



FRE-GY 6883 Financial Computing  
Course Team Projects

# Evaluate the impact of quarterly earnings report on stock price movement

**Team Members**

Zhiheng Wang  
Ken Chen  
Jingzhao Zhang  
Yiwen Zhuang  
Zedi Qiu

5/15/2021



# Introduction

- **Research Topic - PEAD anomaly**

- An earnings surprise occurs when a company's reported quarterly or annual profits are above or below analysts' expectations.
- The tendency of stock prices to drift (over time) in the direction of earnings surprise post the announcement of earnings is termed as PEAD anomaly.

- **Project description**

- C++ based console application
- Fetches data online and calculates statistics
- Present calculation results with GNU plot

## Task Allocation

Name	Task
Zhiheng Wang	Stock data storage design, 4 statistic calculation
Ken Chen	Zack query, Yahoo API query, calendar manager, sanity check, unit test, DB operations
Jingzhao Zhang	Vector/Matrix class, operator overloading, flow chart
Yiwen Zhuang	Bootstrap algorithm, Main engine
Zedi Qiu	GNU plot, Main engine design

## Group Selected Stocks (Matlab)

### Goals:

- From Zacks, **pull 2020 3rd quarter earnings releases**
- For all Russell 1000 stocks, **sort and divide them into 3 groups**

### Details:

$$\text{Surprise\%} = \frac{\text{Reported EPS} - \text{EPS Estimate}}{\text{abs(EPS Estimate)}}$$

- Calculate **earnings surprise** for each stock (Surprise % from Zacks).
- Sort all the surprises in **ascending order**, and split all the stocks into **3 groups** with relatively equivalent numbers of stocks:
  - Highest surprise group: Beat Estimate Group
  - Lowest surprise group: Miss Estimate Group
  - The rest stocks in between: Meet Estimate Group
- Save the results in a **CSV file(s)** for using by your C++ application.

```
function [ earnings ] = scrapeEarningsZacks( Stock )  
s=urlread('http://zacks.thestreet.com/CompanyView.php','post',{'ticker',Stock});  
try etst=strfind(s,'Surprise%</strong></div></td>');  
.....
```

```
% sort surprises data in ascending order, get loc(ranking)  
surprises = cell2mat(results(:,7));  
[~, loc] = sort(surprises);  
.....
```

```
% split all the stocks into 3 groups  
for i = 1:3  
    group_start = floor(used_ticker_nums * (i-1) / 3) + 1;  
    group_end = floor(used_ticker_nums * i / 3);  
    disp([group_start, group_end]);  
    results_group = sorted_results(group_start:group_end, :);  
    if i == 1  
        filename = 'Miss';  
    elseif i==2  
        filename = 'Meet';  
    else  
        filename = 'Beat';  
    end  
    writecell(results_group, [filename, '.csv']);  
end  
.....
```

