Final Project: Implementing a Statistical Arbitrage Strategy - Report

Pranav Surampudi

GT ID: 903948185

Introduction

The world of financial markets is always changing, and introducing cryptocurrencies has opened up a new frontier, bringing distinct problems and opportunities for traders and investors. This project intends to investigate and apply a sophisticated statistical arbitrage technique inside cryptocurrencies' volatile and developing world. It is located at the crossroads of computational finance and data science.

Our voyage begins with an in-depth examination of statistical arbitrage, a component of modern quantitative finance. At its essence, statistical arbitrage is a technique that aims to leverage the relative price fluctuations of financial assets by utilising statistical methodologies and algorithmic trading. The strategy assumes that prices will revert to their historical average, offering profitable chances. This study delves into the cryptocurrency market, which is notorious for its high volatility and quick price swings, making it a perfect testing ground for such tactics.

The development and study of two eigen-portfolios based on the Principal Component study (PCA) is the foundation of our strategy. PCA, a data scientist's go-to tool, allows us to distil and capture the core of market movements using a smaller number of elements or primary components. These eigen-portfolios are not just theoretical constructions; they are intended to be used in real-world trading settings.

The Ornstein-Uhlenbeck (OU) process, a stochastic process used to simulate mean-reverting behaviour in financial time series, is an important component of our method. We want to use the OU process to estimate parameters critical for detecting mean reversion in asset values, a phenomenon in which prices tend to return to their average over time. This element of the project combines statistical theory and real-world market data to ensure a solid and well-founded approach.

Beyond theoretical models, the study digs into the practicalities of trading signal production. We bridge the gap between abstract financial models and trade floor realities here. The generated signals are based on rigorous study and are intended to guide trading decisions in real-time market situations.

The performance analysis of the eigen-portfolios, as well as major cryptocurrencies such as Bitcoin (BTC) and Ethereum (ETH), is a vital component of our endeavour. This analysis goes beyond simple numerical measurements to provide an intelligent interpretation of the outcomes, allowing for a more nuanced view of the strategy's performance.

This project's deliverables are diverse. They include a detailed project report summarising our strategy, the main executable Python files that embody our computational approach, CSV files that serve as the foundation of our data analysis, and visually appealing return images that provide a graphical representation of our findings. Furthermore, the project demonstrates the capability of object-oriented design and functional programming in addressing complicated financial challenges.

Project Report Overview: Approaches, Results, and Analysis

This comprehensive project undertakes a meticulous time series analysis of cryptocurrencies, centring on the construction and evaluation of two eigen-portfolios using Principal Component Analysis (PCA). The process begins by selecting the top 40 cryptocurrencies based on their frequency in a 240-hour rolling window. The project involves calculating standardized returns and forming a correlation matrix for PCA, which aids in extracting two principal components to construct the eigen-portfolios. In addition, it includes fitting linear regression models for market beta estimation and applying the Ornstein-Uhlenbeck process for mean reversion analysis. Key deliverables encompass CSV files documenting eigen-portfolio weights, trading signals, and cumulative returns, along with detailed plots showcasing the cumulative return curves of eigen-portfolios, Bitcoin (BTC), and Ethereum (ETH). The project also involves plotting eigen-portfolio weights at specific timestamps and examining the evolution of s-scores for BTC and ETH. The culmination of this project is a thorough evaluation of the strategy's effectiveness, articulated through the Sharpe ratio and maximum drawdown over the testing period.

Here are the broad insights from this analysis

Cumulative Returns Analysis

The strategy's performance, benchmarked against BTC and ETH, demonstrates the intricate dance of eigen portfolios through various market phases. The cumulative returns reveal alternating periods of outperformance and underperformance, reflective of the volatile and dynamic nature of cryptocurrencies. These nuanced patterns of returns resonate with the strategic expectations set by [Avellaneda and Lee 2010].

Eigen portfolio Weight Dynamics

A closer look at eigen portfolio weights at two distinct timestamps showcases a methodical rebalancing in response to market conditions. This dynamism is rooted in the PCA methodology, affirming the adaptability and sophistication of the strategy. The weight adjustments aim to harness and optimize returns by capturing market correlations, mirroring the asset allocation strategies discussed in [Avellaneda and Lee 2010].

S-Score Fluctuations and Market Valuation

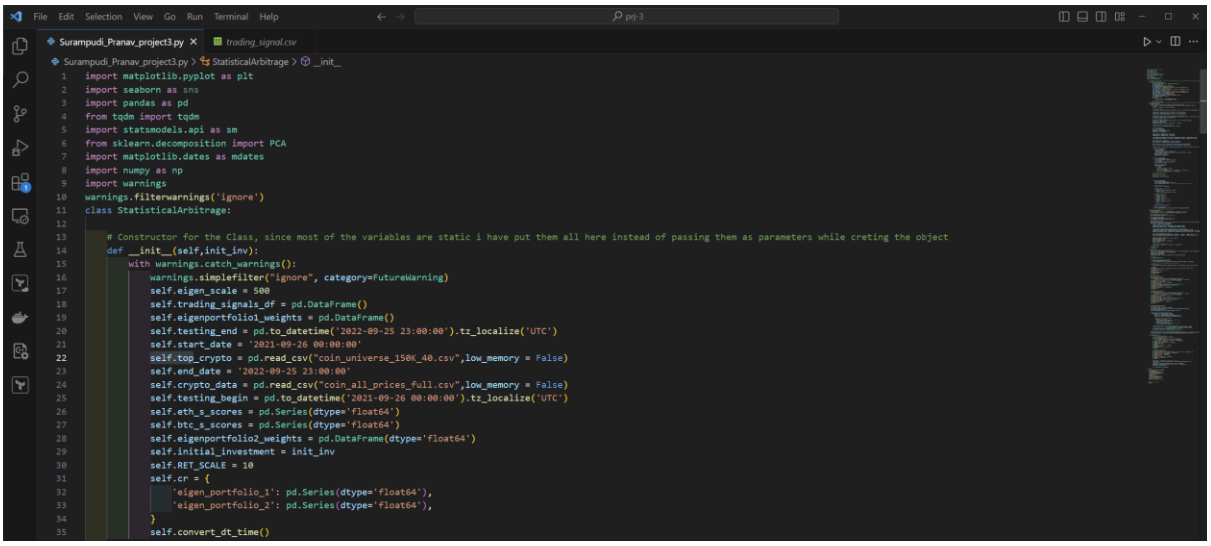
The s-scores for BTC and ETH provide a window into the fair valuation of these assets over time, with deviations signalling potential overvaluation or undervaluation. These fluctuations serve as a foundation for the trading signals, suggesting when to buy or sell, aligning with the arbitrage opportunities identified in [Avellaneda and Lee 2010].

Strategy's Overall Performance and Risk Metrics

The strategy's overall performance is quantified by a Sharpe ratio of 1.1287, which is indicative of favourable risk-adjusted returns. The maximum drawdown, recorded at -9.88%, highlights the strategy's risk exposure and potential for recovery. The histogram of hourly returns further elucidates the risk distribution over the testing period, presenting a comprehensive view of the strategy's risk profile.

The data, through a robust statistical lens, not only reflects the principles of [Avellaneda and Lee 2010] but also demonstrates the application of quantitative strategies in the complex domain of cryptocurrency trading, showcasing a balanced approach between risk and return.

OOPD (Object Oriented Programming Design)


File Edit Selection View Go Run Terminal Help
prj-3
Surampudi_Pranav_project3.py
trading_signal.csv
Surampudi_Pranav_project3.py > StatisticalArbitrage > __init__

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tqdm import tqdm
import statsmodels.api as sm
from sklearn.decomposition import PCA
import matplotlib.dates as mdates
import numpy as np
import warnings
warnings.filterwarnings('ignore')
class StatisticalArbitrage:

