Evaluating the Impact of Quarterly Earnings Report on Stock Price Movement

FRE-GY 6883

Financial Computing

Team Member

Tejus Setlur
Eeshaan Asodekar
Naman Rathi
Syed Ahzam Tariq
Xuanhong Shao
Adios Utebayev

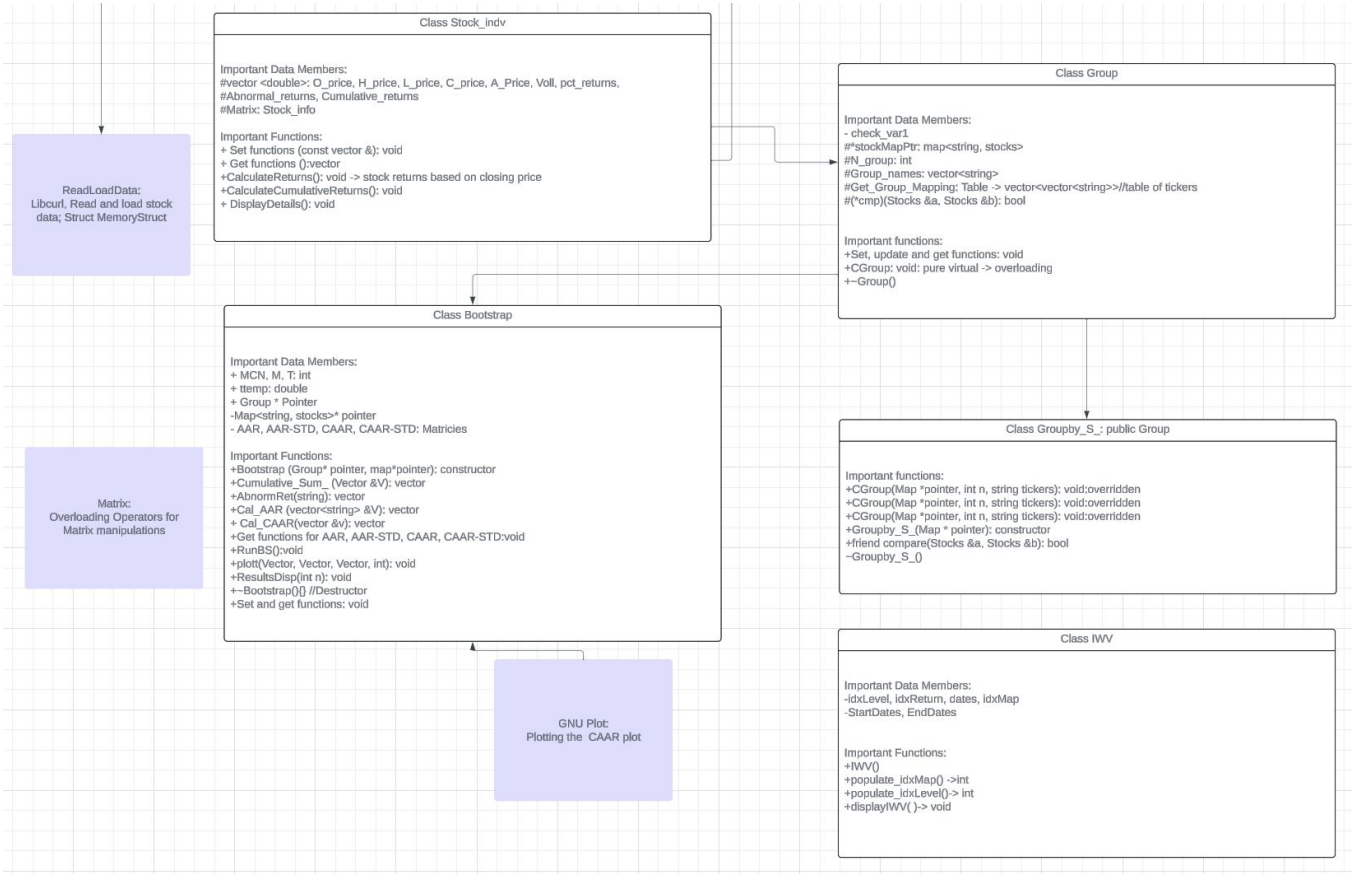
Project Outline

- Our project aimed to examine the returns of stocks in the Russell 3000 index and study how financial reporting affects their returns.

