Final Project: Implementing a Statistical Arbitrage Strategy

ISYE 6767: Design and Implementation of Computational Finance Models

Fall 2024

Author: Tianyi Zhu
GT ID: 903880510

Contents

1 Introduction

1.1 Problem Statement

This project focuses on implementing and analyzing a statistical arbitrage strategy in the cryptocurrency market. The strategy, inspired by eigenportfolio techniques, applies Principal Component Analysis (PCA) to derive eigenportfolios and risk factors for generating trading signals. The goal is to assess the profitability and robustness of this approach in a dynamic and volatile market.

1.2 Data Source

The dataset used for this project includes hourly price data for over 120 cryptocurrencies obtained from FTX. The analysis period spans from February 19, 2021, to September 26, 2022. Two key datasets, *coins_all_prices.csv* and *coins_universe_150K_40.csv*, provide the necessary price data and coin selection information. The project focuses on 40 cryptocurrencies with the highest market capitalization for analysis.

1.3 Project Objectives

The primary objectives of this project are as follows:

1. Compute eigenportfolio weights for the top two eigenvalues of the empirical correlation matrix derived from normalized returns of 40 cryptocurrencies.
2. Generate trading signals based on calculated statistical metrics, particularly s-scores.
3. Evaluate the strategy's performance using key financial metrics, including the Sharpe ratio and Maximum Drawdown (MDD).
4. Visualize the strategy's cumulative return and hourly return distribution.

1.4 Significance of the Study

The growing interest in cryptocurrencies as an alternative asset class highlights the need for innovative quantitative trading strategies. By exploring eigenportfolios and s-scores, this project aims to provide insights into the application of statistical arbitrage in highly volatile markets. Additionally, the study demonstrates the practical implementation of advanced portfolio management techniques.

2 Strategy

2.1 Overview

The proposed strategy utilizes Principal Component Analysis (PCA) to derive eigenportfolios and incorporates statistical arbitrage techniques based on the Ornstein-Uhlenbeck (OU) process. This strategy aims to capitalize on market inefficiencies in the cryptocurrency market by analyzing normalized returns, residuals, and s-scores for the top-performing cryptocurrencies.

2.2 Key Components of the Strategy

1. **Data Preprocessing:** The strategy begins by loading and cleaning price and coin data from CSV files. Timestamp conversion is performed to align data with the chosen start date. Missing data is handled using forward filling, and the dataset is filtered based on a rolling window size.

2. **Normalization and Rolling Window:** Returns are calculated as percentage changes in prices, followed by normalization. The normalized returns are then analyzed using rolling windows to capture temporal changes in the cryptocurrency market.

3. **Eigenportfolio Construction:** Using the rolling window data, the empirical correlation matrix is computed. PCA is applied to this matrix to extract the eigenvalues and eigenvectors. The top two eigenportfolios are constructed, normalized by the standard deviation of the returns.

4. **Factor Returns and Residuals:** Factor returns are computed by projecting the returns onto the eigenportfolios. Regression analysis is performed to estimate residuals and calculate parameters for the OU process, such as mean reversion rate, long-term mean, and volatility.

5. **Signal Generation:** Based on the OU process, s-scores are computed for each cryptocurrency. Trading signals are generated based on predefined thresholds:
   - **Buy (Long):** S-score below -1.4
   - **Sell (Short):** S-score above 1.15
   - **Close Buy (Close Long):** S-score below 0.75
   - **Close Sell (Close Short):** S-score above -0.5

6. **Cleaning and Visualization:** The generated signals are stored in a strategy actions DataFrame. Columns with missing values are dropped to ensure clean execution. Cumulative growth of the eigenportfolios is plotted to visualize their performance over time.

2.3 Mathematical Framework

Ornstein-Uhlenbeck Process: The residuals of the regression are modeled using the OU process, defined as:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t,$$

where θ is the rate of mean reversion, μ is the long-term mean, σ is the volatility, and dW_t represents the Wiener process.

S-Score Calculation: The s-score is computed as:

$$S = \frac{\text{Residual} - \mu}{\sigma_{\text{equilibrium}}},$$

where $\sigma_{\text{equilibrium}}$ is the equilibrium standard deviation derived from the OU parameters.

2.4 Implementation Highlights

The strategy is implemented as a Python class, `CryptoStrategy`, which encapsulates all functionalities, including data preprocessing, eigenportfolio construction, residual estimation, and signal generation. Modular functions ensure flexibility and scalability for analyzing other datasets or extending the model.

2.5 Expected Outcomes

By leveraging eigenportfolios and statistical arbitrage, the strategy aims to:

- Identify profitable opportunities in the cryptocurrency market.
- Minimize risk through diversification and mean-reverting behavior of residuals.
- Provide a robust framework for applying PCA and statistical models in volatile markets.

3 Task 1: Eigenportfolio Analysis and Cumulative Returns

3.1 Objective

The primary goal of this task is to compute the eigenportfolio weights corresponding to the two largest eigenvalues of the empirical correlation matrix of the top 40 tokens for each hour over the testing period. The eigenvectors corresponding to the largest and second-largest eigenvalues are saved into separate CSV files, with rows indexed by timestamp and columns representing the tokens. Additionally, we plot the cumulative return curves for four assets over the testing period: the first eigenportfolio, the second eigenportfolio, BTC, and ETH.

