Designing & Implementation of a Robo-Advising System

Final Report

IEDA3330

Wong Cheng Loong (20916293), Nikhil Joseph (20895645), Yong Peng Tan (20924032), Raghav Shukla (20887612)

Table of Contents

1. Introduction

1.1. Objective

For this project, our group chose to focus on the example topic, “Designing and Implementing a Robo-Advising System,” for several key reasons:

Firstly, we believed that adopting a more hands-on approach to the project would foster the flow of creative and innovative ideas, enabling us to tackle real-world scenarios and successfully apply the concepts we have learned throughout the course.

Secondly, rather than solely focusing on the calculation and pricing of stocks, bonds, or other financial assets using historical data or machine learning techniques—which we felt would be too straightforward—we aimed to develop a product that could provide potential users with actionable insights. Specifically, we wanted to create a system that allowed users to identify the optimal returns they could achieve based on their individual risk tolerance when investing in various S&P 500 and other stocks.

In this report, we will outline the goals and objectives of the robo-advising system, provide insight into our planning process, and discuss the implementation and coding of the system. We will also include an example use case to help readers better understand how the system operates. Finally, we will conclude by reflecting on the overall process, discussing potential improvements that could be made to the system, and highlighting how we incorporated multiple course concepts into this project.

1.2. The Aim of the Robo-Advising System

The portfolio optimization system is designed to assist users in constructing an investment portfolio that aligns with their specific constraints and preferences. The system accepts user inputs, including the ticker names of the securities they wish to include in their portfolio and their desired investment constraints, such as expected return or acceptable risk levels. Based on this information, the system evaluates the feasibility of achieving the specified goals with the provided securities.

If the user’s desired return is too high or the desired risk is too low to be realistically achieved with the selected securities, the system notifies the user that the constraints are infeasible. This ensures that users have realistic expectations and prevents the creation of unattainable portfolio configurations.

If the constraints are feasible, the system calculates the optimal weightage for each security in the portfolio to meet the user’s specific goals. The calculated weights, expressed as decimals, represent the proportion of the total investment to allocate to each security. The system uses advanced mathematical techniques, such as quadratic programming and mean-variance optimization, to determine the allocation that minimizes risk or maximizes return, depending

on the user's objectives. The final output consists of a list of the selected securities and their corresponding ideal weights, presented in a clear and actionable format.

This system provides users with actionable recommendations that can be directly implemented in their brokerage accounts to construct portfolios that align with their investment goals and risk tolerance. It ensures efficiency by automating the portfolio optimization process and saves users time and effort while delivering precise, data-driven results. Additionally, the system enhances transparency by informing users when their constraints are unrealistic and suggesting feasible alternatives. By combining advanced optimization techniques with user-centric design, the portfolio optimization system empowers investors to make informed decisions and achieve their financial objectives effectively.

2. Designing the System

2.1 Initial Planning

During the initial planning phase, we began by defining the input and output requirements for the robo-advising system. Specifically, we decided that the system should allow users to input the securities they wish to include in their portfolio, along with their desired risk or return characteristics. The system would then output a set of optimal weights that constructs a portfolio satisfying these constraints.

To achieve this, we broke the problem into four smaller subproblems. The first step was to design a mechanism for the user to input stock ticker symbols, which would then be converted into a list of pre-processed and cleaned time series price data for each security. This step ensured that the raw user input could be transformed into usable data for further analysis. The second step involved calculating key statistics for each security, such as mean return, standard deviation, and variance, while also generating a covariance matrix to measure the relationships between the securities selected by the user. These calculations formed the foundation for constructing optimized portfolios.

With this secondary data in hand, the third step focused on data visualization. Specifically, we planned to generate and plot the efficient frontier, providing users with a visual representation of the trade-offs between risk and return for the securities in their portfolio. Finally, the fourth and final step was to ask the user for their desired risk or return constraints and, using the efficient frontier and calculated data, determine the optimal portfolio weights that satisfy the specified criteria. This step completed the user journey, from input to actionable output. The following walkthrough of our code will be presented in a similar fashion.

2.2. Coding

2.2.1. Data Acquisition

For the data acquisition process, we began by prompting the user to input the ticker symbols of the securities they wished to include in their portfolio, using a specified input format. The

input string was then parsed and separated into individual strings, each corresponding to a ticker symbol.

