GEORGIA INSTITUTE OF TECHNOLOGY

ISYE 6767 FINAL PROJECT

Implementing a Statistical Arbitrage Strategy

Student Name:
Chongrui Wang

Student ID:
xxx

December 14, 2023

Contents

1 Introduction

This document outlines the implementation of a statistical arbitrage strategy using a series of Python functions. Each function contributes to the overall process of identifying and exploiting market inefficiencies based on statistical analysis.

The primary objective of the provided code is to develop a robust framework for financial market analysis, particularly for trading signal generation. This framework encompasses several sophisticated statistical and mathematical techniques, tailored to address key challenges in financial time series analysis and trading strategy formulation. The core components of the problem addressed are as follows:

1.1 Application of PCA to Normalized Stock Return Correlation Matrix

The initial phase of the analysis involves the application of Principal Component Analysis (PCA) to the normalized correlation matrix of stock returns. This step is crucial for several reasons:

- **Dimensionality Reduction:** Stock markets generate vast amounts of data. PCA helps in reducing the dimensionality of this data, making it more manageable and less prone to overfitting.
- **Identification of Principal Components:** PCA aids in identifying the principal components that explain the maximum variance in the stock returns. These components serve as the primary risk factors in the market.

1.2 Fitting the OU Process to Residuals

After extracting the principal risk factors, the next step involves fitting an Ornstein-Uhlenbeck (OU) process to the residuals of the returns not explained by these factors. The OU process is a mean-reverting stochastic process characterized by its tendency to revert to a long-term mean value over time. The significance of this step includes:

- **Mean Reversion Modeling:** The OU process models the mean-reverting behavior of stock prices, which is a common phenomenon in financial markets.
- **Residual Analysis:** By analyzing the residuals with the OU process, we can capture the stock-specific behaviors that are not explained by the market-wide risk factors.

1.3 Estimation of Mean Reversion Process

The OU process parameters estimated from the residuals provide insights into the mean reversion characteristics of individual stocks. This includes:

- **Speed of Mean Reversion:** How quickly the stock prices revert to their mean values.
- **Equilibrium Level:** The long-term mean level to which the stock prices revert.

1.4 Construction of S-Score for Trading Signals

The final and most critical step in the framework is the construction of the S-score. The S-score is a quantitative metric derived from the mean reversion parameters and is used to determine the trading signals - whether to take a long or short position in a stock. The process involves:

- **Quantitative Scoring:** Stocks are scored based on their mean reversion characteristics.
- **Signal Generation:** Based on the S-scores, trading signals are generated. A high S-score might indicate a long position, while a low S-score could suggest a short position.

In summary, the code provides a comprehensive approach to identifying trading opportunities in financial markets by leveraging statistical techniques like PCA and the OU process. The combination of these methods allows for a nuanced understanding of market dynamics and individual stock behaviors, culminating in the generation of data-driven trading signals.

2 Object-oriented Design

2.1 FinancialAnalyzer Class Functions

2.1.1 `__init__`

Purpose: Initializes a new instance of the `FinancialAnalyzer` class. Currently, it does not perform any specific initialization tasks.

Parameters: None, except for the implicit `self` parameter which refers to the instance of the class itself.

2.1.2 `select_common_tokens`

Purpose: Selects common tokens based on market capitalization within a specified time frame. It aims to filter tokens that are frequently present in the market capitalization data over the given time window. Specifically, It filtered out the stocks not frequently present in the market capitalization data over the given time window and the stocks with less than 80% valid data.

Parameters:

- `market_cap_df (pd.DataFrame)`: A DataFrame containing market capitalization data for different tokens.
- `t (str)`: A string representing the end time of the time window for the token analysis.
- `M (int)`: An optional integer specifying the time window in hours. Defaults to 240 hours.

Returns: A list containing the selected tokens.

2.1.3 `compute_hourly_returns`

Purpose: Computes hourly returns for a list of tokens over a specified time period. This function aims to calculate the percentage change in price for each token on an hourly basis within the given time frame. We shall first calculate the stock return and normalize it by minus its mean and then dividing its standard deviation.

Parameters:

- `price_df (pd.DataFrame)`: A DataFrame containing price information for various tokens.
- `tokens_list (list)`: A list of tokens for which the hourly returns are to be computed.
- `t (str)`: A string representing the end time of the time window for the return calculation.
- `M (int)`: An optional integer specifying the time window in hours. Defaults to 240 hours.

Returns: A Dataframe representing the matrix of stock return.

2.1.4 `compute_matrices`

Purpose: Computes the empirical correlation and variance-covariance matrices from a DataFrame of returns. This is essential for understanding the relationships between different tokens' returns and for further financial analysis like risk assessment and portfolio optimization.

Parameters:

- `returns_df (pd.DataFrame)`: A DataFrame containing returns data for various tokens.

Returns: A tuple containing two elements:

- The correlation matrix of the tokens' returns.
- The variance-covariance matrix of the tokens' returns.

2.1.5 `compute_pca`

Purpose: Applies Principal Component Analysis (PCA) to the covariance matrix to identify the principal components. This is useful for dimensionality reduction and identifying the main factors that explain the variability in the dataset.

Parameters:

- `covariance_matrix (pd.DataFrame)`: The empirical covariance matrix derived from the returns data.
- `n_components (int)`: The number of principal components to retrieve. Defaults to 2.

Returns: A tuple containing two elements:

- The eigenvectors (principal components) of the covariance matrix.
- The eigenvalues associated with each principal component.

2.1.6 `construct_risk_factors`

Purpose: Constructs risk factors from the eigenvectors obtained from PCA. This function is used to normalize the eigenvectors by the standard deviations of the assets, thereby creating weights for each risk factor.

Parameters:

- `eigenvectors (np.array)`: The eigenvectors obtained from the PCA analysis.
- `std_devs (pd.Series)`: A series of standard deviations for each asset in the returns data.

Returns: A DataFrame containing the weights for each risk factor, normalized by the standard deviations of the assets.

2.1.7 `calculate_factor_return`

Purpose: Calculates the factor return of risk factors for a given period. This function aims to determine the contribution of each risk factor to the returns of the assets in the specified period.

Parameters:

- `risk_factors (pd.DataFrame)`: The DataFrame containing the risk factor weight vectors.
- `returns_at_k (pd.DataFrame)`: The DataFrame containing the asset returns for a specific period.

Returns: An array representing the factor returns for each risk factor.

2.1.8 `estimate_residuals_and_coefficients`

Purpose: Estimates regression coefficients and residuals for each token. This function is used in statistical analysis to understand the relationship between the returns of the tokens and the factor returns, capturing the idiosyncratic movements of each token.

Parameters:

- `returns_df (pd.DataFrame)`: The DataFrame containing the returns of tokens.
- `factor_returns (pd.DataFrame)`: The DataFrame containing the factor returns.

Returns: A tuple containing two DataFrames:

- The first DataFrame contains the regression coefficients for each token.
- The second DataFrame contains the residuals for each token.

2.1.9 `estimate_ou_parameters_all_tokens`

Purpose: Estimates the Ornstein-Uhlenbeck (OU) parameters from the residuals for all tokens. This function is used for mean-reversion analysis in financial time series, characterizing the speed of mean reversion, long-term mean, and volatility of each token.

Parameters:

- `residuals (pd.DataFrame)`: The residuals from the regression analysis for all tokens.
- `delta_t (float)`: The time interval, defaulting to $\frac{1}{8760}$, which represents an hourly interval in a year.

Returns: A DataFrame containing the OU parameters (kappa, m, sigma, sigma_eq) for all tokens.

2.1.10 `calculate_s_scores_all_tokens`

Purpose: Calculates the s-scores for all tokens based on the estimated OU parameters and the residuals. The s-score is a statistical measure used in mean-reversion strategies, indicating how far a current price is deviated from its historical mean.

Parameters:

- `ou_parameters (pd.DataFrame)`: A DataFrame containing the OU parameters for all tokens.
- `residuals (pd.DataFrame)`: The residuals from the regression analysis for all tokens.

Returns: A Series containing the s-scores for all tokens.

2.1.11 `generate_trading_signals`

Purpose: Generates trading signals based on s-scores and predefined thresholds. This function is designed to identify trading opportunities by categorizing tokens as 'buy', 'sell', or 'hold' based on their current s-score relative to the thresholds.

Parameters:

- `s_scores (pd.Series)`: Series containing the s-scores for all tokens.
- `thresholds (dict)`: A dictionary containing threshold values for generating trading signals, including thresholds for buying, selling, and closing positions.

Returns: A Series containing the trading signals for all tokens, indicating actions like 'buy to open', 'sell to open', 'close short position', 'close long position', or 'no signal'.

2.1.12 generate_signals_for_dates

Purpose: Generates trading signals for a range of dates, considering price and universe data, along with predefined thresholds. This function aims to systematically create trading signals over a specified date range, incorporating various steps of financial analysis.

Parameters:

- `start_date (str)`: The start date for the trading signal generation.
- `end_date (str)`: The end date for the trading signal generation.
- `prices (pd.DataFrame)`: A DataFrame containing price information of tokens.
- `universe (pd.DataFrame)`: A DataFrame containing the universe of cryptocurrencies.
- `thresholds (dict)`: A dictionary containing the threshold values for signals.

Returns: A dictionary of DataFrames containing the trading signals for each date within the specified range.

2.1.13 plot_cumulative_returns

Purpose: Plots the cumulative returns of eigen-portfolios, BTC, and ETH within a specified date range. This function is used to visually compare the performance of different investment strategies and major cryptocurrencies over time.

Parameters:

- `stock_prices (pd.DataFrame)`: A DataFrame containing hourly stock prices.
- `portfolio_df (pd.DataFrame)`: A DataFrame containing portfolio weights.
- `start_date (str)`: The start date for the analysis.
- `end_date (str)`: The end date for the analysis.

2.1.14 plot_eigenportfolio_weights

Purpose: Plots the eigenportfolio weights at two specified timestamps. This function visualizes the distribution of weights in the eigenportfolios at specific points in time, aiding in understanding the composition and changes in the portfolios.

Parameters:

- `prices (pd.DataFrame)`: A DataFrame containing price information.
- `universe (pd.DataFrame)`: A DataFrame containing the universe of cryptocurrencies.
- `timestamp1 (str)`: The first timestamp for the plot.
- `timestamp2 (str)`: The second timestamp for the plot.

