Using Genetic Algorithms to Select Best Performance Stock Portfolios

CAPSTONE

**Submitted in Partial Fulfillment
of the Requirements
for the Degree of**

Master of Science (Finance and Risk Engineering)

at the

**NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING**

by

Miao Lu

August 2019

Acknowledgement

I would like to extend my sincere gratitude to Prof. Song Tang for his expert guidance and assiduous supervision despite his busy schedule. I appreciate the patience with which he would tackle my doubts and the time he would spare from his busy schedule to meet and discuss with me and give me lots of valuable suggestions. His inputs to this project are invaluable. I would also like to thank the Finance and Risk Engineering department at NYU Tandon School of Engineering to give me the opportunity to work on this advanced topic for my capstone project.

Miao Lu
August, 2019

Abstract

This capstone project develops an algorithm to create an optimum portfolio which contains 10 stocks from S&P500 by using Genetic Algorithm. Firstly, the algorithm randomly selects 100 portfolios from S&P500 companies. Each portfolio is assigned a score based on their trading data, fundamental analysis and some technical indicators. Portfolio is then optimized by using the Genetic Algorithm which includes the implementation of reproduction, crossover and mutation. After meeting the stopping critera, the best portfolio has been obtained. The effectiveness of the best portfolio has been tested by measuring its performance in 6 months back test and 1-month probation test. It is found that the obtained portfolio can beat the market in the back test in 6 month averagely and also in the probation test.

Contents

Chapter 1

Introduction

1.1 Introduction

This capstone project first sets up market data retrieval using Unicorn data and parses the market data in Json format. It is followed by creating sqlite3 database with tables for storing the market data retrieved online. In this project, S&P500 stocks information, trading data and fundamental data of each stock in S&P500 and U.S. 10 Year Treasury rate are retrieved in different tables. Then the project implements fundamental class for stock fundamental data, trading class for each stocks daily trading information, stock class for each stock market data, and portfolio class. In portfolio class, we focus on implementing the fitness function for each portfolio. Then, implementation of Genetic Algorithm(GA) is created in the genetic algorithm class. In this class, the initial population is firstly created, then each population has reproduction process based on old populations fitness. Finally, the crossover and mutation for each generation are operated. After running GA, the best portfolio can be obtained. Then we do the back test to measure the performance of the best portfolio. If the result of back test is satisfying, the probation test will be performed. Otherwise, the fitness function will be changed. The flowchart for the main tasks in this project is shown in 1.1.

1.2 Statement Of The Problem

In this project, 10 years (from 1/1/2008 to 12/31/2018) of daily market data and fundamental data for each stock in S&P500 and SPY data are used for designing the fitness function. We run the GA to get the best portfolio. Then data from 1/1/2019 to 6/31/2019 is used for portfolio back testing. Data from 7/1/2019 to 7/31/2019 is used for portfolio probation testing.

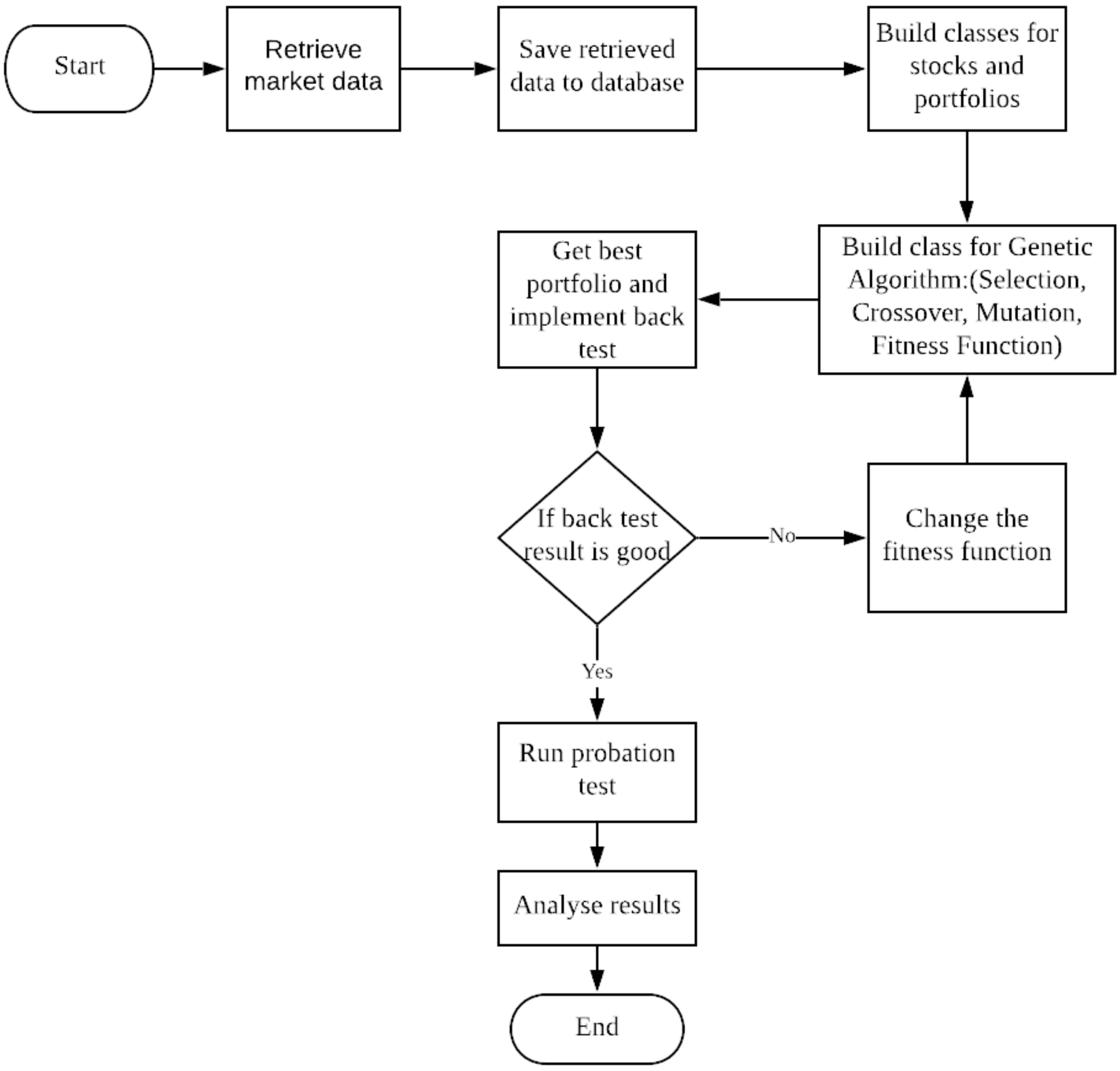

Start
Retrieve market data
Save retrieved data to database
Build classes for stocks and portfolios
Build class for Genetic Algorithm:(Selection, Crossover, Mutation, Fitness Function)
Get best portfolio and implement back test
If back test result is good
No
Change the fitness function
Yes
Run probation test
Analyse results
End

Figure 1.1: Main tasks of project

1.3 Objective

The objective of this project is to find the best performance portfolio. To obtain this goal, we first need to create suitable fitness function for this problem. The best portfolio is expected to beat SPY in back testing and then in probation testing.