Using the Yahoo Finance API, accessed through the 'yfinance' library in python, these ticker symbols were passed into the API to retrieve the historical closing prices for the selected securities. The data was requested over a one-year period with a daily time step. We determined that this dataset provided a reasonable level of granularity, enabling the reliable calculation of key statistics such as standard deviation and mean return. However the date intervals can easily be changed later to account for different client needs.

```
Enter stock tickers, separated by commas: aapl,nvda,msft,amzn
You entered: ['aapl', 'nvda', 'msft', 'amzn']
                                 aapl       nvda        msft        amzn
Date
2021-01-06 00:00:00-05:00  123.796448  12.582565  205.164703  156.919006
2021-01-07 00:00:00-05:00  128.020798  13.310220  211.003052  158.108002
2021-01-08 00:00:00-05:00  129.125748  13.243137  212.288666  159.134995
2021-01-11 00:00:00-05:00  126.123718  13.587016  210.229813  155.710495
2021-01-12 00:00:00-05:00  125.947723  13.450613  207.755188  156.041504
...                               ...        ...         ...         ...
2022-01-03 00:00:00-05:00  179.076599  30.070988  326.287720  170.404495
2022-01-04 00:00:00-05:00  176.803818  29.241367  320.692810  167.522003
2022-01-05 00:00:00-05:00  172.100861  27.558167  308.382111  164.356995
2022-01-06 00:00:00-05:00  169.227921  28.131214  305.945312  163.253998
2022-01-07 00:00:00-05:00  169.395187  27.201761  306.101196  162.554001
```

Figure 2.2.1.A: Closing price table for selected stocks

The choice of a
one-year period strikes a balance between capturing long-term trends and avoiding excessive influence from short-term market volatility. By incorporating a time step of one day, the dataset includes sufficient observations to infer meaningful statistical measures while ensuring that short-term fluctuations do not disproportionately affect the results. This approach ensures that the data is robust and well-suited for subsequent portfolio optimization and analysis.

```python
# Function to fetch stock data from Yahoo Finance
def fetch_stock_data(tickers, start_date, end_date):
    closing_prices = {}
    for ticker in tickers:
        stock_data = yf.Ticker(ticker).history(start=start_date, end=end_date, interval='1d')
        if stock_data.empty:
            print(f"Stock with ticker symbol: {ticker} does not exist.")
            continue
        closing_prices[ticker] = stock_data['Close']
    return pd.DataFrame(closing_prices)
```

Figure 2.2.1.B: Querying YFinance API for Price Data

Given that our robo-advisor is generally intended for long-term buy and hold strategies, we feel that factoring in short term volatility is unimportant to the investment goals of our target clients and making sure that long term price volatility is adequately represented in our model is a higher priority.

