Evaluation of Portfolio Optimization Allocation Weight Adjustment

1 Abstract

Whereas the portfolio optimization weight assignment methodology within the real industry is relatively complicated with support of machine learning algorithms, this research aims to evaluate the fitness of two prevalent portfolio optimization theories in academia, mean-variance optimization and risk-parity optimization, with comparison to the market data. In this study, 7 different ETFs with different levels of stocks included are selected: DIA, ALTY, UYM, AIA, AIQ, BATT, AIEQ. The live market data across those ETFs were downloaded from yahoo finance on which the realized algorithms based on the logic of mean-variance optimization and risk-parity optimization theories were implemented. With this method, the research managed to compare the performance of mean-variance and risk-parity algorithms and real market ones through different time scales with the use of return rate, volatility, and sharpe ratio. And the market performance data was all downloaded from ETFGlobal as a reference. The study finally observed a general pattern that whereas both mean-variance and risk-parity algorithms perform much better than the current market ones, mean-variance algorithms can be ranked as the one with strongest performances among the three, which indicates the fitness and implications of these fundamental optimization theories in the real market. However, as there's a lack of information about the weights utilized by the industry as well as the change of portfolio stocks, further investigations may be required to eliminate those effects.

2 Introduction

Regarding the construction of an optimal portfolio, the selection of component stocks as well as the weight allocation has been two important aspects to study in order to optimize the overall rate of return for the risk. This study will focus on the question on whether the prevalent theoretical optimization theories such as mean-variance portfolio and risk-parity portfolio theory perform better than the market ones employed by different firms. More specifically, it works on application of theories in the optimization improvement on the weight assignment process using the stocks included in market-constructed ETFs. To achieve this, 7 ETFs across industries were selected with different numbers of component stocks, including DIA(30 stocks), ALTY(18 stocks), UYM(38 stocks), AIA(56 stocks), AIQ(81 stocks), BATT(85 stocks), AIEQ(114 stocks). Due to the data access problems, the study assumes that the component stocks within each stock will not change within short time and hence the most recent portfolio component will be utilized directly.

Based on the market data of those stocks within each ETF, the study experiments the efficiency of mean-variance portfolio and risk-parity portfolio compared to the current market algorithm efficiency. In order to evaluate the optimal time frame for the algorithms to work and investigate time impacts, the tests were conducted on monthly, quarterly, yearly time frame as a comparison base. With the aim to dive deeper in the main stock contributor for the performance difference, each stock within ETFs was also looped over to test the impact of its deletion or swapping will influence the comparison result.

3 Methodology

As discussed in the introduction part, the study mainly focuses on the implementation of two prevalent optimization theories, mean-variance and risk-parity, on the stock data from 7 different ETFs. These ETFs are chosen based on the number of stocks included as well as the region of stocks included, with the aim to cover a sample representative of the market diversity. In addition, the data selection also facilitates the investigation on whether the final result can be generalized to ETFs with various sizes and regions. And to further investigate time impacts, the testing process was repeated through a 1-month, 2-month, 1-quarter, 2 quarter, 3-quarter, 1-year, 2-year, 3-year time frame as a comparison training data base whereas each stock within ETFs was also explored separately to determine the main stock contributor to the performance difference. In the following part, the two major optimization theories as well as their implementation in Python will be introduced whereas the main contributor selection methodology will also be discussed in greater detail.

3.1 Mean-Variance Theory & Implementation

Mean-Variance Portfolio Theory was introduced by Harry Markowitz in 1952 and the main idea under hidden this theory is that investors seek to minimize variance for a given level of expected return or equivalent, maximize expected return for a given constraint on variance.

