Project 2 - Integer Programming

RM 294 - Optimization

Portfolio Selection - Tracking the NASDAQ-100 Index

Project Team:

Manasa Maganti: mm226524
Varsha Manju Jayakumar: vm26476
Utkarsh Garg: ug797
Medha Nalamada: mrn789

Contents

PROBLEM OVERVIEW

We are tasked to construct an index fund that tracks the NASDAQ-100 using a subset of stocks from the index. The goal is to build a new portfolio with 'm' number of stocks (such that m<n) that maximizes the similarity to the overall index while minimizing associated costs and complexity. This portfolio with m stocks closely tracks the performance of the NASDAQ-100 index by selecting a subset of stocks and determining their optimal weights. By doing so, we are trying to represent the market movements from technological and innovation-driven companies while ba;lancing the risk, return and sector diversity. Given a set of n stocks, the objective is to use integer programming to identify an optimal set of m stocks that can replicate the index's movement. This allows for a more focused investment strategy with reduced costs by narrowing down to a select group of securities that efficiently captures the essence of NASDAQ-100. Through this analysis, we can derive the best weightages for each selected stock, thereby creating a portfolio that aligns closely with index's performance while considering the constraints of the integer programming model.

METHODOLOGY

Exploratory Data Analysis and Cleaning

Data Overview:
The data used for this project includes two CSV files: the daily prices of the index and its component stocks for 2023 and 2024. All our calculations will be performed on 2023 data and later assessed against the returns from 2024 to substantiate our findings. Before beginning with stock selection, we will present a quick overview of the data. Here, we are calculating two moving averages—the 30-day and 90-day moving averages—of the NASDAQ-100 index. The 30-day moving average gives a closer look at shorter-term trends, while the 90-day moving average highlights longer-term trends. This helps identify whether the index was generally moving upward or downward throughout the year.

