ST7: Modélisation Mathématiques des Marchés Financiers

BNP Paribas

Portfolio Construction and Risk Minimization

Aimad Guallal

Edouard Waquier

Felipe Freire Pinto

Henrique Persin

Gif-sur-Yvette, 7/04/2021.

Contents

1 Introduction

In the framework of a project on Portfolio Construction supervised by a member of BNP Asset Management, the purpose of this paper is two fold: presentation of the results obtained after the construction and back test of several market portfolios and the discussion regarding their performance and risk profiles.

In this project, we have worked only with US equities as possible risky assets when constructing our portfolio.

To construct our different investment strategies, we we guided by the principles of Modern Portfolio Theory, constructed from the works of Markowitz, Tobin and Sharpe. Essentially, under MPT, an investor tries to maximize its expected returns whilst minimizing the risk simultaneously (6). The idea can be summarised by noticing that when working with US equities, there exists infinitely many possible combinations of them to construct an investment, but there are certain combinations that are capable to generate a maximum return per unit of risk. The main goal is thus to find this allocation strategy.

It is important to remark that this theory is founded upon some hypothesis such as the rationality of investor, the efficiency of markets, trade-off returns-risk, existence of a risk free asset that can be used to borrow an unlimited amount of money, nonexistence of transaction costs and symmetry of information. Some of this hypothesis can be easily contested, but with some modifications, the results obtained are still very useful.

In the development of the project, the group has done an extensive modeling work and analysis using various techniques in order to optimise the choice and allocation of asset. Throughout this paper, we will go through all these optimisation methods presenting their theoretical value before illustrating the pros and the cons using the results we obtained. For each of them, it is relevant to note that we will always consider the following constraint: $\sum_{i=1}^{n} x_i = 1$, ie. all our money is invested. There will be no constraint for long and short positions.

We will admit throughout this paper the following conventions: ω represents the weights of each stock in the portfolio, Σ is the covariance matrix of all considered assets, μ is the vector of expected returns for the assets. Finally, the results presented in sections 3 to 7 were obtained in a training simulation with 1 year horizon and a rebalancing of

the portfolio every 5 months. If not mentioned otherwise, the following stock were used: Apple ('AAPL'), Google ('GOOG'), IBM ('IBM'), Tesla ('TSLA'), BlackRock ('BLK'), Amazon ('AMZN'), Coty ('COTY') and Pfizer('PFE').

2 Forecasting Returns and Variance

The first step in constructing a portfolio is to have a clear input: a vector of expected returns and a matrix of variance-covariance of assets. This pair defines our risk-return curve and is thus a fundamental piece of Modern Portfolio Theory.

Predicting prices and estimating the variance of assets can be, by itself, a project apart due to the extreme complexity of financial data. Since the goal of the present project is not to develop predictive models, we will restrict ourselves to simpler statistical models, that even though lack predictive power, are capable of very well modelling the signal of the data and not its noise.

The models can be divided into two categories: for the returns and for the variance.

We will start with the returns time series. As we know, the prices are usually non-stationary and so before fitting any model, we need first to apply some transformation to the data to give it a stationary qualification. In order to avoid an intermediary step of differentiating our series enough so that it becomes stationary, we will use a ARIMA(p,d,q) to fit our data accordingly. This model has a differentiation parameter d that will help us remove the non-stationarity of the series, by differentiating it d times.

To allow for different behaviours of different stocks, it suffices to choose multiple values for the triplet (p,d,q) and verify which ones are for stationary series.

However, by trying different models, we introduce another degree of complexity by the need to choose one of those models. To do so, we have appealed to the Akaike information criterion (AIC) of information theory, that combines the performance of the model on the set and the amount of parameters required, thus reducing the likelihood of choice of an overfitted model.

In the term of performance, however, we don't know if the predictions with the ARIMA model will be good enough to start our work with optimization methods. Indeed, after a

few steps the prediction will become a constant value, which hardly reflects any market expectation and thus reduces its precision for longer time horizons.

As a way around this problem, we used a functionality available at the Yahoo finance library. With some extra information, we can increase or reduce the strength given to the signal of expected returns calculated earlier. The information used is the recommendations from banks and brokers to buy, sell or hold the studied stock. To combine these signals we introduced a scoring system to the recommendations attributing + 2 for a strong buy, + 1 for a buy, + 0.5 for a outperform and similarly - 2 for a strong sell, - 1 for a sell, - 0.5 for a underperform. After a normalization step, both signals can be combined to generate a prediction for the expected returns. This new signal is expected to encompass information on market expectations and price evolution's.

Finally, to estimate the covariance matrix of our returns, a two step process has been done. With a simple estimator, we have determined the correlation between the assets which we can postulate that remains constant over the analysed period. Then, by using a GARCH model, we calculated the prediction at horizon t of the variance of each asset. By combining these results, we have the desired covariance matix. We have decided to use this model for the variance, over evaluating it from the ARIMA series or using a VGARCH, due to its robustness in fitting and lower complexity for the data.

3 Minimum Variance Portfolio

We will start our analysis with the Minimum Variance Portfolio, which is a very straightforward strategy in the eye of MPT. The idea is to find a combination of weights ω such that the total portfolio variance is minimized. However, we constraint our search for weights such that the overall return is greater than a predefined lower bound. This strategy is built on the risk averse profile of investors and the existing trade-off between risk and returns.

The problem can be written mathematically as follows: find $\omega \in argmin\ \omega^T \Sigma \omega$ such that $\mu^T \omega \geq \mu^*$ for a certain limit $\mu^* > 0$.

However, as we can see in the two figures bellow, the cumulative returns path is quite dependent on the lower limit of the portfolio returns μ^* and in the input vector μ. We

must note that this optimisation was done with the additional constraint that $\sum \omega_i = 1$, and that's why the results shown appear to be upside down. We would expect higher returns for the green curve and less returns for the blue one, while the variance having the opposite behavior. Indeed, if we remove the constraint we get the result shown bellow.

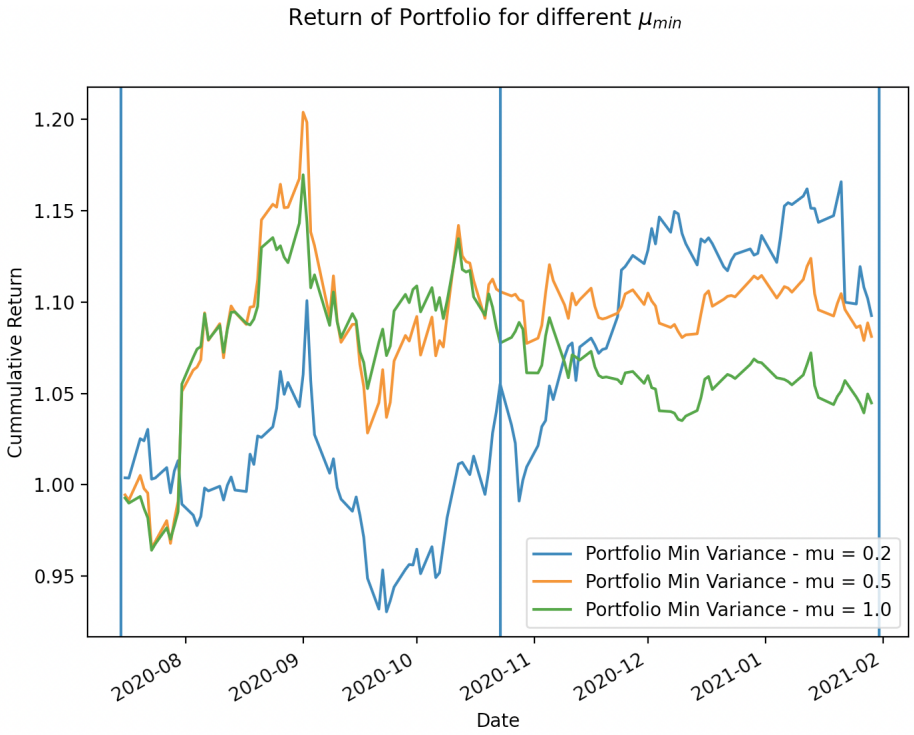

Figure 1: Cumulative returns path as a function of μ^*

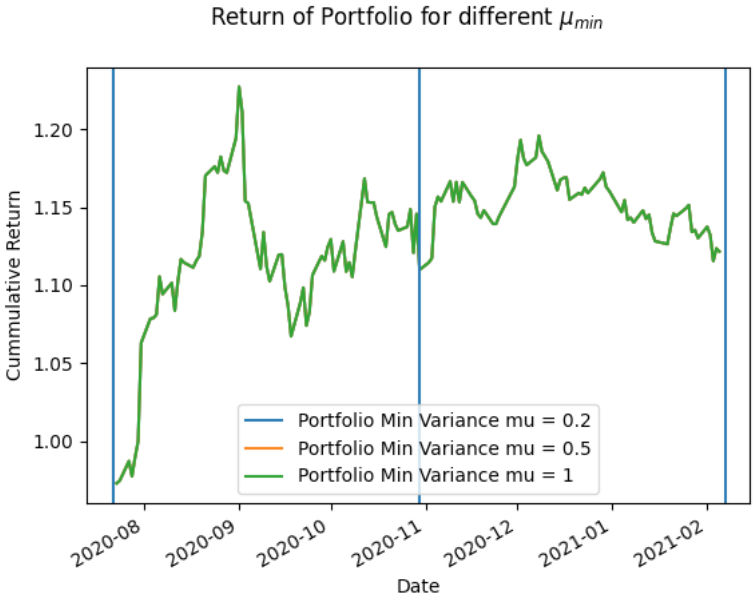

Figure 2: Minimum Variance Results without the sum constraint

It is also interesting to see how the method responds to noisy data or uncertain input.

