Evaluate the impact of quarterly earnings report on stock price movement

FRE-GY 6883 Financial Computing Course Team Project

Prepared by
Hanlu Ni, Huanxin Zhang, Xingyuan Ding, Juntao Zhang,
Peishan Liu, Qiaomin Wang

Prepared for
Dr. Song Tang

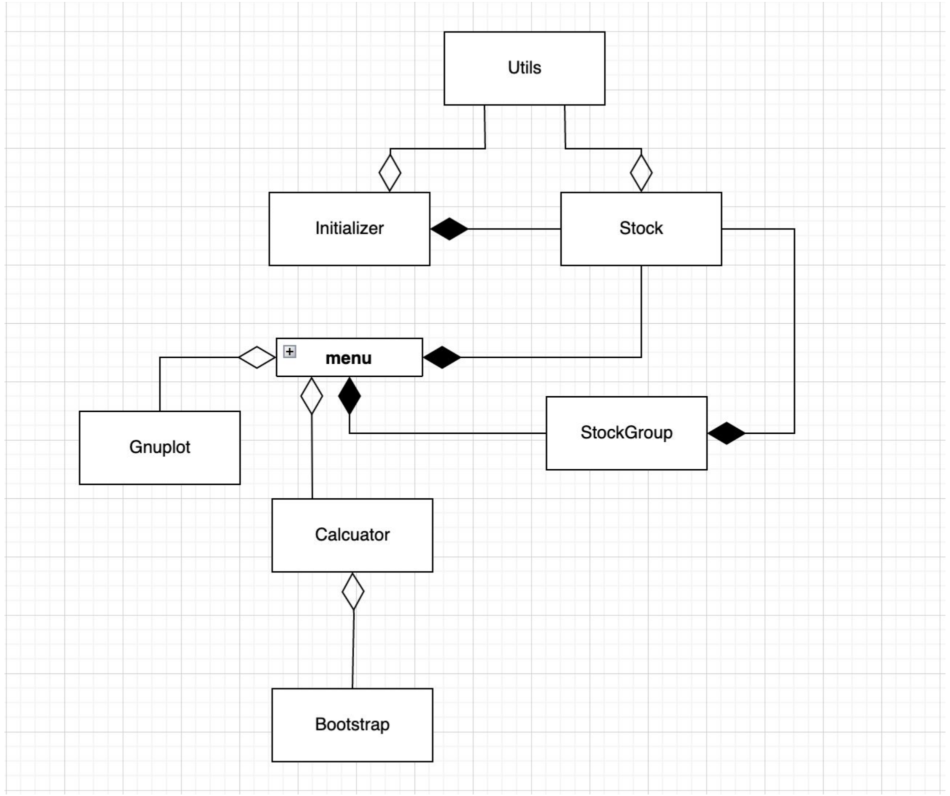
Task Allocation

Name	Task
Xingyuan Ding	Initiator
Hanlu Ni	StockGroup
Huanxin Zhang	Bootstrap
Juntao Zhang	Calculator
Peishan Liu	Gnuplot
Qiaomin Wang	Menu