3.2 Methodology

3.2.1 Correlation Matrix and Eigenportfolio Computation

For each hourly interval during the testing period:

1. The top 40 tokens are selected based on their non-missing return data in the normalized return matrix.
2. An empirical correlation matrix of these tokens is computed using their returns over a rolling window.
3. Principal Component Analysis (PCA) is applied to the correlation matrix to compute eigenvectors and eigenvalues.
4. The eigenvectors corresponding to the largest and second-largest eigenvalues are normalized to represent eigenportfolio weights.

The eigenportfolio weights for the two largest eigenvalues are stored in two separate CSV files. Each CSV file is indexed by timestamp and contains columns representing the token weights.

3.2.2 Cumulative Return Calculation

The cumulative returns for the first and second eigenportfolios, BTC, and ETH are computed over the testing period. For eigenportfolios:

- The portfolio return is calculated as the weighted sum of token returns, where weights are derived from the eigenvectors.
- Cumulative returns are obtained by compounding the daily portfolio returns over time.

3.2.3 Visualization

The cumulative return curves for the first eigenportfolio, the second eigenportfolio, BTC, and ETH are plotted in a single figure for comparison. This visualization highlights the performance of the eigenportfolios relative to BTC and ETH.

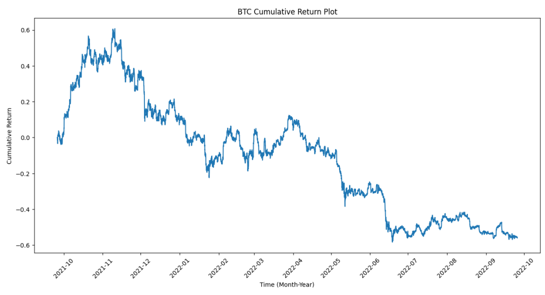


Figure 1: BTC Cumulative Return

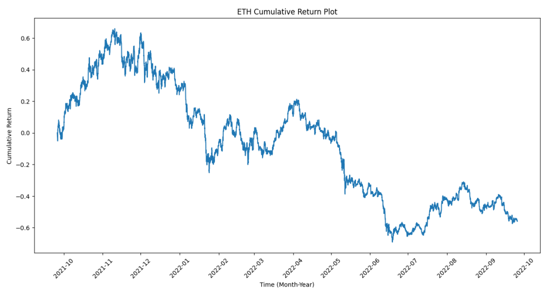


Figure 2: ETH Cumulative Return

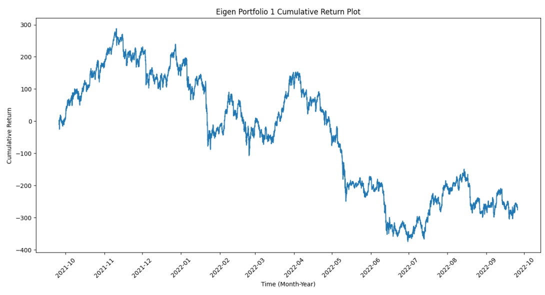


Figure 3: Eigen Portfolio 1

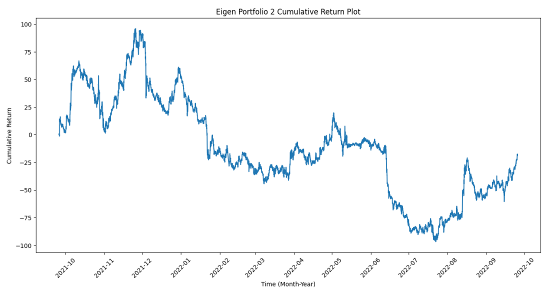


Figure 4: Eigen Portfolio 2

3.3 Results

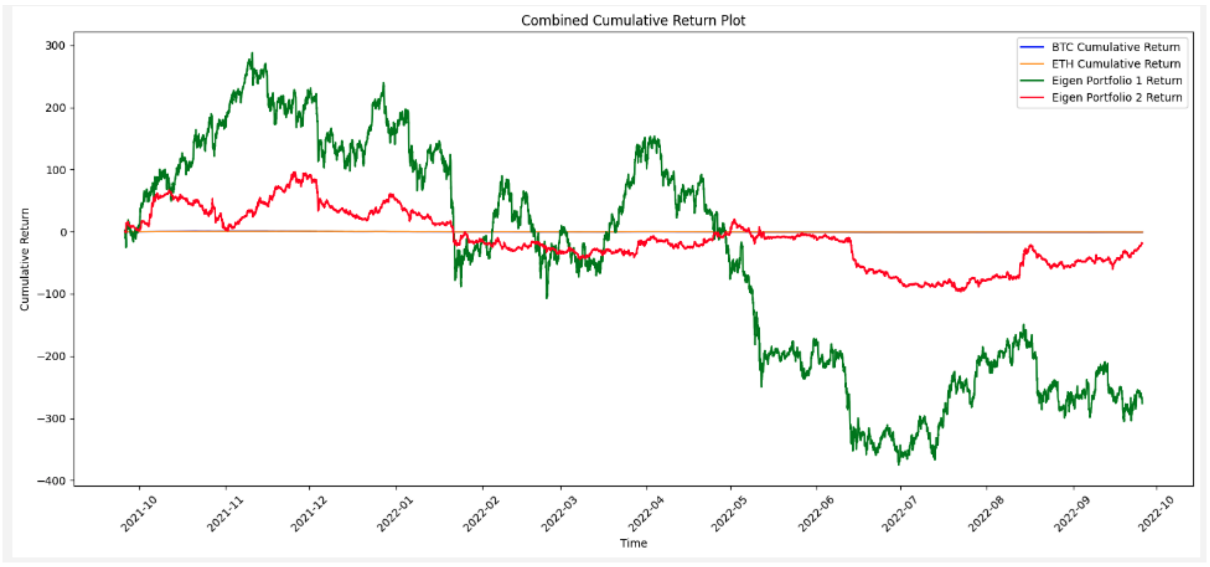


Figure 5: Cumulative return curves for BTC, ETH, and the first two eigenportfolios over the testing period.

