AlphaFactorGenerator

xz

July 5, 2023

Abstract

This is a document for AlphaFactorGenerator

Contents

1 Document For AlphaFactorGenerator

1.1 Introduction

This is a tool designed to assist in Alpha production research. Users can prepare data in a fixed format locally, and the program can automatically generate factors based on the given input. The design intention of the product is to separate data preparation, calculation logic, and factor performance evaluation in Alpha research and development, leaving data preparation and factor evaluation to the users, while extracting and summarizing common logic parts in factor construction process.

* This tool summarizes a wide range of common cross-sectional signal processing methods, making it convenient to generate signals for production research.

* This tool optimizes the process of factor generation and accelerates the factor search process.

* This tool provides calculation logic based on univariate and bivariate analysis.

* This tool can help extract temporal and cross-sectional features.

1.2 Quick Start

1.2.1 Install Docker

https://www.runoob.com/docker/windows-docker-install.html. This is a web site show you how to install docker.

1.2.2 Download image

https://pan.baidu.com/s/1yhDU3qQdyQp3_HLJnhMnQQ password:7ljg

1.2.3 Run Demo

- Start the image
sudo docker run -it centos-alphafactorgen-0.0.3 /bin/bash
- Change to the directory
cd /home/factorgen0.0.1
- Run the demo
./EXE
- Check the result
result is stored in /tmp/output*.csv, the first example will generate 4700 new features for the dataframe specified by AA.
- To run other demo or data, you need to change test.cpp, and use compile.sh to compile the EXE. For example, the second example will generate 34000 fetures for two input dataframes.
- If your computer has limited memory, you may encounter a 'core dump'. Please try using a smaller number of formulas for testing.

1.3 Feature Extraction In Alpha Research

In stock Alpha strategy research, we need to generate a set of forecast values as the factor values for a given day, and in most cases, we utilize some other information from the previous n days to assist in generating the factor values for the current day($alpha[di]$).

$$alpha[di] = F(S)$$

where
$alpha[i]$:epresents the factor values of the current day, which is a vector with a length equal to the number of stocks.
F:represents the function we use for decision-making.
S:represents a collection of information from the previous several days, which can be in the form of matrix sets and may even include a network of stock relationships.

Here are some examples:

1. For example, let's consider $S = \{AA\}$, where AA represents the stock's past 20-day returns. We can arrange this information as an $N \times M$ matrix, where $N = 20$ and M represents the number of stocks. If we define the function F to be the maximum value of the last 4 days, then the final signal value can be represented as $FINAL\_max(AA, 4)$..

Here's the Python code:

def FINAL_max(AA, sz): return AA.iloc[-sz:].max()

2. For example, let $S = \{AA, MM\}$. AA represents the stock's past twenty-day returns and can be arranged into an $N \times M$ matrix, where $N = 20$ represents the number of days and M represents the number of stocks. MM is a binary matrix where 1 indicates an increase in the stock's price for the day, and 0 indicates a decrease. MM has the same dimensions as AA. We define F as the standard deviation of the samples where MM is greater than 0. The final signal value is denoted as $FINAL\_fstd(AA, MM)$.

Here's the Python code:

```
def FINAL_fstd(AA, MM): return AA[pd.DataFrame(index=MM.index, columns=MM.columns, data=MM.values >0)].std( axis=0)
```

3. For example, let's consider $S = \{AA, BB\}$. AA represents the stock prices for the past twenty days and can be arranged in an $N \times M$ matrix, where $N = 20$ and M represents the number of stocks. BB represents the trading volume of the stocks and has the same dimensions as AA. The factor F is calculated by weighting AA with BB.

Here's the Python code:

```
def FINAL_weighted(x, y): return (x * y.abs()).sum() / (y.abs()).sum()
```

1.4 Input Format

1.4.1 input data format

Input (parameter AA or BB) is a csv format. It is a dataframe, its column is stock ids, its index is time (usually).

Table 1: input data format

	stock 0	stock 1	···	stock M-1
line 0	X[0][0]	X[0][1]	···	X[0][M-1]
line 1	X[1][0]	X[1][1]	···	X[1][M-1]
···	···	···	···	···
line N - 1	X[N-1][0]	X[N-1][1]	···	X[N-1][M-1]

1.4.2 Note

* The order of rows in the matrix is meaningful, with the last row representing the information closest to the current trading time point. The order should not be reversed.
* The number of rows in the matrix should not be too large.
* The meaning represented by each row of the matrix can be days, quarters, minutes, seconds, or any other unit of time.
* The number of rows in the matrix can be set to a power of 2 if possible (not mandatory) because there may be FFT operations involved, and using a non-power-of-2 size would require padding or resampling to achieve a power-of-2 size.