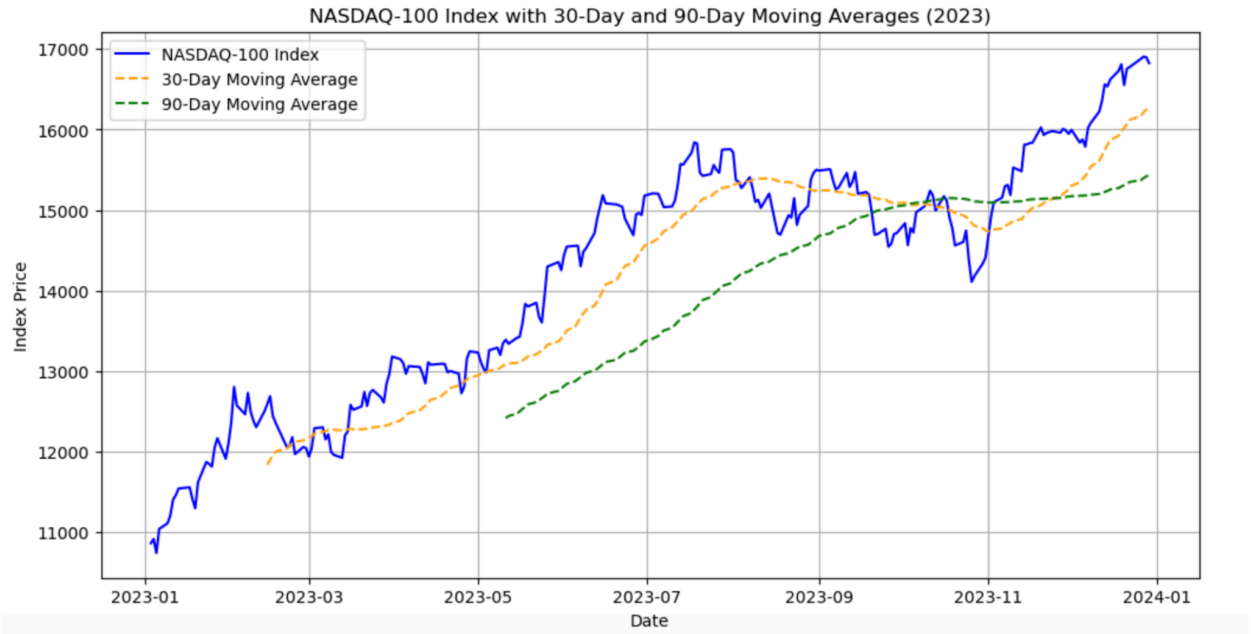


Figure 1: 2023 Index Prices with 30 & 90 Day Moving Averages

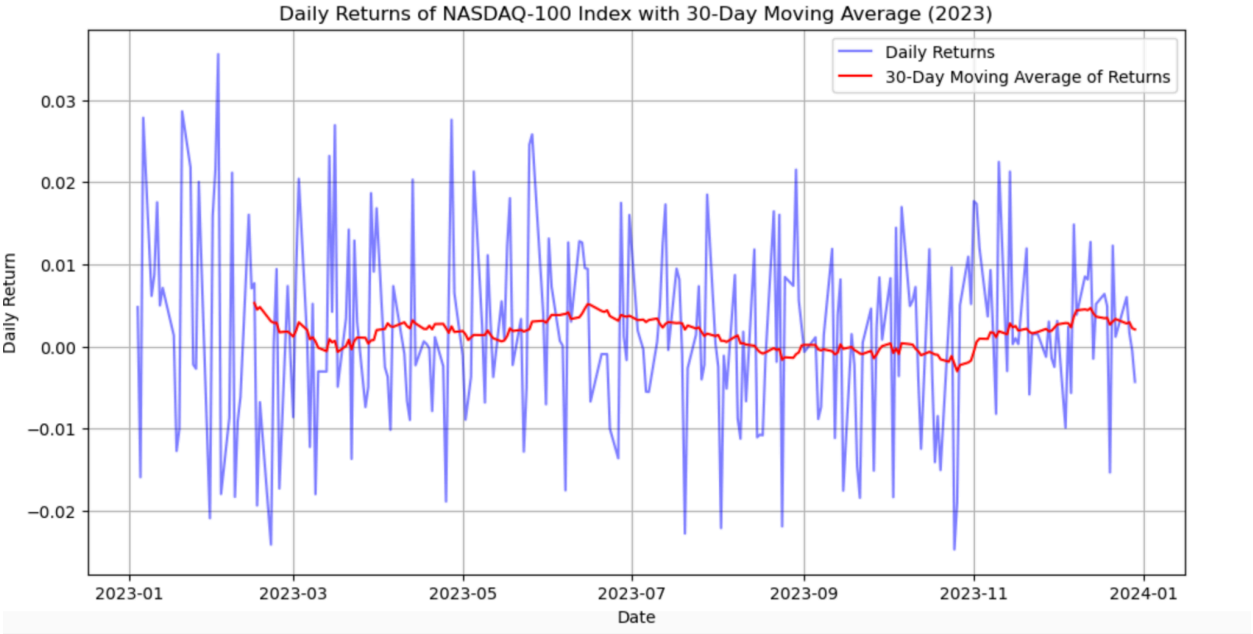


Figure 2: 2023 Daily Returns with 30 Day Moving Average

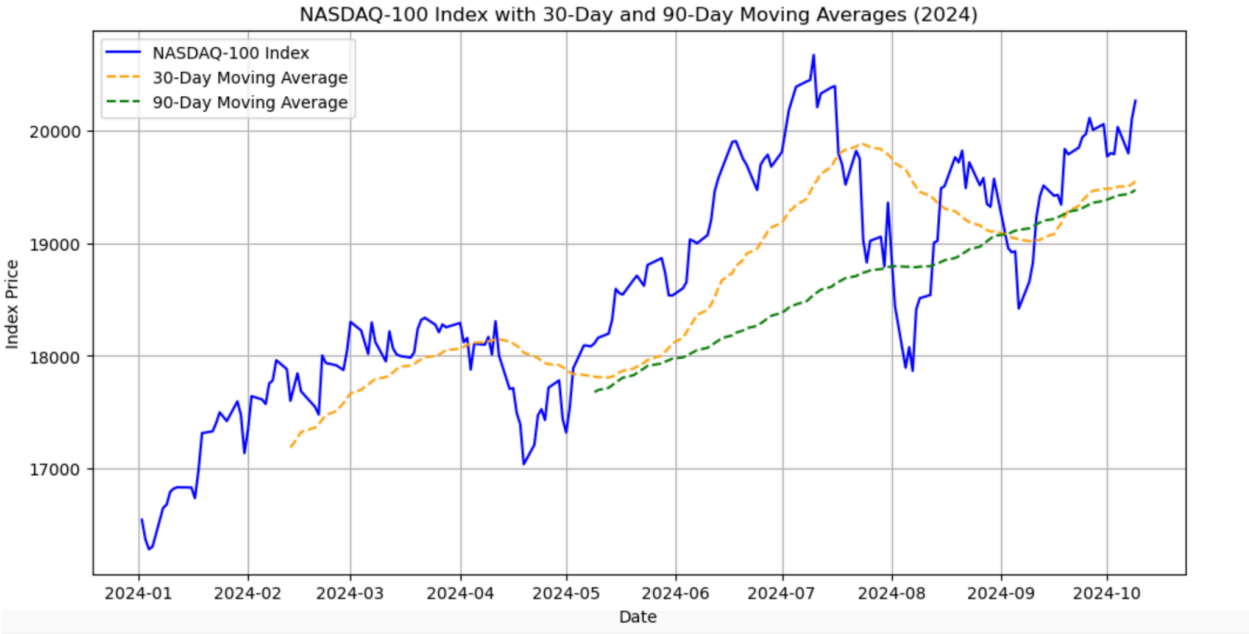


Figure 3: 2024 Index Prices with 30 & 90 Day Moving Averages

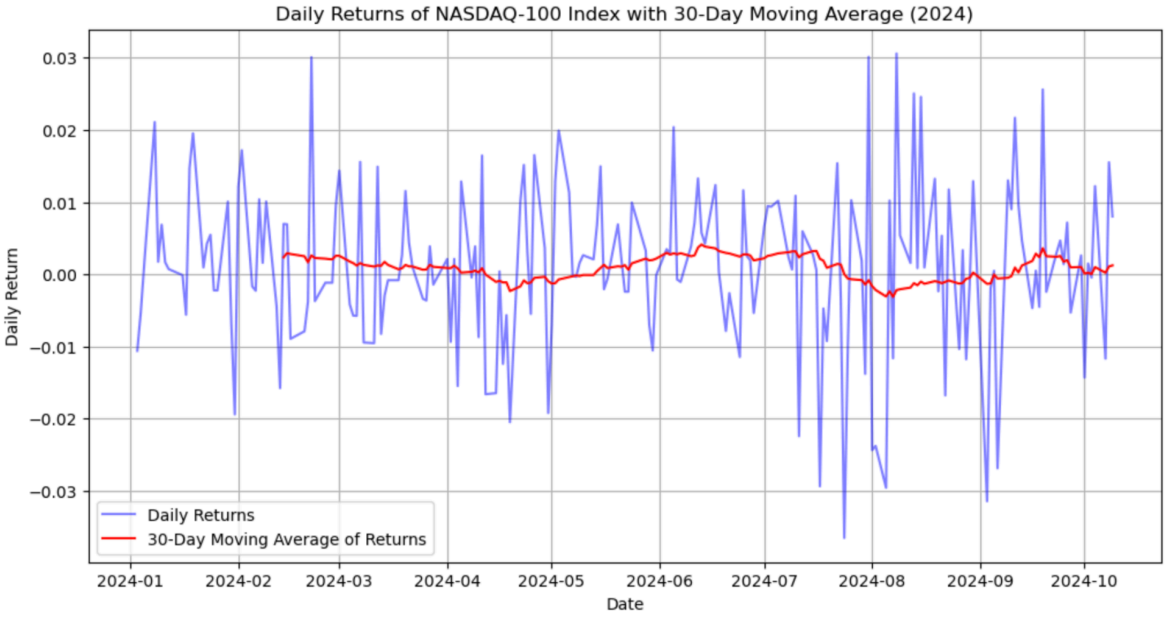


Figure 4: 2024 Daily Returns with 30 Day Moving Average

- In Figures 1&3, the blue line represents the daily price of the NASDAQ-100 index, while the orange and green dashed lines represent the 30-day and 90-day moving averages, respectively
- The prices in 2024 follow the same upward trajectory that they did in 2023, reaching a peak at the end of the year, but with higher volatility and more prominent fluctuations

- The 30-day moving average captures short-term movements more closely, revealing relatively stable growth in 2023, while 2024 displays more pronounced fluctuations, particularly from mid-year onwards, suggesting heightened market volatility (this is also indicated in Figures 2&4).
- The 90-day moving average in both years provides a smoother, long-term view, reflecting steady growth in 2023 and a delayed response to the increased volatility in 2024
- It is necessary to keep in mind that these differences may impact our analysis while testing on the 2024 data set, but we can take certain steps to account for such discrepancies

Data Preparation

Imputing the missing values:

In preparing the dataset for analysis, we addressed missing values using a two-step approach:

- First,the initial values in the 'ARM' column were imputed with 0 to reflect the stock fund's inception timeline (referenced from web sources). Since the ARM stock fund commenced after specific dates, earlier periods naturally show no holdings or activity. This approach ensures the dataset accurately represents the fund's unavailability during its initial phases, clearly distinguishing pre-launch from active periods.
- For all other columns, we applied linear interpolation to estimate missing values based on neighboring data points, ensuring continuity within the time series. After interpolation, we applied forward and backward filling as a final measure to address any remaining gaps, thereby minimizing any potential distortions due to missing data thereby maintaining continuity without introducing artificial volatility.

Daily Returns Calculation:

We calculate the daily returns for stocks through percentage change in stock's value over the given period as compared to absolute price difference. It is done so because:

- **Standardization Across Stocks**: Percentage change enables direct comparison across stocks with different prices. A $10 change varies in significance for a $20 vs. $500 stock, but a 5% change is consistent.
- **Reflects True Investment Return**: Investors focus on proportional returns. A 5% gain has the same meaning for low- and high-priced stocks, unlike absolute price changes.
- **Accounts for Price Scale**: Percentage change adjusts for stock price differences, preventing large-price stocks from skewing analysis.
- **Consistency Over Time**: Percentage change remains consistent, unaffected by splits or dividends, making trend analysis easier.
- **Portfolio Management**: Percentage change supports clear comparisons and balanced risk management, aligning with portfolio growth metrics.

Daily returns for each stock were calculated by measuring the percentage change in stock price from one day to the next. The formula used was:

$$\text{Daily Return}(t) = \frac{\text{Stock Price}(t) - \text{Stock Price}(t-1)}{\text{Stock Price}(t-1)} \times 100$$

Part 1 - Calculating the similarity through correlation matrix

Correlation captures how closely the returns of two stocks move together, which is essential for constructing a fund that mirrors the performance of a broader index with fewer stocks. The similarity matrix ρ is crucial for selecting the best stocks for your index fund. The elements ρij represent the similarity between stocks i and j, which can be effectively measured using the correlation of their returns.

In order to calculate the similarity matrix, we have experimented with the following approaches:

- **Correlation (CORR Function)**: Using the correlation function helps identify the linear relationships between the returns of different stocks, providing a measure of how similarly they move. Employing this method is ideal for constructing a similarity matrix that can guide stock selection, as it highlights pairs of stocks with strong co-movements
- **Cosine Similarity**: We have experimented with cosine similarity as it can offer insights into how aligned the returns of different stocks are, which can be beneficial for diversification in the portfolio
- **Euclidean Distance**: We have used Euclidean distance for assessing how distinct or dissimilar the stocks are from one another. This was helpful in identifying stocks with minimal overlap, supporting the selection of diverse assets for the portfolio

Approach	**Absolute error - train**	**Absolute error - test**
Correlation	1.110919990701496	1.143649610075574
Euclidean Distance	3.9972047147331837	3.6108158026361794
Cosine Similarity	1.210918887701345	1.1976743848236247

Among the three metrics, the correlation function was chosen for constructing the final similarity matrix because it yielded the least in-sample training error. This indicated that using correlation resulted in the most accurate representation of stock relationships, optimizing the performance of the constructed portfolio.

Below is the graph representing the top 10 positive and negative correlations between different stocks. This was depicted to help us in validating the results for various stocks selected as part of portfolio construction.