Figure 5 illustrates the cumulative return curves for BTC, ETH, and the eigenportfolios. The first eigenportfolio demonstrates a distinct trend, capturing the dominant factor driving market returns, while the second eigenportfolio represents a secondary, orthogonal factor.

3.4 Discussion

The eigenportfolios provide insights into the latent factors influencing the cryptocurrency market. The first eigenportfolio corresponds to the market-wide trend, while the second eigenportfolio captures a more nuanced, sector-specific trend. Comparing their performance with BTC and ETH highlights the eigenportfolios' ability to capture diverse risk-return dynamics.

4 Task 2: Analysis of Eigen-Portfolio Weights at Key Timestamps

In this task, we examine the eigen-portfolio weights associated with the largest two eigenvalues of the empirical correlation matrix at two specific timestamps: 2021-09-26T12:00:00+00:00 and 2022-04-15T20:00:00+00:00. These timestamps were chosen to highlight distinct market conditions, reflecting potential shifts in token dynamics over time. The eigen-portfolio weights are sorted from largest to smallest for clarity and plotted to provide visual insights into the contributions of individual tokens to the principal components.

4.1 Methodology

The eigen-portfolio weights were computed as follows:

- The empirical correlation matrix was derived from the returns of the top 40 tokens based on data availability and trading volume.
- Principal Component Analysis (PCA) was applied to extract the eigenvalues and eigenvectors of the correlation matrix. The first two eigenvectors, corresponding to the largest and second-largest eigenvalues, represent the directions of maximum variance in the data.
- Eigen-portfolio weights were calculated as the normalized coefficients of the eigenvectors, indicating the relative importance of each token in the corresponding eigen-portfolios.
- For each timestamp, the weights were sorted in descending order to identify the most significant tokens contributing to the eigen-portfolios.

The analysis was conducted separately for the two timestamps to capture any temporal changes in the structure of the token market.

4.2 Results for 2021-09-26T12:00:00+00:00

Figures 6 and 7 display the eigen-portfolio weights for the first and second eigenvectors, respectively, at 2021-09-26T12:00:00+00:00. The key observations are as follows:

- **Eigenvector 1:** YFI exhibited the highest positive weight, suggesting its dominant influence in the first principal component. CEL had the most negative weight, indicating its contrasting behavior relative to YFI and other tokens.
- **Eigenvector 2:** CEL demonstrated an exceptionally high positive weight, overshadowing other tokens in the second principal component. This highlights CEL's unique role in the market dynamics at this timestamp.

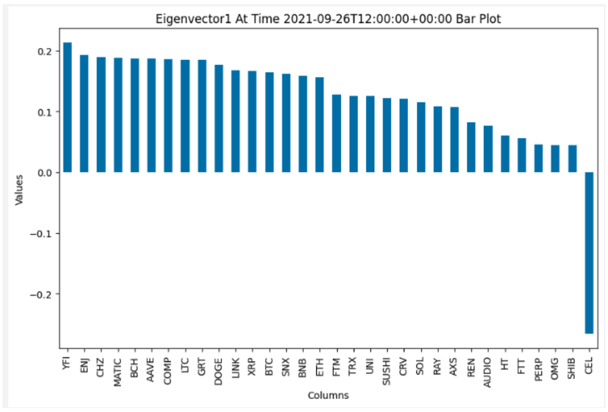


Figure 6: Eigenvector 1 Weights at 2021-09-26T12:00:00+00:00

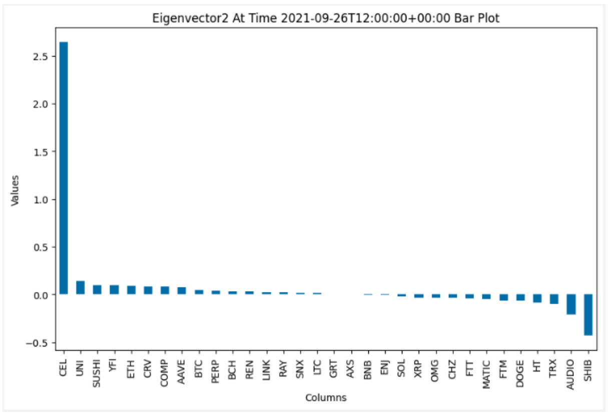


Figure 7: Eigenvector 2 Weights at 2021-09-26T12:00:00+00:00

4.3 Results for 2022-04-15T20:00:00+00:00

Figures 8 and 9 show the eigen-portfolio weights for the first and second eigenvectors, respectively, at 2022-04-15T20:00:00+00:00. The main findings include:

- **Eigenvector 1:** CRO was the most significant token with the highest positive weight, while WAVES had the most negative weight, indicating opposing behaviors in the first principal component.
- **Eigenvector 2:** HT and LTC emerged as the tokens with the highest positive contributions, whereas SHIB displayed the largest negative weight, reflecting its distinct role in the second principal component.