## Group Selected Stocks

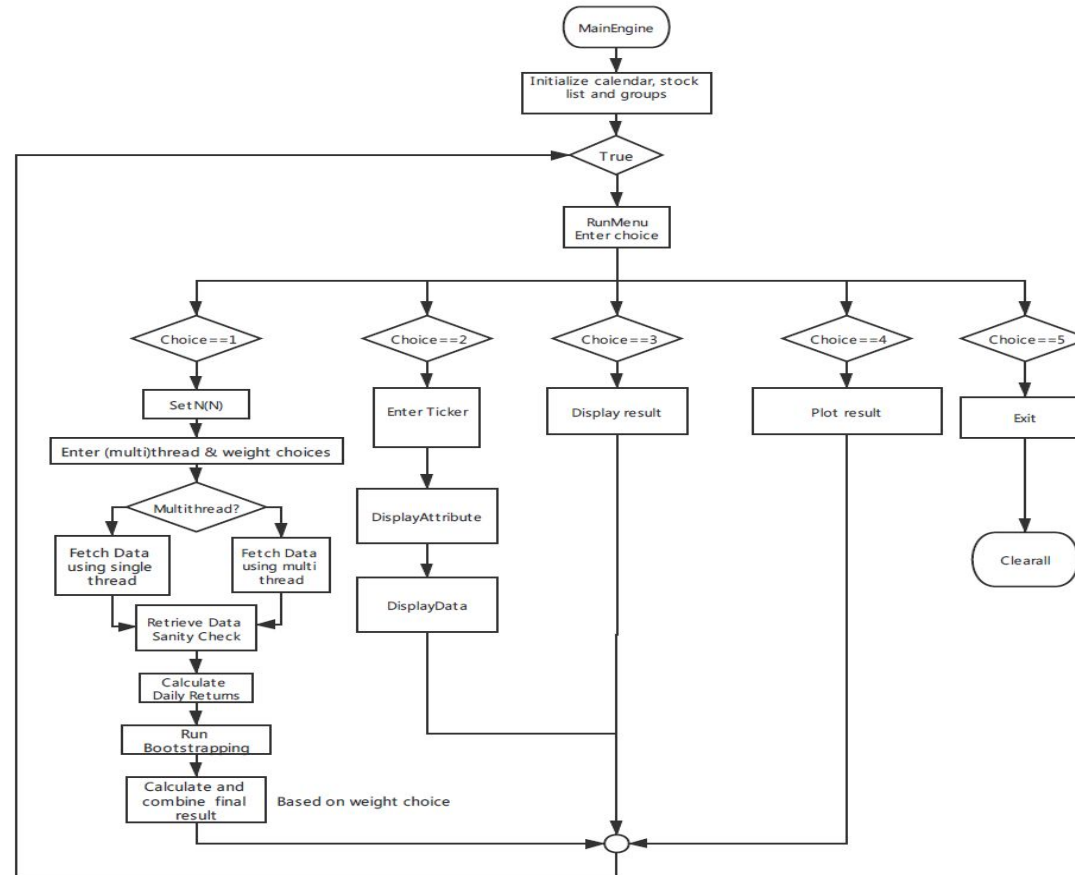
### Files

- **GetEPS.m**: main script to query data and split data.
- **scrapeEarningsZacks.m**: lib to query data from Zacks.
- **FilterEPS.m**: filter and split data based on all\_surprises.csv.
- **all\_surprises.csv**: ungrouped data.
- **Miss.csv**: Miss estimate group data.
- **Meet.csv**: Meet estimate group data.
- **Beat.csv**: Beat estimate group data.
- **Russell\_1000\_component\_stocks.csv**: Russell 1000 stock names.

### Data example:

Ticker	Announcement Date	Period Ending	Estimate	Reported	Surprise	Surprise%
AAPL	27-JAN-2021	Oct 2020	1.41	1.68	0.27	19.15
AI	29-OCT-2020	Sep 2020	0.69	0.73	0.04	5.80