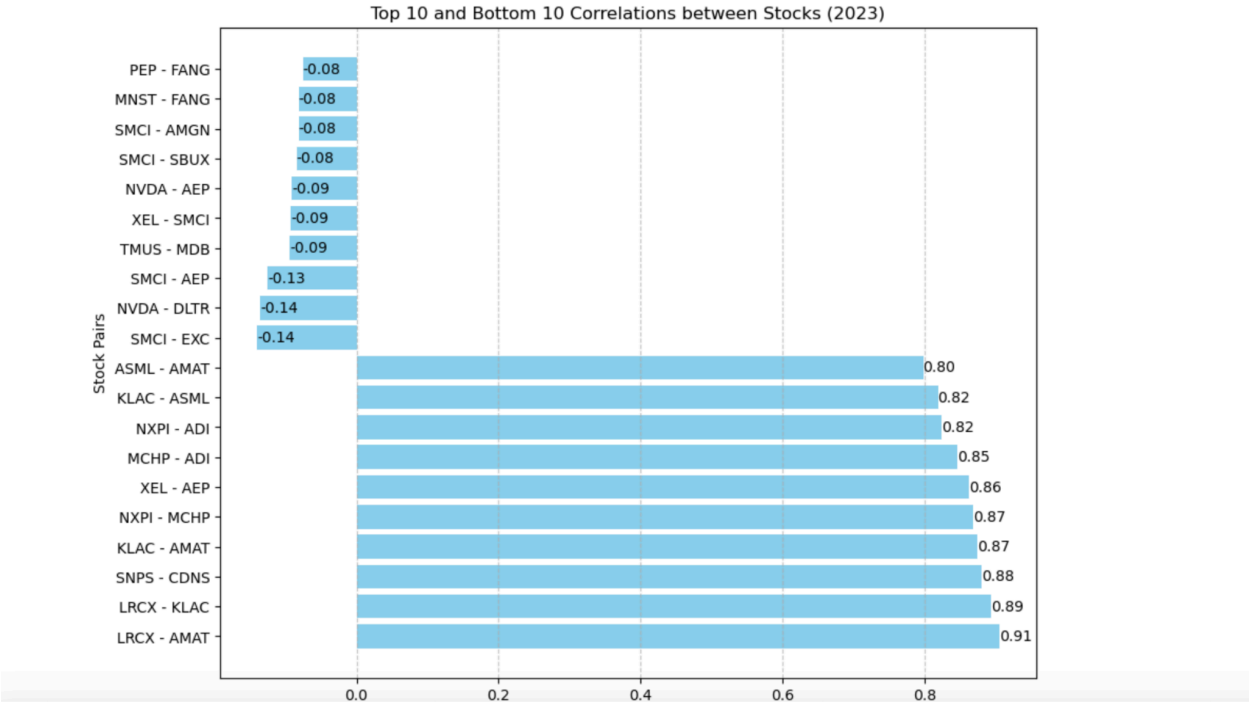


Figure 5: Correlation between Top & Bottom 10 stocks

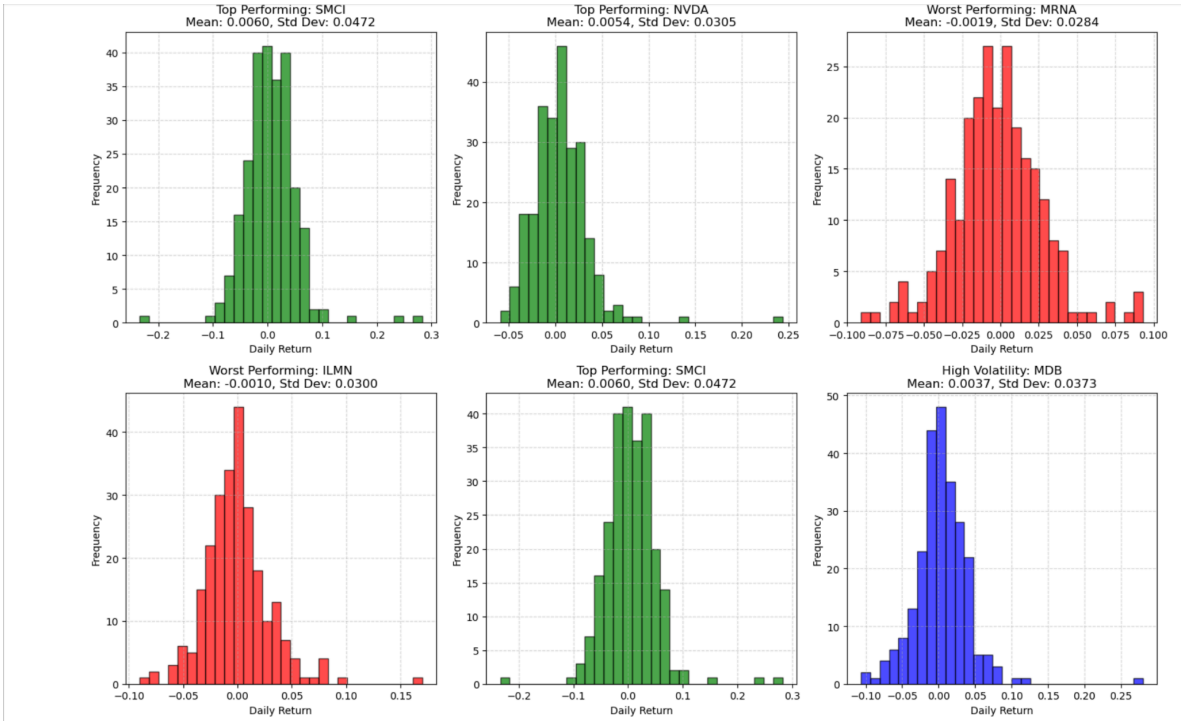


Figure 6: Daily returns of Top Vs Bottom vs Volatile Performing stocks

To analyze stock performance and volatility, we examined daily returns for the top two performers, worst two performers, and the two most volatile stocks, as shown in Figure 6. Each histogram displays the distribution of daily returns, color-coded as green for top performers, red for worst performers, and blue for high-volatility stocks. Most returns cluster around zero, indicating typical returns close to breakeven. Stocks with tightly concentrated returns, like the top performers, show lower day-to-day fluctuations, suggesting stability. In contrast, high-volatility stocks have broader distributions, showing greater variability. Some distributions also exhibit slight skewness, indicating occasional outliers with larger returns. Comparing these histograms provides insights into each stock's risk and return profile: stocks with wider distributions offer higher potential returns but carry more risk, while narrower distributions imply more consistent performance. This analysis helps identify stocks with strong, stable growth versus those with higher risk or underperformance.

Part 2 - Stock Selection

At the core of our project is a systematic approach to selecting stocks for constructing an index fund that effectively tracks the NASDAQ-100. We achieved this by formulating an integer programming model designed to optimize the selection of a limited number of stocks, denoted as m, for inclusion in the fund. This selection process relied on a similarity matrix, ρ, which quantifies relationships between stocks, serving as a critical foundation for identifying the most representative stocks.

Overview:

- Binary Decision Variables: We introduced two sets of binary decision variables. The binary matrix X, of size N x N times contains elements X(i,j) that indicate whether stock *j* in the index is the optimal representative for stock *i*. Additionally, we defined a binary vector Y, where each Y[i] denotes the inclusion or exclusion of stock i in the index fund.
- Objective Function: Our primary objective was to maximize the overall similarity between selected stocks and their representatives. We formulated this by creating an objective function that sums the correlation values between stocks, as represented in the similarity matrix ρ , weighted by the binary decision variables in X.

$$\max_{x,y} \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

- Constraints: Several constraints were applied to direct the stock selection process. The first constraint ensured that the total number of selected stocks, represented by the sum of Y[i] variables, equaled the target fund size, m. The second constraint required each

stock in the index to have exactly one representative in the fund. The third constraint stipulated that stock i could be represented by stock *j* only if *j* was included in the fund.