1.4 Outline Of Report

The report is organized as follows.
Chapter 2 provides the background of portfolio selection and genetic algorithm.
Chapter 3 will show how we retrieve, parse data and then store data into database.
Chapter 4 will show the class design for this project.
Chapter 5 will describe the application of GA for portfolio selection in detail.
Chapter 6 will provides the fitness functions used and the corresponding back test and probation results.
Chapter 7 will conclude this report and give some future work.

Chapter 2

Background

2.1 Portfolio Selection

Portfolio selection is an active research topic these days. It aims to help investor maximize the return while minimize the risk of the portfolio at the same time. To find the optimum portfolio, investors need to consider the allocation of their wealth to different assets and also need to find the most suitable weights for those selected assets.
Modern Portfolio Theory(MPT) proposed by Markowitz (1952), attempts to achieve an expected return while minimize an objective function of portfolio risk. In this theory, portfolio mean acts as return and portfolio variance is treated as risk. Markowitz is awarded a Nobel prize for MPT because this theory was considered as an important advance in modern financial theory. However, according to Gyamerah (2014), this theory is challenged by criticisms because it is based on several unrealistic assumptions. For example, this theory assumes that markets are efficient and all investors are rational. This assumption is apparently unrealistic in real life market. Additionally, the weights of assets in the portfolio are required to be real numbers. However each asset has its minimum trading amounts in the real market. Hence, the implementation of MPT is difficult with quadratic or linear programming models according to Lin & Liu (2007). The model which considers realistic constraints such as minimum transaction lots is proposed by Bermúdez & Segura & Vercher (2011).
In this project, we assume the weights for the optimum portfolio are proportional to market capitalizations of stocks. In this case, to find the best performance stock portfolios, we only need to decide which stocks should be included in the portfolio in order to maximize the portfolios performance.

2.2 Genetic Algorithm

In the 1970s, John Holland first invented the Genetic Algorithm, which is considered as an significant contribution to scientific and engineering areas. GA is on the basis of Darwin's theory

of survival of the fittest according to Gyamerah (2014). It mimics natural selection process and biological genetics. Darwin's theory states that the fittest in the population is more likely to survive and has offsprings while the individual who does not "fit" well will eventually die out. This algorithm was popularized by Holland's student, David Goldberg, who use this method to deal with a difficult problem about controlling gas-pipeline transmission according to Haupt & Haupt (2003). After that time, lots of problems have been solved by using this algorithm.
GA has been widely applied to solve optimization problems. The advantages of GA include its wide acceptance in lots of disciplines, easy use and its global perspective according to Goldberg (1989). Additionally, Haupt & Haupt (2003) states that it can be applied to many optimization problems which cannot be solved by those traditional means.
There are many real-world applications for GA. In engineering field, Hughes (1990) stated that GA has been applied to design turbine blades of jet engines by General Electric. This application improves the efficiency for about 2 percent, which is a great contribution to this area. According to Goldberg (1994), GA has also been applied to communication network design. Similarly, the efficiency has been improved dramatically. Design time has been cut for about 97 percent with the help of GA. Another application, stated by Caldwell & Johnston (1991) is criminal identification, GA has been used to replace artiest' sketches. In this system, the witness is required to rate the faces generated from computer to find the face which looks like the criminal most. Based on the rating provided by witness, new faces will be generated until a likeness of the criminal's face is found. GAs also have lots of application in financial area, including design of credit score model, predicting bankruptcy and portfolio selection, which is the main task for this project.
The following is the main processes of implementing GA, according to Drake & Marks (1998):

1. First generation. The initial population is created randomly. The population is made up of chromosomes which consist of genes.

2. Assigning fitness. Individuals in population are assigned a fitness score by a fitness function to determine whether they fit well.

3. Selection. Each population has reproduction process. In this process, individual with higher fitness score has more chance to survive and offspring. This process is implemented by using the selection method. There are several available selection methods, including Roulette Wheel Selection and Elitism Selection.

4. Crossover. This step is analogous to biological crossover. The previous population first has pair-wise combination and then creating the new population by swapping part of their genes. One-point crossover, multi-point crossover and uniform crossover are commonly used methods.

5. Mutation. Each individual has a certain probability to change their gene value. This process aims to diversify the population.

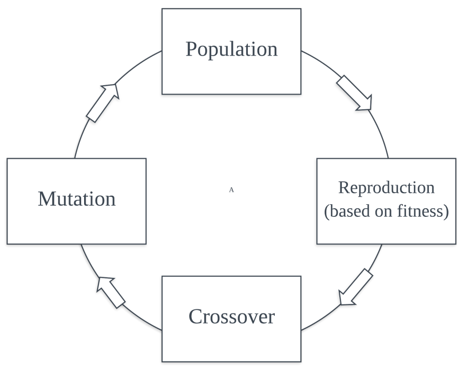


Figure 2.1: Genetic algorithm process

The Figure 2.1 shows evolution of generations.

Chapter 3

Data

3.1 Introduction

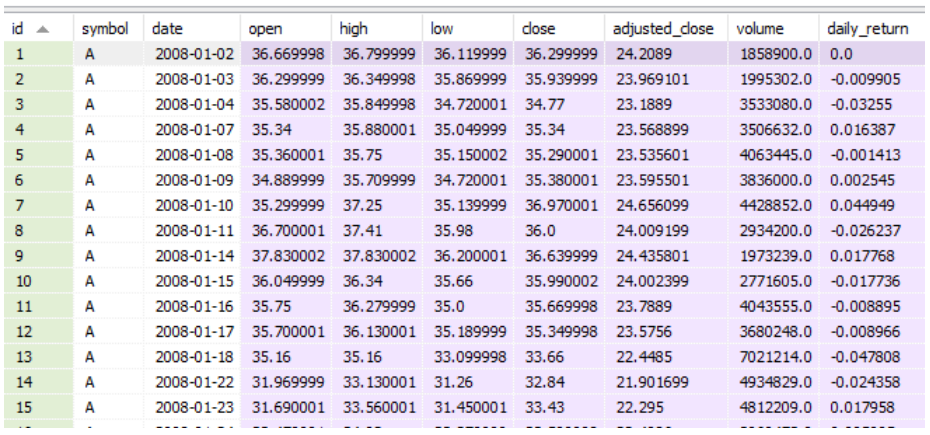
When we retrieve online data and save them into database, libcurl, json and sqlite3 are used. They are great software packages and applications and can be used widely in different interfaces. According to Salmanzadeh (2002), libcurl is a file transfer library. It can be used for all common transfer-related needs and different interfaces. In this project, we will use C++ for libcurl which helps us make simple HTTP requests to programmatically download data in web page. Json stands for JavaScript Object Notation, and is widely used to transfer data as it store information in an standardized manner. Hence, we can access data in human-readable way. (Droettboom, et al (2019)) In this project, data we aims to download is in Json format. Sqlite3, also a software library, is a compact free SQL database engine which can be used easily for creating and using a database.(Hombashi (2019)) In this project, the functions sqlite3.connect(), sqlite3_open(), sqlite3_close(), sqlite3_get_table() and sqlite3_exec() for creating, inserting and dropping tables are used.

3.2 Data Tables

1. "TickerName": stock daily trading data from eodhistoricaldata and is in Json format. Symbol, date, open, high, low, close, adjusted close and volume from 2008-01-01 to recent are included. Additionally, according to the retrieved adjusted close price, we calculated the daily return and save to the table. The table is shown in below Figure 3.1.