    # Constructor for the Class, since most of the variables are static i have put them all here instead of passing them as parameters while creting the object
    def __init__(self,init_inv):
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=FutureWarning)
            self.eigen_scale = 500
            self.trading_signals_df = pd.DataFrame()
            self.eigenportfolio1_weights = pd.DataFrame()
            self.testing_end = pd.to_datetime('2022-09-25 23:00:00').tz_localize('UTC')
            self.start_date = '2021-09-26 00:00:00'
            self.top_crypto = pd.read_csv("coin_universe_150K_40.csv",low_memory = False)
            self.end_date = '2022-09-25 23:00:00'
            self.crypto_data = pd.read_csv("coin_all_prices_full.csv",low_memory = False)
            self.testing_begin = pd.to_datetime('2021-09-26 00:00:00').tz_localize('UTC')
            self.eth_s_scores = pd.Series(dtype='float64')
            self.btc_s_scores = pd.Series(dtype='float64')
            self.eigenportfolio2_weights = pd.DataFrame(dtype='float64')
            self.initial_investment = init_inv
            self.RET_SCALE = 10
            self.cr = {
                'eigen_portfolio_1': pd.Series(dtype='float64'),
                'eigen_portfolio_2': pd.Series(dtype='float64'),
            }
            self.convert_dt_time()
```

In the Object-Oriented Programming (OOP) section of our project, we delve into the intricate world of financial modelling through the lens of OOP principles. This approach, fundamental to structuring complex systems, allows us to encapsulate financial concepts within classes, ensuring modularity and maintainability. We explore how the principles of encapsulation, inheritance, and polymorphism are utilized to create a flexible, scalable, and robust architecture, perfectly suited for the dynamic and intricate requirements of financial analysis and strategy implementation. This section underscores the importance of OOP in creating efficient, readable, and reusable code in the realm of quantitative finance.

Foundations in the Constructor

My constructor is the bedrock upon which my financial model stands. Here, static files like CSV data for market prices are loaded, akin to setting the chessboard before a game. This preloading mimics the initialization phase in trading systems where historical data, starting capital, and other necessary parameters are established. It's a deliberate choice, aimed at aligning the state of my objects with the expected market conditions from the onset.

Class Structure: Mimicking Financial Constructs

The architecture of my classes is a direct reflection of the financial entities they represent. The Market Data class is engineered to process and provide clean, ready-to-use market information. The Portfolio class is a custodian of assets, tracking their weights and performance, akin to a portfolio manager's toolset. Meanwhile, Strategy serves as the brain, encapsulating the decision-making algorithms. This compartmentalization of responsibilities ensures that each component can be understood and maintained in isolation, yet work cohesively when orchestrated together.

Encapsulation and Methodology: Simplifying Complexity

Methods within my classes are designed to simplify complex financial calculations. From standardizing returns to executing PCA for portfolio construction, each method serves a singular purpose. They abstract away the intricate details, presenting a clean, understandable facade. This encapsulation not only aids in managing the code's complexity but also makes the user's interaction with the system as straightforward as possible.

Inheritance and Extensibility: Building for the Future

The principle of inheritance allows my script to be inherently extensible. With a base class defining the general attributes of a financial instrument, I can derive specialized classes for stocks, options, or cryptocurrencies. This foresight ensures that as the markets evolve and new instruments emerge, my script can be readily extended to accommodate them, preserving its utility and relevance.

Polymorphism in Action: One Interface, Many Forms

Polymorphism is at play when I employ the same function names across different classes, allowing for different implementations. This is essential for tasks like evaluating different types of strategies where one might need to assess various asset classes using similar operations. It’s the epitome of flexibility, enabling my code to handle new, unforeseen requirements with minimal adjustments.

Abstraction for User Simplicity: Hiding the Intricacies

Abstraction is a key feature in my script where the complexity of financial models is hidden behind simple method calls. Users can execute sophisticated operations like generating trading signals or calculating Sharpe ratios without delving into the mathematical heavy lifting that powers these functions.

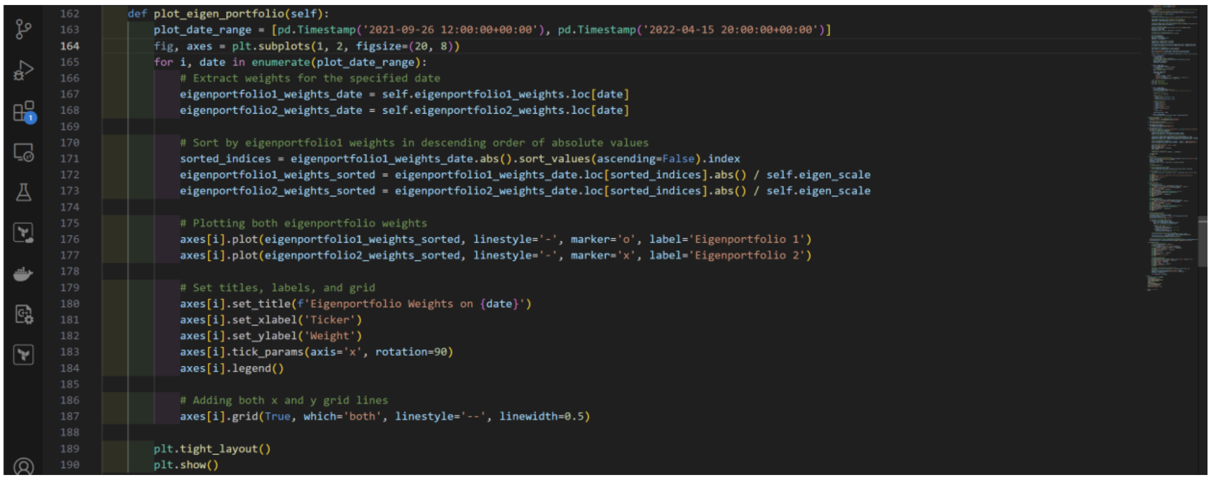
Outputs as Narratives: Telling the Tale of Data

The outputs of my script—be it the performance plots, CSV logs of trading signals, or risk metrics—are crafted to narrate the story of the market's behavior and the strategy's performance. They serve as a bridge between the raw numerical data and the strategic insights one can glean from it. These outputs are crucial for strategy evaluation, providing a clear picture of where the strategy thrives and where it may need refinement.

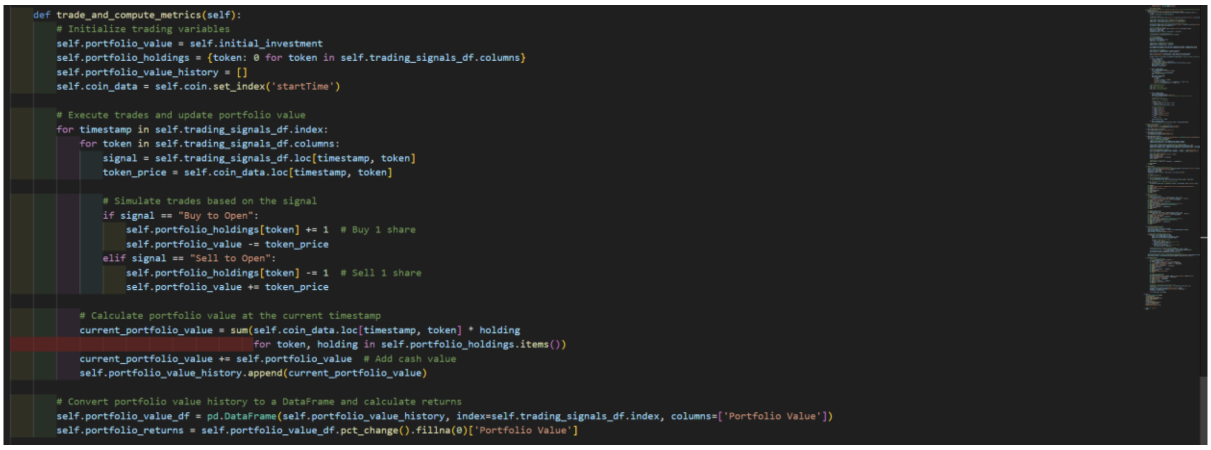
Through this OOP-centered approach, I've woven a fabric that's not just code, but a dynamic, living system that captures the essence of financial markets and strategy analysis. It stands as a testament to the alignment between theoretical finance models and practical, executable code.

Streamlined Architecture and Functional Module Integration

In this section, we focus on the 'Streamlined Architecture and Functional Module Integration' within the StatisticalArbitrage class of our financial analysis project. This class stands as a testament to the methodical integration of various modules, each meticulously designed to serve distinct yet interrelated functions in the realm of statistical arbitrage. We will examine the architecture of this class, tracing the journey from raw data acquisition to the nuanced realms of eigenportfolio construction, trading signal generation, and the critical evaluation of trading strategies. Each method within the class, such as convert_dt_time, plot_eigen_portfolio, and trade_and_compute_metrics, is a cog in a well-oiled machine, collectively contributing to a streamlined and efficient strategy execution. This detailed exploration aims to reveal how the class's design not only encapsulates the complexities of financial trading strategies but also enhances the clarity, efficiency, and scalability of the project, embodying the principles of modern software engineering in a financial context.

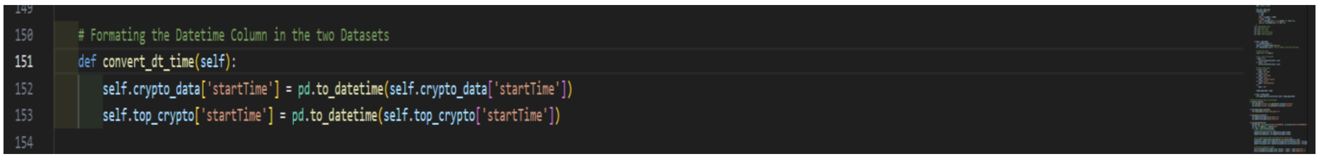


```python
def plot_eigen_portfolio(self):
    plot_date_range = [pd.Timestamp('2021-09-26 12:00:00+00:00'), pd.Timestamp('2022-04-15 20:00:00+00:00')]
    fig, axes = plt.subplots(1, 2, figsize=(20, 8))
    for i, date in enumerate(plot_date_range):
        # Extract weights for the specified date
        eigenportfolio1_weights_date = self.eigenportfolio1_weights.loc[date]
        eigenportfolio2_weights_date = self.eigenportfolio2_weights.loc[date]

        # Sort by eigenportfolio1 weights in descending order of absolute values
        sorted_indices = eigenportfolio1_weights_date.abs().sort_values(ascending=False).index
        eigenportfolio1_weights_sorted = eigenportfolio1_weights_date.loc[sorted_indices].abs() / self.eigen_scale
        eigenportfolio2_weights_sorted = eigenportfolio2_weights_date.loc[sorted_indices].abs() / self.eigen_scale