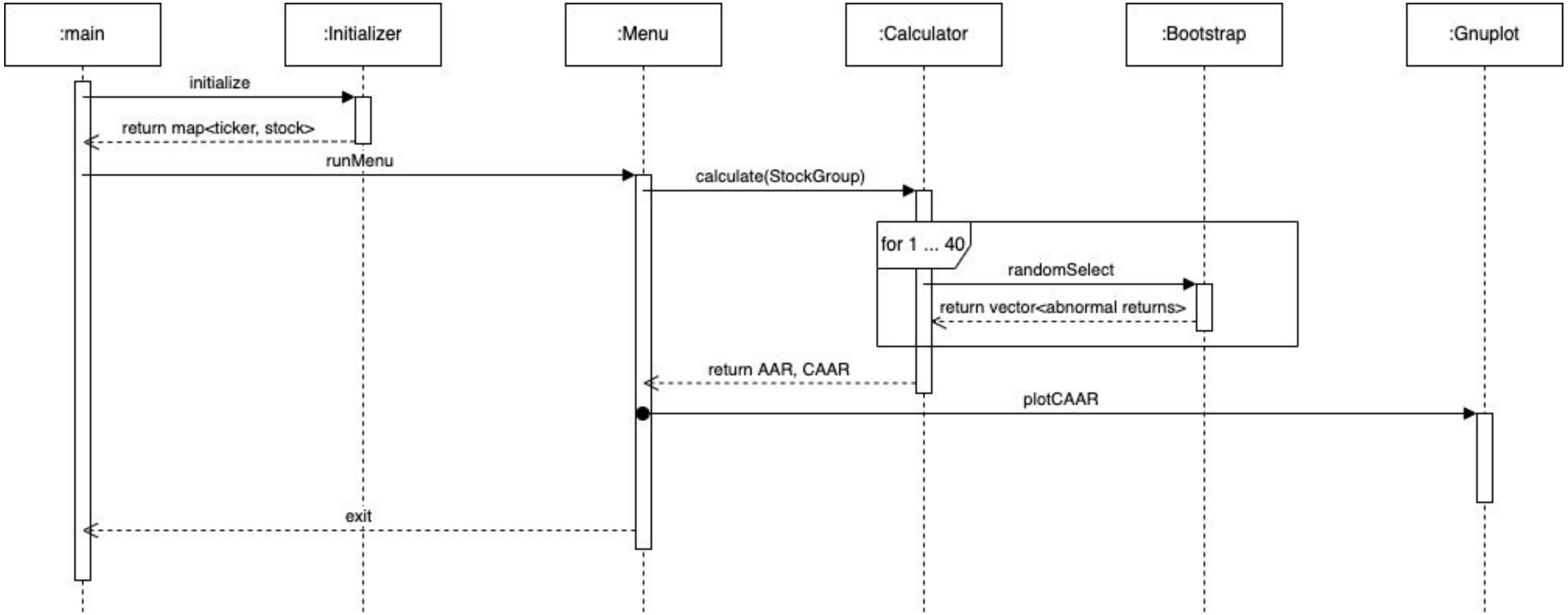
Executive Summary

- Purpose: create a C++ console interface to help people analyze the impact of earnings announcement on the Russell 1000 stock prices
- Steps:
  - Step1: Get data from eodhistorical.com
  - Step2: Categorize data into 3 groups
  - Step3: Bootstrap and calculate AAR and CAAR
  - Step4: Plot the results using Gnuplot

UML Diagram- Relationship among Modules



Flow Chart


:main
:Initializer
:Menu
:Calculator
:Bootstrap
:Gnuplot
initialize
return map<ticker, stock>
runMenu
calculate(StockGroup)
for 1 ... 40
randomSelect
return vector<abnormal returns>
return AAR, CAAR
plotCAAR
exit

Fetch data & clean data

- Read tickers from Russell 1000 stock list
- Retrieve the daily prices of all the tickers from EOD Historical Data with libcurl
- Retrieve the daily prices of the IWB benchmark
- Read the surprising factors of each stock from the given EarningAnnouncements.csv
- Combine the daily prices and the surprising factors of each stock into an object called Stock
- Create a map between tickers and Stock objects for future usage