Defining x_k as the fraction of the wealth that one puts in asset k at time $t = 0$ and r as the return vector of n risky assets, each of which has expectation $\mu = (\mu_k)_{1\leq k\leq n}$, $\Sigma = (\sigma_{ij})_{1\leq i,j\leq n}$ as covariance. Then, in the version to minimize variance corresponding to expected return, the theory for portfolio without risk-free assets can be formulated as:

$$\min_x \quad \tfrac{1}{2}x^T\Sigma x \ \text{ s.t. } x^T\mu = p, x^T I = 1 \tag{1}$$

On the other hand, for portfolio with a risk-free asset, changing the notation of $x = (x_1, x_2, ..., x_n)$ as weight of portfolio weights on the n risky assets whereas $1 - \sum_{k=1}^{n} x_k$ as weight on the risk-free asset, then the theory can be formulated as:

$$\min_x \quad \tfrac{1}{2}x^T\Sigma x \ \text{ s.t. } (1 - \textstyle\sum_{k=1}^{n} x_k)r + x^T\mu = p \tag{2}$$

With the help of Lagrange multiplier method, the final solution of optimal weight can be computed with the formula below corresponding to above 2 statements and form 2 efficient frontier, on which we can find the final optimal portfolio construction using tangency points.

for the portfolio without risk-free assets:

$$x^* = \Sigma^{-1}(\lambda_1 I + \lambda_2 \mu)$$
$$\lambda_1 = \frac{\gamma - \beta p}{\alpha\gamma - \beta^2}$$

and

$$\lambda_2 = \frac{\alpha p - \beta}{\alpha\gamma - \beta^2}$$

where

$$\alpha = I^T \Sigma^{-1} I, \beta = I^T \Omega^I \mu, \gamma = \mu^T \Sigma^{-1} \mu$$

for the portfolio with risk-free assets:

$$x^* = \frac{p - r}{(\mu - rI)^T \Sigma^{-1} (\mu - rI)} \Sigma^{-1}(\mu - rI)$$

In terms of the algorithm implementation, the study employs the functions such as CovarianceShrinkage, EfficientFrontier in pypfopt packages directly to compute the optimal weights for the portfolio. However, considering possible further requirement to edit part of the algorithm for better improvement, another customized algorithm was also designed by using scipy.optimize package to solve the variance minimization problem directly. The efficient frontier is then plotted based on the computed result whereas an optimal portfolio was detected from the frontier with maximum Sharpe ratio (part of the code at the end of report) Using the algorithm, the expected return rate, volatility, as well as Sharpe ratio were also obtained to be compared with market parameters for performance evaluation.

3.2 Risk-Parity Theory & Implementation

Risk-Parity Portfolio was first invented by All Weather Fund in 1996 and the main idea behind is to achieve an equal balance between the risk associated with each asset class or portfolio component. In other words, the aim of this strategy was to equate the risk contribution of each portfolio component so that lower risk assets will always be assigned with higher allocations and vice versa.

Still defining Σ as the covariance matrix of n assets whereas x is the weight matrix for n assets. In addition to these traditional notations, volatility of the portfolio can be expressed as $\sigma(x) = \sqrt{x^T \Sigma x}$. Through this token, the portfolio strategy can then be formulated into another non-linear optimization problem:

$$\min_x \quad \textstyle\sum_{k=1}^n (x_k - \frac{\sigma(x)^2}{n(\Sigma x)_k})^2 \text{ s.t. } \sum_{k=1}^n x_k = 1, x_k > 0 \tag{3}$$

Similarly as the method in the implementation for the mean-variance strategy, the scipy.minimize package is again used to solve the non-linear optimization problem to determine the optimal portfolio weight whereas other portfolio parameters are also computed for comparison with market ones(code attached at the end of this report).

3.3 Stock Contributor Determination Methodology

After comparing the results computed by above two allocation algorithms on real stock data through different time frames, it's essential to explore which of the stocks are main contributor to the performance differences while also validating if the portfolio performances matches the individual stock trends. In order to achieve the aim, each of the stock within the ETF is looped over to be deleted or replaced with the US 10-year Treasury before the two allocation algorithms to be operated so that the performance difference will indicate evidence about the importance of that stock (part of the code attached at the end of report). On the other hand, the live price data for each of the stock was also obtained as a foundation of an analysis on daily volatility rate and overall change rate to be compared with the portfolio parameters and determine whether those values are reasonable or not.