# Flow Chart





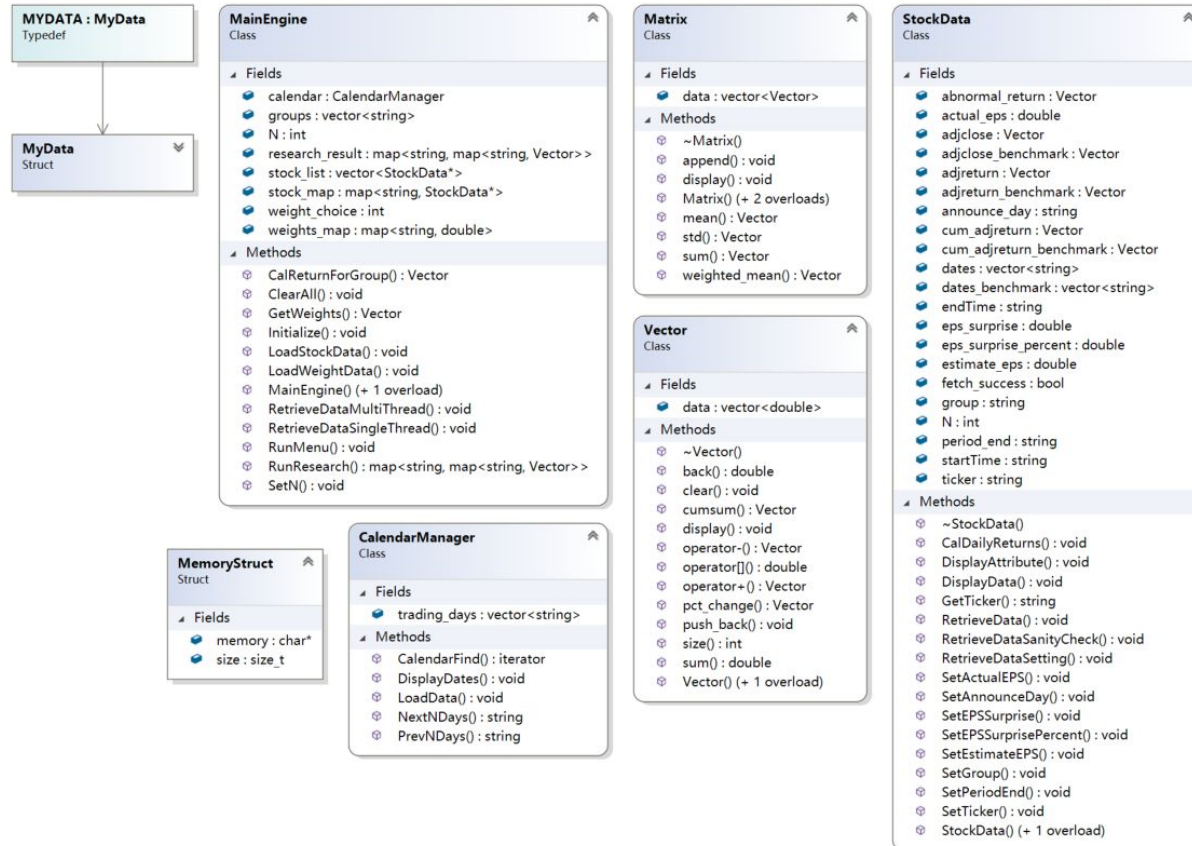
## Flow Chart Explanation 1

Create an instance of MainEngine that combines all methods

### Initialize()

- |   |  |                              |   |
|---|--|------------------------------|---|
| 1 | CalendarManager <b>calendar</b>          | calendar.LoadData()          | -get all trading days   |
| 2 | vector<StockData*> <b>stock_list</b>     | LoadStockData()              | -get stock list for 3 groups from local grouped EPS files<br>-call load_stock_data() from Utils.cpp<br>-initialized a stock_list of type vector<StockData*> |
| 3 | map<string, double> <b>weights_map</b>   | LoadWeightData()             | -load IWB weights from local csv file   |
| 4 | map<string, StockData*> <b>stock_map</b> | create_stock_map(stock_list) | -create a stock_map using function in Utils.cpp<br>-ticker with StockData* that contains all info about each stock  |
| 5 | vector<string> <b>groups</b>             | push_back("")                | -initialize by push back 3 groups' name   |

# UML





## Explanations: Vector class

- **Function**

Store vector<double> type of data

Methods to perform calculations

Vector
+data: vector<double>
+push_back(const double &val)
+clear()
+size()
+back()
+pct_change()
+sum()
+cumsum()
+display()
+operator-
+operator[]
+operator+

push\_back new value  
clear all values in the vector  
get the size of the vector  
get the last element of the vector  
calculate % change by row  
calculate sum of all elements  
calculate the cumulative sum  
display all elements  
take the difference of two Vectors  
access value in Vector by index  
add two Vectors together

## Explanations: Matrix class

- **Function**

Store vector<Vector> type of data  
Methods to perform calculations for  
AAR, AAR-STD, CAAR, CAAR-STD

Matrix
+data: vector<Vector>
+append() +mean() +std() +sum() +weighted_mean() +display()

append row at the end  
calculate mean by column  
calculate std by column  
calculate sum by column  
calculate weighted mean by column  
display elements in the matrix

## Explanation: CalendarManager class

- **Function**

Locate the all the trading dates and earning announcement date and display the trading dates before and after the earnings announcement date

CalendarManager
- trading_days: vector<string>
+ LoadData() + CalendarFind() + NextNDays() + PrevNDays() + DisplayDates()

All trading days from 1/3/2011 to 4/27/2021.

Load trading days from local file.

Find the location of a date in a vector.

Find the N-th trading day after day0.

Find the N-th trading day before day0.

Display trading\_days.

## StockData class

- Calculate and display the information for each single stock in the index.