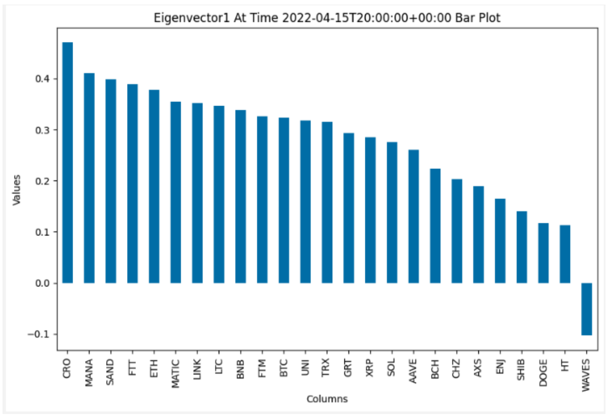


Figure 8: Eigenvector 1 Weights at 2022-04-15T20:00:00+00:00

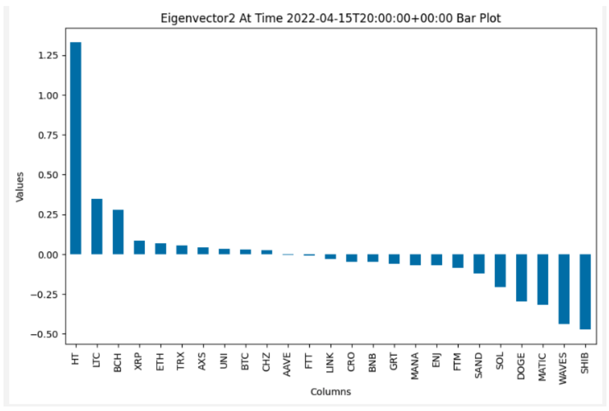


Figure 9: Eigenvector 2 Weights at 2022-04-15T20:00:00+00:00

4.4 Interpretation of Results

The eigen-portfolio weights offer insights into the market structure and token interactions at different times:

- The first eigenvector generally captures broad market trends or common factors driving token returns. Tokens with high positive weights are strongly aligned with this trend, while tokens with large negative weights are inversely related.
- The second eigenvector highlights idiosyncratic behaviors or secondary factors. Tokens with extreme weights in this eigenvector often exhibit unique market dynamics or idiosyncratic risks.
- The temporal variation in eigen-portfolio weights between the two timestamps suggests a shift in market structure, potentially driven by macroeconomic events, token-specific developments, or changes in investor sentiment.

4.5 Significance of Analysis

This analysis aligns with the methodology proposed by Avellaneda and Lee (2010) and demonstrates how PCA can uncover latent factors in token markets. By examining the eigen-portfolio weights, we gain a deeper understanding of the key drivers of market behavior and the relative importance of individual tokens.

5 Task 3: Evolution of S-Score for BTC and ETH

5.1 Overview

The S-score is a critical statistical metric designed to capture deviations of an asset's return from its expected behavior, based on the residuals from regression analysis using eigen-portfolios derived in Task 1. This metric not only identifies anomalies in asset returns but also serves as a potential signal for trading decisions. In this task, we analyze the evolution of the S-scores for Bitcoin (BTC) and Ethereum (ETH) over the testing period from `2021-09-26 00:00:00` to `2021-10-25 23:00:00`.

The primary objective is to identify patterns, trends, and abnormalities in the S-scores and explore their implications for understanding the behavior of these major cryptocurrencies in the context of market dynamics.

5.2 Methodology

The calculation of S-scores follows the methodology outlined in Section 3, which involves:

1. Computing the residuals of BTC and ETH by regressing their returns on the factors derived from the two principal eigen-portfolios.
2. Standardizing the residuals using the parameters of an Ornstein-Uhlenbeck (OU) process to compute the S-scores.
3. Aggregating S-scores at hourly intervals over the testing period to provide a temporal view of deviations.

The resulting S-scores are plotted to reveal their temporal dynamics, shedding light on periods of overperformance or underperformance relative to expectations derived from the eigen-portfolios.

5.3 Results

Figures 10 and 11 present the evolution of the S-scores for BTC and ETH, respectively. The results demonstrate significant variability in the S-scores, reflecting both assets' inherent volatility and market conditions during the testing period.

- **BTC S-Score:** The BTC S-score displays multiple instances of extreme values, reaching as high as 4.0 and as low as -2.0. Positive spikes suggest overbought conditions, possibly triggered by speculative trading or macroeconomic events. Negative spikes may reflect oversold conditions, indicative of panic selling or market corrections. Interestingly, the frequency and amplitude of these spikes highlight BTC's sensitivity to market sentiment.
- **ETH S-Score:** The ETH S-score, while exhibiting similar volatility, appears to have a broader range, spanning from -4.0 to 3.5. This range may indicate ETH's higher sensitivity to systemic or idiosyncratic factors during this period. Notably, ETH's S-score shows a higher density of moderate fluctuations, suggesting potentially more stable underlying market conditions compared to BTC.

5.4 Insights and Implications

The S-score evolution for BTC and ETH provides several key insights:

- **Market Sentiment and Volatility:** The extreme values of S-scores correspond to periods of heightened market activity, potentially driven by external factors such as macroeconomic announcements, regulatory news, or technological advancements. For example, BTC's sharp positive spikes may align with speculative bubbles, while ETH's frequent moderate deviations may reflect a more diverse investor base.