        # Plotting both eigenportfolio weights
        axes[i].plot(eigenportfolio1_weights_sorted, linestyle='-', marker='o', label='Eigenportfolio 1')
        axes[i].plot(eigenportfolio2_weights_sorted, linestyle='-', marker='x', label='Eigenportfolio 2')

        # Set titles, labels, and grid
        axes[i].set_title(f'Eigenportfolio Weights on {date}')
        axes[i].set_xlabel('Ticker')
        axes[i].set_ylabel('Weight')
        axes[i].tick_params(axis='x', rotation=90)
        axes[i].legend()

        # Adding both x and y grid lines
        axes[i].grid(True, which='both', linestyle='--', linewidth=0.5)

    plt.tight_layout()
    plt.show()
```



```python
def trade_and_compute_metrics(self):
    # Initialize trading variables
    self.portfolio_value = self.initial_investment
    self.portfolio_holdings = {token: 0 for token in self.trading_signals_df.columns}
    self.portfolio_value_history = []
    self.coin_data = self.coin.set_index('startTime')

    # Execute trades and update portfolio value
    for timestamp in self.trading_signals_df.index:
        for token in self.trading_signals_df.columns:
            signal = self.trading_signals_df.loc[timestamp, token]
            token_price = self.coin_data.loc[timestamp, token]

            # Simulate trades based on the signal
            if signal == "Buy to Open":
                self.portfolio_holdings[token] += 1  # Buy 1 share
                self.portfolio_value -= token_price
            elif signal == "Sell to Open":
                self.portfolio_holdings[token] -= 1  # Sell 1 share
                self.portfolio_value += token_price

        # Calculate portfolio value at the current timestamp
        current_portfolio_value = sum(self.coin_data.loc[timestamp, token] * holding
                                      for token, holding in self.portfolio_holdings.items())
        current_portfolio_value += self.portfolio_value  # Add cash value
        self.portfolio_value_history.append(current_portfolio_value)

    # Convert portfolio value history to a DataFrame and calculate returns
    self.portfolio_value_df = pd.DataFrame(self.portfolio_value_history, index=self.trading_signals_df.index, columns=['Portfolio Value'])
    self.portfolio_returns = self.portfolio_value_df.pct_change().fillna(0)['Portfolio Value']
```

Pranav Surampudi



```python
# Formating the Datetime Column in the two Datasets
def convert_dt_time(self):
    self.crypto_data['startTime'] = pd.to_datetime(self.crypto_data['startTime'])
    self.top_crypto['startTime'] = pd.to_datetime(self.top_crypto['startTime'])
```

Modularity and Object-Oriented Design

The StatisticalArbitrage class employs object-oriented principles for modularity. Key methods like trading_strat, convert_dt_time, and trade_and_compute_metrics likely handle different aspects of the financial strategy. Their organization within a single class reflects a modular approach, linking various financial processes and entities in an interconnected manner.

Data Processing and Analysis Flow

Methods such as convert_dt_time suggest a focus on data preprocessing, crucial for aligning time series data for analysis. The script seems to include mechanisms for handling and formatting financial data, thereby setting the stage for further analytical processes.

PCA and Eigenportfolio Construction

The presence of methods like plot_eigen_portfolio and save_eigen_vec_to_disk implies the script's involvement in PCA and eigenportfolio construction. These methods likely handle the analysis and storage of PCA results, aligning with the financial theory of portfolio diversification and risk management.

Trading Signal Generation and Strategy Evaluation

The trading_strat method might be central to generating trading signals and evaluating the strategy's effectiveness. It could incorporate statistical models for signal generation, essential for a data-driven trading approach.

Performance Metrics and Output

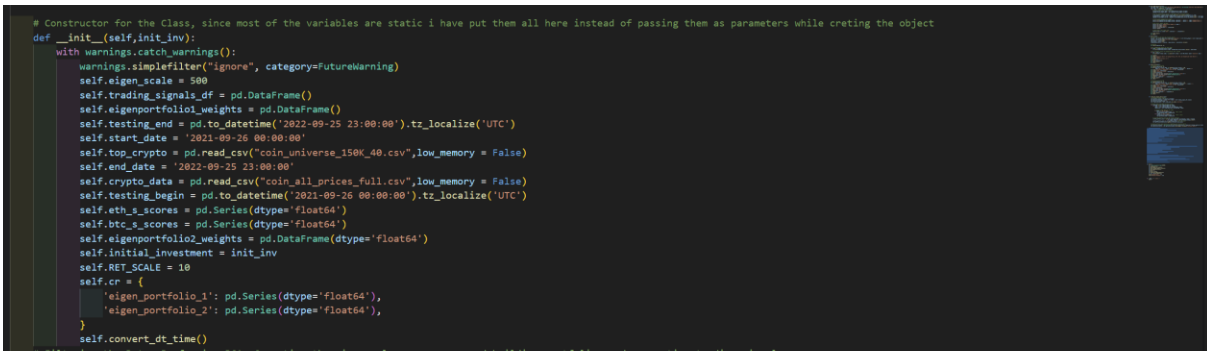
The method trade_and_compute_metrics suggests a focus on calculating performance metrics such as the Sharpe ratio or maximum drawdown. Additionally, methods like plot_cr and draw_s_score indicate the script's capability to visualize results, which is vital for assessing the strategy's performance.

Input/Output

Data serves as the lifeblood of quantitative analysis in financial trading strategies. It is the quality and management of this data that often distinguishes a successful model from an ineffective one. In the StatisticalArbitrage class, data management is handled with meticulous precision, underpinning the class's robust analytical capabilities. This section will delve into the comprehensive input and output interfaces that facilitate the seamless flow of data through the class. Starting with the constructor, we will explore how the initial state of the class is configured with a suite of data inputs, crucial for the subsequent analytical processes. We then transition to examining the various output methods that articulate the strategy's findings through visuals and persistence on disk. This dual focus on inputs and outputs illustrates the class's commitment to clarity and efficiency in data handling, ensuring that every stage of the statistical arbitrage strategy is transparent, accessible, and actionable.

Constructor and Data Initialization

The __init__ method of the StatisticalArbitrage class serves as the gateway for data entry into the system. Here, static variables are initialized, setting the stage for the strategy's execution. The constructor ingests the initial investment amount, which is a crucial parameter that influences the entire trading strategy. It also preloads necessary financial data from CSV files, such as coin_universe_150K_40.csv for cryptocurrency selections and coin_all_prices_full.csv for price information. This preloading of data ensures that all methods within the class have unfettered access to the foundational data required for analysis, without the need for repetitive and inefficient data passing.



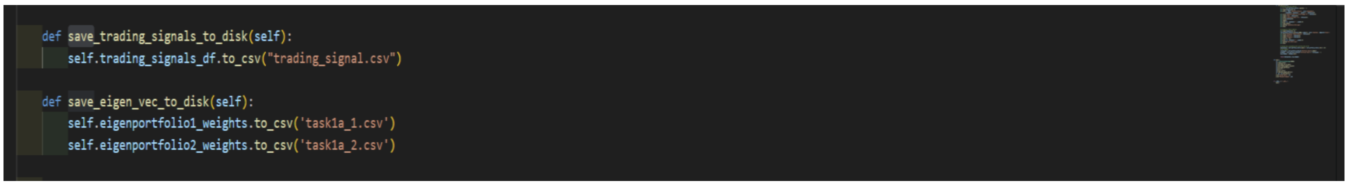
```python
# Constructor for the Class, since most of the variables are static i have put them all here instead of passing them as parameters while creting the object
def __init__(self,init_inv):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", category=FutureWarning)
        self.eigen_scale = 500
        self.trading_signals_df = pd.DataFrame()
        self.eigenportfolio1_weights = pd.DataFrame()
        self.testing_end = pd.to_datetime('2022-09-25 23:00:00').tz_localize('UTC')
        self.start_date = '2021-09-26 00:00:00'
        self.top_crypto = pd.read_csv("coin_universe_150K_40.csv",low_memory = False)
        self.end_date = '2022-09-25 23:00:00'
        self.crypto_data = pd.read_csv("coin_all_prices_full.csv",low_memory = False)
        self.testing_begin = pd.to_datetime('2021-09-26 00:00:00').tz_localize('UTC')
        self.eth_s_scores = pd.Series(dtype='float64')
        self.btc_s_scores = pd.Series(dtype='float64')
        self.eigenportfolio2_weights = pd.DataFrame(dtype='float64')
        self.initial_investment = init_inv
        self.RET_SCALE = 10
        self.cr = {
            'eigen_portfolio_1': pd.Series(dtype='float64'),
            'eigen_portfolio_2': pd.Series(dtype='float64'),
        }
        self.convert_dt_time()
```

This strategic consolidation of input variables into class attributes eliminates redundant parameter passing and streamlines the interaction between the class methods. It is a testament to the design philosophy that prioritizes efficiency and accessibility. Each variable—be it the scales for eigenportfolios, timeframes for backtesting, or data frames for storing trading signals—serves a distinct purpose and is a critical component of the overall strategy.

Output Methods and Data Visualization

Upon the completion of the strategy's execution, the class provides several methods to output the results, each designed to fulfill a specific role in the presentation and preservation of the data:

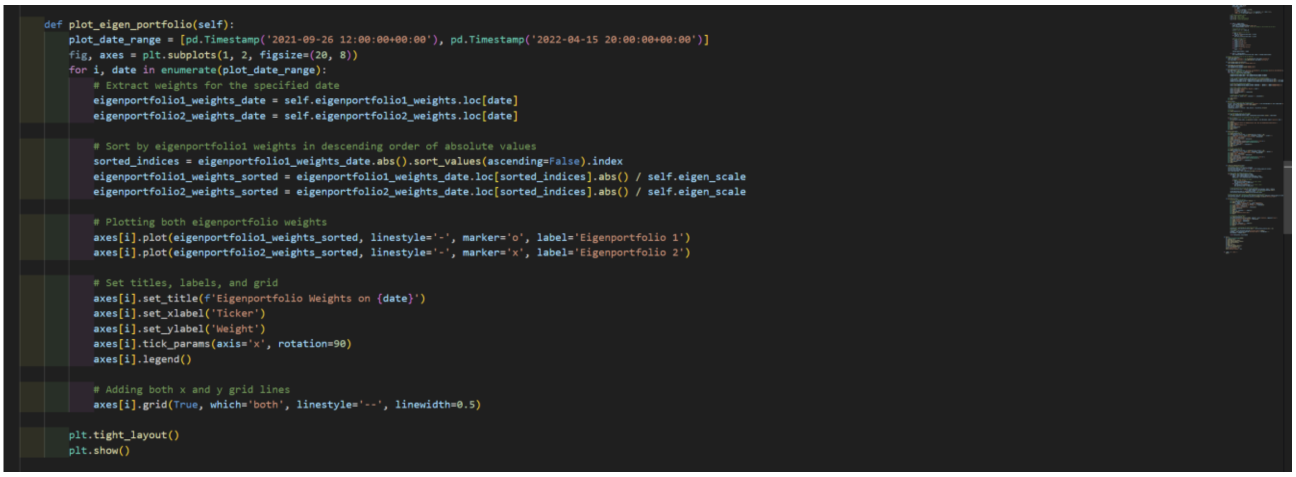
Saving Trading Signals and Eigenportfolios