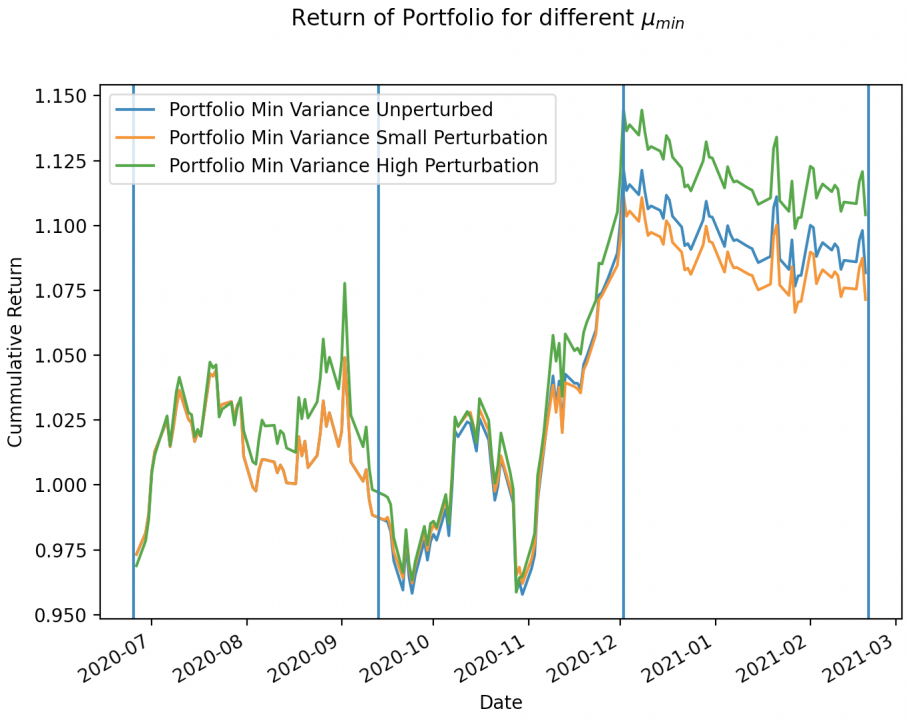

Figure 3: Cumulative Returns Perturbed

The noise that was introduced in figure 2, was done by adding a gaussian vector of variance 2% and 20%. We can see that at the end, the difference between those portfolios is much larger than these percentages. This sensitivity to parameters is a problem when it comes to financial data since it is highly noisy. Hence, we would expect multiple changes in the trajectory after rebalancing the portfolio, which when introducing transaction costs can be a great disadvantage.

Since there is a great influence from the hyperparameter μ^* in the returns, we can try and optimise the Sharpe ratio for the training set by setting multiple strategies with different lower bounds. Note that this estimate of the Sharpe Ratio was done with a zero risk free return and that's why it is significantly high. Moreover, we can compare the risk profile of the strategies with the Value at Risk for different values of μ^*. The results are shown bellow:

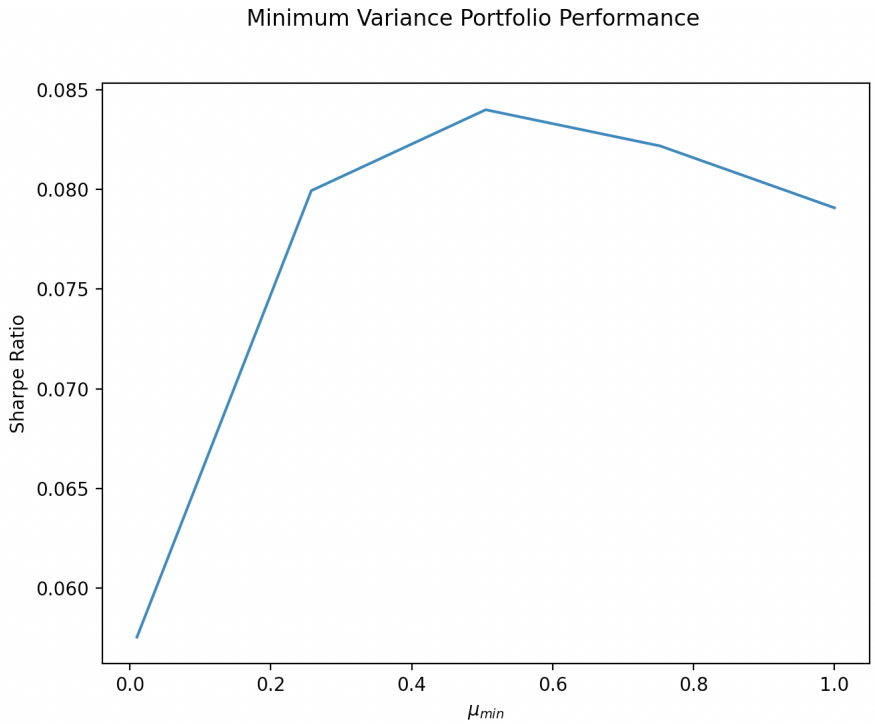

Figure 4: Minimal Variance Sharpe versus μ^*

μ^*	VaR 95 at 1mo horizon
0.0100	-0.086276
0.2575	-0.088107
0.5050	-0.092981
0.7525	-0.093209
1.0000	-0.092418

Table 1: Value at Risk for different levels of μ^*

We observe that for values close to 0.5, the Sharpe ratio is maximum but VaR is more elevated. Depending on the investor, the hyperparameter choice can lead to different results.

4 Markowitz Portfolio

In this section, we will show the results obtained with the Markowitz Portfolio, which its problem can be written as follows : find $\omega \in argmax\ \mu^T\omega$ under constraints such that $\omega^T\Sigma\omega \leq \sigma^*$, where ω and Σ are the same as above.

This method is similar to the previous one and consist of maximizing returns for a limited amount of variance. As we did before, we can evaluate the impact of choosing different upper bounds for the portfolio variance in the following figure:

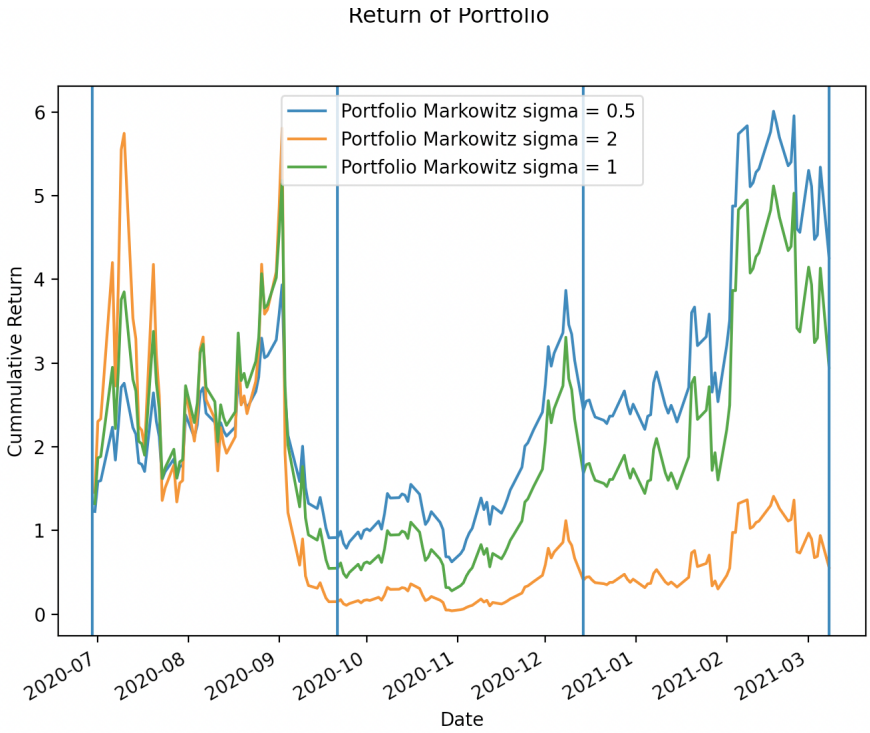

Figure 5: Cumulative returns path as a function of σ^*

For the lowest σ^*, we notice that returns are quite important comparing to the highest one in our case. Hence, that is not seem significant since we expect increasing returns with highest risk i.e, when we give a high value of the maximum variance σ^*. This is due to the fact that we predict μ and σ , so probably this may be take into account in our analysis. And to verify the impact of σ, the figure bellow shows the change of Sharpe ratio:

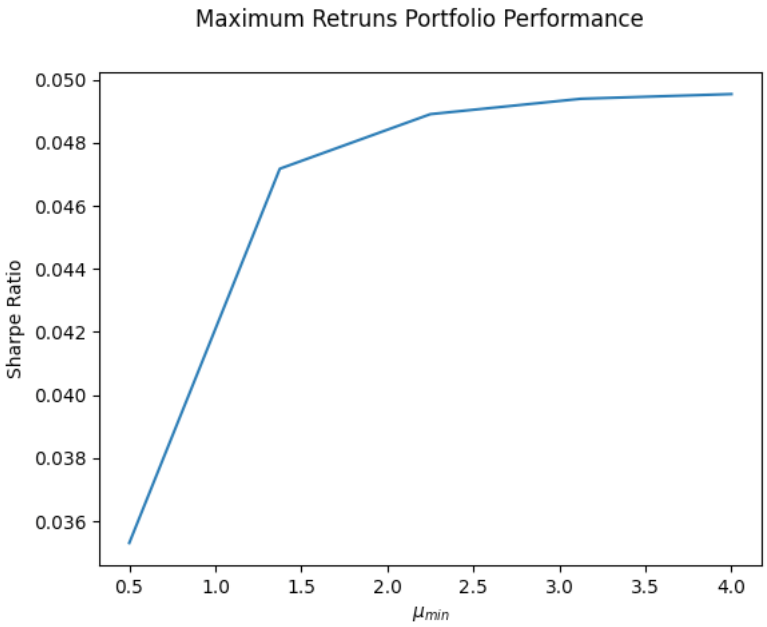

Figure 6: Sharpe Ratio for evolving σ^*

Finally, we can study the Value at Risk for different values of our upper bound to better evaluate this strategy. For this consider the table bellow:

σ^*	VaR 95 at 1mo horizon
0.500	-0.072293
1.375	-0.072460
2.250	-0.072289
3.125	-0.072168
4.000	-0.072083

Table 2: Value at Risk for Markowitz Portfolio

We can verify that up to $\sigma^* = 4$ we have the Sharpe Ratio evolving with σ^* and VaR decreasing (but increasing in absolute terms), indicating that a compromise should be made according to investor profile.