Stock

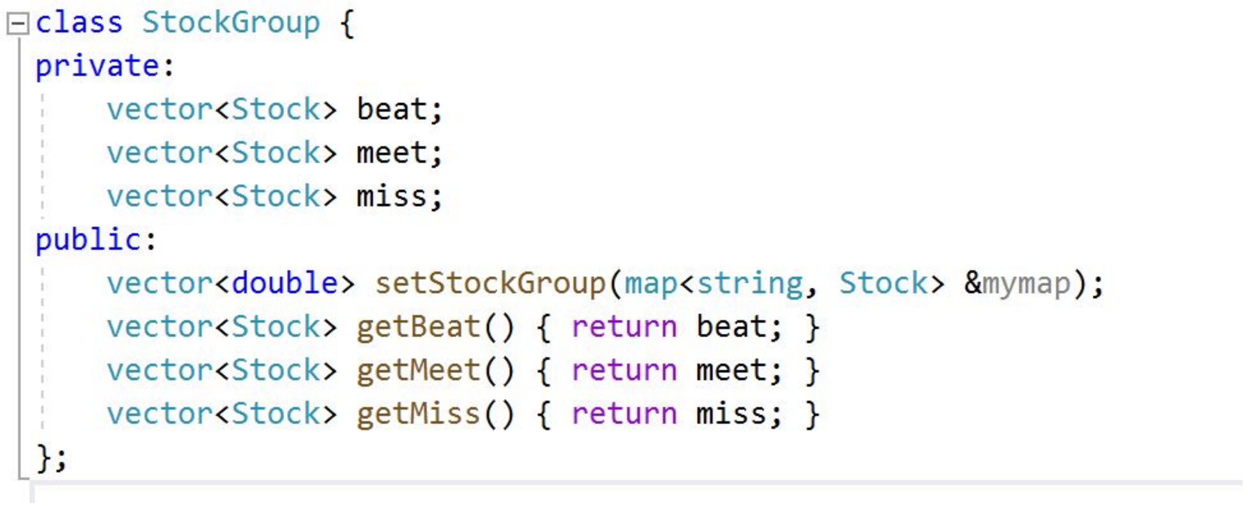
```cpp
class Stock {
private:
    string ticker;
    vector<string> dates;
    vector<double> prices;
    int earningDateIndex;
    string earningDate;
    string periodEndDate;
    double estimate;
    double reported;
    double surprise;
    double surprisePercentage;
public:
    Stock(){};
    Stock(string ticker, vector<string> dates, vector<double> prices, string earningDate, string periodEndDate,
        double estimate, double reported, double surprise, double surprisePercentage);

    vector<double> getprice(){return prices;}
    double getSurprisePercentage(){return surprise;}
    vector<double> calculateCumulativeDailyReturn() const;
    vector<double> calculateDailyReturn(const int n) const;
    vector<double> calculateAbnormalDailyRetrun(const StockPrice& benchmark, const int n) const;
    void displayInformation() const;
};
```

```cpp
struct StockPrice {
    vector<string> dates;
    vector<double> prices;
};
```

```cpp
struct EarningInfo {
    string earningDate;
    string periodEndDate;
    double estimate;
    double reported;
    double surprise;
    double surprisePercentage;
};
```

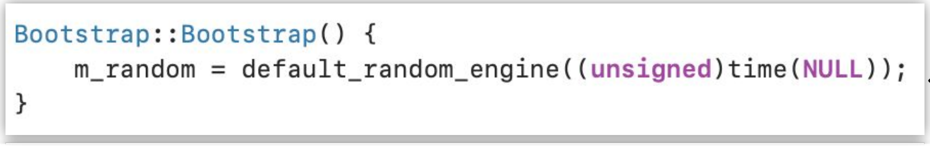
Categorize stocks

- Create class StockGroup to handle three groups



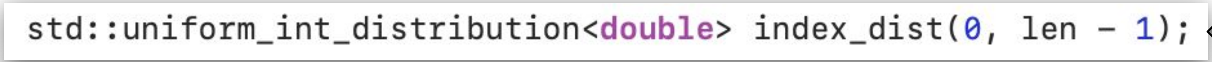
```
class StockGroup {
private:
    vector<Stock> beat;
    vector<Stock> meet;
    vector<Stock> miss;
public:
    vector<double> setStockGroup(map<string, Stock> &mymap);
    vector<Stock> getBeat() { return beat; }
    vector<Stock> getMeet() { return meet; }
    vector<Stock> getMiss() { return miss; }
};
```

Bootstrap Explanation



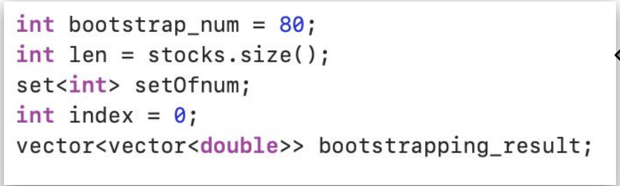
```
Bootstrap::Bootstrap() {
    m_random = default_random_engine((unsigned)time(NULL));
}
```

Use random number engine class to generates random numbers.



```
std::uniform_int_distribution<double> index_dist(0, len - 1);
```

