Portfolio Optimization

Anthony Li

September 2023

Introduction

Portfolio optimization is a fundamental concept in finance that involves designing an investment portfolio with the goal of achieving the best possible trade-off between risk and return. There have been many techniques and studies in this area. The most popular methods involve maximizing sharpe ratio, minimising variance, and portfolio efficient frontiers.

In this project, the focus is to explore different methods to calculate the optimal portfolio weights across the SP500 component stocks. We will fix the rebalancing frequency to monthly across all methods and use a 12 month training period.

We will use daily returns for calculating the expected return for each stock and the covariance matrix. For calculating the monthly returns of the portfolio, we will used the monthly returns for each stock. The monthly returns are calculated as the percentage change between the last trading days of each month.

Libraries and Data Preprocessing

```
# Importing Libraries
import statsmodels.api as sm
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
# Import optimization libraries
from scipy.optimize import minimize
```

```
# Connecting Google Collaboratory to Google Drive folder
from google.colab import drive
drive.mount('/content/gdrive')

import os
os.chdir("./gdrive/My Drive/Quantitative Finance Projects/Statistical Arbitrage Trading Strategy/")
```

```
     Mounted at /content/gdrive
```

```
# Importing .csv of the prices of all stocks in the S&P500
Prices = pd.read_csv("price.csv", index_col=0)

# For simplicity, dropping all stocks that have any NaN
Prices =Prices.dropna(axis=1)

Prices.head()
```

	A	AAL	AAP	AAPL	ABC	ABT	ACGL	ACN	
Date									
2010-01-04	20.434929	4.496876	38.432693	6.572423	22.101795	19.778837	7.994444	32.970871	37.09
2010-01-05	20.212959	5.005957	38.204258	6.583786	21.944103	19.619041	7.967778	33.174648	37.70
2010-01-06	20.141132	4.798555	38.537392	6.479064	21.736616	19.727999	7.933333	33.527321	37.61
2010-01-07	20.115025	4.939965	38.527885	6.467087	21.388039	19.891426	7.886667	33.495968	36.88
2010-01-08	20.108498	4.845690	38.680153	6.510081	21.620419	19.993120	7.871111	33.362751	36.68

5 rows × 438 columns

```
# Convert the 'Date' column to a datetime index
Prices['Date'] = pd.to_datetime(Prices.index)
Prices.set_index('Date', inplace=True)
```

```
# Calculate the monthly returns
monthly_returns = Prices.resample('M').ffill().pct_change()


# Calculate daily returns
returns = Prices.pct_change().dropna()


dates = {'date': Prices.index.values}
dates = pd.DataFrame(dates)

dates['date'] = pd.to_datetime(dates['date'])

dates['year_month'] = dates['date'].dt.to_period('M')


returns['YEAR_MONTH'] = dates['year_month'].values[1:]
Prices['YEAR_MONTH'] = dates['year_month'].values
monthly_returns['YEAR_MONTH'] = returns['YEAR_MONTH'].unique()


Months = returns['YEAR_MONTH'].unique()
Months
```

```
<PeriodArray>
['2010-01', '2010-02', '2010-03', '2010-04', '2010-05', '2010-06', '2010-07',
 '2010-08', '2010-09', '2010-10',
 ...
 '2019-03', '2019-04', '2019-05', '2019-06', '2019-07', '2019-08', '2019-09',
 '2019-10', '2019-11', '2019-12']
Length: 120, dtype: period[M]
```

Table of Contents

1 Modern Portfolio Theory

Modern Portfolio Theory (MPT) is a foundational concept in finance developed by Harry Markowitz in the 1950s. It provides a framework for optimizing investment portfolios by considering the trade-off between risk and return. Here's a summary of Modern Portfolio Theory:

Key Concepts:

1. **Diversification:** MPT emphasizes the benefits of diversifying investments across a mix of assets rather than concentrating in a single asset. Diversification can reduce portfolio risk while maintaining or enhancing returns.
2. **Risk and Return Relationship:** MPT recognizes that investors seek to maximize returns while minimizing risk. It defines risk as the volatility or standard deviation of an asset's returns and suggests that investors can achieve higher expected returns by accepting more risk.
3. **Efficient Frontier:** MPT introduces the concept of the efficient frontier, a set of portfolios that offer the highest expected return for a given level of risk or the lowest risk for a given level of return. Portfolios on the efficient frontier are considered optimal.
4. **Mean-Variance Optimization (MVO):** MPT's cornerstone is MVO, which aims to find the optimal portfolio by considering the expected returns and covariances of assets. MVO seeks the portfolio that maximizes return or minimizes risk, given an investor's risk tolerance.
5. **Asset Correlations:** MPT acknowledges that the correlation and covariance between asset returns play a crucial role in determining portfolio risk. Negative or low correlations between assets enhance diversification benefits.