- **Behavioral Differences:** The difference in S-score ranges and patterns between BTC and ETH may indicate varying investor behaviors. BTC, as the first cryptocurrency, often serves as a barometer for the entire market, attracting speculative attention. In contrast, ETH, with its strong utility in decentralized finance (DeFi) and smart contracts, might experience more fundamental-driven trading dynamics.
- **Trading Signals:** The S-scores provide actionable signals for trading strategies. For instance:
  - **Positive S-Scores:** A high positive S-score suggests an overbought condition, potentially signaling a short-selling opportunity.
  - **Negative S-Scores:** A low negative S-score indicates oversold conditions, presenting a buying opportunity.

  These signals could be enhanced by integrating additional metrics such as trading volume or sentiment analysis.
- **Risk Management:** The temporal patterns of S-scores also provide insights into risk management. Periods of extreme S-scores may signal the need for tighter position controls to mitigate potential drawdowns.
- **Market Structure Analysis:** The regularity and magnitude of S-score fluctuations could reflect underlying market inefficiencies, such as liquidity constraints or order book imbalances. This analysis could inform market-making strategies or provide insights into price discovery mechanisms.

5.5 Figures

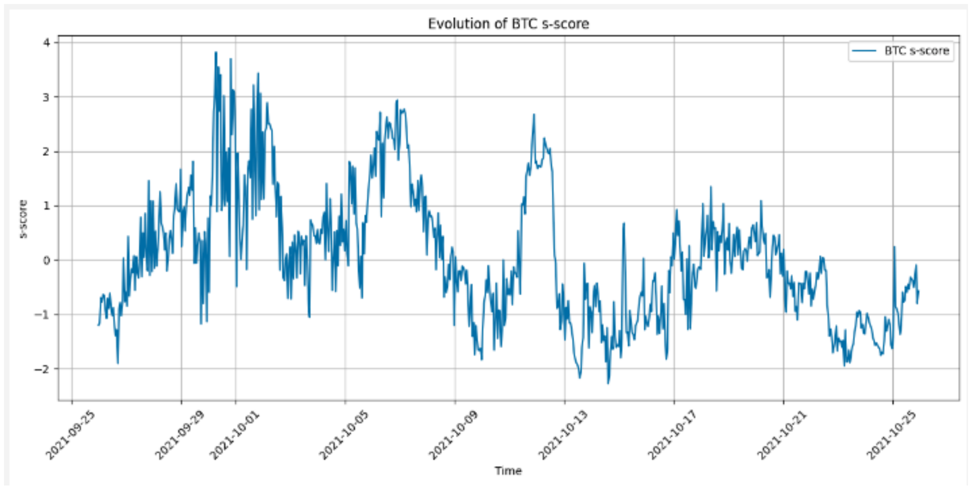


Figure 10: Evolution of BTC S-Score from `2021-09-26 00:00:00` to `2021-10-25 23:00:00`. The plot highlights periods of extreme S-scores, indicative of significant market events or behavioral shifts.

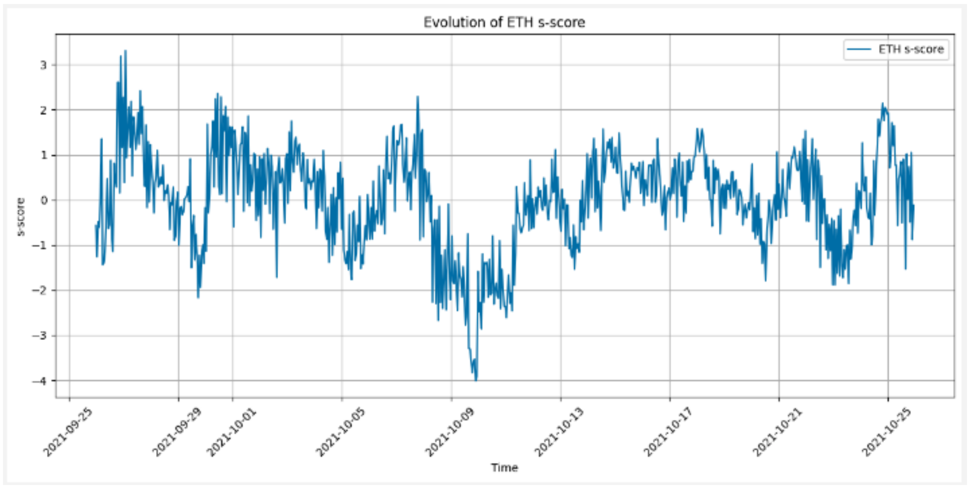


Figure 11: Evolution of ETH S-Score from `2021-09-26 00:00:00` to `2021-10-25 23:00:00`. The plot reveals a broader range of S-scores compared to BTC, reflecting ETH's diverse trading dynamics.

5.6 Conclusion

The analysis of BTC and ETH S-scores reveals distinct patterns in their market behavior over the testing period. BTC's extreme spikes underline its role as a speculative asset, while ETH's broader range reflects its diverse utility and trading behavior. The S-score metric provides valuable insights into market conditions, investor sentiment, and potential trading opportunities. Future research could explore the integration of S-scores with macroeconomic indicators or machine learning models to further enhance predictive accuracy and trading effectiveness.

