Integrated Asset Selection and Allocation for Portfolio Optimization

Arthur Costa Corvo (30158092)

Abstract

This project explores portfolio optimization through an integrated approach that combines asset selection and allocation, aiming to construct multiple portfolios with an optimal risk-adjusted return and selecting the top-performing portfolio.

Introduction

Portfolio optimization is a two-step process involving **asset selection** (deciding which assets to invest in) and **asset allocation** (determining how much to invest in each asset). In this project, I used Python to simultaneously integrate these two steps, evaluating portfolios formed from a larger set of securities, optimizing each portfolio's asset allocation to determine the overall optimal portfolio' asset selection.

Specifically, the program retrieves historical return data for two stocks from each of the eleven industries within the NASDAQ. With 22 total number of stocks and only 11 being selected (one from each industry), the program determines the $2^{11} = 2{,}048$ possible portfolio combinations. Then, it optimizes each portfolio's asset allocation to achieve the highest possible Sharpe ratio. Finally, it determines which portfolio achieves the highest possible Sharpe among all the combinations.

Goal

For each portfolio combination, the program calculates the Sharpe ratio, which represents a portfolio's excess return (above a risk-free rate) per unit of risk, providing a metric for comparing risk-adjusted returns. In this project, the risk-free rate is represented by yields on 10-year Treasury bills. The standard optimization problem is:

$$\max\ S = \frac{E[R_P] - R_f}{\sigma_P}, \qquad \text{or} \quad \max_{w_i}\ S = \frac{\sum_{i=1}^{n} w_i \times E[R_i] - R_f}{\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i \times w_j \times \sigma_{i,j}}}$$

Subject to:

$$0 \le w_i \le 0.18$$

Where:

S: Sharpe Ratio

$E[R_P]$: Portfolio Expected Return

R_f: Risk − Free Rate

σ_P: Portfolio Standard Deviation

$w_i, E[R_i]$: Security i's Weight, Expected Return $\forall i = 1, \dots, 11$

$\sigma_{i,j}$: Securities i and j Covariance of Returns

After calculating each of the 2,048 portfolio Sharpe ratios, the program states which portfolio combination achieves the highest risk-adjusted return overall, what were the securities selected, and the weights assigned. With this information, I was able to test this portfolio on a different scenario, which allowed me to see how well this portfolio fared in different market conditions.

Data Handling

The data handling file begins by defining a time range of 365 days for the training period (back test) and evaluation period (forward test). It then defines a dictionary of selected stocks, with two stocks from each of eleven sectors in the NASDAQ, along with the S&P 500 index as the market benchmark and the 10-year Treasury yield as the risk-free rate.

Term Project - 01_Data_Handling.py

```python
# Date Range
time_range = 365  # Time range for Backtest, Forward Test; in Days
end_date = datetime(2024, 11, 3)
beg_date = end_date - timedelta(days = 2 * time_range)

# Define Security Dictionary Keys
STOCK_1 = 'Stock #1'
STOCK_2 = 'Stock #2'

# Define 'The Market' and 'The Risk-Free' Tickers
market_ticker    = '^GSPC'  # S&P 500
risk_free_ticker = '^TNX'   # CBOE Interest Rate 10 Year T Note

# Define the Security Dictionary with Stock Tickers
securities_dict = {
    'Information Technology':   {STOCK_1: 'AAPL',  STOCK_2: 'MSFT'},
    'Healthcare':               {STOCK_1: 'LLY',   STOCK_2: 'NVO'},
    'Financials':               {STOCK_1: 'JPM',   STOCK_2: 'V'},
    'Consumer Discretionary':   {STOCK_1: 'AMZN',  STOCK_2: 'TSLA'},
    'Consumer Staples':         {STOCK_1: 'WMT',   STOCK_2: 'COST'},
    'Energy':                   {STOCK_1: 'BP',    STOCK_2: 'EQNR'},
    'Industrials':              {STOCK_1: 'GE',    STOCK_2: 'CAT'},
    'Materials':                {STOCK_1: 'LIN',   STOCK_2: 'BHP'},
    'Utilities':                {STOCK_1: 'NEE',   STOCK_2: 'SO'},
    'Real Estate':              {STOCK_1: 'PLD',   STOCK_2: 'AMT'},
    'Communication Services':   {STOCK_1: 'GOOGL', STOCK_2: 'META'}}
```