2 Instruction To Functions

2.1 Operators Category

Based on the output data type or meaning, operators can be roughly divided into the following four categories:

* The first category corresponds to the final output signal (final factor value), and these operators use 'FINAL' as a prefix.

* The second category corresponds to intermediate variables in the output, which can be understood as adjustments or transformations of the input matrix. These operators are not the final signals, but rather intermediate results. Operators in this category have the prefix 'ADJ'.
* The third category corresponds to the output of binary (0/1) matrices, which can be understood as filtering out certain intervals. These operators are not the final signals but also intermediate results. Operators in this category have the prefix 'FILTER'.
* The fourth category consists of only one operator, $ADJ\_creategroup$, which creates stock groups based on the input.

2.2 Overview On Extracted Features

2.2.1 $FILTER\_nsigma$

$std :: vector < std :: vector < double >> FILTER\_nsigma(conststd :: vector < std :: vector < double >> \&arg1, doublecutoff);$
Filter out intervals that are less than cutoff * std.

2.2.2 $FILTER\_rsigma$

$std :: vector < std :: vector < double >> FILTER\_rsigma(conststd :: vector < std :: vector < double >> \&arg1, doublecutoff);$
Filter out intervals that are greater than cutoff * std.

2.2.3 $FILTER\_peak$

$std :: vector < std :: vector < double >> FILTER\_peak(conststd :: vector < std :: vector < double >> \&arg1);$
Filter out peak intervals

2.2.4 $FILTER\_up$

$std :: vector < std :: vector < double >> FILTER\_up(conststd :: vector < std :: vector < double >> \&arg1);$
Filter out intervals greater than the previous sample

2.2.5 $FILTER\_down$

$std :: vector < std :: vector < double >> FILTER\_down(conststd :: vector < std :: vector < double >> \&arg1);$
Filter out intervals smaller than the previous sample.

2.2.6 $FINAL\_raw$

$std :: vector < std :: vector < double >> FINAL\_raw(conststd :: vector < std :: vector < double >> \&arg1);$
arg1 is the input dataframe, extract the last row as a feature.

2.2.7 $FINAL\_rank$

$std :: vector < std :: vector < double >> FINAL\_rank(conststd :: vector < std :: vector < double >> \&arg1);$
arg1 is the input dataframe, extract the last row and use its quantiles as features.

2.2.8 $FINAL\_max$

$std :: vector < std :: vector < double >> FINAL\_max(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, extract the maximum value of the last bd rows as a feature

2.2.9 $FINAL\_min$

$std :: vector < std :: vector < double >> FINAL\_min(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, extract the minimum value of the last bd rows as a feature.

2.2.10 $FINAL\_mean$

$std :: vector < std :: vector < double >> FINAL\_mean(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, extract the mean value of the last bd rows as a feature.

2.2.11 $FINAL\_std$

$std :: vector < std :: vector < double >> FINAL\_std(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, extract the standard deviation of the last bd rows as a feature.

2.2.12 $FINAL\_skew$

$std :: vector < std :: vector < double >> FINAL\_skew(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
”arg1 is the input dataframe, extract the skewness of the last bd rows as a feature.”

2.2.13 $FINAL\_kurt$

$std :: vector < std :: vector < double >> FINAL\_kurt(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, extract the kurtosis of the last bd rows as a feature.

2.2.14 $FINAL\_delta$

$std :: vector < std :: vector < double >> FINAL\_delta(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, subtract the last row from the row that is bd rows before the last row and use it as a feature.

2.2.15 $FINAL\_correlation$

$std :: vector < std :: vector < double >> FINAL\_correlation(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 and arg2 are input dataframes, calculate the correlation between them and use it as a feature.

2.2.16 $FINAL\_covariance$

$std :: vector < std :: vector < double >> FINAL\_covariance(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 and arg2 are input dataframes, calculate the covariance between them and use it as a feature.

2.2.17 $FINAL\_corr$

$std :: vector < std :: vector < double >> FINAL\_corr(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

arg1 is an input dataframe. Calculate the autocorrelation coefficient.