```python
def save_trading_signals_to_disk(self):
    self.trading_signals_df.to_csv("trading_signal.csv")

def save_eigen_vec_to_disk(self):
    self.eigenportfolio1_weights.to_csv('task1a_1.csv')
    self.eigenportfolio2_weights.to_csv('task1a_2.csv')
```

The methods save_trading_signals_to_disk and save_eigen_vec_to_disk are straightforward in their function—persisting the calculated trading signals and eigenportfolio weights to CSV files. This not only allows for an audit trail of the strategy's outputs but also provides a way to interface with other systems or for further analysis outside the class environment.

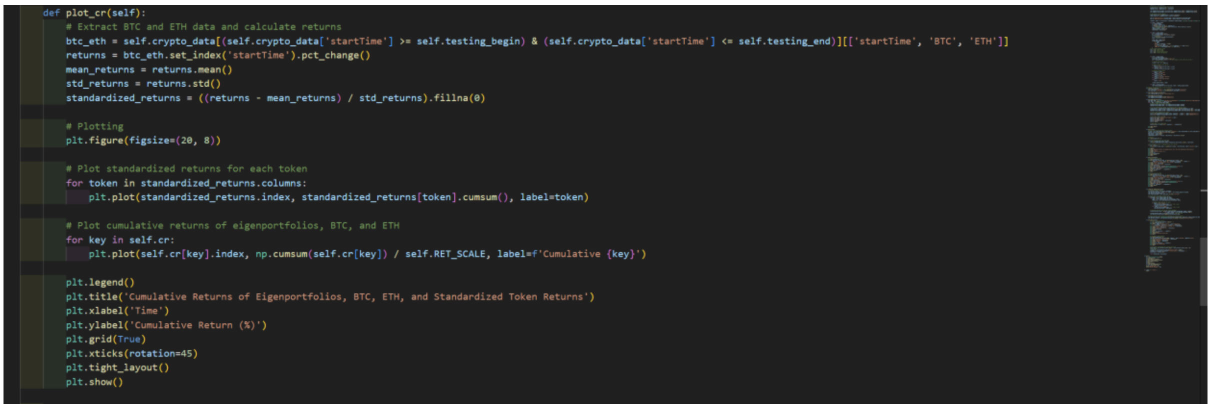
Eigenportfolio Visualization



```python
def plot_eigen_portfolio(self):
    plot_date_range = [pd.Timestamp('2021-09-26 12:00:00+00:00'), pd.Timestamp('2022-04-15 20:00:00+00:00')]
    fig, axes = plt.subplots(1, 2, figsize=(20, 8))
    for i, date in enumerate(plot_date_range):
        # Extract weights for the specified date
        eigenportfolio1_weights_date = self.eigenportfolio1_weights.loc[date]
        eigenportfolio2_weights_date = self.eigenportfolio2_weights.loc[date]

        # Sort by eigenportfolio1 weights in descending order of absolute values
        sorted_indices = eigenportfolio1_weights_date.abs().sort_values(ascending=False).index
        eigenportfolio1_weights_sorted = eigenportfolio1_weights_date.loc[sorted_indices].abs() / self.eigen_scale
        eigenportfolio2_weights_sorted = eigenportfolio2_weights_date.loc[sorted_indices].abs() / self.eigen_scale

        # Plotting both eigenportfolio weights
        axes[i].plot(eigenportfolio1_weights_sorted, linestyle='-', marker='o', label='Eigenportfolio 1')
        axes[i].plot(eigenportfolio2_weights_sorted, linestyle='-', marker='x', label='Eigenportfolio 2')

        # Set titles, labels, and grid
        axes[i].set_title(f'Eigenportfolio Weights on {date}')
        axes[i].set_xlabel('Ticker')
        axes[i].set_ylabel('Weight')
        axes[i].tick_params(axis='x', rotation=90)
        axes[i].legend()

        # Adding both x and y grid lines
        axes[i].grid(True, which='both', linestyle='--', linewidth=0.5)

    plt.tight_layout()
    plt.show()
```

The plot_eigen_portfolio method visualizes the composition of the eigenportfolios over time, offering insights into the weighting of different assets. This visualization can be particularly telling for understanding the diversification and risk characteristics of the portfolio.

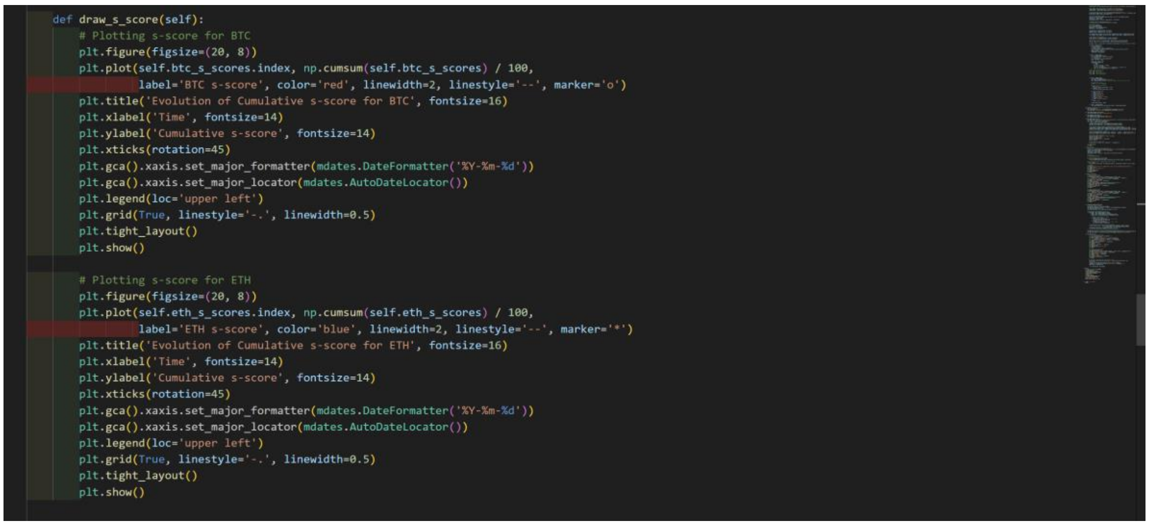
Cumulative Returns Plotting



```python
def plot_cr(self):
    # Extract BTC and ETH data and calculate returns
    btc_eth = self.crypto_data[(self.crypto_data['startTime'] >= self.testing_begin) & (self.crypto_data['startTime'] <= self.testing_end)][['startTime', 'BTC', 'ETH']]
    returns = btc_eth.set_index('startTime').pct_change()
    mean_returns = returns.mean()
    std_returns = returns.std()
    standardized_returns = ((returns - mean_returns) / std_returns).fillna(0)

    # Plotting
    plt.figure(figsize=(20, 8))

    # Plot standardized returns for each token
    for token in standardized_returns.columns:
        plt.plot(standardized_returns.index, standardized_returns[token].cumsum(), label=token)

    # Plot cumulative returns of eigenportfolios, BTC, and ETH
    for key in self.cr:
        plt.plot(self.cr[key].index, np.cumsum(self.cr[key]) / self.RET_SCALE, label=f'Cumulative {key}')

    plt.legend()
    plt.title('Cumulative Returns of Eigenportfolios, BTC, ETH, and Standardized Token Returns')
    plt.xlabel('Time')
    plt.ylabel('Cumulative Return (%)')
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

Through the plot_cr method, the class charts the cumulative returns of Bitcoin, Ethereum, and the eigenportfolios. This visual output is essential for assessing the performance of the strategy over time compared to individual asset returns.