The program retrieves historical data for the selected stocks using the Yahoo Finance API. Daily log-normal returns are calculated for each stock, as well as for the market benchmark (S&P 500) and the risk-free rate (10-year Treasury yield). The data is then split into the back test and training periods.

Term Project - 01_Data_Handling.py

```python
# EXECUTION
## Security Returns
securities_lst = get_securities(securities_dict)
adjclose_df = get_data(securities_lst, beg_date, end_date)
lnreturn_df = calc_returns(adjclose_df)
backtest_df, frwdtest_df = split_data(lnreturn_df)

## Market Reuturns
adjclose_mkt = get_data(market_ticker, beg_date, end_date)
lnreturn_mkt = calc_returns(adjclose_mkt)
backtest_mk, frwdtest_mk = split_data(lnreturn_mkt)

## Risk-Free Returns
adjclose_rf = get_risk_free(risk_free_ticker, beg_date, end_date)
backtest_rf, frwdtest_rf = split_data(adjclose_rf)

# Exporting results to CSV
backtest_df.to_csv('backtest_df.csv')
backtest_rf.to_csv('backtest_rf.csv')
frwdtest_df.to_csv('frwdtest_df.csv')
frwdtest_mk.to_csv('frwdtest_mk.csv')

# Save securities_dict to a JSON file
with open('securities_dict.json', 'w') as f:
    json.dump(securities_dict, f)
```

Optimization

The optimization file begins by loading the results generated in the data handling step, including stock returns and risk-free rates for the first period. It then defines performance metric functions such as expected return, standard deviation, and Sharpe ratio. Since SciPy does not contain a maximization function, the program defined the negative Sharpe ratio which will be minimized. This effectively allows the algorithm to maximize the original Sharpe ratio during optimization.

Term Project - 02_Optimization.py

```
21  # Calculates Portfolio Expected Returns
22  def expected_return(weights, returns_df):
23      return np.sum(returns_df.mean() * weights) * 252
24      # Annualined Expected Return (252 Trading Days)
25
26  # Calculates Portfolio Standard Deviation
27  def standard_deviation(weights, cov_matrix):
28      return np.sqrt(weights.T @ cov_matrix @ weights)
29
30  # Calculates Portfolio Sharpe Ratio
31  def sharpe_ratio(weights, returns_df, cov_matrix, risk_free_rate):
32      portfolio_return = expected_return(weights, returns_df)
33      portfolio_stdev = standard_deviation(weights, cov_matrix)
34      return (portfolio_return - risk_free_rate) / portfolio_stdev
35
36  # Define Negative Sharpe Ratio for Minimization
37  def neg_sharpe_ratio(weights, returns_df, cov_matrix, risk_free_rate):
38      return -sharpe_ratio(weights, returns_df, cov_matrix, risk_free_rate)
39  # SOURCE: https://www.youtube.com/watch?v=9GA2WlYFeBU&t=973s
```

The next step involves generating all possible portfolio combinations from the predefined set of securities, with each combination containing one stock from each sector. These combinations are stored in a nested list to allow systematic iteration in the optimization process. Using the first portfolio combination, the file sets up the optimization constraints and bounds: the weights must be greater than or equal to zero (non-negativity), sum to 1 (100% allocation) and each stock weight is capped at 18%. Initial weights are equally distributed across the selected stocks in the first combination, providing a starting point for the optimization algorithm.