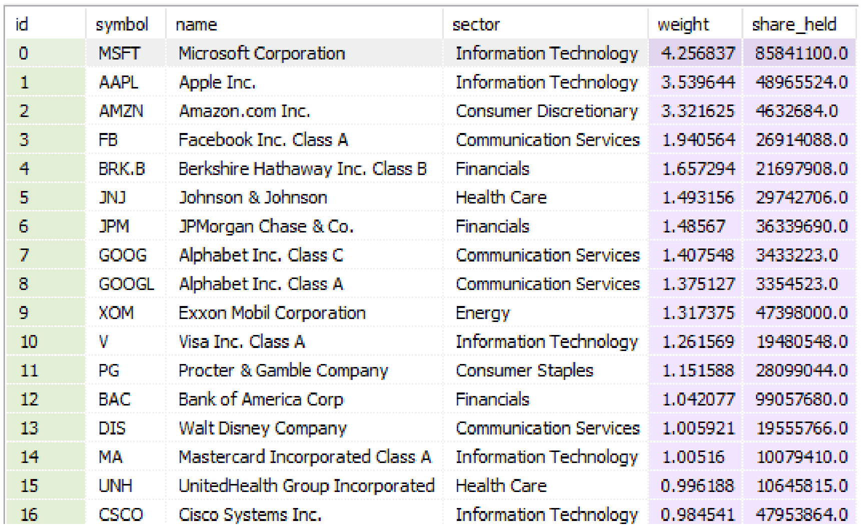
2. "S&P500": S&P500 component stocks information from EXCEL file which includes name, sector, symbol, weight and share held. The table is shown in below Figure 3.2.

3. "stock_fundamental": stock fundamental data from eodhistoricaldata and is in Json format. Symbol, 52 week high, 52 week low, 50 days MA, 200 days MA, PE Ratio, beta, dividend yield, profit margin, return on equity and return on asset are included in this table for



id	symbol	date	open	high	low	close	adjusted_close	volume	daily_return
1	A	2008-01-02	36.669998	36.799999	36.119999	36.299999	24.2089	1858900.0	0.0
2	A	2008-01-03	36.299999	36.349998	35.869999	35.939999	23.969101	1995302.0	-0.009905
3	A	2008-01-04	35.580002	35.849998	34.720001	34.77	23.1889	3533080.0	-0.03255
4	A	2008-01-07	35.34	35.880001	35.049999	35.34	23.568899	3506632.0	0.016387
5	A	2008-01-08	35.360001	35.75	35.150002	35.290001	23.535601	4063445.0	-0.001413
6	A	2008-01-09	34.889999	35.709999	34.720001	35.380001	23.595501	3836000.0	0.002545
7	A	2008-01-10	35.299999	37.25	35.139999	36.970001	24.656099	4428852.0	0.044949
8	A	2008-01-11	36.700001	37.41	35.98	36.0	24.009199	2934200.0	-0.026237
9	A	2008-01-14	37.830002	37.830002	36.200001	36.639999	24.435801	1973239.0	0.017768
10	A	2008-01-15	36.049999	36.34	35.66	35.990002	24.002399	2771605.0	-0.017736
11	A	2008-01-16	35.75	36.279999	35.0	35.669998	23.7889	4043555.0	-0.008895
12	A	2008-01-17	35.700001	36.130001	35.189999	35.349998	23.5756	3680248.0	-0.008966
13	A	2008-01-18	35.16	35.16	33.099998	33.66	22.4485	7021214.0	-0.047808
14	A	2008-01-22	31.969999	33.130001	31.26	32.84	21.901699	4934829.0	-0.024358
15	A	2008-01-23	31.690001	33.560001	31.450001	33.43	22.295	4812209.0	0.017958

Figure 3.1: Table for Stock A



id	symbol	name	sector	weight	share_held
0	MSFT	Microsoft Corporation	Information Technology	4.256837	85841100.0
1	AAPL	Apple Inc.	Information Technology	3.539644	48965524.0
2	AMZN	Amazon.com Inc.	Consumer Discretionary	3.321625	4632684.0
3	FB	Facebook Inc. Class A	Communication Services	1.940564	26914088.0
4	BRK.B	Berkshire Hathaway Inc. Class B	Financials	1.657294	21697908.0
5	JNJ	Johnson & Johnson	Health Care	1.493156	29742706.0
6	JPM	JPMorgan Chase & Co.	Financials	1.48567	36339690.0
7	GOOG	Alphabet Inc. Class C	Communication Services	1.407548	3433223.0
8	GOOGL	Alphabet Inc. Class A	Communication Services	1.375127	3354523.0
9	XOM	Exxon Mobil Corporation	Energy	1.317375	47398000.0
10	V	Visa Inc. Class A	Information Technology	1.261569	19480548.0
11	PG	Procter & Gamble Company	Consumer Staples	1.151588	28099044.0
12	BAC	Bank of America Corp	Financials	1.042077	99057680.0
13	DIS	Walt Disney Company	Communication Services	1.005921	19555766.0
14	MA	Mastercard Incorporated Class A	Information Technology	1.00516	10079410.0
15	UNH	UnitedHealth Group Incorporated	Health Care	0.996188	10645815.0
16	CSCO	Cisco Systems Inc.	Information Technology	0.984541	47953864.0

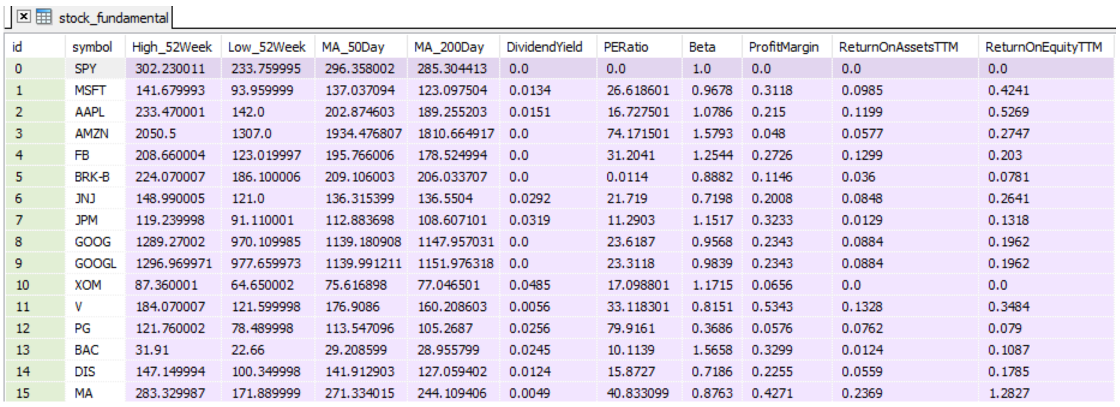
Figure 3.2: Table: S&P500

each stock and also SPY. The table is shown in below Figure 3.3.

4. "RiskFreeRate": treasury yield index market data from eodhistoricaldata and is in Json format. Symbol rf, date, open, high, low, close, adjusted close are included in this table. The table is shown in below Figure 6.1.