- We collected stock price data via the EODHD API and sorted them according to surprise levels, analyzing the Average Abnormal Return (AAR) and Cumulative AAR (CAAR) for each category through a bootstrap method.

- The CAAR versus time plot revealed a distinct inflection point for beat and miss groups around their earnings announcement date, leading to varying performance paths for each group thereafter.

UML Design

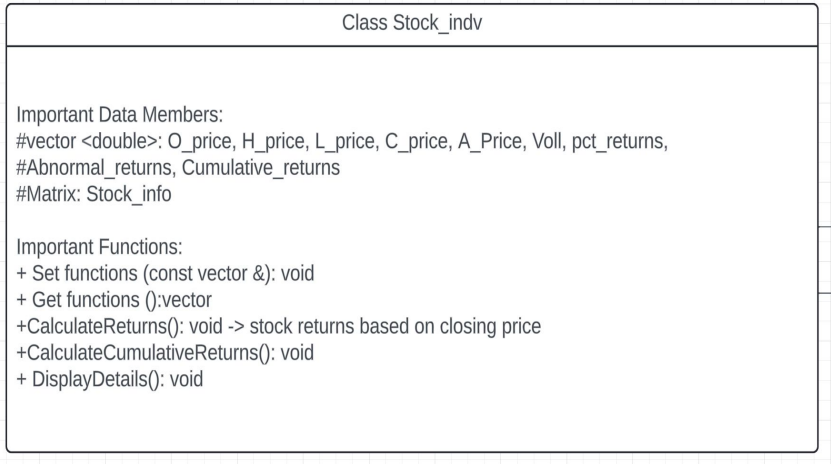

Class Stock_indv
Important Data Members:
#vector <double>: O_price, H_price, L_price, C_price, A_Price, Voll, pct_returns,
#Abnormal_returns, Cumulative_returns
#Matrix: Stock_info
Important Functions:
+ Set functions (const vector &): void
+ Get functions ():vector
+CalculateReturns(): void -> stock returns based on closing price
+CalculateCumulativeReturns(): void
+ DisplayDetails(): void
Class Group
Important Data Members:
- check_var1
#*stockMapPtr: map<string, stocks>
#N_group: int
#Group_names: vector<string>
#Get_Group_Mapping: Table -> vector<vector<string>>//table of tickers
#(*cmp)(Stocks &a, Stocks &b): bool
Important functions:
+Set, update and get functions: void
+CGroup: void: pure virtual -> overloading
+~Group()
ReadLoadData:
Libcurl, Read and load stock data; Struct MemoryStruct
Class Bootstrap
Important Data Members:
+ MCN, M, T: int
+ ttemp: double
+ Group * Pointer
-Map<string, stocks>* pointer
- AAR, AAR-STD, CAAR, CAAR-STD: Matricies
Important Functions:
+Bootstrap (Group* pointer, map*pointer): constructor
+Cumulative_Sum_ (Vector &V): vector
+AbnormRet(string): vector
+Cal_AAR (vector<string> &V): vector
+ Cal_CAAR(vector &v): vector
+Get functions for AAR, AAR-STD, CAAR, CAAR-STD:void
+RunBS():void
+plott(Vector, Vector, Vector, int): void
+ResultsDisp(int n): void
+~Bootstrap(){} //Destructor
+Set and get functions: void
Class Groupby_S_: public Group
Important functions:
+CGroup(Map *pointer, int n, string tickers): void:overridden
+CGroup(Map *pointer, int n, string tickers): void:overridden
+CGroup(Map *pointer, int n, string tickers): void:overridden
+Groupby_S_(Map * pointer): constructor
+friend compare(Stocks &a, Stocks &b): bool
~Groupby_S_()
Matrix:
Overloading Operators for Matrix manipulations
GNU Plot:
Plotting the CAAR plot
Class IWV
Important Data Members:
-IdxLevel, idxReturn, dates, idxMap
-StartDates, EndDates
Important Functions:
+IWV()
+populate_idxMap() ->int
+populate_idxLevel()-> int
+displayIWV( )-> void

Task Allocation

- Group module, stock module, ReadLoadData - Tejus and Eeshaan
- Bootstrapping, Overloading and Presentation - Syed and Naman
- Main Menu design, Plotting and Multithreading - Aidos and Xuanhong

Stock_indv Module

- The Stocks class is designed to hold all relevant information about a single stock.
- Its attributes are set as protected to prevent unintended alterations.
- Set and Get functions are included, allowing users to modify the attribute values in a controlled manner.



