

Portfolio Selection

Tian Qin, Ruochen Wang, Yi Zhang, Yuying Zhuo

2025/06/08

1 Introduction

To face the challenge in nowadays dynamic financial environment, it is a critical part to construct a diversified and optimized investment portfolio for risk management and desired returns. The goal of this project is to explore the development of a portfolio composed of 10 publicly traded companies over a six-year historical period. The primary objective is to employ statistical and computational methods to analyze the trade-off between risk and return, and to determine optimal investment weights that balance these two factors.

To access it, we gathered historical share prices and annual dividend data for each selected investment. The data was used to calculate annualized returns and standard deviations, which serve as proxies for expected return and risk, respectively. Additionally, Python scripts were customized to simulate multiple portfolio combinations using Monte Carlo methods to build a correlation matrix and construct a risk-return map.

Through this analysis, we aim to identify a portfolio within a user-defined range of acceptable risks and report its average risk, average return, and individual investment weights. This report summarizes the data acquisition process, analytical methodologies, simulation results, and offers key insights into portfolio optimization.

2 Methodology

Portfolio Construction Framework

The methodology adopted in this study is based on the principles of Modern Portfolio Theory (MPT), which models the trade-off between risk and return using historical asset performance data. A diversified portfolio is constructed using ten selected investment assets over a defined six-year period. The portfolio design involves the calculation of historical returns, estimation of risk through variance and covariance, and the optimization of asset weights to maximize returns under acceptable risk constraints.

Return and Risk Estimation

For each investment, the mean return was computed using historical data. Risk was measured by the variance of returns, and inter-asset relationships were captured using a covariance matrix. These metrics served as the foundation for assessing overall portfolio behavior.

The return R of a portfolio with asset weights X_i and individual returns R_i is expressed as:

$$R = \sum_{i=1}^N X_i R_i$$

The portfolio risk, represented by variance V , is calculated as:

$$V = \sum_{i=1}^N \sum_{j=1}^N X_i X_j C_{ij}$$

where C_{ij} is the covariance between assets i and j , and N is the number of assets.

Simulation and Optimization

A Monte Carlo simulation approach was implemented to explore many possible portfolios by randomly assigning asset weights that satisfy the constraint:

$$\sum_{i=1}^N X_i = 1, \quad X_i \geq 0$$

Thousands of portfolios were generated and evaluated based on their respective risk and return values, and a risk-return scatter plot was employed to visualize the distribution of portfolio outcomes. To identify the optimal portfolio, constraints were defined for minimum and maximum acceptable portfolio risk. Within this range, the portfolio yielding the highest return was selected as the optimal solution. Summary statistics, including average return, average risk, and average investment weights, were then computed for all qualifying portfolios.

3 Results and Discussions

3.1 Ticker Selection Strategy

We selected these investment tickers to ensure a diversified portfolio that captures a broad range of asset classes and economic sectors, helping to reduce overall portfolio risk. Firstly, SPY acts as a benchmark for the overall U.S. equity market due to its comprehensive coverage of the S&P 500. Secondly, we included QQQ along with sector ETFs (such as XLF, XLE, XLV, XLP, and XLRE) to provide targeted exposure across key industries like technology, financials, energy, healthcare, consumer staples and real estate - capturing the effects of different phases of the economic cycle. VIG was chosen for its focus on high-quality companies with consistent dividend growth, offering stable income. AGG, as a bond ETF, adds low-risk, steady returns and acts as a hedge against equity market volatility. Finally, GLD offers protection against inflation and systemic risk, further enhancing portfolio resilience. All selected ETFs have high trading volumes and liquidity, ensuring that investors can buy or sell them at fair prices when needed.

3.2 Data Sources for Prices and Dividends

The data was retrieved via an API call in a Python program, using the `yfinance` library to access historical data directly from Yahoo Finance.

3.3 Selection of Desired Risk Range

A realistic range for the minimum and maximum desired risk was set between 0.0012 and 0.0015 (i.e., 12.0 to 15.0 percent squared).