Term Project - 02_Optimization.py

```
41  # Generates List with All possible Portfolio Combinations
42  choices = [(sector[STOCK_1], sector[STOCK_2]) for sector in securities_dict.values()]
43  combinations = list(itertools.product(*choices))
44
45  # Uses the first combination to determine constraints and bounds applied to all combinations
46  first_combination = combinations[1]
47  constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}
48  bounds = [(0, 0.18) for _ in range(len(first_combination))]
49  initial_weights = np.array([1 / len(first_combination)] * len(first_combination))
```

The file then iterates through each combination in the list, calculating the Sharpe ratio for each possible portfolio configuration. During this loop, the minimize function from the SciPy library is used to adjust the weights within the specified bounds, maximizing the Sharpe ratio for each portfolio. Throughout the loop, the code keeps track of the portfolio with the highest Sharpe ratio, storing details such as the optimal Sharpe ratio value, stock combination, optimal weights, filtered data, and the covariance matrix of the selected assets.

Term Project - 02_Optimization.py

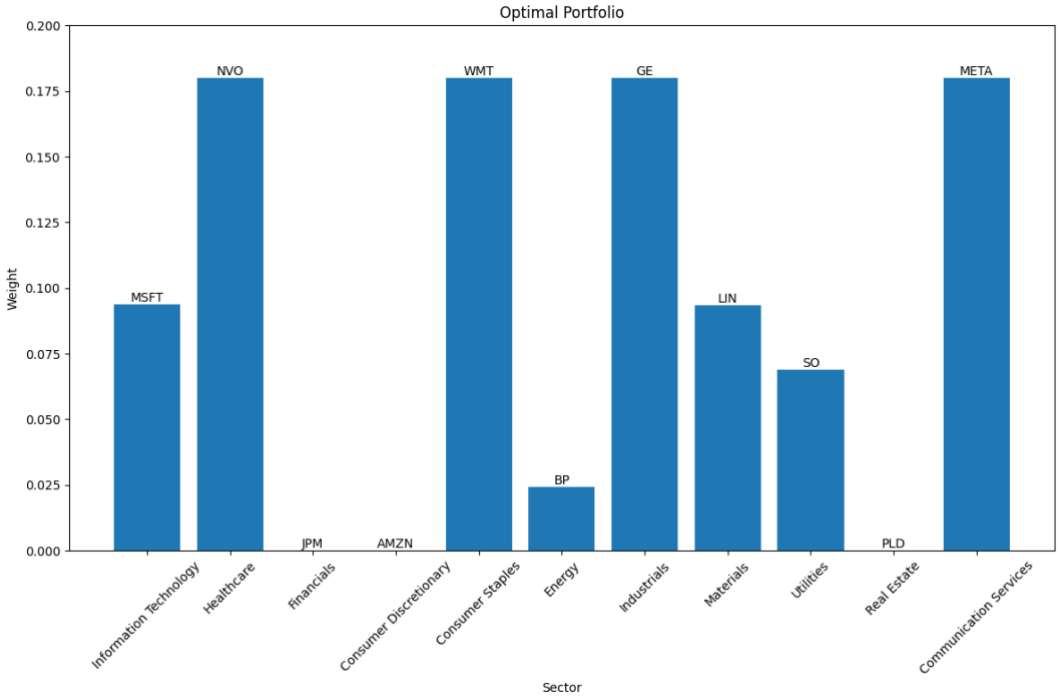
```python
# Main Algo Loop: Calculates Sharpe for each Combination
optimal_sharpe = 0  # Sets initial optimal Sharpe as 0
for combination in tqdm(combinations, desc = 'Evaluating Portfolio Combinations'):
    filtered_df = backtest_df[list(combination)]  # Filter dataframe for stocks in this combination
    covariance_mx = filtered_df.cov() * 252  # Calculate the covariance matrix for the dataframe
    result = minimize(neg_sharpe_ratio, initial_weights, args = (filtered_df, covariance_mx, risk_free),
                      method='SLSQP', constraints = constraints, bounds = bounds)
    # SOURCE: https://www.youtube.com/watch?v=9GA2WlYFeBU&t=973s

    sharpe = -result.fun.round(4)

    # Keeps track of the Optimal Sharpe
    if sharpe > optimal_sharpe:
        optimal_sharpe = sharpe                         # Records Optimal Portfolio Sharpe
        optimal_combination = combination               # Records Optimal Combination of Stocks
        optimal_weights = np.round(result.x, 4)         # Records Optimal Portfolio Weights (4 Decimals)
        optimal_returns_df = filtered_df.copy()         # Saves a Copy of Optimal Filtered Dataframe
        optimal_covariance_mx = covariance_mx.copy()    # Saves a Copy of Optimal Covariance Matrix
```