- **Function**

Calculate and display the  
information for each single  
stock in the index

StockData
- ticker
- group
- announce_day
- period_end
- estimate_eps
...
- startTime
- endTime
- N
- fetch_success
- dates
- adjclose
- adjreturn
- adjclose_benchmark
...
- abnormal_return
...
+ RetrieveDataSetting()
+ RetrieveDataSanityCheck()
+ CalDailyReturns()
+ DisplayAttribute()
+ DisplayData()

Input N.

If download data success, fetch\_success=true.

Set startTime, endTime. Clear previous data.

After fetching data, sanity check if data is valid.

Calculate return and abnormal return.

Display attribution information of the stock.

Display price/return data of the stock.

## Explanations: MainEngine class

- **Function:** pull the information for each individual stock, conduct research and display the results. Clients can access the stock data and plot the results using manu.

MainEngine	
-Stock_list	Store the stocks and corresponding price information for each group in a STL map
-ticker	Stock symbol
-weight	Stock Market Cap weights according to the bechmark
-AAR	Abnromal return for each stock
-CAAR	Cumuated Abnormal Return for first T days
-AARt	Daily abnormal returns for each group of stocks
-AAR_mean	Mean of AARt from bootstrap Algorithm
-CAAR_mean	Mean of AARt from bootstrap Algorithm
-AAR_std	Standard Deviation of AARt from bootstrap Algorithm
-CAAR_std	Standard Deviation of CAAR from bootrap Algorithm
-% Earnings Surprise	(Reported EPS- EPS Estimate)/abs(EPS Estimate)
+ Initialize()	Initialize stock list and the group information
+ LoadStockData()	Load and combine the stock list
+ LoadWeightData()	Load the stock market cap weights according to the benchmark
+ RetrieveDataSingleTread()	Retrieve all the stock data and display return information with single thread
+ RetrieveDataMultiTread()	Retrieve all the stock data and display return information with multiple thread
	Run bootstrap algorithm and calculate the mean and standard deviation of Abnormal returns and the cumulative standard abnormal returns
+ RunResearch()	
+ CalReturnForGroup()	Calculate the abnormal return for each group of stock
+ GetWeights()	Get the weights information for each group of stock
+ RunMenu()	Run the Manu which can search the information for individual stock, and AAR, AAR-SD, CAAR, and CAAR_STD for one group
+ ClearAll()	Delete all the preivously initialized pointers

## Explanations: Bootstrap algorithm

```
map<string, vector<StockData*>> bootstrapping(vector<StockData*> stock_list, int seed)
{
    int bootstrap_num = BOOTSTRAP_NUM;
    default_random_engine random(time(NULL) + seed);
    std::uniform_real_distribution<double> dist(0.0, 1.0);

    → vector< pair<StockData*, double> > vec;
    for (int i = 0; i < stock_list.size(); i++)
    {
        vec.push_back(pair<StockData*, double>(stock_list[i], dist(random)));
    }

    → sort(vec.begin(), vec.end(), cmp);
```

**Shuffle stocks** by giving random number to each stock and sort by the value.

**Select 50 stocks for each groups**  
(eg. “Beat Group”)

```
vector<StockData*> beat_result;
for (auto iter : vec)
{
    if ((iter.first->group == "Beat") && (iter.first->fetch_success))
    {
        beat_result.push_back(iter.first);
        if (beat_result.size() == bootstrap_num) break;
    }
}
```

```
map<string, vector<StockData*>> bootstrap_result;
bootstrap_result["Beat"] = beat_result;
bootstrap_result["Meet"] = meet_result;
bootstrap_result["Miss"] = miss_result;

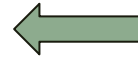
return bootstrap_result;
```

**Store** data in a map for further calculation and plotting

## Explanations: Bootstrap algorithm

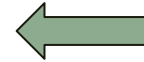
```
map<string, map<string, Vector>> MainEngine::RunResearch()
```

```
map<string, vector<Vector>> AAR;  
map<string, vector<Vector>> CAAR;  
map<string, map<string, Vector>> result;
```



Map initialization

```
for (int i = 0; i < RUN_BOOTSTRAP_NUM; i++)  
{  
    map<string, vector<StockData*>> bootstrap_result = bootstrapping(stock_list, i);  
    for (auto iter : groups)  
    {  
        AAR[iter].push_back(CalReturnForGroup(bootstrap_result[iter]));  
        CAAR[iter].push_back(AAR[iter].back().cumsum());  
    }  
}
```