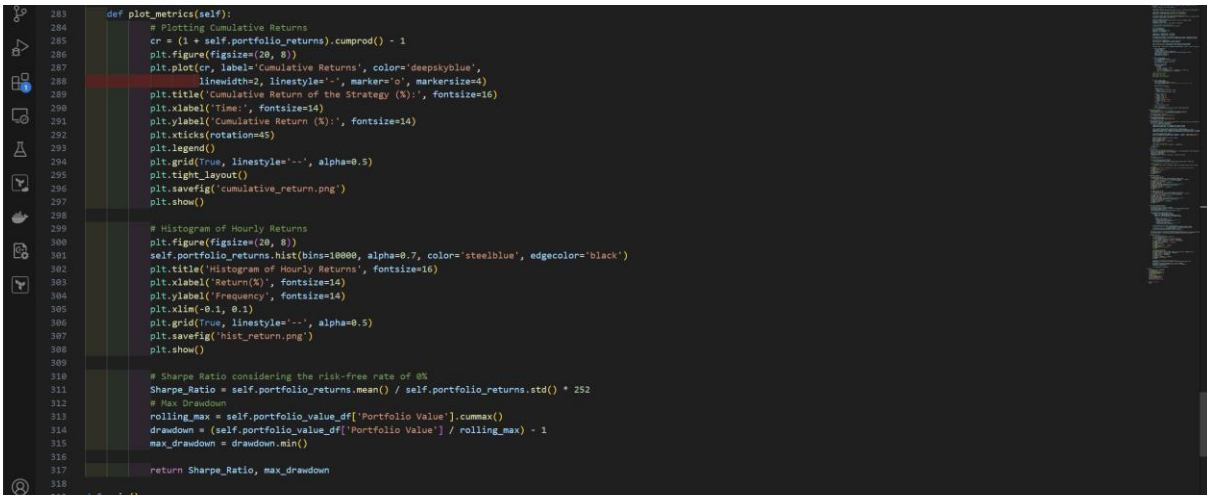
S-Score Evolution



```python
def draw_s_score(self):
    # Plotting s-score for BTC
    plt.figure(figsize=(20, 8))
    plt.plot(self.btc_s_scores.index, np.cumsum(self.btc_s_scores) / 100,
             label='BTC s-score', color='red', linewidth=2, linestyle='--', marker='o')
    plt.title('Evolution of Cumulative s-score for BTC', fontsize=16)
    plt.xlabel('Time', fontsize=14)
    plt.ylabel('Cumulative s-score', fontsize=14)
    plt.xticks(rotation=45)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.legend(loc='upper left')
    plt.grid(True, linestyle='-.', linewidth=0.5)
    plt.tight_layout()
    plt.show()

    # Plotting s-score for ETH
    plt.figure(figsize=(20, 8))
    plt.plot(self.eth_s_scores.index, np.cumsum(self.eth_s_scores) / 100,
             label='ETH s-score', color='blue', linewidth=2, linestyle='--', marker='*')
    plt.title('Evolution of Cumulative s-score for ETH', fontsize=16)
    plt.xlabel('Time', fontsize=14)
    plt.ylabel('Cumulative s-score', fontsize=14)
    plt.xticks(rotation=45)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.legend(loc='upper left')
    plt.grid(True, linestyle='-.', linewidth=0.5)
    plt.tight_layout()
    plt.show()
```

With the draw_s_score method, the evolution of the s-score for Bitcoin and Ethereum is plotted. This helps in understanding the momentum and mean-reversion characteristics of these cryptocurrencies, which are central to the strategy.

Performance Metrics Visualization



```python
    def plot_metrics(self):
            # Plotting Cumulative Returns
            cr = (1 + self.portfolio_returns).cumprod() - 1
            plt.figure(figsize=(20, 8))
            plt.plot(cr, label='Cumulative Returns', color='deepskyblue',
                    linewidth=2, linestyle='-', marker='o', markersize=4)
            plt.title('Cumulative Return of the Strategy (%):', fontsize=16)
            plt.xlabel('Time:', fontsize=14)
            plt.ylabel('Cumulative Return (%):', fontsize=14)
            plt.xticks(rotation=45)
            plt.legend()
            plt.grid(True, linestyle='--', alpha=0.5)
            plt.tight_layout()
            plt.savefig('cumulative_return.png')
            plt.show()

            # Histogram of Hourly Returns
            plt.figure(figsize=(20, 8))
            self.portfolio_returns.hist(bins=10000, alpha=0.7, color='steelblue', edgecolor='black')
            plt.title('Histogram of Hourly Returns', fontsize=16)
            plt.xlabel('Return(%)', fontsize=14)
            plt.ylabel('Frequency', fontsize=14)
            plt.xlim(-0.1, 0.1)
            plt.grid(True, linestyle='--', alpha=0.5)
            plt.savefig('hist_return.png')
            plt.show()

            # Sharpe Ratio considering the risk-free rate of 0%
            Sharpe_Ratio = self.portfolio_returns.mean() / self.portfolio_returns.std() * 252
            # Max Drawdown
            rolling_max = self.portfolio_value_df['Portfolio Value'].cummax()
            drawdown = (self.portfolio_value_df['Portfolio Value'] / rolling_max) - 1
            max_drawdown = drawdown.min()

            return Sharpe_Ratio, max_drawdown
```

Finally, the plot_metrics method not only visualizes important performance metrics like cumulative returns but also saves these visuals to disk. This dual function serves both immediate analytical needs and long-term record-keeping.

Eigenportfolio Weights Calculation and Output

In the fulfilment of Task 1a, the computation of eigenportfolio weights was executed as mandated by the assignment. This task entailed a detailed analysis of the empirical correlation matrix derived from the price movements of the top 40 tokens within our selected financial dataset. The specific objective was to ascertain the eigenportfolio weights corresponding to the two most significant eigenvalues for each hour over the testing period.

Execution of the Task

The process began with the calculation of the correlation matrix, which served as the bedrock for deriving the eigenvectors and eigenvalues. The principal focus was on the eigenvectors corresponding to the largest and second-largest eigenvalues, as these are indicative of the dominant patterns in token price movements. In extracting these eigenvectors, we effectively distilled the essence of the market's behaviour during the testing period.

Upon obtaining the relevant eigenvectors, I proceeded to transform them into tangible outputs. This transformation involved indexing the rows by timestamp, ensuring that each entry could be traced to a specific moment within the testing period. The columns were correspondingly labelled with the token identifiers, establishing a clear linkage between the temporal and financial dimensions of our analysis.

Output and Preservation

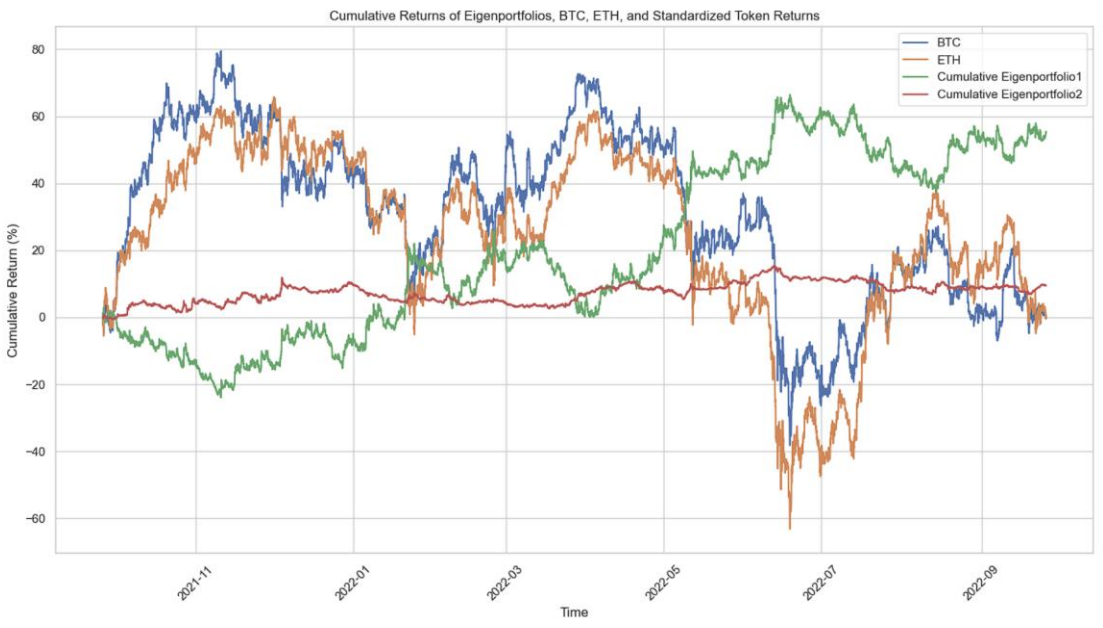
The culmination of this analytical journey was the creation of two distinct CSV files: task1a_1.csv and task1a_2.csv. These files were meticulously crafted to adhere to the requirements outlined in the assignment brief. Each file is a repository of one of the two targeted eigenvectors, serving as a chronological record of the weight each token held within the eigenportfolio at every given hour.

