Portfolio Optimization with Hierarchical Clustering and Markowitz

Rommel Lopez Garcia

Business Problem

The business problem revolves around the need to manage investment portfolios efficiently, maximizing risk-adjusted return and minimizing overall risk through proper diversification. The use of clustering and the Markowitz model, along with the incorporation of regularization techniques, allows for effective asset selection and the construction of an optimized portfolio, helping investors make informed and strategic decisions in a dynamic and often uncertain financial environment.

1. **Hierarchical Clustering**: The hierarchical clustering algorithm is used to group assets with similar return patterns, based on a correlation matrix. This helps investors identify assets that could form a cohesive group from the perspective of their market behavior.

2. **Markowitz Portfolio Optimization**: Subsequently, the business problem is addressed using Markowitz's theory to optimize the weights of the selected assets, maximizing the Sharpe ratio, which measures risk-adjusted return.

Ticker Definition

```
TICKERS = [
    # Technology
    'AAPL', 'MSFT', 'GOOGL', 'NVDA','AMZN', 'META', 'INTC', 'AMD', 'CSCO',

    # Healthcare
    'JNJ', 'PFE', 'MRK', 'UNH', 'ABBV', 'BMY', 'GILD', 'REGN', 'VRTX', 'LLY',

    # Energy
    'XOM', 'CVX', 'SLB', 'BP', 'COP', 'OXY', 'EOG', 'KMI', 'PSX', 'HAL',

    # Financials
    'JPM', 'GS', 'BRK-B', 'C', 'BAC', 'WFC', 'MS', 'AXP', 'BLK', 'SCHW',

    # Consumer Staples
    'KO', 'PEP', 'PG', 'WMT', 'MO', 'CL', 'KMB', 'EL', 'STZ', 'COST',

    # Consumer Discretionary
    'DIS', 'TSLA', 'NFLX', 'HD', 'NKE', 'SBUX', 'MCD', 'TGT', 'LOW',

    # Industrials
    'BA', 'CAT', 'UPS', 'LMT', 'MMM', 'GE', 'HON', 'DE', 'UNP', 'CSX',

    # Telecommunications
    'VZ', 'T', 'TMUS', 'CHT', 'CMCSA', 'EA', 'NOK', 'ERIC', 'CCI', 'SBAC',

    # ETFs
    'SPY', 'QQQ', 'VTI', 'EWW', 'IEMG', 'EFA', 'VWO', 'IXC', 'XLE', 'XLF',
]
```

Dataset Description

The dataset used consists of historical closing prices for various financial assets across different economic sectors, aiming for proper diversification. These data are obtained from Yahoo Finance. The assets are identified by their ticker symbols (e.g., AAPL for Apple, MSFT for Microsoft).

- **Data Period**: From January 2014 to December 2024.
- **Features**: Closing prices of the selected assets.

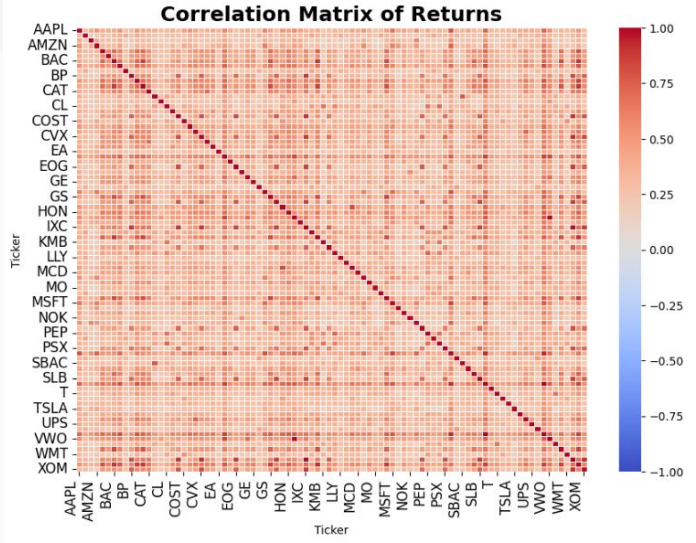
Exploratory Data Analysis (EDA):

Logarithmic Returns:
The closing prices are transformed into logarithmic returns to measure price variations over time. This analysis is crucial because logarithmic returns are more accurate for financial modeling, as they account for compounding over time and provide a better representation of the underlying asset price movements.