4 Results

Through the three steps taken, the research was able to compare the performance parameter of computed portfolios and market ones, study the impact of time lengths of training data, as well as the main contributor for each portfolio, which are shown separately below.

4.1 Algorithm Performance Comparison Result

Through implementation of treatments for all of the monthly-, quarterly-, and yearly-prediction, three tables 1,2,3 were generated below, presenting the comparison result on return rate, volatility rate, as well as Sharpe ratio data among three portfolios constructed in different ways. However, because of the data access issues, the Sharpe ratio and volatility rate for market portfolios were null here and hence the main comparison occurred between return rate.

ETF Name	Market Return	Mean-Variance Return	Mean-Variance Volatility	Mean-Variance Sharpe Ratio	Risk-Parity Return	Risk-Parity Volatility	Risk-Parity Sharpe Ratio
DIA	−9.18%	10.43%	0.07	1.40	−8.66%	0.07	−1.28
ALTY	−6.86%	1.27%	0.10	0.13	−4.09%	0.05	−0.75
UYM	−26.06%	98.75%	0.16	6.10	−7.11%	0.09	−0.76
AIA	−5.10%	16.90%	0.06	2.71	−0.72%	0.05	−0.15
AIQ	−2.68%	33.46%	0.08	3.98	−6.93%	0.07	−0.95
BATT	0.39%	32.96%	0.12	2.64	−2.53%	0.08	−0.31
AIEQ	−6.38%	27.00%	0.13	2.06	−9.09%	0.10	0.96

Table 1: Monthly Predicted Parameter Comparison Result for ETFs

ETF Name	Market Return	Mean-Variance Return	Mean-Variance Volatility	Mean-Variance Sharpe Ratio	Risk-Parity Return	Risk-Parity Volatility	Risk-Parity Sharpe Ratio
DIA	−11.56%	53.80%	0.08	6.64	0.86%	0.07	0.13
ALTY	−12.38%	10.25%	0.07	1.50	0.36%	0.06	0.07
UYM	−83.03%	128.08%	0.18	7.20	6.97%	0.12	0.59
AIA	−9.25%	14.90%	0.09	1.69	−0.63%	0.08	−0.08
AIQ	−18.14%	32.63%	0.08	3.97	−7.04%	0.11	−0.63
BATT	−10.65%	18.60%	0.30	0.61	−1.67%	0.12	−0.13
AIEQ	−12.92%	107.88%	0.13	8.33	0.79%	0.11	0.07

Table 2: Quarterly Predicted Parameter Comparison Result for ETFs

ETF Name	Market Return	Mean-Variance Return	Mean-Variance Volatility	Mean-Variance Sharpe Ratio	Risk-Parity Return	Risk-Parity Volatility	Risk-Parity Sharpe Ratio
DIA	−10.30%	57.10%	0.18	3.09	26.45%	0.14	1.89
ALTY	−16.46%	36.80%	0.12	2.88	20.71%	0.12	1.74
UYM	−79.86%	246.80%	0.44	5.67	−6.12%	0.17	0.36
AIA	−31.67%	143.80%	0.34	4.23	29.37%	0.29	1.01
AIQ	−27.51%	506.20%	0.41	12.29	5.60%	0.14	0.41
BATT	−11.98%	99.20%	0.53	1.87	32.07%	0.28	1.15
AIEQ	−22.74%	691.50%	0.37	18.79	11.02%	0.13	0.85

Table 3: Yearly Predicted Parameter Comparison Result for ETFs

4.2 Time Frame Comparison Result

In order to compare the time frame impact, the above parameter comparison test also happens in the case when training data of different sizes is employed: 1-month for monthly prediction, 2-month for monthly prediction, 3-month for monthly prediction,(4-month, 5-month, 6-month also included) 1-quarter for quarterly prediction, 2-quarter for quarterly prediction, 3-quarter for quarterly prediction, (4-quarter, 5-quarter, 6-quarter also included), 1-year for yearly prediction, 2-year for yearly prediction, 3-year for yearly prediction. And for simplicity, the result computed for the shortest 3 periods (1-,2-,3- units of time) are presented in the table 4,5,6 below, with each table has returns defined as monthly-return, quarterly return, and annual return respectively.