2.2.18 $FINAL\_cov$

$std :: vector < std :: vector < double >> FINAL\_cov(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

arg1 is an input dataframe. Calculate the autocovariance coefficient.

2.2.19 $FINAL\_weighted$

$std :: vector < std :: vector < double >> FINAL\_weighted(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

2.2.20 $FINAL\_slope$

$std :: vector < std :: vector < double >> FINAL\_slope(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

perform linear regression on the last 'bd' points of the input dataframe, where 'y' is the values of the dataframe and 'x' is [0, 1, 2, ..., bd-1]. The slope of the linear regression will be calculated and used as a feature.

2.2.21 $FINAL\_intercept$

$std :: vector < std :: vector < double >> FINAL\_intercept(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

'arg1 is the input dataframe, y=arg1 last bd points, x=[0,1,2,...,bd-1], calculate linear regression with the intercept term as the feature'

2.2.22 $FINAL\_residual$

$std :: vector < std :: vector < double >> FINAL\_residual(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

arg1 is the input dataframe, y=arg1 last bd points, x=[0,1,2,...,bd-1], calculate linear regression, with the residual as the feature.

2.2.23 $FINAL\_residual2$

$std :: vector < std :: vector < double >> FINAL\_residual2(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

arg1 is the input dataframe, y=arg1 last bd points, x=[0,1,2,...,bd-1], calculate linear regression, with the relative error as the feature

2.2.24 $FINAL\_ar1a$

$std :: vector < std :: vector < double >> FINAL\_ar1a(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the coefficients of the AR(1) model as the feature

2.2.25 $FINAL\_ar1b$

$std :: vector < std :: vector < double >> FINAL\_ar1b(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the intercept term of the AR(1) model as the feature.

2.2.26 $FINAL\_ar1e$

$std :: vector < std :: vector < double >> FINAL\_ar1e(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the residuals of the AR(1) model as the feature.

2.2.27 $FINAL\_ar2e$

$std :: vector < std :: vector < double >> FINAL\_ar2e(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the relative error of the AR(1) model as the feature.

2.2.28 $FINAL\_gar1e$

$std :: vector < std :: vector < double >> FINAL\_gar1e(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp, intgr);$

2.2.29 $FINAL\_gar2e$

$std :: vector < std :: vector < double >> FINAL\_gar2e(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp, intgr);$

2.2.30 $FINAL\_ar1e2$

$std :: vector < std :: vector < double >> FINAL\_ar1e2(conststd :: vector < std :: vector < double >> \&arg1, intbd);$

2.2.31 $FINAL\_ar2e2$

$std :: vector < std :: vector < double >> FINAL\_ar2e2(conststd :: vector < std :: vector < double >> \&arg1, intkp);$

2.2.32 $FINAL\_arima1a$

$std :: vector < std :: vector < double >> FINAL\_arima1a(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the coefficients of the ARIMA(1) model as the feature

2.2.33 $FINAL\_arima1b$

$std :: vector < std :: vector < double >> FINAL\_arima1b(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the intercept term of the ARIMA(1) model as the feature.

2.2.34 $FINAL\_arima1e$

$std :: vector < std :: vector < double >> FINAL\_arima1e(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the intercept term of the ARIMA(1) model as the feature.

2.2.35 $FINAL\_arima1e2$

$std :: vector < std :: vector < double >> FINAL\_arima1e2(conststd :: vector < std :: vector < double >> \&arg1, intbd);$
arg1 is the input dataframe, use the relative error of the ARIMA(1) model as the feature.

2.2.36 $FINAL\_fmax$

$std :: vector < std :: vector < double >> FINAL\_fmax(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 and arg2 are input dataframes. Retrieve the intervals in arg2 where the label is 1, and calculate the maximum value of arg1 within those intervals.

2.2.37 $FINAL\_fmin$

$std :: vector < std :: vector < double >> FINAL\_fmin(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 and arg2 are input dataframes. Retrieve the intervals in arg2 where the label is 1, and calculate the minimum value of arg1 within those intervals

2.2.38 $FINAL\_fmean$

$std :: vector < std :: vector < double >> FINAL\_fmean(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the mean of arg1 within that interval.

2.2.39 $FINAL\_fstd$

$std :: vector < std :: vector < double >> FINAL\_fstd(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the standard deviation of arg1 within that interval.

2.2.40 $FINAL\_fskew$

$std :: vector < std :: vector < double >> FINAL\_fskew(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the skewness of arg1 within that interval.