Finally, the file structures the optimal portfolio's details into a dataframe, categorizing each stock by its sector and storing the stock ticker along with its corresponding weight. This information will be used in the last step, in which will test this portfolio on the last year of data.

```python
# ORGANIZE RESULTS
data = []
for stock, weight in zip(optimal_combination, optimal_weights):
    # Finds the sector for each stock
    sector = next(sector_name for sector_name, stocks in securities_dict.items() if stock in stocks.values())
    data.append([sector, stock, weight])
# Converts the list to a DataFrame and arrange columns
optimal_portfolio_df = pd.DataFrame(data, columns = ['Sector', 'Stock', 'Weight'])
```

Optimization Results

The resulting optimal portfolio had a Sharpe ratio of 3.0856, an expected return of 0.5389, and a standard deviation of 0.1595.



Application

The application phase begins by loading the necessary datasets for forward testing. This includes the returns for all stocks during the second period, the overall market returns, and the optimized portfolio composition computed in the previous phase. The portfolio information is loaded from a JSON file and organized into a dataframe, which separates the stock tickers and their respective weights for further analysis.

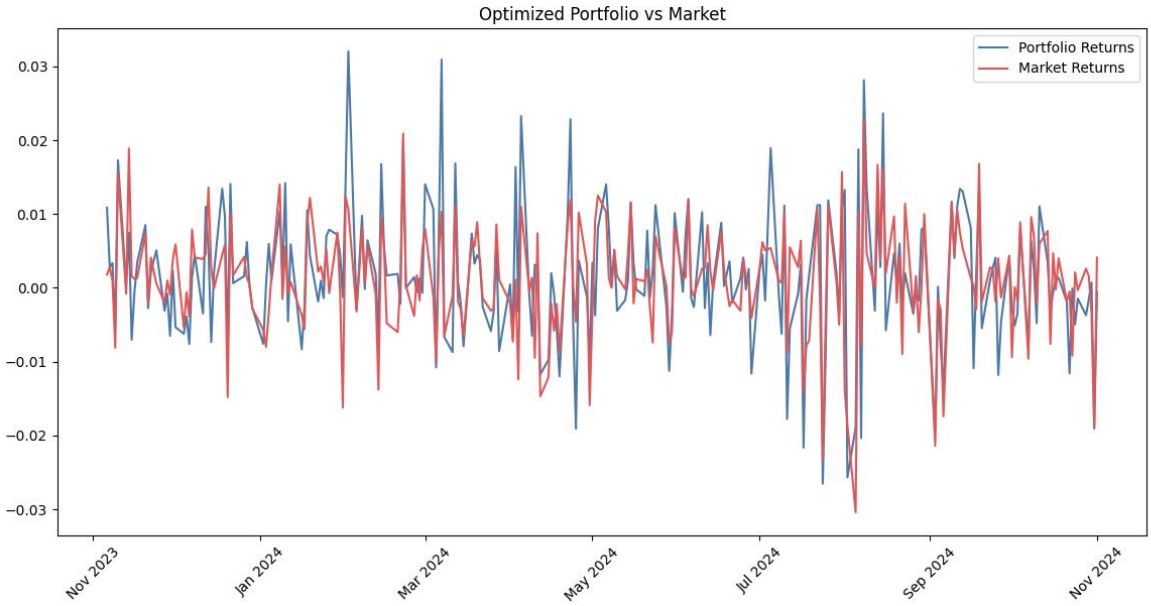
Term Project - 03_Application.py

```
9  # LOADING DATA
10 frwdtest_df = pd.read_csv('frwdtest_df.csv', index_col = 0)
11 frwdtest_mk = pd.read_csv('frwdtest_mk.csv', index_col = 0)
12 with open('optimal_portfolio.json', 'r') as json_file:
13     optimal_portfolio_data = json.load(json_file)
14
15 optimal_portfolio = pd.DataFrame.from_dict(optimal_portfolio_data, orient = 'index')
16 tickers = optimal_portfolio['Stock'].values
17 weights = optimal_portfolio['Weight'].values
```