2 Mean-Variance Optimization

In mean-variance optimization, we measure risk as the variance of the portfolio returns and the return to be the expected portfolio return. The objective is to maximize the expected return while minimizing the volatility of return.

Key Concepts:

1. **Expected Return (Mean):** MVO starts by estimating the expected return for each asset in the portfolio. This represents the average or most likely return an investor can expect to earn from each asset.
2. **Risk (Variance or Standard Deviation):** MVO quantifies risk as the variance or standard deviation of returns for each asset. Higher risk assets have greater variability in returns.
3. **Covariance Matrix:** MVO considers the relationships between asset returns by calculating the covariance between pairs of assets. A positive covariance indicates that assets tend to move in the same direction, while a negative covariance suggests they move in opposite directions.

2.1 Minimium Variance Portfolio

The Minimum Variance Portfolio (MVP) is a specific portfolio within the context of Modern Portfolio Theory (MPT). It is a portfolio that is constructed to minimize the overall portfolio variance, or equivalently, the portfolio's standard deviation. The MVP represents the portfolio with the lowest possible level of risk for a given set of assets or asset classes.

Key characteristics of the Minimum Variance Portfolio include:

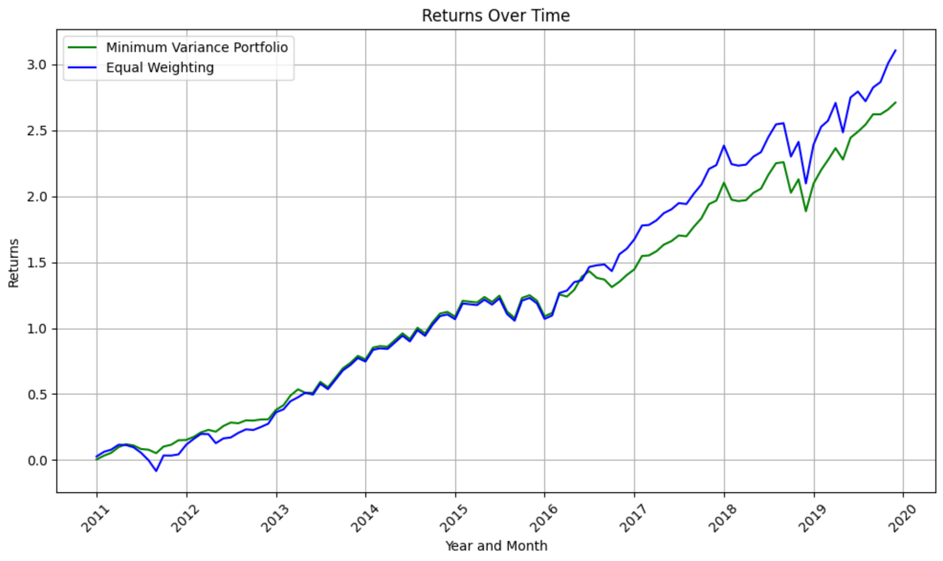
1. **Risk Minimization:** The primary objective of the MVP is to minimize portfolio risk. It seeks to achieve the lowest possible level of volatility or standard deviation of returns.
2. **Diversification:** The MVP typically achieves its risk reduction through diversification. By allocating investments across different assets with low or negative correlations, it reduces the portfolio's sensitivity to the price movements of any single asset.
3. **Efficient Frontier:** The MVP is one point on the Efficient Frontier, a curve that represents all possible portfolios that offer the highest expected return for a given level of risk or the lowest risk for a given level of expected return. The Efficient Frontier is derived from Mean-Variance Optimization (MVO), which considers the trade-off between risk and return.
4. **Weighting Scheme:** The asset allocation or weighting of assets in the MVP depends on the covariance matrix of asset returns. The assets with lower correlations and lower individual volatilities will typically receive higher allocations in the MVP.
5. **Risk-Return Trade-Off:** While the MVP minimizes risk, it may not offer the highest expected return. The MVP is considered the portfolio of choice for extremely risk-averse investors who prioritize capital preservation over return maximization.