ETF Name	Time	Market Return	Mean-Variance Return	Mean-Variance Volatility	Mean-Variance Sharpe Ratio	Risk-Parity Return	Risk-Parity Volatility	Risk-Parity Sharpe Ratio
DIA	1-month	−9.18%	10.43%	0.07	1.40	−8.66%	0.07	−1.28
	2-month	−9.18%	10.34%	0.05	2.02	−2.96%	0.05	0.54
	3-month	−9.18%	10.22%	0.05	2.06	−0.82%	0.05	−0.15
ALTY	1-month	−6.86%	1.27%	0.10	0.13	−4.09%	0.05	−0.75
	2-month	−6.86%	6.08%	0.05	1.25	1.13%	0.05	0.25
	3-month	−6.86%	6.85%	0.05	1.49	2.08%	0.04	0.48
UYM	1-month	−26.06%	98.75%	0.16	6.10	−7.11%	0.09	−0.76
	2-month	−26.06%	45.90%	0.12	3.76	1.38%	0.07	0.19
	3-month	−26.06%	25.62%	0.11	2.43	1.88%	0.08	0.25
AIA	1-month	−5.10%	16.90%	0.06	2.71	−0.72%	0.05	−0.15
	2-month	−5.10%	13.13%	0.06	2.06	0.83%	0.04	0.20
	3-month	−5.10%	3.46%	0.09	0.40	−0.67%	0.05	−0.13
AIQ	1-month	−2.68%	33.46%	0.08	3.98	−6.93%	0.07	−0.95
	2-month	−2.68%	10.32%	0.09	1.18	−6.05%	0.07	−0.90
	3-month	−2.68%	16.74%	0.09	1.86	−2.44%	0.07	−0.33
BATT	1-month	0.39%	32.96%	0.12	2.64	−2.53%	0.08	−0.31
	2-month	0.39%	11.68%	0.10	1.16	0.89%	0.10	0.09
	3-month	0.39%	12.87%	0.10	1.35	1.81%	0.10	0.18
AIEQ	1-month	−6.38%	27.00%	0.13	2.06	−9.09%	0.10	0.96
	2-month	−6.38%	28.51%	0.10	2.97	−3.93%	0.08	−0.49
	3-month	−6.38%	44.47%	0.09	4.75	−0.48%	0.08	−0.06

Table 4: Monthly Predicted Parameter Comparison Result for ETFs based on Different Training Sets

ETF Name	Time	Market Return	Mean-Variance Return	Mean-Variance Volatility	Mean-Variance Sharpe Ratio	Risk-Parity Return	Risk-Parity Volatility	Risk-Parity Sharpe Ratio
DIA	1-quarter	−11.56%	53.80%	0.08	6.64	0.86%	0.07	0.13
	2-quarter	−11.56%	29.90%	0.08	3.83	2.23%	0.07	0.33
	3-quarter	−11.56%	15.15%	0.07	2.05	1.45%	0.06	0.24
ALTY	1-quarter	−12.38%	10.25%	0.07	1.50	0.36%	0.06	0.07
	2-quarter	−12.38%	12.63%	0.06	2.07	4.18%	0.05	0.83
	3-quarter	−12.38%	8.33%	0.07	1.22	2.01%	0.05	0.42
UYM	1-quarter	−83.03%	128.08%	0.18	7.20	6.97%	0.12	0.59
	2-quarter	−83.03%	39.08%	0.11	3.49	9.61%	0.10	0.97
	3-quarter	−83.03%	36.38%	0.15	2.37	10.17%	0.10	1.04
AIA	1-quarter	−9.25%	14.90%	0.09	1.69	−0.63%	0.08	−0.08
	2-quarter	−9.25%	12.00%	0.08	1.42	2.11%	0.07	0.31
	3-quarter	−9.25%	8.63%	0.08	1.09	−0.12%	0.07	−0.02
AIQ	1-quarter	−18.14%	32.63%	0.08	3.97	−7.04%	0.11	−0.63
	2-quarter	−18.14%	13.43%	0.09	1.42	0.65%	0.10	0.07
	3-quarter	−18.14%	14.98%	0.14	1.09	0.54%	0.09	0.06
BATT	1-quarter	−10.65%	18.60%	0.30	0.61	−1.67%	0.12	−0.13
	2-quarter	−10.65%	19.70%	0.23	0.87	5.38%	0.12	0.43
	3-quarter	−10.65%	25.33%	0.20	1.27	8.75%	0.11	0.79
AIEQ	1-quarter	−12.92%	107.88%	0.13	8.33	0.79%	0.11	0.07
	2-quarter	−12.92%	23.10%	0.09	2.64	1.06%	0.10	0.11
	3-quarter	−12.92%	25.53%	0.10	2.53	−0.47%	0.10	−0.05

