as much as GME. Once we use this tree to reorganize our assets and update the correlation matrix we can see that it now looks more like a diagonal matrix, and we can clearly see the clusters of correlated assets.

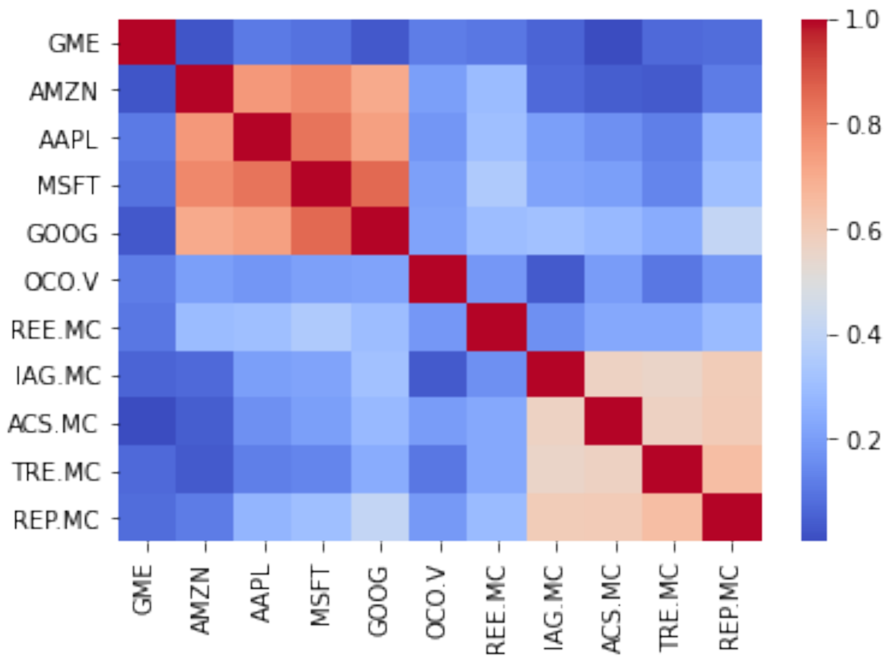


FIGURE 2.6: Updated correlation matrix

Now we are ready for our last step: weight allocation. As we explained, we initialize all our weights at a value of 1 and then start from the top of the hierarchical tree. Every bifurcation defines two clusters that will compete for weights. In the first step we will compare the "cluster" formed by GME against the one formed by the rest of the stocks. In the second step the first cluster is formed by the four global multinationals (AMZN, AAPL, MSFT and GOOG) and it will compete against the second cluster containing OCO.V and the Spanish companies. Inside every cluster, subclusters will me made as well. For example, out of the large global companies, HRP considers that AMZN can be split up first and compared with the remaining three, and unsurprisingly it also separates OCO.V, a Canadian mineral explorer working in Mexico, from the group of Spanish companies early in the process. After the allocation is complete we compare the results we obtain from HRP with the results of some of the classical methods: minimum variance (MV), risk parity (RP) and uniform weights (Unif).

	HRP	**MV**	**RP**	**Unif**
GME	0.5130	0.4375	0.3603	9.0909
AMZN	23.3498	41.7889	16.4011	9.0909
AAPL	8.2981	-1.9115	10.0986	9.0909
MSFT	5.0862	-28.8795	11.4706	9.0909
GOOG	6.2536	24.5276	14.1034	9.0909
OCO.V	4.4302	0.4018	2.8555	9.0909
REE.MC	35.0759	53.4189	24.7810	9.0909
IAG.MC	3.5279	-1.9741	2.4925	9.0909
ACS.MC	0.55035	5.0107	5.1570	9.0909
TRE.MC	3.9406	7.1005	6.0778	9.0909
REP.MC	4.0212	0.0792	6.2022	9.0909

TABLE 2.5: Weights (in %)

	Expected Return	**Volatility**	**Sharpe Ratio**	**Max DD**	**Div Ratio**
HRP	0.520703	0.413588	1.258988	−0.236543	0.108533
MV	0.108486	0.248472	0.436610	−0.240712	0.088778
RP	0.165559	0.296241	0.558865	−0.250039	0.097347
Unif	0.784052	0.416880	1.880764	−0.255251	0.112841

TABLE 2.6

The main goal of this test is not to do a quantitative analysis, since the investment universe we have chosen is arbitrary and does not resemble a real world problem. However, it is a good illustration of the clustering process of HRP works and how the clusters that are formed make sense intuitively. It is interesting, in any case, that HRP manages to offer a good expected return without too much cost in the volatility compared to other methods. The uniform weights offer a similar result in these areas, but presents the largest maximum drawdown, while HRP keeps it minimal. In our following tests we will aim to determine if this performance is also shown in more general contexts.

2.5.2 Second test: applying HRP to ETFs

In our first test, we selected the initial set of individual company stocks from our common knowledge of what we could consider to be good companies (Apple or Amazon are in general considered to be large, solid and reliable companies with good results), but without any founded economy knowledge about them. After further investigation we found out that large investment companies tend to use a type of assets known as ETFs (exchange traded funds). We will define and discuss about ETFs in chapter 3 when we talk about Scalable Capital's strategy. We thought that it would be interesting to apply HRP in an investment universe of ETFs to test its performance with this kind of asset. For this test in particular we used the investment portfolio of Betterment, a large financial advisory firm. We looked at the price history of these assets from February 2020 to February 2021.

This is the initial correlation matrix that can be calculated from the prices.

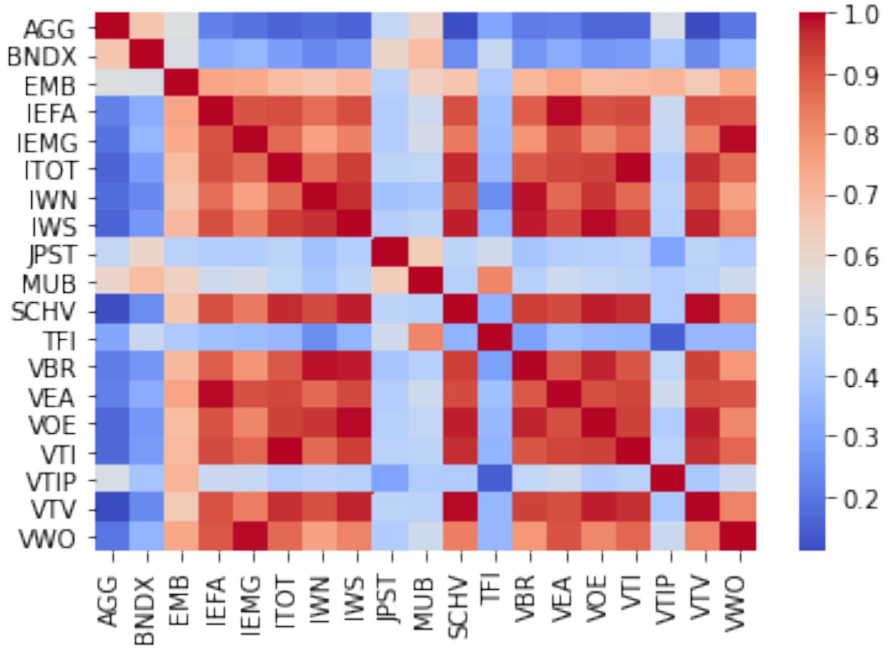


FIGURE 2.7: Correlation matrix

The clustering process that uses this matrix leads to a clustering tree much more complex than the one in the previous example:

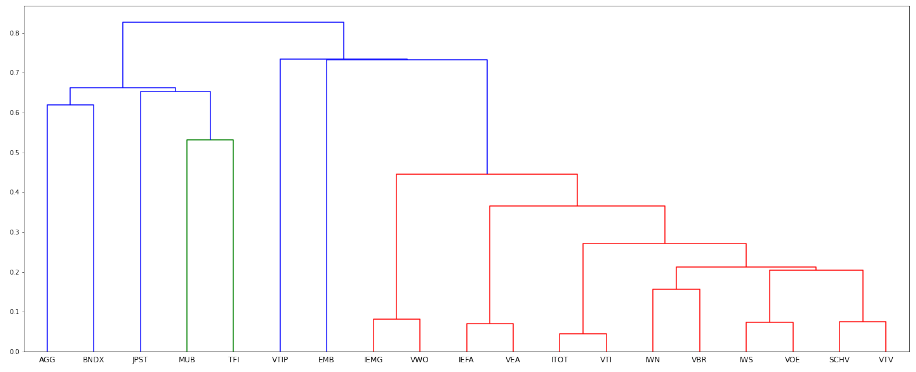


FIGURE 2.8: Cluster tree

It is also enlightening to look at the ordered correlation matrix to see the reasoning behind the clusters. In this case, we can observe clearly how there is a strongly correlated cluster formed by over half of the items in the list.

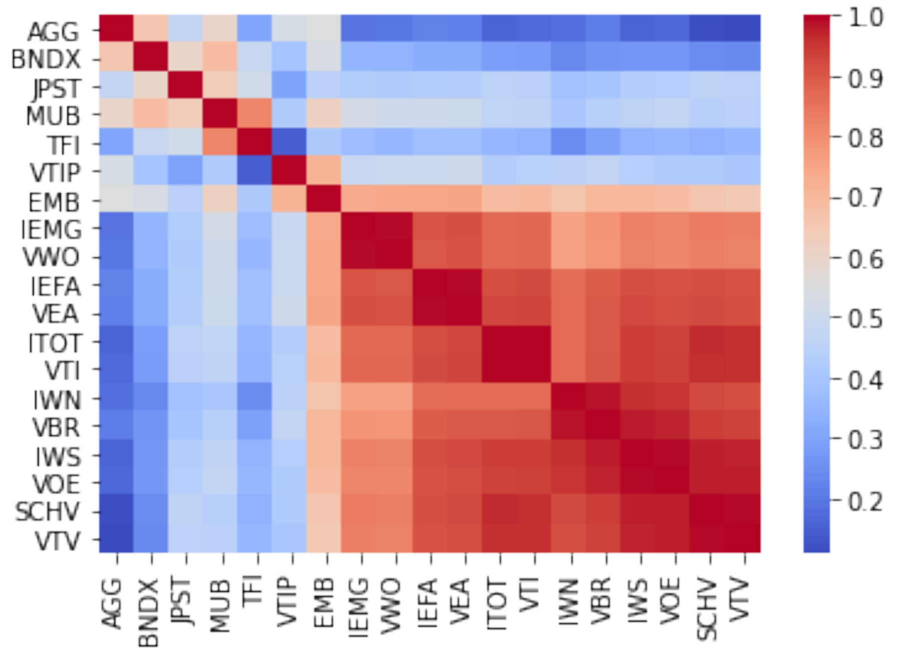


FIGURE 2.9: Updated correlation matrix

Once the weight allocations are complete we compare again the expected performance of this method and the other ones.

	Expected Return	**Volatility**	**Sharpe Ratio**	**Max DD**	**Div Ratio**
HRP	0.040447	0.180274	0.224364	−0.053301	0.069832
MV	0.019157	0.016240	1.179671	−0.088886	0.039828
RP	0.013317	0.036773	0.362126	−0.060851	0.085254
unif	0.154214	0.238123	0.647626	−0.268592	0.069871

TABLE 2.7

While it is true that HRP offers again the highest expected return, except for that of the uniform method, this time the volatility of its portfolio is too high in comparison to the others. This is understandable when compared to minimum variance, since the whole point of the method is to minimize the volatility, but it is surprising when compared to risk parity, which is a similar method. When it comes to the maximum drawdown, HRP is the best performing method again. Overall we can observe here how using the tools in Markowitz' model in the various methods allows us to greatly reduce the risk of a great loss in comparison to the behaviour of the investment universe as a whole, represented by the uniform weight allocation.