Correlation Matrix:
An important part of EDA is visualizing the correlation between assets. This helps to understand the relationships between different assets and how they can be grouped based on their behaviors.

Analyzing this matrix allows us to identify assets that are highly correlated, which could be grouped together to diversify risk. This insight is valuable for portfolio construction, as it highlights which assets move similarly, and therefore, should not be overrepresented in the portfolio to maintain effective diversification.

```
data = yf.download(TICKERS, start="2014-01-01", end="2024-12-31")['Close']

# Calculate Logarithmic Returns
returns = np.log(data / data.shift(1)).dropna()

# Correlation Matrix
correlation_matrix = returns.corr()

# Adjust figure size based on the number of assets
plt.figure(figsize=(10,7))

sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm',  # Remove values
            vmin=-1, vmax=1,  # Limit colorbar range
            linewidths=0.5,  # Cell borders
            linecolor='white')  # Change cell border color

# Improve presentation
plt.title("Correlation Matrix of Returns", fontsize=18, fontweight='bold')
plt.xticks(rotation=90, ha='right', fontsize=12)  # Increase column label size
plt.yticks(rotation=0, fontsize=12)  # Increase row label size

plt.show()
```



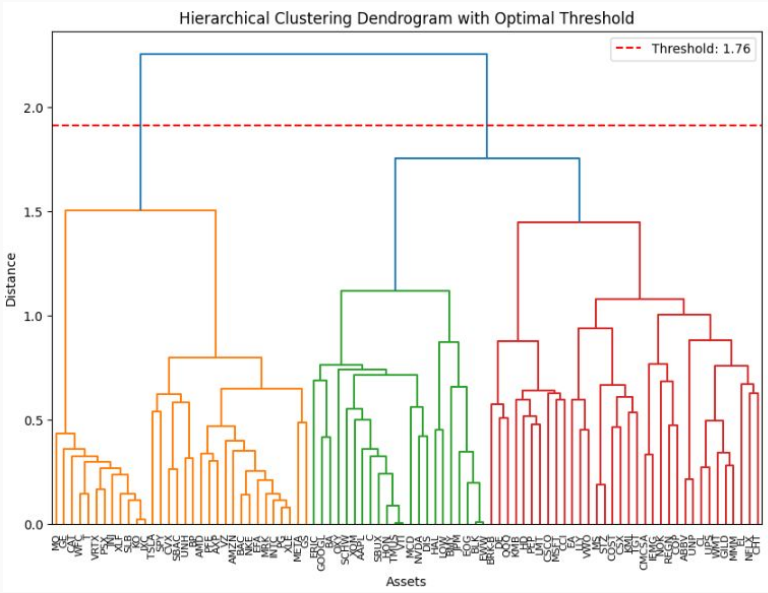
Hierarchical Clustering

```python
# Normalize the data for Hierarchical Clustering
scaler = StandardScaler()
scaled_returns = scaler.fit_transform(returns.T)
```

```python
# Convert the correlation matrix to a distance matrix (1 - correlation)
distance_matrix = 1 - correlation_matrix
# Convert the distance matrix to condensed form
condensed_distance_matrix = squareform(distance_matrix)
```

```python
# Generate the linkage with the best linkage method
Z = linkage(condensed_distance_matrix, method='ward')  # You can change the method here

# Calculate the optimal number of clusters using the Dendrogram (using the optimal threshold)
distances = Z[:, 2]  # Extract the distances
distance_diff = np.diff(distances)  # Difference between consecutive distances
threshold_index = np.argmax(distance_diff)  # Largest gap in the distance
optimal_threshold = distances[threshold_index]  # Optimal threshold

# Use fcluster to obtain clusters based on the threshold
clusters = fcluster(Z, optimal_threshold, criterion='distance')
```