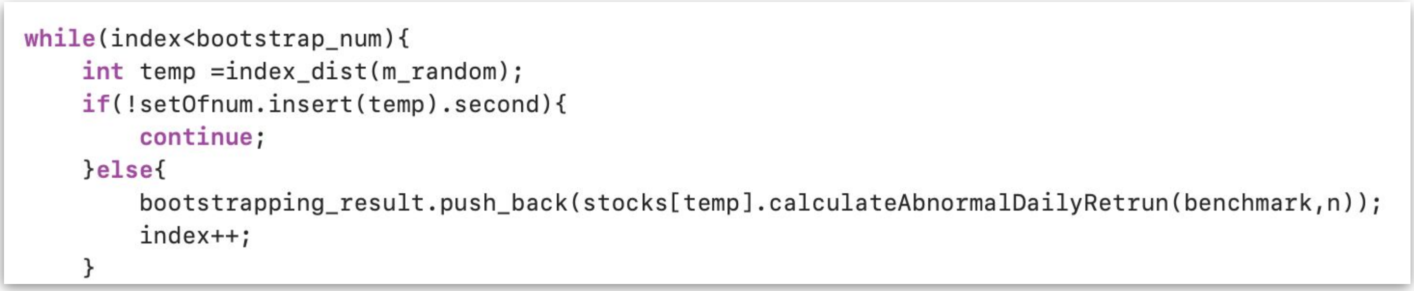
Produces random integer values uniformly distributed on the closed interval



```
int bootstrap_num = 80;
int len = stocks.size();
set<int> setOfnum;
int index = 0;
vector<vector<double>> bootstrapping_result;
```

Initializations

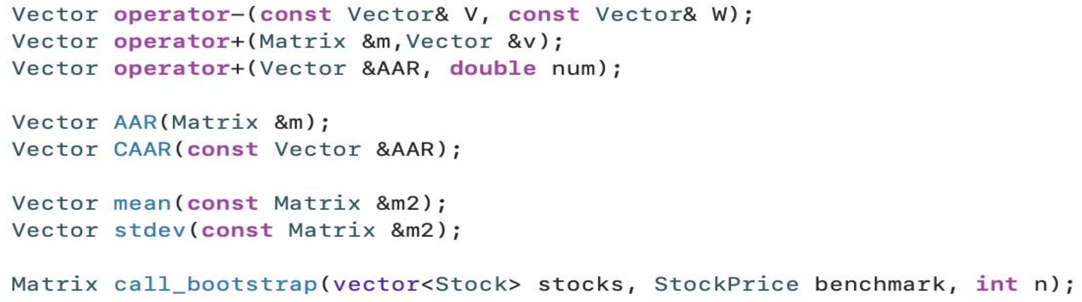
Bootstrap 80 times to get the difference between stock price return and IWB return for each randomly selected stock



```
while(index<bootstrap_num){
    int temp =index_dist(m_random);
    if(!setOfnum.insert(temp).second){
        continue;
    }else{
        bootstrapping_result.push_back(stocks[temp].calculateAbnormalDailyRetrun(benchmark,n));
        index++;
    }
```

