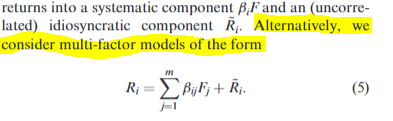
**Project Report:**

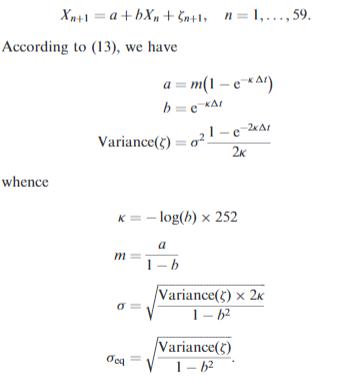
In this project we have implemented the Statistical arbitrage strategy proposed in Avellaneda and Lee 2010 on the universe of 40 crypto currencies sourced from FTX and analyzed the performance of the strategy

In this project we have used multiple classes and functions detailed below to implement the strategy:

* Initially we have calculated the factors that are responsible for the returns based on two principal components of PCA.

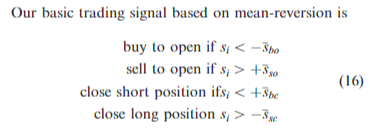


* After the factors are determined we have individually regressed it with stock returns and calculated the residual returns unexplained by the factors. The unexplained risk by factors is considered as the idiosyncratic component and assumed to be responsible for Alpha generation
* Also, we have auto regressed the cumulative residuals to find out the parameters that are used in the S score.

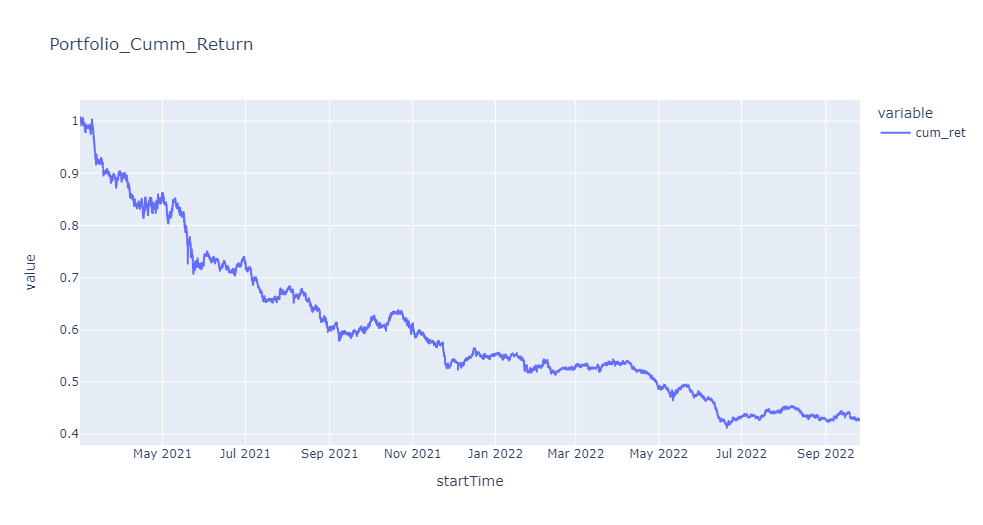
* Once the S scores are generated we have applied the signal function to check if a particular stock needs to be long or short or close (+1,-1,0) respectively.





Based on the above brief overview of the flow of the implementation, we have traded in each hour based on the universe of the top market cap coins/tickers between Mar 2021 – Sep 2022 where we have generated signal based on the data of past 240 hours/10 days.

Cumulative return based on this strategy is -60%, if we have invested 1 dollar at the start, we would end up with 40 cents by the end of period. These results are in sync with market if we consider BTC as market benchmark (which fell from around $49,000 to around $19,000, refer appendix for the trend of BTC). Sharpe Ratio of -0.03143 with benchmark return considered as 0. Maximum Drawdown for the period is 256%.



Return Distribution is negatively skewed with near normal distribution which is inline with the portfolio performance.

Chart, histogram

Description automatically generated

**Results:**

Plots of cumulative return curves of the 4 assets over the testing period in one figure.