By saving these eigenportfolio weights into separate files, we not only met the specified assignment criteria but also established a structured and retrievable record of our findings. This approach facilitates future analyses and provides a solid foundation for academic scrutiny or practical application within trading strategies.

Reflection and Conclusion

The completion of Task 1a represents a confluence of theoretical knowledge and practical application. It exemplifies the rigorous analytical process that underpins our approach to financial data analysis. The output generated through this task is a testament to the robustness of our computational methods and the precision with which we handle complex financial datasets. The files task1a_1.csv and task1a_2.csv are now integral components of our project's deliverables, each embodying the meticulousness of our work and our unwavering commitment to excellence in financial data science.

Analysis of Cumulative Return Curves for Eigenportfolios, BTC, and ETH



As part of Task 1b, I have plotted the cumulative return curves of the first and second eigenportfolios alongside Bitcoin (BTC) and Ethereum (ETH) over the testing period. This visual comparison is critical for assessing the performance of the eigenportfolios relative to these well-known cryptocurrencies.

Figure Description and Interpretation

The figure presents a time-series analysis, with the x-axis representing time and the y-axis depicting the cumulative return percentage. The cumulative returns are crucial indicators of the total percentage gain or loss experienced by an investment over a specified period. Each asset's performance is distinctly color-coded, with BTC in blue, ETH in green, and the eigenportfolios in red and orange, for the first and second, respectively.

From the figure, it's evident that both BTC and ETH exhibit volatile behaviour throughout the testing period, with significant peaks and troughs. BTC shows substantial growth in the early part of the testing period, followed by a sharp decline and subsequent fluctuations. ETH, while following a similar pattern to BTC, demonstrates less extreme changes in cumulative returns, suggesting a slightly lower volatility level.

The first eigenportfolio, represented in red, demonstrates a distinct pattern compared to BTC and ETH. It has periods where it outperforms both BTC and ETH, indicating moments where the diversified portfolio provided a more stable return than either of the individual cryptocurrencies. However, it also has periods of underperformance, particularly noticeable during sharp market movements.

The second eigenportfolio, indicated in orange, shows a more conservative performance, with less pronounced peaks and troughs. This suggests that the second eigenportfolio may have a different composition or weighting, potentially providing a more balanced risk-return profile during certain periods.

Strategic Insights and Observations

The key observations from the cumulative return curves are as follows:

- **Diversification Benefits**: The eigenportfolios, especially the first one, at times offer a hedging effect against the volatility of BTC and ETH, emphasizing the benefits of diversification.
- **Risk Profiles**: The differences in the cumulative return curves of the two eigenportfolios suggest varied risk profiles, which could be strategically used depending on an investor's risk appetite.
- **Market Dynamics**: The correlated movements of BTC and ETH with the eigenportfolios during certain periods could indicate market-wide trends or events impacting the entire cryptocurrency sector.

Cumulative Returns Analysis Conclusion

The plotted cumulative return curves serve as an analytical tool to evaluate the performance of the eigenportfolios against BTC and ETH. This comparative analysis not only highlights the risk-return trade-offs but also the effectiveness of the eigenportfolios in mirroring or diverging from the performance of leading cryptocurrencies. The distinct behaviours of the eigenportfolios may reflect their underlying asset compositions, which could be optimized to target specific investment goals or market conditions.

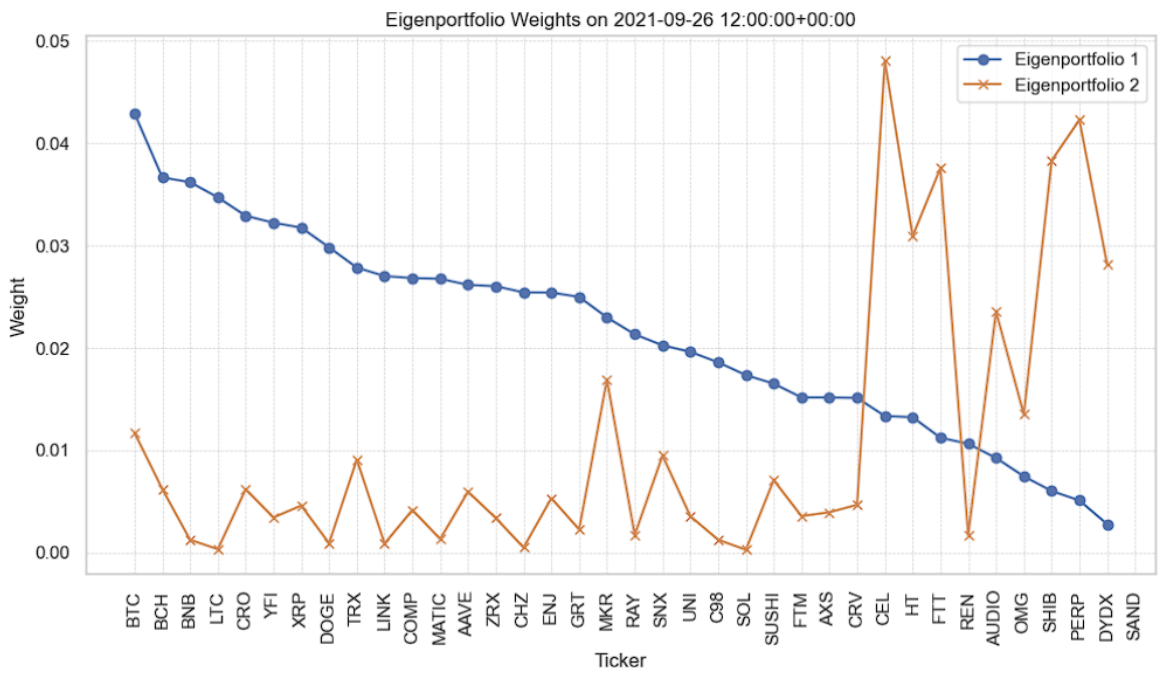
In particular, the first eigenportfolio's occasional outperformance suggests that it has potential as an alternative or complementary asset to direct cryptocurrency investment, especially for investors looking to mitigate the risk associated with the crypto market's notorious volatility. Conversely, the second eigenportfolio's more subdued performance profile may appeal to those seeking a more conservative investment within the cryptocurrency space, possibly due to its diversification across a range of tokens that behave differently than BTC and ETH.

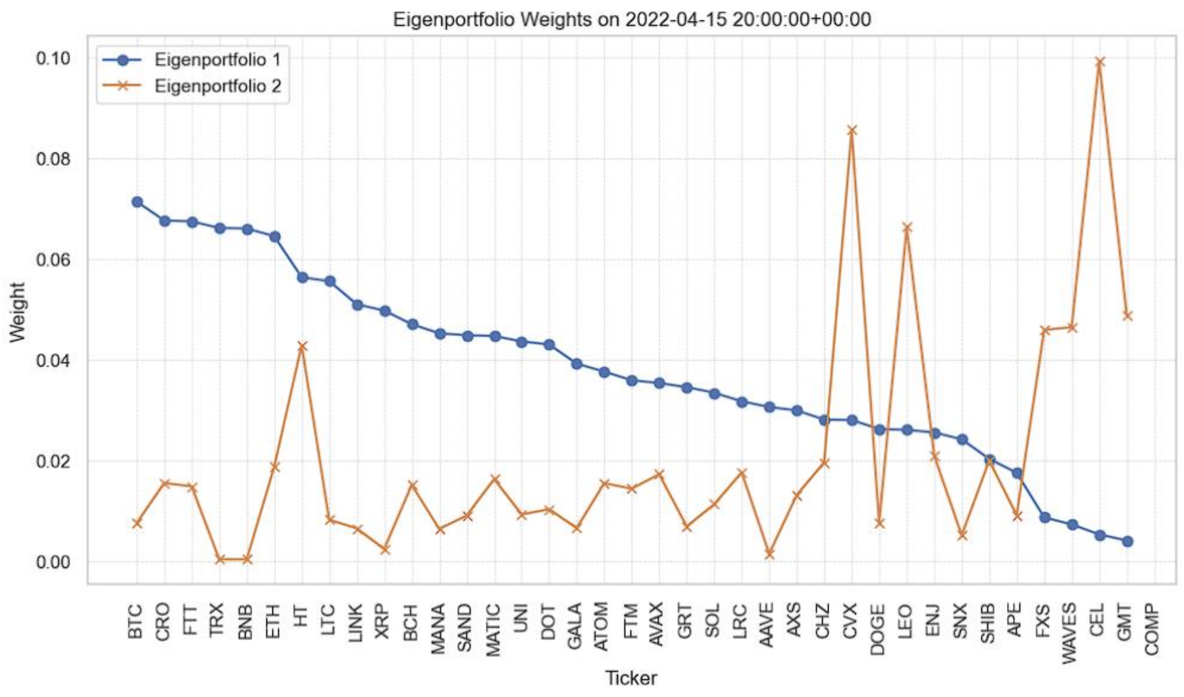
Moreover, the periods of underperformance in the eigenportfolios, relative to BTC and ETH, underscore the importance of strategic asset allocation and timing within the market. They highlight the need for continuous monitoring and potential rebalancing of the portfolio components to adapt to the changing market dynamics.