2.1.15 `ensure_btc_eth_in_list`

Purpose: Ensures that "BTC" and "ETH" are included in the list of tokens. This function is a utility used to guarantee that Bitcoin and Ethereum are always considered in the analysis, given their significance in the cryptocurrency market.

Parameters:

- `str_list (list)`: A list of token strings.

Returns: The modified list with "BTC" and "ETH" included if they were not already present.

2.1.16 `plot_s_scores`

Purpose: Plots the s-scores for selected tokens between specified start and end dates. This function visualizes the evolution of the s-scores over time for individual tokens, aiding in the analysis of their mean-reversion behavior.

Parameters:

- `prices (pd.DataFrame)`: A DataFrame containing price information.
- `universe (pd.DataFrame)`: A DataFrame containing the universe of cryptocurrencies.
- `tokens (list)`: A list of tokens to plot.
- `start_date (str)`: The start date for the plot.
- `end_date (str)`: The end date for the plot.

Returns: This function does not return a value but generates and displays plots for the s-scores of the specified tokens.

2.1.17 plot_eigenportfolio_weights_v2

Purpose: An alternative function for plotting the eigenportfolio weights at two specified timestamps. This function offers another method of visualizing the distribution of weights in the eigenportfolios at specific points in time.

Parameters:

- `eigenvector_df (pd.DataFrame)`: A DataFrame containing eigenvectors with timestamps as the index.
- `timestamp1 (str)`: The first timestamp for the plot.
- `timestamp2 (str)`: The second timestamp for the plot.

2.2 SignalGenerator Class Functions

The `SignalGenerator` class is designed to generate trading signals for financial assets, specifically tailored for cryptocurrency markets. It uses statistical scores and predefined thresholds to make decisions.

2.2.1 __init__

Purpose: Initializes a new instance of the `SignalGenerator` class. It sets up a dictionary to keep track of current positions for each token.

Attributes:

- `positions (dict)`: A dictionary to store current positions for each token, initialized as empty.

2.2.2 generate_trading_signals

Purpose: Generates trading signals based on s-scores, thresholds, and current positions. This method systematically processes a DataFrame of s-scores to determine trading actions such as 'buy', 'sell', or 'hold'.

Parameters:

- `df (pd.DataFrame)`: A DataFrame containing the s-scores for all tokens with timestamps.
- `thresholds (dict)`: A dictionary containing the threshold values for generating trading signals.

Process: The method iterates through each timestamp and token in the DataFrame. Based on the current position and the s-score of a token, it decides whether to open or close a position, or to hold. The thresholds for these decisions are extracted from the `thresholds` dictionary.

Returns: A DataFrame with columns 'Token', 'Signal', and 'Timestamp', where 'Signal' indicates the trading action and is indexed by 'Timestamp' and 'Token'.

2.3 PortfolioBacktester Class Functions

2.3.1 `__init__`

Purpose: Initializes a new instance of the `PortfolioBacktester` class with a specified starting capital and optional transaction fee.

Parameters:

- `starting_capital (float)`: The initial amount of capital for the portfolio.
- `transaction_fee (float)`: Optional transaction fee as a percentage of the trade value, defaulting to 0.00.

2.3.2 `backtest_portfolio`

Purpose: Backtests the portfolio against a given set of trading signals and asset prices.

Parameters:

- `signals_df (pd.DataFrame)`: A DataFrame containing trading signals for assets.
- `prices_df (pd.DataFrame)`: A DataFrame containing price data for assets.

Process: Iterates through each signal and updates the portfolio accordingly. It also calculates and updates the portfolio's value over time.

2.3.3 `process_signal`

Purpose: Processes a given trading signal for a specific asset.

Parameters:

- `signal (str)`: The trading signal (e.g., 'buy', 'sell').
- `token (str)`: The asset identifier.
- `price (float)`: The price of the asset.
- `fee (float)`: The transaction fee.
- `timestamp (datetime)`: The time of the trade.

2.3.4 buy_to_open, sell_to_open, close_long_position, close_short_position

Purpose: These functions handle specific types of trades like opening or closing long/short positions.

2.3.5 update_portfolio_value

Purpose: Updates the total value of the portfolio based on current asset prices and holdings.

2.3.6 write_transactions_to_file

Purpose: Writes all transactions to a file for record-keeping.

2.3.7 plot_cumulative_return_and_calculate_metrics

Purpose: Plots the cumulative return of the portfolio and calculates performance metrics like Sharpe Ratio and Max Drawdown.

Parameters:

- `df (pd.DataFrame)`: A DataFrame containing the portfolio value history.
- `risk_free_rate (float)`: The risk-free rate used in Sharpe Ratio calculation.

3 Implementation Steps

1. **Token Selection and Return Computation:** Use `select_common_tokens` and `compute_hourly_returns` in the `FinancialAnalyzer` class to identify common tokens and compute their hourly returns.

2. **Matrix Computation:** Calculate correlation and variance-covariance matrices with `compute_matrices` in the `FinancialAnalyzer` class.
3. **PCA Analysis:** Perform PCA using `compute_pca` in the `FinancialAnalyzer` class to extract principal components.
4. **Risk Factor Construction:** Construct risk factors from eigenvectors with `construct_risk_factors` in the `FinancialAnalyzer` class.
5. **Factor Return Calculation:** Calculate factor returns using `calculate_factor_return` in the `FinancialAnalyzer` class.
6. **Regression Analysis:** Estimate regression coefficients and residuals for each token using `estimate_residuals_and_coefficients` in the `FinancialAnalyzer` class.
7. **OU Parameter Estimation:** Estimate OU parameters for all tokens with `estimate_ou_parameters_all_tokens` in the `FinancialAnalyzer` class.
8. **S-Score Calculation:** Calculate s-scores for all tokens using `calculate_s_scores_all_tokens` in the `FinancialAnalyzer` class.
9. **Trading Signal Generation:** Generate trading signals based on s-scores with `generate_trading_signals` in the `SignalGenerator` class.
10. **Backtesting:** Conduct backtesting using `backtest_portfolio` in the `PortfolioBacktester` class, based on the generated trading signals.
11. **Performance Analysis:** Analyze the performance and calculate metrics like Sharpe Ratio and Maximum Drawdown using `plot_cumulative_return_and_calculate_metrics` in the `PortfolioBacktester` class.

4 Code Input and Output

4.1 Input

Figure 1 shows the input of my codes, you may find them in **main.py**.

start_date and **end_date** are dates for generating eigenvectors, eigen portfolios, s scores, signals, and doing backtesting.

timestamp1 and **timestamp2** are two timestamps used to plot the eigen portfolio weights.

s_score_start_date and **s_score_end_date** are dates for calculating and plotting s scores for BTC and ETH.

thresholds are the thresholds we used to generate signals based on the s scores.

tokens is the list of tokens which we want to plot s scores from.

4.2 Output

Figure 2 shows the output of my codes. The first bar is used to calculate eigenvectors, eigen portfolio weights, and s scores for all timestamps. The second bar is used to determine signals based on the s scores I have generated. The third bar called generating signals is used to calculate s score for BTC and ETH, since there is no ETH data within the window if we used the filter method mentioned before, I have to include BTC and ETH to make sure they have data within this window and generate s scores again. The last bar is used to do backtesting based on the signals. Then, this code will also output the Sharpe ratio and Max Drawdown of my strategy.

The first and third bars are generated by class **FinancialAnalyzer**, the second bar is generated by class **SignalGenerator**, and the fourth bar is generated by class **PortfolioBacktester**.

5 Task 1

5.1 Task 1a

Figure 3 and Figure 4 shows the screenshot of the first and second eigenvectors.

5.1.1 Eigenvector DataFrames Structure

The given screenshots depict two separate DataFrames corresponding to the first and second eigenvectors. These DataFrames are used to represent the principal components derived from a dataset, commonly used in the field of financial analysis to capture the underlying factors affecting the price movements of various tokens.

5.1.2 Index

- The index of the DataFrame is labeled as `TimeStamp`, representing the specific dates and times at which the data points were recorded. It follows a chronological order, typically in hourly intervals.
- The format of the timestamps appears to be `YYYY-MM-DD HH:MM:SS`.

5.1.3 Columns

- The columns represent different tokens whose market behavior is being analyzed. Examples of tokens include `1INCH`, `AAVE`, `ALGO`, and so forth.
- Each column is prefixed with `eigenvect`, indicating that the values within are components of an eigenvector corresponding to that particular token.

5.1.4 Values

- The values within each cell of the DataFrame are the eigenvector components. These numerical values represent the weight or contribution of each token to the principal component at a given timestamp.
- The values are typically represented as floating-point numbers and can be positive or negative, indicating the direction of the token's contribution to the principal component.
- Nan value means the stock is not selected in our eigen portfolio at this timestamp.

5.1.5 Interpretation

- A positive value indicates that the token's movement is in the same direction as the principal component.
- Conversely, a negative value indicates movement in the opposite direction to the principal component.
- The first eigenvector has most of its value as negative, which is different from the paper. This difference may come from the different dataset we use. In the paper it uses daily price and look back 252 days, while we use hourly price with a window of 240 hours. Thus, our first eigenvector may not associated with the market portfolio.
- The second eigenvector is much more balanced, which is consistent to the results in the paper. However, it is unsure what component it should represent without further examination.

5.1.6 Conclusion

These DataFrames provide a snapshot of the market's structure at each timestamp, revealing which tokens move together and which move in opposition. This information can be crucial for portfolio construction, risk management, and identifying trading opportunities.

5.2 Task 1b

Figure 5 shows the cumulative return curve for the 4 different assets.

5.2.1 Assets Described

- **BTC (Blue Line)**: Represents the cumulative return for Bitcoin.
- **ETH (Orange Line)**: Represents the cumulative return for Ethereum.
- **Eigen-Portfolio 1 (Green Line)**: Constructed from the first eigenvector of a PCA on asset returns.

- **Eigen-Portfolio 2 (Red Line)**: Constructed from the second eigenvector of the same PCA.

5.2.2 Interpretation of Eigen-Portfolios

The cumulative returns of Eigen-Portfolio 1 and Eigen-Portfolio 2 indicate the performance of hypothetical portfolios constructed by taking positions in assets weighted according to the first and second principal components respectively, derived from a PCA.

- A principal component is a linear combination of the original variables (in this case, tokens' returns) with the largest possible variance. The first eigenvector captures the most variance within the dataset.
- The second eigenvector is orthogonal to the first and captures the next highest variance within the dataset.