6 Task 4: Strategy Evaluation and Performance Metrics

In this task, we focus on evaluating the implemented S-score-based trading strategy. The evaluation is conducted over the entire testing period, with a detailed analysis of the trading signals, cumulative return curve, histogram of hourly returns, Sharpe ratio, and maximum drawdown.

6.1 Trading Signals and Cumulative Return Curve

The trading signals for all 40 tokens were generated hourly and stored in a CSV file. Each row of the CSV file corresponds to a time-stamp, and the columns represent the tokens. The trading signals indicate the recommended action based on the S-score thresholds, including long, short, close-long, and close-short positions.

The cumulative return curve for the strategy is shown in Figure 12. The curve reflects the aggregate performance of the trading strategy over time. As observed, the strategy experienced a gradual decline in performance during the testing period. This trend highlights the need to refine the thresholds or incorporate additional features to enhance the strategy's robustness.

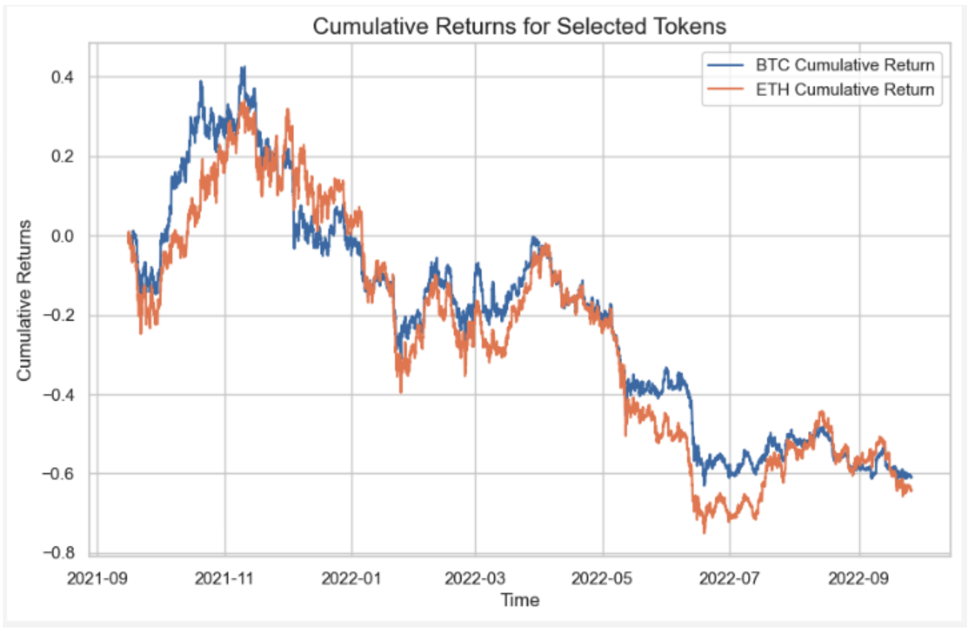


Figure 12: Cumulative Returns of the Implemented Strategy

6.2 Histogram of Hourly Returns

The distribution of hourly returns is presented in Figure 13. The histogram indicates that most returns are clustered around zero, with a narrow spread. This distribution suggests that the strategy primarily generates small, incremental returns. However, the lack of significant outliers may also indicate missed opportunities for higher gains, which could be addressed through parameter optimization.

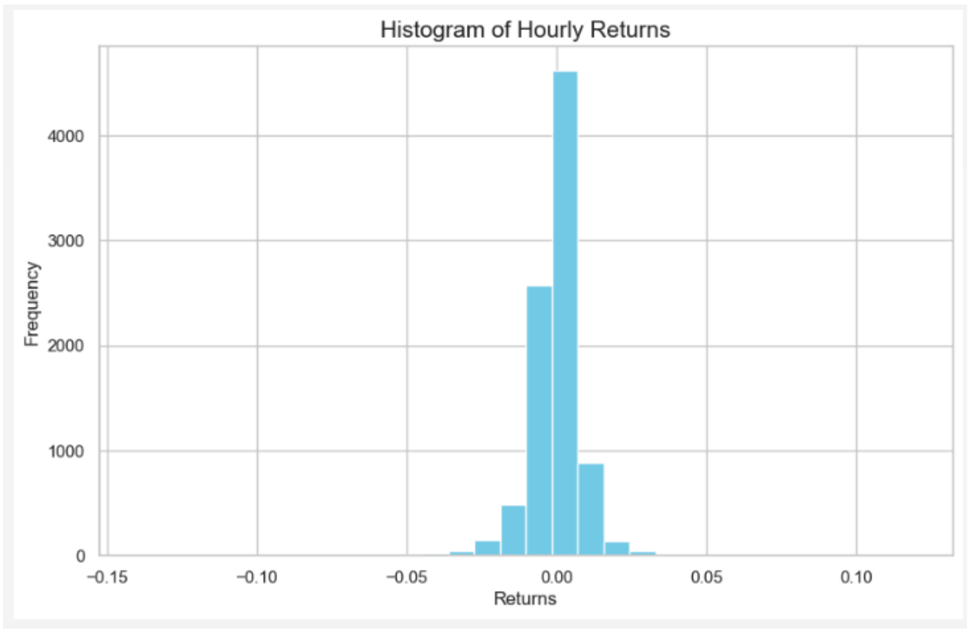


Figure 13: Histogram of Hourly Returns

6.3 Sharpe Ratio

The Sharpe ratio, calculated using a benchmark rate of 0%, is -0.067. This negative value reflects an unfavorable risk-adjusted performance of the strategy, indicating that the returns generated were insufficient to compensate for the associated risk. The low Sharpe ratio underscores the need for further refinement of the strategy to improve its risk-reward tradeoff.