1. **We need to ensure there are m stocks in our subset:** $\sum_{j=1}^{n} y_j = m$
2. **Each stock i should only have one "best representative" stock j in the index:** $\sum_{j=1}^{n} x_{ij} = 1 \; for \; i = 1, 2, \ldots, n$
3. **We must guarantee that stock i is best represented by stock j only if j is in the fund:** $x_{ij} \leq y_j \; for \; i, j = 1, 2, \ldots, n$

- Optimization: With the model and constraints defined, we used Gurobi to perform the optimization, aiming to identify the ideal stock combination that maximized similarity while respecting the specified constraints.

```python
def select_stocks(m):
    #-----------------------------------------------------------------------
    ## STOCK SELECTION
    #-----------------------------------------------------------------------
    # Model
    model = gp.Model()
    model.ModelSense = GRB.MAXIMIZE
    select_pair = model.addVars(stock_list, stock_list, vtype=GRB.BINARY, name='select_pair')
    select_stock = model.addVars(stock_list, vtype=GRB.BINARY, name='select_stock')

    model.addConstr(sum(select_stock[i] for i in stock_list) == m, "stock_count")

    for i in stock_list:
        model.addConstr(sum(select_pair[i, j] for j in stock_list) == 1, "mapping")

    model.addConstrs((select_pair[i, j] <= select_stock[j] for i in stock_list for j in stock_list), "presence")

    model.setObjective(sum(correlation_dict[i][j] * select_pair[i, j] for i in stock_list for j in stock_list))
    model.Params.OutputFlag = 0
    model.optimize()

    selected_stocks = []
    for i in stock_list:
        for j in stock_list:
            if select_pair[i, j].X == 1 and j not in selected_stocks:
                selected_stocks.append(j)

    return model, selected_stocks
```

Code snippet for Stock Selection

- Output: The model produced various outputs, including the objective value (indicating the level of achieved similarity) and the values of the binary decision variables X and Y. These outputs specify which stocks were selected for the index fund and how they correspond to their representatives. Notably, the optimization identified the top 5 stocks—['SNPS', 'NXPI', 'INTU', 'PEP', 'HON']—as the most representative for the NASDAQ-100 index.

```
# Objective Value
model.objVal
```

```
51.44502840588646
```

```
selected_stocks
```

```
['SNPS', 'NXPI', 'INTU', 'PEP', 'HON']
```

Part 3 - Portfolio Weight Optimization

After finalizing the stock selection process, the next critical step was to determine the portfolio weights for the chosen stocks. Our primary goal was to minimize the discrepancy between the returns of the NASDAQ-100 index and the weighted sum of returns from the selected stocks—a classic tracking error minimization problem.

This optimization initially involved a non-linear approach due to the absolute value function. To address this challenge, we reformulated it as a linear program. The primary complexity lay in managing the absolute value function, which is inherently non-linear. To overcome this, we linearized the absolute value function by introducing auxiliary variables and corresponding constraints. This approach allowed us to efficiently calculate the optimal portfolio weights, ensuring the selected stocks closely track the NASDAQ-100 index.

Overview:

- Objective Function: Our primary goal was to minimize the tracking error between the index return and the fund return by reducing the sum of the 'D' elements representing discrepancies

$$\min_{w} \sum_{t=1}^{T} \left| q_t - \sum_{i=1}^{m} w_i r_{it} \right|$$

where q_t is the return of the index at time t, w_i is the weight of stock i in the portfolio, and r_{it} is the return of stock i from the subset. The constraints restrict the weights to above 0, meaning that no stocks in the portfolio can be shorted. Additionally, they must sum to 1 to represent a complete investment strategy

- Constraints: To ensure accurate optimization, several constraints were implemented: the sum of all weights add up to 1 each weight is greater than or equal to 0 and less than or equal to 1.
  Introduced constraints that made sure the difference between the index return and the fund return was greater than or equal to the absolute tracking error. These constraints maintained the tracking accuracy of the index fund

$$s.t. \sum_{i=1}^{m} w_i = 1$$

$$w_i \geq 0.$$

$$d_t \geq q_t - \sum_{i=1}^{m} w_i r_{it}$$

$$d_t \geq -(q_t - \sum_{i=1}^{m} w_i r_{it})$$

- Optimization: We solved this linear program in Gurobi, aiming to minimize the tracking error.

```python
def calculate_weights(m, selected_stocks):
    #-----------------------------------------------------------------------
    ## PORTFOLIO WEIGHTS
    #-----------------------------------------------------------------------
    weight_model = gp.Model()
    weight_model.ModelSense = GRB.MINIMIZE

    weights = weight_model.addVars(selected_stocks, vtype=GRB.CONTINUOUS, ub=[1] * len(selected_stocks), name='weights')
    abs_diff = weight_model.addVars(index_returns_train.index, vtype=GRB.CONTINUOUS, name='abs_diff')

    weight_model.addConstr(sum(weights[i] for i in selected_stocks) == 1, "sum_weights")  # weights sum to 1

    weight_model.addConstrs(
        (
            abs_diff[date] >= (
                index_returns_train[date] - sum(
                    weights[i] * returns_dict_train[i].get(date, 0)  # Using get to handle missing dates
                    for i in selected_stocks
                )
            ) for date in index_returns_train.index
        ), "abs_constraint_1"
    )

    weight_model.addConstrs(
        (
            abs_diff[date] >= -(
                index_returns_train[date] - sum(
                    weights[i] * returns_dict_train[i].get(date, 0)  # Using get to handle missing dates
                    for i in selected_stocks
                )
            ) for date in index_returns_train.index
        ), "abs_constraint_2"
    )

    weight_model.setObjective(sum(abs_diff[date] for date in index_returns_train.index))

    time_limit = 60  # Define time limit in seconds
    weight_model.Params.OutputFlag = 0
    weight_model.Params.TIME_LIMIT = time_limit

    weight_model.optimize()

    # 2023 (In-sample)
    returns_train_portfolio = pd.Series(index=index_returns_train.index, dtype=float)
    for date in index_returns_train.index:
        returns_train_portfolio.loc[date] = sum(weights[i].X * returns_dict_train[i].get(date, 0) for i in selected_stocks)

    # 2024 (Out-of-sample)
    returns_test_portfolio = pd.Series(index=index_returns_test.index, dtype=float)
    for date in index_returns_test.index:
        returns_test_portfolio.loc[date] = sum(weights[i].X * returns_dict_test[i].get(date, 0) for i in selected_stocks)

    weights_df = pd.DataFrame(index=stock_list)
    weights_df[f'm_{m}'] = np.nan
    for i in selected_stocks:
        weights_df[f'm_{m}'].loc[i] = weights[i].X
    weights_df = weights_df[weights_df[f'm_{m}'].notna()]

    # Evaluation
    abs_error_train = weight_model.objVal
    abs_error_test = sum(abs(index_returns_test - returns_test_portfolio))

    return weight_model, weights_df, abs_error_train, abs_error_test

# Running Portfolio Weights Calculation
weight_model, weights_df, abs_error_train, abs_error_test = calculate_weights(5, selected_stocks)
```

Code snippet for Portfolio Weight Calculation

- Output: The code generated key outputs, including the minimized tracking error (objective value) and the optimal portfolio weights for the selected stocks.

	m_5
HON	0.164023
INTU	0.224791
NXPI	0.177157
PEP	0.189129
SNPS	0.244900

The below table represents the absolute error(obj_val) and RMSE for train and test data with increasing values of m number of stocks. As we can see, the error reduces over the period of time with an increase in the number of stocks.