5.2.3 Analysis of Returns

The plot suggests the following:

- The Eigen-Portfolio 1 and Eigen-Portfolio 2 follow different paths which suggest they are capturing different aspects of market behavior.
- The first eigenportfolio might be capturing the general market trend, while the second could be identifying a different, possibly uncorrelated factor affecting asset prices.
- The similarity in the movement patterns of BTC and ETH with the eigenportfolios at certain times may indicate market conditions where these cryptocurrencies are strongly influencing or are aligned with the broader market factors captured by the eigenportfolios.
- The times when the eigenportfolios diverge significantly from BTC and ETH could indicate market conditions where factors other than the market trends of these two major cryptocurrencies are at play.
- After analyzing the eigenvalues, I find that the first eigenvalue is significantly larger than others, which means the first component explain the most of the variance. Thus, the second component may not explain the variance well, which may account for the bad performance of its cumulative return (looks like noise data).

5.2.4 Conclusion

The cumulative returns for the eigenportfolios can provide insights into underlying risk factors and can be used for diversification in portfolio management. The returns of these portfolios relative to BTC and ETH can also offer an understanding of how alternative strategies might perform against holding the major cryptocurrencies directly.

6 Task 2

Figure 6, Figure 7, Figure 8, and Figure 9 show the Eigen Portfolio weights at different timestamp in both dollar amount and weights. By dollar amounts, I mean just dividing eigenvectors by the standard deviation of the stock return. By weights, I mean normalizing dollar amount weights to make their summation equals to 1.

The provided plots represent the weights of various assets in two eigenportfolios at specific time points. The weights are derived from the principal components of a covariance matrix, which are generated through a process known as Principal Component Analysis (PCA). The PCA technique is commonly used in portfolio management to identify the underlying factors that drive asset returns.

Also, I plot the eigenvectors as well for reference, you can find them in Figure 10 and Figure 11.

6.1 Eigenportfolio Weights in Dollar Amount

In Figure 6 and Figure 7, the weights in dollar amounts reflect the proportion of the portfolio's capital allocated to each asset in absolute terms. A positive weight indicates a long position, while a negative weight suggests a short position. The magnitude of the weight signifies the size of the position in dollars.

6.2 Eigenportfolio Weights in Relative Weights

In Figure 8 and Figure 9, the relative weights, on the other hand, are the proportion of each asset's weight relative to the total portfolio. They sum up to 1 or -1, depending on whether the portfolio is long or short overall. These weights are dimensionless and indicate the asset's relative importance within the portfolio.

6.3 Interpretation of Values

- Large positive or negative weights in dollar terms indicate a significant bet on the rise or fall of an asset's price, respectively.
- In relative terms, assets with larger absolute weights have a greater influence on the portfolio's performance.
- The range of second eigen portfolio weights are larger than the first's, which means the second eigen portfolio is much more volatile. As what I have analyzed during task 1a, this may come from the result that the range of first eigenvector is much smaller because it capture most of the variance and has a good capability to explain stock return.
- Since most of values in our first eigen portfolio is negative, it may not represent the market portfolio as the paper suggests. However, the second eigen portfolio has more balanced values, it may have the same meaning as the one from the paper.

6.4 Conclusion

The eigenportfolio weights plots are crucial for understanding the risk and return profile of the portfolio. The distribution of weights among different assets reflects the strategy's market views and the risk it is willing to take. Monitoring these weights over time can provide insights into how the portfolio's risk exposures evolve.

7 Task 3

Figure 12 and Figure 13 illustrate the evolution of the s-scores for Bitcoin (BTC) and Ethereum (ETH) over time. The s-score is a statistical measure that indicates how far a given asset is from its theoretical equilibrium as defined by a cointegration model.

7.1 Understanding s-score

- The s-score is calculated as the number of standard deviations by which the current price deviates from a model-implied equilibrium value.
- A high positive s-score indicates that the asset's price is above the equilibrium value suggested by the model, potentially signaling an overvalued state.
- Conversely, a high negative s-score suggests that the asset's price is below the equilibrium, potentially signaling an undervalued state.

7.2 Analysis of BTC s-score Plot

Figure 12 shows fluctuations around the equilibrium with several spikes, both positive and negative. These spikes may represent moments when BTC was overbought or oversold relative to the equilibrium value predicted by the model.

7.3 Analysis of ETH s-score Plot

Similarly, Figure 13 shows fluctuations and is indicative of how ETH's price varied in relation to its theoretical equilibrium. The frequency and magnitude of deviations could reflect the asset's volatility and the market's reaction to new information.

7.4 Analysis

Notice that if we set the thresholds for signals as the same as the ones in the paper, we will generate most of the long or short signals for both BTC and ETH at the same time. It makes sense since when I plot the cumulative return of BTC and ETH within this time interval, they have similar return curve. It means when we get long or short signals, it is actually more like long or short signals for the whole cryptocurrency sector instead of specific stock.

7.5 Conclusion

The analysis of s-scores for BTC and ETH provides insight into the behavior of these cryptocurrencies in the context of a mean-reversion trading strategy. When the s-score crosses predefined threshold levels, it may trigger buy or sell signals within the strategy. This approach assumes that prices will revert to their long-term equilibrium, offering potential opportunities for profit.

8 Task 4

8.1 Task 4a

Figure 14 shows the screenshot of part of the signals I generated.

8.1.1 Structure of the Trading Signals DataFrame

The provided screenshot shows a DataFrame of trading signals generated for various tokens over time. Each cell in the DataFrame contains a trading signal based on the s-score for that token at the given timestamp.

8.1.2 Index

- The index of the DataFrame is `Timestamp`, representing the specific points in time when the trading signals are evaluated.

8.1.3 Columns

- The columns represent different cryptocurrencies (tokens) for which the signals are being generated.

8.1.4 Values

- The values are trading signals: 'buy to open', 'sell to open', 'close long position', 'close short position', or 'hold'.
- 'buy to open' indicates a signal to take a long position, 'sell to open' is a signal to take a short position, 'close long position' and 'close short position' indicate signals to exit those respective positions, and 'hold' indicates no action is advised at that timestamp.

8.1.5 Logic Behind Signal Generation

The `SignalGenerator` class is responsible for generating these signals based on the following logic:

- If the current position is 'none' or 'long' and the s-score is less than or equal to the 'buy to open' threshold, a 'buy to open' signal is generated.
- If the current position is 'none' or 'short' and the s-score is greater than the 'sell to open' threshold, a 'sell to open' signal is generated.
- If the current position is 'long' and the s-score is greater than the 'close long position' threshold, a 'close long position' signal is generated.
- If the current position is 'short' and the s-score is less than the 'close short position' threshold, a 'close short position' signal is generated.
- If none of these conditions are met, a 'hold' signal is generated.
- For those stocks not in our eigen portfolio, we shall give a nan value for its signal.

The `SignalGenerator` class maintains a dictionary of current positions to determine the appropriate signal based on the s-score and the defined thresholds.

8.1.6 Conclusion

The signals DataFrame is a systematic way to translate quantitative metrics (s-scores) and defined thresholds into actionable trading decisions. It allows for a disciplined approach to market entry and exit, potentially reducing emotional biases in trading.

8.2 Task 4b

The backtest results comprise two plots: one for the portfolio's cumulative return over time and another showing the distribution of the portfolio's hourly returns.

8.2.1 Portfolio Cumulative Return

The cumulative return plot (Figure 15) of the portfolio represents the aggregated percentage change in the portfolio's value over time. It reflects the combined effect of individual asset returns within the portfolio, adjusted for the portfolio's asset allocation.

Analysis

- The plot indicates the total gain or loss experienced by the portfolio from the start of the period under consideration.
- A positive slope in the plot signifies a period of overall portfolio growth, while a negative slope points to a decline in portfolio value.
- Sharp increases or decreases in the cumulative return curve can signal high volatility or significant market events impacting the portfolio.
- The timing and magnitude of peaks and troughs can give insights into the portfolio's response to market conditions and the efficacy of the trading strategy employed.

8.2.2 Distribution of Hourly Returns

The distribution of hourly returns (Figure 16) provides a visual representation of the frequency and magnitude of the portfolio's returns over each hour within the backtesting period. It is a vital tool for understanding the risk characteristics of the trading strategy.

Analysis

- The histogram centers around zero, with tails extending to both positive and negative returns.
- A narrow and tall distribution indicates consistent returns with less variability, whereas a wide and flat distribution suggests high volatility.
- The presence of outliers or a long tail in one direction can indicate skewness in returns, implying an asymmetric risk profile.
- By examining the spread of the returns, one can assess the potential for extreme outcomes, either gains or losses.

8.2.3 Backtesting Strategy

The backtesting strategy is implemented in the `PortfolioBacktester` class. Here's a breakdown of its logic:

1. The portfolio starts with a defined amount of cash and no holdings.
2. Trading signals are processed as they occur, influencing the portfolio's positions and cash balance.
3. A transaction fee is applied to each trade, reducing the portfolio's cash balance.

4. The portfolio's value is updated based on the current market prices and the holdings after each signal is processed.

5. The transactions are recorded and can be saved to a file for review.

6. The cumulative return of the portfolio is plotted over time, and the distribution of hourly returns is analyzed to gauge performance and risk.

8.2.4 Conclusion

The backtest provides insights into the effectiveness of the trading strategy. Key performance metrics like the Sharpe Ratio and Max Drawdown are calculated to assess the risk-adjusted returns and the strategy's potential drawdown risk. These metrics are critical for understanding the trade-off between risk and reward in the tested strategy.

8.3 Task 4c

The backtest of the trading strategy has yielded several key performance metrics which are crucial for understanding the strategy's effectiveness and risk profile.

8.3.1 Sharpe Ratio

The Sharpe Ratio is a measure of the excess return per unit of risk in an investment asset or a trading strategy. Here's what the calculated ratios indicate:

- **Hourly Sharpe Ratio:** 0.0032
  This is a very low ratio, suggesting that the excess return of the strategy over the risk-free rate is minimal on an hourly basis.

- **Daily Sharpe Ratio:** 0.0157
  When scaled to daily returns, the Sharpe Ratio is still quite low, which means the strategy does not yield high excess returns compared to the daily risk incurred.

- **Annually Sharpe Ratio:** 0.2493
  Over an annual scale, the Sharpe Ratio is under 0.3, which is often considered suboptimal in traditional finance contexts. This indicates that the strategy's annual returns are not significantly higher than the risk-free rate given the volatility experienced.

