Statistical Arbitrage C++ reference file

Volter Entoma

November 2024

Overview

This codebase is a collection of utilities to analyze stock data, including price retrieval, statistical calculations, data normalization, and hypothesis testing. The main components are organized into the following classes:

1. **Stock** - Holds historical data for a stock.
2. **StockUtils** - Provides helper functions to retrieve specific price data.
3. **StockAnalysis** - Contains methods for statistical analysis and comparison.
4. **StandardNormalDistribution** - Implements standard normal distribution functions for hypothesis testing.
5. **FileReader** - Handles file operations for reading stock data and stock listings.

Class Summaries

1. Stock

Holds and manages historical data for a single stock, with methods to store, retrieve, and print data.

- **HistoricalData** Type: `std::unordered_map<std::string, DataMap>`
  - Maps dates to `DataMap`, where `DataMap` stores different price types (e.g., "close", "high").

Methods

- **addData(const std::string& date, const std::string& key, double value)**
  - Adds a data entry (price or volume) for a specific date.

- **getData(const std::string& date, const std::string& key) const**
  - Retrieves the value of a specific key (price or volume) on a given date.
- **printHistoricalData() const**
  - Prints all historical data in a formatted output for review.
- **getHistoricalData() const**
  - Returns the `HistoricalData` object for access by other classes.

2. StockUtils

A utility class with methods to extract price data from a `Stock` object for specified conditions.

Methods

- **getPriceData(const Stock& stock, const std::string& priceType)**
  - Retrieves all historical data for a specified price type (e.g., "close") from a `Stock` object.
- **getPriceDataInRange(const Stock& stock, const std::string& priceType, const std::string& startDate, const std::string& endDate)**
  - Retrieves data for a specific price type within a specified date range.

3. StockAnalysis

Provides statistical and analytical tools for processing stock data, including calculating differences, mean, standard deviation, normalization, and price-volume analysis.

Methods

- **calculateMean(const std::unordered_map¡std::string, double¿& data)**
  - Calculates the mean of the values in a given data dictionary.
- **calculateStandardDeviation(const std::unordered_map¡std::string, double¿& data)**
  - Calculates the standard deviation of the values in a data dictionary.
- **calculateStatistics(const std::unordered_map¡std::string, double¿& data)**

  - Returns both mean and standard deviation of data, using caching to store calculated values.

- **clearCache()**
  - Clears the cache used for storing mean and standard deviation of specific datasets.

- **calculateDifference(const Stock& stock1, const Stock& stock2, const std::string& priceType, const std::string& startDate, const std::string& endDate)**
  - Calculates the difference in specified price types between two stocks over the same date range, returning a dictionary of differences.

- **normalizeToEarliestDate(const std::unordered_map¡std::string, double¿& data)**
  - Normalizes values in a data dictionary relative to the earliest date value, useful for base-adjusted comparisons.

- **hasValidDataInRange(const Stock& stock1, const Stock& stock2, const std::string& priceType, const std::string& startDate, const std::string& endDate)**
  - Checks if both stocks contain valid, non-zero data for a specific price type in the specified date range.

- **calculatePriceVolume(const Stock& stock, const std::string& date, const std::string& priceType = "close")**
  - Computes the product of price and volume for a given date and price type.

- **calculateAveragePriceVolume(const Stock& stock, const std::string& startDate, const std::string& endDate, const std::string& priceType)**
  - Calculates the average price-volume for a stock over a specified date range.

- **sortMap(std::map¡std::string, double¿ givenMap)**
  - Sorts a map by its values, returning a sorted vector of pairs.

4. StandardNormalDistribution

Implements functions for the standard normal distribution, which are helpful for hypothesis testing.

Methods

- **pdf(const double& x)**
  - Returns the probability density function (PDF) for a given x.
- **cdf(const double& x)**
  - Returns the cumulative density function (CDF) for a given x.
- **inv_cdf(const double& quantile)**
  - Computes the inverse CDF (probit function) for a given quantile, used in hypothesis testing.

5. FileReader

Handles file reading operations, such as loading stock data into `Stock` objects and reading stock listings from a file.

Methods

- **loadStockDataFromFile(const std::string& filename, Stock& stock)**
  - Loads data from a file into a `Stock` object, parsing price and volume fields.
- **readNYSEListings(const std::string& filename)**
  - Reads stock listings from a file, returning a vector of stock names or symbols.

Key Processing Flows

Loading Stock Data

Use `FileReader::loadStockDataFromFile` to populate a `Stock` object with historical data from a CSV file, where each line corresponds to one day's trading data.