2.5.3 Third test: applying HRP to an investment bank's portfolio

JP Morgan is one of the world's largest investment banks. Since the original idea of this work came from a research from the J.P.Morgan company [3], it seemed natural to try to replicate their results or at least to compare our results and conclusions to theirs. J.P.Morgan performed three different experiments, increasing the volume of data in each one. Gerard Alba, managing director of business and investments in Banc Sabadell d'Andorra, looked for and provided us with the data that was used in such studies. We found out that many of their indices were not of public domain, so the simulations will definitely have a different result from the one presented in their paper.

The first test begins at page 12, and there is a list of 16 stocks involved. However, from this list, 6 indices were restricted, meaning that we could find them but could not have access to their values; and 4 others did not exist, meaning that we could not find their values or their historical data. So, our test was cut short at 7 stocks. Additionally, some stocks had missing data, and since the HRP needs a matrix-like structure, we had to remove all rows were there was some missing data.

Exist	Restricted	Don't exist
• Curve Select (JMABDJSE)	• J.P. Morgan Volecule UST (JPVLUTYO)	• CarryMax Future (JCMXF6US)
• ERP Global Momentum (ERJPMOGU)	• Seasonal Spreads (JMABSSPE)	• Mozaic Fixed Income (JMOZFI2U)
• ERP Global Value (ERJPVLGU)	• Compendium (JCOPC)	• CurveTrader M+ (JPCVTUS)
• FX Currency Carry Pairs (JPFCCP02)	• Commodity volatility (JMAB184E)	• Equity Volatility Basket with Mean Reversion (JPOSGLMR)
• FX Trends (JTRDX2BU)	• Divimont Short dated Index (JPEDIUFX)	
• FX PPP DM (JPFCVA01)		
• FX Volecule Basket (JPVOFXB2)		

TABLE 2.8: Caption

This is the correlation matrix we obtain from the data.

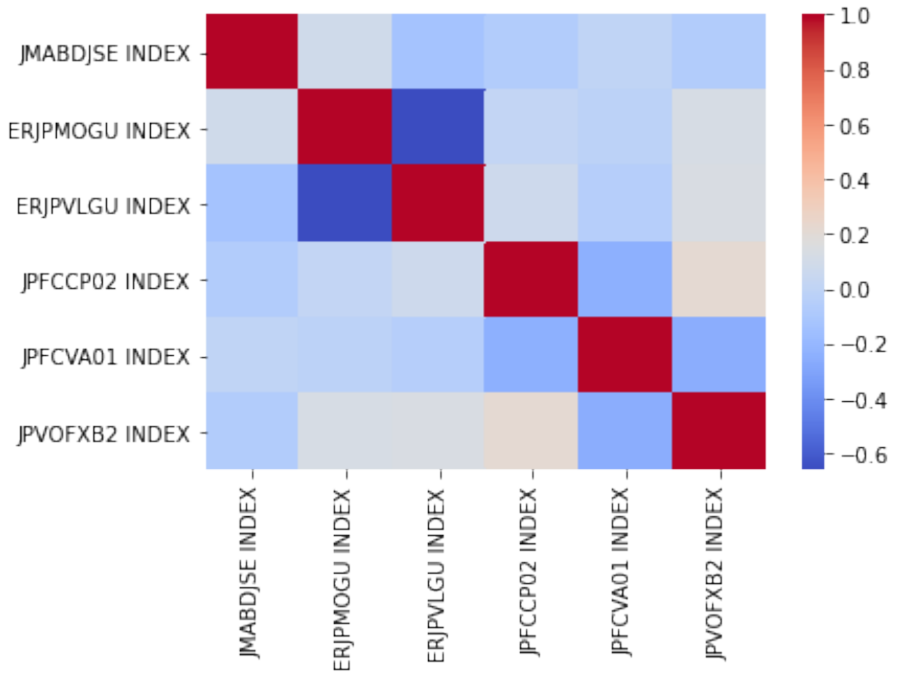


Figure 2.10: Correlation matrix

We run HRP and this is the cluster structure it creates:

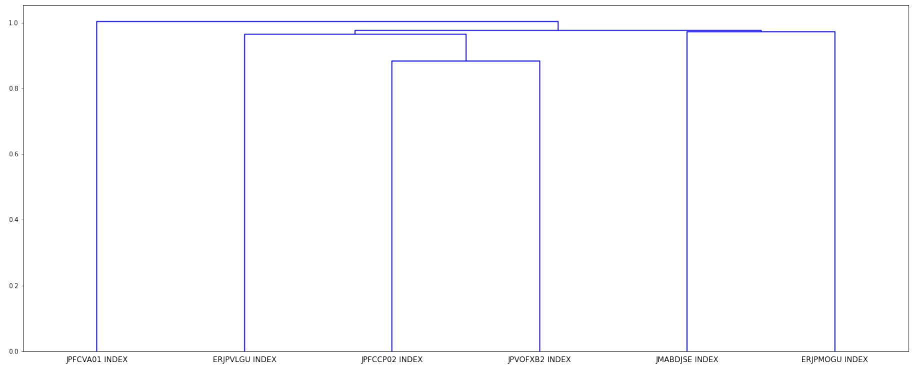


Figure 2.11: Cluster tree

Leading to the following re-ordered correlation matrix.

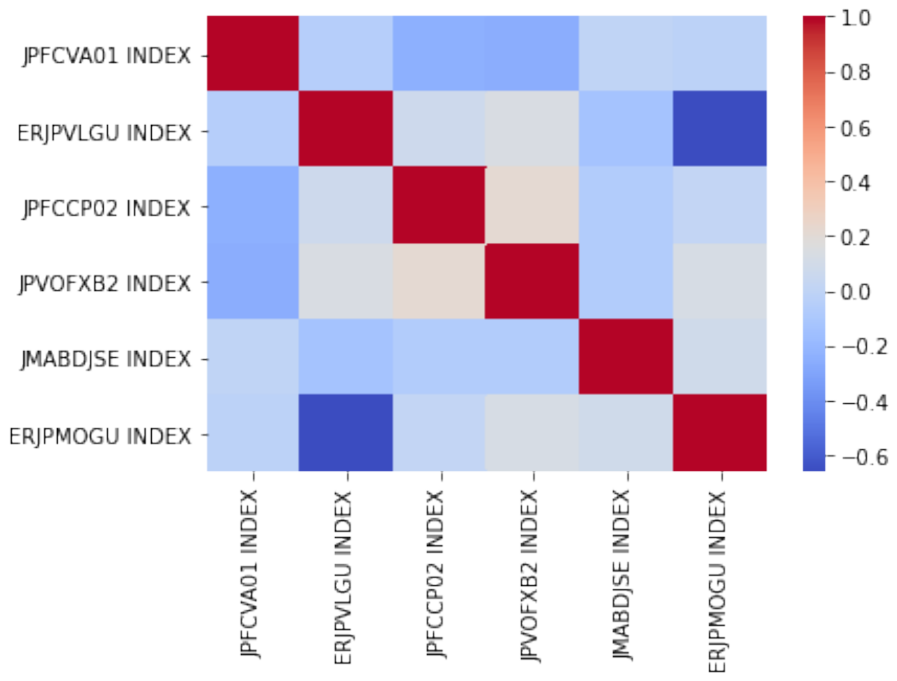


FIGURE 2.12: Updated correlation matrix

We can observe that in this case most of the assets are weakly correlated, meaning that we do not have clearly well defined clusters like in the previous examples.

	Expected Return	**Volatility**	**Sharpe Ratio**	**Max DD**	**Div Ratio**
HRP	-0.017768	0.021420	-0.829503	-0.128596	0.112680
MV	-0.016790	0.012561	-1.336687	-0.122035	0.160715
RP	-0.019005	0.012753	-1.490225	-0.129609	0.145115
unif	0.003873	0.023277	0.166368	-0.063035	0.173010

TABLE 2.9

When we analyze the performance of our methods in this investment universe there are not significant differences between the three algorithms, except for HRP having a larger volatility once again.

The second test begins at page 16 and there is a list of 31 stocks involved. From such list, 16 stocks were restricted and 9 did not exist. We could only have access to 6 indices, meaning that this was the test that we performed with the least data.

Exist	Restricted	Don't exist
•ERP Global Multi-Factor (ERJPMFGU)	• Japan Mean Reversion (CIJPJMLS)	• CarryMax Future (JCMXF6US)
• Curve Select (JMABDJSE)	• QES (JPUSQEM2)	• Curve Trader M+ (JPCVTUS)
•FX PPP DM (JPFCVA01)	• Kronos US (JPMZKRN2)	• Mozaic Fixed Income Series 2 (JMOZF2CE)
• FX PPP EM (JPFCVA05)	• Compendium Fundamental (Seasonal Spreads)	• FX Trends (JTRDFXB2)
• FX Equity Momentum (JPMZVEL1)	• WTI Crude Continuum (JMABCCLE)	• EEquity Volatility Basket (JPOSGLSO)
• US Volatility Term Premia (JPMZVP4G)	• Curve WTI (JMC13CLA)	Volemont Commodity Basket (JPVOBA1E)
•Ranked Alpha (JMABRALO)	• FX Volatility Basket (JPVOFXB1)	
	• Commodity Curve Value (JMABCCVP)	• Equity Volatility Basket with MR (JPOSGLMR)
	• Govt. Bond Carry (JGCTRCCU)	• Macro Hedge Enhanced VT 2 Short-Only (JPMZVMS2)
	• FX Markowitz (JPFCDYB1)	
	• FX Carry (JPFCARRU)	
	• FX CCP (JPFCCP01)	
	• J.P. Morgan Volecule UST (JPVLUTYO)	
	• Commodity Volatility Basket with BE (JMAB184E)	
	• Macro Hedge US Short Only VT 2 (JPMZVUS2)	

TABLE 2.10: Caption

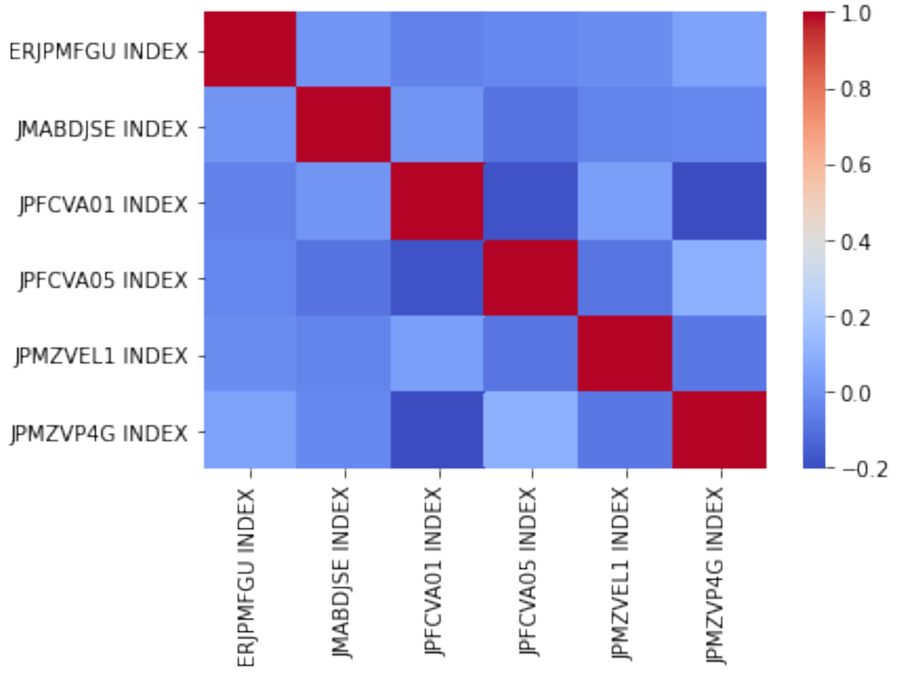


FIGURE 2.13: Correlation matrix

Similarly to the previous test, the assets are weakly correlated and the correlation matrix is already quasi-diagonal to begin with, without many changes or significant clusters to be made.