8.3.2 Max Drawdown

The Maximum Drawdown is the maximum observed loss from a peak to a trough of a portfolio, before a new peak is attained. The Max Drawdown for this strategy is -27.53%, which indicates that at some point, the portfolio lost over a quarter of its value from the peak before recovering. This is a significant drawdown, reflecting a potentially high risk in the trading strategy.

8.3.3 Conclusion

The performance metrics indicate that the trading strategy might not be providing sufficient compensation for the risks it entails. The Sharpe Ratios at different scales are on the lower end, suggesting the excess returns are not substantial compared to the risks. Additionally, the Max Drawdown shows a considerable potential loss, which could be alarming for risk-averse investors. These insights highlight the importance of risk management and may prompt a review of the trading strategy to improve its risk-adjusted returns.

The performance of this strategy is not that good as in the paper. One possible reason is that we do not buy or sell sector ETF to hedge our portfolio. Also, we do not select stocks based on the kappa, some stocks may take a long time to reverse to its expected value. These may negatively affect the performance of our strategy.

8.4 Task 4c Extension 1: Upper Bound

Figure 17, 19, and 21 show the cumulative return when we have upper bound number of shares we can hold as 5, 10, and 20. Figure 18, 20, and 22 show the histogram of the hourly return respectively.

In the ongoing development of our trading strategy, a key consideration is the implementation of constraints on the maximum number of shares that can be held in a long or short position. This risk management technique, known as setting an 'upper bound', is designed to limit the potential exposure of the portfolio to any single asset, thereby potentially reducing the risk of substantial losses from outsized positions.

8.4.1 Rationale

Setting upper bounds on position sizes helps to:

- Diversify risk across a range of assets, rather than concentrating it.
- Mitigate the impact of extreme movements in individual asset prices on the overall portfolio.
- Ensure compliance with regulatory or self-imposed leverage and exposure limits.

8.4.2 Performance Metrics with Upper Bound Constraints

The table below summarizes the performance metrics of our trading strategy under different constraints for the maximum number of shares for any position. Each 'upper bound' scenario is tested to observe the effects on the Sharpe Ratio and the Maximum Drawdown.

Upper Bound	Hourly SR	Daily SR	Annually SR	Max Drawdown
5 Shares	0.0044	0.0215	0.3420	-0.6334
10 Shares	0.0061	0.0299	0.4745	-0.5669
20 Shares	0.0075	0.0365	0.5799	-0.5988
No constraint	0.0032	0.0157	0.2493	-0.2753

Table 1: Portfolio Performance Metrics with Upper Bounds on Position Sizes

8.4.3 Analysis

As the upper bounds on position sizes increase, we observe an increase in both the Sharpe Ratio and the Maximum Drawdown. While a higher Sharpe Ratio suggests a more favorable risk-adjusted return, the increased Maximum Drawdown indicates higher risk of loss. This suggests a trade-off where allowing larger positions can lead to higher returns but also greater risks.

8.4.4 Conclusion

The implementation of upper bounds on the number of shares for long or short positions is a crucial aspect of risk management in portfolio construction. The analysis indicates that while higher position size limits may enhance returns, they also increase the portfolio's risk profile. Careful consideration must be given to determine the appropriate balance between risk and return for the strategy's objectives.

In further refining our trading strategy, we have considered the impact of transaction costs on overall performance. Given that the strategy with an upper bound of 20 shares yielded the highest Sharpe Ratio, it was selected for further evaluation under varying transaction fee structures.

8.5 Task 4c Extension 2: Transaction Costs

Transaction costs can significantly affect the profitability of a trading strategy. They represent the costs of trading assets and can include brokerage fees, bid-ask spreads, slippage, and other operational costs. Even small fees can compound over numerous transactions, thus reducing the net return of a strategy.

8.5.1 Performance Metrics with Transaction Costs

The table below shows how the performance metrics of our strategy change with different levels of transaction fees applied:

Transaction Fee	Hourly SR	Daily SR	Annually SR	Max Drawdown
0	0.0075	0.0365	0.5799	-0.5988
0.001	0.0051	0.0252	0.3994	-0.6829
0.003	0.0009	0.0044	0.0691	-0.7759

Table 2: Portfolio Performance Metrics with Different Transaction Fees

8.5.2 Analysis of Transaction Cost Impact

The results demonstrate that the inclusion of transaction fees has a dampening effect on the Sharpe Ratio, indicating that transaction costs erode the risk-adjusted returns of the strategy. As the transaction fee increases, the Sharpe Ratio decreases, reflecting lower profitability per unit of risk. Moreover, the Maximum Drawdown becomes more severe with higher transaction fees, suggesting increased risk of substantial losses.

8.5.3 Concluding Insights

The analysis underscores the importance of accounting for transaction costs in the design and evaluation of trading strategies. While transaction fees may seem negligible on a per-trade basis, their cumulative effect can be substantial, especially for strategies that involve frequent trading. Thus, it is imperative to optimize the balance between trading frequency, position sizing, and transaction costs to enhance the risk-adjusted returns of the strategy.

9 Appendix

```python
# The main section of the script
if __name__ == "__main__":

    # Define time period for analysis and backtesting
    start_date = '2021-09-26T00:00:00+00:00'
    end_date = '2022-09-25T23:00:00+00:00'

    # Additional timestamps for specific analyses
    timestamp1 = '2021-09-26T12:00:00+00:00'
    timestamp2 = '2022-04-15T20:00:00+00:00'

    # Define time period for calculating S-scores
    s_score_start_date = '2021-09-26T00:00:00+00:00'
    s_score_end_date = '2021-10-25T23:00:00+00:00'

    # Set thresholds for different trading signals
    thresholds = {
        's_bo': 1.25,  # Threshold for buying to open
        's_so': 1.25,  # Threshold for selling to open
        's_bc': 0.75,  # Threshold for closing short positions
        's_sc': 0.5    # Threshold for closing long positions
    }

    # Define the tokens to analyze
    tokens = ['BTC', 'ETH']
```

Figure 1: The Input of Codes

```
E:\Gatech\ISYE 6767\Final Project>python main.py
Generating Signals:   4%|█                                    | 322/8760 [01:06<31:38,  4.44it/s]E
:\Gatech\ISYE 6767\Final Project\Analyzer.py:233: RuntimeWarning: invalid value encountered in sqrt
  sigma_eq = np.sqrt(np.var(ou_model.resid) / (1 - b**2))
Generating Signals:  11%|███                                  | 993/8760 [03:26<25:07,  5.15it/s]E
:\Gatech\ISYE 6767\Final Project\Analyzer.py:233: RuntimeWarning: invalid value encountered in sqrt
  sigma_eq = np.sqrt(np.var(ou_model.resid) / (1 - b**2))
Generating Signals: 100%|████████████████████████████████████| 8760/8760 [30:03<00:00,  4.86it/s]
Determining Signals: 100%|███████████████████████████████████| 8760/8760 [00:29<00:00, 296.92it/s]
Generating Signals:  45%|████████████████                     | 322/720 [01:17<02:07,  3.12it/s]E
:\Gatech\ISYE 6767\Final Project\Analyzer.py:233: RuntimeWarning: invalid value encountered in sqrt
  sigma_eq = np.sqrt(np.var(ou_model.resid) / (1 - b**2))
Generating Signals: 100%|█████████████████████████████████████| 720/720 [03:11<00:00,  3.75it/s]
Doing Backtest: 100%|████████████████████████████████████████| 8760/8760 [00:57<00:00, 151.32it/s]
Hourly Sharpe Ratio: 0.0032053940045977535
Daily Sharpe Ratio: 0.01570315947168178
Annually Sharpe Ratio: 0.24927992856035056
Max Drawdown: -0.27527479471533167
```

Figure 2: The Output of Codes

	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect
Token	1INCH	AAVE	ALGO	APE	ATOM	AUDIO	AVAX	AXS	BAT	BCH	BIT	BNB	BNT	BTC
TimeStamp														
2021-09-26 00:00:00	-0.197546					-0.088025		-0.132269		-0.194473		-0.200654		-0.191166
2021-09-26 01:00:00	-0.197799					-0.090223		-0.131877		-0.19426		-0.200617		-0.190795
2021-09-26 02:00:00	-0.197315					-0.090717		-0.130907		-0.193895		-0.200234		-0.190215
2021-09-26 03:00:00	-0.198169					-0.095537		-0.134676		-0.191813		-0.199222		-0.189008
2021-09-26 04:00:00	-0.198796					-0.096307		-0.134431		-0.192421		-0.199719		-0.188875
2021-09-26 05:00:00	-0.201597					-0.097016		-0.135159		-0.194973		-0.20262		-0.190991
2021-09-26 06:00:00	-0.201995					-0.096706		-0.134964		-0.195268		-0.203062		-0.191004
2021-09-26 07:00:00	-0.201833					-0.096633		-0.134907		-0.19523		-0.202964		-0.190983
2021-09-26 08:00:00	-0.201462					-0.097258		-0.137148		-0.194945		-0.201457		-0.190806
2021-09-26 09:00:00	-0.204025					-0.104746		-0.139921		-0.199319		-0.204675		-0.189579
2021-09-26 10:00:00	-0.204703					-0.104341		-0.142036		-0.201152		-0.206008		-0.190608
2021-09-26 11:00:00	-0.205784					-0.104485		-0.140943		-0.200159		-0.204633		-0.190377
2021-09-26 12:00:00	-0.205629					-0.104683		-0.140866		-0.199968		-0.204477		-0.190112
2021-09-26 13:00:00	-0.205134					-0.105062		-0.140884		-0.200169		-0.204072		-0.190123
2021-09-26 14:00:00	-0.205256					-0.103868		-0.139976		-0.200915		-0.204523		-0.190062
2021-09-26 15:00:00	-0.205743					-0.104433		-0.139901		-0.201577		-0.204914		-0.190084
2021-09-26 16:00:00	-0.205489					-0.108121		-0.139819		-0.201491		-0.204902		-0.190296
2021-09-26 17:00:00	-0.205501					-0.131582		-0.13859		-0.201135		-0.205283		-0.188762
2021-09-26 18:00:00	-0.204983					-0.141813		-0.138329		-0.201973		-0.205854		-0.186683
2021-09-26 19:00:00	-0.205276					-0.141892		-0.138341		-0.202382		-0.206103		-0.18679
2021-09-26 20:00:00	-0.204807					-0.14213		-0.138938		-0.202805		-0.206303		-0.186644

