FRE6883 - Financial Computing
Earnings Surprise for Russell 3000 ETF Stocks

Team 4

Juan Camilo Mennes, Harsh Kulkarni, Charles Wu, Joaquin Garay, Rhugved Bhojane

Chapters

- Summary
- Russell 3000 3Q24 Earnings Analysis
- Our classes/objects
- Program overview
- Output
- Contributions

NYU

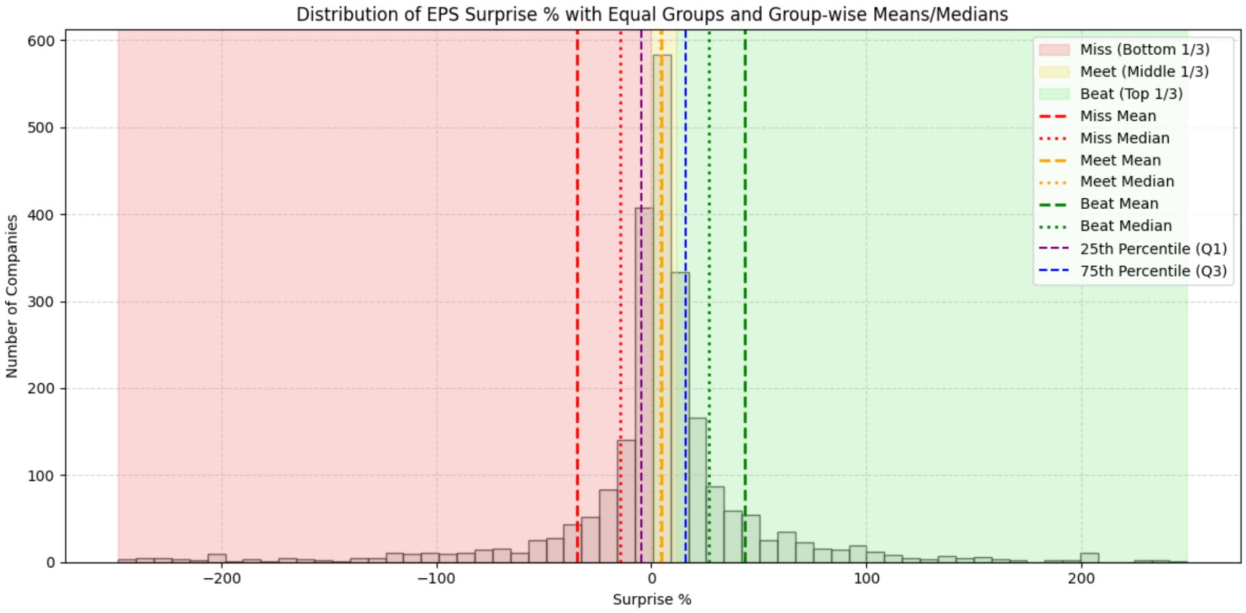
Executive Summary

- Program objective:
  - Retrieve price data of Russell 3000 ETF stocks around 4Q24 earnings announcements
  - Classify stocks based in 3 groups based on earnings surprise (Zacks)
  - Analyze group/stock excess returns vs IWV benchmark performance (user defined window around earnings release)
- Key Results:
  - On average, following 4Q24 earnings announcement:
    - Beat group outperforms index (up to 40 days after)
    - Meet/Miss groups underperform the index (larger effect for Miss)

Russell 3000 3Q24 Earnings Analysis

- **Stock universe:** 2552
- **How do we split them?** Earning Surprise (%)
  - Sort stocks from lowest to highest earning surprise
  - Equally sized groups by percentile:
    - *Miss* group: 0 to 33th percentile [min: -23700; median: -17; max: 0]
    - *Meet* group: 33th-66th percentile [min: 0; median: 4; max: 12]
    - *Beat* group: 66th-99th percentile [min: 12; median: 31; max: 14928]