#stocks	obj_val_corr	obj_val_train	rmse_train	obj_val_test	rmse_test
5	51.445028	1.110920	0.006210	1.143650	0.007810
10	57.334558	0.923451	0.005032	0.866268	0.006135
20	65.187926	0.859755	0.004712	0.765656	0.005635
30	71.484965	0.744583	0.004422	0.714833	0.005268
40	77.237471	0.637388	0.003714	0.722149	0.005313
50	82.643246	0.528155	0.003155	0.688768	0.004876
60	87.473282	0.496423	0.003048	0.672020	0.004868
70	91.730926	0.344847	0.002313	0.604592	0.004473
80	95.478613	0.216453	0.001846	0.497925	0.003966
90	98.383982	0.161659	0.001636	0.424246	0.003641
100	100.000000	0.151559	0.001646	0.404956	0.003571

Part 4 - Mixed-Integer Programming - Big M Optimization Technique

Instead of performing the stock selection and weight calculation in different steps, an alternative approach to the portfolio weight selection problem is to reformulate it as a Mixed-Integer Program (MIP) that allows control over the number of non-zero weights. Instead of optimizing weights only for pre-selected stocks, we can optimize across all available stocks by introducing binary variables to indicate whether each stock's weight w_i is non-zero. This will reformulate the problem as:

$$min_w \sum_{t=1}^{T} \left| q_t - \sum_{i=1}^{n} w_i r_{it} \right|$$ (now includes all n stocks instead of the m subset)

However, we will also have to add additional constraints to account for all our requirements. In addition to our original weight constraints (positive, sum to 1), we will now include:

$$\sum_{i=1}^{n} y_i = m$$ **(stock selection constraint)**

$$w_i \leq M \bullet y_i \, for \, each \, stock \, i$$ **(big M constraint)**

The big M constraint ensures that w_i is 0 if y_i if 0 (the stock is not selected).

```
# Create DataFrames to store returns and weights for different portfolio sizes
returns_train_2 = pd.DataFrame({date_column: train_data[date_column], index_column: index_returns_train})
returns_test_2 = pd.DataFrame({date_column: test_data[date_column], index_column: index_returns_test})
weights_2 = pd.DataFrame(index=correlation_matrix.index)
k = 0
bigM = 1

# Define the list of m values based on the number of stocks
lst_ms = [5] + [*range(10, num_stocks, 10)] + [num_stocks]

for m in lst_ms:
    start = time.time()

    # Model
    weight_model = gp.Model()
    weight_model.ModelSense = GRB.MINIMIZE

    # Decision Variables
    weights = weight_model.addVars(stock_list, vtype=GRB.CONTINUOUS, ub=[1] * len(stock_list), name='weights')
    selected = weight_model.addVars(stock_list, vtype=GRB.BINARY, name='selected')
    abs_diff = weight_model.addVars(date_range_train, vtype=GRB.CONTINUOUS, name='abs_diff')

    # Constraints
    weight_model.addConstr(sum(weights[i] for i in stock_list) == 1, "sum_weights")
    weight_model.addConstrs(
        (abs_diff[j] >= (index_returns_train[j] - sum(weights[i] * returns_dict_train[i][j] for i in stock_list)) fo
        "abs_value_1"
    )
    weight_model.addConstrs(
        (abs_diff[j] >= -(index_returns_train[j] - sum(weights[i] * returns_dict_train[i][j] for i in stock_list)) f
        "abs_value_2"
    )
    weight_model.addConstrs((weights[i] <= selected[i] * bigM for i in stock_list), "force_w0_y0")
    weight_model.addConstr(sum(selected[i] for i in stock_list) == m, "portfolio_size")

    # Objective Function
    weight_model.setObjective(sum(abs_diff[j] for j in date_range_train))

    # Set model parameters
    weight_model.Params.OutputFlag = 0
    weight_model.Params.TIME_LIMIT = 3600

    # Solve
    weight_model.optimize()
```

Code Excerpt from Big M Technique

The below table represents the absolute error(obj_val) and RMSE for train and test data with increasing values of m number of stocks using the Big-M optimization technique

#stocks	obj_val_train	rmse_train	obj_val_test	rmse_test
5	0.695614	0.003821	0.797546	0.005583
10	0.405680	0.002561	0.654864	0.004700
20	0.248537	0.001818	0.550729	0.004145
30	0.195504	0.001674	0.427863	0.003648
40	0.170068	0.001627	0.405442	0.003551
50	0.158784	0.001616	0.404606	0.003561
60	0.154650	0.001632	0.402224	0.003544
70	0.151988	0.001643	0.400432	0.003571
77	0.151565	0.001646	0.404294	0.003575
78	0.151559	0.001649	0.404962	0.003580
78	0.151559	0.001649	0.404956	0.003580

FINDINGS AND INSIGHTS

Results from Linear Integer Programming Method

After performing the optimization, our results for choosing m = 5 resulted in the following stocks being picked with these respective weightages:

```
Selected Stocks for m=5: ['SNPS', 'NXPI', 'INTU', 'PEP', 'HON']
Weights for m=5:
  SNPS: 0.2449
  NXPI: 0.1772
  INTU: 0.2248
  PEP: 0.1891
  HON: 0.1640
```

The stocks chosen here show how our optimization prioritized diversification and reduction of risk to ensure that the index is closely replicated. Our index fund allocates larger weights to companies like Synopsys, Intuit, and NXP Semiconductors, leaning towards high-growth, innovative tech segments, aligning with NASDAQ-100's core characteristics. We have also increased the number of stocks in the portfolio to validate our error metric.
Below are the graphs to compare the absolute error metric and the RMSE between in-sample and out-of-sample data

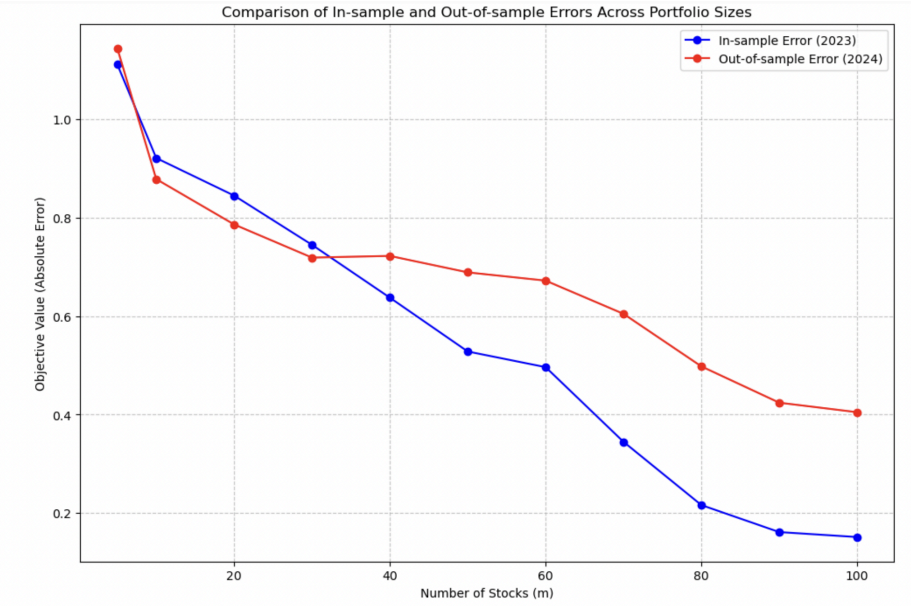


Figure 7: Comparing In-sample and out-of-sample errors across different values of m

- This graph shows how the error between our portfolio's returns and the NASDAQ-100 index changes as we increase the number of stocks in the portfolio.

- Both the in-sample error (2023) and out-of-sample error (2024) decrease significantly as the portfolio grows from 5 to about 100 stocks.
- This trend suggests that adding more stocks improves the portfolio's ability to track the index more accurately. However, beyond 30 stocks, the out-of-sample error tends to increase while the in-sample error continues to diminish.
- This finding suggests that a portfolio size of around 30-40 stocks is optimal for balancing tracking accuracy and portfolio complexity.