This range was chosen based on practical considerations: risk levels below 0.0010 yield too few viable portfolio samples and overly conservative investments with low returns, while risk levels above 0.0020 introduce excessive volatility, which may not align with risk-averse investor preferences.

As shown in the risk-return scatter plot, the increase in return becomes marginal once the risk exceeds 15 percent squared. Within the selected range, we are able to identify portfolios that offer a balanced trade-off between moderate risk and relatively high returns.

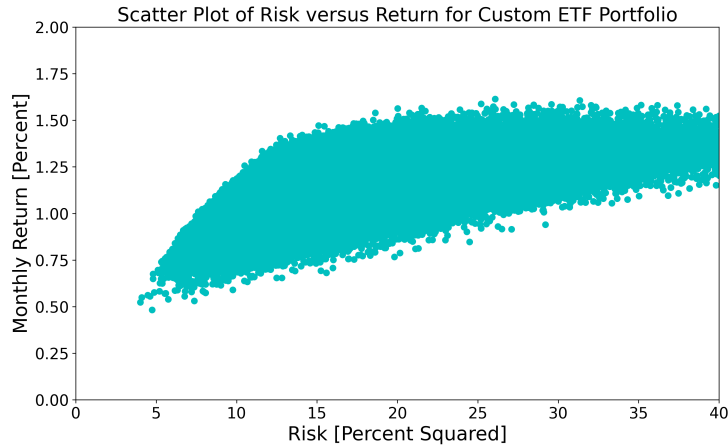


Figure 1: Scatter Plot of Risk versus Return for Custom ETF Portfolio

3.4 Simulation Parameters and Stability Considerations

In our analysis, we set `nSimulations` to 90, which provides a sufficient division of the risk interval. Values below 30 may result in a coarse segmentation that fails to capture subtle risk-return dynamics, while values above 100 significantly increase computation time with diminishing marginal benefits.

The minimum risk (`minRisk`) was set to 0.0010 (10.0 percent squared), as values below 0.0005 are overly conservative and yield few viable portfolios. The lower bound should include the most conservative investment options.

The maximum risk (`maxRisk`) was set to 0.0030 (30.0 percent squared), since values above 0.0050 typically represent overly aggressive portfolios that are unsuitable for risk-averse investors. The upper bound should cover a realistic range of portfolio choices while excluding extreme cases.

Improper parameter settings may lead to simulation issues. An overly high `minRisk` or overly low `maxRisk` can leave some risk intervals empty, while an excessively large `nSimulations` value increases computation time without causing a crash.

3.5 Portfolio Results: Risk, Return, and Asset Allocation

Based on the results of the simulation within the desired risk range, the average monthly risk was approximately 0.00134362 (13.44% squared annualized), and the average monthly return was approximately 1.39%.

The corresponding average investment weights for the 10 ETFs were:

ETF	SPY	QQQ	XLFX	XLE	XLV	XLP	XLRE	VIG	AGG	GLD
Weight	0.054	0.269	0.023	0.069	0.008	0.046	0.000	0.069	0.000	0.462

Table 1: Average Investment Weights

Investment allocation based on a \$10,000 portfolio:

ETF	SPY	QQQ	XLFX	XLE	XLV	XLP	XLRE	VIG	AGG	GLD
Amount (\$)	538	2692	231	692	77	462	0	692	0	4615

Table 2. Investment Allocation Based on \$10,000

4 Conclusion

In this project, a six-year historical dataset of ten diversified investments was used to construct and simulate a portfolio using Python. The selected tickets represented different sectors and regions to ensure diversification.