Russell 3000 3Q24 Earnings Analysis

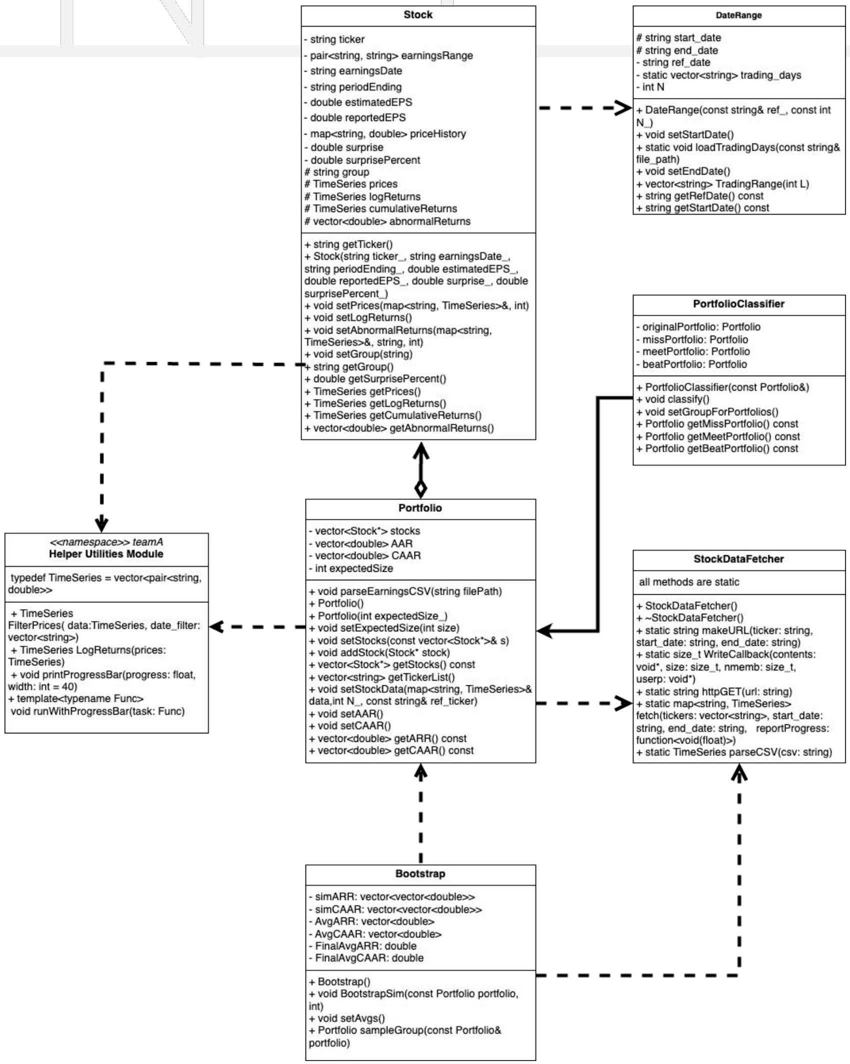


NYU

Description of Classes

1. **Portfolio**
   - Stores a group of Stock objects and helps compute AAR and CAAR
2. **Stock**
   - Stores stock info. from earnings, price/returns data, etc.
3. **StockFetcher**
   - Use libcurl to retrieve data from EODHD
4. **PortfolioClassifier**
   - Stores Portfolio objects and help split them in groups
5. **Bootstrapping**
   - Stores results from simulations on a Portfolio object

How our classes interact



```
Stock
- string ticker
- pair<string, string> earningsRange
- string earningsDate
- string periodEnding
- double estimatedEPS
- double reportedEPS
- map<string, double> priceHistory
- double surprise
- double surprisePercent
# string group
# TimeSeries prices
# TimeSeries logReturns
# TimeSeries cumulativeReturns
# vector<double> abnormalReturns
+ string getTicker()
+ Stock(string ticker_, string earningsDate_, string periodEnding_, double estimatedEPS_, double reportedEPS_, double surprise_, double surprisePercent_)
+ void setPrices(map<string, TimeSeries>&, int)
+ void setLogReturns()
+ void setAbnormalReturns(map<string, TimeSeries>&, string, int)
+ void setGroup(string)
+ string getGroup()
+ double getSurprisePercent()
+ TimeSeries getPrices()
+ TimeSeries getLogReturns()
+ TimeSeries getCumulativeReturns()
+ vector<double> getAbnormalReturns()
```

```
DateRange
# string start_date
# string end_date
- string ref_date
- static vector<string> trading_days
- int N
+ DateRange(const string& ref_, const int N_)
+ void setStartDate()
+ static void loadTradingDays(const string& file_path)
+ void setEndDate()
+ vector<string> TradingRange(int L)
+ string getRefDate() const
+ string getStartDate() const
```

```
PortfolioClassifier
- originalPortfolio: Portfolio
- missPortfolio: Portfolio
- meetPortfolio: Portfolio
- beatPortfolio: Portfolio
+ PortfolioClassifier(const Portfolio&)
+ void classify()
+ void setGroupForPortfolios()
+ Portfolio getMissPortfolio() const
+ Portfolio getMeetPortfolio() const
+ Portfolio getBeatPortfolio() const
```

```
<<namespace>> teamA
Helper Utilities Module
typedef TimeSeries = vector<pair<string, double>>
+ TimeSeries FilterPrices( data:TimeSeries, date_filter: vector<string>)
+ TimeSeries LogReturns(prices: TimeSeries)
+ void printProgressBar(progress: float, width: int = 40)
+ template<typename Func>
void runWithProgressBar(task: Func)
```

```
Portfolio
- vector<Stock*> stocks
- vector<double> AAR
- vector<double> CAAR
- int expectedSize
+ void parseEarningsCSV(string filePath)
+ Portfolio()
+ Portfolio(int expectedSize_)
+ void setExpectedSize(int size)
+ void setStocks(const vector<Stock*>& s)
+ void addStock(Stock* stock)
+ vector<Stock*> getStocks() const
+ vector<string> getTickerList()
+ void setStockData(map<string, TimeSeries>& data,int N_, const string& ref_ticker)
+ void setAAR()
+ void setCAAR()
+ vector<double> getARR() const
+ vector<double> getCAAR() const
```

```
StockDataFetcher
all methods are static
+ StockDataFetcher()
+ ~StockDataFetcher()
+ static string makeURL(ticker: string, start_date: string, end_date: string)
+ static size_t WriteCallback(contents: void*, size: size_t, nmemb: size_t, userp: void*)
+ static string httpGET(url: string)
+ static map<string, TimeSeries> fetch(tickers: vector<string>, start_date: string, end_date: string,   reportProgress: function<void(float)>)
+ static TimeSeries parseCSV(csv: string)
```

```
Bootstrap
- simARR: vector<vector<double>>
- simCAAR: vector<vector<double>>
- AvgARR: vector<double>
- AvgCAAR: vector<double>
- FinalAvgARR: double
- FinalAvgCAAR: double
+ Bootstrap()
+ void BootstrapSim(const Portfolio portfolio, int)
+ void setAvgs()
+ Portfolio sampleGroup(const Portfolio& portfolio)
```

NYU

Data Structures

1. **Data from API:** map<string,TimeSeries>
    - Map key = ticker. We define TimeSeries as vector<pair<string, double>> i.e. {“2025-05-13”, 24.672}

2. **Stock info:** string, double for earnings data and TimeSeries for prices/returns

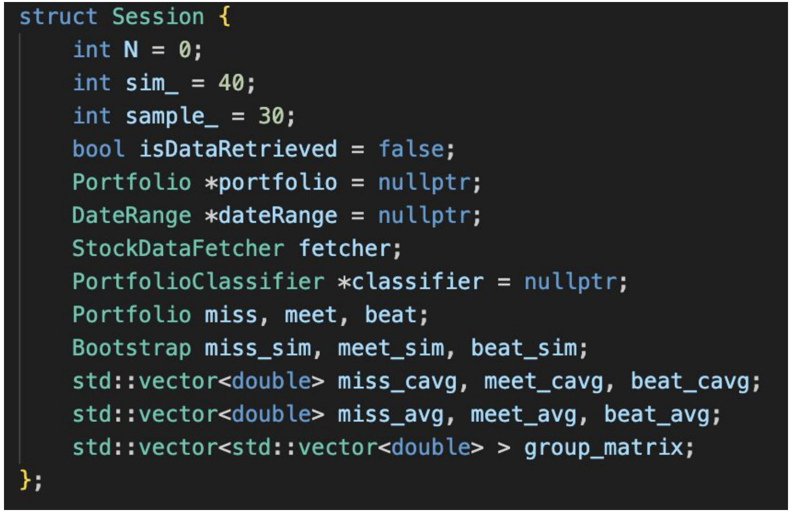
3. **Portfolio:** vector<Stocks*> and vector<double> for AAR-CAAR