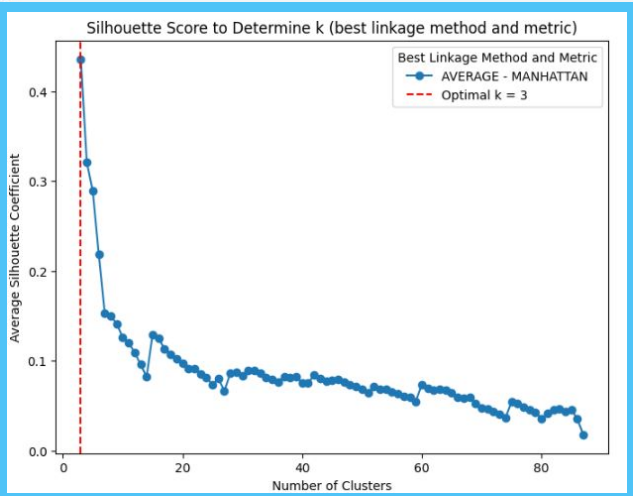
```python
# Get the optimal number of clusters from the dendrogram
optimal_k_dendrogram = len(set(clusters))

# Plot the Dendrogram with the optimal threshold
plt.figure(figsize=(10, 7))
dendrogram(Z, labels=TICKERS, leaf_rotation=90, leaf_font_size=8)
separation = optimal_threshold * 1.09
plt.axhline(y=separation, color='r', linestyle='--', label=f'Threshold: {optimal_threshold:.2f}')
plt.title("Hierarchical Clustering Dendrogram with Optimal Threshold")
plt.xlabel("Assets")
plt.ylabel("Distance")
plt.legend()
plt.show()
```



Evaluation of Clustering Quality

```python
def evaluate_silhouette(k, linkage_method, metric, data):
    try:
        # Perform clustering and calculate the Silhouette Score
        clustering = AgglomerativeClustering(n_clusters=k, linkage=linkage_method)
        labels = clustering.fit_predict(data)
        return silhouette_score(data, labels, metric=metric)
    except Exception as e:
        print(f"Error evaluating k={k} with {linkage_method} and metric {metric}: {e}")
        return -1  # Return an invalid value if there's an error

# Initialize evaluation variables
best_score = -1
optimal_k_silhouette = 0

# Dictionary to store results only for the best linkage method and metric
silhouette_scores = {}

# The range of k goes from a minimum value to a dynamic maximum value
min_k = 3  # Minimum value of k, you can adjust it as needed
max_k = len(TICKERS)  # Maximum value of k will be the total number of assets
metrics = ['euclidean', 'manhattan', 'cosine']
methods = ['ward', 'complete', 'average', 'single']

# Evaluation for different values of k, linkage method, and metrics
for k in range(min_k, max_k):  # We use the dynamic range
    for linkage_method in methods:
        for metric in metrics:
            score = evaluate_silhouette(k, linkage_method, metric, scaled_returns)
            if score != -1:  # Only store valid results
                if score > best_score:
                    best_score = score
                    best_linkage = linkage_method
                    best_metric = metric
                    optimal_k_silhouette = k  # Update the optimal k when a better score is found
```

```python
# Plot the Silhouette Score vs. Number of Clusters for the best method and metric
# Only take the values corresponding to the best method and metric
plt.figure(figsize=(8, 6))

# Recalculate the Silhouette Score for the values of k with the best method and metric
best_method_scores = []
for k in range(min_k, max_k):
    score = evaluate_silhouette(k, best_linkage, best_metric, scaled_returns)
    if score != -1:
        best_method_scores.append((k, score))

# Convert to DataFrame for easier visualization
df_scores = pd.DataFrame(best_method_scores, columns=["k", "score"])

# Plot the result
plt.plot(df_scores['k'], df_scores['score'], marker='o', label=f'{best_linkage.upper()} - {best_metric.upper()}')

# 3. Add a vertical line for the optimal k
plt.axvline(x=optimal_k_silhouette, color='red', linestyle='--', label=f'Optimal k = {optimal_k_silhouette}')

# Titles and labels
plt.title("Silhouette Score to Determine k (best linkage method and metric)")
plt.xlabel("Number of Clusters")
plt.ylabel("Average Silhouette Coefficient")
plt.legend(title="Best Linkage Method and Metric")
plt.show()
```