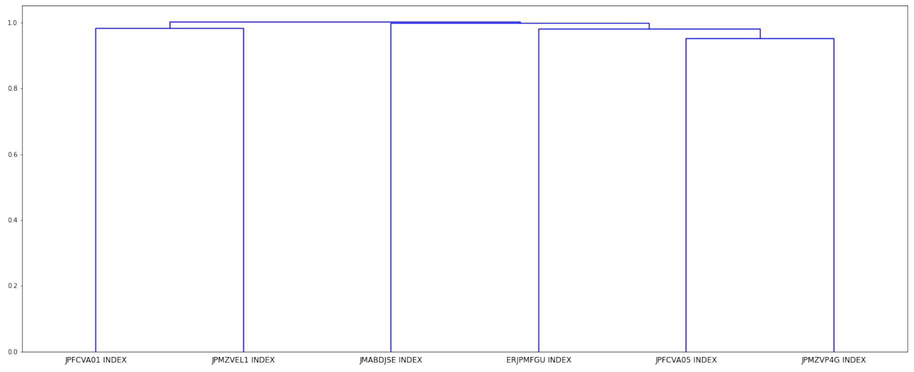


FIGURE 2.14: Cluster tree

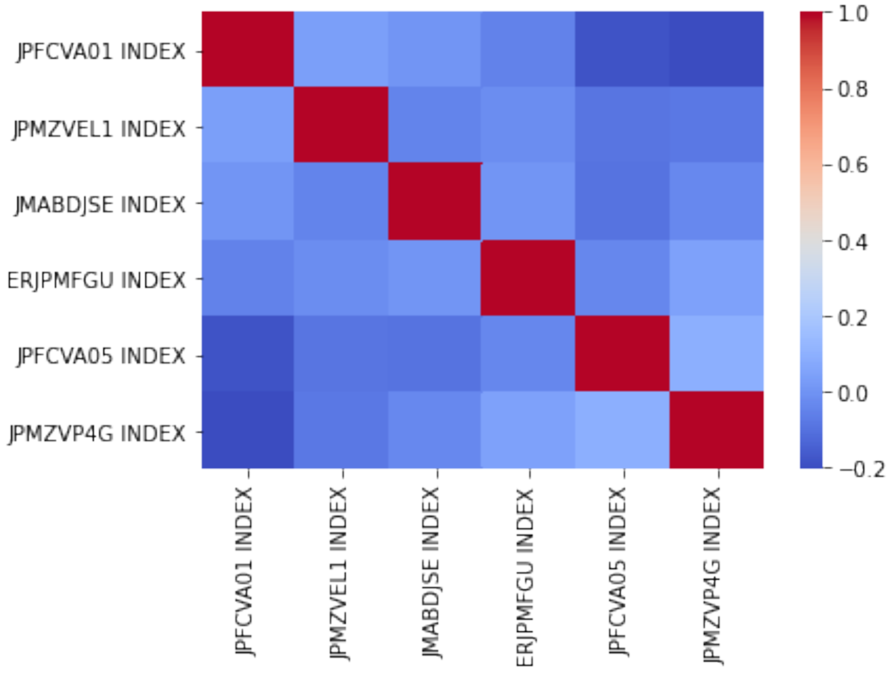


FIGURE 2.15: Updated correlation matrix

	Expected Return	Volatility	Sharpe Ratio	Max DD	Div Ratio
HRP	0.010418	0.043983	0.236855	-0.138541	0.082576
MV	-0.020605	0.013465	-1.530278	-0.140370	0.153471
RP	-0.020507	0.013876	-1.477863	-0.138335	0.134062
unif	-0.014046	0.028184	-0.498374	-0.154905	0.146640

TABLE 2.11

In this case, HRP is the only method that offers us a reasonable expected return, however it is again at the expense of a higher volatility. It also does a good job reducing the maximum drawdown, as well as RP.

Finally, the last experiment begins at page 19 with a list of 16 stocks. This time, we could find all the data: MSCI ACWI Gross Total Return USD Index (M2WD), MSCI World Gross Total Return USD Index (M2WO), MSCI Emerging Markets Gross Total Return USD Index (M2EF), MSCI USA Gross Total Return USD Index (M2US), MSCI Europe Gross Return EUR Index (M8EU), MSCI Japan Gross Return JPY Index (M8JP), MSCI AC Asia Pacific Net Total Return USD Index (M1AP), Bloomberg Barclays Global Aggregate Government Total Return Index Hedged USD (LGAGTRUH), Bloomberg Barclays US Govt Total Return Value Unhedged USD (LUAGTRUU), Bloomberg Barclays EuroAgg Treasury Total Return Index Value Unhedged EUR (LEATTREU), Bloomberg Barclays Global Aggregate Corporate Total Return Index Hedged USD (LGCPTRUH), Bloomberg Barclays US Corporate Total Return Value Unhedged USD (LUACTRUU), Bloomberg Barclays Pan European Aggregate Corporate TR Index Hedged EUR (LP05TREH), Bloomberg JPMorgan Asia Dollar Index (ADXY), Dollar Index (DXY), Bloomberg Commodity Index Total Return (BCOMTR).

In this case we have a completely different data set, with a correlation matrix that presents many large values allowing HRP to make well defined clusters.

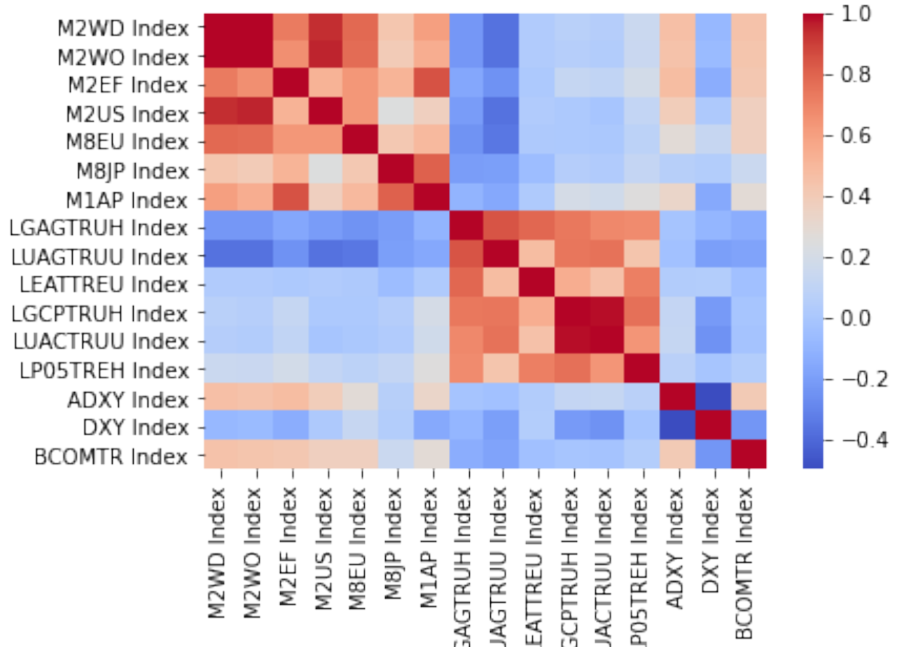


FIGURE 2.16: Correlation matrix

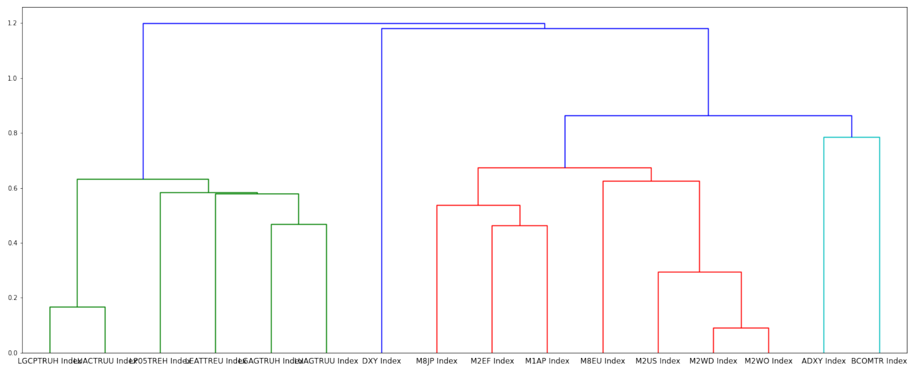


FIGURE 2.17: Cluster tree

The clustering process obtains two large clusters and an individual asset. The differences between them become clear when we look at the quasi-diagonalized correlation matrix.

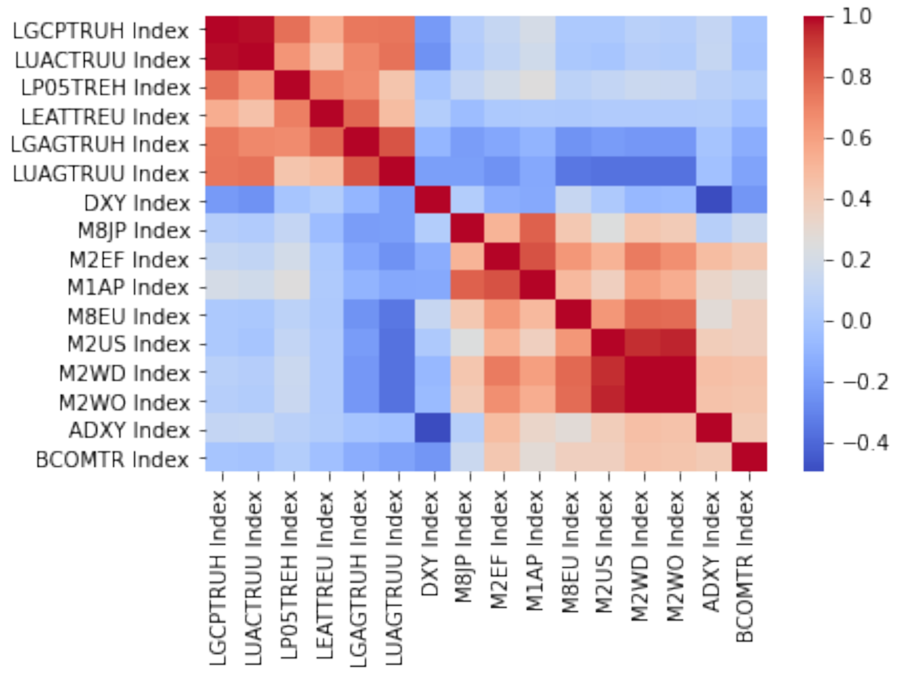


FIGURE 2.18: Updated correlation matrix

	Expected Return	Volatility	Sharpe Ratio	Max DD	Div Ratio
HRP	-0.056334	0.116700	-0.482723	-0.263790	0.083572
MV	-0.017786	0.012269	-1.449624	-0.293380	0.010857
RP	-0.027585	0.025798	-1.069265	-0.278101	0.107360
unif	-0.045692	0.068510	-0.666940	-0.497368	0.098176

TABLE 2.12: Caption

Despite again having good results when it comes to the maximum drawdown, HRP presents again a higher volatility than its alternatives.

It is important to keep in mind, however, that these magnitudes are calculated as a prediction of the performance that portfolios will have in the future. To know the real results each portfolio would have offered we would need data of future years to simulate their evolution, which is the base for our next case study.