In the below we had to scale the weights of the Eigen Portfolio in order to get the returns inbound

Chart, line chart

Description automatically generated

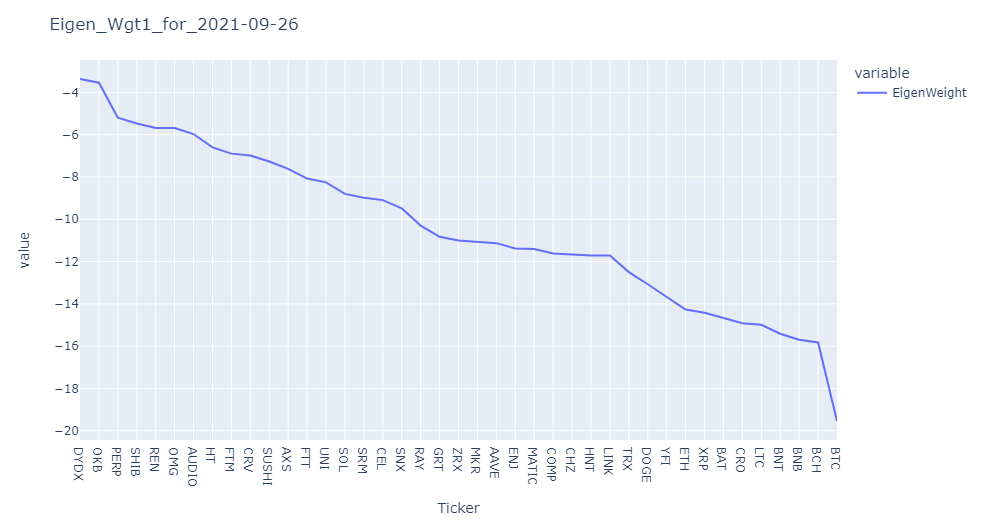
BTC, ETH cumulative return

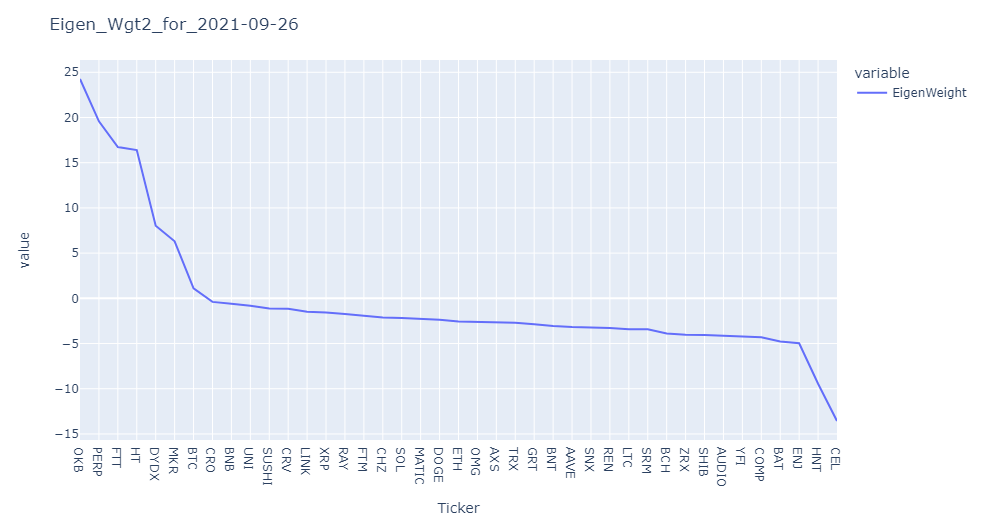
Graphical user interface, chart, scatter chart