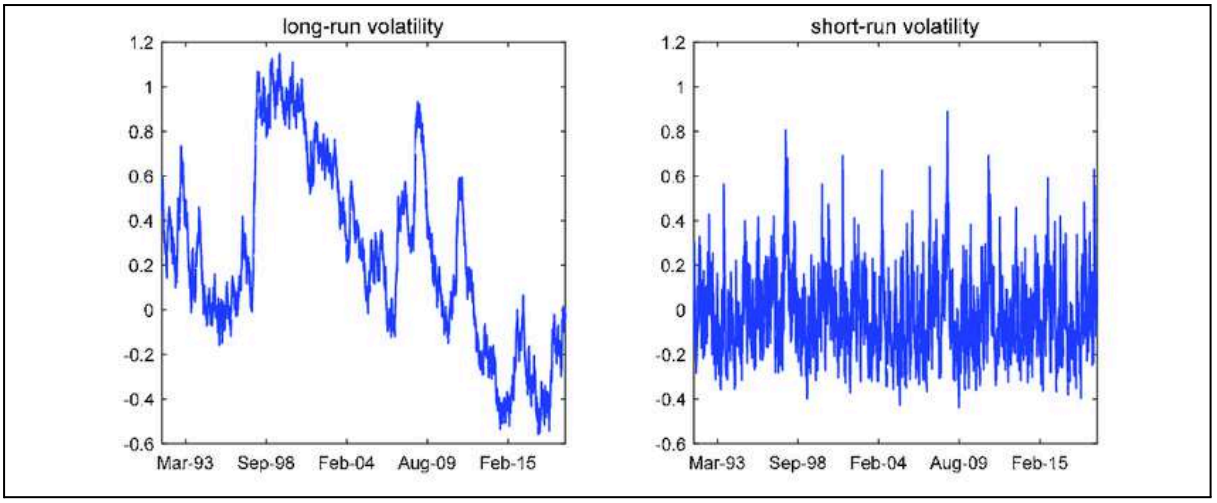

Figure 2.2.1.C: We aim to capture the long term volatility but disregard short term volatility

2.2.2. Statistical Analysis

For the statistical analysis, we first applied mean-variance analysis to calculate the mean return, standard deviation, and variance for each security specified by the user. These metrics were then displayed to the user, providing a clear frame of reference for the risk-return characteristics of their selected securities:

```python
# Function to calculate mean, standard deviation, and variance
def calculate_statistics(returns):
    if isinstance(returns, pd.Series):
        returns = returns.to_frame()
    mean_returns = returns.mean().values
    std_devs = returns.std().values
    variances = returns.var().values
    stats_df = pd.DataFrame({
        'Mean Return': mean_returns,
        'Std Dev': std_devs,
        'Variance': variances
    }, index=returns.columns)
    return stats_df
```

Figure 2.2.2.A: Code to calculatestatistics on selected stocks

```
Daily Statistics:
       Mean Return   Std Dev  Variance
aapl      0.001359  0.015730  0.000247
nvda      0.003438  0.028402  0.000807
msft      0.001665  0.013327  0.000178
amzn      0.000253  0.015118  0.000229
```

Figure 2.2.2.B: Statistics for selected stocks

Then, using the closing price dataframe, we then create a covariance matrix that is used in the algebra later on to generate the efficient frontier. The entries in the matrix consist of the computed covariance of every possible ordered pairing of n securities specified by the client, resulting in an n x n covariance matrix :

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \text{Cov}_{12} & \dots & \text{Cov}_{1n} \\ \text{Cov}_{21} & \sigma_2^2 & \dots & \text{Cov}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}_{n1} & \text{Cov}_{n2} & \dots & \sigma_n^2 \end{bmatrix}$$

Figure 2.2.2.C: Covariance Matrix

The portfolio variance (σ_P^2) is expressed as:

$$\sigma_P^2 = \mathbf{w}^\top \Sigma \mathbf{w}$$

```
portfolio_std_dev = np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 252, weights)))
results[1, i] = portfolio_std_dev
```

Figure 2.2.2.D: Variance as a function of weights and covariance, with corresponding code

2.2.3. Efficient Frontier Generation

The generation of the efficient frontier can be broken down into several steps. The first step involves generating a large set of random weight combinations for the securities included in the portfolio. In our implementation, we generated 10,000 random combinations of weights, though this number can be adjusted as needed to achieve the desired level of granularity.

```
# Function to calculate the efficient frontier
def calculate_efficient_frontier(returns, risk_free_rate=0.0, num_portfolios=10000):
    num_assets = returns.shape[1]
    results = np.zeros((3, num_portfolios))  # Portfolio returns, risks, and Sharpe ratios
    weights_record = []

    for i in range(num_portfolios):
        weights = np.random.random(num_assets)
        weights /= np.sum(weights)  # Normalize weights to sum to 1
        weights_record.append(weights)

        portfolio_return = np.sum(returns.mean() * weights) * 252  # Annualized returns
        results[0, i] = portfolio_return (Note that this section of code corresponds to the earlier figure x.x.x.x)

        portfolio_std_dev = np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 252, weights)))  # Annualized risk
        results[1, i] = portfolio_std_dev

        results[2, i] = (portfolio_return - risk_free_rate) / portfolio_std_dev  # Sharpe ratio

    return results, weights_record
```

Figure 2.2.3.A: Code for generating the efficient frontier

Each set of weights is then passed into the portfolio variance formula (as delineated in figure to calculate the associated variance and standard deviation. Subsequently, the expected return for each set of weights is also calculated. This process results in a set of coordinates for each portfolio, where the x-coordinate represents the standard deviation (risk) and the y-coordinate represents the expected return. These coordinates collectively form the efficient frontier, providing a visual representation of the risk-return trade-offs for the portfolio.