Figure 3: The Screenshot of Part of First Eigenvector

	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect	eigenvect
Token	1INCH	AAVE	ALGO	APE	ATOM	AUDIO	AVAX	AXS	BAT	BCH	BIT	BNB	BNT	BTC
TimeStamp														
2021-09-26 00:00:00	-0.004101					0.3634515		-0.050279		0.019705		0.0380847		-0.054992
2021-09-26 01:00:00	-0.002587					0.3523778		-0.045483		0.0159877		0.0356092		-0.059982
2021-09-26 02:00:00	-0.005139					0.3412101		-0.042758		0.0116973		0.0314455		-0.066767
2021-09-26 03:00:00	-0.001083					0.342231		-0.006399		-0.00154		0.0271748		-0.06939
2021-09-26 04:00:00	-0.000295					0.3457046		-0.006657		-0.002942		0.0261839		-0.06934
2021-09-26 05:00:00	0.0044388					0.3520053		-0.026462		-0.005763		0.0307452		-0.068796
2021-09-26 06:00:00	0.005822					0.3458806		-0.024576		-0.003392		0.0299205		-0.068378
2021-09-26 07:00:00	0.0043967					0.3445749		-0.028231		-0.002931		0.0304337		-0.067925
2021-09-26 08:00:00	-0.003014					0.3211634		-0.024187		0.0039305		0.0452849		-0.066252
2021-09-26 09:00:00	-0.026456					-0.292267		-0.046747		-0.030535		-0.00865		0.0615274
2021-09-26 10:00:00	-0.046253					-0.277637		-0.034264		-0.035494		-0.013907		0.0581615
2021-09-26 11:00:00	-0.049046					-0.281737		-0.030071		-0.03236		-0.009759		0.0615671
2021-09-26 12:00:00	-0.047783					-0.265622		-0.03648		-0.033804		-0.007495		0.0516875
2021-09-26 13:00:00	-0.039075					-0.288982		-0.039508		-0.034698		-0.007301		0.0528841
2021-09-26 14:00:00	-0.038648					-0.290593		-0.038935		-0.036059		-0.008062		0.053552
2021-09-26 15:00:00	-0.035697					-0.292796		-0.040772		-0.033704		-0.006456		0.0543486
2021-09-26 16:00:00	-0.033949					-0.302954		-0.040622		-0.03199		-0.007196		0.0566022
2021-09-26 17:00:00	-0.058025					-0.166194		-0.055548		-0.038567		0.0081915		0.0489542
2021-09-26 18:00:00	-0.060043					-0.138537		-0.051993		-0.02899		0.0150561		0.0444196
2021-09-26 19:00:00	-0.056009					-0.136051		-0.055047		-0.02549		0.0152856		0.0448314
2021-09-26 20:00:00	-0.058002					-0.133719		-0.054722		-0.025465		0.0153044		0.0444737
2021-09-26 21:00:00	-0.056907					-0.143625		-0.053539		-0.024393		0.0165382		0.0501946
2021-09-26 22:00:00	0.0638388					0.1094436		0.053668		0.0260288		-0.017319		-0.043764
2021-09-26 23:00:00	0.0638537					0.1077073		0.0515236		0.0268888		-0.01648		-0.04052
2021-09-27 00:00:00	0.0659945					0.1043212		0.0490478		0.0282325		-0.017601		-0.038309
2021-09-27 01:00:00	0.0654719					0.1026995		0.0492858		0.0343511		-0.015103		-0.037191
2021-09-27 02:00:00	0.0668097					0.093193		0.07047		0.0436728		-0.026105		-0.034035
2021-09-27 03:00:00	0.0612895					0.0892184		0.0616091		0.0344084		-0.02875		-0.037425
2021-09-27 04:00:00	0.060764					0.0921584		0.0630706		0.0344358		-0.028474		-0.036187

Figure 4: The Screenshot of Part of Second Eigenvector

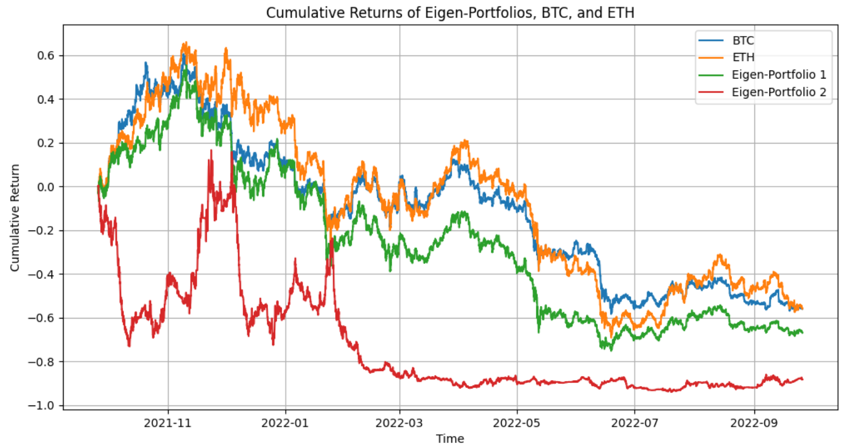

Figure 5: The Cumulative Return of the 4 Assets

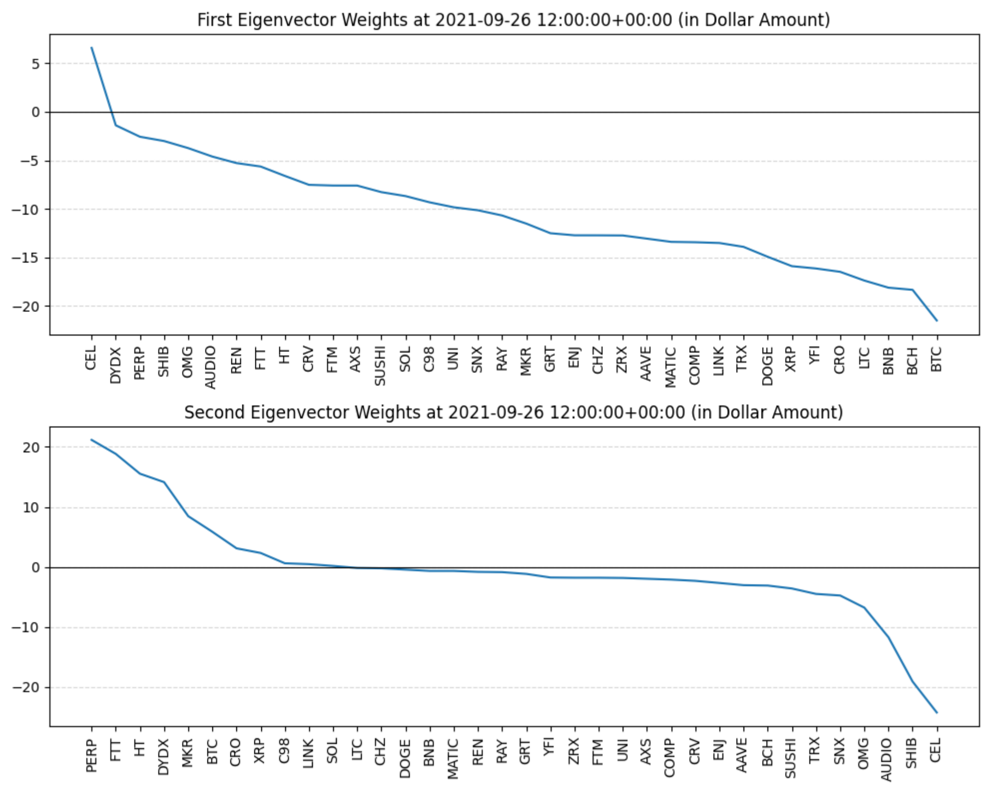

Figure 6: The Eigen Portfolio Weights at 2021-09-26T12 (in Dollar Amount)

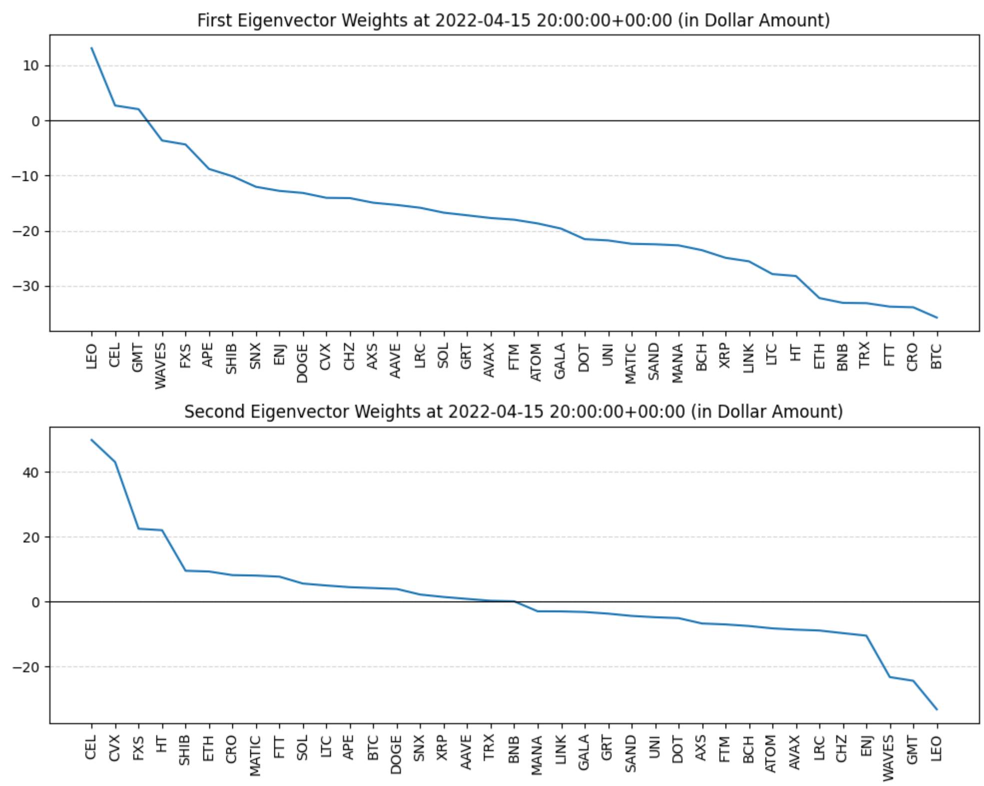

Figure 7: The Eigen Portfolio Weights at 2022-04-15T20 (in Dollar Amount)