Appendix

```

1 #Compute asset allocation and the resulting risk
   versus return based on
2 #Markowitz 1952
3 #Take closing price of 10 stocks
4 #This code uses a randomized approach to assign the
   portfolio weights
5 #Import libraries
6 import numpy
7 import random
8
9 #Analysis with custom data
10 inputFileName = "StockHistory-10-Custom.txt"
11 outputFileName = "PortfolioAnalysis-10-Custom.txt"
12
13 #Symbols:
14 #SPY, QQQ, XLF, XLE, XLV, XLP, XLRE, VIG, AGG, GLD
15 StockIndex = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
16 StockDividend = [0.015, 0.007, 0.018, 0.032, 0.014,
   0.025, 0.03, 0.018, 0.025, 0.0]
17
18 #Load data into vectors

```

```

19 data = numpy.loadtxt(inputFileName)
20 nDataset = len(data)
21 nYears = 6
22 nStocks = 10
23 nSample = 500000
24
25 # For monthly return analysis consider this value for
    number of days in a month
26 returnFrequencyDays = int(nDataset / (nYears * 12))
27 daysInYear = int(nDataset/nYears)
28
29 #Define price matrix to be filled later
30 StockPrice = numpy.zeros((nDataset, nStocks))
31 for i in range(0, nStocks):
32     for j in range(0, nDataset):
33         StockPrice[j][i] = data[j, StockIndex[i]]
34
35 #First calculate the stock price return, i.e. share
    gain, through price history
36 StockShareGain = numpy.zeros((nDataset-
    returnFrequencyDays, nStocks))
37 StockReturn = numpy.zeros((nDataset-
    returnFrequencyDays, nStocks))
38 x = [i for i in range(0, returnFrequencyDays)]
39
40 for i in range(0, nStocks):
41     #Find the returns in each of the time periods
42     for j in range(0, nDataset-returnFrequencyDays):
43         StockShareGain[j][i] = (StockPrice[j +
            returnFrequencyDays, i] -
44                                 StockPrice[j, i]) /
45                                 StockPrice[j, i]
46         StockReturn[j][i] = StockShareGain[j][i] + \
            StockDividend[i]*
            returnFrequencyDays/
            daysInYear
47
48 MeanStockReturn = numpy.zeros((nStocks))

```

```

49 CovStockReturn = numpy.zeros((nStocks, nStocks))
50
51 # Set vectors y and z
52 Vectory = numpy.zeros((nDataset-returnFrequencyDays))
53 Vectorz = numpy.zeros((nDataset-returnFrequencyDays))
54
55 #print the return of the selected stocks
56 for i in range(0, nStocks):
57     MeanStockReturn[i] = numpy.mean(StockReturn[:, i])
58     print('Stock Index, Average Monthly Return [
59         Percent]: ',
60         i, numpy.round(100 * MeanStockReturn[i], 2))
61
62 for y in range(0, nStocks):
63     for z in range(0, nStocks):
64         for x in range(0, nDataset-returnFrequencyDays
65             ):
66             Vectory[x] = StockReturn[x][y]
67             Vectorz[x] = StockReturn[x][z]
68             Covariance = numpy.cov(Vectory, Vectorz)
69             CovStockReturn[y][z] = Covariance[0][1]
70             print('Stock Index y, z, Covariance of Return
71                 [Percent Squared]: ',
72                 y, z, numpy.round(10000 * CovStockReturn
73                     [y][z], 1))
74
75 #Now we should sample random portfolios for
76 #weights X0, X1, ... , X9 >= 0 subject to sum Xi = 1
77 X = numpy.zeros((nSample, nStocks))
78 Return = numpy.zeros((nSample))
79 Risk = numpy.zeros((nSample))
80
81 #Find random combinations of X0, X2, ..., X9
82 #assuming Delta X = 0.1, subject to Sum Xi = 1
83 #Loop over sample
84 for a in range(0, nSample):
85     #Random selection of stock
86     for b in range(0, nStocks):