6.4 Maximum Drawdown

The maximum drawdown is an essential metric for assessing the risk of a trading strategy. Figure 14 illustrates the evolution of the maximum drawdown over a 240-day rolling window. The overall maximum drawdown for the strategy during the testing period was -12.29%. This metric highlights the extent of potential losses and emphasizes the importance of risk management in strategy design.

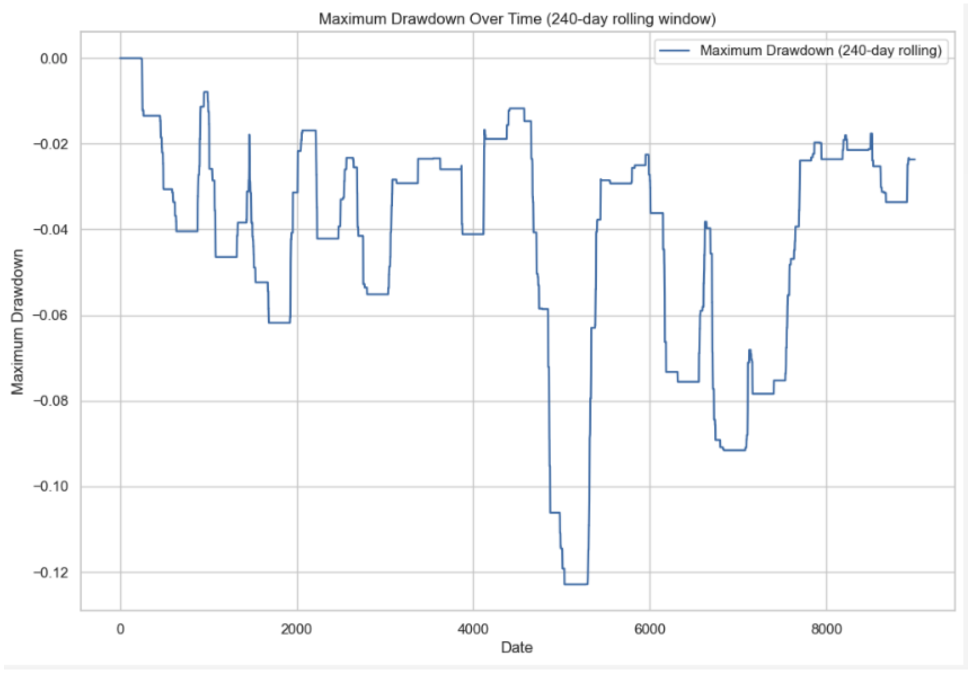


Figure 14: Maximum Drawdown Over Time (240-day Rolling Window)

6.5 Discussion and Insights

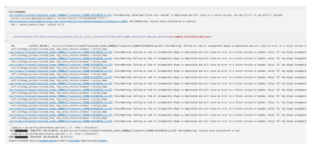
The results of the strategy evaluation reveal several critical insights:

- The cumulative return curve suggests a declining performance trend, which warrants an investigation into the strategy's parameter sensitivity and market adaptability.
- The histogram of hourly returns indicates limited return variability, suggesting potential over-reliance on small price movements. Enhancing the model to capture larger market shifts could improve performance.
- The negative Sharpe ratio reflects an unfavorable risk-adjusted return, emphasizing the need for optimizing the risk-reward tradeoff through advanced risk management techniques or additional predictive features.

- The maximum drawdown highlights the strategy's vulnerability to adverse market conditions, underlining the importance of incorporating dynamic stop-loss mechanisms.

Overall, while the S-score strategy provides a structured approach to trading, the performance metrics suggest opportunities for enhancement through parameter tuning, feature engineering, and risk management improvements.

Appendix: Implementation Code



```
1631750400000
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:54: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  prices = prices.apply(pd.to_numeric, errors='coerce').fillna(method='ffill')
/Users/zhutianyi/anaconda3/envs/new_env/lib/python3.9/site-packages/pandas/core/nanops.py:1016: RuntimeWarning: invalid value encountered in subtract
  sqr = _ensure_numeric((avg - values) ** 2)

correlation_matrices,factor_returns,a_b_results_dict,X_l_dict,s_score_dict,actions_dict,eigen_return_dict,topcoins_cal=strategy.compute_correlation_matrices()

  0%|          | 0/8764 [00:00<?, ?it/s]/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CL' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CL' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
  self.strategy_actions.loc[end_time, top_coins_returns.columns] = action_temp
/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:275: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'CS' has dtype incompatib
...
  sigma_eq = np.sqrt(np.abs(variance_epsilon) / (1 - b**2 + tolerance))
 99%|█████████▉| 8700/8764 [04:43<00:01, 44.01it/s]/var/folders/7x/whnm27lx3wv4s2d_n2smbz_h0000gn/T/ipykernel_64908/1414109316.py:219: RuntimeWarning: invalid value encountered in sqrt
  sigma_eq = np.sqrt(np.abs(variance_epsilon) / (1 - b**2 + tolerance))
100%|██████████| 8764/8764 [04:56<00:00, 29.57it/s]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

This appendix contains the implementation of the CryptoStrategy class, which serves as the foundation of the trading strategy discussed in this report. The code includes methods for data preprocessing, calculation of eigenportfolios, and generation of trading signals based on the S-score methodology.