**Bootstrap 40 times;**  
get AAR CAAR for  
each time.

Get average and standard deviation  
for AAR, CAAR for each group



```
for (auto iter : groups)  
{  
    Matrix AAR_mean(AAR[iter]);  
    Matrix CAAR_mean(CAAR[iter]);  
    //AAR_mean.display(); // check if random see valid  
    result["AAR_mean"][iter] = Matrix(AAR[iter]).mean();  
    result["CAAR_mean"][iter] = Matrix(CAAR[iter]).mean();  
    result["AAR_std"][iter] = Matrix(AAR[iter]).std();  
    result["CAAR_std"][iter] = Matrix(CAAR[iter]).std();  
}  
research_result = result;  
return result;
```



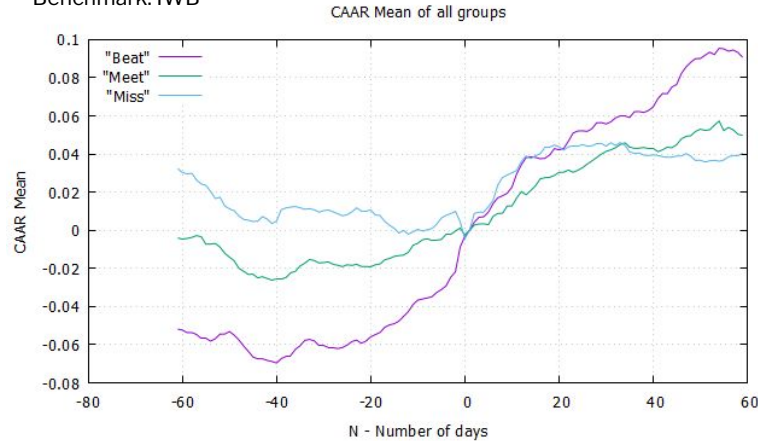
# Final Result

## Equal Weights

$$AAR_t = \frac{1}{M} \sum_{i=1}^M AR_{it}$$

$$CAAR = \sum_{t=-N}^T ARR_t, T = N$$

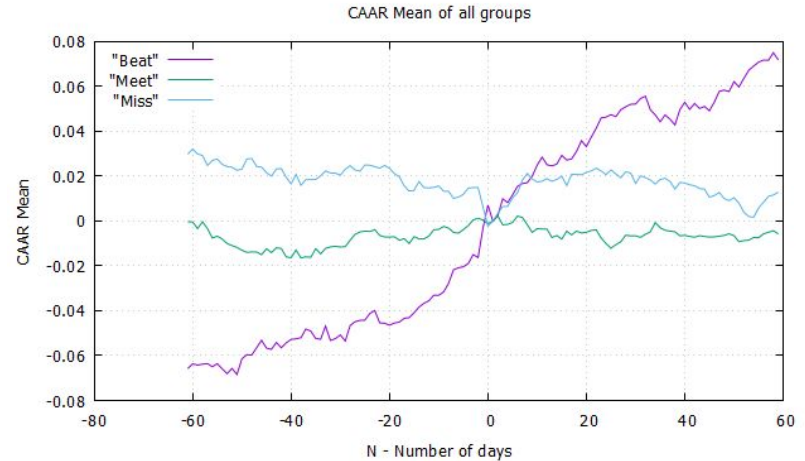
Benchmark: IWB



## Market Cap Weights

$$AAR_t = \frac{1}{\sum_{i=1}^M w_i} \sum_{i=1}^M AR_{it} * w_i$$

Benchmark: EAQL







## Research Conclusion

- Before announcement day, stocks in Beat group show positive abnormal return while stocks in Miss group show negative abnormal return. This might because investors are reacting to the information in the markets before earnings announcement and market gradually prices in the earnings surprise.
- After announcement day, stocks in Beat group show more positive abnormal returns than other groups. The effect lasts for 60 days which probably implies that investors react slowly to the earnings surprise information. Stocks in Meet/Miss group shows no significant abnormal returns.