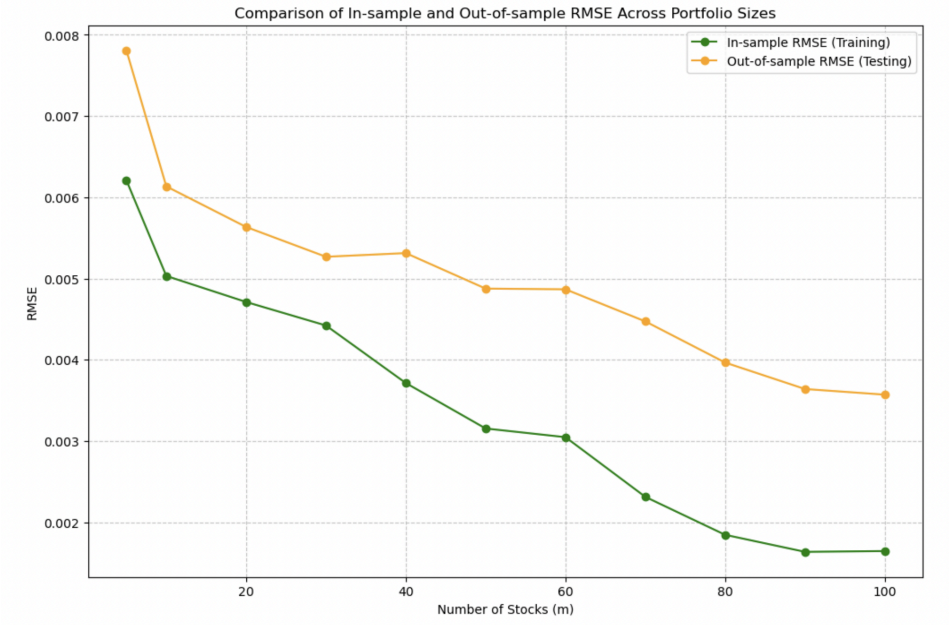
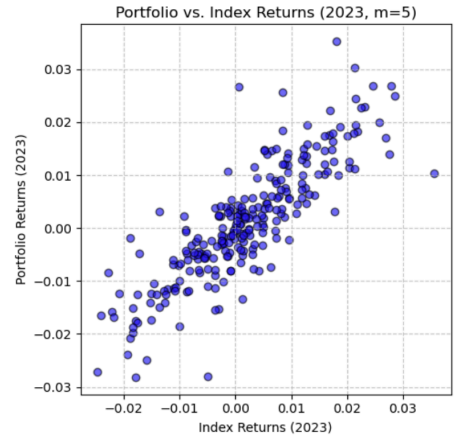
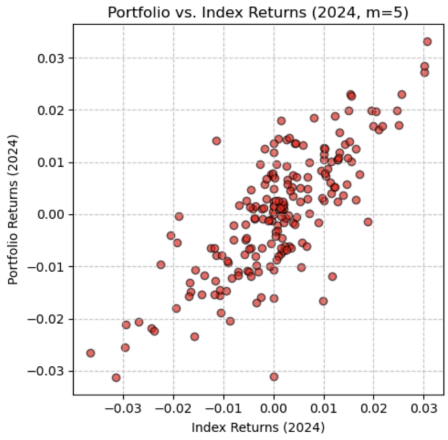
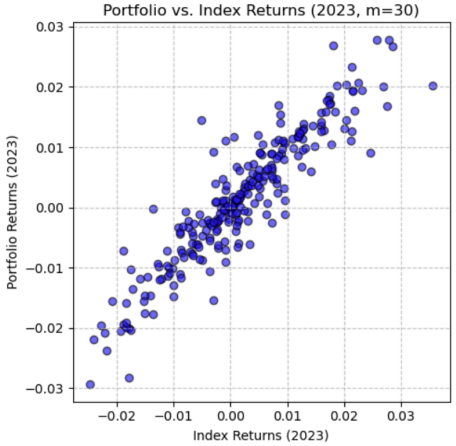


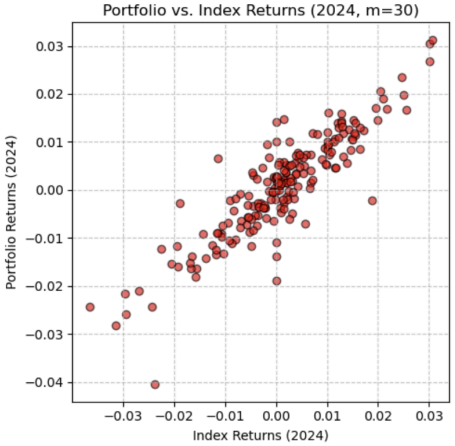
Figure 8: Comparing In-sample and out-of-sample RMSE across different values of m

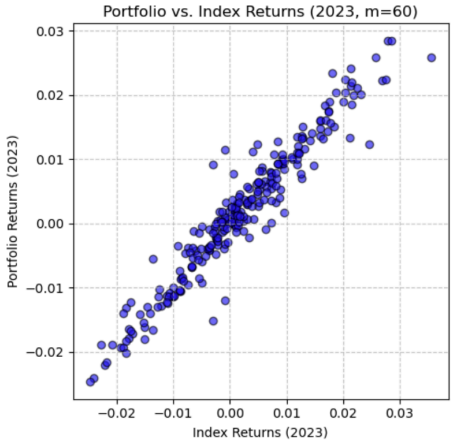
- The RMSE, which measures the average difference between our portfolio and the index returns, decreases as more stocks are added to the portfolio.
- The in-sample RMSE for 2023 is consistently lower than the out-of-sample RMSE for 2024, showing that the portfolio performs slightly better on the training data than on new data.
- This decline in RMSE becomes less significant after around 40–60 stocks, indicating that this range offers a reasonable trade-off between tracking accuracy and simplicity.
- Based on this, a portfolio with 40–60 stocks provides a strong balance between closely replicating the index and avoiding unnecessary complexity.

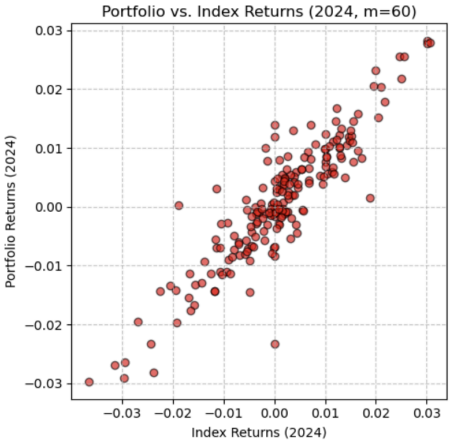

Portfolio vs. Index Returns (2023, m=5)
Portfolio Returns (2023)
Index Returns (2023)


Portfolio vs. Index Returns (2024, m=5)
Portfolio Returns (2024)
Index Returns (2024)


Portfolio vs. Index Returns (2023, m=30)
Portfolio Returns (2023)
Index Returns (2023)


Portfolio vs. Index Returns (2024, m=30)
Portfolio Returns (2024)
Index Returns (2024)


Portfolio vs. Index Returns (2023, m=60)
Portfolio Returns (2023)
Index Returns (2023)


Portfolio vs. Index Returns (2024, m=60)
Portfolio Returns (2024)
Index Returns (2024)

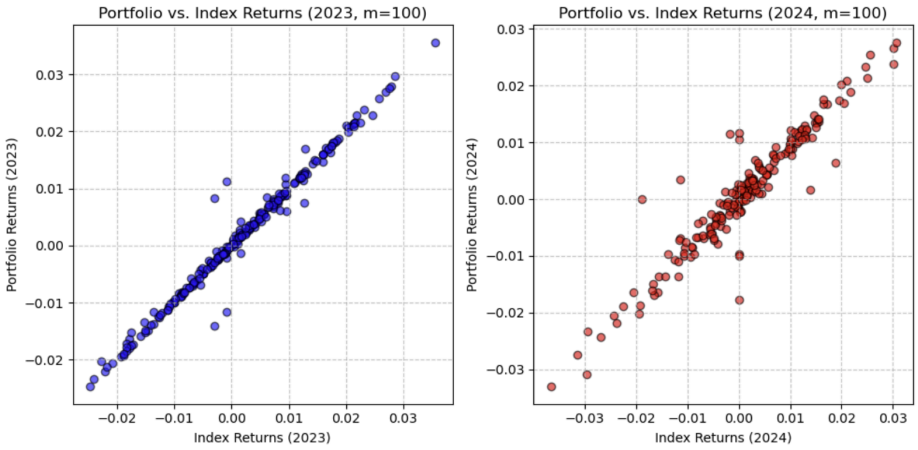


Figure 9: Examining linearity in model outputs as we increase the m values (m=5 m= 30, m = 60 and m=100)

- This graph shows the relationship between portfolio returns and index returns for different years (2023 and 2024) and different portfolio sizes (m = 5, 30, 60, 100).
- As the portfolio size (m) increases, the scatter plots show a stronger linear relationship between portfolio and index returns, indicating that larger portfolios align more closely with index performance.
- This suggests that as m grows, portfolio returns become more predictable and more correlated with the index returns.