2.6 Tests with time evolution

We designed a test that simulates how a portfolio guided by HRP or other method would perform in an extended period of time, a simulation that resembles much more how an investor can use these methods in the real world. The initial step will be identical to our previous tests: from a data sample of one year we will let our method decide what weights it considers optimal for our portfolio. Afterwards, we will observe how this portfolio evolves within the next month. Since we have the individual asset prices, we can calculate how much the price of the total portfolio will increase or decrease. After this month has passed, we will run the optimization method again using the updated last year of data. For example, in January 2018 we will use data from January 2017 all the way to January 2018 to calculate the portfolio's weights, then observe how this portfolio performs during February. After that, we can use the data from February 2017 to February 2018 to calculate new optimal weights, recalibrate the portfolio to these, and evaluate its performance in March. This way we will obtain a particular variation in price every month, which will be the real return of the portfolio, not just the expected one. With this return we can track the evolution in the total portfolio's value throughout time.

It is also important to take into account the transaction cost there is when moving capital from an asset to another. The presence of this cost favours methods that are stable and do not need to make large changes on their weight allocations when the data from the assets is updated. This is a new concept that did not arise when making static simulations and will now play an important role when evaluating a method's performance.

For these simulations we used JP Morgan's ETFs, of which we dispose of data from January 2015 to March 2021.

2.6.1 First data set

Our first data set corresponds to page 12 of JPM's article. If we recall section 2.5.3, this data set is formed by 6 fairly uncorrelated assets. In first place, we will run a simulation with this data set assuming that there is no transaction cost, to have a base line of how the different methods perform. This is the evolution of the price of our portfolio with the different four methods, starting with a relative value of 1:

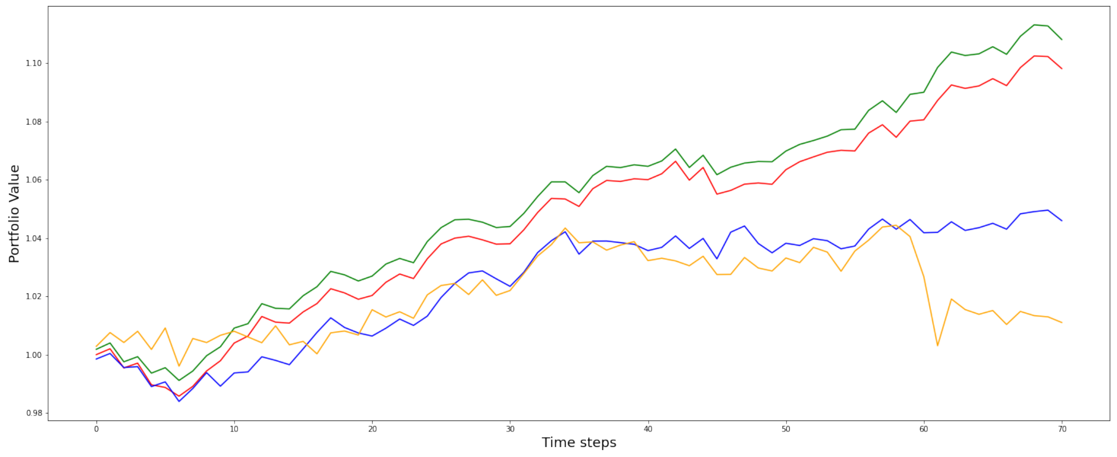


FIGURE 2.19: Value evolution of portfolios created with HRP, RP, MV and unif.

We can observe that HRP and RP give the best profit, having both a very similar evolution. MV does not perform as well as the risk parity methods, but it is clear that it outperforms the naive uniform weights method. In particular, we see once again how all the methods manage to palliate sudden downfalls that the market suffers as a whole, which happens in this example during the 2018 year.

In our previous tests, HRP had usually given us higher expected returns and volatility than other methods. We can compute the expected return that every method provides monthly to then compare it to the real returns they achieve. The following graph illustrates the expected returns of all methods step by step.

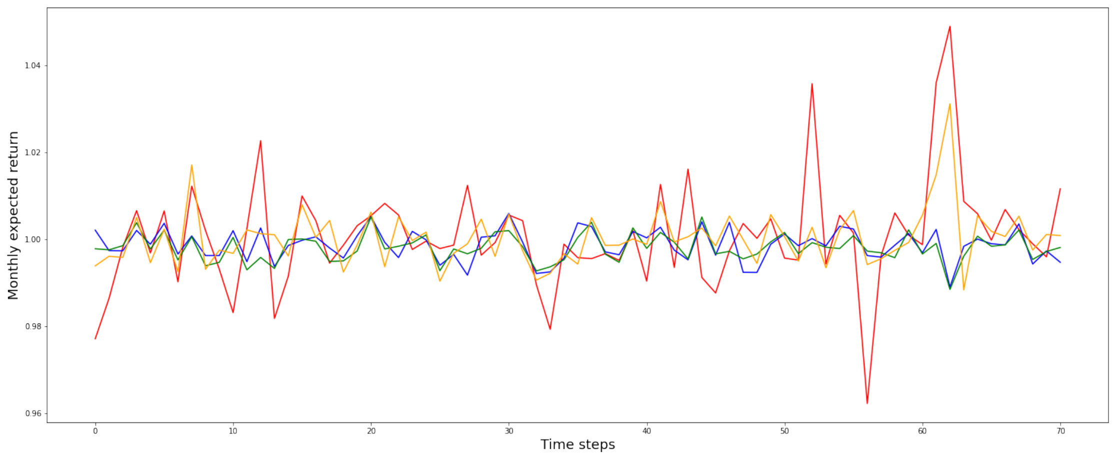


FIGURE 2.20: Expected returns of HRP, RP, MV and unif.

It can be observed in the image that HRP and RP present very similar, correlated expected returns. However, HRP's present a higher variance, which is coherent with the higher volatility it came up with in most cases. However, it is natural to ask how this expected return compares to the real return that HRP's portfolio has every month. If we compare both we obtain the following.

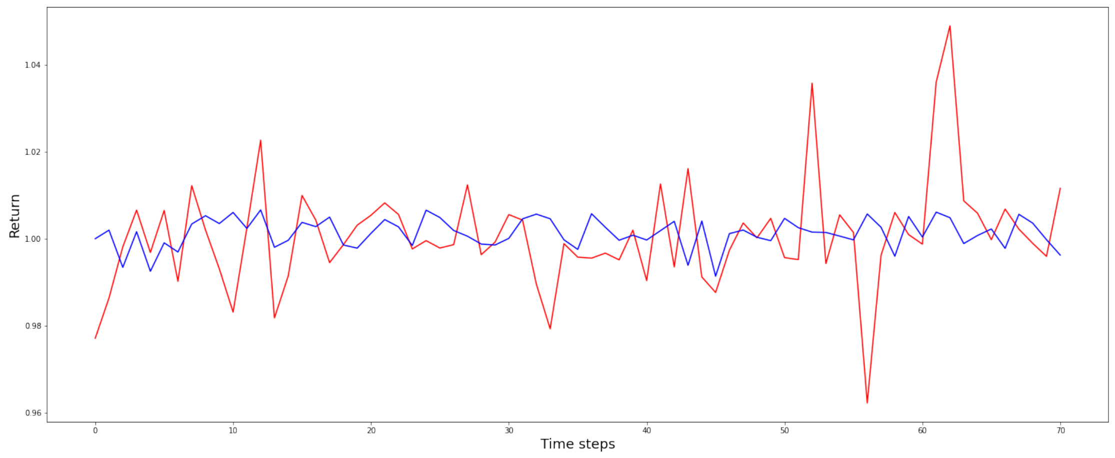


FIGURE 2.21: HRP's expected return vs real return.

It turns out that a real portfolio that follows the HRP method would have presented a much smaller variance than expected theorically. We found that this feature is particular of HRP, since other methods present more similarity in their real vs expected returns. For example, for minimum variance the comparison is illustrated in figure 2.22.

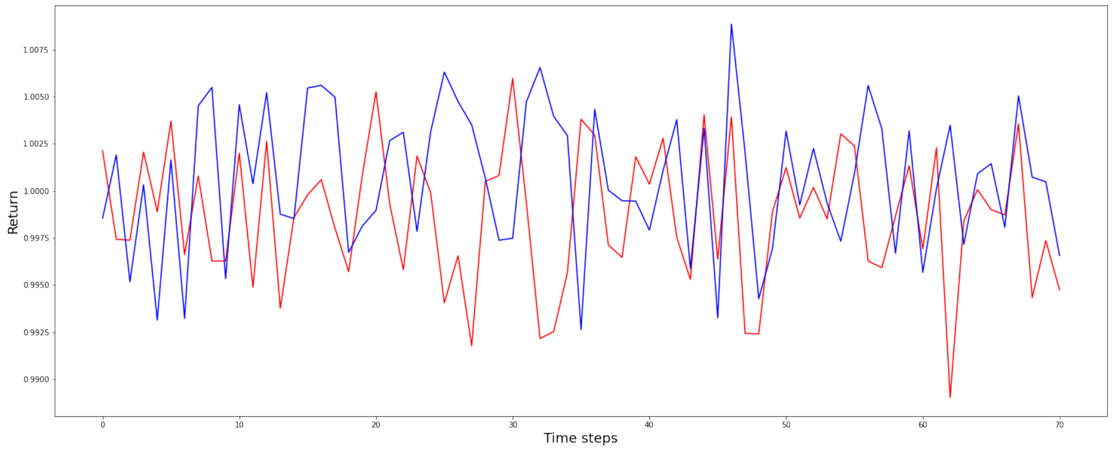


FIGURE 2.22: MV's expected return vs real return.

The reason for this difference could be explained by the fact that HRP introduces a previous step, the tree clustering, which does not stem directly from an optimization performed in the Markowitz model, therefore not giving maximum priority to the magnitudes that we are measuring, in contrast to methods like MV that minimize the volatility. However, as we can seen, that does not necessarily mean a poor performance.

There is a last factor to add to the simulation, the transaction cost that one has to assume when moving capital from an asset to another. This factor rewards methods that are not excessively sensitive to changes in data, since they could need to move large amounts of funds every time the portfolio is re-calibrated. For this test we have assumed a transaction cost of 0.2%. Transaction cost must be computed and subtracted from the return at every step. When adding this factor, the previous evolution shown in figure 2.19 becomes the following:

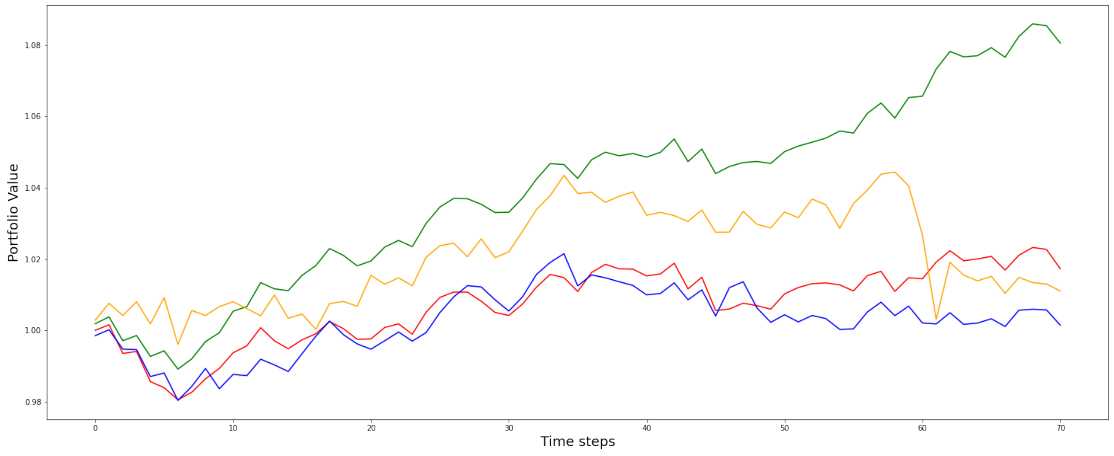


FIGURE 2.23: Evolution of portfolios with transaction cost using HRP, RP, MV and unif.