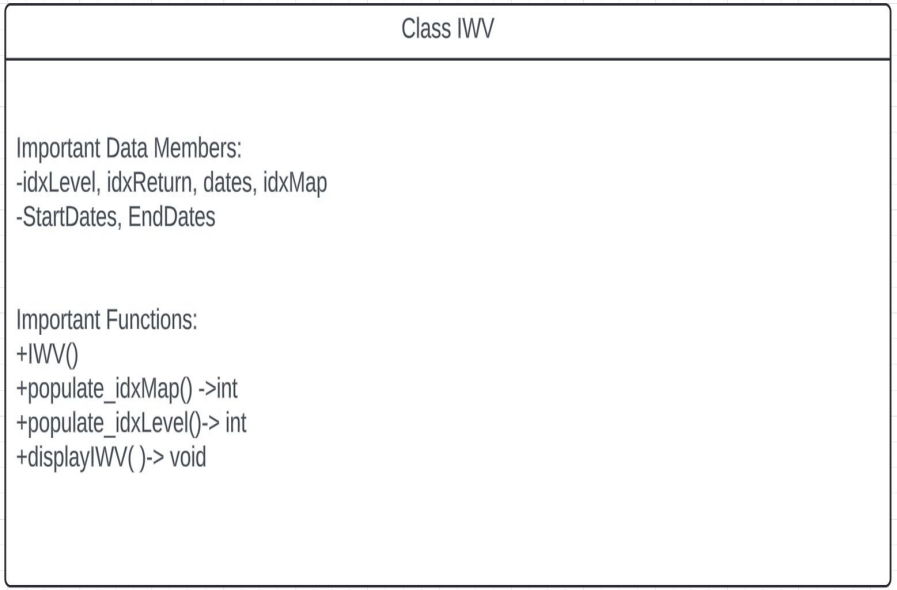
NYU

ReadLoadData Module

- The ReadLoadData files are designed for retrieving, storing, and managing data elements.
- Essential functions include FormatDate, LoadEarnings, FetchData, and write_to_stock.
- The FormatDate function (FormatDate(string date)) is utilized to reformat the date field, particularly converting the month's name to a numeric format (e.g., "Jan" to "01"), simplifying date iterations.
- LoadEarnings (LoadEarnings(map& data)) reads from a CSV file, temporarily stores the data in a stock object, and then adds this object to our stock map using the ticker as the key.
- FetchData (FetchData(map& data, string tickers)) is employed to acquire data from EOD sources and fill the stock map with this information.

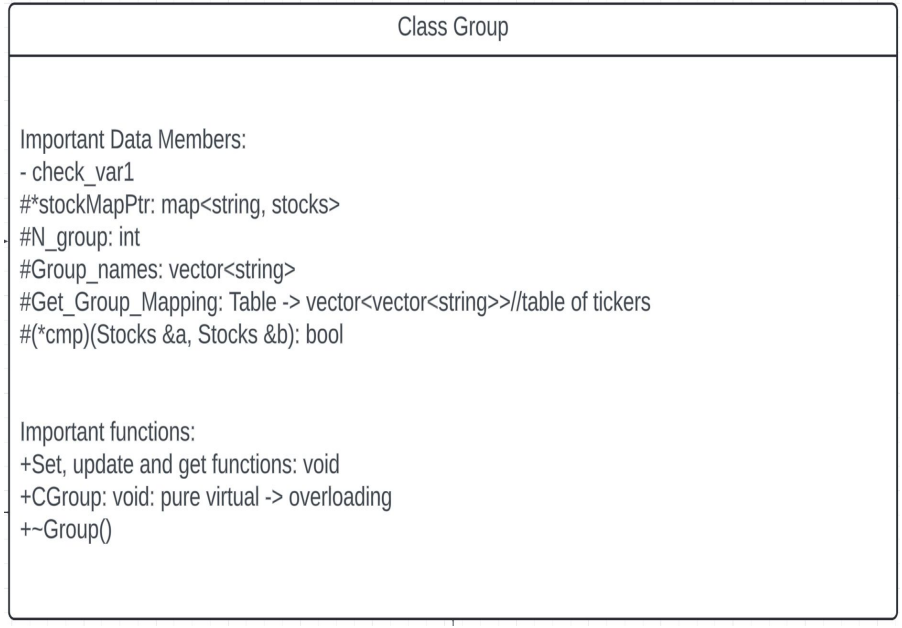
IWV Module

- IWV was a class designed to hold the data of the IWV benchmark
- It was used in previous versions of the code to compute the abnormal returns in the Bootstrap class
- In the current version of the code the MapPtr holds the IWV returns as one of its key value pairs