4. **Bootstrapping:** vector<vector<double>> matrix for simulation results, vector<double> for averages of simulations and double for full averages

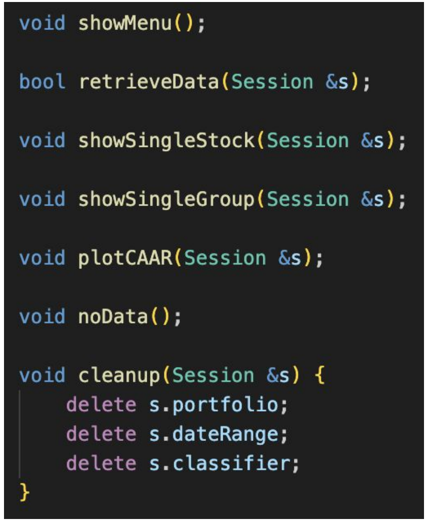
5. **AAR-CAAR results:** vector<vector<double>> matrix

Behind the menu & int main()

To store our objects and data, and scroll in the menu without them going out of scope, we use

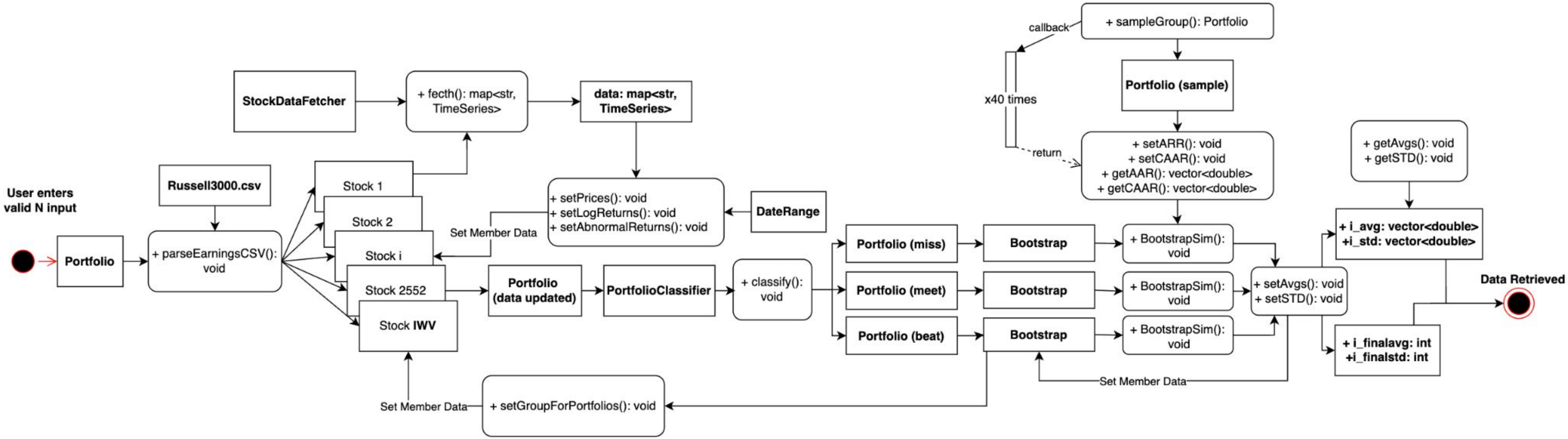


```
struct Session {
    int N = 0;
    int sim_ = 40;
    int sample_ = 30;
    bool isDataRetrieved = false;
    Portfolio *portfolio = nullptr;
    DateRange *dateRange = nullptr;
    StockDataFetcher fetcher;
    PortfolioClassifier *classifier = nullptr;
    Portfolio miss, meet, beat;
    Bootstrap miss_sim, meet_sim, beat_sim;
    std::vector<double> miss_cavg, meet_cavg, beat_cavg;
    std::vector<double> miss_avg, meet_avg, beat_avg;
    std::vector<std::vector<double> > group_matrix;
};
```