We observe that despite still being the second best performing method, HRP is the one that has been handicapped the most because of the transaction cost, which is sign that the allocations it gives vary greatly with time. It is interesting that HRP presents this issue while RP does not, even when both methods have shown a very similar performance without taking transaction cost into account. To have a better insight to this matter we can plot the weights that a particular method chooses in every point in time to see if the changes it makes are rather extreme or smooth. For example the RP method gives us the following weights.

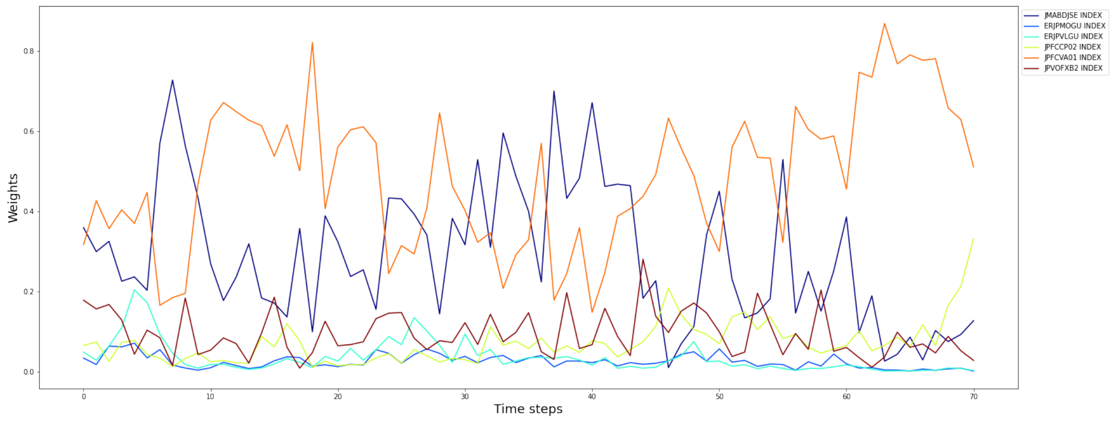


FIGURE 2.24: Weights using RP.

It seems like RP values greatly a couple of assets of which it makes most of the portfolio, sometimes choosing one over the other or balancing both with a similar weight, while keeping the other weights at low values. It looks like a reasonable evolution and does not present extreme changes, which accounts for the low costs RP suffers. However, HRP's graph looks much more chaotic.

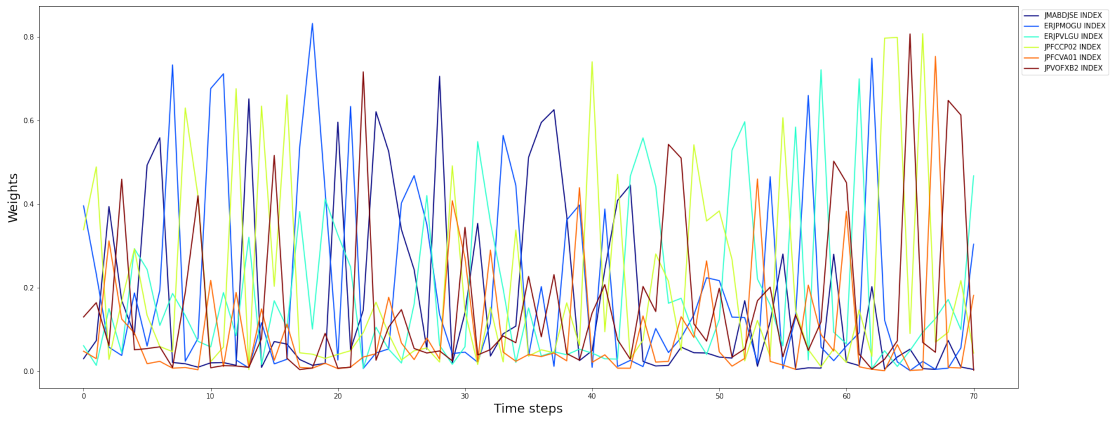


FIGURE 2.25: Weights using HRP.

While HRP also chooses one or two favourite assets of which it makes most of the portfolio, these vary greatly at every point in time. Having to move almost the entirety of the capital in every step, it is logical that this method is the most affected by the transaction costs. To solve this issue it could be possible to implement an in-between step before updating the weights: comparing the expected performance of the new ones against the previous one and deciding if the improvement is worth the transaction cost. If it is not, the updating can be postponed until the next time period.

2.6.2 Second data set

The second data set for this simulation corresponds to page 19 of JPM's article and contains many more assets, 16. This set also contains more strongly correlated assets. In this case HRP and RP do not outperform the other methods when it comes to actual returns.

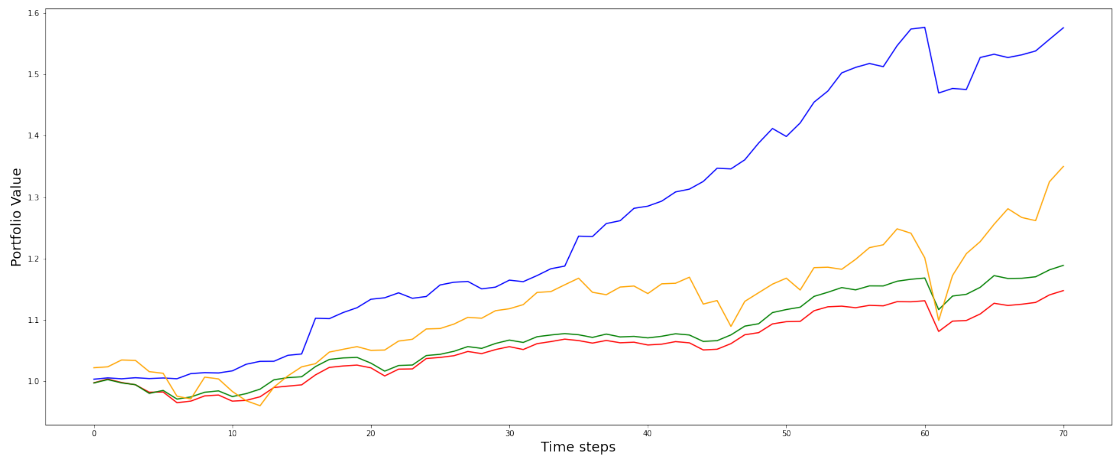


FIGURE 2.26: Value evolution of portfolios created with HRP, RP, MV and unif.

However, there are two important remarks to be made about this case. First, as in the previous case, the real returns that HRP generates are much more stable and consistent than what is predicted using the Markowitz model.

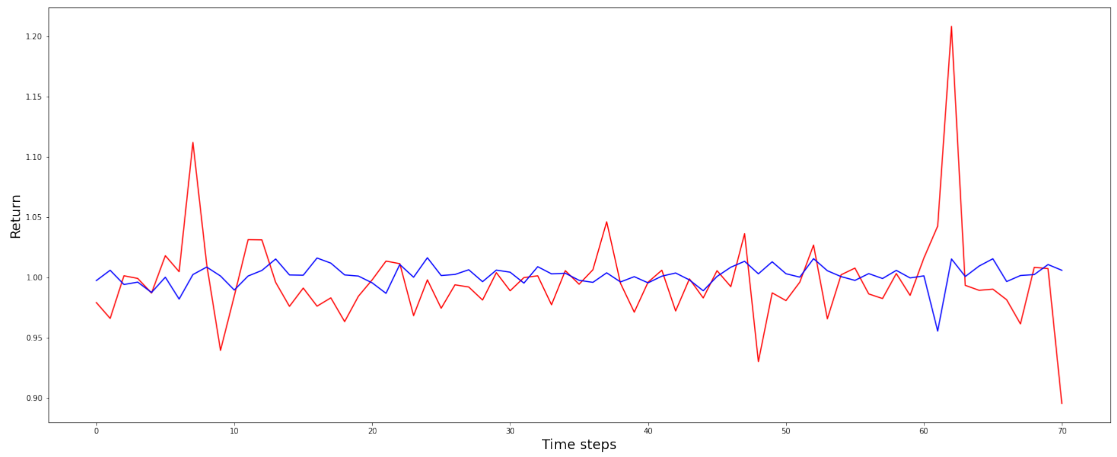


FIGURE 2.27: HRP's expected return vs real return.

Secondly, when we add transaction cost to the equation, HRP does not underperform like it did with the first data set. In fact, it keeps up with RP which again does not present heavy losses from these costs and maintains a similar evolution to the one without them.

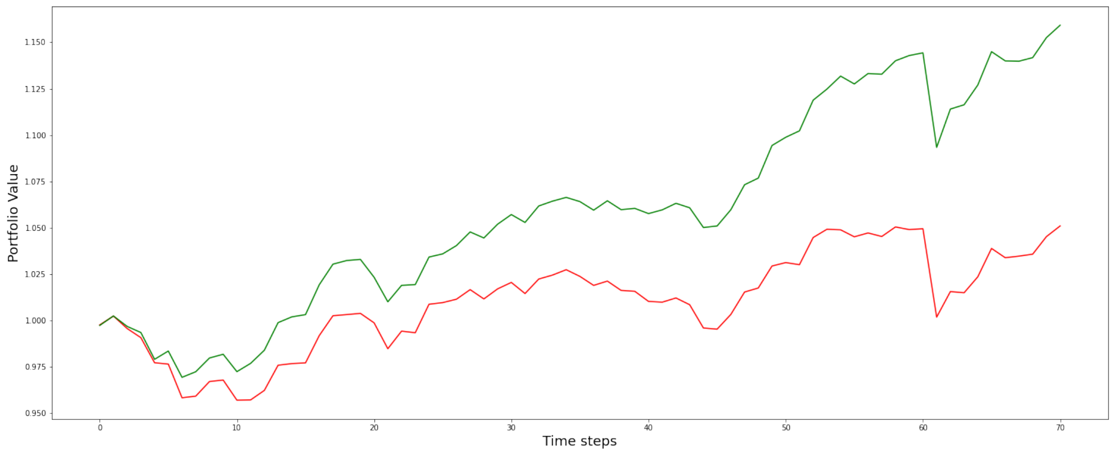


FIGURE 2.28: Comparison of the evolution of portfolio value of HRP and RP with transaction costs.

This leads us to believe that, even though in this case HRP did not outperform methods like MV, its issue with re-calibrating portfolios too extremely and causing too many costs is situational and does not appear in every scenario.

Chapter 3

Robo-advisory

Robo-advisors are platforms that give financial analysis and advice to their users in a fast and personalized way. In today's world we have to work with large quantities of information that are shared at high speed, and the amount of customers grows exponentially. Therefore it is clear that a platform that can automatically process data and give advice to thousands of customers at the same time can have a lot of advantages over traditional financial advisory. For example, it could be able to detect when certain parameters of a portfolio are outside of the expected range and it can quickly inform the user or even make automatic changes to solve the issue. In this chapter we will give examples of successful robo-advisors and explain their methods, to see the differences in their approach to the classical methods we have showed until now and to see if we can use HRP to improve on some of their results.

3.1 Scalable capital

Scalable capital is a German company founded in 2014 with the goal of using modern technology to provide investing information and advice to wealthy investors. Scalable capital's platform consists of five steps.

- Selection and updating of the investing universe to provide a good repertoire of diversified investment opportunities.
- Profiling of the investor, classifying their risk tolerance, financial status and investment goals.
- Design of the optimal asset allocation according to the investor's profile.
- Monitoring and continuous adjusting of the portfolio to maintain its target features.
- Regular update of the investor's profile.

3.1.1 Investment Universe

Scalable capital invests in funds rather than in individual assets. These are baskets of investment instruments that represent an specific asset class. In particular they choose Exchange Traded Funds (ETF's). ETF's are indexes of assets, meaning that they are composed of many assets (stocks, commodities, bonds...), but are traded just like regular stocks. This means that their price fluctuates during the day while they are being traded, unlike mutual funds which price gets updated only once daily. They have diverse advantages over conventional mutual funds: lower management cost, higher price flexibility (since it changes during the trading hours), and higher transparency when it comes to revealing what exact index and components they are based on. To make a selection of ETF's Scalable capital looks at multiple qualities that they may or may not present.