2.2.41 $FINAL\_fkurt$

$std :: vector < std :: vector < double >> FINAL\_fkurt(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the kurtosis of arg1 within that interval.

2.2.42 $FINAL\_fdmax$

$std :: vector < std :: vector < double >> FINAL\_fdmax(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 and arg2 are input dataframes. Retrieve the interval where arg2 is marked as 1, calculate the difference between the maximum value of arg1 within that interval and the maximum value of arg1 overall.

2.2.43 $FINAL\_fdmin$

$std :: vector < std :: vector < double >> FINAL\_fdmin(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 and arg2 are input dataframes. Retrieve the interval where arg2 is marked as 1, calculate the difference between the minimum value of arg1 within that interval and the minimum value of arg1 overall.

2.2.44 $FINAL\_fdmean$

$std :: vector < std :: vector < double >> FINAL\_fdmean(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 and arg2 are input dataframes. Retrieve the interval where arg2 is marked as 1, calculate the difference between the mean value of arg1 within that interval and the mean value of arg1 overall.

2.2.45 $FINAL\_fddmean$

$std :: vector < std :: vector < double >> FINAL\_fddmean(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 and arg2 are input dataframes. Retrieve the interval where arg2 is marked as 1, calculate the ratio between the mean value of arg1 within that interval and the mean value of arg1 overall.

2.2.46 $FINAL\_fdstd$

$std :: vector < std :: vector < double >> FINAL\_fdstd(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the difference between the standard deviation of arg1 within that interval and the overall standard deviation of arg1.

2.2.47 $FINAL\_fddstd$

$std :: vector < std :: vector < double >> FINAL\_fddstd(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 arg2 are input dataframes. Retrieve the interval where arg2 is marked as 1, and calculate the standard deviation of arg1 within that interval compared to the overall standard deviation of arg1.

2.2.48 $FINAL\_fdskew$

$std :: vector < std :: vector < double >> FINAL\_fdskew(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the difference between the skewness of arg1 within that interval and the skewness of arg1 overall.

2.2.49 $FINAL\_fdkurt$

$std :: vector < std :: vector < double >> FINAL\_fdkurt(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$

arg1 arg2 are input dataframes. Retrieve the interval marked as 1 in arg2 and calculate the difference between the kurtosis of arg1 within that interval and the kurtosis of arg1 overall.

2.2.50 $FINAL\_argmax$

$std :: vector < std :: vector < double >> FINAL\_argmax(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is an input dataframe. Calculate the position where the maximum value occurs within the last kp points

2.2.51 $FINAL\_argmin$

$std :: vector < std :: vector < double >> FINAL\_argmin(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is an input dataframe. Calculate the position where the minimum value occurs within the last kp points

2.2.52 $FINAL\_chebyshev$

$std :: vector < std :: vector < double >> FINAL\_chebyshev(conststd :: vector < std :: vector < double >> \&arg1, intn, intkp);$
arg1 is the input dataframe, projecting onto the Chebyshev polynomial of order 'n' as a feature

2.2.53 $FINAL\_chebyshev2$

$std :: vector < std :: vector < double >> FINAL\_chebyshev2(conststd :: vector < std :: vector < double >> \&arg1, intn, intkp);$
arg1 is the input dataframe, projecting onto the Chebyshev2 polynomial of order 'n' as a feature

2.2.54 $FINAL\_hermite$

$std :: vector < std :: vector < double >> FINAL\_hermite(conststd :: vector < std :: vector < double >> \&arg1, intn, intkp);$
arg1 is the input dataframe, projecting onto the Hermite polynomial of order 'n' as a feature

2.2.55 $FINAL\_laguerre$

$std :: vector < std :: vector < double >> FINAL\_laguerre(conststd :: vector < std :: vector < double >> \&arg1, intn, intkp);$
arg1 is the input dataframe, projecting onto the Laguerre polynomial of order 'n' as a feature

2.2.56 $FINAL\_cided$

$std :: vector < std :: vector < double >> FINAL\_cided(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, where the average change in value between each point and the previous point is used as a feature.(std)

2.2.57 $FINAL\_cidednormalized$

$std :: vector < std :: vector < double >> FINAL\_cidednormalized(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, where the average change in value between each point and the previous point is used as a feature(normalized).

2.2.58 $FINAL\_centroid$

$std :: vector < std :: vector < double >> FINAL\_centroid(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, use the centroid of the input as a feature.