Table 5: Quarterly Predicted Parameter Comparison Result for ETFs based on Different Training Sets

ETF Name	Time	Market Return	Mean-Variance Return	Mean-Variance Volatility	Mean-Variance Sharpe Ratio	Risk-Parity Return	Risk-Parity Volatility	Risk-Parity Sharpe Ratio
DIA	1-year	−10.30%	57.10%	0.18	3.09	26.45%	0.14	1.89
	2-year	−10.30%	61.50%	0.36	1.65	18.86%	0.25	0.75
	3-year	−10.30%	34.40%	0.25	1.28	18.73%	0.24	0.78
ALTY	1-year	−16.46%	36.80%	0.12	2.88	20.71%	0.12	1.74
	2-year	−16.46%	13.70%	0.26	0.46	9.12%	0.25	0.36
	3-year	−16.46%	20.90%	0.29	0.66	11.25%	0.22	0.52
UYM	1-year	−79.86%	246.80%	0.44	5.67	−6.12%	0.17	0.36
	2-year	−79.86%	150.40%	0.41	3.65	−6.12%	0.17	−0.36
	3-year	−79.86%	160.70%	0.37	4.33	−6.12%	0.17	0.36
AIA	1-year	−31.67%	143.80%	0.34	4.23	29.37%	0.29	1.01
	2-year	−31.67%	70.80%	1.84	0.39	34.26%	0.29	1.17
	3-year	−31.67%	46.20%	1.53	0.30	39.00%	0.29	1.36
AIQ	1-year	−27.51%	506.20%	0.41	12.29	5.60%	0.14	0.41
	2-year	−27.51%	490.90%	0.48	10.18	8.56%	0.14	0.63
	3-year	-27.51%	494.70%	0.50	9.85	10.06%	0.13	0.63
BATT	1-year	−11.98%	99.20%	0.53	1.87	32.07%	0.28	1.15
	2-year	−11.98%	189.80%	0.51	3.76	97.86%	0.33	2.95
	3-year	−11.98%	128.60%	0.45	2.86	86.18%	0.29	2.95
AIEQ	1-year	−22.74%	691.50%	0.37	18.79	11.02%	0.13	0.85
	2-year	−22.74%	167.10%	0.34	4.93	9.37%	0.12	0.77
	3-year	−22.74%	89.00%	0.20	4.43	7.42%	0.12	0.62

Table 6: Yearly Predicted Parameter Comparison Result for ETFs based on Different Training Sets

4.3 Main Contributor Investigation Result

Through iterating through each stock within the ETF, the study was able to filter out the most important contributors corresponding to different periods. Whereas the contributor for monthly prediction and quarterly distribution is relatively even to some extent, the table below represents the major contributors corresponding to each ETF through different time frames as well as their corresponding overall price change rate.