Application of Optimal Clustering

- **Cluster Label Assignment:**

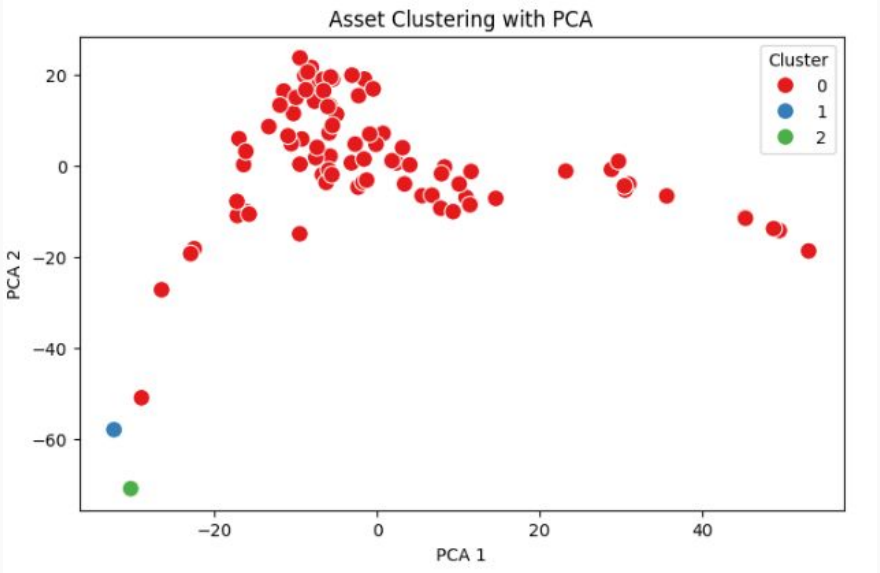
Each asset is assigned to a specific cluster. This is done using the optimal number of clusters determined earlier.

- **Visualization with PCA:**

The dimensionality is reduced using PCA, which helps represent the clusters in a 2D space. This visualization makes it easier to understand how the assets are grouped.

```
# Apply Hierarchical Clustering with the optimal number of clusters
agglomerative_clustering = AgglomerativeClustering(n_clusters=optimal_k_silhouette, linkage=best_linkage)
labels = agglomerative_clustering.fit_predict(scaled_returns)

joblib.dump(agglomerative_clustering, '../models/agglomerative_clustering.pkl')

print("Clustering model saved successfully.")
```

```
# Visualize Clusters with PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_returns)

plt.figure(figsize=(8, 5))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1], hue=labels, palette='Set1', s=100)
plt.title("Asset Clustering with PCA")
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.legend(title="Cluster")
plt.show()
```



Selection of Representative Assets

```python
# Select a Representative Asset from Each Cluster based on Average Return
df_cluster = pd.DataFrame({'Ticker': TICKERS, 'Cluster': labels})

risk_free_rate = 0.03 / 252  # Convert to daily returns

def select_optimal_asset(assets_in_cluster):
    valid_assets = set(assets_in_cluster) & set(returns.columns)  # Intersection between both sets
    if not valid_assets:
        return None

    # Convert the intersection back to a list
    valid_assets = list(valid_assets)

    average_returns = returns[valid_assets].mean()
    volatilities = returns[valid_assets].std() + 1e-8  # Add a small value to avoid division by zero
    sharpe_ratios = (average_returns - risk_free_rate) / volatilities

    return sharpe_ratios.idxmax()  # Return the ticker with the highest Sharpe ratio

# Select the optimal assets from each cluster
selected_assets = df_cluster.groupby('Cluster')['Ticker'].apply(select_optimal_asset).dropna().values

# Show selected assets for portfolio optimization
print("Selected Assets for Portfolio Optimization:", selected_assets.tolist())
```

```
Selected Assets for Portfolio Optimization: ['NVDA', 'ERIC', 'GOOGL']
```

Sharpe Ratio:

Asset Selection: Within each cluster, we select the asset with the best Sharpe Ratio, as this represents the asset that has achieved the best risk-adjusted return.

The asset with the highest Sharpe Ratio has demonstrated the best relationship between return and volatility, making it the most attractive for the portfolio.