Results from the Big M Technique

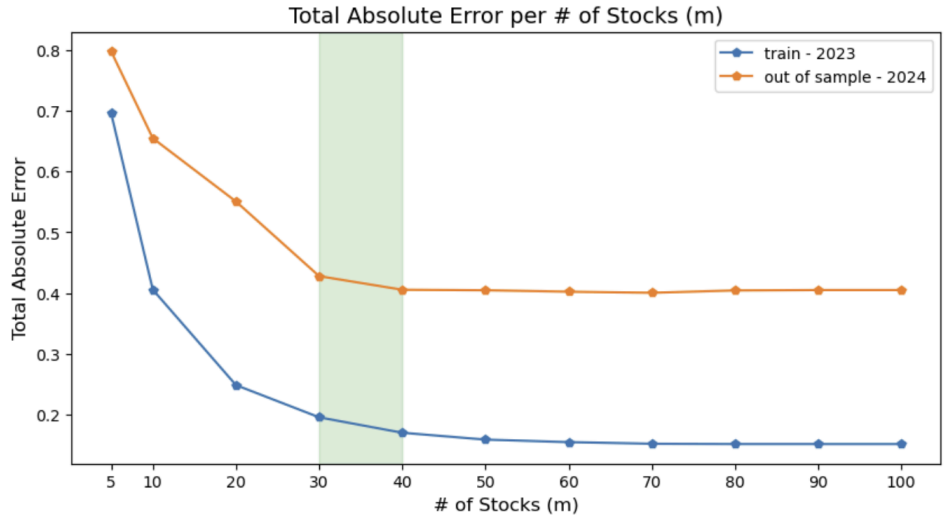


Figure 10: Line graph representing the total absolute error for different values of m using Big-M technique

- This graph shows the total absolute error for portfolios with different numbers of stocks (m) for both training data (2023) and out-of-sample data (2024).
- The training error (blue line) decreases significantly as the portfolio size increases, flattening out after around 30-40 stocks.
- The out-of-sample error (orange line) also decreases with more stocks but stabilizes beyond 30 stocks, indicating that adding more stocks doesn't lead to significant improvements in tracking accuracy

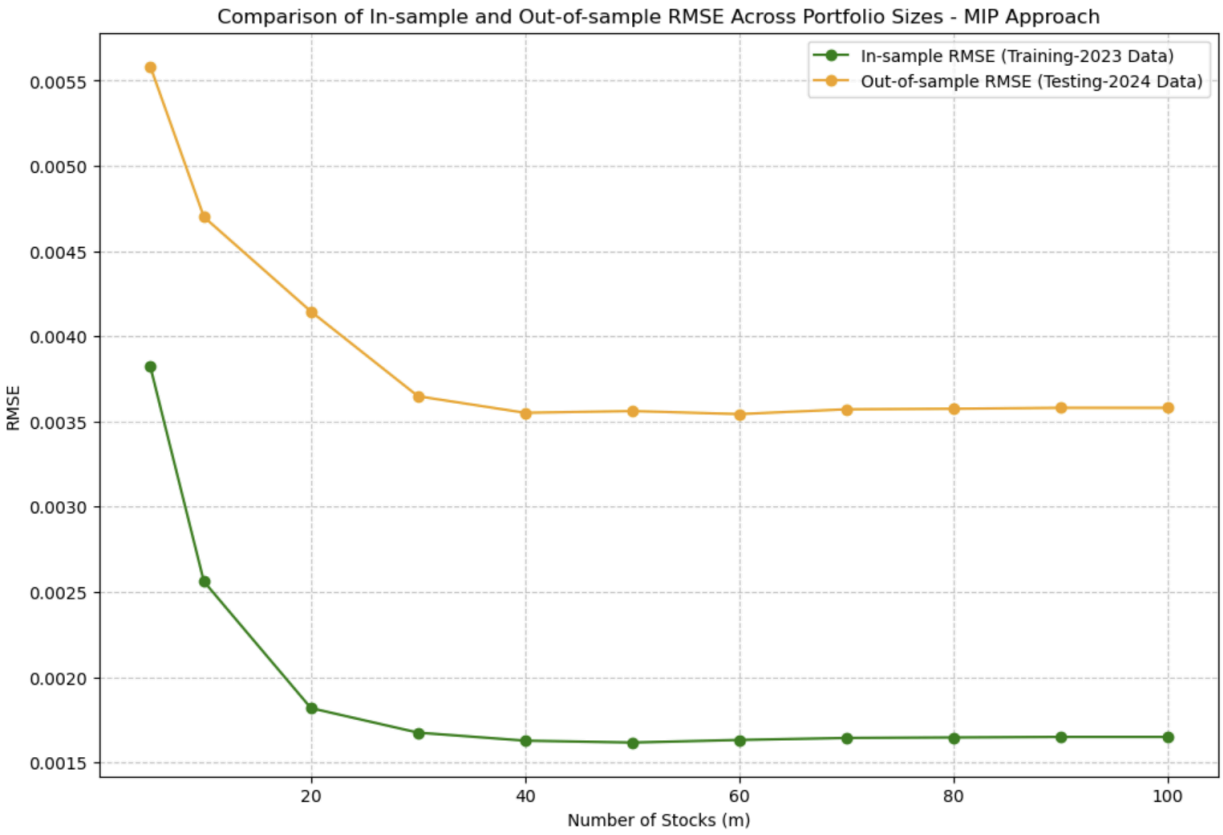
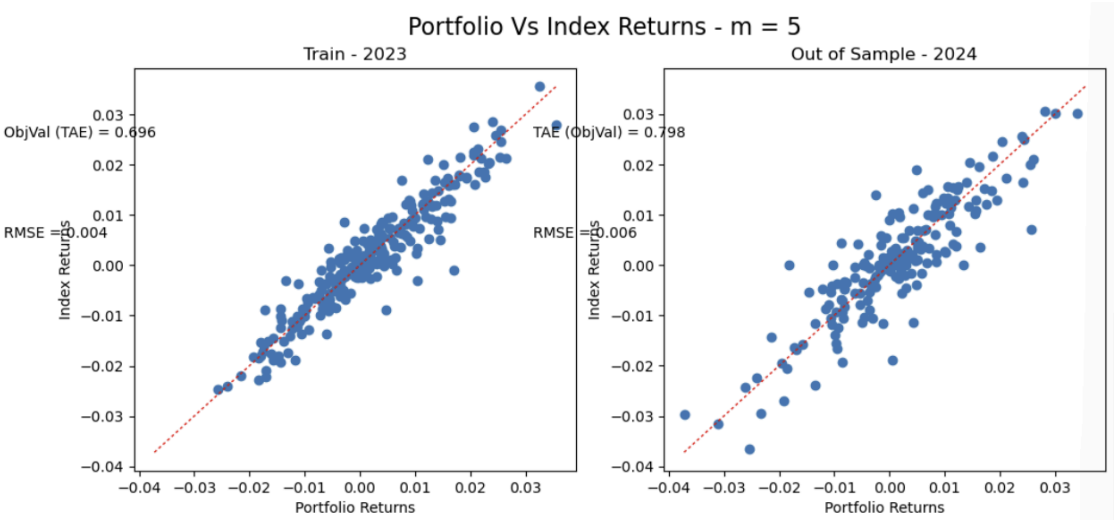
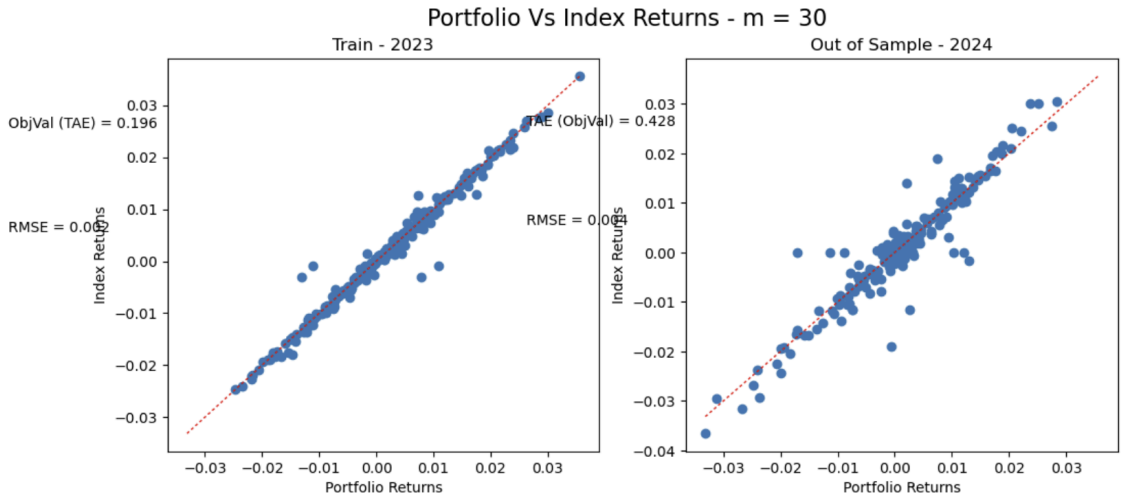
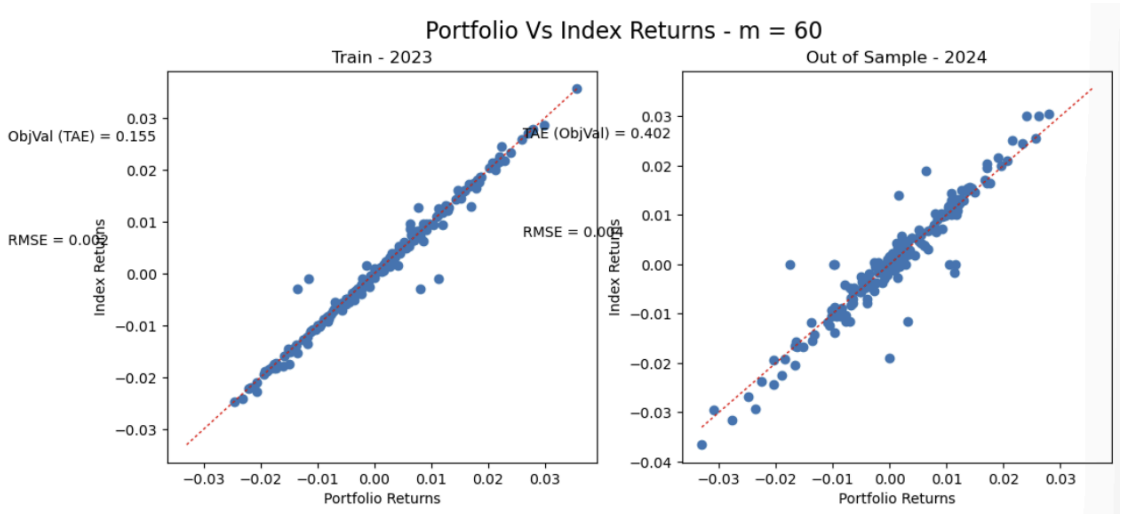