Calculator

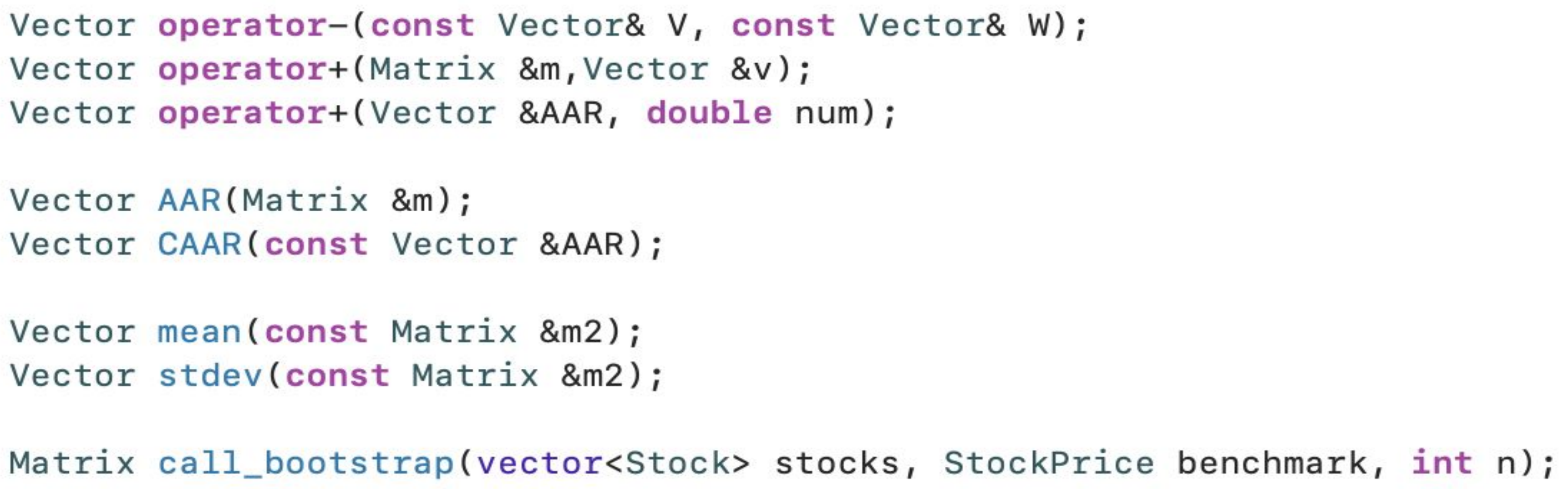
- We created vector operators to help us calculate matrices and vectors. We also used overload methods to help managing operators.
- We used call_bootstrap function to get the bootstrap. This function gives us a matrix of size 80x2N. Each cell contains $R_{it} - R_{mt}$
- $R_{it} = (Price_t - Price_{t-1}) / Price_{t-1}$
- For the output, we transformed data into two matrices: AAR and CAAR, and then we calculated the average and standard deviation of AAR and CAAR. We stored our outputs in a new matrix called Output.



```
Vector operator-(const Vector& V, const Vector& W);
Vector operator+(Matrix &m,Vector &v);
Vector operator+(Vector &AAR, double num);

Vector AAR(Matrix &m);
Vector CAAR(const Vector &AAR);

Vector mean(const Matrix &m2);
Vector stdev(const Matrix &m2);

Matrix call_bootstrap(vector<Stock> stocks, StockPrice benchmark, int n);
```

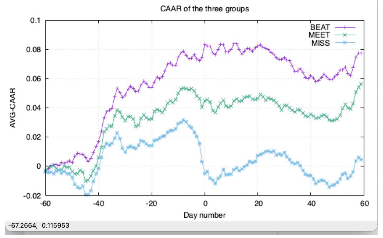
Calculator

- AAR: transform from matrix to a AAR vector
- CAAR: calculated cumulative average abnormal return
- Mean and stdev: calculated the mean and standard deviation of vector AAR and CAAR



```cpp
Vector operator-(const Vector& V, const Vector& W);
Vector operator+(Matrix &m,Vector &v);
Vector operator+(Vector &AAR, double num);

Vector AAR(Matrix &m);
Vector CAAR(const Vector &AAR);

Vector mean(const Matrix &m2);
Vector stdev(const Matrix &m2);

Matrix call_bootstrap(vector<Stock> stocks, StockPrice benchmark, int n);
```