5 Risk-based Portfolios

In this section, we will be discussing some of the risk-based portfolios. The main difference in this approach is that the vector is expected returns will not be considered as

an input to the optimisation problem.

When it comes to implement some of the most classical strategies, as the Markowitz mean-variance solution, there are several challenges, as the uncertain characteristic of the expected returns, which introduces some noise in the portfolio optimisation problem, leading to an inconsistent asset allocation.

As another fact to take into account are the considerations made by this type of strategy, which can be considered questionable, is related with the expected returns, which are considered equal among the assets in the portfolio.

By disconsidering the expected returns parameter, the risk-based strategies are intrinsically considering that the returns are equal for all stocks. This hypothesis may seem a bit questionable at first, but by doing so, we can reason the problem through a risk-only approach and minimize to the investor some measure of variance of his portfolio.

As some of examples for this kind of approach, we can consider the Equally Weighted strategy, the Inverse Variance, Equal Risk Budget and the Minimum Variance (which has already been explained previously on this paper, but is now introduce with $\mu = 1$).

If we recall the definitions given above, and introduce the marginal contribution of risk as:

$$MRC_i = \frac{\partial \sigma_p}{\partial w_i} = \frac{\sigma_{ip}}{\sigma_p} \tag{1}$$

where, σ_{ip} is the covariance of the i-th asset and the portfolio. We can then formulate the problem mathematically as follows:

$$min\ D(f(w_i; \gamma, \delta)) s.t. \sum_{n=1}^{\infty} w_i = 1 \tag{2}$$

where f can be interpreted as modified risk contributions and D is a dispersion metric. (4).

Here, the function f can be defined as $f(w_i; \gamma, \delta) = \frac{w_i^{\gamma}}{\sigma_i^{\delta}} \times MRC_i$. One can change the parameters γ and δ order to construct a risk based portfolio.

As explained earlier, the objective must be minimize some risk measure without taking into account the return of the portfolio. Considering this condition, one should expect a slight weight for assets which grows the portfolio overall risk. The comparison between

the strategies can be seen in the image bellow.

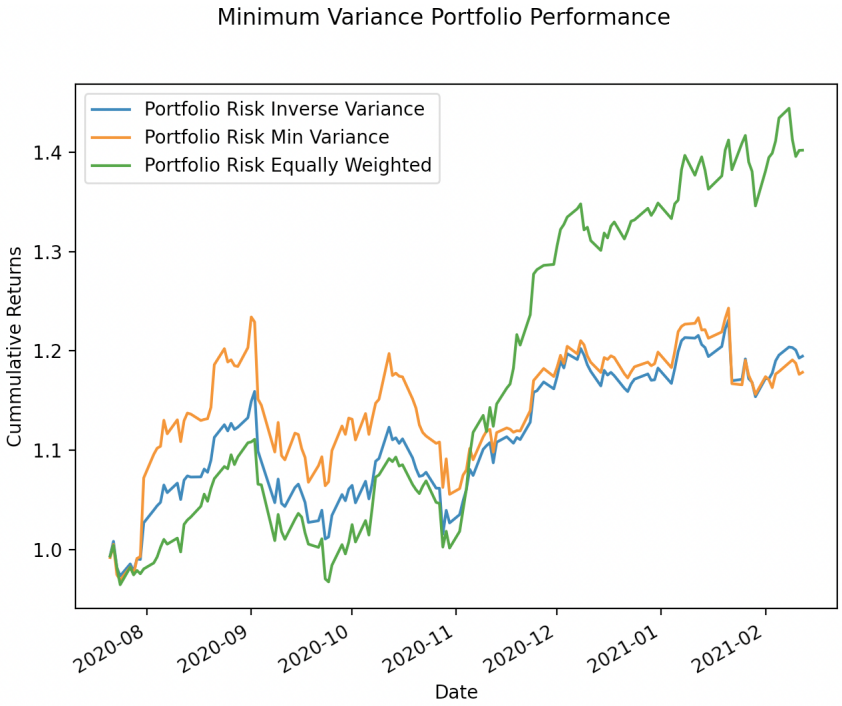

Figure 7: Cumulative returns of Risk-based portfolios

Furthermore, the comparison can be extendend in the following table, where Sharpe Ratio and Value at Risk are presented.

Strategy	VaR 95 at 1mo horizon	Sharpe Ratio
Portfolio Risk Inverse Variance	-0.101982	0.025244
Portfolio Risk Min Variance	-0.116675	0.012630
Portfolio Risk Equally Weighted	-0.109747	0.167621

Table 3: Risk based Portfolios Comparison

From the figure, it is interesting to notice the difference between the return of the Equally Weighted strategy compared to the other two presented. One can state that the higher return comes from a higher exposure to market risks, size and value factors. It is possible to show that there is an advantage when it comes to the Sharpe ratio, which shows the one other possible advantage of the Equally-Weighted portfolio. However, when it comes to the Value at Risk, it is the Inverse Variance that carries an edge.

6 Robust Portfolio

The idea behind this portfolio technique comes from the observation that the calculation of the expected returns can be uncertain and that this can greatly impact the weights on our portfolio. Indeed, as we saw in the previous sections, when using traditional strategies such as those presented before, changing the input of the vector μ even by small amounts can lead to very different results at the end.

Thus, we will introduce to our problem the vector of true returns and we will suppose that our observation is only enough to give us an approximation of its value. Therefore, when we are solving our minimization problem, we need to consider the fact that we are uncertain of the returns and so our cost can be as low or as high depending on the boarder of our uncertainty frontier. It is then necessary to optimize the problem with respect to the worst case scenario ie, the maximum value taken by our cost function for μ in the defined region.

This region will be defined by an elliptical form, identified by Ω a symmetric positive definite matrix. The robust portfolio is a result of the following optimisation problem:

$$max\ \mu^T\omega - \kappa\sqrt{\omega^T\Omega\omega} - \lambda\omega^T\Sigma\omega \tag{3}$$

where, λ is a risk aversion coefficient and κ, a error aversion term.

For a first simulation of the method, consider an example where we have taken returns for Apple (AAPL), Amazon (AMZN), Google (GOOG), IBM (IBM) and Tesla (TSLA). In this example, we will compare the impact of the additional error term, by comparing the following three scenarios: $\kappa = 0$, $\kappa \neq 0$ and $\Omega = \Sigma$ and finally $\kappa \neq 0$ and $\Omega = diag(\Sigma)$. The results are presented in the figure bellow. We took $\kappa = 5$ and $\lambda = 8$ for the simulations.

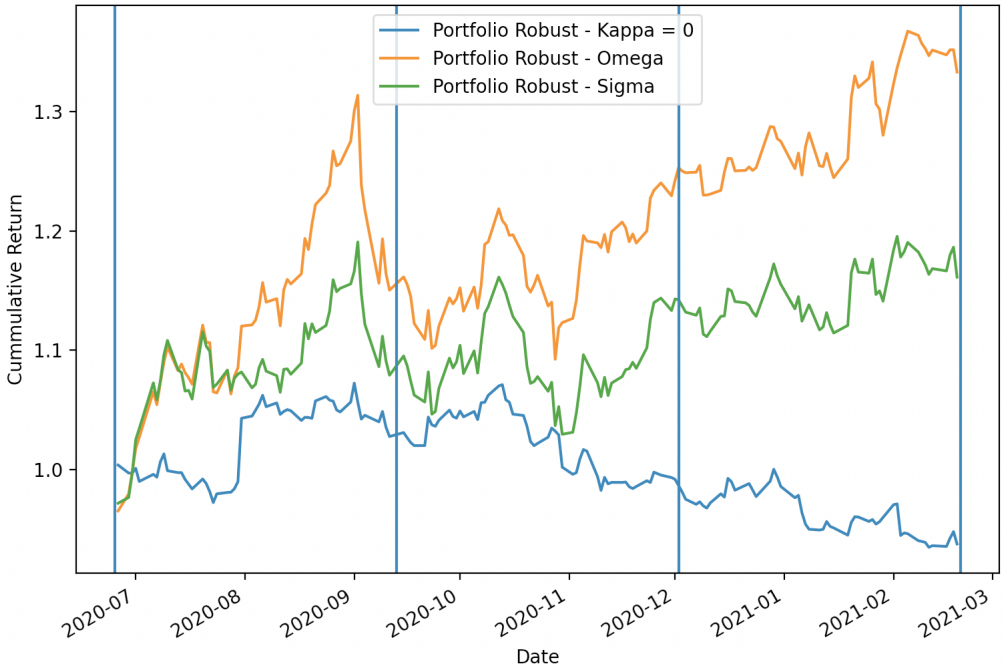

Figure 8: Influence of the Error Term on Optimisation on Cumulative returns

As we can see, the addition of the error term can greatly improve the portfolio's asset allocation for the same set of inputs. This is even more the case considering that the model developed to determine the expected returns is not too robust. Moreover, we can see that the definition of this uncertainty region also affects our final results. We observe a higher cumulative return in the case of the diagonal matrix as an input which makes sense since it is compatible with the hypothesis that our errors are independent of each other. If not, we add supplementary variables that change the source of uncertainty and propagates it trough the assets, leading to a more conservative result.

This comparison can be further extended in the table bellow. We can see, by introducing the error term we are significantly increasing the Sharpe Ratio, but the Value at Risk also increases (in absolute value). Moreover, by choosing a smaller parametric definition, we may be able to increase Sharpe, but we also suffer from a higher risk.