3.3 Data Retrieval

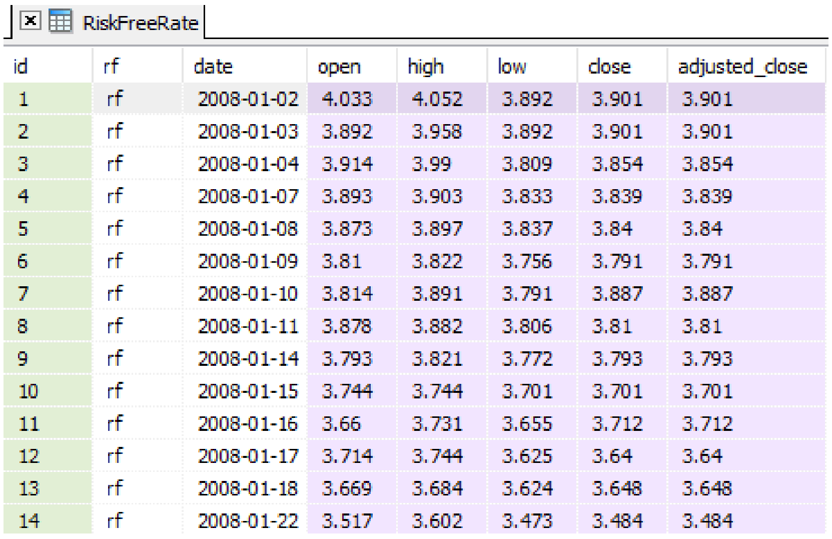
We first get S&P500 component stocks information from EXCEL file and then save it to our database with table named as NewSP500. Then we use tickers obtained from NewSP500 to retrieve the corresponding S&P500 stocks daily trading information and their fundamental data respectively. Function RetrieveMarketData() takes in ticker name, start date, end date,



stock_fundamental

id	symbol	High_52Week	Low_52Week	MA_50Day	MA_200Day	DividendYield	PERatio	Beta	ProfitMargin	ReturnOnAssetsTTM	ReturnOnEquityTTM
0	SPY	302.230011	233.759995	296.358002	285.304413	0.0	0.0	1.0	0.0	0.0	0.0
1	MSFT	141.679993	93.959999	137.037094	123.097504	0.0134	26.618601	0.9678	0.3118	0.0985	0.4241
2	AAPL	233.470001	142.0	202.874603	189.255203	0.0151	16.727501	1.0786	0.215	0.1199	0.5269
3	AMZN	2050.5	1307.0	1934.476807	1810.664917	0.0	74.171501	1.5793	0.048	0.0577	0.2747
4	FB	208.660004	123.019997	195.766006	178.524994	0.0	31.2041	1.2544	0.2726	0.1299	0.203
5	BRK-B	224.070007	186.100006	209.106003	206.033707	0.0	0.0114	0.8882	0.1146	0.036	0.0781
6	JNJ	148.990005	121.0	136.315399	136.5504	0.0292	21.719	0.7198	0.2008	0.0848	0.2641
7	JPM	119.239998	91.110001	112.883698	108.607101	0.0319	11.2903	1.1517	0.3233	0.0129	0.1318
8	GOOG	1289.27002	970.109985	1139.180908	1147.957031	0.0	23.6187	0.9568	0.2343	0.0884	0.1962
9	GOOGL	1296.969971	977.659973	1139.991211	1151.976318	0.0	23.3118	0.9839	0.2343	0.0884	0.1962
10	XOM	87.360001	64.650002	75.616898	77.046501	0.0485	17.098801	1.1715	0.0656	0.0	0.0
11	V	184.070007	121.599998	176.9086	160.208603	0.0056	33.118301	0.8151	0.5343	0.1328	0.3484
12	PG	121.760002	78.489998	113.547096	105.2687	0.0256	79.9161	0.3686	0.0576	0.0762	0.079
13	BAC	31.91	22.66	29.208599	28.955799	0.0245	10.1139	1.5658	0.3299	0.0124	0.1087
14	DIS	147.149994	100.349998	141.912903	127.059402	0.0124	15.8727	0.7186	0.2255	0.0559	0.1785
15	MA	283.329987	171.889999	271.334015	244.109406	0.0049	40.833099	0.8763	0.4271	0.2369	1.2827

Figure 3.3: Table: Stock fundamental data



RiskFreeRate

id	rf	date	open	high	low	close	adjusted_close
1	rf	2008-01-02	4.033	4.052	3.892	3.901	3.901
2	rf	2008-01-03	3.892	3.958	3.892	3.901	3.901
3	rf	2008-01-04	3.914	3.99	3.809	3.854	3.854
4	rf	2008-01-07	3.893	3.903	3.833	3.839	3.839
5	rf	2008-01-08	3.873	3.897	3.837	3.84	3.84
6	rf	2008-01-09	3.81	3.822	3.756	3.791	3.791
7	rf	2008-01-10	3.814	3.891	3.791	3.887	3.887
8	rf	2008-01-11	3.878	3.882	3.806	3.81	3.81
9	rf	2008-01-14	3.793	3.821	3.772	3.793	3.793
10	rf	2008-01-15	3.744	3.744	3.701	3.701	3.701
11	rf	2008-01-16	3.66	3.731	3.655	3.712	3.712
12	rf	2008-01-17	3.714	3.744	3.625	3.64	3.64
13	rf	2008-01-18	3.669	3.684	3.624	3.648	3.648
14	rf	2008-01-22	3.517	3.602	3.473	3.484	3.484

Figure 3.4: Table: RiskFreeRate

data url and api key to get market data for each stock starting from start date to end date. Those market data for each stock is parsed from Json format and then saved as an table with name being ticker of stock in database by using the function PopulateStockTable() which takes in ticker name, pointer of the database connection object and json object to parse and then save data to the database.

3.4 Database Design

Our database comprises of 508 tables with one to many relationship, as shown in Figure 3.5. Each ticker table with ”date” as primary key and ”symbol” as foreign key maps to table NewSP500. Additionally, each ticker table also has ”symbol” as foreign key which maps to table stock_fundamental. Finally, the table storing the data for 10 Years Treasury rate with date as primary key and also foreign key maps to each ticker table.

3.5 Data cleaning

Since the data obtained online has some missing values, so before running the program, we first need to fill those missing values. In this project, we have missing data in trading data for each stock in S&P500 and also the corresponding fundamental data. For daily market data, we first calculate the daily return by using the adjusted price for each stock, then the missing value of daily return is added with the value in the previous trading day. For the fundamental data, when retrieving those data, the null value is replaced with the large number 100000. We then assign them with suitable values when the fundamental data is in use.