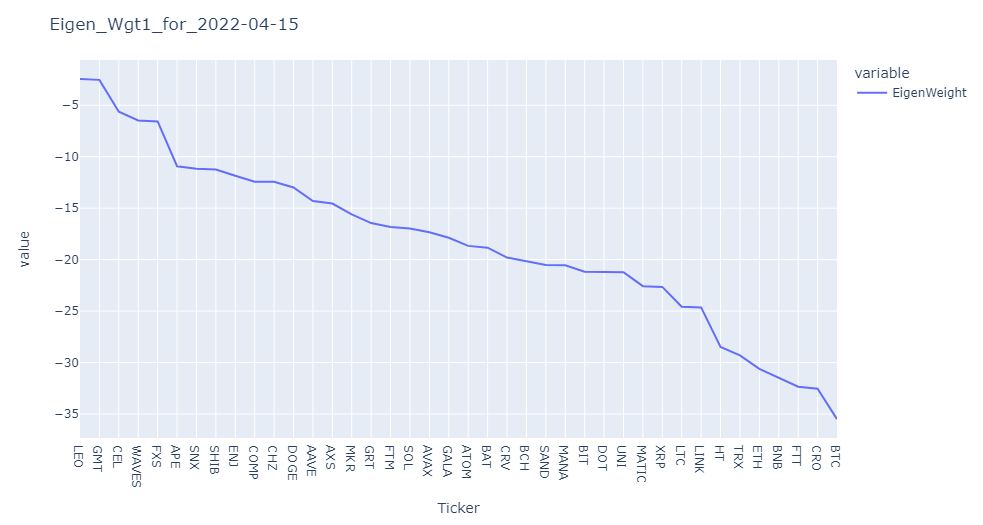
Description automatically generated

Plots of Eigen Vector weights in descending order for requested timepoints

The first eigen portfolio weight curve is steep like a line, whereas the second eigen portfolio weights is in the form of titled curve