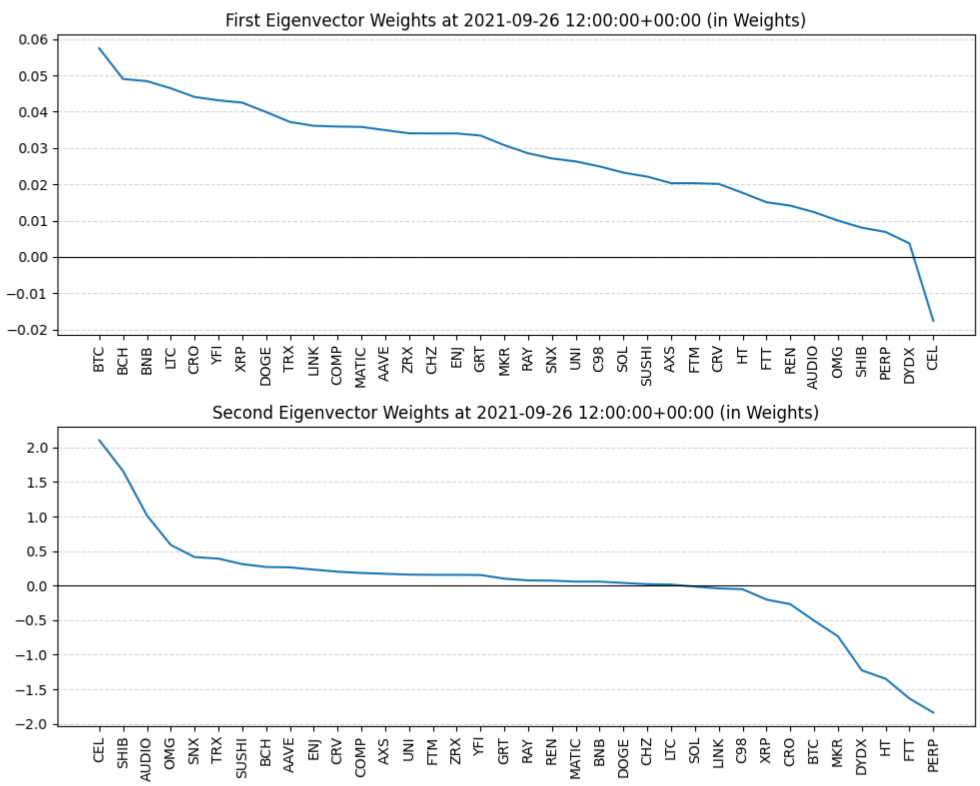

Figure 8: The Eigen Portfolio Weights at 2021-09-26T12 (in Weights)

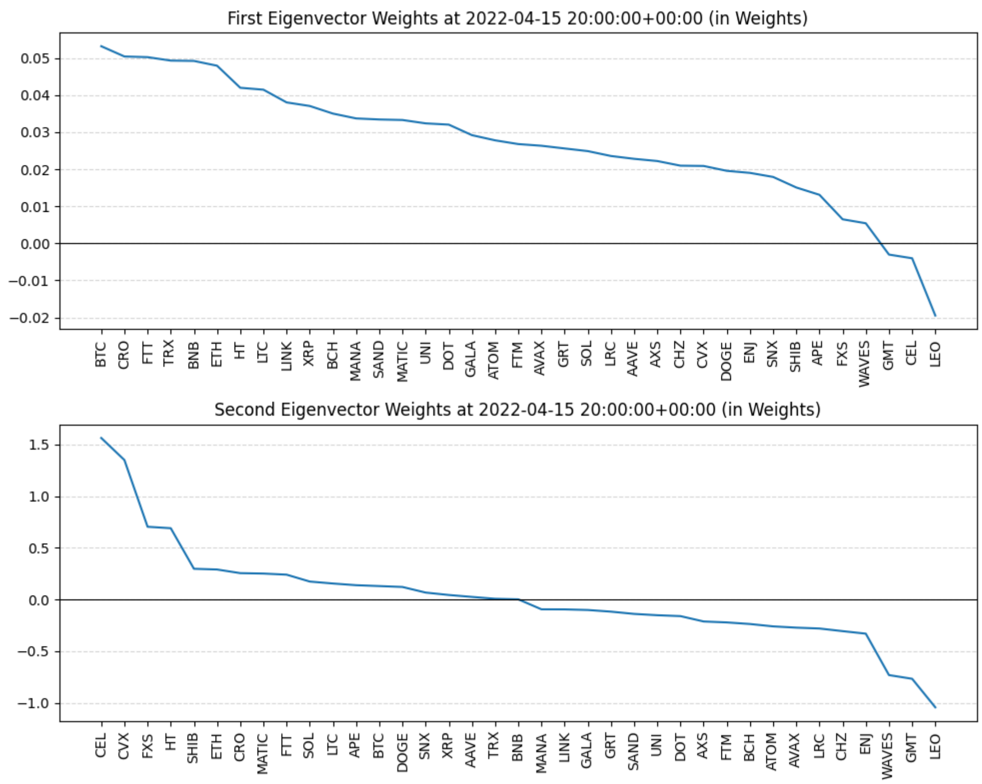

Figure 9: The Eigen Portfolio Weights at 2022-04-15T20 (in Weights)