AAPL	NVDA	MSFT	AMZN	Portfolio Std Dev	Portfolio Mean Return
0.25	0.25	0.25	0.25	0.231	0.112
0.40	0.30	0.20	0.10	0.252	0.126
0.10	0.40	0.30	0.20	0.269	0.128
0.30	0.10	0.50	0.10	0.237	0.116
0.25	0.15	0.25	0.35	0.222	0.103

Figure 2.2.3.B: An excerpt of 5 random weightages with associated return and risk

With the set of calculated coordinates, we can proceed to plot these points and construct the final graph of the efficient frontier:

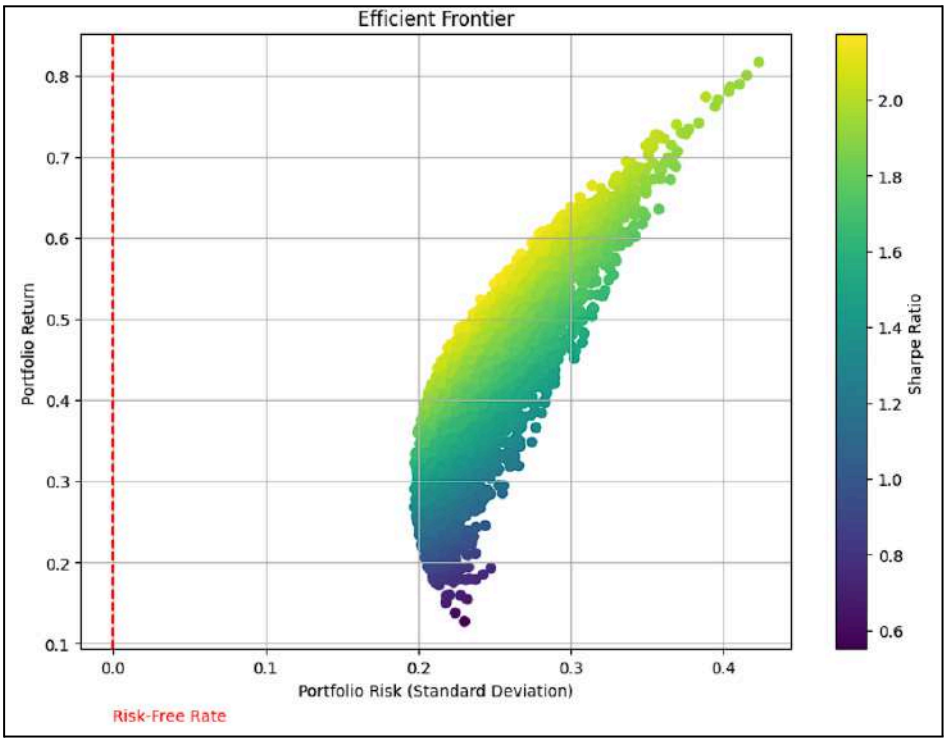

Figure 2.2.3.C: Final efficient frontier

In this example, the assets selected (NVDA, AAPL, MSFT, AMZN) and the 10,000 randomly generated weights provided sufficient diversification to produce a set of coordinates that neatly form a well-defined quadratic curve representing the efficient frontier.

However, it is important to note that this is not always the case. In some instances, the assets chosen may lack sufficient diversification, or the randomness of the generated weights may fail to provide adequate coverage of the portfolio space, resulting in a less obvious quadratic frontier shape. These limitations and their implications will be further examined in the Results section, specifically in Sections 3.1 and 3.2.

2.2.4. Weightage Calculation

To determine the optimal weighting for each security, we employed a mean-variance optimization approach based on modern portfolio theory. The objective function evaluates the expected return/standard deviation of the portfolio in relation to the weights assigned to each asset, which serve as the decision variables. Users can specify either a target risk (standard deviation) to maximize the expected return or a target return to minimize the corresponding risk. This design results in two distinct optimization problems based on the user's input.