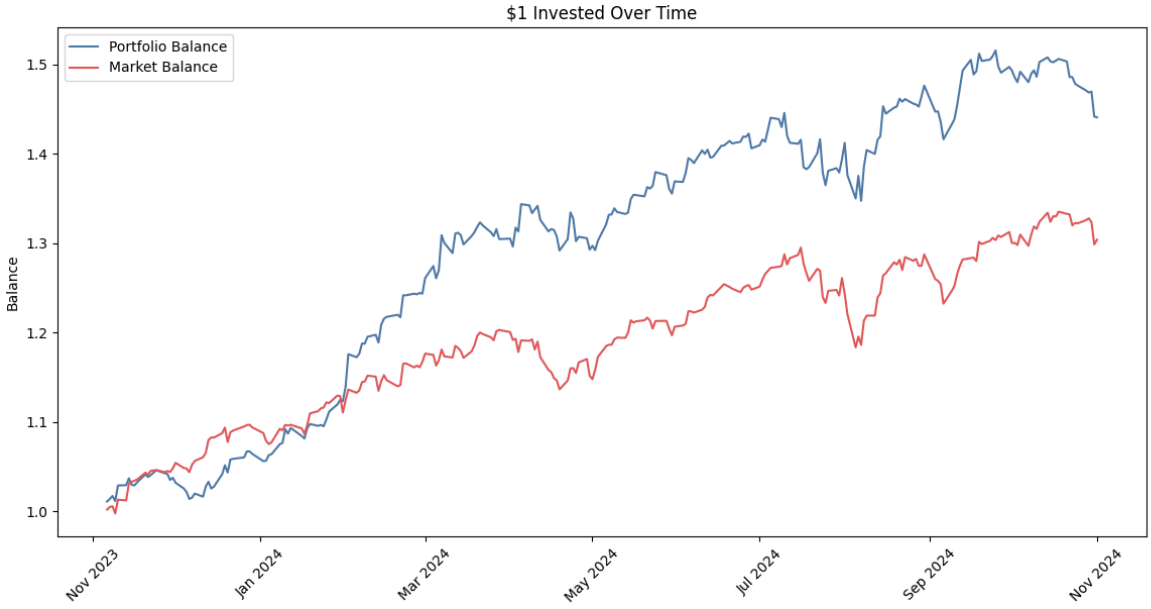
With the data loaded, the program filters the dataframe to retrieve only the selected stocks. It then calculates the portfolio's daily returns by taking a weighted sum of each stock's return and also retrieves the daily market return for comparison. To track portfolio growth, it calculates cumulative returns by compounding the daily returns, resulting in a daily balance trajectory for both the portfolio and the market. This setup enables a side-by-side comparison of the portfolio's performance relative to the broader market throughout the testing period.

```
19 # Creates a copy of the returns dataframe with only the relecant columns (tickers)
20 filtered_df = frwdtest_df[[ticker for ticker in tickers if ticker in frwdtest_df.columns]].copy()
21
22 # DAILY RETURNS
23 # Calculate the weighted sum (portfolio return) for each row and assign it to a new column
24 filtered_df['Portfolio Return'] = filtered_df.dot(weights)
25 filtered_df['Market Return'] = frwdtest_mk['^GSPC'] # Keeps Market Returns the Same
26
27 # DAILY BALANCES
28 # Step 1: Calculate Growth (1 + Portfolio Return) for each row
29 filtered_df['Portfolio Growth'] = 1 + filtered_df['Portfolio Return']
30 filtered_df['Market Growth'] = 1 + filtered_df['Market Return']
31
32 # Step 2: Calculate the Cumulative Return (Cumulative Product) for each row
33 filtered_df['Portfolio Balance'] = filtered_df['Portfolio Growth'].cumprod()
34 filtered_df['Market Balance'] = filtered_df['Market Growth'].cumprod()
35
36 # Create the results DataFrame with Returns and Balances
37 returns_df = filtered_df[['Portfolio Return', 'Portfolio Balance', 'Market Return', 'Market Balance']]
38 returns_df.index = pd.to_datetime(returns_df.index)
```