# Enhancement

- **Unit test – Google Test**

Test-Driven development.

```
Microsoft Visual Studio Debug Console
Running main() from c:\a_work\32\s\thirdparty\googletest\googl
===== Running 10 tests from 4 test cases.
Global test environment set-up.
-----
RUN      1 test from BootstrapTest
OK       BootstrapTest.resultSize
OK       BootstrapTest.resultSize (22 ms)
-----
1 test from BootstrapTest (23 ms total)

-----
RUN      3 tests from CalendarManagerTest
OK       CalendarManagerTest.prevDays
RUN      CalendarManagerTest.prevDays (11 ms)
OK       CalendarManagerTest.nextDays
RUN      CalendarManagerTest.nextDays (11 ms)
OK       CalendarManagerTest.nextDaysOutOfBounds
RUN      CalendarManagerTest.nextDaysOutOfBounds (11 ms)
N too large for future days
N too large for future days
OK       CalendarManagerTest.nextDaysOutOfBounds (11 ms)
-----
3 tests from CalendarManagerTest (34 ms total)

-----
RUN      3 tests from VectorTest
OK       VectorTest.size
OK       VectorTest.size (0 ms)
RUN      VectorTest.pushBack
OK       VectorTest.pushBack (0 ms)
RUN      VectorTest.operatorOverload
OK       VectorTest.operatorOverload (0 ms)
-----
3 tests from VectorTest (2 ms total)

-----
RUN      3 tests from MatrixTest
OK       MatrixTest.sum
OK       MatrixTest.sum (1 ms)
RUN      MatrixTest.mean
OK       MatrixTest.mean (0 ms)
RUN      MatrixTest.weighted_mean
OK       MatrixTest.weighted_mean (0 ms)
-----
3 tests from MatrixTest (5 ms total)

-----
Global test environment tear-down
=====
10 tests from 4 test cases ran. (67 ms total)
PASSED  10 tests.
```

- **Fetch data with multi-threading**

Producer-consumers model.

Create thread pool.

Time consumption decreases from 660s to 88s  
with a pool of 10 threads.

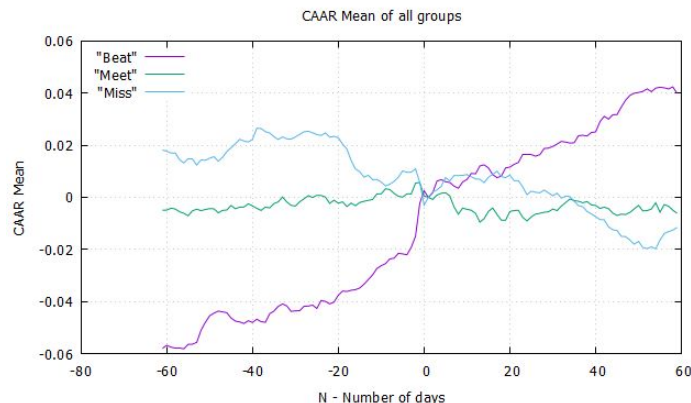
```
void thread_producer(tasks){
    for(task in tasks)
        mutex.lock();
        queue.push_front(task);
        mutex.unlock();
        cond_variable.notify_all();
    mutex.lock();
    queue.push_front(poison_pill);
    mutex.unlock();
    cond_variable.notify_all();
}
```

```
void thread_consumer(){
    while(1){
        while (queue.empty())
            cond_variable.wait(mutex);
        if(queue.back()==poison_pill)
            terminate thread;
        else:
            mutex.lock();
            task = pop_back();
            mutex.unlock();
            run task;
    }
}
```

# Enrichment

- EQAL as benchmark**

We calculate AARt by equal weights.  
It makes sense to use equal weighted  
Russell 1000 ETF as benchmark.  
However EQAL is not as liquid as IWB.



- Manage local data with SQLite3 DB**

Data loading time decreases from 660s to 13s.  
Robust to Internet/Yahoo API conditions.