Strategy	VaR 95 at 1mo horizon	Sharpe Ratio
Portfolio Robust kappa = 0	-0.074104	0.030723
Portfolio Robust Sigma	-0.084224	0.056823
Portfolio Robust Omega	-0.093062	0.085549

Table 4: Sharpe and VaR Comparison for the Uncertainty Region

In order to assess the performance of the model vis-à-vis uncertain inputs, consider the scenario where the expected returns are perturbed by a white noise ϵ. For this, we will consider the response to the unperturbed signal followed by the response for a small perturbation ($\mathcal{N}(0, 5\%)$) and a high perturbation ($\mathcal{N}(0, 20\%)$). We observe that, even though we are adding a term that is roughly of the order of 1/10th and 1/5th of the returns, the final result doesn't seem to be that far apart. This is the case since we introduced an error term to our optimisation that makes it more robust to problems such as this.

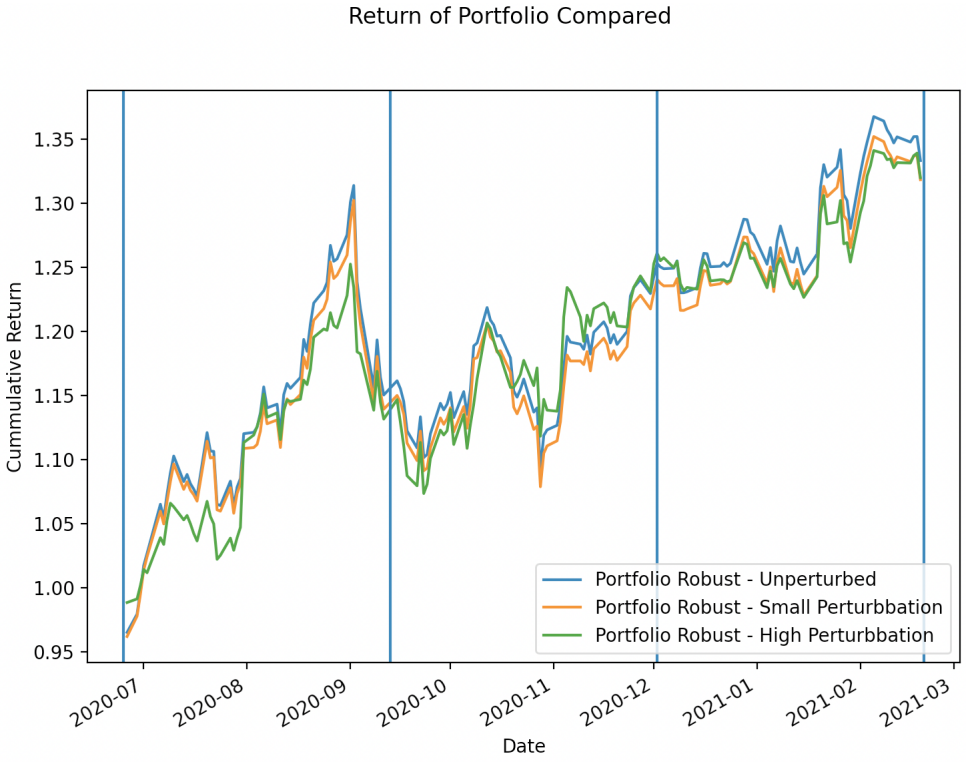

Figure 9: Robust Portfolio Performance with Noise

Finally, it is important to verify the impact that the values of κ and λ can have in our final result. In this analysis, we will construct a heat-map for the Sharpe ratio, for the case $r_f = 0$, ie. with zero risk less return, for varying values of κ and λ.

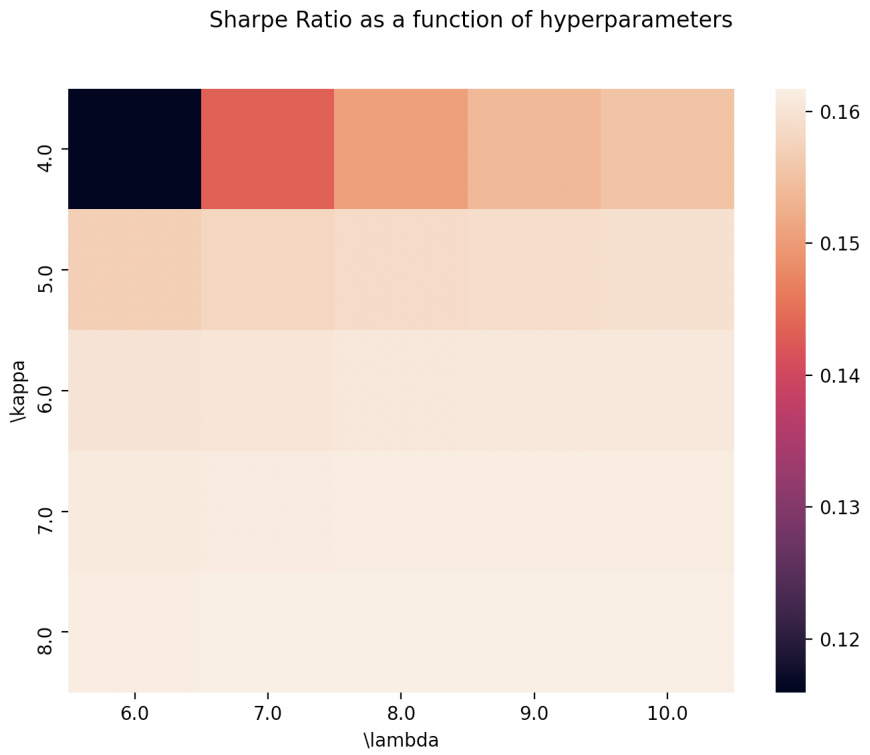

Figure 10: Performance of Robust Optimisation as a function of parameters

7 Black Litterman Portfolio

A final portfolio we have considered was constructed with the Black-Litterman model. The difference from this model to all others is the fact that it is constructed using Bayes' Theory in Statistics. In general, it is composed of a prior and a likelihood distributions that combined give us a more general form of our asset's return and variance, that can then be used as input to the optimisation.

First of all, in order to construct the prior, we use market information and reverse optimise the market portfolio to find the equilibrium excess return. This is done using the market capitalization value of the companies whose stock we are evaluating and the price variance-covariance matrix.

Afterwards, we will use our predictions to calculate the likelihood distribution parameters and then define our posterior function. Following this algorithm, we are then able to input more robust estimates of the returns expectations and variance to some optimiser.

In this part, we have used a more quantitative form of the views vector, as being the

output from our prediction step.

Consider the following example, where we have taken data from Apple (AAPL), IBM (IBM), BlackRock (BLK), Amazon (AMZN), ExxonMobbil (XOM) and Nike (Nike). Since we are required additional information such as historical values of market cap, we will need to restrict our universe to the stocks for which this information is readily available.

The first comparison shows the difference of cumulative results for the Black Litterman model and the Minimum Variance model, which is in turn also used to find the optimal weights of the former. The idea is to verify the impact of the prior and likelihood combination. The figure bellow compares them both for the same set of parameters.

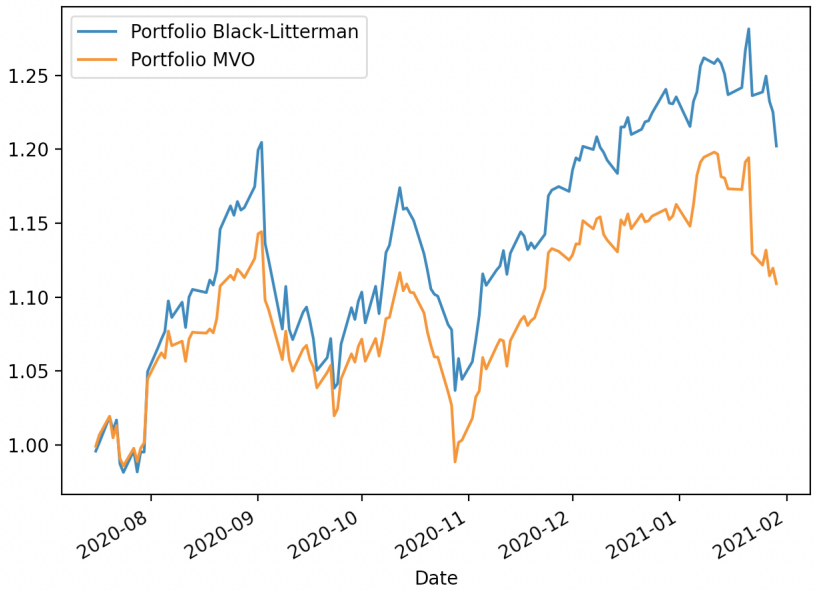

Figure 11: Cumulative Returns Comparison

As we can see, there is a great impact on the cumulative returns of both series when simulated. This clearly indicates that our knowledge about the expected returns is better modeled by some distribution for which we can better determine its prior and likelihood distributions. Nonetheless, we can verify whether this new uncertainty on the parameters is similar to the one introduced in the robust optimisation by comparing both models when Black-Litterman is optimised with the robust objective function. As we can see in the figure bellow, the difference is much narrower as if the new model doesn't introduce much more hedging capacity on the risk-return pair.

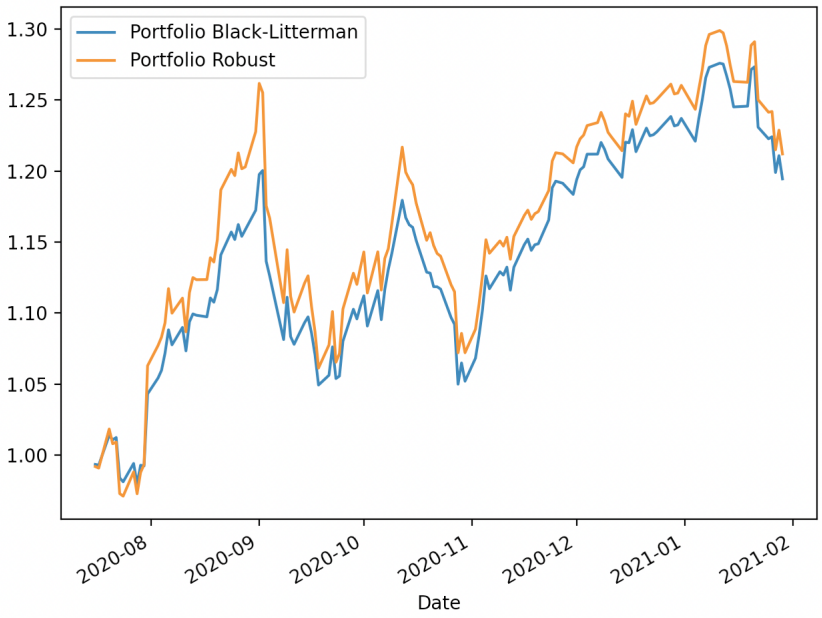

Figure 12: Cumulative Returns Comparison

8 Comparison between Models

Now, having developed those models and tested them separately, trying to find optimal hyperameters on a training set and validating its results, it is natural to compare them in a simulation with the exact same data. Moreover, we will introduce a higher rebalancing frequency of the portfolio, which will recalculate its weights every 2 months. The result can be seen in the image bellow.