Group Module

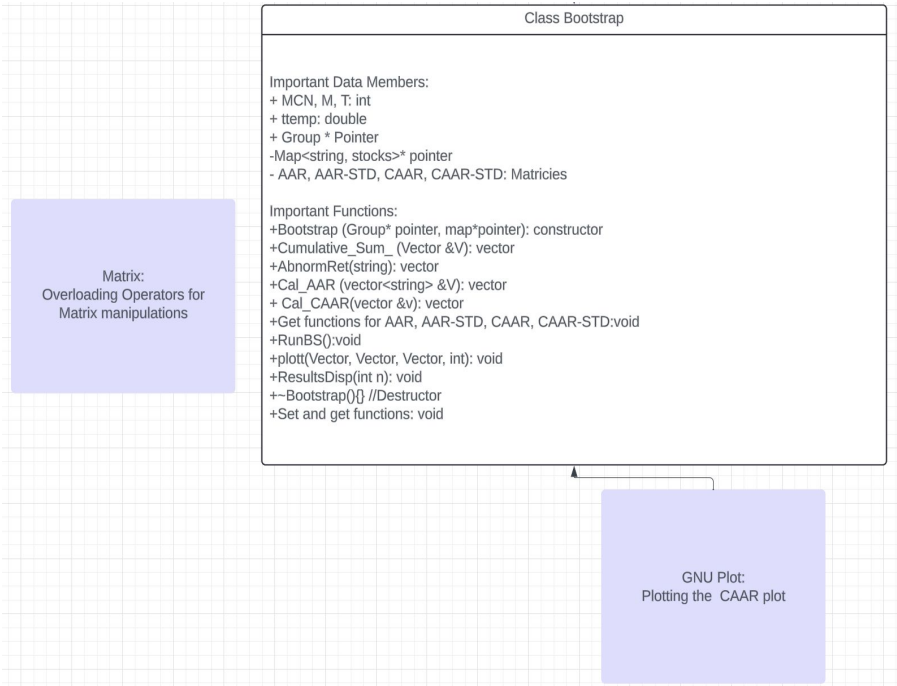
- Group is an abstract pure virtual class intended for generating a Matrix of N_groups from a stock map object
- Derived from Group, Groupby_Surprise is a class used to form a Matrix of stock groups based on the 'surprise percent' attribute in stock objects
- Additional classes can be extended from Group, allowing for the specification of different criteria to create various new groups



NYU

Bootstrap Module

- Enables bootstrapping for 3 different groups: Beat, Miss and Meet
- Creates samples through random selection of stock tickers from the Group class, we used shuffling for this
- Computes a matrix for each group over time, detailing average AAR, AAR Standard Deviation, average CAAR, and CAAR Standard Deviation
- In each iteration, it calculates vectors of 2*N size for AAR, CAAR, etc., for all given stocks, and incorporates these into the overall results



Matrix Module

- The class provides a set of mathematical operations for vectors and matrices, such as multiplication, addition, and dot product calculation.
- It includes functions for both vector-to-vector and scalar-to-vector operations.
- These have been used extensively in the Bootstrap class