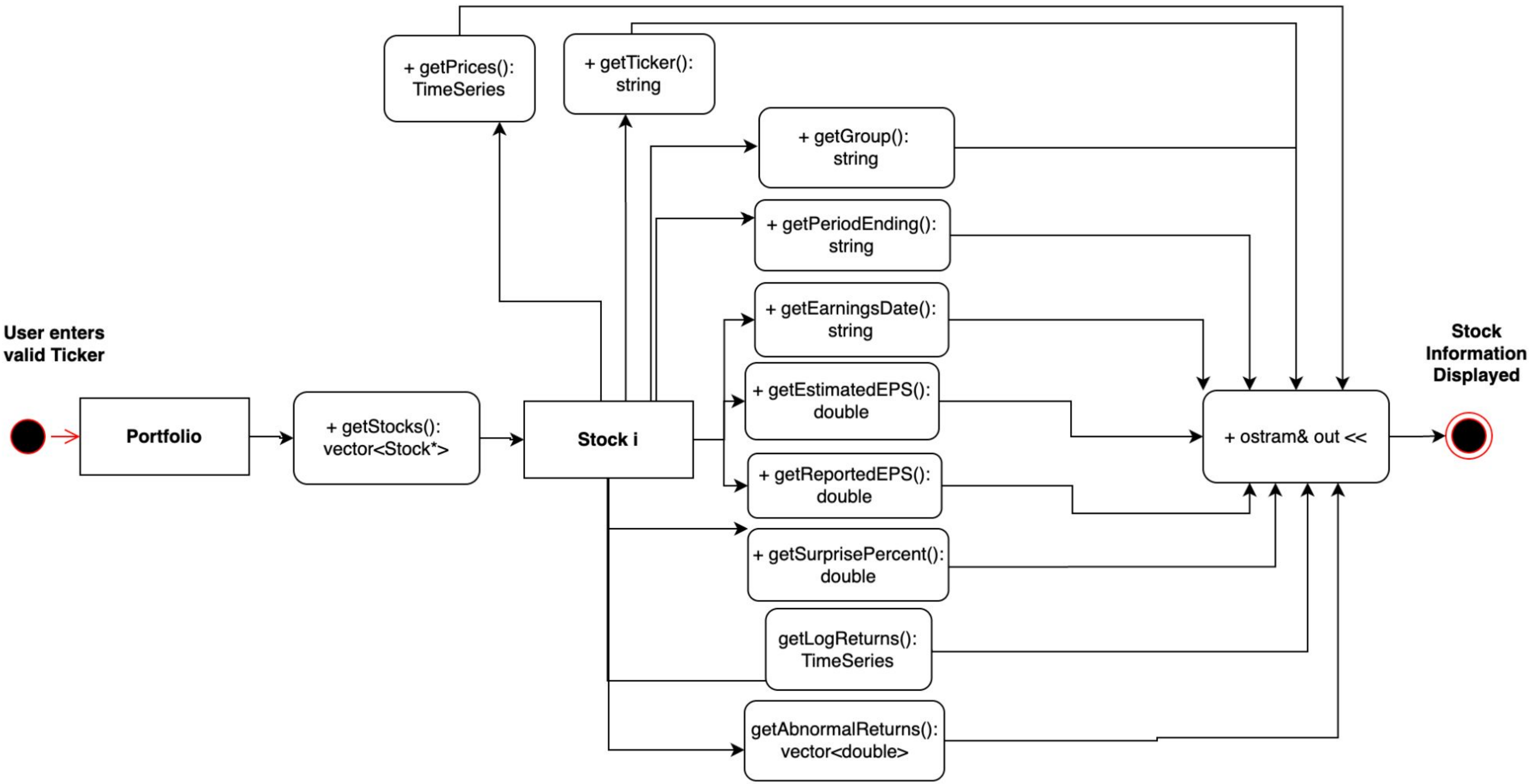


```
void showMenu();

bool retrieveData(Session &s);

void showSingleStock(Session &s);

void showSingleGroup(Session &s);

void plotCAAR(Session &s);

void noData();

void cleanup(Session &s) {
    delete s.portfolio;
    delete s.dateRange;
    delete s.classifier;
}
```