We can clearly see different returns profiles, that ranged from negative 1% to almost 35%. If we compare it to some traditional benchmarks, we have that the SP 500 had a 28% increase (2) and the Dow Jones Industrial Average had a 47.5% increase (3).

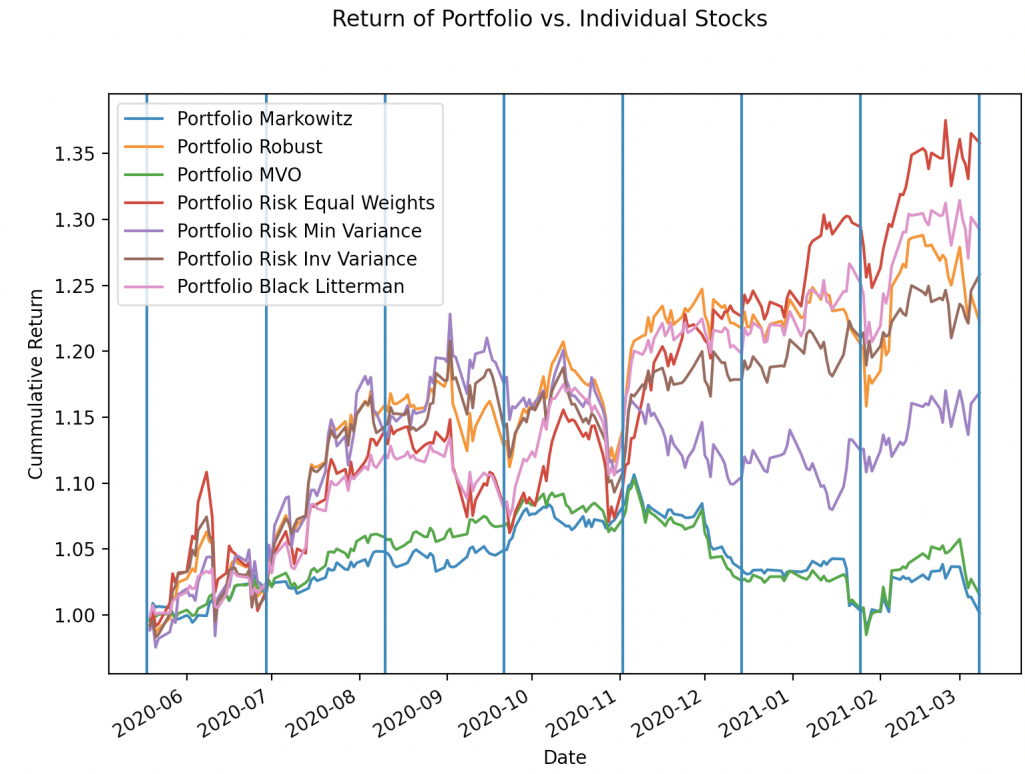

Figure 13: Backtest for 1 year

However, one cannot limit its analysis to simple returns. We can analyse other aspects of the portfolios as the Sharpe ratio, the Value at Risk and Maximum Drawdown. The following tables present this information.

It is important to remark that the risk free rate has been taken equal to zero, in order to neglect the discounted prices process.

Table 5: Sharpe Ratio Comparison

Portfolio	Sharpe Ratio
Portfolio Markowitz	0.004315
Portfolio Robust	0.091930
Portfolio MVO	0.015510
Portfolio Risk Equal Weights	0.130471
Portfolio Risk Min Variance	0.067095
Portfolio Risk Inv Variance	0.108737
Portfolio Black Litterman	0.138705

Table 6: Value at Risk Comparison

Portfolio	VaR 95 at 1mo horizon	VaR 99 at 1mo horizon
Portfolio Markowitz	-0.041819	-0.059063
Portfolio Robust	-0.086174	-0.122125
Portfolio MVO	-0.045687	-0.064554
Portfolio Risk Equal Weights	-0.090598	-0.128591
Portfolio Risk Min Variance	-0.095263	-0.134875
Portfolio Risk Inv Variance	-0.082107	-0.116439
Portfolio Black Litterman	-0.070333	-0.099861

Table 7: Maximum Drawdown Comparison

Portfolio	Maximum 1 week Drawdown (%)
Portfolio Markowitz	-8.716032
Portfolio Robust	-4.379877
Portfolio MVO	-6.126360
Portfolio Risk Equal Weights	-3.555846
Portfolio Risk Min Variance	-4.188754
Portfolio Risk Inv Variance	-4.639988
Portfolio Black Litterman	-4.174433

As we can see, those tables put into perspective some of the core characteristics of our allocation strategies. In our case, maximum returns doesn't imply highest Sharpe nor does smallest VaR give us a smaller drawdown. The choice of the portfolio depends greatly on the risk and the return seeking profile of the investor. There is no single strategy that stands out from the rest.

Even though those results are interesting, we can still add another constraint to approximate the problem to the real world. Every transaction incurs with some cost and so an investor is wise to maximize his or hers utility in a neighbourhood of the initial weights. In order to introduce this phenomena, we will add a term on the objective function that

writes as:

$$\psi \times |\omega - \omega_{old}| \tag{4}$$

where ω is the variable we seek to optimise and ω_{old} are the previous weights. This term will be added as a penalty term to the Markowitz, Minimum Variance and Robust Portfolios in the appropriate form.

The results for the trajectory of the returns is shown bellow for $\psi = 0.3$

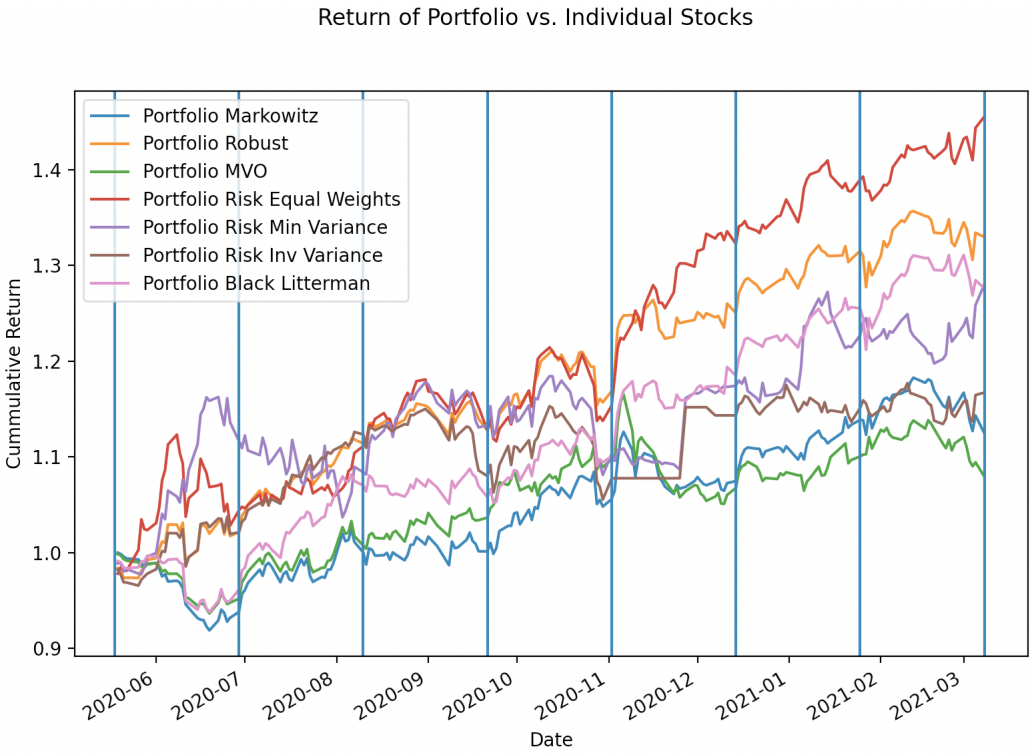

Figure 14: Backtest with transaction costs

The returns for the non-changed portfolios remain practically the same, whereas for the 3 others we see that this constraint regularized the weights in a way that reduced noise, giving them a similar, if not better, performance.

A more in depth analysis is presented in the tables bellow.

Table 8: Sharpe Ratio with Transaction Costs

Portfolio	Sharpe Ratio
Portfolio Markowitz	0.065819
Portfolio Robust	0.147983
Portfolio MVO	0.045527
Portfolio Risk Equal Weights	0.171094
Portfolio Risk Min Variance	0.097541
Portfolio Risk Inv Variance	0.078379
Portfolio Black Litterman	0.136404

Table 9: Value at Risk with Transaction Costs

Portfolio	VaR 95 at 1mo horizon	VaR 99 at 1mo horizon
Portfolio Markowitz	-0.072171	-0.102175
Portfolio Robust	-0.073887	-0.104945
Portfolio MVO	-0.070347	-0.099514
Portfolio Risk Equal Weights	-0.083928	-0.119317
Portfolio Risk Min Variance	-0.101620	-0.144047
Portfolio Risk Inv Variance	-0.079147	-0.112107
Portfolio Black Litterman	-0.068739	-0.097588

Table 10: Maximum Drawdown with Trasaction Costs

Portfolio	Maximum 1 week Drawdown(%)
Portfolio Markowitz	-118.412738
Portfolio Robust	-3.345952
Portfolio MVO	-7.208853
Portfolio Risk Equal Weights	-2.691090
Portfolio Risk Min Variance	-14.075403
Portfolio Risk Inv Variance	-3.130739
Portfolio Black Litterman	-7.504981

We can conclude that the introduction of this new cost structure, that is not taken

into account in Modern Portfolio initial Theory can impact significantly the performance of the models. Indeed, the introduction of costs after any transaction seems reasonable if we think that some investors can be price makers in the market. Hence, every transaction that they make incurs some change in the price that wasn't accounted for and that may, nonetheless, be sufficient to surpass the signal of expected returns.