Portfolio Optimization

```python
# Get returns and covariance matrix of selected assets
selected_returns = returns[selected_assets]
average_returns = selected_returns.mean()
cov_matrix = selected_returns.cov()

# 3. Optimization function to find the best asset combination
def regularized_sharpe_ratio(weights, average_returns, cov_matrix, lambda_ridge=0.05):
    return_expected_return = np.sum(weights * average_returns)  # Expected return of the portfolio
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))  # Portfolio volatility
    regularization = lambda_ridge * np.sum(weights**2)  # Regularization term (Ridge)
    return -((return_expected_return - risk_free_rate) / portfolio_volatility) + regularization  # Minimize regularized
# Efficient Frontier Optimization
# Required parameters
num_assets = len(selected_assets)
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})  # Constraint that weights sum to 1
volatilities = np.sqrt(np.diag(cov_matrix))  # Volatility of each asset
bounds = tuple((0.01, min(0.50, 1 / vol)) for vol in volatilities)  # Limit weights between 0.01 and 0.50 or 1/volatility

# Initial random weights
initial_weights = np.random.dirichlet(np.ones(num_assets), size=1)[0]

# Perform optimization
optimization_result = minimize(regularized_sharpe_ratio, initial_weights, args=(average_returns, cov_matrix),
                               method='SLSQP', bounds=bounds, constraints=constraints)
optimal_weights = optimization_result.x  # Optimal weights found
```

Ridge Regularization: Regularization is used to prevent overfitting by penalizing large values in the portfolio weights. This ensures that the model does not assign too much weight to assets that could be overly volatile or unrepresentative.

Optimization: Using techniques such as scipy.optimize, we adjust the asset weights to find the combination that maximizes the Sharpe ratio.

Markowitz Model:

Markowitz optimization focuses on creating an efficient portfolio in which risk (volatility) is minimized for a given level of expected return, or return is maximized for a given level of risk.

Efficient Frontier: All portfolios on this frontier represent optimal combinations of risk and return.

Expected Return and Volatility:

We calculate the return and volatility of each possible portfolio using the historical returns of the selected assets, weighted according to the assigned portfolio weights.

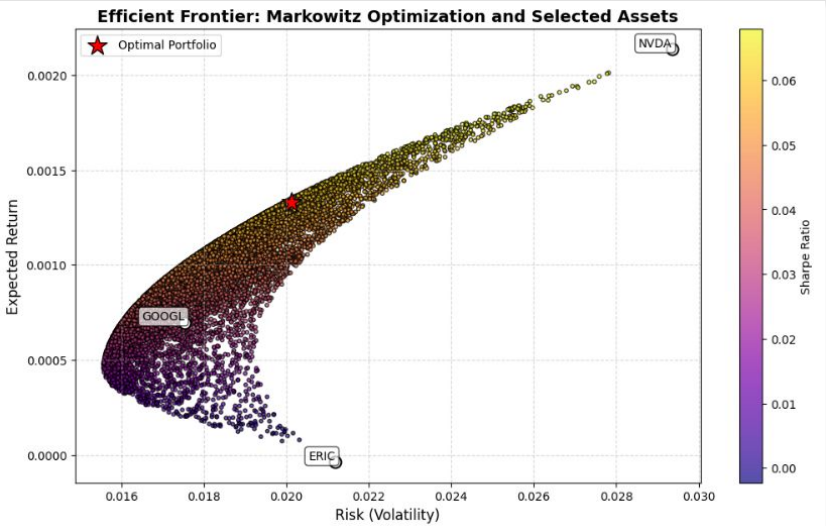
Generation of the Efficient Frontier:

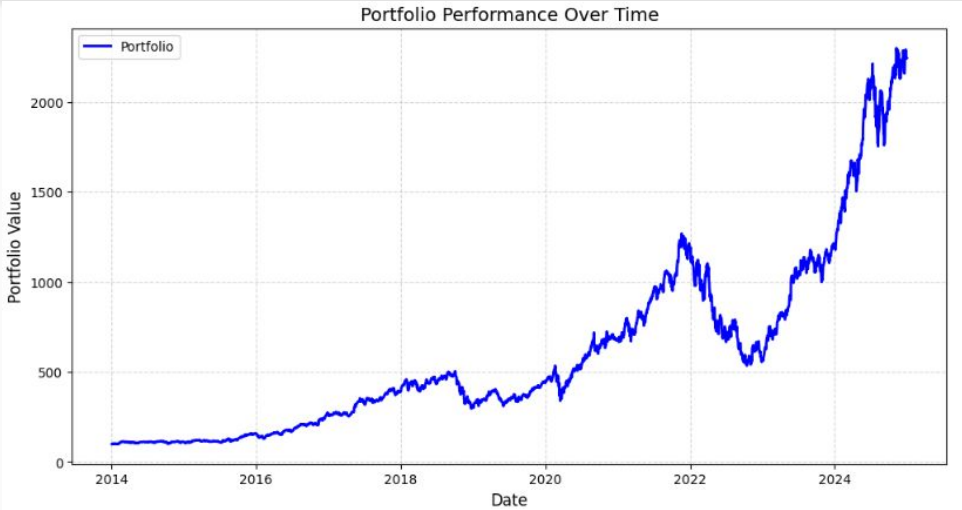
We perform different random combinations of weights and calculate their return and risk, allowing us to construct the efficient frontier.

```
# Calculate expected return and risk of the optimal portfolio
portfolio_return = np.sum(optimal_weights * average_returns)  # Expected return of the portfolio
portfolio_volatility = np.sqrt(np.dot(optimal_weights.T, np.dot(cov_matrix, optimal_weights)))  # Portfolio risk
optimal_sharpe = (portfolio_return - risk_free_rate) / (portfolio_volatility + 1e-8)  # Sharpe ratio
```

```
# Generate random portfolios for the efficient frontier
num_portfolios = 10000  # Number of portfolios to generate
results = np.zeros((3, num_portfolios))  # Return, Risk, Sharpe Ratio
weights_record = np.zeros((num_portfolios, num_assets))  # Record of weights for each portfolio

for portfolio_idx in range(num_portfolios):
    weights = np.random.uniform(0.01, 0.50, num_assets)  # Weights between 0.01 and 0.50 for each asset
    weights /= np.sum(weights)  # Normalize to sum to 1 (fully invested portfolio)

    # Record the generated weights for this portfolio
    weights_record[portfolio_idx, :] = weights
    # Calculate expected return of the portfolio
    return_portfolio = np.dot(weights, average_returns)  # Use np.dot for efficient multiplication (weighted sum)
    # Calculate portfolio volatility (risk) using the covariance matrix
    volatility_portfolio = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))  # Calculate volatility
    # Calculate Sharpe ratio for this portfolio (adjusted for risk-free rate)
 sharpe = (return_portfolio - risk_free_rate) / volatility_portfolio if volatility_portfolio != 0 else 0
 # Record portfolio results: return, risk, Sharpe
 results[:, portfolio_idx] = [return_portfolio, volatility_portfolio, sharpe]
```

Portfolio Return and Risk Calculation






```
Optimal portfolio weights based on hierarchical clustering and Markowitz optimization:
NVDA: 50.00%
ERIC: 12.02%
GOOGL: 37.98%
Expected Return of the Optimal Portfolio: 0.13%
Volatility of the Optimal Portfolio: 2.01%
Sharpe Ratio of the Optimal Portfolio: 0.0601
```

```
Weighted portfolio return: -0.16%
Accumulated portfolio price in the last period: 2244.74
```

Final Results

1. **Optimal Number of Clusters (Dendrogram):**
    - Optimal number of clusters: **2**
    - Identification based on dendrogram visualization.
2. **Optimal Threshold:**
    - Threshold: **1.76**
    - Marks the cutoff point in the clustering hierarchy.
3. **Best Linkage Method:**
    - Method: **Average**
    - Metric: **Manhattan**
    - **Silhouette Score:** 0.44 (moderate clustering quality).

Asset	Weight (%)
NVDA (NVIDIA)	**50.00**
ERIC (Ericsson)	**11.87**
GOOGL (Google)	**38.13**

Metric	Value
Rendimiento Esperado (Retorno modesto)	**0.13%**
Volatilidad (Riesgo bajo)	**2.01%**
Índice de Sharpe (Baja relación retorno-riesgo)	**0.0601**
Rendimiento Ponderado (Pequeña pérdida)	**-0.16%**

Final Portfolio Value:

- **Accumulated Portfolio Value:** 2245.67
- Positive growth from the initial value of 100.

Conclusions

Although the optimization model based on hierarchical clustering and Markowitz already provides a solid solution for creating efficient portfolios, there are key areas for improvement. The combination of new clustering techniques, multi-objective optimization, incorporating additional data, and advanced simulations can further increase the model's efficiency and enhance decision-making in more complex scenarios. By implementing these improvements, it is possible to optimize the risk-adjusted return more accurately and adapt it to a changing financial environment.