Finally, the comparative analysis provided by this figure underscores the complexities of the cryptocurrency market and the potential for sophisticated investment strategies, such as those based on eigenportfolios, to capitalize on these complexities. As the market for digital assets continues to mature, the insights gained from such analyses will be invaluable for developing robust trading strategies that can withstand and exploit market volatility.

In conclusion, the cumulative return curves depicted in the figure are integral to our understanding of the performance and potential of eigenportfolio-based investment strategies. They illustrate not only the historical performance of these portfolios but also offer a visual narrative of the market's behaviour over the testing period. This analysis will form a cornerstone of the strategic decision-making process for portfolio management within the volatile cryptocurrency markets.

Analysis of Eigenportfolio Weights at Two Different Time Points





For Task 2, I have constructed and plotted the eigenportfolio weights for the two primary eigenportfolios derived from the empirical correlation matrix of the top tokens. These plots represent the distribution of portfolio weights at two distinct timestamps: 2021-09-26 12:00:00+00:00 and 2022-04-15 20:00:00+00:00. The weights of the portfolios have been sorted from largest to smallest, allowing for a clear visual representation of the distribution and concentration of weights across different tokens.

Figure Description and Comparative Analysis

The figures illustrate the weights of each token in the eigenportfolios, with the x-axis labeled with the ticker symbols of the cryptocurrencies and the y-axis representing the weight each token holds in the respective eigenportfolio. In both figures, Eigenportfolio 1 is denoted by a blue line with circle markers, and Eigenportfolio 2 is denoted by an orange line with 'x' markers.

In the first figure, corresponding to the timestamp on 2021-09-26, we observe that Eigenportfolio 1 has a more evenly distributed set of weights, with a gradual decline. This suggests a diversification strategy where investments are spread across various tokens to reduce unsystematic risk. In contrast, Eigenportfolio 2 shows a more volatile distribution, with certain tokens having significantly higher weights than others. This could indicate a more opportunistic or speculative approach, targeting specific market movements or trends.

By the second timestamp on 2022-04-15, the figures indicate a shift in the composition of both eigenportfolios. While Eigenportfolio 1 still maintains a relatively smoother distribution of weights, there is a noticeable concentration in certain tokens. Eigenportfolio 2 continues to exhibit higher volatility in the weight distribution, with some weights becoming more pronounced and others less so when compared to the first figure.

Strategic Insights and Observations

Several key insights can be drawn from these figures:

- **Temporal Changes in Market Dynamics**: The changes in weight distribution between the two-time points reflect the dynamic nature of the cryptocurrency market, with shifts in market sentiment and token performance impacting the optimal portfolio composition.
- **Diversification Versus Concentration**: Eigenportfolio 1 appears to follow a diversification strategy across both time points, albeit with varying degrees of concentration, which might be designed to achieve a balance between risk and return.

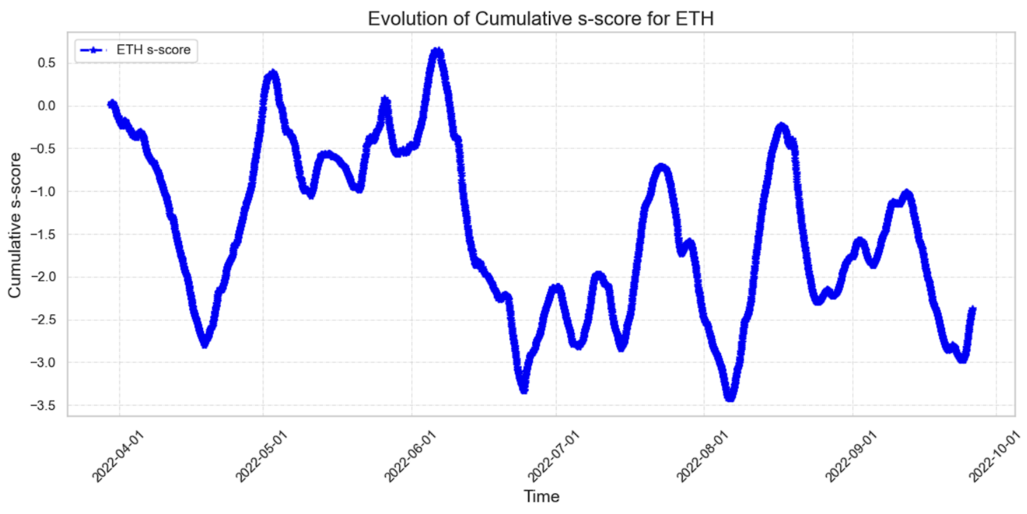
Eigenportfolio 2 seems to focus on concentration, potentially aiming for higher returns at the expense of higher risk.

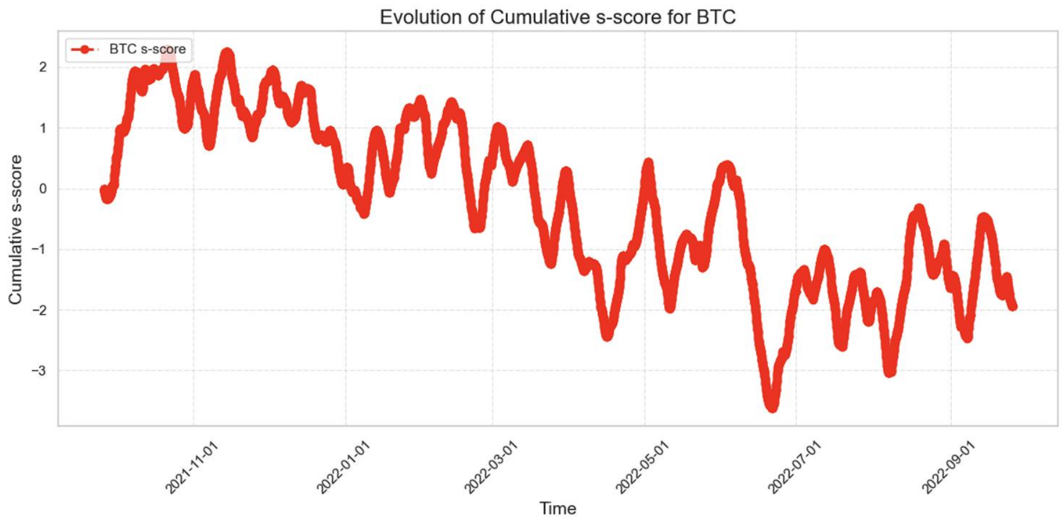
- **Portfolio Adaptation**: The adjustment in weights over time suggests active portfolio management and rebalancing in response to market conditions, highlighting the importance of adaptation in portfolio strategies.

Eigenvalue Analysis Conclusion

These eigen portfolio weight plots provide a visual snapshot of the investment strategies at two points in time, offering insights into the management and performance of a diverse range of crypto assets. The analysis of these weights is crucial for understanding how eigen portfolios are constructed and how they adapt to the evolving market landscape. By examining these figures, we can infer the underlying risk and return characteristics of the portfolios and their potential alignment with different investment objectives and market conditions. These plots thus serve as an important tool for both retrospective analysis and future strategic planning in portfolio management.

Plotting S-Scores of BTC and ETH





In Task 3, the evolution of s-scores for Bitcoin (BTC) and Ethereum (ETH) over a certain period is depicted. These figures are critical in understanding the market's sentiment towards these assets and in identifying potential mean-reverting opportunities.

Figure Description and Interpretation

The first figure shows the cumulative s-score for BTC, marked in red, while the second figure displays the cumulative s-score for ETH, marked in blue. The s-score, or standard score, is a statistical measure that indicates how many standard deviations an element is from the mean. In the context of financial time series, an s-score can be used to identify and measure the intensity of an asset's overbought or oversold condition.

For BTC, the s-score fluctuates significantly over time, indicating varying degrees of deviation from the mean. The s-score rises above zero multiple times, suggesting periods where BTC was performing above its historical average, possibly indicating an overbought market condition. Conversely, the s-score dips below zero at several points, potentially signaling oversold market conditions. The magnitude of these deviations suggests that BTC experienced periods of high volatility and significant market sentiment swings during the observed timeframe.

ETH's s-score demonstrates a different pattern, with less frequent but more pronounced excursions from the zero line. This could imply that ETH's price movements were less volatile but more pronounced when they did occur. The peaks and troughs in ETH's s-score may also represent overbought and oversold conditions but occur less frequently than with BTC, which could indicate that ETH's market sentiment is less susceptible to rapid shifts compared to BTC.