Application Results



Comparison of Daily Returns: The optimized portfolio demonstrates periods of heightened return volatility compared to the market, with multiple instances where its daily returns exceed those of the market. These peaks suggest that the portfolio's allocation effectively captured certain market upswings, capitalizing on opportunities to maximize gains on individual days. Despite this increased volatility, the alignment between the portfolio's returns and the market's overall pattern indicates that it responds similarly to broader market trends.



Comparison of Cumulative Growth: In terms of cumulative growth, the optimized portfolio shows a clear advantage over the market benchmark. Starting from an initial investment, the portfolio consistently generates higher returns, resulting in a significantly greater balance by the end of the period. This growth trajectory illustrates the success of the optimization process in enhancing risk-adjusted returns, allowing the portfolio to outperform the market steadily over time. The divergence in growth between the portfolio and the market widens over time, underscoring the portfolio's strong, sustained performance.

Conclusion

This program was successful in constructing a diversified portfolio that maximized risk-adjusted returns by optimizing both asset selection and allocation. Through a structured approach, it identified a combination of stocks across multiple sectors and determined the optimal weights for each to achieve the highest Sharpe ratio. The results from forward testing showed that the optimized portfolio outperformed the market benchmark over the test period.

However, it's important to acknowledge the role of hindsight bias in this process. For example, when selecting the securities, I did not know off the top of my head two stocks from each NASDAQ industry. I personally selected some of the stocks, such as Apple, Microsoft, Walmart, and Costco, but had to search for securities in other industries, such as Materials, Utilities, and Real Estate. When searching for the securities, I ended up looking at Forbes' industry rankings, which displayed the top-performing stocks for 2024 by industry. By choosing these stocks, it is no surprise that the portfolio yields positive returns during the forward test period, nor that it outperforms the overall market.

With this in mind, it is important to recognize that past performance does not guarantee future results. The optimization in this project and the application were based on historical data, but market conditions, economic factors, and individual stock performance can change unpredictably. While the portfolio was effective in this testing scenario, future results could vary significantly.

References:

- Forbes Magazine. (2024, November 1). *10 top materials stocks of 2024*. Forbes. https://www.forbes.com/advisor/investing/best-materials-stocks/
- Forbes Magazine. (2024b, November 1). *10 top utilities stocks of 2024*. Forbes. https://www.forbes.com/advisor/investing/best-utilities-stocks/
- Forbes Magazine. (2024, November 1). *10 top real estate stocks of 2024.* Forbes. https://www.forbes.com/advisor/investing/best-real-estate-stocks/
- Markowitz, H.M. (1952, March). "Portfolio Selection". The Journal of Finance. 7 (1): 77–91. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1952.tb01525.x
- O'Connell, Ryan (2023, May 1). Portfolio Optimization in Python: Boost Your Financial Performance. YouTube. https://www.youtube.com/watch?v=9GA2WlYFeBU&t=1032s