Moreover, by including this penalty term, we are expected to move more in directions that have greater impact in the returns-risk pair and so, we could see an improvement in its performance.

We can as a last comparison instrument, bring the information concerning two base line indicators such as the S&P 500 and the Dow Jones Industrial Average. In view of this table, we can see that when constructing portfolios, the result can be very different even if we start off with the same universe of stocks.

Base Line Indicator	Sharpe	VaR 1mo 95%	Maximum Drawdown
S&P 500	0.139337	-0.089648	-3.285959
Dow Jones	0.125356	-0.091957	-2.797382

Table 11: Base line comparison

9 Conclusion

From the previous sections, we can see that our models perform differently and according to some performance indicator, they can fit different investor's profile. However, it is surprising to a certain extent that the risk based models had a better overall performance than some of the more elaborate ones. This may be in turn caused by the oversimplification of the original problem of predicting stock future prices. This is even more clear when we see that strategies that model the uncertainty perform better than those that don't.

Obviously the results obtained here present a certain level of simplification, to start off with the nonexistence of transaction costs and the risk free rate being taken equal to zero.

Therefore, it would be interesting to validate the results with the usage of a more

robust predictor of returns and the introduction of more realistic hypothesis, though the overall conclusion still remains valid.

10 Appendix: Code

All our code can be accessed via the following link on GitHub. Nonetheless, all parts will be presented in this appendix.

10.1 Expected Returns

```
class mu():
    def __init__(self, ticker_series, tickers, rebalancing_freq):
        aux = zip(ticker_series, tickers)
        self.expected_returns =
            np.array([self.fit_ARIMA(ticker_series[ticker_time_series],
            rebalancing_freq) for ticker_time_series in ticker_series])
        self.recommendations =
            np.array([self.fit_recommendations(ticker_series[ticker_time_series],
            ticker_name) for ticker_time_series, ticker_name in aux])
    def fit_ARIMA(self, time_series, rebalancing_freq):
        best_perf = np.inf
        for p_ in range(0,5):
            for d_ in range(0,3):
                for q_ in range(0,5):
                    model = ARIMA(endog=time_series, order = (p_,d_,q_),
                        enforce_stationarity=True)
                    try:
                        model_fit = model.fit()
                        current_perf = model_fit.aic
                        if current_perf < best_perf:
                            mu =
                                (np.prod(model_fit.forecast(rebalancing_freq).values
                                + 1))
```

```
                        best_perf = current_perf
                except:
                    pass
        return mu
    def fit_recommendations(self, time_series, ticker_name):
        start_date = time_series.index.values[0]
        end_date = time_series.index.values[-1]
        try:
            recommendations =
                yf.Ticker(ticker_name).recommendations.loc[start_date:end_date].groupby(['To
                Grade']).count()['Firm']
            recommendations_attr = list(recommendations.index)

            score = 0
            if 'Buy' in recommendations_attr:
                score += (1 * recommendations.loc['Buy'])
            if 'Sell' in recommendations_attr:
                score -= (1 * recommendations.loc['Sell'])
            if 'Strong Buy' in recommendations_attr:
                score += (2 * recommendations.loc['Strong Buy'])
            if 'Strong Sell' in recommendations_attr:
                score -= (2 * recommendations.loc['Strong Sell'])
            if 'Outperform' in recommendations_attr:
                score += (0.5 * recommendations.loc['Outperform'])
            if 'Underperform' in recommendations_attr:
                score -= (0.5* recommendations.loc['Underperform'])

            if len(recommendations_attr) > 0:
                return score/len(recommendations_attr)
            else:
                return 0
        except:
            return 0
    def get_expected_returns(self):
```

```
        return self.expected_returns
    def get_recommendations(self):
        return self.recommendations
    def get_mu(self):
        return (self.recommendations/sum(abs(self.recommendations)) +
            self.expected_returns/sum(abs(self.expected_returns)))
```

10.2 Covariance Forecast

```
class Sigma():
    def __init__(self, ticker_series, time_horizon):
        self.time_horizon = time_horizon
        self.variance =
            np.diag([self.compute_variance(ticker_series[ticker_time_series],self.time_horizon)
            for ticker_time_series in ticker_series])
        self.corr = self.compute_correlation(ticker_series)
        self.sigma = self.compute_sigma(self.variance, self.corr,
            self.time_horizon)
    def compute_correlation(self, ticker_series):
        P = ticker_series.corr(method = 'pearson').to_numpy()
        return P
    def compute_variance(self, time_series, time_horizon):
        best_perf = np.inf
        for p_ in range(1, 10):
            for q_ in range(1, 10):
                model = arch_model(time_series, p = p_, q = q_, mean = 'constant')
                model_fit = model.fit()
                current_perf = model_fit.aic
                if current_perf < best_perf:
                    var = (model_fit.forecast(horizon =
                        time_horizon).variance.values)[-1][0]
                    best_perf = current_perf
        return var
```

```
    def compute_sigma(self, sigma, P, time_horizon):
        return time_horizon * (np.sqrt(sigma) @ P @ np.sqrt(sigma))
    def get_variance(self):
        return self.variance
    def get_correlation(self):
        return self.corr
    def get_sigma(self):
        return self.sigma
```

10.3 Dataloader

```
class Dataloader():
    def __init__(self, period, tickers_list, rebalacing_freq_months):

        self.period = period
        self.tickers = tickers_list
        self.rebalacing_freq_months = rebalacing_freq_months

        data_close =
            pd.DataFrame(yfinance.Ticker(tickers_list[0]).history(period =
            period).Close)
        data_close.columns = [tickers_list[0]]

        data_open = pd.DataFrame(yfinance.Ticker(tickers_list[0]).history(period
            = period).Open)
        data_open.columns = [tickers_list[0]]

        data_volume =
            pd.DataFrame(yfinance.Ticker(tickers_list[0]).history(period =
            period).Volume)
        data_volume.columns = [tickers_list[0]]

        for ticker in tickers_list[1:]:
```

```
            ticker_close = pd.DataFrame(yfinance.Ticker(ticker).history(period =
                period).Close)
            ticker_close.columns = [ticker]

            ticker_open = pd.DataFrame(yfinance.Ticker(ticker).history(period =
                period).Open)
            ticker_open.columns = [ticker]



            ticker_volume = pd.DataFrame(yfinance.Ticker(ticker).history(period
                = period).Volume)
            ticker_volume.columns = [ticker]
            if len(ticker_close.index) != 0:
                data_close = data_close.join(ticker_close, how = 'outer')
                data_open = data_open.join(ticker_open, how = 'outer')
                data_volume = data_volume.join(ticker_volume, how = 'outer')
            else:
                print('Ticker not Available')

        self.close_prices = data_close
        self.open_prices = data_open
        self.volume = data_volume

    def get_close(self):
        start_index = self.close_prices.index[0]
        stop_index = self.close_prices.index[-1] - timedelta(days =
            self.rebalacing_freq_months)
        close_prices_for_rebalancing = []
        dates = []
        while start_index <= stop_index:
            end_rebalancing = start_index + timedelta(days =
                self.rebalacing_freq_months)
            close_df = self.close_prices.loc[start_index : end_rebalancing]
```

```
            close_prices_for_rebalancing.append(close_df)
            start_index = end_rebalancing
            dates.append(start_index)

        return dates, close_prices_for_rebalancing

    def get_all_close(self):
        return self.close_prices

    def get_open(self):
        start_index = self.open_prices.index[0]
        stop_index = self.open_prices.index[-1] - timedelta(days =
            self.rebalacing_freq_months)
        open_prices_for_rebalancing = []
        while start_index <= stop_index:
            end_rebalancing = start_index + timedelta(days =
                self.rebalacing_freq_months)
            open_df = self.open_prices.loc[start_index : end_rebalancing]
            open_prices_for_rebalancing.append(open_df)
            start_index = end_rebalancing

        return open_prices_for_rebalancing
    def get_volume(self):
        start_index = self.volume.index[0]
        stop_index = self.volume.index[-1] - timedelta(days =
            self.rebalacing_freq_months)
        volume_prices_for_rebalancing = []
        while start_index <= stop_index:
            end_rebalancing = start_index + timedelta(days =
                self.rebalacing_freq_months)
            volume_df = self.volume.loc[start_index : end_rebalancing]
            volume_prices_for_rebalancing.append(volume_df)
            start_index = end_rebalancing
```

```
        return volume_prices_for_rebalancing
```

10.4 Maximum Returns Portfolio

```
class Markowitz():
    def __init__(self, mu, sigma, w_old, psi, max_var):
        self.w = self.calculate_weights(mu, sigma, w_old, psi, max_var)

    def calculate_weights(self, mu, sigma, w_old, psi, max_var):
        dim = mu.shape[0]
        w_opt = cvx.Variable(dim)
        contr_1 = cvx.quad_form(w_opt, sigma)
        constr_2 = sum(w_opt)
        if len(w_old) == 0:
            obj = cvx.Maximize(w_opt @ mu)
            problem = cvx.Problem(obj, [contr_1 <= max_var, constr_2 == 1])
        else:
            obj = cvx.Maximize(w_opt @ mu - psi * cvx.norm(w_opt - w_old, 1)**2)
            problem = cvx.Problem(obj, [contr_1 <= max_var, constr_2 == 1])

        try:
            problem.solve(verbose = True)
        except:
            problem.solve(solver = 'SCS')
        w = w_opt.value

        return w
    def get_weights(self):
        return self.w
```