First, how expensive they are in terms of commissions, fees, management costs, etc. This is defined in terms of the Total Expense Ratio (TER) calculated as the total cost of the fund compared to the cost of the assets that it contains. This measures how many expenses a fund has and the lower it is, the better for the investor. We should also take into account the liquidity of the ETF, it is preferable to invest in an ETF with large trading volumes which have lower transaction costs. There is also the accuracy of the ETF when it comes to tracking the underlying index they are based on. There are strategies that invest only in a subset of the assets of the index to reduce the cost, but Scalable Capital prefers ETFs with high accuracy in the tracking of the index. When it comes to diversification it is clearly ideal to diversify as much as possible, however investing in very broad indexes can increase the TER since these often include elements with low liquidity. Therefore it is best to find a balance between diversifying as much as possible and keeping the expenses low. Also, in every country tax regulations may vary as well as the currency. Both have to be taken into account when selecting its ETF universe. It is also preferable to invest in ETFs as fractional as possible, with cheaper units so that it is possible to choose more accurately the amount we wish to invest or make more precise adjustments. Lastly, when it comes to evaluating the risk of an ETF it might be necessary to not only look to its market risk but also at other types. For example, some ETFs do not buy the underlying asset but replicate its price by buying derivative agreements with a third party. In this case, the possibility that this third party is not able to fulfill its part adds risk to the equation.

3.1.2 Risk Assessment

Each client of Scalable Capital has a different risk level they are willing to assume. Formally risk is associated with the fluctuation in prices, both positive and negative, however when investors think about risk their main goal is to avoid losses. That is why Scalable Capital focuses on downside risk measures when choosing the optimal porfolio for an investor's risk level.

Alternative risk measures to classic volatility

Assessing risk solely based on volatility or standard deviation can sometimes be an oversimplification and present a series of disadvantages. To start with it can be difficult for investors to fully understand its meaning and infer the potential loss from its value. Also, both unexpected gains and loses contribute to the volatility, with the assumption that these are symmetric. However we are only really concerned about the unexpected losses, which, in reality, tend to have more extreme behaviours than gains. In mathematical terms, the gaussian model for the returns is not completely accurate since they present certain skewness or asymmetry. Therefore it is useful to define other terms to describe the risk an investment carries.

- *Value at risk (VaR)*: this is the loss that would only appear with less probability than a given one. For example, if the VaR at 5% has a value of 10% this means that only in 5% of cases we could observe a loss of 10%. In a gaussian distribution this is equivalent to finding the point that leaves a tail of 5% probability to its left.

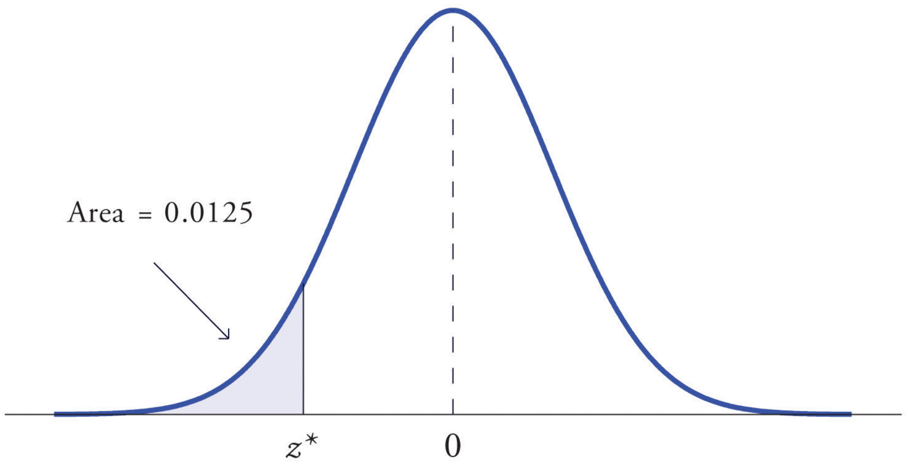


FIGURE 3.1: Gaussian tail with a 1.25% probability.

- *Expected shortfal (ES)*: we define the ES as the expected loss that we would have if we are outside the confidence interval used for the VaR. In the previous example, this would be the expected loss with the conditioned probability that the loss exceeds 10%, or the expected loss within the realm of the 5% worst possibilities.
- *Maximum drawdown (MDD)*: the maximum drawdown gives us a good idea of the greatest loss we could have and is therefore a useful measure that can be taken into account when trying to create an optimal portfolio.

Comparison of VaR, ES and MDD

It is necessary to choose what importance to give each one of these concepts when constructing a portfolio, taking into account diverse qualities in which they might perform in different ways. When it comes to comprehension, for example, VaR is the value that offers investors a most useful first idea of the risk. The statement that only one year of every 20 or 50 is going to have a loss greater than a given value is intuitive and simple to understand. Maximum Drawdown is also a very useful and comprehensible way of illustrating the maximum possible loss, but in this sense one could say that it is embedded in the VaR since it can be seen as the value at risk when we approach the 100% probability. That is why it might be more useful to look at the 95% VaR which is based on a less extreme case. The ES comes from a conditional probability, which can be not very intuitive, especially in cases like this where the condition is strict and we are making calculations under a very unlikely hypothesis. That is why when requesting information from customers or presenting them with choices Scalable Capital uses mostly the VaR to represent risk. It is also important to evaluate how well each one of the three risk indicators suits the investment universe, since different asset types can lead to different performances. For example, ES does not perform well in high risk universes where extreme events are likely. In a fat-tailed distribution the ES can not be calculated, and even though there are models to try to replicate it they make assumptions on the nature of the price distributions, which is not ideal.

The suitability of a risk measure also depends on the shape of the price distributions for the assets. A linear derivative is a financial instrument whose price depends linearly on the price of the underlying asset, but the price of a non-linear derivative, like an option, can depend on many other factors. This is important to determine if a risk measure is subadditive. Subadditivity is a property in probability that means that the value of a function evaluated in the sum of two distributions will be lower than the sum of the function evaluated in each individual distribution. For example, VaR is subadditive for linear derivatives and it stems from the subadditivity of the standard deviation of normal distributions. If we have two normal distributions X and Y with standard deviations σ_X and σ_Y and correlation ρ the standard deviation of the sum is

$$\sigma_{X+Y} = \sqrt{\sigma_X^2 + \sigma_Y^2 + 2\rho\sigma_X\sigma_Y} \leq \sigma_X + \sigma_Y.$$

Since VaR can be calculated as a multiple of the volatility it conserves this property, which is very useful since the opposite, superadditivity, leads to an overestimation of the risk of the portfolio. The investment universe of Scalable Capital does not contain non-linear ETFs and therefore it is safe to use VaR as a risk measure. In this case of linearity it is also true that the ES is proportional to the VaR, meaning that there is no extra information to be obtained from it.

Another important aspect is the ability of a risk measure is to allow backtesting and empirical model validation. Backtesting means evaluating the performance of a given strategy on historical data different to the one used previously to construct the strategy. For a risk measure to be used properly in backtesting it needs to have a property named elicitability. For a variable to be called elicitable there must exist a scoring function whose expected value under a distribution takes its minimum in the variable's value. For example, we know that the mean of a distribution Y minimizes the function $f(x) = \mathbb{E}(Y-x)^2$. In this regard, VaR is elicitable while ES lacks this property, making it practically impossible to validate its effectiveness by backtesting. Next is the computational robustness. Since MDD and ES have to do with the lower extreme of the distribution it is natural that they are more susceptible to alterations. To tackle this problem, one can try to look at a higher number of observations, not just a few on the end of the tail, but then the actual shape of the distribution of the tail is less defined and becomes similar to the original distribution. Another strategy is to look only at the most extreme observations and then use a model to theorize the rest of the distribution. The drawback is, of course, that we are making an arbitrary assumption on the distribution of the tail. These issues when it comes to robustness speak in favour of the use of VaR. When it comes to computational complexity, however, VaR is not necessarily a convex function of the weights and therefore it can present certain difficulty to compute. In this regard ES is preferable, so it is important to consider the question of when it worth it to go over the extra computational complexity in order to work with VaR.

3.1.3 Optimisation

After carefully assessing an investor's maximum risk level, Scalable Capital tries to find the portfolio that offers the maximum benefit without surpassing this risk level. As mentioned before, they focus on downside risk measures instead of looking at volatility like most methods based on the Markowitz model do. Therefore one of the constraints of the problem is not surpassing the VaR an investor can accept. There are also individual constraints on individual assets and classes that limit the maximum amount of capital that can be invested in them, to ensure diversification. Scalable capital also introduces a penalization for excessive trading to their optimization problem, to avoid excessive transaction costs like we have discussed in chapter 2. Mathematically, this optimization problem can be presented as:

Maximize: $\mu(w) - p(\Delta w)$

Subject to: $\text{VaR(w)} \leq VaR^*$

$lb_n \leq w_n \leq ub_n$, for all assets $n = 1, 2, \dots, N$

$lb_g \leq a'_g w \leq ub_g$, for all asset groups $g = 1, 2, \dots, G$

$$\sum_{i=0}^{N} w_i = 1$$

Where:

- w is the vector of weights.
- Δw the changes in the vector of weights.
- $\mu(w)$ is the expected return.
- $p(\Delta w)$ is the cost associated with the changes in weights.
- lb_n and ub_n are the lower and upper bounds for individual assets.
- lb_g and ub_g are the lower and upper bounds for asset classes or groups.
- a_g is the vector that selects the assets that belong to group g.

Chapter 4

Conclusions

In conclusion, we have found that HRP presents certain characteristics that classical methods based on Markowitz' model do not have and can lead to advantages in several situations. A particular advantage is its human-like behaviour of comparing elements to group them in clusters, which can make its results more easily understandable for investors. It is also a method that still has room for improvement. Looking into its issue with transaction costs and searching for a solution can improve its performance. The tree clustering step that HRP uses can also be used in other methods: for example we have seen that Scalable capital improves on the classical concept of variance and correlation as a risk measure, but their optimization problem still involves the concept of separating assets into different groups or clusters. That is why we believe that looking for a way to implement HRP's concept into problems like this as well evaluating its performance more diverse situations can definitely be profitable.