2.2.59 $FINAL\_meanabsdiff$

$std :: vector < std :: vector < double >> FINAL\_meanabsdiff(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, where the average change in value between each point and the previous point is used as a feature.(abs difference)

2.2.60 $FINAL\_countabovemean$

$std :: vector < std :: vector < double >> FINAL\_countabovemean(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, the number of samples above the mean as a feature.

2.2.61 $FINAL\_countbelowmean$

$std :: vector < std :: vector < double >> FINAL\_countbelowmean(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, the number of samples below the mean as a feature.

2.2.62 $FINAL\_fftresidual$

$std :: vector < std :: vector < double >> FINAL\_fftresidual(conststd :: vector < std :: vector < double >> \&arg1, intbd, intkp);$
arg1 is the input dataframe, using FFT to remove the first KP low-frequency components.

2.2.63 $FINAL\_fftresidualh$

$std :: vector < std :: vector < double >> FINAL\_fftresidualh(conststd :: vector < std :: vector < double >> \&arg1, intbd, intkp);$
arg1 is the input dataframe, using FFT to remove the first KP high-frequency components.

2.2.64 $FINAL\_walshresidual$

$std :: vector < std :: vector < double >> FINAL\_walshresidual(conststd :: vector < std :: vector < double >> \&arg1, intbd, intkp);$
arg1 is the input dataframe, using walsh transformation to remove the first KP components.

2.2.65 ADJ_divstd

$std :: vector < std :: vector < double >> ADJ\_divstd(conststd :: vector < std :: vector < double >> \&arg1);$
arg1 is the input dataframe, adjusting the standard deviation of each row.

2.2.66 ADJ_demean

$std :: vector < std :: vector < double >> ADJ\_demean(conststd :: vector < std :: vector < double >> \&arg1);$
arg1 is the input dataframe, adjusting the mean of each row.

2.2.67 ADJ_rank

$std :: vector < std :: vector < double >> ADJ\_rank(conststd :: vector < std :: vector < double >> \&arg1);$
arg1 is an input dataframe, with each column mapped to its quantiles.

2.2.68 ADJ_svd

$std :: vector < std :: vector < double >> ADJ\_svd(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is an input dataframe. Utilizing SVD, it undergoes singular value adjustment using method 1

2.2.69 $ADJ\_svdflat$

$std :: vector < std :: vector < double >> ADJ\_svdflat(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is an input dataframe. Utilizing SVD, it undergoes singular value adjustment using method 2

2.2.70 $ADJ\_svdenergy$

$std :: vector < std :: vector < double >> ADJ\_svdenergy(conststd :: vector < std :: vector < double >> \&arg1, doublepercent);$
arg1 is an input dataframe. Utilizing SVD, it undergoes singular value adjustment using method 3

2.2.71 ADJ_svdaux

$std :: vector < std :: vector < double >> ADJ\_svdaux(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp);$
arg1 and arg2 are input dataframes. The SVD of arg2 is used to adjust arg1

2.2.72 $ADJ\_fftresidual$

$std :: vector < std :: vector < double >> ADJ\_fftresidual(conststd :: vector < std :: vector < double >> \&arg1, intbd, intkp);$
arg1 is the input dataframe, remove the low-frequency components of the Fourier transform(rolling version)

2.2.73 ADJ_fft

$std :: vector < std :: vector < double >> ADJ\_fft(conststd :: vector < std :: vector < double >> \&arg1, intkp);$
arg1 is the input dataframe, remove the low-frequency components of the Fourier transform.

2.2.74 $ADJ\_fftflatten$

$std :: vector < std :: vector < double >> ADJ\_fftflatten(conststd :: vector < std :: vector < double >> \&arg1);$
arg1 is an input dataframe. Adjustments are made to its power spectral density

2.2.75 *ADJ_dwt*

$std :: vector < std :: vector < double >> ADJ\_dwt(conststd :: vector < std :: vector < double >> \&arg1, intkp)$;
arg1 is the input dataframe, denoise using wavelet transform

2.2.76 *ADJ_dsmooth*

$std :: vector < std :: vector < double >> ADJ\_dsmooth(conststd :: vector < std :: vector < double >> \&arg1, intkp, doublecut)$;
arg1 is the input dataframe, wavelet transform low-frequency components

2.2.77 *ADJ_drough*

$std :: vector < std :: vector < double >> ADJ\_drough(conststd :: vector < std :: vector < double >> \&arg1, intkp, doublecut)$;
arg1 is the input dataframe, wavelet transform high-frequency components