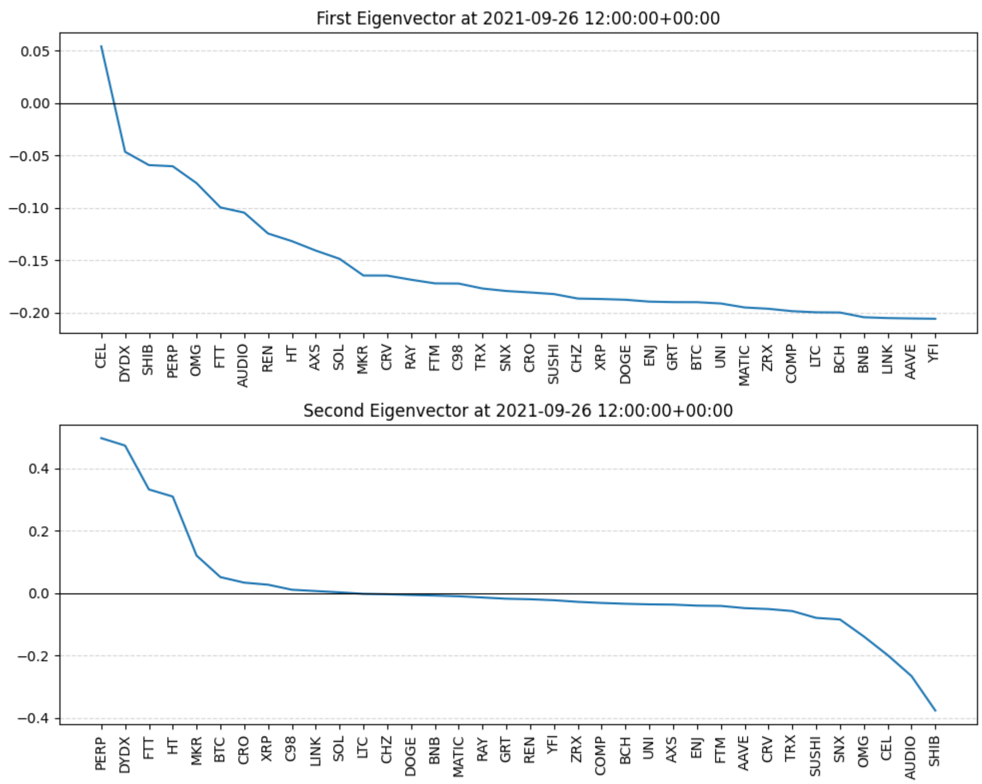

Figure 10: The Eigenvector at 2021-09-26T12

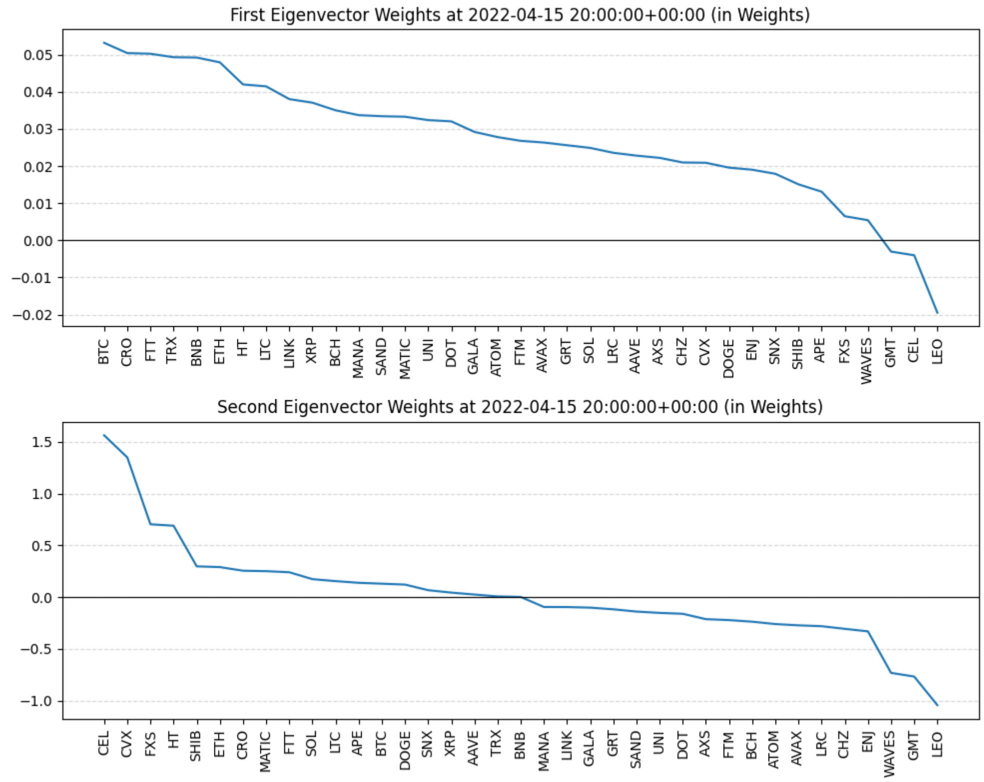

Figure 11: The Eigenvector at 2022-04-15T20

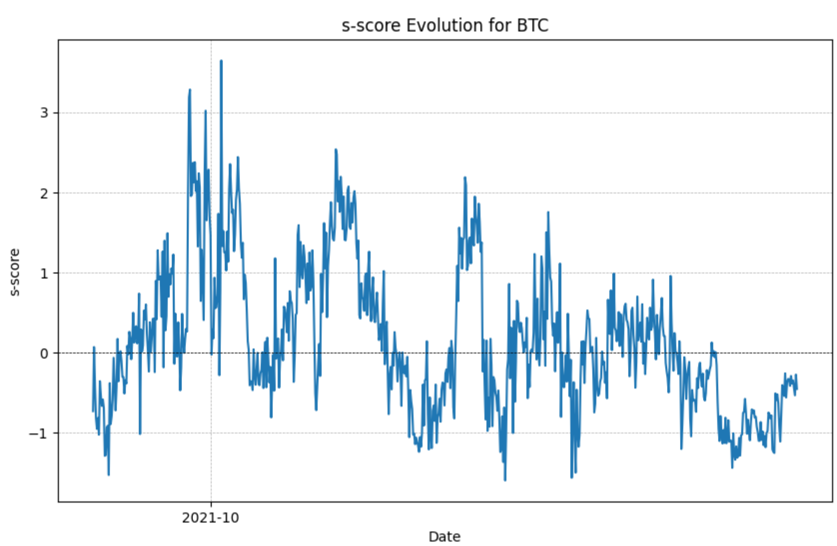

Figure 12: The S-score for BTC

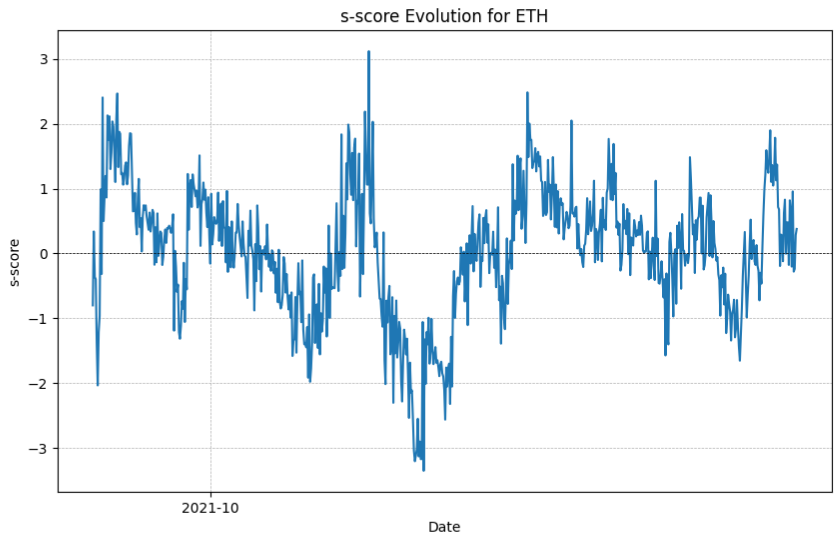

Figure 13: The S-score for ETH

Token	Signal 1INCH	Signal AAVE	Signal ALGO	Signal APE	Signal ATOM	Signal AUDIO	Signal AVAX	Signal AXS	Signal BAT	Signal BCH	Signal BIT	Signal BNB	Signal BNT	Signal BTC	Signal C98	Signal CEL
Timestamp																
2021-09-26 00:00:00	hold					buy to open		hold		hold		hold		hold	hold	buy to ope
2021-09-26 01:00:00	sell to open					buy to open		hold		hold		hold		hold	hold	buy to ope
2021-09-26 02:00:00	hold					buy to open		hold		hold		hold		hold	hold	buy to ope
2021-09-26 03:00:00	sell to open					hold		hold		hold		buy to open		hold	buy to op	buy to ope
2021-09-26 04:00:00	hold					hold		hold		hold		buy to open		hold	buy to op	buy to ope
2021-09-26 05:00:00	sell to open					hold		hold		hold		buy to open		hold	buy to op	buy to ope
2021-09-26 06:00:00	hold					hold		hold		hold		hold		hold	hold	buy to ope
2021-09-26 07:00:00	sell to open					hold		hold		hold		hold		hold	buy to op	buy to ope
2021-09-26 08:00:00	sell to open					hold		hold		hold		buy to open		hold	buy to op	buy to ope
2021-09-26 09:00:00	hold					close long position		hold		hold		buy to open		hold	buy to op	close lon
2021-09-26 10:00:00	sell to open					buy to open		hold		hold		buy to open		hold	buy to op	hold
2021-09-26 11:00:00	sell to open					buy to open		hold		hold		buy to open		hold	buy to op	hold
2021-09-26 12:00:00	sell to open					hold		hold		hold		buy to open		buy to op	buy to op	hold
2021-09-26 13:00:00	sell to open					buy to open		hold		hold		buy to open		hold	buy to op	hold
2021-09-26 14:00:00	sell to open					buy to open		hold		hold		buy to open		hold	buy to op	hold
2021-09-26 15:00:00	sell to open					hold		hold		hold		buy to open		hold	buy to op	hold
2021-09-26 16:00:00	sell to open					close long position		hold		hold		buy to open		buy to op	buy to op	hold
2021-09-26 17:00:00	sell to open					hold		hold		hold		buy to open		close lon	buy to op	hold
2021-09-26 18:00:00	sell to open					hold		hold		buy to open		buy to open		hold	buy to op	hold
2021-09-26 19:00:00	sell to open					hold		hold		buy to open		buy to open		hold	buy to op	hold
2021-09-26 20:00:00	sell to open					hold		hold		hold		hold		hold	buy to op	hold
2021-09-26 21:00:00	sell to open					hold		hold		buy to open		buy to open		hold	buy to op	hold
2021-09-26 22:00:00	sell to open					hold		hold		buy to open		buy to open		hold	buy to op	hold
2021-09-26 23:00:00	sell to open					hold		hold		buy to open		buy to open		hold	buy to op	hold
2021-09-27 00:00:00	sell to open					hold		hold		buy to open		buy to open		hold	buy to op	hold