Retrieving Price Data

`StockUtils::getPriceData` or `getPriceDataInRange` retrieves specific price types (e.g., close, open) from a `Stock` object. If data needs filtering by a date range, use `getPriceDataInRange`.

Performing Analysis on Stock Data

- **Mean and Standard Deviation**: Use `StockAnalysis::calculateMean`, `calculateStandardDeviation`, or `calculateStatistics` to compute statistical summaries.
- **Normalization**: Use `StockAnalysis::normalizeToEarliestDate` to normalize prices relative to the earliest date, useful for comparing stocks.

Hypothesis Testing with StandardNormalDistribution

The `StandardNormalDistribution` class allows hypothesis testing on calculated means, where you can compute p-values using `cdf` for a null hypothesis.

Main.cpp Overview

This code performs an analysis of stock pairs based on their historical adjusted closing prices within a specified date range. It follows these main steps:

1. **Load NYSE Stock Listings**: Retrieves a list of stock names from a text file.
2. **Filter Stocks**: Filters out stocks that do not meet the criteria for valid dates or liquidity.
3. **Normalize and Compare Pairs**: Pairs stocks, normalizes them to a base date, computes differences, and calculates statistical measures (mean and standard deviation) for these differences.
4. **Statistical Testing**: Computes p-values for each pair's mean difference under the null hypothesis that the mean is zero.
5. **Output Results**: Outputs the results (mean, standard deviation, p-value) for each pair to a CSV file.

Detailed Steps and Equations

1. Load NYSE Stock Listings

```
std::vector<std::string> nyseListings = FileReader::readNYSEListings(NYSE_LISTINGS_FILE);
```

This reads a list of NYSE stock symbols from a file (`NYSE_LISTINGS_FILE`), storing each symbol in a vector `nyseListings`.

2. Filter Stocks Based on Criteria

Each stock is loaded, and two filters are applied:

- **Liquidity Check**: Stocks are filtered based on average price-volume, ensuring they exceed a threshold (`PRICE_VOLUME_THRESH`). This is calculated using:
$$\text{Average Price-Volume} = \frac{\sum_i \text{Price}_i \times \text{Volume}_i}{\text{Number of Days}}$$
- **Date Range Validation**: Stocks are compared against a reference stock (e.g., "SPY") to ensure they have valid data for all dates in the specified range.

3. Normalize and Compare Pairs

The core comparison is done pairwise for stocks. For each stock in `nyseListings`:

1. **Normalization**:

   - Each stock's adjusted close prices are normalized relative to the earliest available date in the specified range. This sets the price at the earliest date to 1.0, and all other prices are adjusted relative to this baseline.

$$\text{Normalized Price}_{\text{date}} = \frac{\text{Price}_{\text{date}}}{\text{Price}_{\text{earliest date}}}$$

2. **Calculate Differences**:

   - The normalized prices of a base stock and a comparison stock are subtracted to compute a daily difference vector:

$$\text{Difference}_{\text{date}} = \text{Normalized Price}_{\text{base stock, date}} - \text{Normalized Price}_{\text{comparison stock, date}}$$

3. **Statistical Measures**:

   - For each difference vector, the mean and standard deviation are calculated:
     - **Mean**:

$$\text{Mean} = \frac{1}{N}\sum_{i=1}^{N} \text{Difference}_i$$

     - **Standard Deviation**:

$$\text{Standard Deviation} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\text{Difference}_i - \text{Mean})^2}$$

4. **Store Minimum Standard Deviation Pair**:

   - Among comparison pairs, only the pair with the smallest standard deviation is kept, associating it with its mean and standard deviation in `stockPairMean` and `stockPairStandardDeviation`.

4. Hypothesis Testing (P-Value Calculation)

For each stock pair's mean difference and standard deviation, a p-value is calculated to test the null hypothesis that the mean difference is zero:

1. **Compute Test Statistic**:

- If the standard deviation is not zero, the test statistic is:

$$Z = \frac{|\text{Mean}|}{\text{Standard Deviation}}$$

2. **P-Value from CDF**:

   - The p-value is then calculated as the probability of observing a value as extreme as Z under the standard normal distribution:

     $$\text{P-Value} = 2 \times (1 - \Phi(Z))$$

     Here, $\Phi(Z)$ is the cumulative distribution function (CDF) of the standard normal distribution.

   - If the standard deviation is zero, the p-value is set to 1.0, as no variance in data means the null hypothesis cannot be rejected.

5. Output Results to File

The code writes the following values for each stock pair to `Pairs_Mean_Standard_deviation_P_value.txt`:

- **Pair Name** (e.g., "AAPL-MSFT")
- **Mean** of the price differences
- **Standard Deviation** of the price differences
- **P-Value** from the hypothesis test