ETF	Monthly-prediction	Quarterly-prediction	Yearly-prediction
DIA	BA −31.64%	CVX −31.64%	AAPL 78.24%
ALTY	QYLD −13.27%	PPL −15.56%	TNX −51.28%
UYM	X −35.70%	DNA −50.63%	MP 221.70%
AIA	3690.HK 18.49%	2269.HK −26.32%	1211.HK 211.39%
AIQ	NFLX −47.37%	AMBA −55.60%	TSLA 720.05%
BATT	LGO −27.16%	SLI −43.12%	1585.HK 497.01%

Table 7: Major Contributor Evaluation

5 Discussion

As the result tables above present, a general trend across all ETFs that mean-variance portfolio works best among the three portfolio constructions, whereas risk-parity portfolio also works slightly better than the market ones. However, it was also discovered that mean-variance strategy also generates greater volatility than risk-parity ones whereas its Sharpe ratio is still higher. This is probably because of the nature of risk-parity strategy to ensure each stock's risk contribution the same and hence is more stable whereas mean-variance strategy is more likely to assign extreme weights.

In addition, the longer time period for the two theoretical strategies to be operated, the better performance they'll generate compared to the market ones. In addition to that, in terms of impact of training data time, it can be widely observed that a shorter time length of training works better as the extension of it usually generate large deviations compared to previous version. On the other hand, the evaluation of main contributor also matches with the computed result, indicating that the algorithm correctly assign suitable weights on strong-performing stocks and generate much larger return as well as Sharpe ratio compared to market existing ones. Hence, overall the study has detected a pattern that mean-variance portfolio is the best one to work on ETFs in this case, although with a relative high volatility, while risk-parity ranked the second. Hence, it is recommended that mean-variance portfolio may be suitable for risk-seeking investors whereas risk-parity portfolio strategy may match more of a risk-averse investor.

However, as the study assume that the portfolio components don't experience large switching or changing during the training time period, which is a bit unrealistic to some extent. Some extra information and data sources may be required as a next step to improve the current study and further investigate if the conclusion here still hold. But overall, this study presents that the fundamental theoretical optimization strategies can still fit the current real market, based on which adjustments can be made to facilitate and improve the portfolio construction efficiency.

6 Appendix

```
from pypfopt.expected_returns import mean_historical_return
from pypfopt.risk_models import CovarianceShrinkage
# calculate the expected return using historical data
mu = mean_historical_return(portfolio, returns_data=False)
# calculate the covariance matrix
S = CovarianceShrinkage(portfolio).ledoit_wolf()
from pypfopt.efficient_frontier import EfficientFrontier
# generate the efficient frontier
ef = EfficientFrontier(mu,S)
from pypfopt import plotting
# plot the efficient frontier
fig, ax = plt.subplots()
plotting.plot_efficient_frontier(ef,ax=ax,show_assets=True)
#plt.show()
from pypfopt import plotting
# plot the efficient frontier
fig, ax = plt.subplots()
plotting.plot_efficient_frontier(ef,ax=ax,show_assets=True)
#plt.show()
#import matplotlib.pyplot as plt
# plot the bar chart of the weights
plt.bar(dict(cleaned_weights).keys(),dict(cleaned_weights).values() , align='
    center')
plt.show()
# performance of the portfolio
```

```python
ef.portfolio_performance(verbose=True)
```