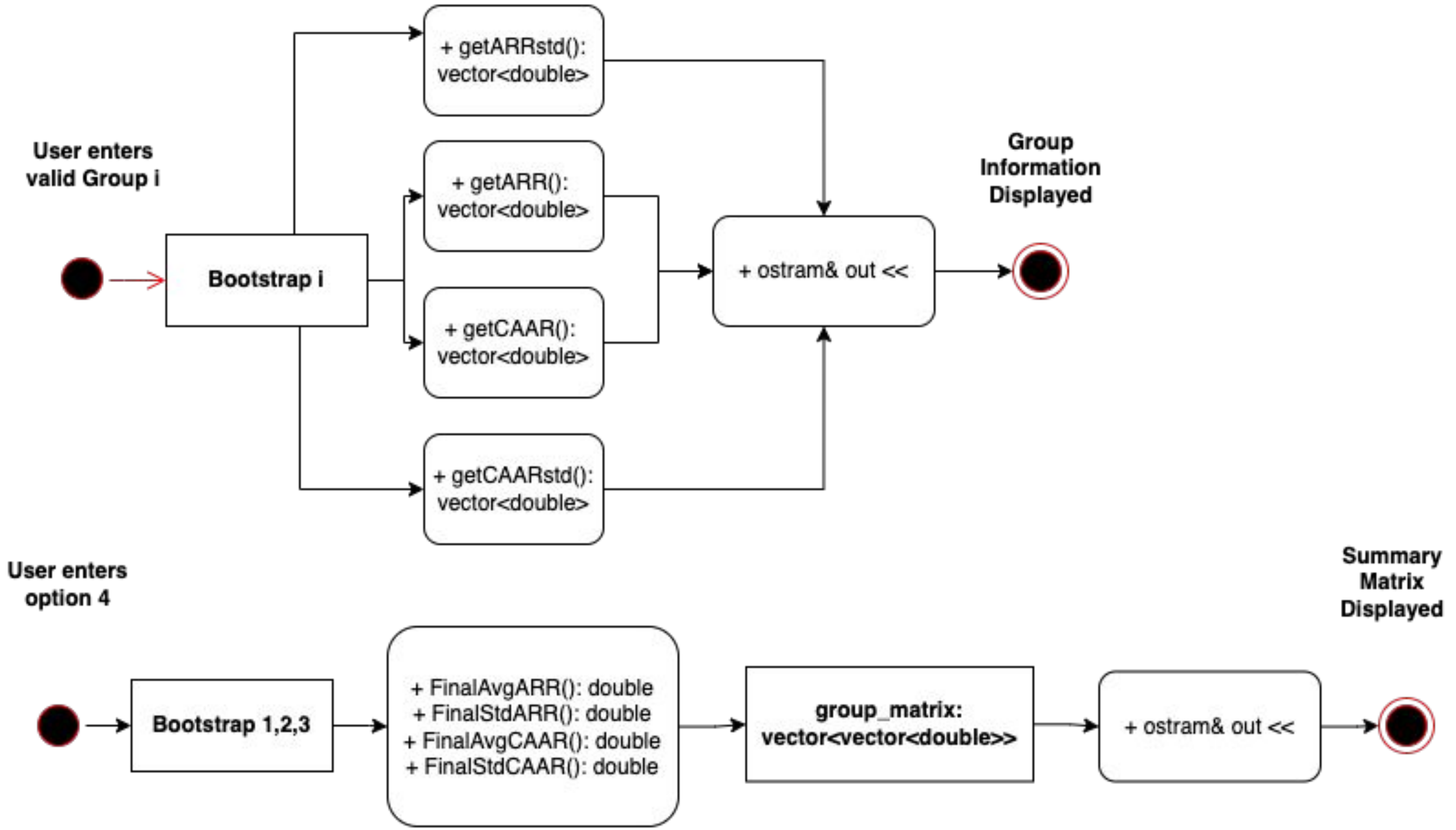
Option 1: Retrieving data



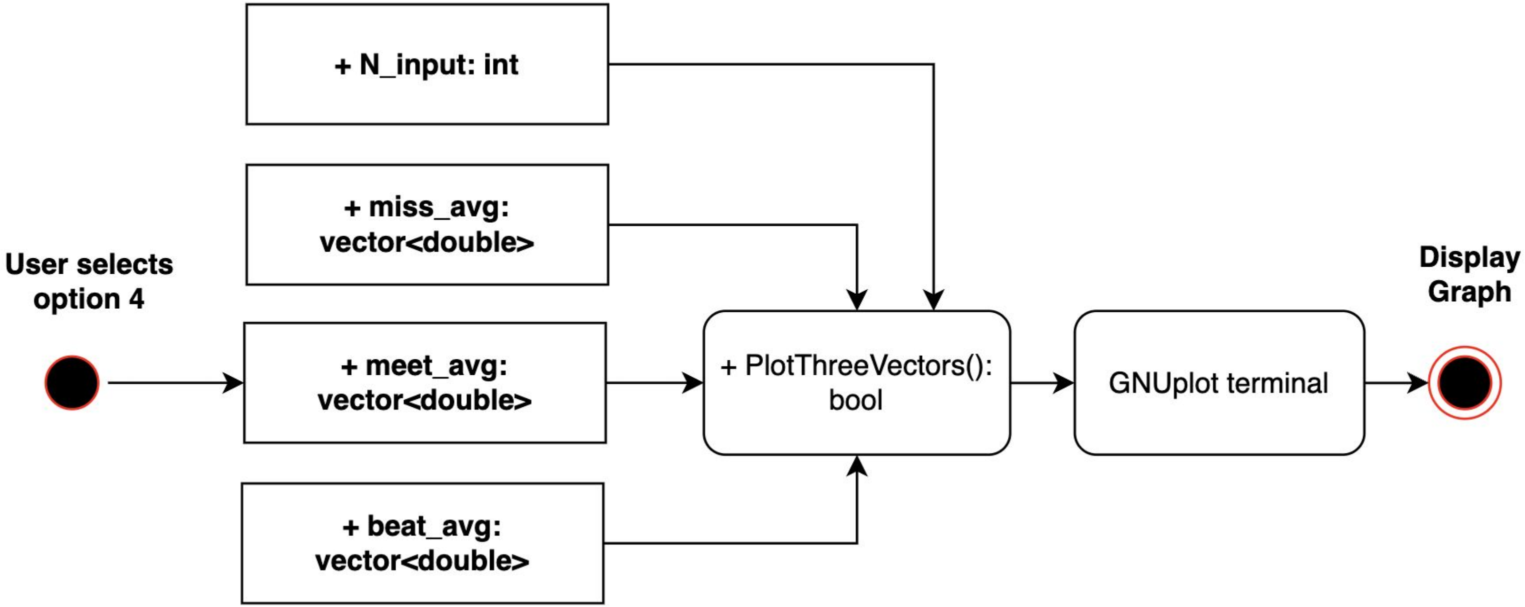
Option 2: Stock Info



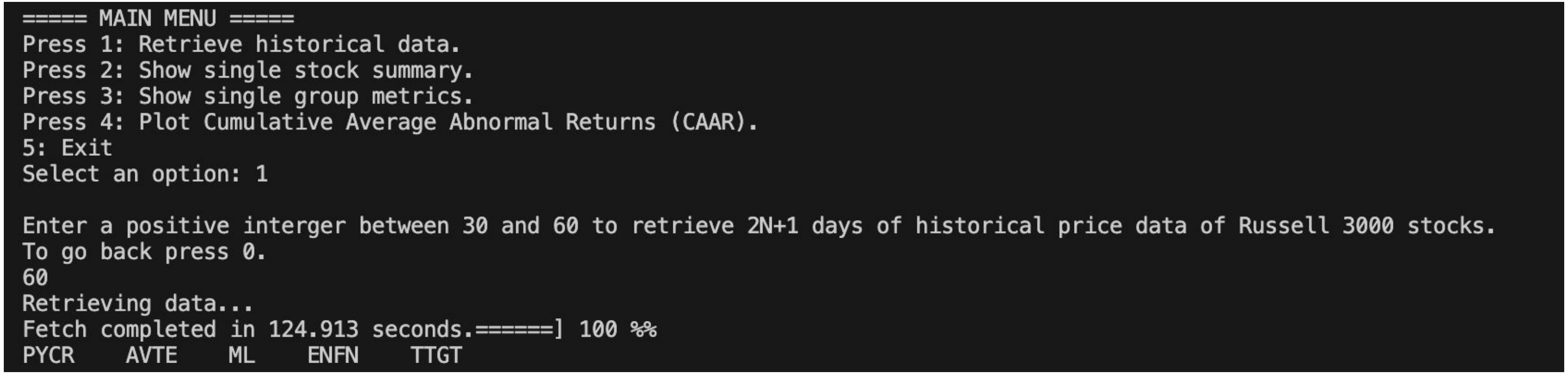
Option 3: Group Info


User enters valid Group i
Bootstrap i
+ getARRstd(): vector<double>
+ getARR(): vector<double>
+ getCAAR(): vector<double>
+ getCAARstd(): vector<double>
+ ostram& out <<
Group Information Displayed
User enters option 4
Bootstrap 1,2,3
+ FinalAvgARR(): double
+ FinalStdARR(): double
+ FinalAvgCAAR(): double
+ FinalStdCAAR(): double
group_matrix: vector<vector<double>>
+ ostram& out <<
Summary Matrix Displayed