Gnuplot



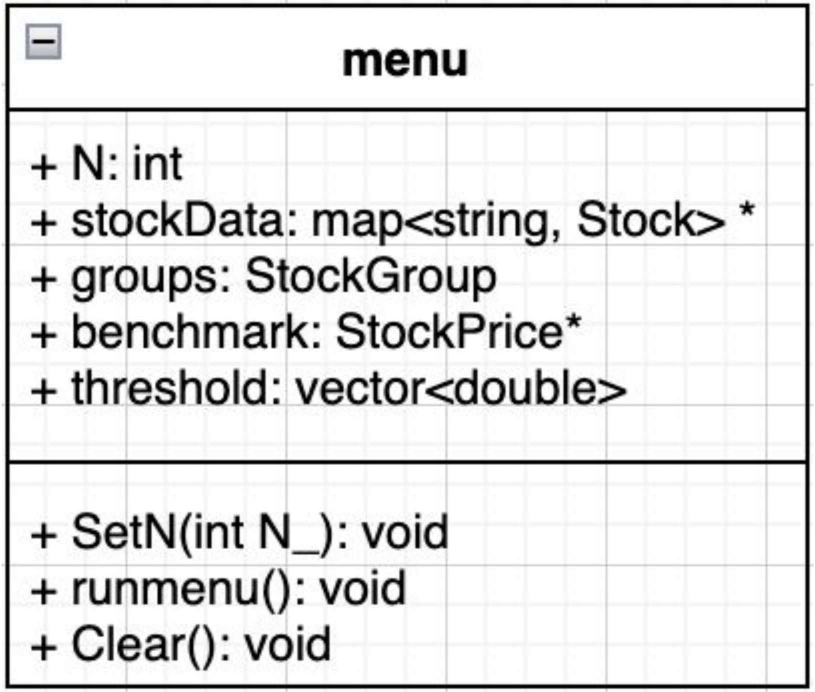
- **Reference:** http://www.gnuplot.info/
- **Gnuplot Flags:**
  - -persist: option so that the plot remains on the screen afterwards
  - '-' specifies that the data are inline
  - 'e' specifies the ending of each data block
- **Usage in C++ for inline data plotting:**

```
plotHandle = popen("/opt/local/bin/gnuplot -persist", "w");
fprintf(plotHandle, "plot '-' using 1:2 with linespoints t 'BEAT', '-' using 1:2 with
linespoints t 'MEET', '-' using 1:2 with linespoints t 'MISS'\n");
for (int i = 0; i < caar_group1.size(); i++)
{
    if (!isnan(caar_group1[i]) && caar_group1[i] < 10 && caar_group1[i] > -10) {
        fprintf(plotHandle, "%d %f\n", i-N, caar_group1[i]);
    }
}
fprintf(plotHandle, "%s\n", "e");
```

- **Plotting Performance Comparison**
  - Inline data vs. file writing to local

```
=======Menu=======
- Enter N (Default is 60) (enter 1)
- Pull information for one stock from one group (enter 2)
- Show AAR, AAR-SD, CAAR and CAAR-STD for one group (enter 3)
- Show the gnuplot graph with CAAR for all 3 groups (enter 4)
- Exit (enter 5)

Enter your choice: 1

Enter N: 60
The N you entered is: 60


=======Menu=======
- Enter N (Default is 60) (enter 1)
- Pull information for one stock from one group (enter 2)
- Show AAR, AAR-SD, CAAR and CAAR-STD for one group (enter 3)
- Show the gnuplot graph with CAAR for all 3 groups (enter 4)
- Exit (enter 5)

Enter your choice: 4

#####################
#####################
case 1 : inline data for ploting. It took: 772 microseconds.
case 2 : stored data for ploting. It took: 1307 microseconds.
```