```

```

83         RandStock = random.randint(0, nStocks-1)
84         X[a][RandStock] = X[a][RandStock] + 1/nStocks
85
86     #Calculate the sum of X0 to X9 and warn user if
87     not close to 1
88     SumX = numpy.sum(X[a][:])
89     if SumX > 1.001 or SumX < 0.999:
90         print('Warning, Sum of X[a][:] is not close to
91             1, it is: ', SumX)
92
93     Return[a] = X[a][0] * MeanStockReturn[0] + \
94                 X[a][1] * MeanStockReturn[1] + \
95                 X[a][2] * MeanStockReturn[2] + \
96                 X[a][3] * MeanStockReturn[3] + \
97                 X[a][4] * MeanStockReturn[4] + \
98                 X[a][5] * MeanStockReturn[5] + \
99                 X[a][6] * MeanStockReturn[6] + \
100                 X[a][7] * MeanStockReturn[7] + \
101                 X[a][8] * MeanStockReturn[8] + \
102                 X[a][9] * MeanStockReturn[9]
103
104     # To calculate risk, all the variances and
105     covariances must be considered
106     for y in range(0, nStocks):
107         for z in range(0, nStocks):
108             Risk[a] = Risk[a] + X[a][y] * X[a][z] *
109             CovStockReturn[y][z]
110
111 #Write data to file
112 outputFile = open(outputFileName, "w")
113 outputFile.write("#Sample \t Risk \t Return \t "
114                 "X0 \t X1 \t X2 \t X3 \t X4 \t "
115                 "X5 \t X6 \t X7 \t X8 \t X9 \n")
116
117 #Now loop through data points and write to file
118 for a in range(0, nSample):
119     outputFile.write("%i \t %f \t %f \t "
120                     "%f \t %f \t %f \t %f \t %f \t "

```

```

117         "%f \t %f \t %f \t %f \t %f \n"
118         % (a, Risk[a], Return[a],
119           X[a][0], X[a][1], X[a][2], X[a]
120             [3], X[a][4],
121           X[a][5], X[a][6], X[a][7], X[a]
122             [8], X[a][9]))
123
124 #Close file
125 outputFile.close()
126
127 print("\nData analysis complete, results saved to",
128       outputFileFileName)
129
130 print("\nPortfolio analysis description:")
131 print("X0 = SPY (S&P 500 ETF)")
132 print("X1 = QQQ (Nasdaq 100 ETF)")
133 print("X2 = XLF (Financial Sector ETF)")
134 print("X3 = XLE (Energy Sector ETF)")
135 print("X4 = XLV (Healthcare Sector ETF)")
136 print("X5 = XLP (Consumer Staples Sector ETF)")
137 print("X6 = XLRE (Real Estate Sector ETF)")
138 print("X7 = VIG (Dividend Growth ETF)")
139 print("X8 = AGG (US Bond ETF)")
140 print("X9 = GLD (Gold ETF)")

```

Algoritmo 1: Analysis1

```

1  # Portfolio selection to maximize return for a given
   level of risk
2  # A portfolio of 10 ETFs
3
4  # Import libraries
5  import numpy
6  import matplotlib.pyplot as plt
7
8  # ETFs: SPY, QQQ, XLF, XLE, XLV, XLP, XLRE, VIG, AGG,
   GLD
9  inputFileFileName = "PortfolioAnalysis-10-Custom.txt"
10 InvAmount = 10000

```

```

11 minRisk = 0.0010
12 maxRisk = 0.0030
13 nSimulations = 90
14 minRiskDesired = 0.0012
15 maxRiskDesired = 0.0015
16
17 dRisk = (maxRisk - minRisk) / nSimulations
18 nAverage = 0.0
19 bestRiskSum = 0.0
20 bestReturnSum = 0.0
21 X0Sum = 0.0
22 X1Sum = 0.0
23 X2Sum = 0.0
24 X3Sum = 0.0
25 X4Sum = 0.0
26 X5Sum = 0.0
27 X6Sum = 0.0
28 X7Sum = 0.0
29 X8Sum = 0.0
30 X9Sum = 0.0
31
32 # Load data into vectors
33 data = numpy.loadtxt(inputFileName)
34 Risk = data[:, 1]
35 Return = data[:, 2]
36 X0 = data[:, 3] # SPY
37 X1 = data[:, 4] # QQQ
38 X2 = data[:, 5] # XLF
39 X3 = data[:, 6] # XLE
40 X4 = data[:, 7] # XLV
41 X5 = data[:, 8] # XLP
42 X6 = data[:, 9] # XLRE
43 X7 = data[:, 10] # VIG
44 X8 = data[:, 11] # AGG
45 X9 = data[:, 12] # GLD
46 nDataset = len(data)
47
48 # Plot risk versus return