Option 4: Display Plot

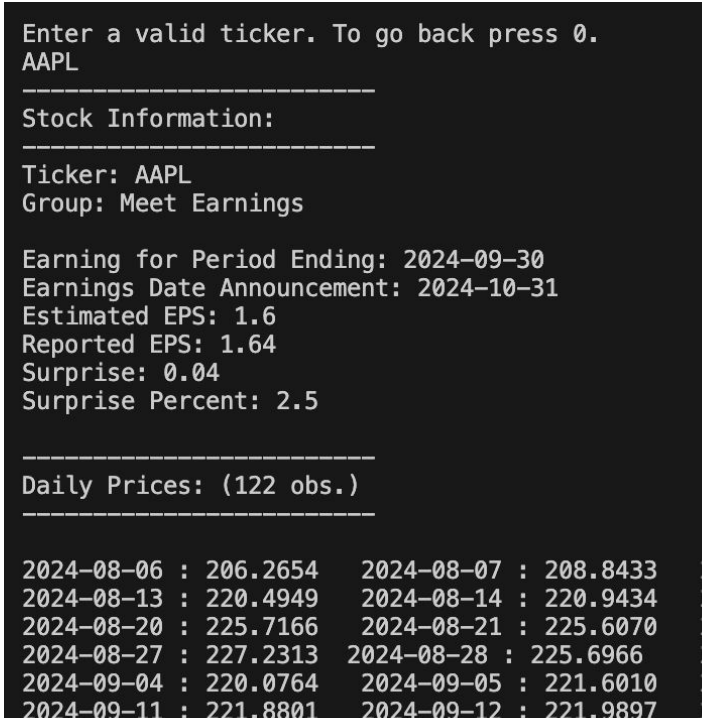

+ N_input: int
+ miss_avg: vector<double>
User selects option 4
+ meet_avg: vector<double>
+ beat_avg: vector<double>
+ PlotThreeVectors(): bool
GNUplot terminal
Display Graph

Outputs: 1. Retrieving Data

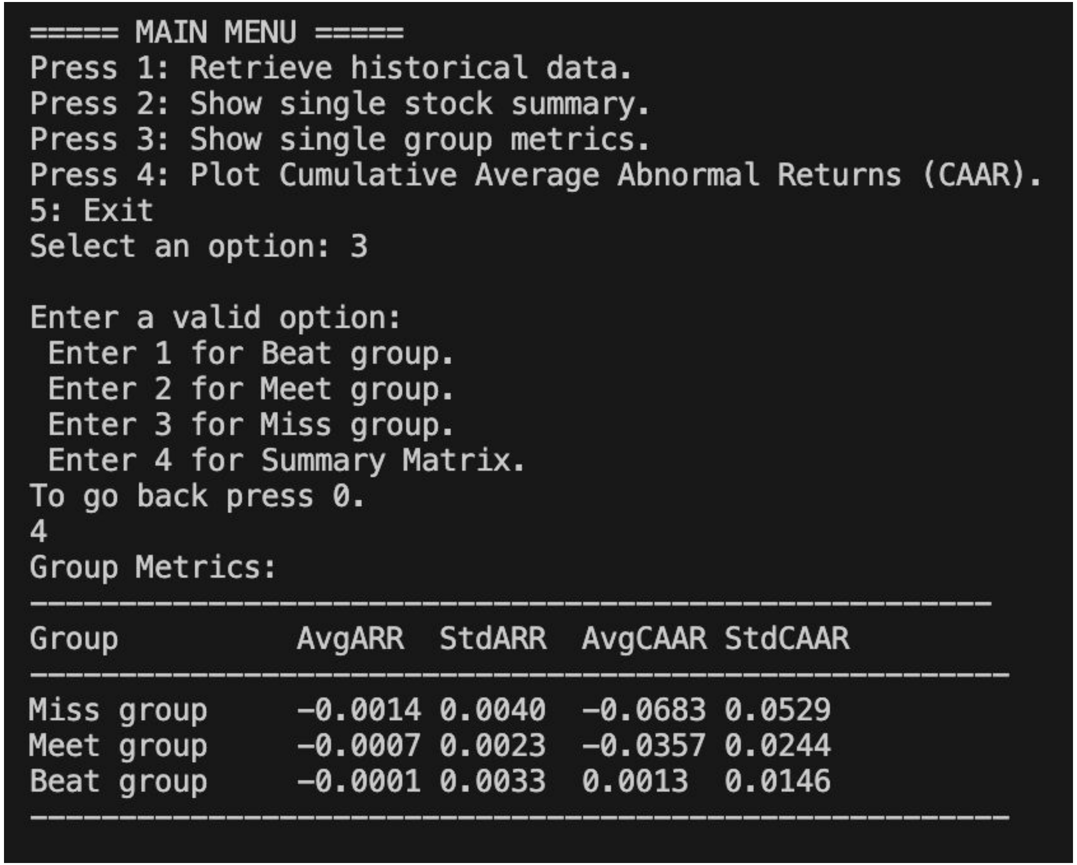


```
===== MAIN MENU =====
Press 1: Retrieve historical data.
Press 2: Show single stock summary.
Press 3: Show single group metrics.
Press 4: Plot Cumulative Average Abnormal Returns (CAAR).
5: Exit
Select an option: 1

Enter a positive interger between 30 and 60 to retrieve 2N+1 days of historical price data of Russell 3000 stocks.
To go back press 0.
60
Retrieving data...
Fetch completed in 124.913 seconds.======] 100 %%
PYCR    AVTE    ML    ENFN    TTGT
```