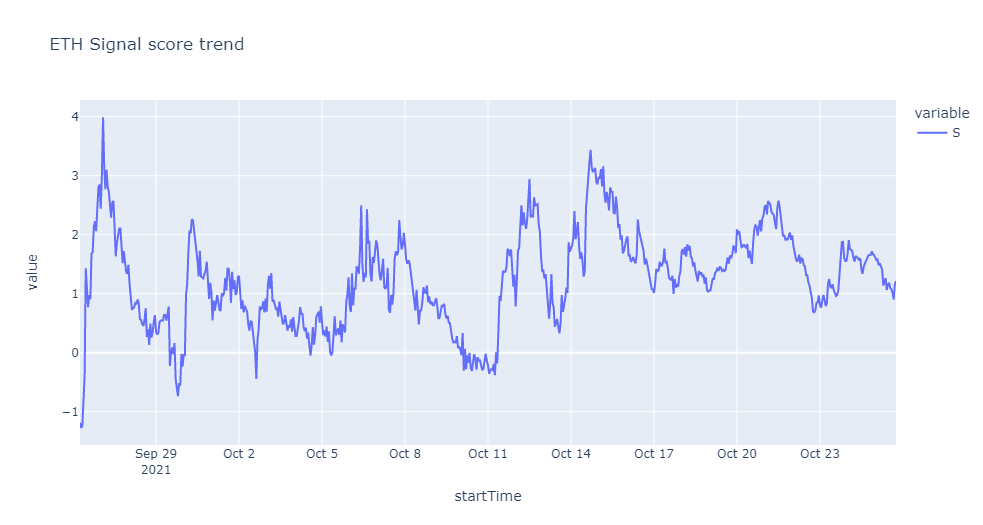




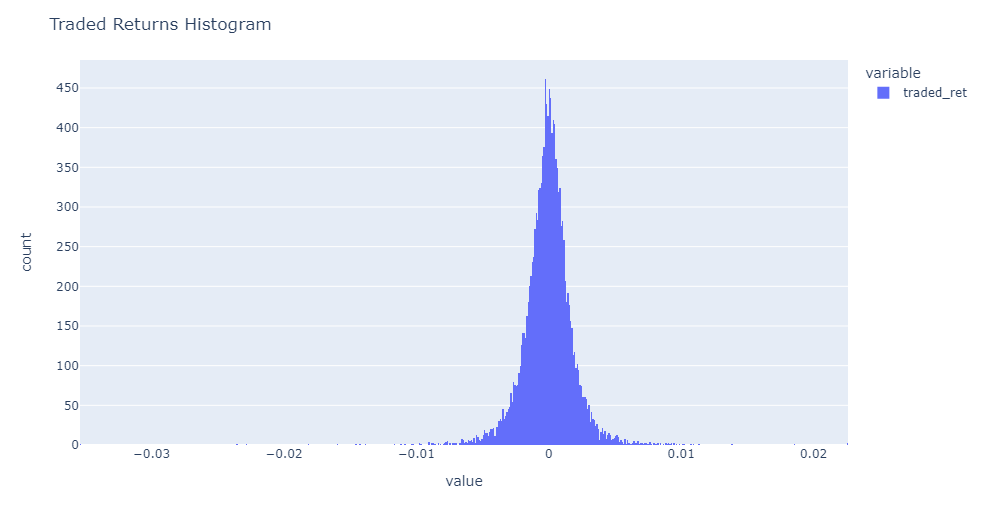
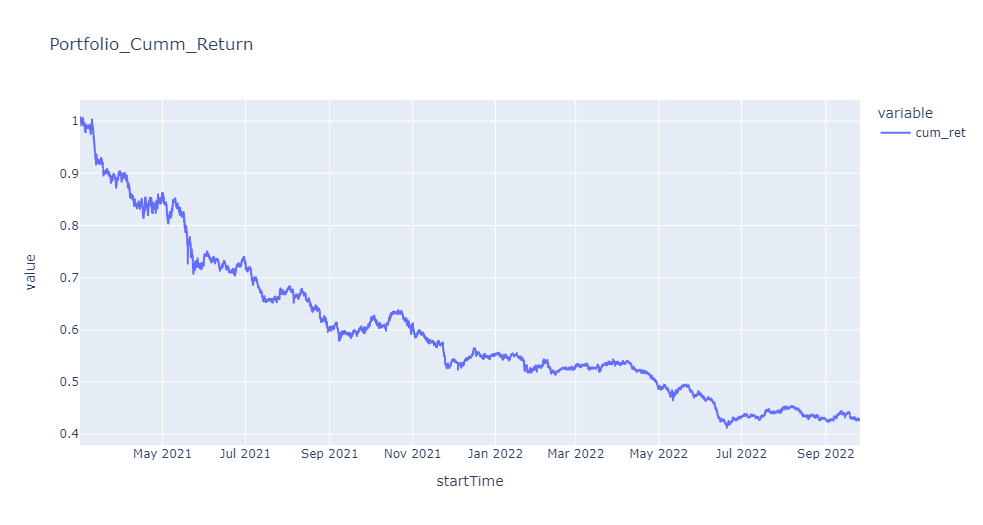
Chart, line chart

Description automatically generated

Plot the evolution of s-score of BTC and ETH from 2021-09-26 00:00:00 to 2021-10-25 23:00:00



Plots of cumulative return and traded return distribution

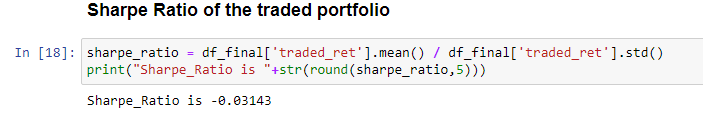


Zoomed image of histogram:

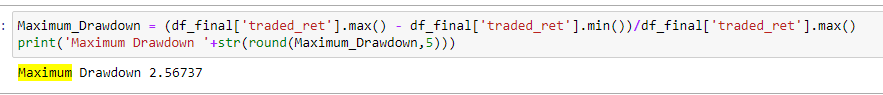
Chart, histogram

Description automatically generated

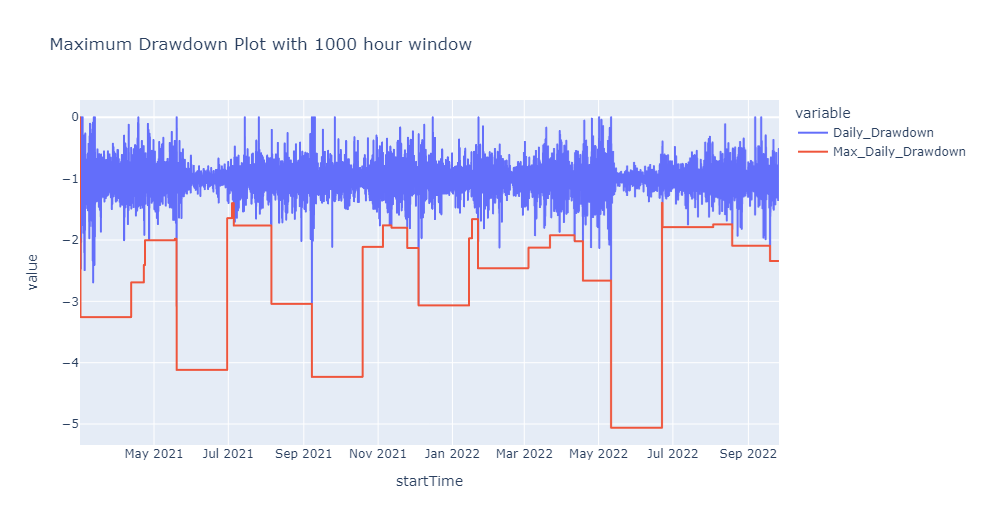
Sharpe Ratio: -0.03143 is the Sharpe ratio of the portfolio based on traded signals

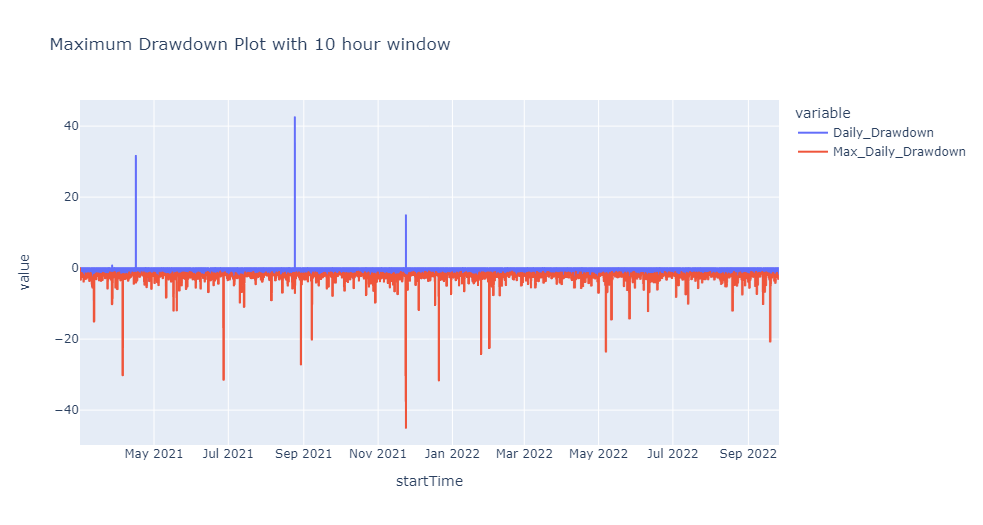
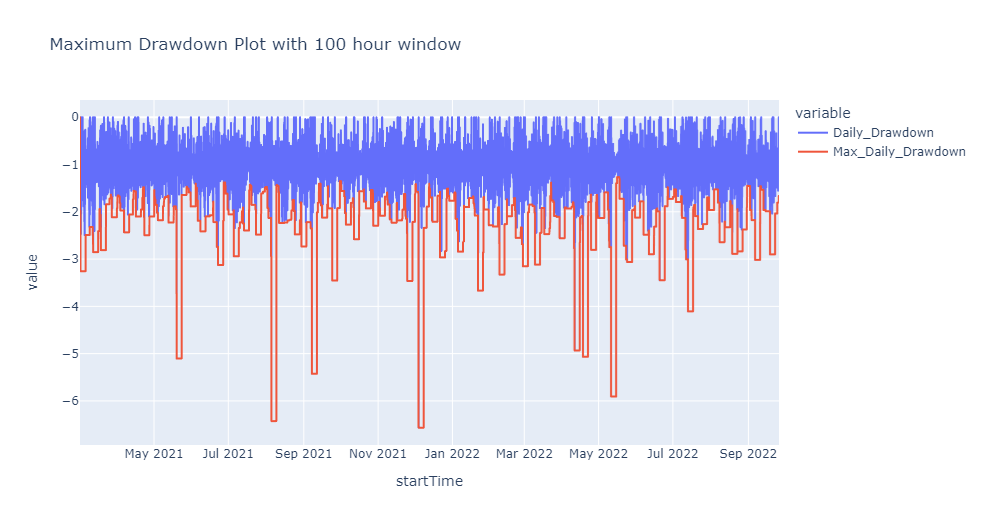