Objective:
Minimize:

$$F(x_1, x_2, x_3, \ldots, x_n) = \sigma$$

Where:

- σ is the portfolio risk.
- $x_1, x_2, x_3, \ldots, x_n$ are the weightages of the assets.

Subject to Constraints:

1. $x_1 + x_2 + x_3 + \cdots + x_n = 1$ (The total weight allocation must equal 1).
2. $x_i \geq 0$ for all i (No negative weights, i.e., no short-selling allowed).

Figure 2.2.4.A: An excerpt of 5 random weightages with associated return and risk

Objective:
Maximize:

$$F(x_1, x_2, x_3, \ldots, x_n) = R$$

Where:

- R is the portfolio return.
- $x_1, x_2, x_3, \ldots, x_n$ are the weightages of the assets.

Subject to Constraints:

1. $x_1 + x_2 + x_3 + \cdots + x_n = 1$ (The total weight allocation must equal 1).
2. $x_i \geq 0$ for all i (No negative weights, i.e., no short-selling allowed).

Figure 2.2.4.B: An excerpt of 5 random weightages with associated return and risk

If the user specifies both a target risk and a target return, the model does not perform optimization, as no trade-offs between risk and return need to be made. In such cases, if the user's specified portfolio risk-return coordinate lies within the feasible region (defined by the

efficient frontier), the model simply assigns weights to achieve the requested, albeit likely inefficient, portfolio.

Conversely, if the specified risk-return combination falls outside the feasible region, the model informs the user that their requirements are infeasible given the selected securities. This ensures the solution remains transparent and aligned with the constraints of the chosen asset set.

The following code demonstrates the declaration of the objective function, taking a set of weights as input and the corresponding return or risk as the output:

```
def objective_function(weights):
    return portfolio_performance(weights)[0]  # Minimize risk
```

Figure 2.2.4.C: Code to define the objective function for maximizing return for a given risk

```
def objective_function(weights):
    return -portfolio_performance(weights)[1]  # Maximize return
```

Figure 2.2.4.D: Code to define the objective function for minimising risk for a given return

The following code sets constraints for the optimization problem:

1. The weightages must sum to 1 to ensure a fully allocated portfolio.
2. The weightages must satisfy the given variables (e.g., if risk is specified, the optimization maximizes return while maintaining the specified risk).
3. The weightages are restricted to values between 0 and 1, prohibiting short-selling or leverage.
4. The initial weightages are evenly distributed across all chosen securities before iteration begins.

```
constraints = [
  1 {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1},
  2 {'type': 'eq', 'fun': lambda weights: portfolio_performance(weights)[1] - target_return}
]

bounds = tuple((0, 1) for _ in range(num_assets)) 3
initial_weights = num_assets * [1. / num_assets,] 4
```

Figure 2.2.4.E: Code to define constraints

The optimization is performed using the minimize/maximize functions from the SciPy library, which then returns the optimal weights to achieve the highest return/lowest risk for the given risk/return respectively.

```
result = minimize(objective_function, initial_weights, bounds=bounds, constraints=constraints)
return result.x if result.success else None
```

Figure 2.2.4.F: Code to solve optimization problem

When run, the code will ask the user their desired risk/return, and output the optimal set of weights for their input:

```
Enter your desired portfolio risk (standard deviation): 0.2

Optimal Weights for Your Portfolio:
       Optimal Weight
aapl     1.498707e-01
nvda     3.628973e-16
msft     7.155081e-01
amzn     1.346212e-01

Sum of Optimal Weights: 1.0
The sum of the optimal weights equals 1.

Expected Portfolio Return: 0.3600765314014183
Portfolio Risk (Standard Deviation): 0.20000000000441226
```

Figure 2.2.4.D: Output weightage for maximised return for a given risk

```
Enter your desired portfolio return: 0.5

Optimal Weights for Minimum Risk at Desired Return:
       Optimal Weight
aapl         0.021003
nvda         0.183662
msft         0.795335
amzn         0.000000

Sum of Optimal Weights: 1.0

Expected Portfolio Return: 0.5000000014165735
Portfolio Risk (Standard Deviation): 0.23066199393992984
```

Figure 2.2.4.D: Output weightage to minimise risk for a given return

This is the final output that the client can further use to inform their investment decisions.