```

```

49 plt.rc('xtick', labels=16)
50 plt.rc('ytick', labels=16)
51 plt.figure(figsize=(12, 7))
52 plt.title('Scatter Plot of Risk versus Return for
    Custom ETF Portfolio', fontsize=20)
53 plt.plot(10000 * Risk, 100 * Return, 'co')
54 plt.xlim([0, 40])
55 plt.ylim([0, 2])
56 plt.xlabel('Risk [Percent Squared]', fontsize=20)
57 plt.ylabel('Monthly Return [Percent]', fontsize=20)
58 plt.savefig('RiskVersusReturn_Custom.png', dpi=300)
59 plt.show()
60
61 print('Best portfolio: Risk, Return, \n'
62       'X0(SPY), X1(QQQ), X2(XLF), X3(XLE), X4(XLV), X5
        (XLP), X6(XLRE), X7(VIG), X8(AGG), X9(GLD) \n
        ')
63
64 for j in range(0, nSimulations):
65     bestReturn = -100
66     for i in range(0, nDataset):
67         if (Risk[i] > minRisk + j * dRisk) \
68             and (Risk[i] < minRisk + (j + 1) *
                dRisk) \
69             and (Return[i] > bestReturn):
70                 bestRisk = Risk[i]
71                 bestReturn = Return[i]
72                 X0Best = X0[i]
73                 X1Best = X1[i]
74                 X2Best = X2[i]
75                 X3Best = X3[i]
76                 X4Best = X4[i]
77                 X5Best = X5[i]
78                 X6Best = X6[i]
79                 X7Best = X7[i]
80                 X8Best = X8[i]
81                 X9Best = X9[i]
82     if (bestRisk >= minRiskDesired) and (bestRisk <=

```

```

maxRiskDesired):
83     nAverage += 1
84     bestRiskSum += bestRisk
85     bestReturnSum += bestReturn
86     X0Sum += X0Best
87     X1Sum += X1Best
88     X2Sum += X2Best
89     X3Sum += X3Best
90     X4Sum += X4Best
91     X5Sum += X5Best
92     X6Sum += X6Best
93     X7Sum += X7Best
94     X8Sum += X8Best
95     X9Sum += X9Best
96
97     print('Simulation, Best Risk [Percent Squared],
           Best Monthly Return [Percent]: ',
           j, numpy.round(10000 * bestRisk, 1), numpy.
           round(100 * bestReturn, 2))
99     print('X0 to X9: ', X0Best, X1Best, X2Best, X3Best
           , X4Best,
           X5Best, X6Best, X7Best, X8Best, X9Best)
101
102     print('\nPortfolio Analysis Results:')
103     print('AvgRisk [Percent Squared], Monthly AvgReturn [
           Percent]: ')
104     print('%0.4f %0.4f' % (10000 * bestRiskSum / nAverage,
           100 * bestReturnSum / nAverage))
105     print('\nAverage ETF Weights:')
106     print('X0(SPY) X1(QQQ) X2(XLF) X3(XLE) X4(XLV) X5(XLP)
           X6(XLRE) X7(VIG) X8(AGG) X9(GLD): ')
107     print('%0.3f %0.3f %0.3f %0.3f %0.3f %0.3f %0.3f %0.3f
           %0.3f %0.3f' %
           (X0Sum / nAverage, X1Sum / nAverage, X2Sum /
           nAverage, X3Sum / nAverage, X4Sum / nAverage,
           X5Sum / nAverage, X6Sum / nAverage, X7Sum /
           nAverage, X8Sum / nAverage, X9Sum / nAverage
           ))
109