2.2.78 *ADJ_slopey*

$std :: vector < std :: vector < double >> ADJ\_slopey(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intbd)$;
arg1 and arg2 are input dataframes, rolling beta coefficient of arg1 with respect to arg2 as the output matrix

2.2.79 *ADJ_intercepty*

$std :: vector < std :: vector < double >> ADJ\_intercepty(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intbd)$;
arg1 and arg2 are input dataframes, rolling intercept term of the regression model of arg1 with respect to arg2 as the output matrix

2.2.80 *ADJ_residualy*

$std :: vector < std :: vector < double >> ADJ\_residualy(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intbd)$;
arg1 and arg2 are input dataframes. Rolling regression model residuals of arg1 with respect to arg2 are output as a matrix.

2.2.81 *ADJ_residualr2y*

$std :: vector < std :: vector < double >> ADJ\_residualr2y(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intbd)$;
arg1 and arg2 are input dataframes. Rolling regression model of arg1 with respect to arg2 at the bd point, relative error terms, are output as a matrix

2.2.82 *ADJ_group*

$std :: vector < std :: vector < double >> ADJ\_group(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2)$;
arg1 and arg2 are input dataframes. Neutralizing arg1 using stock grouping specified by arg2

2.2.83 $ADJ\_creategroup$

$std :: vector < std :: vector < double >> ADJ\_creategroup(conststd :: vector < std :: vector < double >> \&arg1, intbd, intdiv);$
arg1 is an input dataframe. Creating stock groups using the last bd rows of data

2.2.84 ADJ_image1

$std :: vector < std :: vector < double >> ADJ\_image1(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intdiv);$
arg1 and arg2 are input dataframes. Using arg2, arg1 is arranged into an image of size div * div and then undergoes Gaussian filtering.(ADJ_image1, ADJ_image2, ADJ_image3 parameters are different)

2.2.85 ADJ_image2

$std :: vector < std :: vector < double >> ADJ\_image2(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intdiv);$
arg1 and arg2 are input dataframes. Using arg2, arg1 is arranged into an image of size div * div and then undergoes Gaussian filtering.

2.2.86 ADJ_image3

$std :: vector < std :: vector < double >> ADJ\_image3(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intdiv);$
arg1 and arg2 are input dataframes. Using arg2, arg1 is arranged into an image of size div * div and then undergoes Gaussian filtering.

2.2.87 $ADJ\_residualall$

$std :: vector < std :: vector < double >> ADJ\_residualall(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 and arg2 are input dataframes. For each row of arg1, regression is performed on arg2 to obtain the residuals, which are then outputted.

2.2.88 $ADJ\_residualqall$

$std :: vector < std :: vector < double >> ADJ\_residualqall(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2);$
arg1 and arg2 are input dataframes. For each row of arg1, quadratic regression is performed on arg2 to obtain the residuals, which are then outputted.

2.2.89 $ADJ\_sortcurve$

$std :: vector < std :: vector < double >> ADJ\_sortcurve(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp);$
arg1 and arg2 are the input dataframes. By sorting arg1 row by row using arg2 and removing the rolling mean, the adjusted result is returned as output.(method 1)

2.2.90 $ADJ\_sortcurves$

$std :: vector < std :: vector < double >> ADJ\_sortcurves(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp)$;
arg1 and arg2 are the input dataframes. By sorting arg1 row by row using arg2 and removing the rolling mean, the adjusted result is returned as output.(method 2)

2.2.91 $ADJ\_sortfft$

$std :: vector < std :: vector < double >> ADJ\_sortfft(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp)$;
arg1 and arg2 are input dataframes. By sorting arg1 row by row using arg2 and adjusting its Fourier Transform's low-frequency component, the adjusted result is returned as output(method1)

2.2.92 $ADJ\_sortffts$

$std :: vector < std :: vector < double >> ADJ\_sortffts(conststd :: vector < std :: vector < double >> \&arg1, conststd :: vector < std :: vector < double >> \&arg2, intkp)$;
arg1 and arg2 are input dataframes. By sorting arg1 row by row using arg2 and adjusting its Fourier Transform's low-frequency component, the adjusted result is returned as output(method2)

3 Contact Us

3.1 Contact Us

If you find this tool useful for your research, you can contact us at 793063983@qq.com to obtain a standalone .so file for your production (subject to a certain fee). Alternatively, if you have any questions regarding its usage, feel free to reach out to us.