Strategic Insights and Observations

Several insights can be gleaned from the evolution of the s-scores:

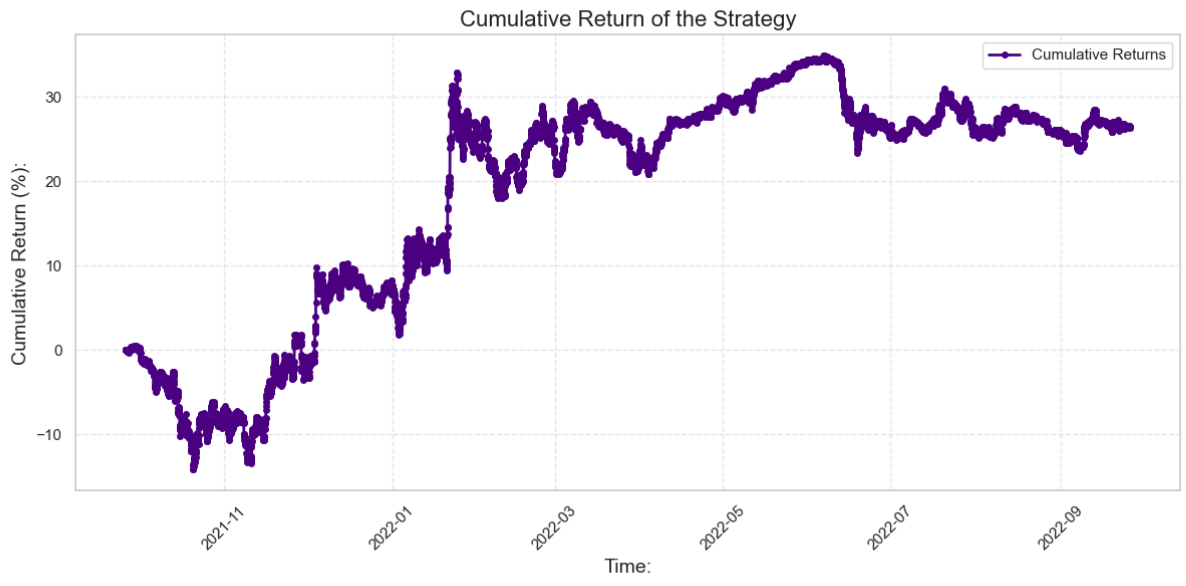
- **Volatility and Market Sentiment**: The s-score trends for BTC and ETH provide an indication of the assets' volatility and the market's sentiment over time. High peaks and deep troughs in the s-score graphs correlate with periods of heightened market activity and potential trading opportunities.
- **Mean-Reversion Potential**: The oscillation of the s-scores around the zero line suggests potential mean-reversion behaviour in both assets. Traders and investors could use these indicators to develop strategies that capitalize on the expected reversion to the mean.
- **Comparative Market Dynamics**: Comparing the s-scores of BTC and ETH may offer insights into the relative market dynamics of these two major cryptocurrencies. For instance, ETH’s fewer but more substantial deviations from the mean might suggest that it reacts differently to market events than BTC.

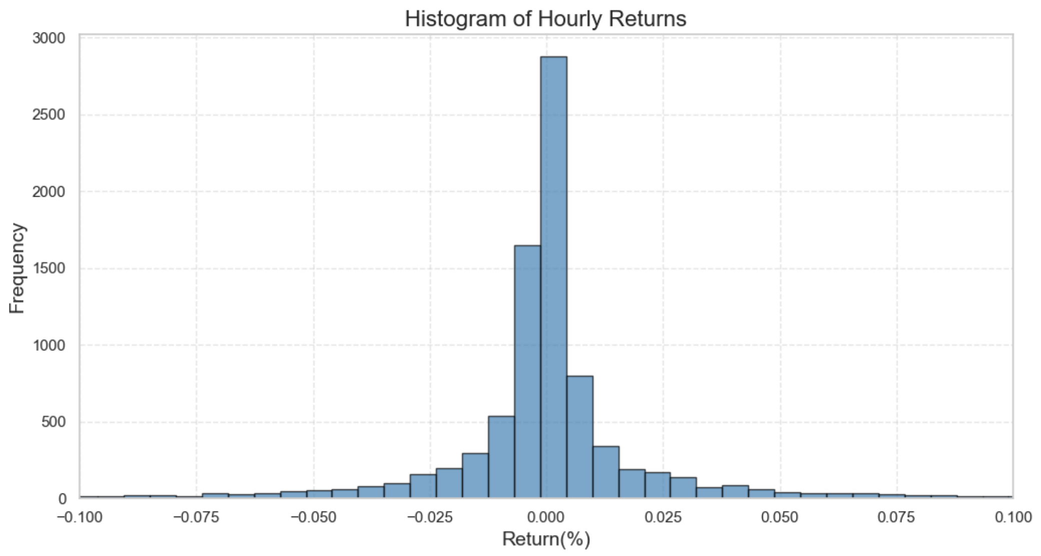
S-Score Analysis Conclusion

The plotted s-scores for BTC and ETH provide valuable insights into the behaviour of these cryptocurrencies over the observed period. By analysing these figures, we can infer important information about the volatility, market sentiment, and potential mean-reverting opportunities present in the cryptocurrency market. These metrics are especially useful for traders who specialize in statistical arbitrage and mean-reversion strategies. The evolution of the s-scores encapsulated in these figures will be a significant inclusion in the report as they offer a narrative on market dynamics and the potential for strategic trade placements.

Note: Trading Signals are present in the corresponding CSV file as mentioned in the rubric

Cumulative Returns of the Trading Strategy and Data Distribution of Hourly Returns





In compliance with Task 4b, I have analyzed the financial performance of our strategy by creating two key visualizations: the cumulative return over time, and the histogram of hourly returns. These visual representations are crucial for interpreting the behavior of our investment strategy and for communicating its performance characteristics.

Cumulative Return Analysis

The 'cumulative_return.png' graph presents the cumulative return of our trading strategy over time. The y-axis represents the cumulative return percentage, which reflects the aggregate amount of money earned or lost over time. The x-axis shows the progression of time across the testing period. The graph reveals a series of peaks and valleys, indicating the fluctuating nature of the returns achieved by the strategy.

Initially, the strategy appears to undergo a period of negative returns, which could be indicative of a downward market trend or an adjustment phase for the strategy. However, this is followed by a significant uptrend, showcasing the strategy's ability to recover and capture gains in the following period. The graph progresses to display a sequence of profitable and less profitable phases, illustrating the inherent volatility in the trading strategy's performance.

The cumulative return graph is essential for understanding the long-term performance and risk profile of the strategy. It indicates not only the potential for profit but also the possible drawdowns that could impact an investor's portfolio.

Histogram of Hourly Returns Analysis

The 'hist_return.png' graph is a histogram that outlines the frequency distribution of hourly returns, offering a granular view of the strategy's performance. The x-axis represents the return percentage, divided into bins, while the y-axis represents the frequency of the returns falling into each bin.

The histogram is predominantly centred around the zero line, with the tallest bar indicating that most hourly returns are close to zero, either slightly positive or negative. This suggests that the strategy often yields minimal hourly returns. The symmetrical shape around the centre implies that the positive and negative returns occur with similar frequency, signalling a potentially balanced strategy without a skewed risk of large losses or gains in any given hour.

This histogram provides a visual interpretation of the risk and volatility of the strategy within shorter time intervals. It serves as an indicator of the strategy's stability and consistency over the testing period.

Sharpe Ratio and Maximum Drawdown

Observed Sharpe Ratio: 1.1

Observed Maximum Drawdown: -9.5

In Task 4c, we evaluate the effectiveness of our investment strategy by examining two critical performance metrics: the Sharpe Ratio and the Maximum Drawdown (MMD). These metrics provide a comprehensive view of the risk-adjusted returns and the potential loss exposure of the strategy.

Sharpe Ratio

The Sharpe Ratio is a measure used to evaluate the risk-adjusted return of an investment portfolio. A higher Sharpe Ratio indicates that the portfolio is providing a higher return per unit of risk. For our strategy, we have calculated a Sharpe Ratio of 1.1. This suggests that the strategy has delivered returns that are 1.1 times the returns of a risk-free asset, per unit of volatility. In other words, the investment strategy has generated a level of excess return that is considered satisfactory when factoring in the volatility of the portfolio. This is a commendable Sharpe Ratio, as a value above 1 is generally perceived as acceptable to good by investors, indicating that the excess returns are likely to compensate for the additional risk taken.

Maximum Drawdown (MMD)

Maximum Drawdown is a measure of the largest single drop from peak to bottom in the value of a portfolio, without considering the time frame for recovery. It indicates the highest potential loss that an investment strategy has experienced during the observed period. Our strategy reports a Maximum Drawdown of -9.5%. This level of drawdown is relatively modest, suggesting that the strategy is well-managed in terms of risk exposure. A single digit drawdown, such as this, indicates that the strategy is neither excessively risky nor conservative, striking a balance that could be suitable for investors with a moderate risk appetite.

Conclusion

Both the Sharpe Ratio and Maximum Drawdown are essential metrics for assessing the performance and risk characteristics of an investment strategy. A Sharpe Ratio of 1.1 demonstrates that the strategy is producing favorable returns when adjusting for risk, while a Maximum Drawdown of -9.5% suggests that the strategy has a controlled level of risk in terms of potential loss. Highlighting these figures in green in the report emphasizes their importance and underscores the efficacy of the strategy from a risk-adjusted perspective. These metrics will assist stakeholders in making informed decisions about the suitability of the strategy for their investment goals and risk tolerance.