Figure 14: The Screenshot of Part of Signals

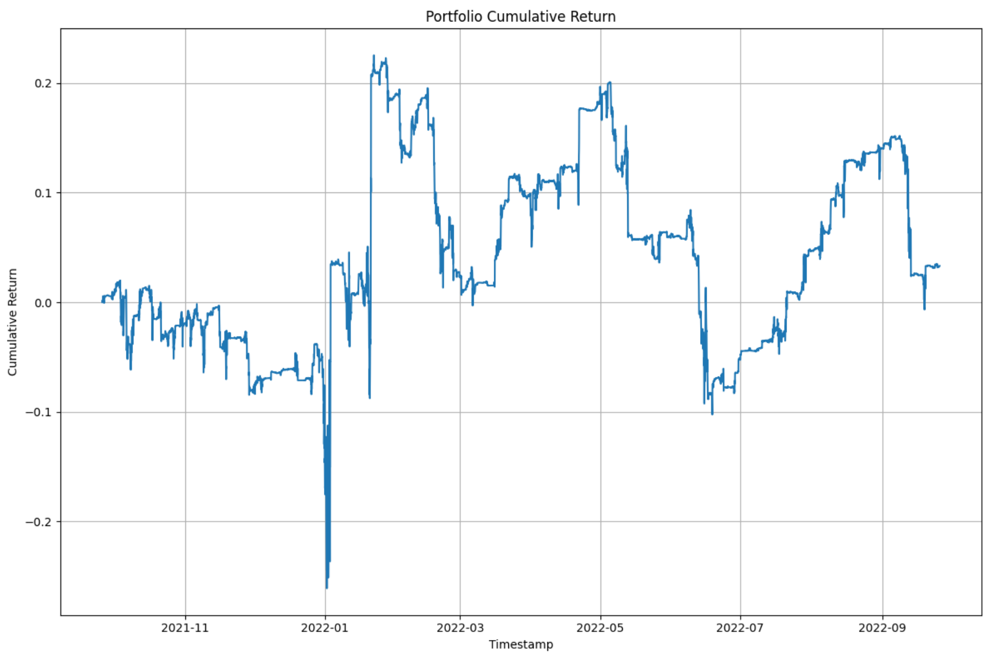

Figure 15: Cumulative Return of the Portfolio

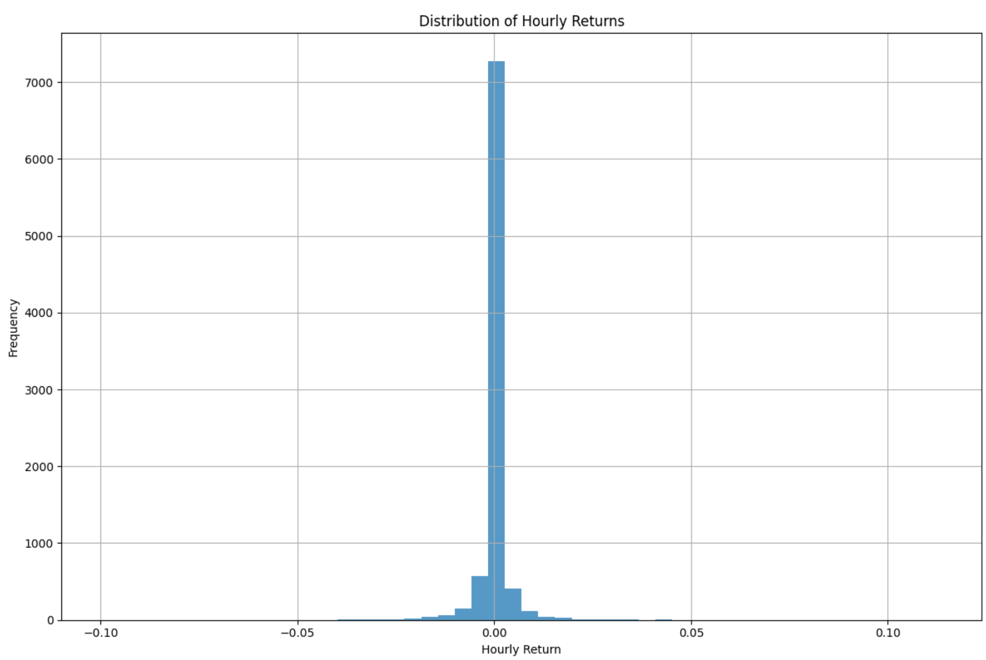

Figure 16: Distribution of Hourly Returns

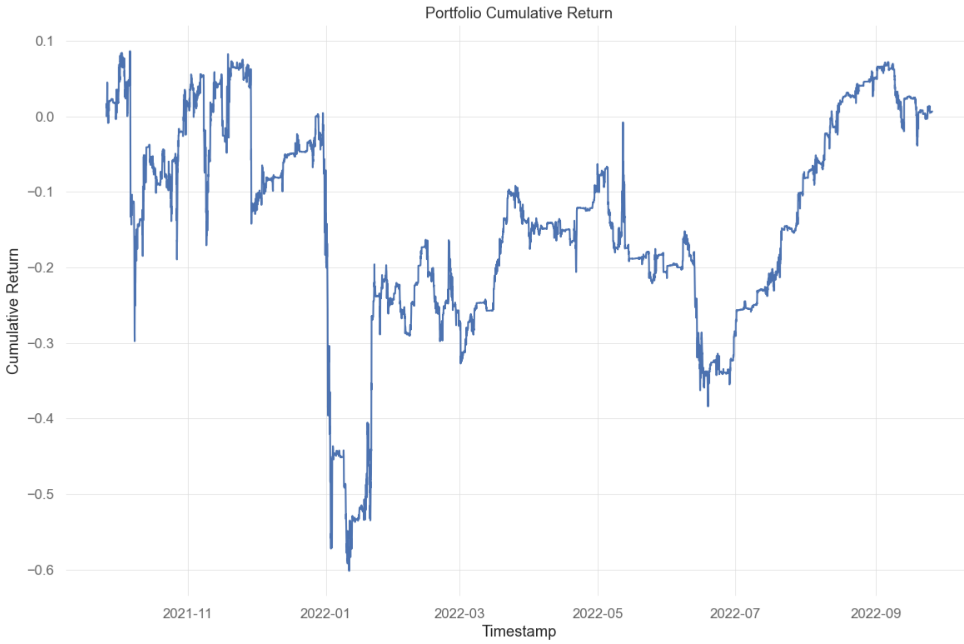

Figure 17: Cumulative Return of the Portfolio with Max Share 5

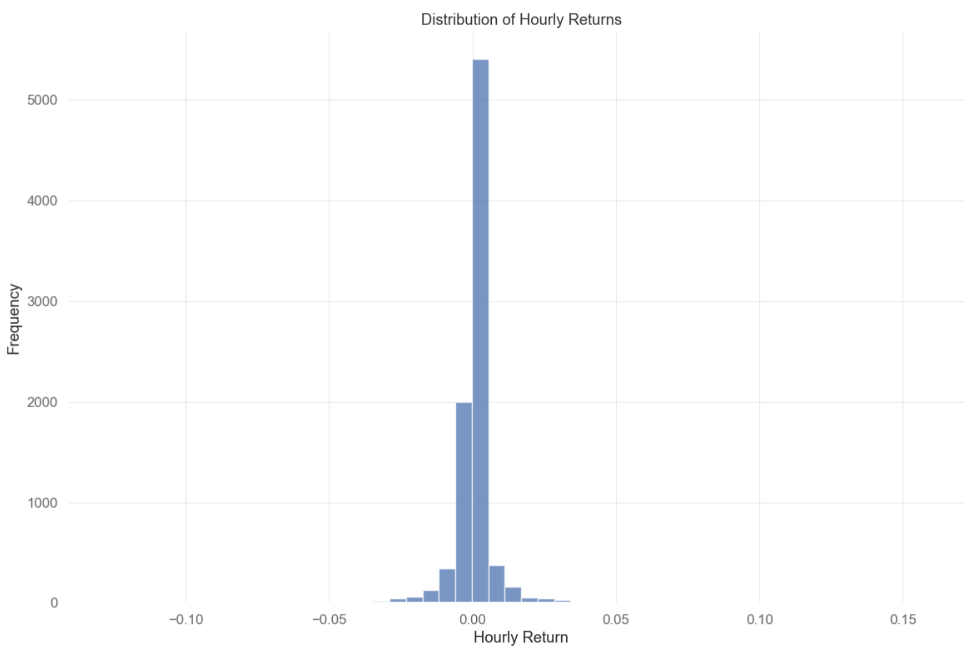

Figure 18: Distribution of Hourly Returns with Max Share 5

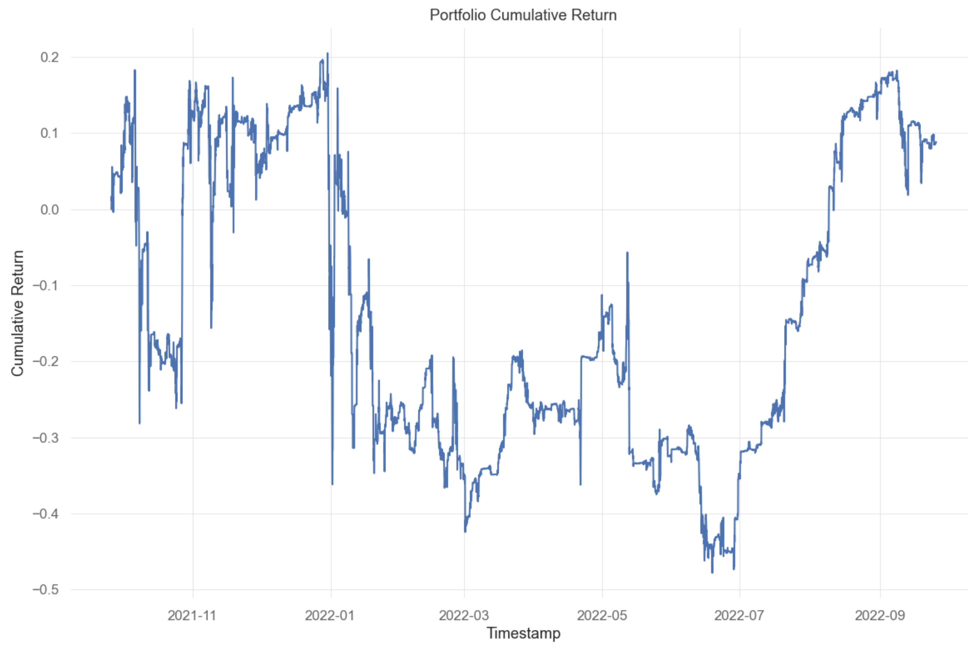

Figure 19: Cumulative Return of the Portfolio with Max Share 10

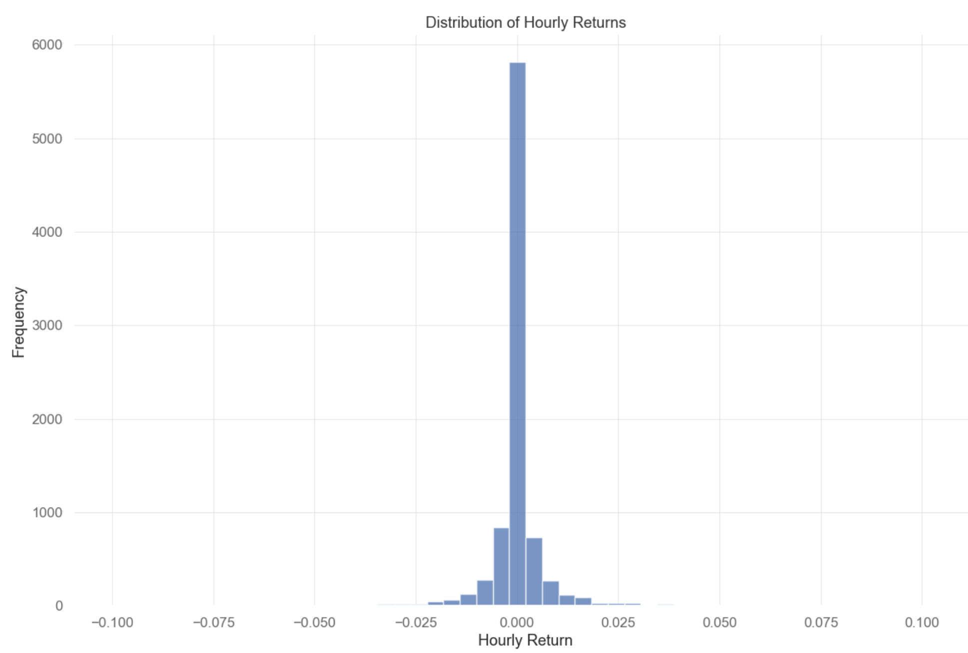

Figure 20: Distribution of Hourly Returns with Max Share 10

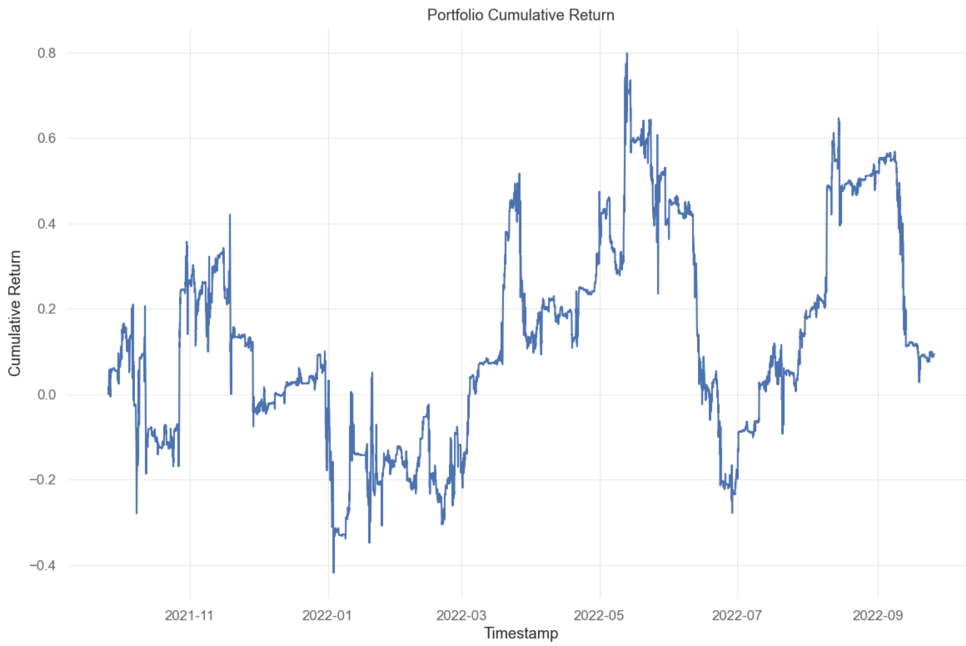

Figure 21: Cumulative Return of the Portfolio with Max Share 20

Figure 22: Distribution of Hourly Returns with Max Share 20