3. Results

3.1. Example (Microsoft & Tesla)

In this part of the report, we will be testing our model with actual stocks, with the first section only having two stocks, Microsoft and Tesla. We first define the date range, which for the two examples discussed in this part will be from January 2021 to January 2022. Next, for the stock tickers, we input the stocks for Microsoft and Tesla, in the format 'MSFT' and 'TSLA' respectively.

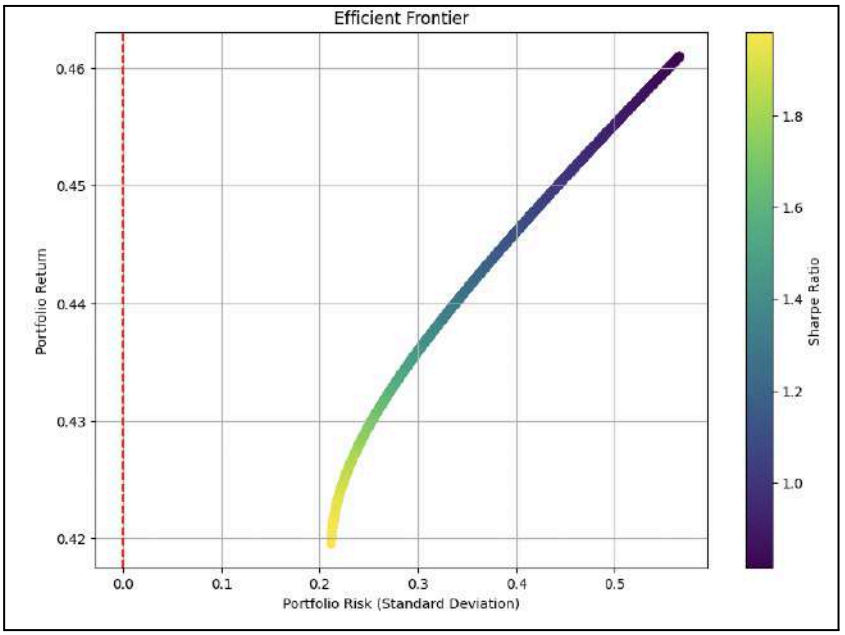

Figure 3.1.A: The following efficient frontier is generated for the Microsoft and Tesla stocks, with the x-axis representing the portfolio risk and y-axis representing the portfolio return.

```
Enter your desired portfolio risk (standard deviation): 0.3

Optimal Weights for Your Portfolio:
       Optimal Weight
msft         0.604849
tsla         0.395151

Sum of Optimal Weights: 1.0
The sum of the optimal weights equals 1.

Expected Portfolio Return: 0.43592434138456787
Portfolio Risk (Standard Deviation): 0.30000000003644284
Enter your desired portfolio return: 0.45

Optimal Weights for Minimum Risk at Desired Return:
       Optimal Weight
msft         0.265441
tsla         0.734559

Sum of Optimal Weights: 0.9999999999999999

Expected Portfolio Return: 0.4499999997600431
Portfolio Risk (Standard Deviation): 0.4418875848182166
```

Figure 3.1.B: For the given example, a desired portfolio risk of 0.3 resulted in optimal weights of 60% for Microsoft and 40% for Tesla, with an expected return of 43.6%. Similarly, for a desired portfolio return of 0.45, the optimal weights were 27% for Microsoft and 74% for Tesla, with a portfolio risk of 44%.

Now, why exactly does the efficient frontier look like this? Having less stocks has a lower diversability, whereas having more stocks has a wider range of risk and return, which is why the efficient frontier is rather narrow and restricted. Having a diverse array of stocks would result in a wider efficient frontier, which we will see in the next example.

3.2. Exemplar (8 Asset Portfolio)

In contrast to the first example, in this exemplar, we will model an efficient frontier using eight different assets, them being Duke Energy Corporation (‘DUK’), Exxon Mobil Corporation (‘XOM’), Chevron Corporation (‘CVX’), Procter & Gamble (‘PG’), Coca-Cola (‘KO’), Alibaba (‘BABA’), S&P 500 ETF (‘SPY’), Vanguard Total Stock Market Index Fund ETF (‘VTI’). The date range for this exemplar will be the exact same as it was for the first example; from January 2021 to January 2022. Next, for the stock tickers, we input the stocks for all eight assets, in their respective bracketed format.