Figure 11: Line graph representing the RMSE for different values of m using Big-M technique

- The graph shows that when we increase the number of stocks from 5 to around 30, both training and testing RMSE improve a lot, which means the portfolio tracks the index much better as we add more stocks.
- After about 30-40 stocks, the out-of-sample RMSE (orange line) stops improving significantly, even though the training RMSE (green line) keeps dropping slightly. This indicates that adding more stocks doesn't help much with performance on new data.


Portfolio Vs Index Returns - m = 5
Train - 2023
Out of Sample - 2024
ObjVal (TAE) = 0.696
RMSE = 0.004
TAE (ObjVal) = 0.798
RMSE = 0.006
Index Returns
Portfolio Returns


Portfolio Vs Index Returns - m = 30
Train - 2023
Out of Sample - 2024
ObjVal (TAE) = 0.196
RMSE = 0.002
TAE (ObjVal) = 0.428
RMSE = 0.004
Index Returns
Portfolio Returns


Portfolio Vs Index Returns - m = 60
Train - 2023
Out of Sample - 2024
ObjVal (TAE) = 0.155
RMSE = 0.002
TAE (ObjVal) = 0.402
RMSE = 0.003
Index Returns
Portfolio Returns

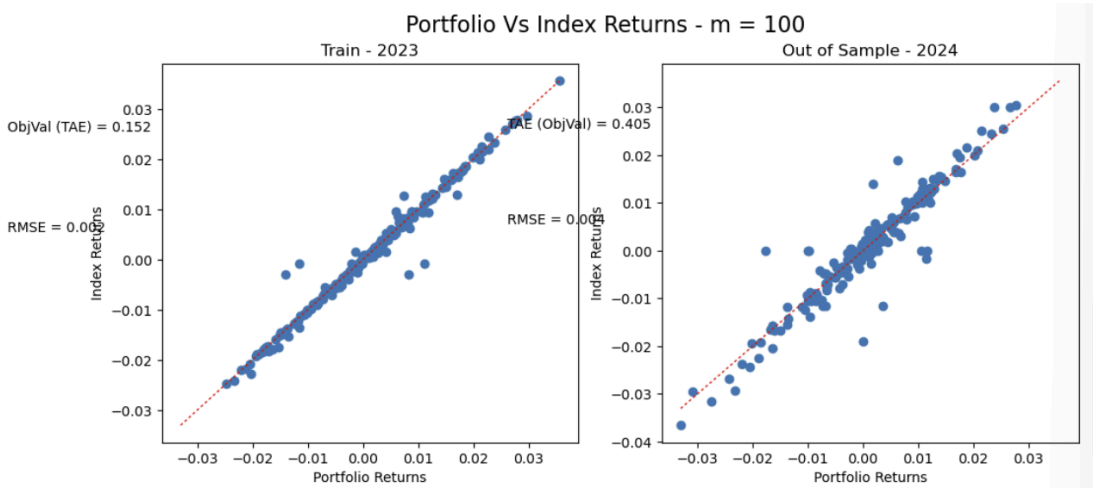


Figure 12: RMSE scores vs the portfolio returns for m=5, m=30, m=60 and m=100 using Big-M technique

- This plot presents the tracking performance of a portfolios of m=5,30,60,100, along with key error metrics: RMSE and Total Absolute Error (TAE) using the big M technique.
- The high RMSE and TAE values reflect considerable tracking errors, as seen in the wide scatter of points away from the ideal line (y = x). The performance worsens under the 2024 data.
- This could be due to the high volatility seen in the index prices from 2024 that the model was not able to predict accurately for.
- A larger portfolio is essential to lower these error values and improve tracking accuracy

Key Challenges

- Managing linear and non-linear constraints: Managing non-linearity in portfolio weights was tough because the initial optimization had non-linear constraints, like absolute value functions, which required us to design the constraints accurately.
- Balancing portfolio size and accuracy: Adding more stocks helped the in-sample performance, but it didn't always lead to better results for out-of-sample data. This sometimes pointed to overfitting, where the model works well on training data but not as much on new data.
- The trade-off between complexity and computation: Using the integer programming model with different constraints took a lot of computing power, especially when we had larger portfolio sizes. This made the analysis slower and more demanding

RECOMMENDATIONS

1. **Using Single-Stage MILP for All Portfolios**: We find that the Single-Stage MILP approach, where we handle stock selection and weighting in one step, is a strong choice for any portfolio size. It helps **cut down tracking error efficiently and avoids the diminishing returns** we see with adding too many stocks.Plus, it reduces initial costs and rebalancing needs, making it a practical solution for both small and large portfolios
2. **Selecting the Optimal Portfolio Size:** Based on our findings, **a portfolio of around 30–40 stocks strikes an ideal balance.** This range provides strong tracking of the NASDAQ-100 index while keeping the portfolio manageable, reducing both RMSE and absolute error without unnecessary complexity. Adding more stocks beyond this range yields diminishing returns in terms of tracking accuracy, making 30–40 stocks a practical and effective choice. Adopting this range aligns with the Single-Stage MILP Optimization approach, allowing us to reduce tracking error efficiently with fewer component stocks
3. **Building for Market Volatility:** Given recent market swings, **we recommend adding features like dynamic weighting or volatility metrics.** These adjustments will help our model stay accurate even in rocky markets, adapting to changes and ensuring steady performance.
4. **Strengthening Data Preparation:** To make the model more resilient to issues like missing data or sudden market shifts, we should use reliable imputation methods and flag volatility indicators. This way, our model can better handle unexpected variations and stay robust.