Appendix A

Code

```
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
from matplotlib import cm
import matplotlib.animation as animation
from matplotlib import rc
rc('animation', html='html5')
warnings.filterwarnings("ignore")


'''
#-------------------------------------------------------------------------
#TEST 1 REDUCED DATASET
stocks = ["AMZN_2018", "MSFT_2018", "GOOG_2018"]
stocks_compl = ["AAPL_2018"] + stocks
#-------------------------------------------------------------------------
'''


#-------------------------------------------------------------------------
#TEST 2 AMPIFIED DATASET
stocks = ["AMZN_2018","GME_2018","IAG.MC_2018","ACS.MC_2018", "OCO.V_2018","TRE.MC_2018",
"MSFT_2018","REE.MC_2018", "GOOG_2018","REP.MC_2018"]
stocks_compl = ["AAPL_2018"] + stocks
#-------------------------------------------------------------------------


'''
#-------------------------------------------------------------------------
#TEST 3 DIVERSIFIED DATASET
stocks = ["BNDX_2018","EMB_2018","IEFA_2018","IEMG_2018", "ITOT_2018",
"IWN_2018", "IWS_2018","JPST_2018",
"MUB_2018","SCHV_2018","TFI_2018", "VBR_2018","VEA_2018", "VOE_2018",
"VTI_2018", "VTIP_2018","VTV_2018", "VWO_2018"]
stocks_compl = ["AGG_2018"] + stocks
#-------------------------------------------------------------------------
'''
```

```python
#Start by reading the file of the first stock
aapl = pd.read_csv(stocks_compl[0]+'.csv', index_col=0)
#Get the column with the information of the closing prices
#The function pct_change does the percentage with respect
#to the previous value in the column
#Thus with the pct_change we go from prices to returns
pct_aapl = aapl["Close"].pct_change()
#We also save the prices so they may be used later
#on to compute different information
prices_aapl = aapl["Close"]
returns = pct_aapl.to_frame()
prices = prices_aapl.to_frame()

#Rename the column from "Close" to the stock name that it came from
returns  = returns.rename(columns={"Close": stocks_compl[0]})
prices  = prices.rename(columns={"Close": stocks_compl[0]})

#Do exactly the same for all the stocks in the list, and join all
#the dataframes together
for stock in stocks:
    df = pd.read_csv(stock+'.csv', index_col=0)
    df_pct = df["Close"].pct_change()
    df_price = df["Close"]
    returns = returns.join(df_pct).rename(columns={"Close": stock})
    prices = prices.join(df_price).rename(columns={"Close": stock})

#The pct_change leaves one NaN at the start of the dataframe, we must take that out.
returns = returns.dropna()
returns.head(), prices.head()


'''
#-----------------------------------------------------------------------
#TEST 4 JPMORGAN PAG 12 VALUES
stocks = ["ERJPMOGU INDEX","ERJPVLGU INDEX","JPFCCP02 INDEX"
,"JPFCVA01 INDEX", "JPVOFXB2 INDEX"]
stocks_compl = ["JMABDJSE INDEX"] + stocks
variable_read = "vals_12.csv"
#-----------------------------------------------------------------------
'''

'''
#-----------------------------------------------------------------------
#TEST 5 JPMORGAN PAG 16 VALUES
stocks = ["JMABDJSE INDEX","JPFCVA01 INDEX","JPFCVA05 INDEX",
"JPMZVEL1 INDEX", "JPMZVP4G INDEX"]
stocks_compl = ["ERJPMFGU INDEX"] + stocks
variable_read = "vals_16.csv"
#-----------------------------------------------------------------------
'''
```

```
'''
#----------------------------------------------------------------------------
#TEST 5 JPMORGAN PAG 19 VALUES
stocks = ["M2WO Index", "M2EF Index","M2US Index","M8EU Index",
"M8JP Index","M1AP Index", "LGAGTRUH Index",
          "LUAGTRUU Index", "LEATTREU Index", "LGCPTRUH Index",
          "LUACTRUU Index", "LP05TREH Index",
          "ADXY Index", "DXY Index", "BCOMTR Index"]
stocks_compl = ["M2WD Index"] + stocks
variable_read = "vals_19.csv"
#----------------------------------------------------------------------------
'''
'''
prices = pd.read_csv(variable_read, decimal=',', index_col=0)
prices = prices.dropna()
prices = prices[stocks_compl]
returns = prices.pct_change().dropna()
returns = returns[stocks_compl]
prices.head(), returns.head()
'''



#Build the correlation matrix and print it for visualization purposes.
corr = returns.corr()
#Vislualization purposes
ax = sns.heatmap(corr, cmap="coolwarm")
#Following the instructions of the HRP, build the distance matrix based
#on the correlation matrix.
d_corr = np.sqrt(0.5*(1-corr))
#The built in function linkage identifies the clusters and returns a matrix
#that describe the cluster formation
link = linkage(d_corr, 'single')
Z = pd.DataFrame(link)

#A dendrogram is built for visualization purposes.
fig = plt.figure(figsize=(25, 10))
dn = dendrogram(Z, labels = stocks_compl)
plt.show()


def get_quasi_diag(link):

    # sort clustered items by distance

    link = link.astype(int)

    # get the first and the second item of the last tuple
    sort_ix = pd.Series([link[-1,0], link[-1,1]])
    # the total num of items is the third item of the last list
    num_items = link[-1, 3]

    # if the max of sort_ix is bigger than or equal to the max_items
    while sort_ix.max() >= num_items:
        # assign sort_ix index with 24 x 24
        sort_ix.index = range(0, sort_ix.shape[0]*2, 2) # odd numers as index

        df0 = sort_ix[sort_ix >= num_items] # find clusters

        # df0 contain even index and cluster index
        i = df0.index
        j = df0.values - num_items #

        sort_ix[i] = link[j,0] # item 1

        df0  = pd.Series(link[j, 1], index=i+1)

        sort_ix = sort_ix.append(df0)
        sort_ix = sort_ix.sort_index()

        sort_ix.index = range(sort_ix.shape[0])


    return sort_ix.tolist()



sort_ix = get_quasi_diag(link)
stocks_compl = np.array(stocks_compl)
df_vis = returns[stocks_compl[sort_ix]]
corr2 = df_vis.corr()
ax = sns.heatmap(corr2, cmap="coolwarm")




def get_cluster_var(cov, c_items):
    cov_ = cov.iloc[c_items, c_items] # matrix slice
    # calculate the inversev-variance portfolio
    ivp = 1./np.diag(cov_)
    ivp/=ivp.sum()
    w_ = ivp.reshape(-1,1)
    c_var = np.dot(np.dot(w_.T, cov_), w_)[0,0]
    return c_var
def get_rec_bipart(cov, sort_ix):
    # compute HRP allocation
    # intialize weights of 1
    w = pd.Series(1, index=sort_ix)

    # intialize all items in one cluster
    c_items = [sort_ix]
    while len(c_items) > 0:
        # bisection
        """
        [[3, 6, 0, 9, 2, 4, 13], [5, 12, 8, 10, 7, 1, 11]]
        [[3, 6, 0], [9, 2, 4, 13], [5, 12, 8], [10, 7, 1, 11]]
        [[3], [6, 0], [9, 2], [4, 13], [5], [12, 8], [10, 7], [1, 11]]
        [[6], [0], [9], [2], [4], [13], [12], [8], [10], [7], [1], [11]]
        """
        c_items = [i[int(j):int(k)] for i in c_items for j,k in
                   ((0,len(i)/2),(len(i)/2,len(i))) if len(i)>1]

        # now it has 2
        for i in range(0, len(c_items), 2):

            c_items0 = c_items[i] # cluster 1
            c_items1 = c_items[i+1] # cluter 2

            c_var0 = get_cluster_var(cov, c_items0)
            c_var1 = get_cluster_var(cov, c_items1)

            alpha = 1 - c_var0/(c_var0+c_var1)

            w[c_items0] *= alpha
            w[c_items1] *=1-alpha
    return w



def compute_MV_weights(covariances):
    inv_covar = np.linalg.inv(covariances)
    u = np.ones(len(covariances))

    x = np.dot(inv_covar, u) / np.dot(u, np.dot(inv_covar, u))
    return pd.Series(x, index = stocks_compl, name="MV")


def compute_RP_weights(covariances):
    weights = (1 / np.diag(covariances))
    x = weights / sum(weights)
    return pd.Series(x, index = stocks_compl, name="RP")


def compute_unif_weights(covariances):
    x = [1 / len(covariances) for i in range(len(covariances))]
    return pd.Series(x, index = stocks_compl, name="unif")




cov = returns.cov()
```

```
weights_HRP = get_rec_bipart(cov, sort_ix)
new_index = [returns.columns[i] for i in weights_HRP.index]
weights_HRP.index = new_index
weights_HRP.name = "HRP"

weights_MV = compute_MV_weights(cov)
weights_RP = compute_RP_weights(cov)
weights_unif = compute_unif_weights(cov)

results = weights_HRP.to_frame()
results = results.join(weights_MV.to_frame())
results = results.join(weights_RP.to_frame())
results = results.join(weights_unif.to_frame())
results
```

```
def compute_ER(weights):
  mean = returns.mean(0)
  rt = weights.values * mean
  return (1 + rt)**252 -1

er_hrp = compute_ER(weights_HRP)
er_hrp.name = "HRP"
er_mv = compute_ER(weights_MV)
er_mv.name = "MV"
er_rp = compute_ER(weights_RP)
er_rp.name = "RP"
er_unif = compute_ER(weights_unif)
er_unif.name = "unif"

ers = er_hrp.to_frame()
ers = ers.join(er_mv.to_frame())
ers = ers.join(er_rp.to_frame())
ers = ers.join(er_unif.to_frame())
ers = ers.sum()
ers.name = "Expected Return"
ers = ers.to_frame()
ers
```

```
def portfolio_volatility(weights, cov):
  return np.sqrt(np.dot(np.dot(weights.values, cov.values), weights.values))*
  np.sqrt(252)

data = [portfolio_volatility(weights_HRP, cov)]
data.append(portfolio_volatility(weights_MV, cov))
data.append(portfolio_volatility(weights_RP, cov))
data.append(portfolio_volatility(weights_unif, cov))
```

```
volatility = pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns=["Volatility"])
volatility



def risk_free():
  return 0

def sharpe_ratio(weights, cov):
  ret_portfolio = compute_ER(weights).sum()
  ret_free = risk_free()
  volatility = portfolio_volatility(weights, cov)

  return (ret_portfolio - ret_free)/volatility

data = [sharpe_ratio(weights_HRP, cov)]
data.append(sharpe_ratio(weights_MV, cov))
data.append(sharpe_ratio(weights_RP, cov))
data.append(sharpe_ratio(weights_unif, cov))
sharpe_R= pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns=["Sharpe Ratio"])
sharpe_R



def compute_mdd(weights):
  df = weights * prices
  df = df.sum(1)
  roll_max = df.cummax()
  daily_drawdown = df/roll_max - 1.0

  #max_dd = daily_drawdown.cummin()
  #return max_dd.min()
  # Plot the results
  #daily_drawdown.plot(figsize=(20, 16))
  #max_dd.plot(figsize=(20, 16))
  #plt.show()
  return daily_drawdown.min()

data = [compute_mdd(weights_HRP)]
data.append(compute_mdd(weights_MV))
data.append(compute_mdd(weights_RP))
data.append(compute_mdd(weights_unif))
dd = pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"],
columns = ["Max DD"])
dd



def diversification_ratio(weights, cov):
  p_volatility = portfolio_volatility(weights, cov)
  return np.dot(np.sqrt(np.diag(cov.values)), weights) / p_volatility

data = [diversification_ratio(weights_HRP, cov)]
data.append(diversification_ratio(weights_MV, cov))
data.append(diversification_ratio(weights_RP, cov))
data.append(diversification_ratio(weights_unif, cov))