Maximum Drawdown: 2.567 i.e. 256%



Below are the plots of Maximum Drawdown for various rolling windows





**Code Implementation:**

We have loaded top 40 tickers each hour and price data in the dataframe in df\_price and df\_ticker dataframes respectively. Then we have created a column in df\_ticker (﻿ticker\_list) which contains list of all the top 40 tickers and merged both the dataframes.

We have iterated over all the rows in merged dataframe created in above step to perform all the below tasks on all the timepoints.

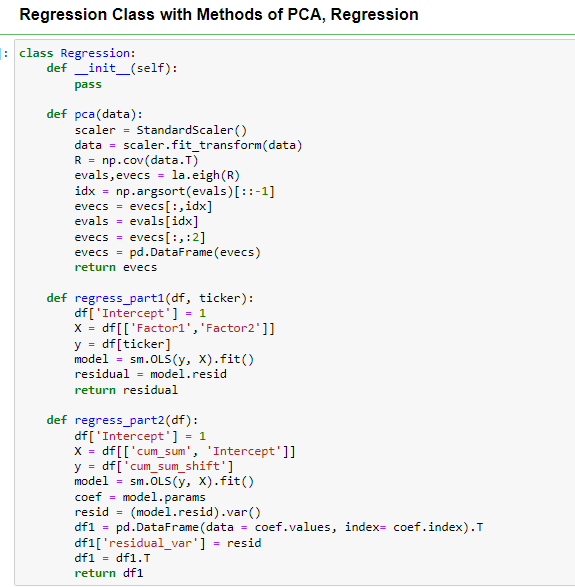
Inside loop we are creating new dataframe for each timepoint by storing data pertaining to only top 40 tickers in that timepoint, then we are creating a time difference column which is basically giving difference between that timepoint and time of data each row is storing and then we are only keeping the data where this difference is less than 240 and equal or more than 240.

The dataframe generated above are created for each datapoint and stored in a dictionary “dict1”.

We have calculated daily returns of each ticker in the “data” dataframe.