StockInfo table

	Ticker	EPS_Ratin	EPS_Actus	EPS_Surpr	EPS_Surpr	Stock_Src	Announc_Da	Period_En
1	NOV	0.1	-0.61	-0.71	-710	Miss	2019-10-28	Sep 2019
2	PEGA	-0.06	-0.23	-0.17	-283.3	Miss	2019-11-07	Sep 2019
3	SBAC	2.08	0.19	-1.89	-90.8	Miss	2019-10-28	Sep 2019
4	INNVH	0.31	0.06	-0.25	-80.6	Miss	2019-10-29	Sep 2019
5	WYNN	0.88	0.17	-0.71	-80.6	Miss	2019-11-06	Sep 2019
6	VST	0.94	0.25	-0.69	-73.4	Miss	2019-11-05	Sep 2019
7	OXY	0.41	0.11	-0.3	-73.1	Miss	2019-11-04	Sep 2019
8	FSLR	1.06	0.29	-0.77	-72.6	Miss	2019-10-24	Sep 2019
9	MOS	0.27	0.08	-0.19	-70.3	Miss	2019-11-04	Sep 2019
10	LBRDK	0.44	0.15	-0.29	-65.9	Miss	2019-11-01	Sep 2019
11	NXST	1.52	0.61	-0.91	-59.8	Miss	2019-11-06	Sep 2019
12	NLOK	0.42	0.18	-0.24	-57.1	Miss	2019-11-07	Sep 2019
13	TRGP	-0.22	-0.34	-0.12	-54.5	Miss	2019-11-07	Sep 2019
14	RNR	0.63	0.29	-0.34	-53.9	Miss	2019-10-29	Sep 2019
15	SGEN	-0.37	-0.54	-0.17	-45.9	Miss	2019-10-29	Sep 2019
16	CVNA	-0.39	-0.56	-0.17	-43.5	Miss	2019-11-06	Sep 2019
17	AIG	0.99	0.56	-0.43	-43.4	Miss	2019-11-01	Sep 2019
18	TRV	2.38	1.43	-0.95	-39.9	Miss	2019-10-22	Sep 2019
19	TDS	0.23	0.15	-0.08	-34.7	Miss	2019-10-31	Sep 2019
20	BA	2.04	1.45	-0.59	-28.9	Miss	2019-10-23	Sep 2019
21	USM	0.38	0.27	-0.11	-28.9	Miss	2019-10-31	Sep 2019
22	SRPT	-1.34	-1.7	-0.36	-26.8	Miss	2019-11-07	Sep 2019
23	ATH	1.8	1.34	-0.46	-25.5	Miss	2019-11-05	Sep 2019
24	MCY	1	0.78	-0.22	-22	Miss	2019-10-28	Sep 2019
25	AGR	0.51	0.4	-0.11	-21.5	Miss	2019-10-29	Sep 2019
26	ASH	0.98	0.77	-0.21	-21.4	Miss	2019-11-18	Sep 2019
27	TDC	0.4	0.32	-0.08	-20	Miss	2019-11-07	Sep 2019
28	WUEC	1.16	0.63	-0.53	-20	Miss	2019-10-16	Sep 2019

MarketData table

	Ticker	Date	Adj_Close
1	NOV	2019-09-16	23.227243
2	NOV	2019-09-17	22.492393
3	NOV	2019-09-18	22.780378
4	NOV	2019-09-19	22.730722
5	NOV	2019-09-20	22.403021
6	NOV	2019-09-23	22.532118
7	NOV	2019-09-24	21.409979
8	NOV	2019-09-25	22.124969
9	NOV	2019-09-26	21.91643
10	NOV	2019-09-27	21.886641
11	NOV	2019-09-30	21.052485
12	NOV	2019-10-01	20.714849
13	NOV	2019-10-02	20.327562
14	NOV	2019-10-03	20.774431
15	NOV	2019-10-04	20.893599
16	NOV	2019-10-07	20.546032
17	NOV	2019-10-08	20.138887
18	NOV	2019-10-09	20.049511
19	NOV	2019-10-10	20.536102

## Reference

- PEAD anomaly  
([https://en.wikipedia.org/wiki/Post%E2%80%93earnings-announcement\\_drift](https://en.wikipedia.org/wiki/Post%E2%80%93earnings-announcement_drift))
- US trading calendar  
([https://tushare.pro/document/2?doc\\_id=253](https://tushare.pro/document/2?doc_id=253))
- C++ Concurrency in Action by Anthony Williams  
(<https://www.cplusplusconcurrencyinaction.com/>)
- Google Test  
(<https://github.com/google/googletest>)
- IWB Holdings Weights  
(<https://www.ishares.com/us/products/239707/ishares-russell-1000-etf>)
- SQLite for C++  
([https://www.tutorialspoint.com/sqlite/sqlite\\_c\\_cpp.htm](https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm))



Thank you