dr = pd.DataFrame(data = data, index=["HRP", "MV", "RP", "unif"], columns = ["Div Ratio
dr



final_results = ers.join(volatility)
final_results = final_results.join(sharpe_R)
final_results = final_results.join(dd)
final_results = final_results.join(dr)
final_results



#Cartera que se mueve en el tiempo

#------------------------------------------------------------------------
#TEST 4 JPMORGAN PAG 12 VALUES
stocks_compl = ["JMABDJSE INDEX", "ERJPMOGU INDEX","ERJPVLGU INDEX",
"JPFCCP02 INDEX","JPFCVA01 INDEX", "JPVOFXB2 INDEX"]
variable_read = "vals_12.csv"
#------------------------------------------------------------------------


'''
#------------------------------------------------------------------------
#TEST 5 JPMORGAN PAG 16 VALUES
stocks_compl = ["ERJPMFGU INDEX", "JMABDJSE INDEX","JPFCVA01 INDEX",
"JPFCVA05 INDEX","JPMZVEL1 INDEX", "JPMZVP4G INDEX"]
variable_read = "vals_16.csv"
#------------------------------------------------------------------------
'''

'''
#------------------------------------------------------------------------
#TEST 5 JPMORGAN PAG 19 VALUES
stocks_compl = ["M2WD Index", "M2WO Index", "M2EF Index","M2US Index",
"M8EU Index", "M8JP Index","M1AP Index", "LGAGTRUH Index",
          "LUAGTRUU Index", "LEATTREU Index", "LGCPTRUH Index",
          "LUACTRUU Index", "LP05TREH Index",
          "ADXY Index", "DXY Index", "BCOMTR Index"]
#stocks_compl = ["LGCPTRUH Index", "DXY Index", "M1AP Index",
"M2WO Index", "BCOMTR Index"]
variable_read = "vals_19.csv"
#------------------------------------------------------------------------
```

```
'''

prices = pd.read_csv(variable_read, decimal=',', index_col=0)
prices = prices.dropna()
prices = prices[stocks_compl]
returns = prices.pct_change().dropna()
returns = returns[stocks_compl]
prices.head(), returns.head()



returns.index = pd.to_datetime(returns.index)
prices.index = pd.to_datetime(prices.index)
returns.tail()


def compute_ER(data, weights, time):
  mean = data.mean(0)
  rt = weights.values * mean
  rt = rt.sum()
  return (1 + rt)**time

def portfolio_volatility(weights, cov, time):
  return np.sqrt(np.dot(np.dot(weights.values, cov.values), weights.values))*np.sqrt(time)



 pp_hrp = []
pp_mv = []
pp_rp = []
pp_unif = []

hrp = []
hrp_weights = pd.DataFrame(columns=stocks_compl)
mv = []
mv_weights = pd.DataFrame(columns=stocks_compl)
rp = []
rp_weights = pd.DataFrame(columns=stocks_compl)
unif = []
unif_weights = pd.DataFrame(columns=stocks_compl)

hrp_er = []
hrp_pv = []

mv_er = []
mv_pv = []

rp_er = []
rp_pv = []

unif_er = []
```

```
unif_pv = []

filenames = []

year = 2015
month = 1

k = 0
while(True):
  start = str(year)+'-'+str(month)
  if month == 12:
    month = 1
    year = year+1
  else:
    month+=1
  if year==2021:
    break

  end =  str(year)+'-'+str(month)
  mask = (returns.index > start) & (returns.index < end)
  train = returns.loc[mask]

  # HRP
  corr = train.corr()
  d_corr = np.sqrt(0.5*(1-corr))
  link = linkage(d_corr, 'single')
  sort_ix = get_quasi_diag(link)
  cov = train.cov()

  fig, ax = plt.subplots()
  sstocks = np.array(stocks_compl)
  df_vis = train[sstocks[sort_ix]]
  corr2 = df_vis.corr()
  im = ax.imshow(corr2, cmap='bwr')

  # We want to show all ticks...
  ax.set_xticks(np.arange(len(stocks_compl)))
  ax.set_yticks(np.arange(len(stocks_compl)))
  # ... and label them with the respective list entries
  ax.set_xticklabels(stocks_compl)
  ax.set_yticklabels(stocks_compl)

  # Rotate the tick labels and set their alignment.
  plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
          rotation_mode="anchor")
  filename = f'{k}.png'
  filenames.append(filename)
  # save frame
  plt.savefig(filename)

  # GUARDAR PESOS EN EL DATAFRAME
```

```
dicts={}
weights_HRP = get_rec_bipart(cov, sort_ix)
new_index = [returns.columns[i] for i in weights_HRP.index]
weights_HRP.index = new_index
weights_HRP.name = "HRP"
for i in range (len(stocks_compl)):
  dicts[stocks_compl[i]] = weights_HRP[i]
hrp_weights = hrp_weights.append(dicts, ignore_index=True)

weights_MV = compute_MV_weights(cov)
for i in range (len(stocks_compl)):
  dicts[stocks_compl[i]] = weights_MV[i]
mv_weights = mv_weights.append(dicts, ignore_index=True)

weights_RP = compute_RP_weights(cov)
for i in range (len(stocks_compl)):
  dicts[stocks_compl[i]] = weights_RP[i]
rp_weights = rp_weights.append(dicts, ignore_index=True)

weights_unif = compute_unif_weights(cov)
for i in range (len(stocks_compl)):
  dicts[stocks_compl[i]] = weights_unif[i]
unif_weights = unif_weights.append(dicts, ignore_index=True)

# GUARDAR RETORNOS EN LAS LISTAS
ss = prices[end].iloc[0]/prices[end].iloc[-1]
a = ss * weights_HRP
hrp.append(a.sum())

a = ss * weights_MV
mv.append(a.sum())

a = ss * weights_RP
rp.append(a.sum())

a = ss * weights_unif
unif.append(a.sum())

# GUARDAR EXPECTED RETURN
hrp_er.append(compute_ER(train, weights_HRP, 21))
mv_er.append(compute_ER(train, weights_MV, 21))
rp_er.append(compute_ER(train, weights_RP, 21))
unif_er.append(compute_ER(train, weights_unif, 21))

# GUARDAR PORTFOLIO VOLATILITY
hrp_pv.append(portfolio_volatility(weights_HRP, cov, 21))
mv_pv.append(portfolio_volatility(weights_MV, cov, 21))
rp_pv.append(portfolio_volatility(weights_RP, cov, 21))
unif_pv.append(portfolio_volatility(weights_unif, cov, 21))

# GUARDAR COSTES
```

```
  if k > 0:
    n = hrp_weights.iloc[k] - hrp_weights.iloc[k-1]
    n = n.abs()
    n = n * ss
    pp_hrp.append(n.sum()*0.001)

    n = mv_weights.iloc[k] - mv_weights.iloc[k-1]
    n = n.abs()
    n = n * ss
    pp_mv.append(n.sum()*0.001)

    n = rp_weights.iloc[k] - rp_weights.iloc[k-1]
    n = n.abs()
    n = n * ss
    pp_rp.append(n.sum()*0.001)

    n = unif_weights.iloc[k] - unif_weights.iloc[k-1]
    n = n.abs()
    n = n * ss
    pp_unif.append(n.sum()*0.001)



  k += 1




 fig = plt.figure(figsize=(25, 10))

# Productos cumulativos para hacer el return
cp_hrp = np.cumprod(np.array(hrp))
cp_mv = np.cumprod(np.array(mv))
cp_rp = np.cumprod(np.array(rp))
cp_unif = np.cumprod(np.array(unif))

# Grafica comparando todos los returns de los diferentes metodos (mes a mes)
plt.plot(cp_hrp, color="red")
plt.plot(cp_mv, color="blue")
plt.plot(cp_rp, color="green")
plt.plot(cp_unif, color="orange")
plt.xlabel("Time steps", fontsize=18)
plt.ylabel("Portfolio Value", fontsize=18)
plt.show()



# Aplicar el coste de rebalanceo a los returns
cost_hrp = np.zeros(len(hrp))
cost_mv = np.zeros(len(mv))
cost_rp = np.zeros(len(rp))
```

```
cost_unif = np.zeros(len(unif))

for i in range(len(hrp)):
  if i==0:
    cost_hrp[i] = hrp[i]
    cost_mv[i] = mv[i]
    cost_rp[i] = rp[i]
    cost_unif[i] = unif[i]
  else:
    cost_hrp[i] = (hrp[i] - pp_hrp[i-1]) * np.product(cost_hrp[i-1])
    cost_mv[i] = (mv[i] - pp_mv[i-1]) * np.product(cost_mv[i-1])
    cost_rp[i] = (rp[i] - pp_rp[i-1]) * np.product(cost_rp[i-1])
    cost_unif[i] = (unif[i] - pp_unif[i-1]) * np.product(cost_unif[i-1])



# Comparamos los returns de cada metodo teniendo en cuenta lo que cuesta
#rebalancear los pesos
fig = plt.figure(figsize=(25, 10))
plt.plot(cost_hrp, color="red")
plt.plot(cost_mv, color="blue")
plt.plot(cost_rp, color="green")
plt.plot(cost_unif, color="orange")
plt.xlabel("Time steps", fontsize=18)
plt.ylabel("Portfolio Value", fontsize=18)
plt.show()


# Grafica que compara los expected returns mes a mes para los diferentes metodos
fig = plt.figure(figsize=(25, 10))
plt.plot(hrp_er, color="red")
plt.plot(mv_er, color="blue")
plt.plot(rp_er, color="green")
plt.plot(unif_er, color="orange")
plt.xlabel("Time steps", fontsize=18)
plt.ylabel("Monthly expected return", fontsize=18)
plt.show()



#Grafica 1: Expected return vs Return real para HRP
fig = plt.figure(figsize=(25, 10))
plt.plot(hrp_er, color="red")
plt.plot(hrp, color="blue")
plt.xlabel("Time steps", fontsize=18)
plt.ylabel("Return", fontsize=18)
plt.show()

#Grafica 2: Expected return vs Return real pero acumulado para HRP
fig = plt.figure(figsize=(25, 10))
plt.plot(np.cumprod(hrp_er), color="red")
plt.plot(cp_hrp, color="blue")
plt.show()



#Grafica 1: Expected return vs Return real para Minimum Variance
fig = plt.figure(figsize=(25, 10))
plt.plot(mv_er, color="red")
plt.plot(mv, color="blue")
plt.xlabel("Time steps", fontsize=18)
plt.ylabel("Return", fontsize=18)
plt.show()

#Grafica 2: Expected return vs Return real pero acumulado para Minimum Variance
fig = plt.figure(figsize=(25, 10))
plt.plot(np.cumprod(mv_er), color="red")
plt.plot(cp_mv, color="blue")
plt.show()


#Grafica 1: Expected return vs Return real para RP
fig = plt.figure(figsize=(25, 10))
plt.plot(rp_er, color="red")
plt.plot(rp, color="blue")
plt.show()

#Grafica 2: Expected return vs Return real pero acumulado para RP
fig = plt.figure(figsize=(25, 10))
plt.plot(np.cumprod(rp_er), color="red")
plt.plot(cp_rp, color="blue")
plt.show()


#Grafica 1: Expected return vs Return real para uniforme
fig = plt.figure(figsize=(25, 10))
plt.plot(unif_er, color="red")
plt.plot(unif, color="blue")
plt.show()

#Grafica 2: Expected return vs Return real pero acumulado para uniforme
fig = plt.figure(figsize=(25, 10))
plt.plot(np.cumprod(unif_er), color="red")
plt.plot(cp_unif, color="blue")
plt.show()


fig = plt.figure(figsize=(25, 10))
plt.plot(pp_hrp, color="red")
plt.plot(pp_mv, color="blue")
plt.plot(pp_rp, color="green")
plt.plot(pp_unif, color="orange")
plt.show()


# Grafica que compara los pesos mes a mes para un metodo
fig = plt.figure(figsize=(25, 10))
colors = cm.jet(np.linspace(0,1,num=len(stocks_compl)))
for i in range(len(stocks_compl)):
  plt.plot(hrp_weights[stocks_compl[i]], color=colors[i])
plt.legend(stocks_compl, bbox_to_anchor=(1, 1))

plt.xlabel("Time steps", fontsize=18)
plt.ylabel("Weights", fontsize=18)
plt.show()


import imageio

# build gif
with imageio.get_writer('mygif.gif', mode='I') as writer:
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)
```

Bibliography

[1] Kilian Axel Schafer Harald Lohre Carsten Rother. *Hierarchical risk parity: Accounting for tail dependencies in multi-asset multi-factor allocations.* 2020.

[2] Emmanuel Jurczenko. *Machine learning for asset management, new developments and financial applications.* 2020.

[3] JP Morgan. *Hierarchical Risk Parity: Enhancing Returns at Target Volatility.* 2016.

[4] JP Morgan. *Systematic Strategies Across Asset Classes Risk Factor Approach to Investing and Portfolio Management.* 2013.

[5] Marcos Lopez de Prado. *Advances in Financial Machine Learning.* 2016.

[6] Marcos Lopez de Prado. *BUILDING DIVERSIFIED PORTFOLIOS THAT OUT-PERFORM OUT-OF-SAMPLE.* 2015.