```

```

110 print('\nInvestment Allocation Based on $10,000:')
111 print('SPY    QQQ    XLF    XLE    XLV    XLP    XLRE    VIG
      AGG    GLD: ')
112 print('%5.0f %5.0f %5.0f %5.0f %5.0f %5.0f %5.0f %5.0f
      %5.0f %5.0f'
113         % (X0Sum / nAverage * InvAmount, X1Sum /
114             nAverage * InvAmount,
115             X2Sum / nAverage * InvAmount, X3Sum /
116             nAverage * InvAmount, X4Sum / nAverage *
117             InvAmount,
118             X5Sum / nAverage * InvAmount, X6Sum /
119             nAverage * InvAmount, X7Sum / nAverage *
120             InvAmount,
121             X8Sum / nAverage * InvAmount, X9Sum /
122             nAverage * InvAmount))

```

Algoritmo 2: Analysis2

```

1 import yfinance as yf
2 import pandas as pd
3 import numpy as np
4 import time
5 from datetime import datetime, timedelta
6
7 # Calculate date from 6 years ago
8 end_date = datetime.now()
9 start_date = end_date - timedelta(days=6*365)
10
11 # Our selected 10 ETFs
12 symbols = ['SPY', 'QQQ', 'XLF', 'XLE', 'XLV', 'XLP', '
      XLRE', 'VIG', 'AGG', 'GLD']
13
14 # Create an empty DataFrame to store closing prices
15 all_data = pd.DataFrame()
16
17 # Get monthly closing prices for each ETF
18 for symbol in symbols:
19     print(f"Getting historical data for {symbol}...")

```

```

20
21     try:
22         # Use yfinance to get monthly data
23         data = yf.download(symbol, start=start_date,
24                             end=end_date, interval='1mo')
25
26         # Extract closing prices
27         close_prices = data['Close']
28
29         # Rename column and add to total dataframe
30         close_prices.name = symbol
31
32         # Merge data
33         if all_data.empty:
34             all_data = pd.DataFrame(close_prices)
35         else:
36             all_data = all_data.join(close_prices, how
37                                     ='outer')
38
39         # Avoid too frequent API requests
40         time.sleep(1)
41
42     except Exception as e:
43         print(f"Error getting data for {symbol}: {e}")
44
45     # Fill missing values (if any)
46     all_data = all_data.fillna(method='ffill')
47
48     # Save as text file, format same as original file
49     header = "#" + "\t".join(symbols)
50     np.savetxt('StockHistory-10-Custom.txt', all_data.
51               values, delimiter='\t', header=header, comments='')
52
53     print("Data collection complete, saved to StockHistory
54           -10-Custom.txt")
55
56     # Get annual dividend data
57     print("\nGetting annual dividend data...")

```

```

54 dividends = []
55
56 for symbol in symbols:
57     try:
58         # Get stock information
59         stock = yf.Ticker(symbol)
60
61         # Try to get dividend yield
62         try:
63             div_yield = stock.info.get('dividendYield', 0)
64             if div_yield is None:
65                 div_yield = 0
66         except:
67             div_yield = 0
68
69         dividends.append(div_yield)
70         print(f"Estimated annual dividend rate for {
71             symbol}: {div_yield:.4f}")
72
73     except Exception as e:
74         print(f"Error getting dividend data for {
75             symbol}: {e}")
76         dividends.append(0.01) # Default value
77
78 print("\nDividend data list:")
79 print(dividends)
80 print("\nFormat for Python code:")
81 print(f"StockDividend = {dividends}")

```

Algoritmo 3: Collect Stock Data