It's important to note that the Minimum Variance Portfolio is just one point on the Efficient Frontier, and investors may choose portfolios that lie on the Efficient Frontier based on their individual risk tolerance and return objectives. The trade-off between risk and return is a central concept in portfolio management, and the MVP represents the extreme end of risk minimization within the context of MPT.

```
# Define an optimization function to minimize portfolio variance
def objective(weights):
    portfolio_variance = np.dot(weights.T, np.dot(cov_matrix, weights))
    return portfolio_variance


# Create an empty DataFrame with the predefined indexes
strategy1_returns = pd.DataFrame(index=Months[12:])

# Add a column for returns with no values (initially set to None)
strategy1_returns['returns'] = None


for month in Months[12:]:

  training_months = Months[ (Months < month) & (Months > month-13)]

  returns_training = returns[returns['YEAR_MONTH'].isin(training_months)]

  # Calculate the covariance matrix of returns
  cov_matrix = returns_training.cov()

  # Calculate the mean returns for each stock
  mean_returns = returns_training.mean()

  # Number of assets (stocks)
  num_assets = len(mean_returns)

  # Define an array of equal initial weights for the portfolio
  initial_weights = np.ones(num_assets) / num_assets

  # Set constraints for the optimization (sum of weights must equal 1)
  constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})

  # Set bounds for the weights (0 <= weight <= 1)
  bounds = tuple((0, 1) for asset in range(num_assets))

  # Run the optimization to find minimum variance portfolio weights
  result = minimize(objective, initial_weights, method='SLSQP', bounds=bounds, constraints=constraints)
  # Extract the optimized weights
  optimal_weights = result.x

  strategy1_returns.loc[month]['returns'] = (optimal_weights * monthly_returns[monthly_returns['YEAR_MONTH'] == month].iloc[:,:-1]).sum(a


strategy1_returns_cum = strategy1_returns.astype(float)

strategy1_returns_cum = (1 + strategy1_returns_cum['returns']).cumprod() - 1

equal_weight_returns =  monthly_returns[12:].sum(axis=1)/num_assets

equal_weight_returns = (1 + equal_weight_returns).cumprod() - 1
```

```
    <ipython-input-14-b08ff186125d>:5: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
      equal_weight_returns =  monthly_returns[12:].sum(axis=1)/num_assets
```

```
# Plot the returns against the year and month
plt.figure(figsize=(10, 6))
plt.plot(strategy1_returns_cum.index.to_timestamp(), strategy1_returns_cum, linestyle='-', label='Minimum Variance Portfolio', color='gre
plt.plot(strategy1_returns_cum.index.to_timestamp(), equal_weight_returns, color='blue', label='Equal Weighting')
plt.xlabel('Year and Month')
plt.ylabel('Returns')
plt.title('Returns Over Time')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(loc='upper left')
plt.show()
```



2.2 Maximum Sharpe Ratio

```
returns
```

	A	AAL	AAP	AAPL	ABC	ABT	ACGL	ACN	
Date									
2010-01-05	-0.010862	0.113208	-0.005944	0.001729	-0.007135	-0.008079	-0.003336	0.006181	0.01
2010-01-06	-0.003554	-0.041431	0.008720	-0.015906	-0.009455	0.005554	-0.004323	0.010631	-0.00
2010-01-07	-0.001296	0.029469	-0.000247	-0.001849	-0.016036	0.008284	-0.005882	-0.000935	-0.01
2010-01-08	-0.000324	-0.019084	0.003952	0.006648	0.010865	0.005112	-0.001972	-0.003977	-0.00
2010-01-11	0.000649	-0.019455	-0.009842	-0.008822	0.011133	0.005086	-0.003106	-0.000940	-0.01
...	...	...	...	...	...	...	...	...	
2019-12-23	0.000822	0.008953	0.011700	0.016318	-0.000817	0.007962	-0.014061	-0.001279	0.00
2019-	0.000469	-0.004096	0.003751	0.000951	-0.002103	-0.000801	-0.001426	0.003700	0.00

```
# Define the objective function to minimize - negative Sharpe ratio
def negative_sharpe_ratio(weights, expected_returns, cov_matrix, risk_free_rate):

    portfolio_return = np.sum(expected_returns * weights)

    portfolio_stddev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

    sharpe_ratio = (portfolio_return - risk_free_rate) / portfolio_stddev

    return -sharpe_ratio
```

```
# Create an empty DataFrame with the predefined indexes
strategy2_returns = pd.DataFrame(index=Months[12:])

# Add a column for returns with no values (initially set to None)
strategy2_returns['returns'] = None
```

```
# Importing Risk free rate
treasury_data = yf.download('^TNX', start='2010-01-05', end='2019-12-31', progress=False)

# Extract the yield of the 10-year Treasury note (which is often used as the risk-free rate)
risk_free_rate = treasury_data['Adj Close']/ 100

# Rename the column for clarity
risk_free_rate = risk_free_rate.rename('Risk_Free_Rate')
```

```
  # Number of assets (stocks)
  num_assets = len(mean_returns)

  # Define an array of equal initial weights for the portfolio
  initial_weights = np.ones(num_assets) / num_assets
```

```
initial_weights.shape()
```

```
(438,)
```

```
# Initial guess for weights (equal allocation)
initial_weights = [1 / len(mean_returns)] * len(mean_returns)
```