Multithreading

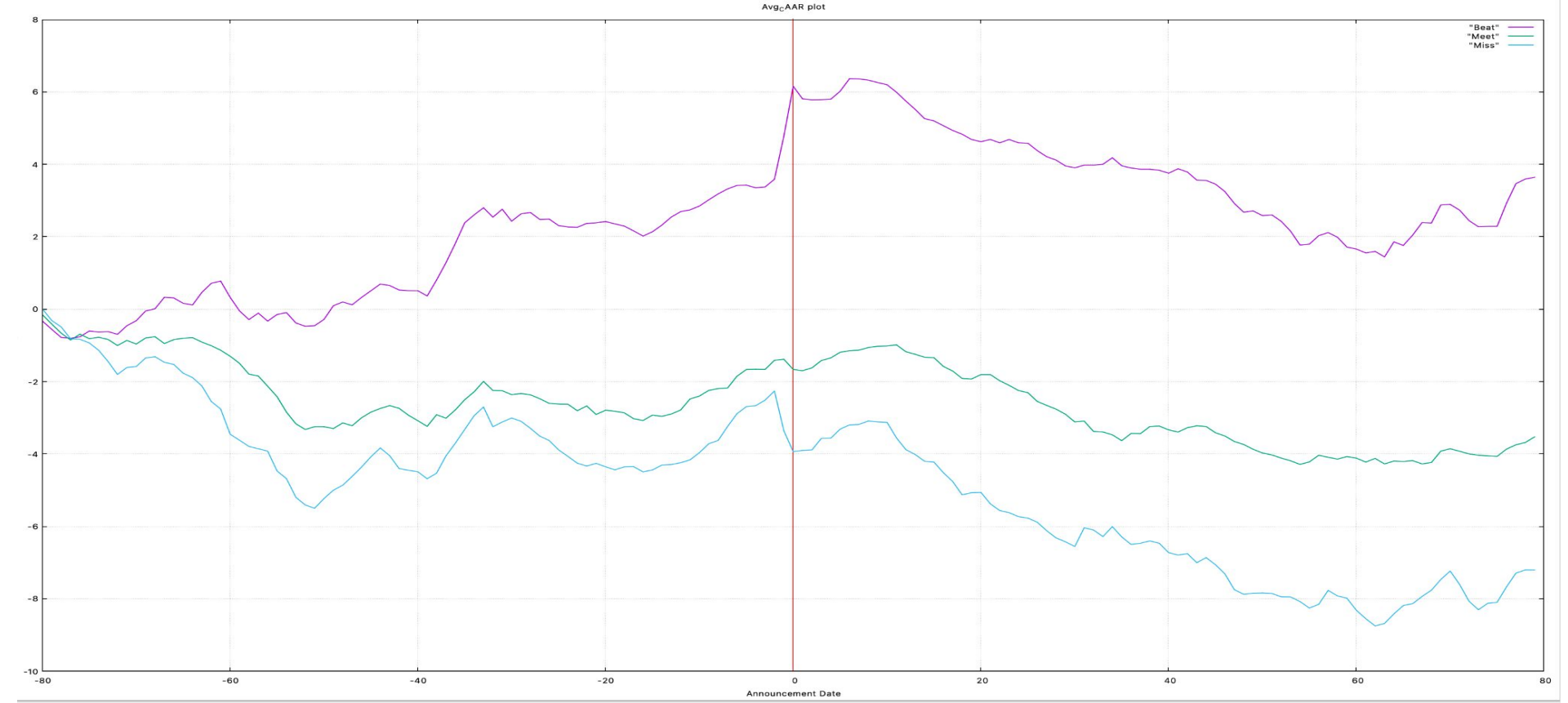
Put simply, multithreading involves running several threads of execution concurrently within a program.

- By allocating different threads to various functions or code segments, the total run time of the code is reduced since these threads operate in parallel, rather than one after the other.
- In our project, we applied multithreading to gather stock data. Rather than sequentially collecting data for stocks in the Russell 3000 index, we grouped them and executed multiple data retrieval calls at the same time, cutting down on the total time needed for our program to run
- In essence, each group has its own thread for data retrieval

Using gnuplot for plotting the graph

- We employ gnuplot_linux to generate graphs for each of the three stock groups
- The plot within the Bootstrap class is saved in PNG format, allowing us to access it later without rerunning the code

CAAR Graph using gnuplot



Conclusion

We conclude from our plot that the beat group and miss group diverges.

- The market, always looking ahead, values stocks based on multiples it considers appropriate, reflecting the stock's earnings forecasts
- Earnings announcements give market players concrete data, prompting a reassessment of valuations and expected future performance
- Before earnings announcements, the beat group's stocks are valued below what is deemed their fair value, leading to an increase in price as market participants adjust their expectations for improved future performance
- Conversely, stocks in the miss group are valued above their fair value prior to the announcement, resulting in a subsequent decrease in price

References

- Course materials and class slides - FRE 6883
- Capiński, M. J., Capiński, M., & Zastawniak, T. (2012). *Numerical methods in finance with C++*. Cambridge University Press
- Duffy, D. J. (2013). *Introduction to C++ for financial engineers: an object-oriented approach*. John Wiley & Sons
- An Introduction to Multithreading in C++20 - Anthony Williams - CppCon 2022 (https://www.youtube.com/watch?v=A7sVFJLJM-A)


NYU
Thank you