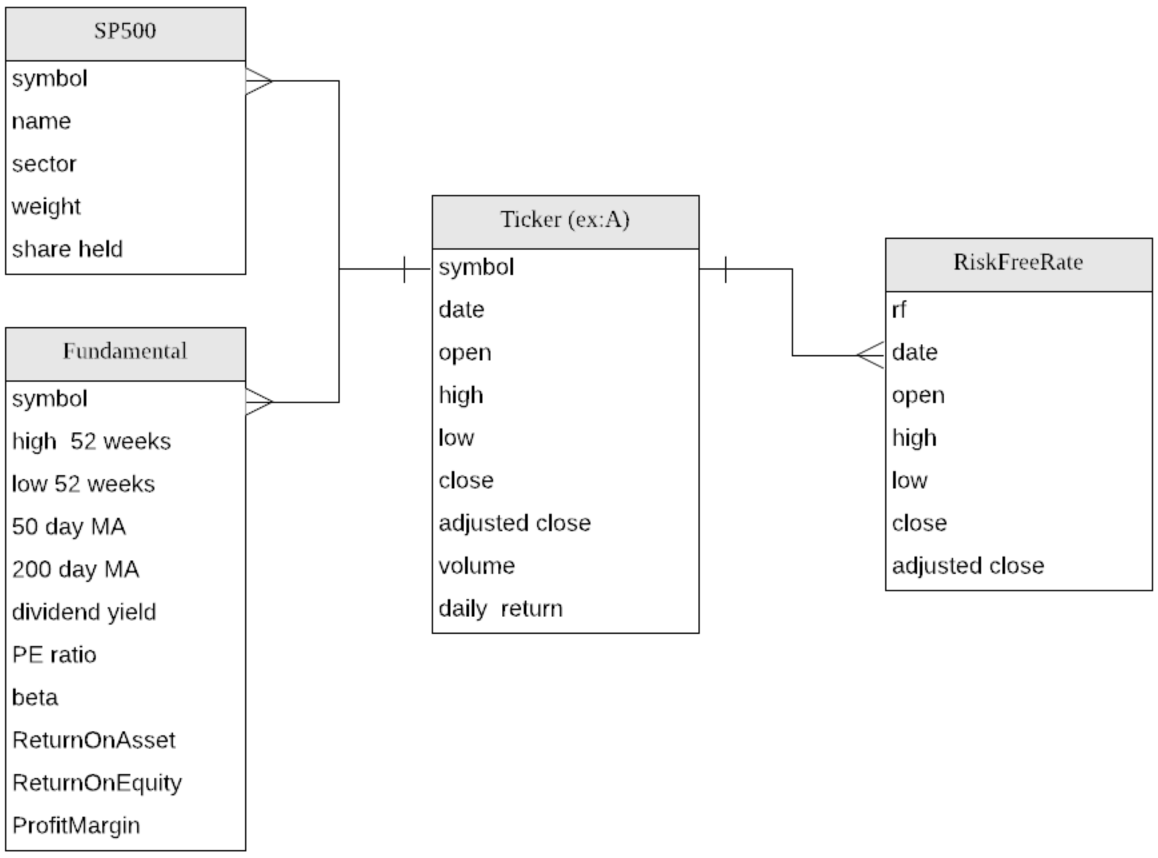


Figure 3.5: Database design

Chapter 4

Class Design

4.1 Introduction

In this project, we have 7 logics and 4 classes in total. Database logic is for data preparation which is described in previous chapter. The logics for fundamental data, trading data, stock and portfolio provide the framework for constructing candidate solutions for GA: portfolio consisting of 10 stocks and fitness function is created based on those logics as well. Genetic algorithm logic is used for implementing selection, crossover and mutation operators. The detail of GA will be provided in the next chapters. This chapter will focus on the design of 4 classes including Fundamental data class, Trade class, Stock class and Portfolio class.

4.2 Class Design

- In Trade class, daily market data including date, high price, low price, open price, close price, adjusted close price, daily volume and daily return of each stock is saved. This class will provide trading data for stock class.
- In Fundamental data class, the fundamental data for each stock are saved. Fundamental data will be included in stock class.
- Stock class mainly includes symbol, list of daily trading data, fundamental information, share held and weight proportional to capitalization of each stock in S&P500.
- Portfolio class mainly includes stocks information, performance measurement calculation and the fitness score calculation.