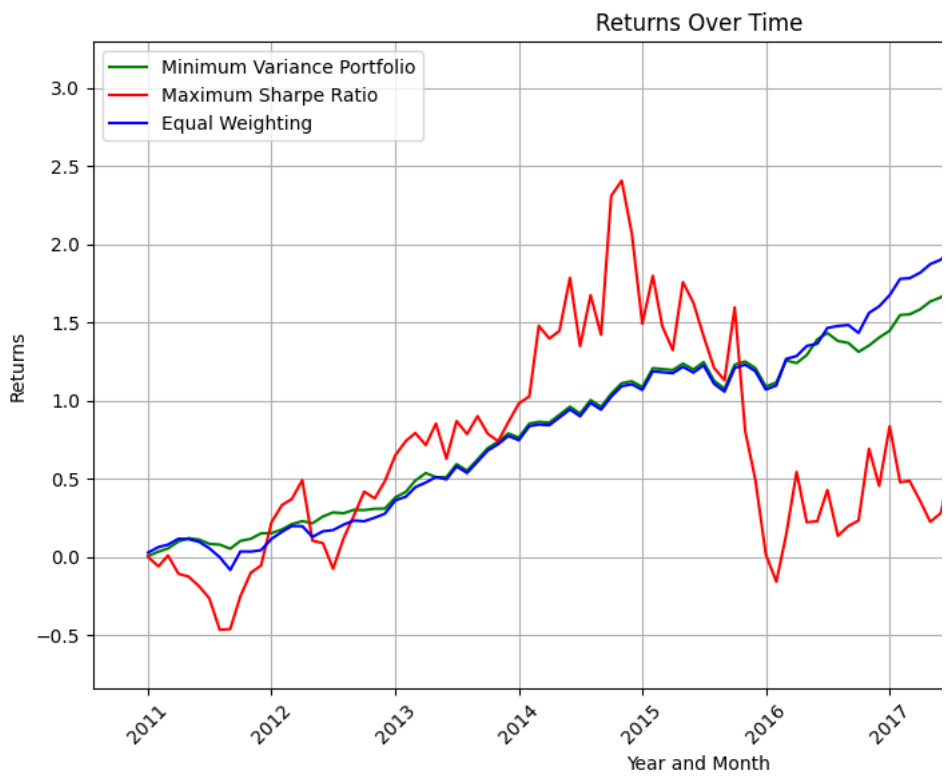
```
len(initial_weights)
```

```
438
```

```
returns = returns.merge(risk_free_rate, left_index=True, right_index=True, how='inner')
```