Outputs: 2. Data for a Stock: AAPL

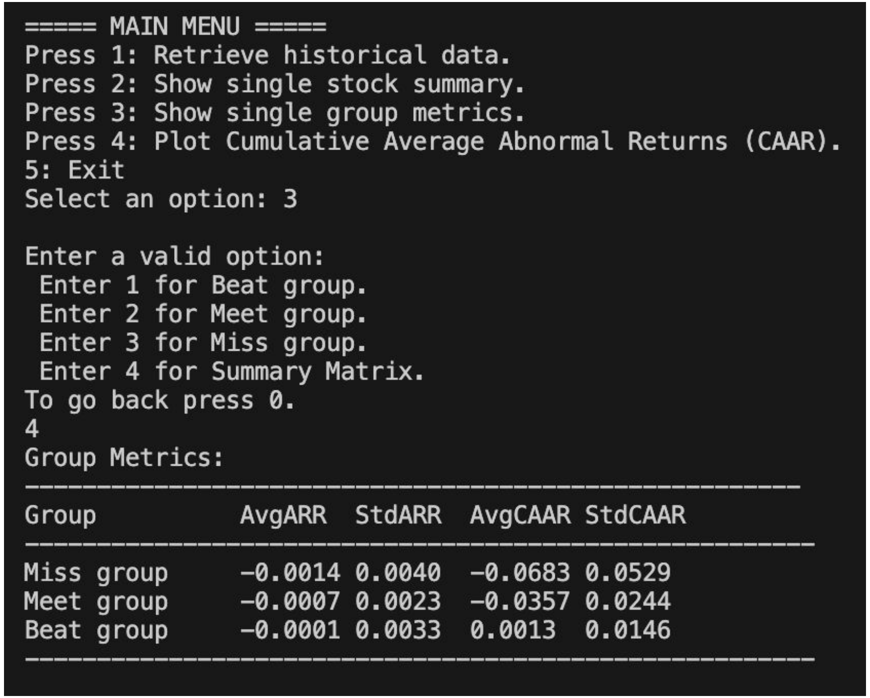


```
Enter a valid ticker. To go back press 0.
AAPL
-------------------------
Stock Information:
-------------------------
Ticker: AAPL
Group: Meet Earnings

Earning for Period Ending: 2024-09-30
Earnings Date Announcement: 2024-10-31
Estimated EPS: 1.6
Reported EPS: 1.64
Surprise: 0.04
Surprise Percent: 2.5

-------------------------
Daily Prices: (122 obs.)
-------------------------

2024-08-06 : 206.2654   2024-08-07 : 208.8433
2024-08-13 : 220.4949   2024-08-14 : 220.9434
2024-08-20 : 225.7166   2024-08-21 : 225.6070
2024-08-27 : 227.2313  2024-08-28 : 225.6966
2024-09-04 : 220.0764   2024-09-05 : 221.6010
2024-09-11 : 221.8801   2024-09-12 : 221.9897
```

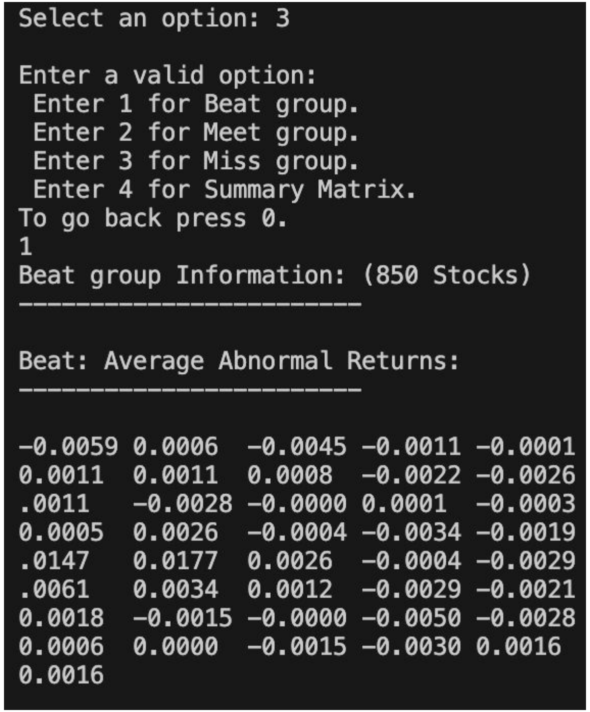


```
===== MAIN MENU =====
Press 1: Retrieve historical data.
Press 2: Show single stock summary.
Press 3: Show single group metrics.
Press 4: Plot Cumulative Average Abnormal Returns (CAAR).
5: Exit
Select an option: 3

Enter a valid option:
 Enter 1 for Beat group.
 Enter 2 for Meet group.
 Enter 3 for Miss group.
 Enter 4 for Summary Matrix.
To go back press 0.
4
Group Metrics:
-----------------------------------------------------
Group          AvgARR  StdARR  AvgCAAR StdCAAR
------------------------------------------------------
Miss group     -0.0014 0.0040  -0.0683 0.0529
Meet group     -0.0007 0.0023  -0.0357 0.0244
Beat group     -0.0001 0.0033  0.0013  0.0146
------------------------------------------------------
```