10.5 Minimum Variance Portfolio

```
class MVO():
```

```
    def __init__(self, mu, sigma, w_old, psi, min_returns):
        self.w = self.calculate_weights(mu, sigma, w_old, psi, min_returns)

    def calculate_weights(self, mu, sigma, w_old, psi, min_returns):
        dim = mu.shape[0]
        w_opt = cvx.Variable(dim)
        constr_1 = w_opt @ mu
        constr_2 = sum(w_opt)
        if len(w_old) == 0:
            obj = cvx.Minimize(cvx.quad_form(w_opt, sigma))
            problem = cvx.Problem(obj, [constr_1 >= min_returns, constr_2 == 1])
        else:
            obj = cvx.Minimize(cvx.quad_form(w_opt, sigma) + psi *
                cvx.norm(w_opt - w_old, 1)**2)
            problem = cvx.Problem(obj, [constr_1 >= min_returns, constr_2 == 1])


        try:
            problem.solve(verbose = False)
        except:
            problem.solve(solver = 'SCS')
        w = w_opt.value
        return w
    def get_weights(self):
        return self.w
```

10.6 Risk Portfolios

```
class RiskPortfolio():
    def __init__(self, sigma):
        dim = sigma.shape[0]
        self.w_equaly_weighted = self.equaly_weighted(dim)
        self.w_inv_variance = self.inv_variance(sigma, dim)
        self.w_min_variance = self.min_variance(sigma, dim)
```

```
def equaly_weighted(self, dim):
    identity = np.identity(dim)
    ones = np.ones((dim, 1))

    return (identity @ ones)/(ones.T @ identity @ ones)

def inv_variance(self, sigma, dim):
    lambda_ = np.diag(np.diag(sigma))
    lambda_2 = lambda_ ** 2
    ones = np.ones((dim, 1))

    return (np.linalg.inv(lambda_2) @ ones)/ (ones.T @
        np.linalg.inv(lambda_2) @ ones)

def min_variance(self, sigma, dim):
    ones = np.ones((dim, 1))

    return (np.linalg.inv(sigma) @ ones) / (ones.T @ np.linalg.inv(sigma) @
        ones)

def get_weights(self, type):
    if type == 'Min Variance':
        return self.w_min_variance
    elif type == 'Inv Variance':
        return self.w_inv_variance
    elif type == 'Equally Weighted':
        return self.w_equaly_weighted
    else:
        print('Not a Valid Type. You can try one of the following: Min
            Variance, Inv Variance, Equally Weighted')
        return None
```

10.7 Robust Portfolio

```
class RobustOptimiser():
    def __init__(self, mu, sigma, omega, w_old, psi, kappa, lambda_):
        self.w = self.calculate_weights(mu, sigma, omega, w_old, psi, kappa,
            lambda_)

    def calculate_weights(self, mu, sigma, omega, w_old, psi, kappa, lambda_):
        dim = mu.shape[0]
        w_rob = cvx.Variable(dim)
        Q = np.linalg.cholesky(omega)
        error_risk_term = cvx.norm(Q @ w_rob, 2)
        risk_aversion_term = cvx.quad_form(w_rob, sigma)
        obj = cvx.Maximize(w_rob @ mu - kappa * error_risk_term - lambda_/2 *
            risk_aversion_term)
        constr_1 = sum(w_rob)
        if len(w_old) == 0:
            obj = cvx.Maximize(w_rob @ mu - kappa * error_risk_term - lambda_/2
                * risk_aversion_term)
            problem = cvx.Problem(obj, [constr_1 == 1])
        else:
            obj = cvx.Maximize(w_rob @ mu - kappa * error_risk_term - lambda_/2
                * risk_aversion_term - psi * cvx.norm(w_rob - w_old, 1) ** 2)
            problem = cvx.Problem(obj, [constr_1 == 1])



        try:
            problem.solve(verbose = True)
        except:
            problem.solve(solver = 'SCS')
        if problem.value != np.inf:
            return w_rob.value
        else:
            print('The problem is unfeasible')
            return None
```

```
    def get_w_robust(self):
        return self.w
```

10.8 Black-Litterman Portfolio

```
class BlackLittermanPortfolio():
    def __init__(self, exp_returns, sigma, market_cap, prior, delta, tau,
        min_returns, omega):
        if len(prior) == 0:
            self.prior = self.calculate_prior(sigma, market_cap, delta, tau)
        else:
            self.prior = prior
        self.likelihood = self.calculate_likelihood(exp_returns, sigma, tau)
        self.posterior = self.master_formula(self.prior, self.likelihood)
        self.optimal_weights = self.optimize_problem(self.posterior, omega,
            min_returns)

    def calculate_prior(self, sigma, market_cap, delta, tau):
        market_cap = (np.array(market_cap)).reshape((-1,1))
        equilibrium_returns = delta * (sigma @ market_cap)
        return [equilibrium_returns, tau * sigma]

    def calculate_likelihood(self, expected_returns, sigma, tau):
        views_vector = (np.array(expected_returns)).reshape((-1,1))
        absolute_pick_matrix = np.identity(views_vector.shape[0])
        omega = np.diag(np.diag(tau * absolute_pick_matrix @ sigma @
            absolute_pick_matrix))
        return [views_vector, omega]

    def master_formula(self, prior, likelihood):
        mu_prior, sigma_prior = prior[0], prior[1]
        mu_likelihood, sigma_likelihood = likelihood[0], likelihood[1]
```

```
        mu_bl = np.linalg.inv(np.linalg.inv(sigma_prior) +
            np.linalg.inv(sigma_likelihood)) @\
            (np.linalg.inv(sigma_prior)@ mu_prior +
                np.linalg.inv(sigma_likelihood) @ mu_likelihood)
        sigma_bl = np.linalg.inv(np.linalg.inv(sigma_prior) +
            np.linalg.inv(sigma_likelihood))

        return [mu_bl, sigma_bl]

    def optimize_problem(self, posterior, omega, min_returns):
        mu, sigma = posterior[0], posterior[1]

        dim = mu.shape[0]
        w = cvx.Variable(dim)
        obj = cvx.Minimize(cvx.quad_form(w, sigma))
        mu_t = np.reshape(mu, (1, -1))
        constr_1 = mu_t @ w
        constr_2 = sum(w)
        problem = cvx.Problem(obj, [constr_1 >= min_returns, constr_2 == 1])
        problem.solve()

        return w.value

    def get_weights(self):
        return self.optimal_weights
    def get_posterior(self):
        return self.posterior
```