Listing 1: The python Source code for Mean-Variance Portfolio using packages

```python
import numpy as np
import pandas as pd
from pandas_datareader import data as web
import scipy.optimize as sco
import scipy.interpolate as sci
def portfolio_returnsn(weights):
    return -(np.sum(df.mean() * weights)) * 253
# Function for computing portfolio return
def portfolio_returns(weights):
    return (np.sum(df.mean() * weights)) * 253
# Function for computing standard deviation of portfolio returns
def portfolio_sd(weights):
    return np.sqrt(np.transpose(weights) @ (df.cov() * 253) @ weights)
#from __future__ import (absolute_import, division,print_function,
    unicode_literals)
#from builtins import *
def mini_vol(target_ret):
    """number is stock  n u m b e r s ret_ave_vector  is the return column vector;
       ret_cov_matrix is the return covariance matrix
       target_ret is target return
    """
    # reverse the covariance matrix

    cons = [{'type':'eq', 'fun': lambda x: portfolio_returns(x)-target_ret},
            {'type':'eq', 'fun': lambda x: np.sum(x) - 1},
           {'type': 'ineq', 'fun': lambda x: x}]
    bounds = tuple((0, 1) for w in weights)
    equal_weights = np.array([1 / len(df.columns)] * len(df.columns))
    # Minimization results
    weights2 = sco.minimize(
        # Objective function
        fun = portfolio_sd,
        # Initial guess, which is the equal weight array
        x0 = equal_weights,
        method = 'SLSQP',
        bounds = bounds,
        constraints = cons)
    #print(weights2)
    return weights2
def sharpe_fun(weights,rf):
    return ((portfolio_returns(weights)-rf) / portfolio_sd(weights))
port_ret = []
```

```python
port_std = []
port_sr=[]

# plot the efficient frontier and create a dataframe for this
ranges = np.arange(0,1, 0.00001)
for i in ranges:
    weight = mini_vol(i)
    port_ret1= portfolio_returns(weight["x"])
    port_std1=portfolio_sd(weight["x"])
    #define rf
    rf=0.2
    port_sr1=sharpe_fun(weight["x"],rf)
    port_ret.append(port_ret1)
    port_std.append(port_std1)
    port_sr.append(port_sr1)

port_std = np.array(port_std)
port_ret = np.array(port_ret)
port_sr== np.array(port_sr)
data = {'Returns':port_ret, 'Volatility':port_std, "Sharpe Ratio": port_sr}
portfolios=pd.DataFrame(data)
# Finding the optimal portfolio
# set risk free rate as 0.02
rf = 0.02 # risk factor
def evaluation(rf):
    return portfolios.iloc[((portfolios['Returns']-rf)/portfolios['Volatility']).
   idxmax()]

evaluation(rf)
#optimal_risky_port = portfolios.iloc[((portfolios['Returns']-rf)/portfolios['
   Volatility']).idxmax()]
#optimal_risky_port
weight=mini_vol((((portfolios['Returns']-rf)/portfolios['Volatility']).idxmax()
   -1)*0.00001)
weight["x"]
```