The figure for class design is show in Figure

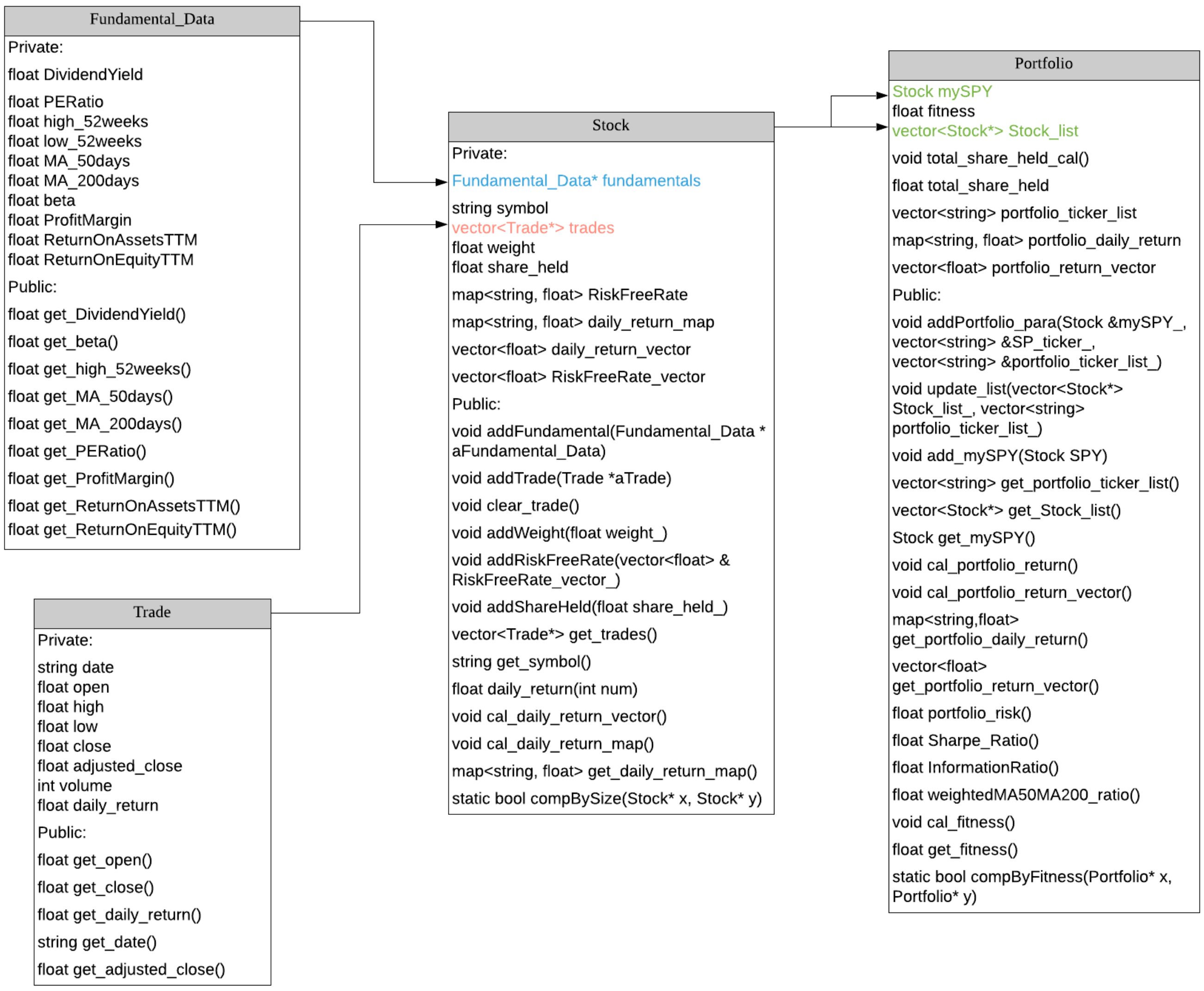

Fundamental_Data
Private:
float DividendYield
float PERatio
float high_52weeks
float low_52weeks
float MA_50days
float MA_200days
float beta
float ProfitMargin
float ReturnOnAssetsTTM
float ReturnOnEquityTTM
Public:
float get_DividendYield()
float get_beta()
float get_high_52weeks()
float get_MA_50days()
float get_MA_200days()
float get_PERatio()
float get_ProfitMargin()
float get_ReturnOnAssetsTTM()
float get_ReturnOnEquityTTM()
Trade
Private:
string date
float open
float high
float low
float close
float adjusted_close
int volume
float daily_return
Public:
float get_open()
float get_close()
float get_daily_return()
string get_date()
float get_adjusted_close()
Stock
Private:
Fundamental_Data* fundamentals
string symbol
vector<Trade*> trades
float weight
float share_held
map<string, float> RiskFreeRate
map<string, float> daily_return_map
vector<float> daily_return_vector
vector<float> RiskFreeRate_vector
Public:
void addFundamental(Fundamental_Data * aFundamental_Data)
void addTrade(Trade *aTrade)
void clear_trade()
void addWeight(float weight_)
void addRiskFreeRate(vector<float> & RiskFreeRate_vector_)
void addShareHeld(float share_held_)
vector<Trade*> get_trades()
string get_symbol()
float daily_return(int num)
void cal_daily_return_vector()
void cal_daily_return_map()
map<string, float> get_daily_return_map()
static bool compBySize(Stock* x, Stock* y)
Portfolio
Stock mySPY
float fitness
vector<Stock*> Stock_list
void total_share_held_cal()
float total_share_held
vector<string> portfolio_ticker_list
map<string, float> portfolio_daily_return
vector<float> portfolio_return_vector
Public:
void addPortfolio_para(Stock &mySPY_, vector<string> &SP_ticker_, vector<string> &portfolio_ticker_list_)
void update_list(vector<Stock*> Stock_list_, vector<string> portfolio_ticker_list_)
void add_mySPY(Stock SPY)
vector<string> get_portfolio_ticker_list()
vector<Stock*> get_Stock_list()
Stock get_mySPY()
void cal_portfolio_return()
void cal_portfolio_return_vector()
map<string,float> get_portfolio_daily_return()
vector<float> get_portfolio_return_vector()
float portfolio_risk()
float Sharpe_Ratio()
float InformationRatio()
float weightedMA50MA200_ratio()
void cal_fitness()
float get_fitness()
static bool compByFitness(Portfolio* x, Portfolio* y)

Figure 4.1: Database design

Chapter 5

Methodology

5.1 Introduction

In this chapter, the theories and concepts used in this project are described. We will focus on the terms and methods in GA and financial concepts. The financial concepts in sections 4.2 come from paper written by Gyamerah (2014). The detail of GA described in section 4.3 is from Goldberg (1989).

5.2 Stock Picking And Portfolio Composition Rules

5.2.1 Portfolio Performance Measurement

Portfolio Return: the gain or loss realized by investing portfolio. The calculation can be the weighted average return of the assets in the portfolio on a daily or long-term basis. The formula for calculating the portfolio return is:

$$r_p = \omega_1 * r_1 + \omega_2 * r_2 + \ldots + \omega_n * r_n,$$

where r_p is the portfolio return, n is the number of assets in this portfolio, r_i, $i = 1, ..., n$ is the asset return, and ω_i is the corresponding weight of this asset.
Portfolio Risk: portfolio volatility can be used to determine portfolio risk. The formula for it is:

$$\sigma_p = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - r_{mean})^2},$$

where r_i is the period return, n is the period for this portfolio, r_{mean} is the mean of the period return.
There are 2 components for portfolio risk:

- Systematic risk: this kind of risk is the inherent risk in the portfolio, which cannot be reduced by diversification.

- Unsystematic risk: this type of risk can be eliminated by diversification.

Beta: beta is a measurement of systematic risk. The stock beta is calculated as

$$\frac{r - r_f}{r_{market} - r_f},$$

where r is the return for the stock, r_f is the risk free rate and r_{market} stands for the market rate return.

We can calculated the portfolio beta by taking weighted average of the stocks' beta.

Treynor Measure: it is also called treynor reward to volatility model. The treynor ratio is used for those portfolios that are fully diversified. The higher the ratio, the better the portfolio. It is the portfolio return in excess of risk free rate to the systematic risk with the formula as:

$$\frac{r_p - r_f}{Beta}$$

Sharpe Ratio: it measures the portfolio's performance by adjusting for its risk. It is the excess return to the portfolio's total risk. The higher the sharpe ratio, the higher the return of the asset compensating for the risk taken. We can calculate sharpe ratio S by:

$$S = \frac{r_p - r_f}{\sigma_p}$$

Jensen Measure: we use J to denote jensen measure:

$$J = r_p - r_{CAPM}$$

where r_{CAPM} is the expected return calculated by Capital Asset Pricing Model(CAPM). CAPM describes the relationship between the excess return and the systematic risk. The formula for CAPM is:

$$r_{CAPM} = r_f + Beta * (r_{market} - r_f)$$

Information Ratio:

The information ratio measure the ratio of the portfolio return in excess of return of benchmark compared with its tracking error.

$$IR = \frac{E(r_p) - E(r_{benchmark})}{\sigma_{r_p} - r_{benchmark}}.$$

5.2.2 Technical Factors For Stock Picking And Portfolio Composition

Moving Average(MA): MA is an widely used indicator in technical analysis. Based on the past prices of assets, it can smooth out the asset price and remove the short-term price fluctuations.

There are two types of moving average that is commonly used. One is simple moving average(SMA) and the other is exponential moving average(EMA). SMA will give equal weights to the price of assets while the EMA will give more weights to the recent prices.

- 50DayMA: is a kind of SMA. It is calculated by taking the average of the asset's closing price for the past 50 days.
- 200DayMA: is similar to 50DayMA. It is calculated by taking the average of the asset's closing price for the past 200 days.

50DayMA and 200DayMA can be used for finding the golden cross. If 50DayMA crosses above 200DayMA, this cross is think of as a bullish signal and it can be treated as an golden cross.
52WeekHigh: is the highest price of the asset during the past 1 year, which can be used for determining an entry point. The investor may buy the asset when the asset price is higher than the 52 week high.
52WeekLow: is the lowest price of the asset during the past 1 year, which can be used for determining an exit point. The investor may sell the asset when the asset price is lower than the 52 week low.
Rate of Change(ROC): is a pure momentum oscillator that measures the percent change in price between the current price and the price several periods before.

$$ROC = 100 * \frac{Close - Close_{n\_periods\_ago}}{Close_{n\_periods\_ago}},$$

5.2.3 Fundamental Factors For Stock Picking And Portfolio Composition

Price to Earnig Ratio(PERatio):

$$PERatio = \frac{Price_{per\_share}}{Earning_{per\_share}}$$

The ratio is used for valuing companies and finding out whether they are overvalued or undervalued.
Profit Margin: that measures the companies' profitability.
Return on Asset (ReturnOnAssets): it is an indicator measuring how profitable a company is relative to its total assets. ROA gives an investor an idea how efficient a company generating earnings by using the companies assets.
Return on Equity (ReturnOnEquity): it is an indicator measuring financial performance of a company is relative to its total equity. ROE gives an investor an idea how efficient a company generating earnings by using the shareholder's equity.

5.3 Genetic Algorithm

5.3.1 Terms In GA

Population: each generation in genetic algorithm.
Individuals: population is made up of individuals or chromosomes. The two different elements in the population are individuals. Each individual in the population is also called chromosome,

which is in analogy to chromosome in biology. The fitness function is applied to each individual.
Gene: a chromosome in genetic algorithm is made up of a sequence of genes. A gene can be represented as string with arbitrary lengths.
Fitness: the fitness function is the function used for evaluating the fitness of each individual in population. The fitness function should be adequate for the specific problem.