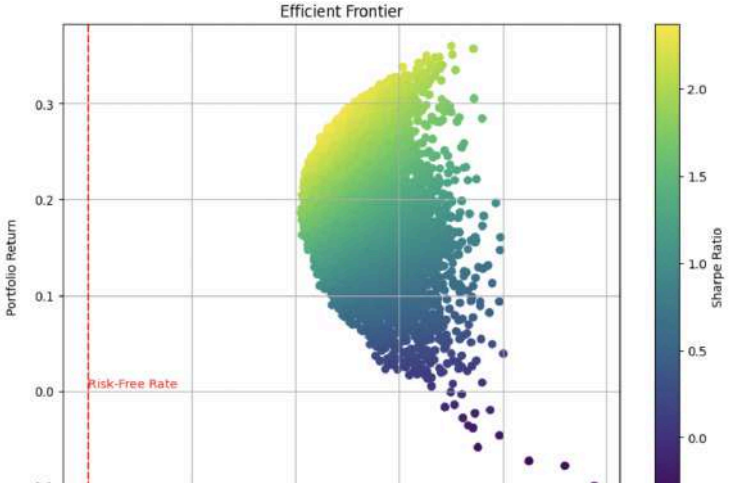

Figure 3.2.A: The following efficient frontier is generated for all eight assets with the x-axis representing the portfolio risk and y-axis representing the portfolio return. As can be seen, this efficient frontier is much wider than the one in the first example; this is because the plethora of assets is much higher and the risk-return characteristics of our choices are more diverse, spanning across multiple sectors and asset classes.

```
Enter your desired portfolio risk (standard deviation): 0.15

Optimal Weights for Your Portfolio:
       Optimal Weight
duk      2.331265e-01
xom      4.158198e-01
cvx      0.000000e+00
pg       7.514356e-02
ko       1.683989e-03
baba     5.529702e-16
spy      2.742261e-01
vti      0.000000e+00

Sum of Optimal Weights: 1.0000000000000007
The sum of the optimal weights equals 1.

Expected Portfolio Return: 0.34642871729964525
Portfolio Risk (Standard Deviation): 0.1500009443982721
Enter your desired portfolio return: 0.25

Optimal Weights for Minimum Risk at Desired Return:
       Optimal Weight
duk      2.325312e-01
xom      1.123774e-01
cvx      0.000000e+00
pg       1.882700e-01
ko       4.927237e-02
baba     0.000000e+00
spy      4.175490e-01
vti      1.019150e-17

Sum of Optimal Weights: 0.9999999999999999

Expected Portfolio Return: 0.24999999999961628
Portfolio Risk (Standard Deviation): 0.10692031408704979
```

Figure: 3.2.B: For the given example, a value of 0.15 was inputted for the desired portfolio risk, and 0.25 for the desired portfolio return. At a desired portfolio risk of 0.15, the optimal weights were allocated 23.3% for Duke Energy, 41.58% for Exxon Mobile, 7.51% for Proctor & Gamble, 0.17% for Coca-Cola, 0.00000055% for Alibaba, and 27.42% for S&P500 ETF. The expected portfolio return would be 34.64%. Now, moving on to the portfolio return, at a return of 0.25, optimal weights were allocated 23.25% for Duke Energy, 11.23% for Exxon Mobile, 18.82% for Proctor & Gamble, 4.93% for Coca-Cola, 41.75% for S&P500 ETF and 0.000000000001% for Vanguard Total Stock Market Index Fund ETF with a portfolio risk of 10.69%

4. Conclusion

4.1. Final Reflection & Discussions

To conclude our report, we will briefly reflect on the process of creating our robo-advising system, the implementation of different course content in the project, as well as what we believe we could have improved on if we were to redesign the system all over again.