Listing 2: The python Source code for Mean-Variance Portfolio without using packages

```python
from __future__ import division
import numpy as np
from matplotlib import pyplot as plt
from numpy.linalg import inv,pinv
from scipy.optimize import minimize
V = np.matrix(df.cov()) # covariance of assets
R = np.matrix(mean_historical_return(portfolio,returns_data=False)) # expected
   return
```

```python
import pandas as pd
import pandas_datareader.data as web
import numpy as np
import datetime
from scipy.optimize import minimize
TOLERANCE = 1e-10
def _allocation_risk(weights, covariances):
    # We calculate the portfolio risk (sigma^2)
    portfolio_risk = np.sqrt((weights * covariances * weights.T))[0, 0]
    # It returns the risk of the weights distribution
    return portfolio_risk
def _assets_risk_contribution_to_allocation_risk(weights, covariances):
    # We calculate the risk of the portfolio
    portfolio_risk = _allocation_risk(weights, covariances)
    # We calculate the contribution of each asset to the risk of total portfolio risk
    assets_risk_contribution = np.multiply(weights.T, covariances * weights.T)/portfolio_risk
    # It returns the contribution of each asset to the risk of the weights
    # distribution
    return assets_risk_contribution
def _risk_budget_objective_error(weights, args):
    # The covariance matrix occupies the first position in the variable
    covariances = args[0]
    # The desired contribution of each asset to the portfolio risk occupies the
    # second position
    assets_risk_budget = args[1]
    # We convert the weights to a matrix
    weights = np.matrix(weights)
    # We calculate the portfolio risk (total portfolio variance)
    portfolio_risk = _allocation_risk(weights, covariances)
    # We calculate the contribution of each asset to the risk of the portfolio
    assets_risk_contribution = _assets_risk_contribution_to_allocation_risk(weights, covariances)
    # We calculate the desired contribution of each asset to the risk of the portfolio
    assets_risk_target = np.asmatrix(np.multiply(portfolio_risk, assets_risk_budget))
    # Error between the desired contribution and the calculated/actual contribution of
    # each asset
    error = sum(np.square(assets_risk_contribution - assets_risk_target.T))[0, 0]
    # It returns the calculated error
    return error
def _get_risk_parity_weights(covariances, assets_risk_budget, initial_weights):
    # Restrictions to consider in the optimisation: only long positions whose
    # sum equals 100%
    constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1.0},
                   {'type': 'ineq', 'fun': lambda x: x})
    # Optimisation process in scipy
    optimize_result = minimize(fun=_risk_budget_objective_error,
                               x0=initial_weights,
                               args=[covariances, assets_risk_budget],
                               method='SLSQP',
                               constraints=constraints,
                               tol=TOLERANCE,
                               options={'disp': False})

    # Recover the weights from the optimised object
    weights = optimize_result.x
    # It returns the optimised weights
    return weights
def get_weights(yahoo_tickers=["UNH", "GS", "HD","MSFT","MCD","AMGN","CAT","V","HON","JNJ","CRM","TRV","CVX","AXP","IBM","AAPL","MMM","BA","PG","WMT","JPM","NKE","DIS","MRK","KO","DOW","VZ","CSCO","WBA","INTC"],
                start_date=datetime.datetime(2019,6,19),
                end_date=datetime.datetime(2022,6,19)):
    # We download the prices from Yahoo Finance
    prices = pd.DataFrame([web.DataReader(t,
                                          'yahoo',
                                          start_date,
                                          end_date).loc[:, 'Adj Close']
                           for t in yahoo_tickers],
                          index=yahoo_tickers).T.asfreq('B').ffill()
    # We calculate the covariance matrix
    #covariances = 252.0 * prices.asfreq('W-FRI').pct_change().iloc[1:, :].cov().values
    covariances=V
    # The desired contribution of each asset to the portfolio risk: we want all
    # asset to contribute equally
    assets_risk_budget = [1 / prices.shape[1]] * prices.shape[1]
    # Initial weights: equally weighted
    #init_weights = [1 / prices.shape[1]] * prices.shape[1]
    init_weights =
   [0.71,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0

    # Optimisation process of weights
    weights =  _get_risk_parity_weights(covariances, assets_risk_budget, init_weights)
    # Convert the weights to a pandas Series
```

```python
    weights = pd.Series(weights, index=prices.columns, name='weight')
    # It returns the optimised weights
    return weights
get_weights()
```

Listing 3: The python Source code for Risk-parity Portfolio

```python
#results_volatility=pd.DataFrame(columns=stocks)
stocks = ["UNH", "GS", "HD","MSFT","MCD","AMGN","CAT","V","HON","JNJ","CRM","TRV"
    ,"CVX","AXP","IBM","AAPL","MMM","BA","PG","WMT","JPM","NKE","DIS","MRK","KO",
    "DOW","VZ","CSCO","WBA","INTC"]
#stock=stocks[1:]
stock=stocks[0:20]+stocks[21:]
portfolio=combine_stocks(stock)
df=pd.DataFrame(columns=portfolio.columns)
for j in df:
    df[j]=portfolio[j].pct_change()
df=df.dropna()
cleaned_weights=get_weights(df,portfolio)
returns=portfolio_returns(df,cleaned_weights,20)
sd=portfolio_sd(df,cleaned_weights,20)
sharpe=sharpe_fun(df,cleaned_weights,20,0.0298)
result=[returns,sd,sharpe]
results_return3[results_return3.columns[20]]=np.array(result)
```

Listing 4: The python Source code for Main Contributor Discovery