5.3.2 Process Of GA

The genetic algorithm is used to find the solution (the individual that have the highest fitness score in simulation) which solves the optimization problem.
To find the solution, the chromosomes are first needed to be encoded according to the problem waited to be solved. Then the first generation is formed. After evaluating the fitness of the individuals in the parent generation, pool of individuals are selected, then crossover and mutation are applied to those selected individuals to form the new generation. The new generations are continuously produced until the maximum number of generations are formed or certain stopping criteria is met.

Initialization

The population size and individual structure is needed to decided firstly. Then the initial population consisting of solutions or individuals is created randomly or methodically.

Selection

The reproduction of the generation based on the selection methods. It is commonly that the new generation has the same number of individuals as the old generation. According to the theory of "survival of the fittest", individuals with high fitness score tend to have more chance to be selected to produce the new population.
There are many methods to implement the selection. Some of them will be discribed here.
Roulette Wheel Selection:
Roulette Wheel Selection is the most used selection which mimics a roulette wheel. In this method, each individual is a sector in the wheel, and the size of the sector is proportional to the fitness value of this individual, which is shown in Figure 5.1. In this diagram, the size of population is 10, and the chromosome with highest fitness score will has the highest chance to be selected. Similarly, the probability of selecting one individual is proportional to individual's fitness as well. The probability is calculated as:

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i},$$

where n is the population size, f_i is the fitness score of the ith individual and P_i is the chance that the ith individual is selected. In this case, the individual having higher fitness score would have more chance to be selected.

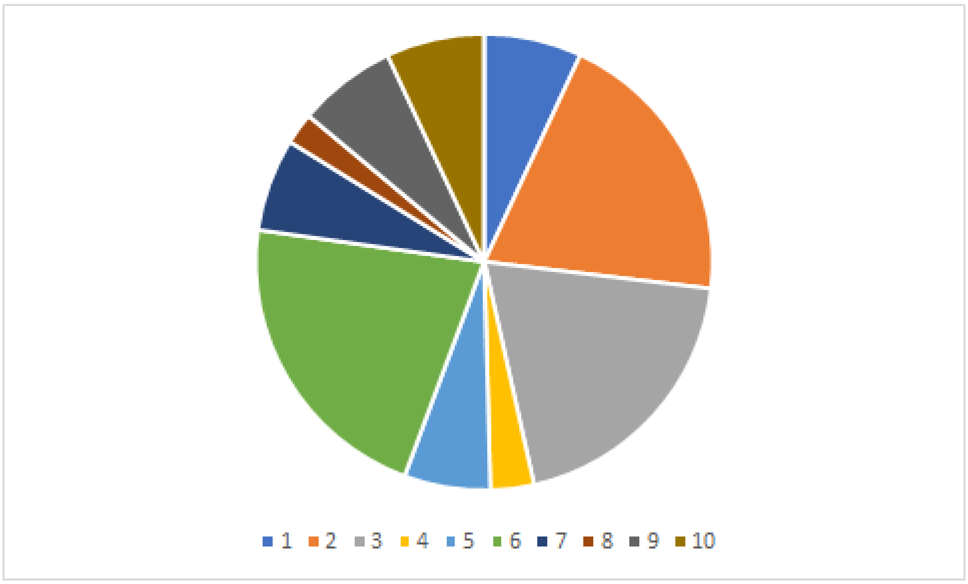


Figure 5.1: Roulette Wheel Selection Diagram

Elitist Selection:
In Elitist Selection method, a small number of individuals which have higher fitness score among the population are passed to the next generation without undergoing crossover and mutation operator. In this case, Elitism prevents the possible destruction of the individuals which "fit" well by avoiding crossover and mutation. The problem is that the number of elite should be limited.

Crossover

Crossover is an recombination process where offspring is produced by combining part of each parent's traits or characteristics. Typically, this process is done by swapping part of two parents' genes. The crossover operator will be applied with specified probability, that is, the crossover rate. Usually, this rate is high.
Some of the crossover methods are introduced below.
One-Point Crossover:
In this crossover type, a single point on parents' chromosomes is randomly selected. The tail part after that point is swapped between the two parents' chromosomes string. Then we get two new offsprings. One-Point cross diagram shown in 5.2
Two-Point Crossover:
This crossover operator will randomly select two different points on parents' chromosomes. Then the corresponding parts between two parents will be interchanged, which will result two new offsprings.

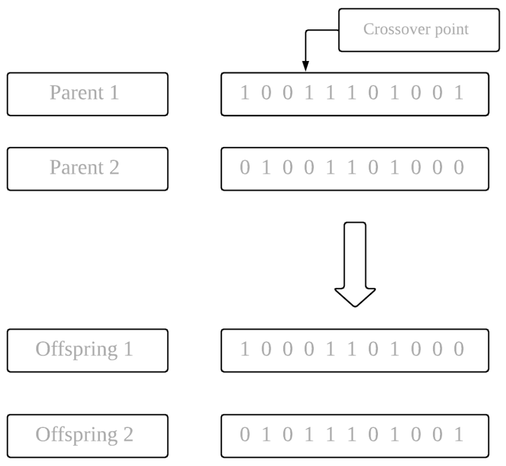


Figure 5.2: One-Point Crossover Diagram

Mutation

After crossover operator is performed, the mutation will be implemented. Mutation operator is used for introducing genetic diversity to the new generation. When mutation occurs, one or more gene value will be changed. The probability of the occurrence of mutation is usually set as small value, which is also reasonable in biology.
The mutation also have many types and here we only introduce uniform method:
Uniform:
First, genes for mutated are uniformly distributed and also chosen. Then, uniform mutation will replace the value of the gene with the value that is uniform selected from predefined intervals.

5.3.3 Applying GA To Find Best Performance Stock Portfolios

Parameters

In this project, each stock stands for a gene and 10 stocks are formed to a chromosome, that is a portfolio. Diagram 5.3 shows the example of one generation. Parameters in this project are defined as:

1. Population size: 100
2. Maximum generations: 1000
3. Mutation rate: 0.03
4. Stocks in portfolio: 10

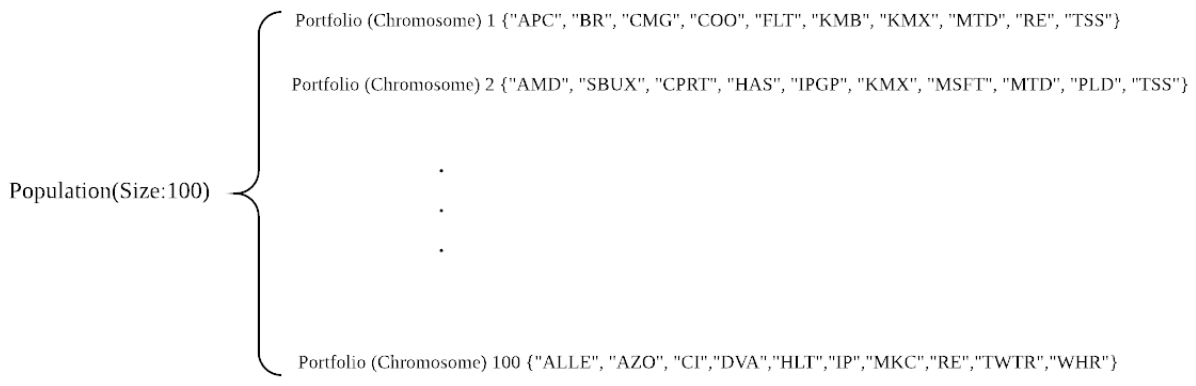

Portfolio (Chromosome) 1 {"APC", "BR", "CMG", "COO", "FLT", "KMB", "KMX", "MTD", "RE", "TSS"}
Portfolio (Chromosome) 2 {"AMD", "SBUX", "CPRT", "HAS", "IPGP", "KMX", "MSFT", "MTD", "PLD", "TSS"}
Population(Size:100)
Portfolio (Chromosome) 100 {"ALLE", "AZO", "CI","DVA","HLT","IP","MKC","RE","TWTR","WHR"}

Figure 5.3: Example of One Generation

Detail Of GA

1. To get the initial population, we randomly select 10 stocks from S&P500. Another way is to choose 10 stocks from each industry. In this project, we create the initial population randomly.