Outputs: 3. Data for a Group: Beat



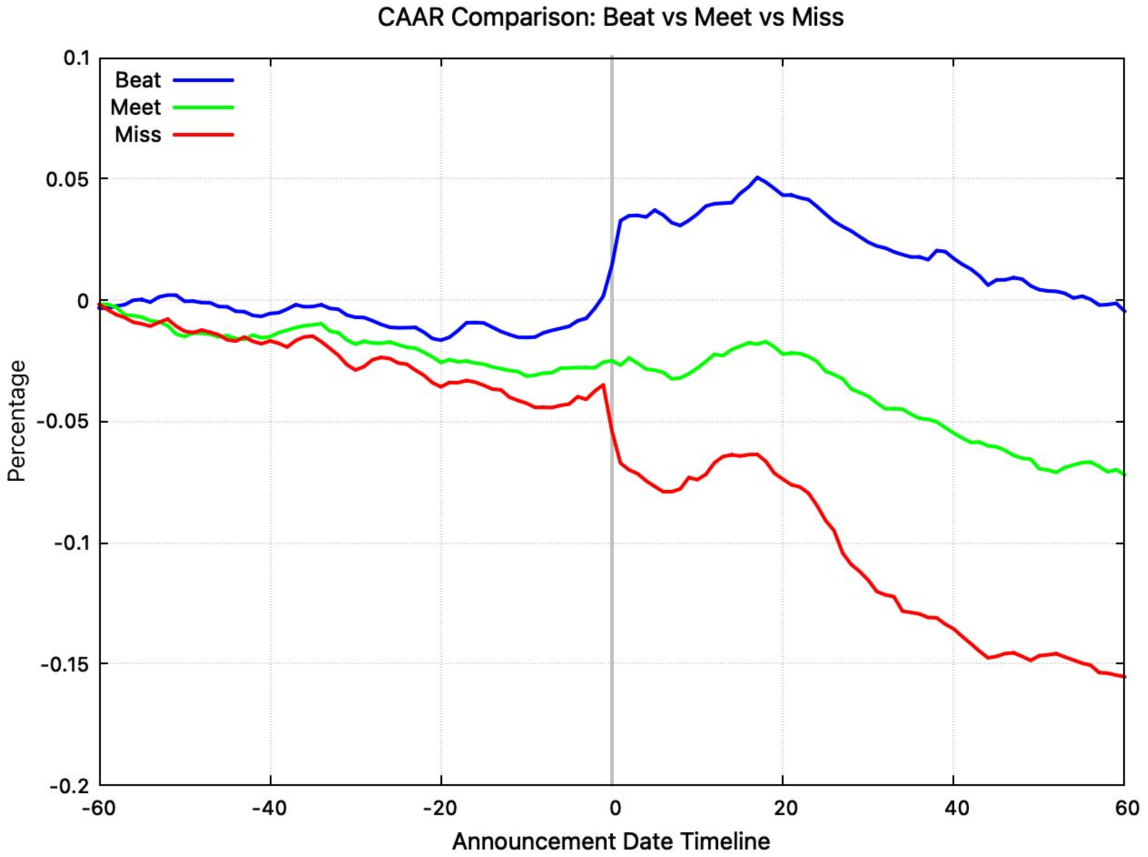
```
===== MAIN MENU =====
Press 1: Retrieve historical data.
Press 2: Show single stock summary.
Press 3: Show single group metrics.
Press 4: Plot Cumulative Average Abnormal Returns (CAAR).
5: Exit
Select an option: 3

Enter a valid option:
 Enter 1 for Beat group.
 Enter 2 for Meet group.
 Enter 3 for Miss group.
 Enter 4 for Summary Matrix.
To go back press 0.
4
Group Metrics:
------------------------------------------------------
Group          AvgARR  StdARR  AvgCAAR StdCAAR
-------------------------------------------------------
Miss group     -0.0014 0.0040  -0.0683 0.0529
Meet group     -0.0007 0.0023  -0.0357 0.0244
Beat group     -0.0001 0.0033  0.0013  0.0146
-------------------------------------------------------
```



```
Select an option: 3

Enter a valid option:
 Enter 1 for Beat group.
 Enter 2 for Meet group.
 Enter 3 for Miss group.
 Enter 4 for Summary Matrix.
To go back press 0.
1
Beat group Information: (850 Stocks)
------------------------


Beat: Average Abnormal Returns:
------------------------


-0.0059 0.0006  -0.0045 -0.0011 -0.0001
0.0011  0.0011  0.0008  -0.0022 -0.0026
.0011   -0.0028 -0.0000 0.0001  -0.0003
0.0005  0.0026  -0.0004 -0.0034 -0.0019
.0147   0.0177  0.0026  -0.0004 -0.0029
.0061   0.0034  0.0012  -0.0029 -0.0021
0.0018  -0.0015 -0.0000 -0.0050 -0.0028
0.0006  0.0000  -0.0015 -0.0030 0.0016
0.0016
```

NYU

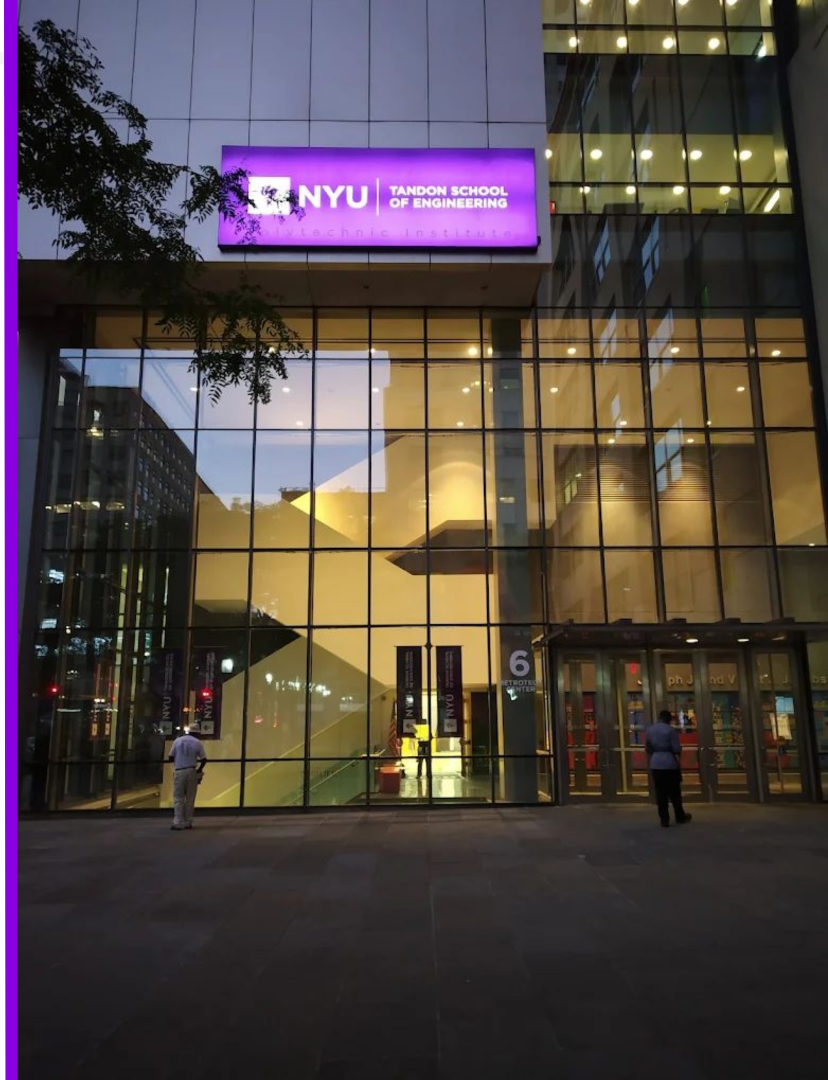
Outputs: 4. CAAR Comparison Plot



Contributions

Team Member	Contribution
Charles Wu	Libcurl API calls, Initial Structure (Retrieve & Portfolio), Debugging
Joaquin Garay	GNU Plot, Menu (main), Debugging
Juan Camilo Mennes (Team Leader)	Algorithm (DateRange & Bootstrapping), Code Structure, Debugging
Harsh Kulkarni	Project Code Structure, Debugging, Code Integration
Rhugved Bhojane	Algorithm (Bootstrapping), UML Diagram, Presentation

THANK YOU!



NYU