Overall, while we were able to meet the primary objective we set for our model, there are numerous improvements we could've made, four in particular. Firstly, the effective frontier graph generated by our system was missing a capital market line, which is crucial in informing users of the linear relation between the expected return versus the risk of a chosen stock. As can be seen in the image below, 'x_1' is the point in the graph which is the minimum risk an individual can take; no risk below the value of x_1 would be a viable amount of risk available for the individual to take. What's more is that the shaded regions a_1 and a_2 are return amounts that the individual using the system can't attain; with a capital market line, it would be possible for users to achieve said returns present in the shaded regions. Additionally, elderly investors look to get returns at minimum risk rates; they take a safer approach as they are looking to retire soon, meaning that they would generally be opposed to opting for higher risk amounts. With the presence of a capital market line, elderly individuals are able to access a wider range of portfolios and risk-return combinations.

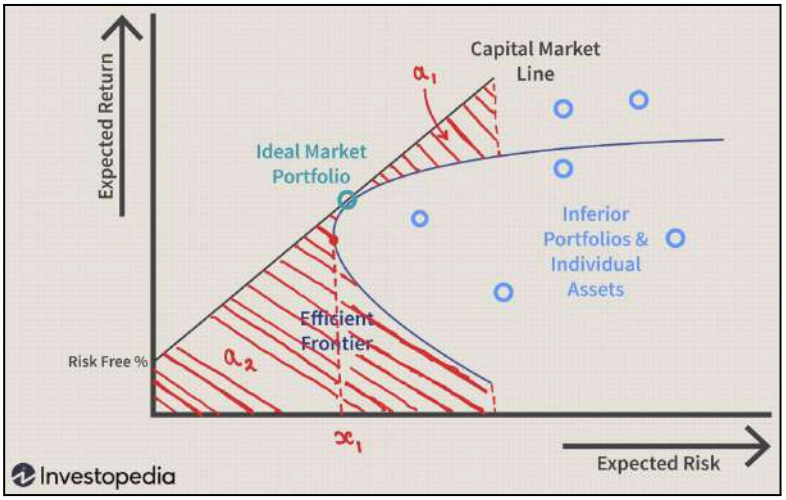

Figure 4.1.A: An excerpt of 5 random weightages with associated return and risk

Besides this, our group also believes that the weights displayed by the robo-advising system could be presented to the user in a more aesthetic and pleasant manner through the rounding of numbers instead of having a ton of values after the decimal. The weightage could even be presented as a percentage. Although the system does give extremely precise numerics with multiple decimal values, most brokers won't allow individuals to purchase, let's say '0.244243' units of a stock; dealings are done in share counts, which is an improvement we would like our system to implement. The system could use capital values inputted by users to calculate the exact number of shares to buy per security as opposed to just a fraction value.

Instead of manually choosing weights, another alternative approach we could have taken would be implementing general quadratic form, which would produce a theoretical line instead of plotting multiple points across the graph; this would help the client using the system better understand theoretical maximum portfolio return. Below is the formula for the general quadratic form, as well as an explanation of what the variables involved indicate.

$$\sigma_P^2 = aR_P^2 + bR_P + c$$

This formula indicates the relation between the portfolio variance and the return. The portfolio variance is indicated by 'σ_P^2', whereas the portfolio return is indicated by 'R_P'. Variable 'a' is essentially the quadratic dependence of the variance on return, variable 'b' is the linear dependence, variable 'c' is the constant term. The derivations of each of these three variables is given below.

$$a = (\mu^T \Sigma^{-1} \mu) - (1^T \Sigma^{-1} \mu)^2$$

$$b = -2(1^T \Sigma^{-1} \mu)$$

$$c = (1^T \Sigma^{-1} \mu)$$

Another key aspect of the project was to apply the course content into the actual system to exhibit thorough understanding of the material taught. Our project included multiple concepts taught throughout the entirety of the course; in this section, we'll discuss the main topics that our group implemented in our robo-advising system. Firstly, our system effectively produces an effective frontier to show optimal return amounts at different risk values, a concept that was explored in the course at an in depth level. Besides this, our system also calculated multiple statistics such as the standard deviation, mean return against variance, as well as the mean return and risk, all being concepts thoroughly discussed in the course.

5. Appendix

GPT Statement

The usage of ChatGPT in this report was limited to the creation of graphics (diagrams visually expressing the mathematical concepts discussed) and to assist in refining the articulation of points that were originally conceived entirely by us. ChatGPT's role was solely to aid in presenting our ideas in a more professional and polished tone.