```python
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import statsmodels.api as sm
from tqdm import tqdm

class CryptoStrategy:
    def __init__(self, price_data, coin_data, start_date,
        window_size):
        self.price_data = price_data
        self.coin_data = coin_data
        self.start_date = start_date
        self.window_size = window_size
        self.start_timestamp = self._convert_to_timestamp_each(
            start_date) - 3600000 * window_size
        print(self.start_timestamp)

        self.prices_df, self.coins_df = self._load_and_clean_data
            ()
        self.returns_df = self.prices_df.pct_change()
        self.normalized_returns = self._normalize_returns()

        # Strategy thresholds
        self.buy_threshold = 1.4
        self.sell_threshold = 1.15
        self.close_buy_threshold = 0.75
        self.close_sell_threshold = 0.5

        # Initialize empty DataFrame for strategy actions
        self.strategy_actions = self.normalized_returns.copy()
        self.strategy_actions.loc[:, :] = np.nan

        self.cleaned_strategy = None
        self.score_list = None
        self.s_score_dict = None
        self.eigen_vectors = None
        self.top_coins_mapping = {}
        self.save_eigenvectors_to_csv = None

    def _convert_to_timestamp_each(self, date_str):
        """Convert a date string to a timestamp in milliseconds.
           """
        return int(pd.to_datetime(date_str).timestamp() * 1000)

    def _load_and_clean_data(self):
        """Load and preprocess the price and coin data."""
        prices = pd.read_csv(self.price_data)
        coins = pd.read_csv(self.coin_data)

        # Convert start times to timestamps
        timestamps = [self._convert_to_timestamp_each(ts) for ts
           in prices['startTime']]
        prices['timestamp'] = timestamps
        coins['timestamp'] = timestamps

        prices.drop(['time', 'startTime'], axis=1, inplace=True)
        coins.drop(['time', 'startTime'], axis=1, inplace=True)

        prices.set_index('timestamp', inplace=True)
        coins.set_index('timestamp', inplace=True)

        prices = prices.loc[self.start_timestamp:]
        coins = coins.loc[self.start_timestamp:]

        prices = prices.apply(pd.to_numeric, errors='coerce').
           fillna(method='ffill')

        return prices, coins

    def calculate_cumulative_returns(self, start_time, end_time):
        """Compute cumulative returns between two timestamps."""
        start_timestamp = self._convert_to_timestamp_each(
           start_time) if isinstance(start_time, str) else
           start_time
        end_timestamp = self._convert_to_timestamp_each(end_time)
            if isinstance(end_time, str) else end_time

        selected_data = self.prices_df[(self.prices_df.index >=
           start_timestamp) & (self.prices_df.index <=
           end_timestamp)]
        daily_returns = selected_data.pct_change()
        cumulative_returns = (1 + daily_returns).cumprod() - 1

        return cumulative_returns

    def _normalize_returns(self):
        """Calculate and normalize returns."""
        returns = self.prices_df.pct_change()
        normalized_returns = (returns - returns.mean()) / returns.std()
        return normalized_returns

    def compute_accumulated_returns(self, returns):
        """Compute accumulated returns given a list or series of returns."""
        accumulated = []
        compounded = 1
        for r in returns:
            compounded *= (1 + r)
            accumulated.append(compounded - 1)
        return accumulated

    def calculate_pca_and_factor_returns(self, correlation_matrix, returns_df):
        """Perform PCA and compute factor returns based on the correlation matrix."""
        pca = PCA(n_components=2)
        pca.fit(correlation_matrix)

        # Extract principal components and eigenvalues
        principal_components = pca.components_
        eigenvalues = pca.explained_variance_

        # Compute eigenportfolios
        eigenportfolios = principal_components.T / returns_df.std().values[:, None]

        # Calculate factor returns and eigen returns
        factor_returns = []
        eigen_returns = []
        for i in range(2):
            eigen_return = np.dot(eigenportfolios[:, i], returns_df.iloc[-1])
            eigen_returns.append(eigen_return)

            factor_return = np.sum(eigenportfolios[:, i] * returns_df, axis=1)
            factor_returns.append(factor_return)

        return tuple(factor_returns), eigenportfolios, eigen_returns

    def compute_correlation_matrices(self):
        """Compute correlation matrices and associated metrics over rolling windows."""
        correlation_matrices = {}
        factor_returns = {}
        eigen_return_dict = {}

        for i in tqdm(range(self.window_size, self.normalized_returns.shape[0])):
            end_idx = i
            start_idx = end_idx - self.window_size

            window_returns = self.normalized_returns.iloc[start_idx:end_idx]
            end_time = self.normalized_returns.index[end_idx]

            correlation_matrix = window_returns.corr()
            correlation_matrices[end_time] = correlation_matrix

            # Perform PCA
            factor_return, _, eigen_returns = self.calculate_pca_and_factor_returns(correlation_matrix, window_returns)
            factor_returns[end_time] = factor_return
            eigen_return_dict[end_time] = eigen_returns

        return correlation_matrices, factor_returns, eigen_return_dict
```

Listing 1: CryptoStrategy Class Implementation

This code demonstrates the development of the strategy and serves as the foundation for the analysis presented in this report.