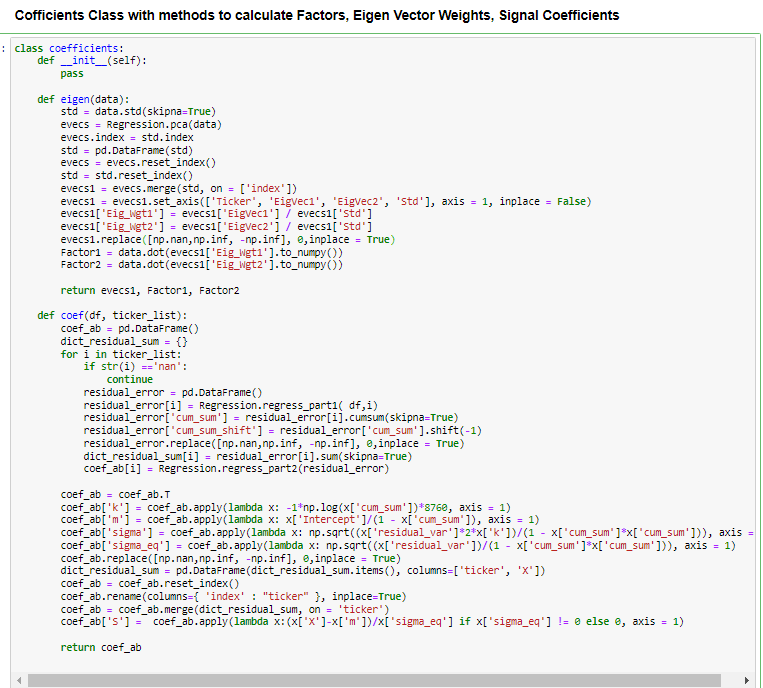
**Classes & Function:**

1. **Regression Class:** 
   * **PCA:** The function takes dataframe containing returns of all the tickers and perform principal component analysis and returns 2 eigen vectors (﻿evecs) as output dataframe.
   * **Regression Part 1:** The function takes dataframe (containing ticker returns and eigen portfolio returns) and ticker as input and performs regression of portfolio return of eigen portfolios with return of each ticker and it gives residual error series (﻿residual) as output.
   * **Regression Part 2:** The function takes dataframe (containing cumulative sum of residual error with shifted cumulative sum of residual error) performs regression of cumulative sum of residual error with shifted cumulative sum of residual error and it gives variance of residual error (﻿residual\_var), regression beta (b) and intercept (a) as output dataframe.

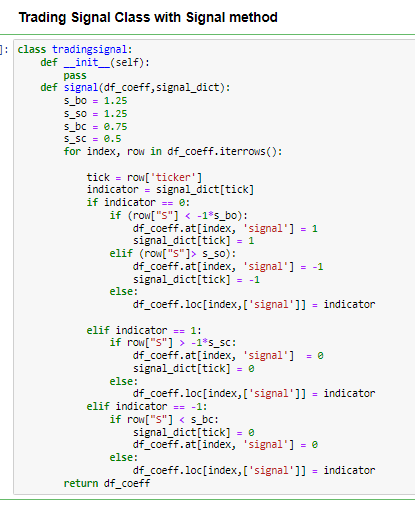


1. **Coefficient Class:**
   * **Eigen:** The function takes dataframe containing returns of each ticker and calls PCA function which returns 2 top eigen vectors, then we are calculating standard deviation of returns for each ticker and dividing values of eigen vectors with standard deviation to get weights of eigen portfolio.
   * **Coef:** Coef function takes dataframe of returns and ticker list as input and

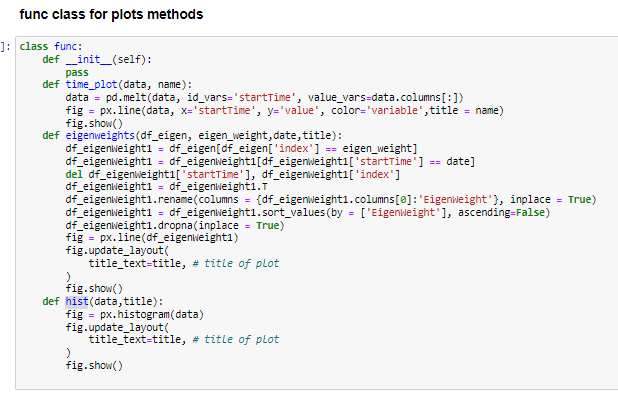
calls ﻿regress\_part1 function with gives residual series which is then modified by adding 2 additional columns cumulative sum of residual error with shifted cumulative sum of residual error. The modified dataframe is then passed to ﻿regress\_part2 which gives regression alpha, beta, and variance of residual error. The results are then utilized to calculate k, m, sigma, sigma\_eq and s-score.



1. **Trading Signal:** 
   * **Signal:** The function takes dataframe containing coefficients (a, b, residual\_var, k, m, sigma, sigma\_eg, s-score etc. returned by Coef function) and uses s-score to determine whether to long or short stock and checks if a long or short positions needs to be closed. This is accomplished by a signal dictionary we have created which keeps track of our position on every ticker.



1. **Func:**
   * **time\_plot:** This function plots the dataframe using plotly.
   * **Eigenweights:** Thisfunction plots the eigen weights in descending order for a particular data.
   * **Hist:** This function plots the histogram distribution for the data.



**Appendix:**