Menu



N: Store the number of days

stockData: Store the stocks information

group: This is the three groups

benchmark: Store the IWB information

threshold: This is the thresholds split stocks

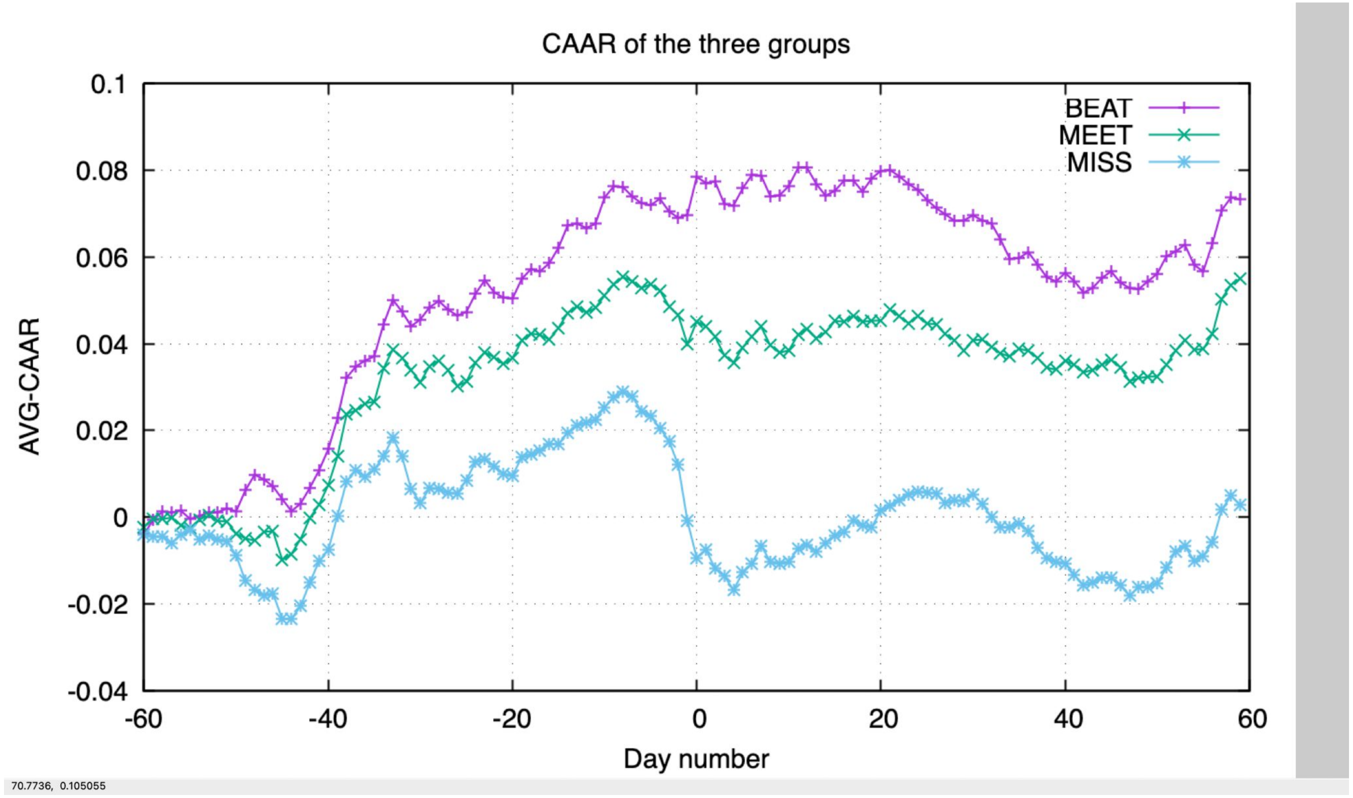
____

Set(int N_): Set the member N

runmenu(): Run the menu

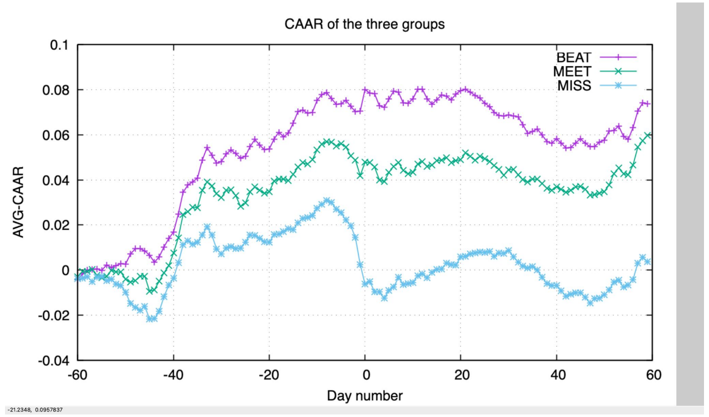
Clear(): Clear all potiner

Gnuplot Result



Research Conclusion

- Stocks in the miss group slide most on concern over bad earnings statement. This indicates market's reactions to a bad news.
- Before the announcement day, investors have higher expectation on beat group. Investors tend to be optimistic in general, as we see all three groups have positive abnormal returns before announcement day.
- After the announcement day, investors' reactions tend to be relatively stable, as we didn't observe volatile movements on the graph.



Enhancement

- Write stock data to local files to add offline support & allow developers to debug faster
- Using inline data for plotting, avoid storing data in local disk

Further Improvement

- Encapsulate our helper functions into objects
- Data analysis on Russell 1000 stocks, and define a better threshold to divide stocks into 3 groups
- Extend the time horizon of our analysis

Reference

- Data fetching (https://eodhistoricaldata.com/financial-apis/api-for-historical-data-and-volumes/)
- Gnuplot Documentation (http://www.gnuplot.info/)
- UML diagram (https://www.uml-diagrams.org/)

Thank You