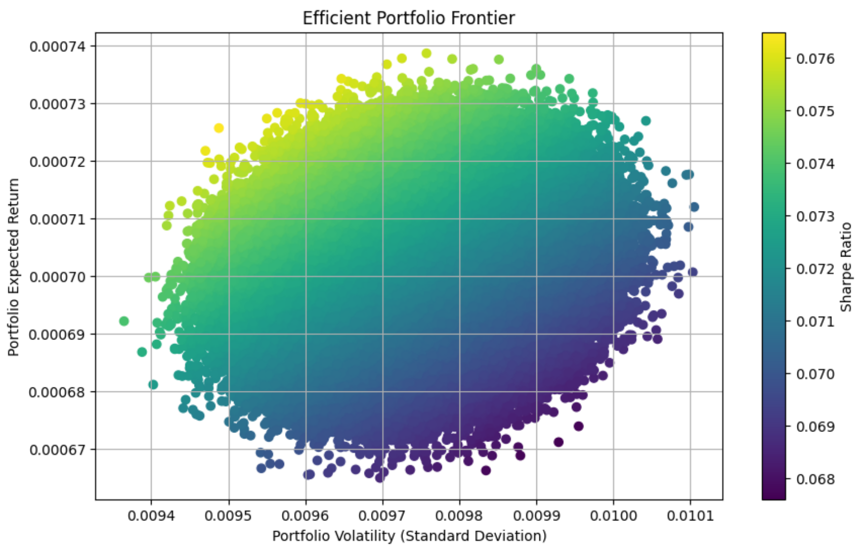
```
for month in Months[12:]:

  training_months = Months[ (Months < month) & (Months > month-13)]

  returns_training = returns[returns['YEAR_MONTH'].isin(training_months)].iloc[:,:-2]
  risk_free_rate = returns[returns['YEAR_MONTH'].isin(training_months)].iloc[:,-1]

  # Calculate the covariance matrix of returns
  cov_matrix = returns_training.cov()
```

```
      cov_matrix = returns_training.cov()

      # Calculate the mean returns for each stock
      mean_returns = returns_training.mean()

      mean_risk_free = risk_free_rate.mean()

      # Number of assets (stocks)
      num_assets = len(mean_returns)

      # Define an array of equal initial weights for the portfolio
      initial_weights = [1 / len(mean_returns)] * len(mean_returns)

      # Set constraints for the optimization (sum of weights must equal 1)
      constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})

      # Set bounds for the weights (0 <= weight <= 1)
      bounds = tuple((0, 1) for asset in range(num_assets))

      # Run the optimization to find minimum variance portfolio weights
      result = minimize(negative_sharpe_ratio, initial_weights, args=(mean_returns, cov_matrix, mean_risk_free),
                        method='SLSQP', bounds=bounds, constraints=constraints)

      # Extract the optimized weights
      optimal_weights = result.x

      strategy2_returns.loc[month]['returns'] = (optimal_weights * monthly_returns[monthly_returns['YEAR_MONTH'] == month].iloc[:,:-1]).sum(a


    strategy2_returns_cum = strategy2_returns.astype(float)

    strategy2_returns_cum = (1 + strategy2_returns_cum['returns']).cumprod() - 1


    # Plot the returns against the year and month
    plt.figure(figsize=(10, 6))
    plt.plot(strategy2_returns_cum.index.to_timestamp(), strategy1_returns_cum, linestyle='-', label='Minimum Variance Portfolio', color='gre
    plt.plot(strategy2_returns_cum.index.to_timestamp(), strategy2_returns_cum, linestyle='-', label='Maximum Sharpe Ratio', color='red')
    plt.plot(strategy2_returns_cum.index.to_timestamp(), equal_weight_returns, color='blue', label='Equal Weighting')
    plt.xlabel('Year and Month')
    plt.ylabel('Returns')
    plt.title('Returns Over Time')
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.legend(loc='upper left')
    plt.show()
```



2.3 Efficient Portfolio Frontier

```
# Define expected returns and standard deviations of assets
expected_returns = returns.iloc[:,:-2].mean()  # Replace with your asset returns
cov_matrix = returns.iloc[:,:-2].cov()


# Number of portfolios to simulate
num_portfolios = 1000000

# Initialize lists to store portfolio returns and volatilities
portfolio_returns = []
portfolio_volatilities = []

# Randomly generate portfolio weights and calculate returns/volatilities
for _ in range(num_portfolios):
    weights = np.random.random(len(expected_returns))
    weights /= np.sum(weights)  # Normalize to ensure they sum to 1

    portfolio_return = np.sum(weights * expected_returns)
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

    portfolio_returns.append(portfolio_return)
    portfolio_volatilities.append(portfolio_volatility)

# Convert lists to NumPy arrays for easier manipulation
portfolio_returns = np.array(portfolio_returns)
portfolio_volatilities = np.array(portfolio_volatilities)

# Create a dictionary to store portfolio statistics
portfolio_stats = {
    'Returns': portfolio_returns,
    'Volatility': portfolio_volatilities,
}

# Convert the dictionary to a Pandas DataFrame for easy analysis (optional)
import pandas as pd
df = pd.DataFrame(portfolio_stats)

# Plot the efficient portfolio frontier
plt.figure(figsize=(10, 6))
plt.scatter(portfolio_volatilities, portfolio_returns, c=portfolio_returns / portfolio_volatilities, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.title('Efficient Portfolio Frontier')
plt.xlabel('Portfolio Volatility (Standard Deviation)')
plt.ylabel('Portfolio Expected Return')
plt.grid(True)
plt.show()
```



Categorising into K groups and Portfolio Optimisation of each group

Kalman Filter for mean-variance optimisation

Dynamic Asset Allocation - adjust risk appetite based on economic signals

Estimating Covariance matrix better - Kalman Filters

Using DCC: Dynamic Conditional Correlation Models

Covariance Matrix: PCA

Comparison of twitter sp500 sentiment index and realised value (only available after 2018)

✓ 6m 10s completed at 11:06 PM