10.9 main

```
from mu import *
from sigma import *
```

```
from dataloader import *
from markowitz import *
from robust_optimiser import *
from mean_variance_portfolio import *
from risk_portfolio import *
from black_litterman import *
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

df_dataprojets = pd.read_excel('portfolio_optimization/DataProjets.xls',
    sheet_name=0)
df_marketcap = pd.read_excel('portfolio_optimization/DataProjets.xls',
    sheet_name=1, index_col=0)

correspondences = df_dataprojets[['Sedol', 'Tickers']]
correspondences.set_index('Sedol', drop = True, inplace = True)
correspondences_dict = correspondences.to_dict()['Tickers']

df_marketcap.index = pd.to_datetime(df_marketcap.index)
df_marketcap_month = df_marketcap.resample('1D')
df_marketcap_month = df_marketcap_month.mean()
df_marketcap_month.fillna(method = 'ffill', inplace = True)
df_marketcap_month.rename(columns = correspondences_dict, inplace = True)

tickers_list = list(df_dataprojets['Tickers'].values)
market_cap_tickers = set(df_marketcap_month.columns)
tickers_list_filtered = np.random.choice(tickers_list, size = 25)
tickers = list(market_cap_tickers.intersection(set(tickers_list_filtered)))
period = '1y'
rebalancing_freq = 2*21

data = Dataloader(period, tickers, rebalancing_freq)
dates, tickers_close_info = data.get_close()
```

```
tickers_marketcap = df_marketcap_month[tickers]

# Initialization:
close_returns = pd.DataFrame()
i = 0
prior = []
markowitz_weights = []
robust_weights = []
mvo_weights = []
psi = 0.3

for close_df in tickers_close_info:
    tickers_close_returns = (close_df/close_df.shift(1)).dropna() - 1
    exp_returns = mu(tickers_close_returns, tickers, rebalancing_freq).get_mu()
    sigma = Sigma(tickers_close_returns,rebalancing_freq).get_sigma()
    omega = np.diag(np.diag(sigma))

    if i > 0:

        markowitz_df =
            (tickers_close_returns.multiply(markowitz_weights)).sum(axis = 1)
        robust_df = (tickers_close_returns.multiply(robust_weights)).sum(axis =
            1)
        mvo_df = (tickers_close_returns.multiply(mvo_weights)).sum(axis = 1)
        risk_inv_df =
            (tickers_close_returns.multiply(risk_weights_inv)).sum(axis = 1)
        risk_min_df =
            (tickers_close_returns.multiply(risk_weights_min)).sum(axis = 1)
        risk_eq_df = (tickers_close_returns.multiply(risk_weights_eq)).sum(axis
            = 1)
        black_litterman_df =
            (tickers_close_returns.multiply(bl_weights)).sum(axis = 1)

        tickers_close_returns['Portfolio Markowitz'] = markowitz_df
```

```
        tickers_close_returns['Portfolio Robust'] = robust_df
        tickers_close_returns['Portfolio MVO'] = mvo_df
        tickers_close_returns['Portfolio Risk Equal Weights'] = risk_eq_df
        tickers_close_returns['Portfolio Risk Min Variance'] = risk_min_df
        tickers_close_returns['Portfolio Risk Inv Variance'] = risk_inv_df
        tickers_close_returns['Portfolio Black Litterman'] = black_litterman_df

        tickers_close_returns.dropna(axis = 0, inplace = True)
        close_returns = close_returns.append(tickers_close_returns)

    else:
        i += 1
        end_date = list(tickers_close_returns.index)[-1]
        marketcap = tickers_marketcap.loc[end_date].values

    markowitz = Markowitz(exp_returns, sigma, markowitz_weights, psi, 3)
    markowitz_weights = markowitz.get_weights()
    markowitz_weights = markowitz_weights/sum(abs(markowitz_weights))

    robust = RobustOptimiser(exp_returns, sigma, omega, robust_weights, psi, 5,
        8)
    robust_weights = robust.get_w_robust()
    robust_weights = robust_weights/sum(abs(robust_weights))

    mvo = MVO(exp_returns, sigma, mvo_weights, psi, 0.6)
    mvo_weights = mvo.get_weights()
    mvo_weights = mvo_weights/sum(abs(mvo_weights))

    risk_portfolio = RiskPortfolio(sigma)
    risk_weights_inv = np.transpose(risk_portfolio.get_weights('Inv Variance'))
    risk_weights_min = np.transpose(risk_portfolio.get_weights('Min Variance'))
    risk_weights_eq = np.transpose(risk_portfolio.get_weights('Equally
        Weighted'))
```

```
    black_litterman = BlackLittermanPortfolio(exp_returns, sigma, marketcap,
        prior, 0.1, 0.75, 0.2, omega)
    bl_weights = black_litterman.get_weights()
    bl_weights = bl_weights/sum(abs(bl_weights))
    prior = black_litterman.get_posterior()

    print('Expected Returns Vector: \n')
    print(exp_returns)
    print('\n ------------ \n')

    print('Covariance Matrix: \n')
    print(sigma)
    print('\n ------------ \n ')

    print('Optimal Weights Markowitz: \n')
    print(markowitz_weights)
    print('\n')
    print('Optimal Weights Robust: \n')
    print(robust_weights)
    print('\n ------------ \n ')
    print('Optimal Weights MVO: \n')
    print(mvo_weights)
    print('\n ------------ \n ')



# ----------------------------------------
# Sharpe Ratio Calculation
close_returns['Portfolio Markowitz'] = close_returns['Portfolio
    Markowitz'].astype(float)
close_returns['Portfolio Robust'] = close_returns['Portfolio
    Robust'].astype(float)
close_returns['Portfolio MVO'] = close_returns['Portfolio MVO'].astype(float)
close_returns['Portfolio Risk Equal Weights'] = close_returns['Portfolio Risk
    Equal Weights'].astype(float)
```

```
close_returns['Portfolio Risk Min Variance'] = close_returns['Portfolio Risk
    Min Variance'].astype(float)
close_returns['Portfolio Risk Inv Variance'] = close_returns['Portfolio Risk
    Inv Variance'].astype(float)

risk_free_return = 0.0
sharpe = {}
sharpe['Portfolio Markowitz'] = np.mean(close_returns['Portfolio Markowitz'] -
    risk_free_return)/np.std(close_returns['Portfolio Markowitz'])
sharpe['Portfolio Robust'] = np.mean(close_returns['Portfolio Robust'] -
    risk_free_return)/np.std(close_returns['Portfolio Robust'])
sharpe['Portfolio MVO'] = np.mean(close_returns['Portfolio MVO'] -
    risk_free_return)/np.std(close_returns['Portfolio MVO'])
sharpe['Portfolio Risk Equal Weights'] = np.mean(close_returns['Portfolio Risk
    Equal Weights'] - risk_free_return)/np.std(close_returns['Portfolio Risk
    Equal Weights'])
sharpe['Portfolio Risk Min Variance'] = np.mean(close_returns['Portfolio Risk
    Min Variance'] - risk_free_return)/np.std(close_returns['Portfolio Risk Min
    Variance'])
sharpe['Portfolio Risk Inv Variance'] = np.mean(close_returns['Portfolio Risk
    Inv Variance'] - risk_free_return)/np.std(close_returns['Portfolio Risk Inv
    Variance'])
sharpe['Portfolio Black Litterman'] = np.mean(close_returns['Portfolio Black
    Litterman'] - risk_free_return)/np.std(close_returns['Portfolio Black
    Litterman'])
df_sharpe = pd.DataFrame(sharpe, index = [0]).T
df_sharpe.columns = ['Sharpe Ratio']
print(df_sharpe.to_latex())

# -------------------------------------------------------
# Value at Risk Calculation
# 1 month horizon at 95%
VaR_1mo_95 = {}
```

```
VaR_1mo_95['Portfolio Markowitz'] = np.mean(close_returns['Portfolio
    Markowitz']) - 1.65 * np.sqrt(21) * np.std(close_returns['Portfolio
    Markowitz'])
VaR_1mo_95['Portfolio Robust'] = np.mean(close_returns['Portfolio Robust']) -
    1.65 * np.sqrt(21) * np.std(close_returns['Portfolio Robust'])
VaR_1mo_95['Portfolio MVO'] = np.mean(close_returns['Portfolio MVO']) - 1.65 *
    np.sqrt(21) * np.std(close_returns['Portfolio MVO'])
VaR_1mo_95['Portfolio Risk Equal Weights'] = np.mean(close_returns['Portfolio
    Risk Equal Weights']) - 1.65 * np.sqrt(21) *
    np.std(close_returns['Portfolio Risk Equal Weights'])
VaR_1mo_95['Portfolio Risk Min Variance'] = np.mean(close_returns['Portfolio
    Risk Min Variance']) - 1.65 * np.sqrt(21) * np.std(close_returns['Portfolio
    Risk Min Variance'])
VaR_1mo_95['Portfolio Risk Inv Variance'] = np.mean(close_returns['Portfolio
    Risk Inv Variance']) - 1.65 * np.sqrt(21) * np.std(close_returns['Portfolio
    Risk Inv Variance'])
VaR_1mo_95['Portfolio Black Litterman'] = np.mean(close_returns['Portfolio
    Black Litterman']) - 1.65 * np.sqrt(21) * np.std(close_returns['Portfolio
    Black Litterman'])

# 1 month horizon at 99%
VaR_1mo_99 = {}
VaR_1mo_99['Portfolio Markowitz'] = np.mean(close_returns['Portfolio
    Markowitz']) - 2.33 * np.sqrt(21) * np.std(close_returns['Portfolio
    Markowitz'])
VaR_1mo_99['Portfolio Robust'] = np.mean(close_returns['Portfolio Robust']) -
    2.33 * np.sqrt(21) * np.std(close_returns['Portfolio Robust'])
VaR_1mo_99['Portfolio MVO'] = np.mean(close_returns['Portfolio MVO']) - 2.33 *
    np.sqrt(21) * np.std(close_returns['Portfolio MVO'])
VaR_1mo_99['Portfolio Risk Equal Weights'] = np.mean(close_returns['Portfolio
    Risk Equal Weights']) - 2.33 * np.sqrt(21) *
    np.std(close_returns['Portfolio Risk Equal Weights'])
VaR_1mo_99['Portfolio Risk Min Variance'] = np.mean(close_returns['Portfolio
    Risk Min Variance']) - 2.33 * np.sqrt(21) * np.std(close_returns['Portfolio
```

```
    Risk Min Variance'])
VaR_1mo_99['Portfolio Risk Inv Variance'] = np.mean(close_returns['Portfolio
    Risk Inv Variance']) - 2.33 * np.sqrt(21) * np.std(close_returns['Portfolio
    Risk Inv Variance'])
VaR_1mo_99['Portfolio Black Litterman'] = np.mean(close_returns['Portfolio
    Black Litterman']) - 2.33 * np.sqrt(21) * np.std(close_returns['Portfolio
    Black Litterman'])

df_var = pd.DataFrame([VaR_1mo_95, VaR_1mo_99], index = ['VaR 95 at 1mo
    horizon', 'VaR 99 at 1mo horizon']).T
print(df_var.to_latex())

# ----------------------------------------
# Maximum Drawdown
#
    https://quant.stackexchange.com/questions/18094/how-can-i-calculate-the-maximum-drawdown-m
window = 21
returns = close_returns[['Portfolio Markowitz', 'Portfolio Robust', 'Portfolio
    MVO', 'Portfolio Risk Equal Weights', 'Portfolio Risk Min Variance',
    'Portfolio Risk Inv Variance', 'Portfolio Black Litterman']]
Roll_Max = returns.rolling(window, min_periods=1).max()
Daily_Drawdown = returns/Roll_Max - 1.0
Max_Daily_Drawdown = Daily_Drawdown.min(axis = 0)
Max_Daily_Drawdown.columns = ['Maximum 1 week Drawdown']
print(Max_Daily_Drawdown.to_latex())


# ----------------------------------------
# Plotting Results
close_returns = (close_returns + 1).cumprod(axis = 0)
close_returns[['Portfolio Markowitz', 'Portfolio Robust', 'Portfolio MVO',
    'Portfolio Risk Equal Weights', 'Portfolio Risk Min Variance', 'Portfolio
    Risk Inv Variance', 'Portfolio Black Litterman']].plot()
for date in dates:
```

```
    plt.axvline(date)
plt.ylabel('Cummulative Return')
plt.suptitle('Return of Portfolio vs. Individual Stocks')
plt.show()
```

Bibliography

[1] Woo Chang Kima, Jang Ho Kimb, and Frank J. Fabozzic
Deciphering robust portfolios, https://www.sabanciuniv.edu

[2] Yahoo Finance
S&P 500 historical prices, https://finance.yahoo.com/quote/%5EGSPC/history/

[3] Yahoo Finance
Dow Jones Industrial Average historical prices, https://finance.yahoo.com/quote/%5EDJI/histo

[4] Emmanuel Jurczenkoa, Thierry Michelb and Jérôme Teiletche
Generalized Risk-Based Investing, http://www.qminitiative.org/UserFiles/files/JurczenkoM

[5] Hudson and Thames, Aditya Vyas
Bayesian Portfolio Optimisation: Introducing the Black-Litterman Model,
https://hudsonthames.org/bayesian-portfolio-optimisation-the-black-litterman-model/

[6] Myles E. Mangram, 2013
A SIMPLIFIED PERSPECTIVE OF THE MARKOWITZ PORTFOLIO THEORY,
GLOBAL JOURNAL OF BUSINESS RESEARCH, VOLUME 7, NUMBER 1