2. For deciding parents for creating children, the 20 percent individuals with highest fitness score will generate 2 parents for reproduction, the 20 percent individuals with lowest fitness score will produce 0 parent for reproduction and those left individuals in population will have 1 parent for creating child portfolios. Then 0.05 percent parents will be passed to next generation without being applied crossover and mutation operator.

3. In this project, One-Point crossover is applied. A random position is chosen, then parents' tail parts are interchanged.

4. We use Uniform type to apply mutation operator. For each portfolio, there is 0.03 probability to have a mutation. Once the mutation occurs, 2 random stocks in this portfolio will be replaced by another 2 random stocks from S&P500.

5. Lastly, creating new population will be stopped after 1000 generations.

We are left with the design of fitness function, which will be described in detail in next chapter.

Chapter 6

Fitness Function And Test Results

6.1 Introduction

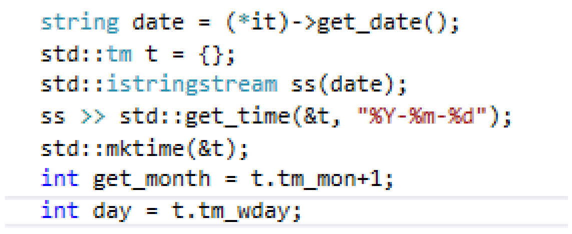
In this chapter, we first design the fitness function, then according the fitness function, we run the genetic algorithm described above to get the best portfolio. We will use the back test to test the performance of the best portfolio.

6.2 Fitness Function And Back Test Process

Fitness function is designed by using technical analysis, fundamental analysis and also portfolio performance measure formulas that described in the previous chapter.
After 1000 generations, the genetic algorithm gives us the best portfolio it found. Then the effectiveness of the best portfolio has been tested by measuring its performance in 6 months back test. The back test will start at Jan,1, 2019.

- Every Monday, \$1,000,000 is spent to purchase for the best portfolio as well as the same amount of SPY, then sell both of them on Friday. In this portfolio, stock is purchased according to their capitalization weight as well. This weekly purchase is repeated for a month.
- Then the portfolio return in excess of SPY return is calculated and if the best portfolio beats SPY, continue doing the step 1 to next month. Otherwise, GA will be run again to pick new best portfolios by including the previous month's data in training.
- If the best portfolio's performance is consistently worse than SPY, the fitness function will be changed.
- During the back test, the transaction and commission fee is ignored.

To run the program, we first need to convert the date in string form $Y - m - d$ to the weekday as we want to buy the best portfolio on Monday and sell it on Friday for each month. The class



```
string date = (*it)->get_date();
std::tm t = {};
std::istringstream ss(date);
ss >> std::get_time(&t, "%Y-%m-%d");
std::mktime(&t);
int get_month = t.tm_mon+1;
int day = t.tm_wday;
```

Figure 6.1: Code: convert date to weekday

istrinstream is used to find the corresponding month and weekday for each date. The code is shown in Figure 6.1:

6.3 Test Results

6.3.1 Fitness Function And Test Result

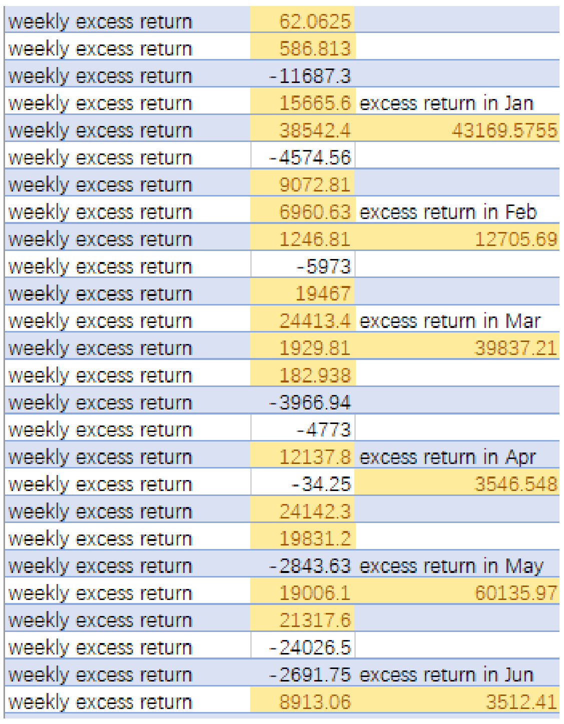
Fitness Function Design

$$Score = InformationRatio * \frac{50}{4} + (Weighted\frac{MA50}{MA200} - 1) * 10 + WeightedROA * 6$$

This fitness function combines information ratio, gold cross defined by 50 days moving average and 200 days moving average and return on asset of companies. $Weighted\frac{MA50}{MA200}$ is calculated by taking weighted average of each stock's $\frac{MA50}{MA200}$. Similarly, $WeightedROA$ is calculated by taking weighted average of each company's return on asset.
This project run GA 2 times by using this fitness function.

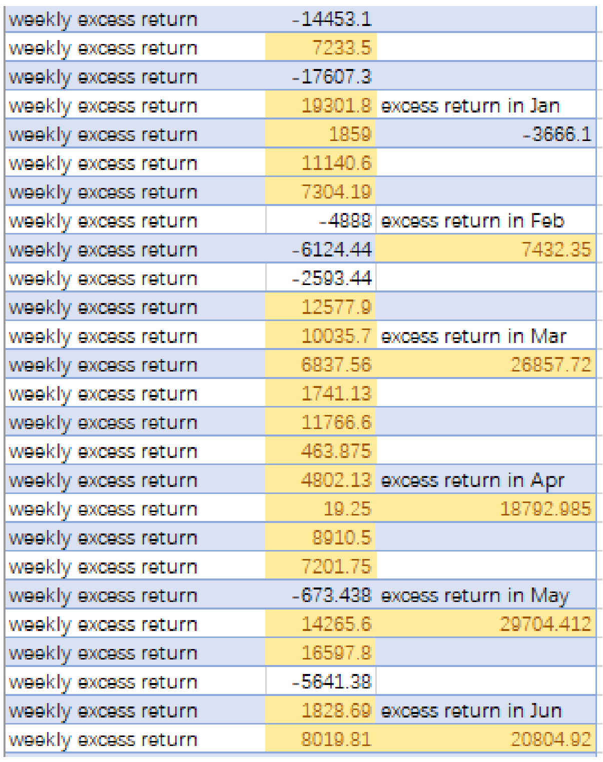
Back Test Result

For the first time, the best portfolio obtained is:
Portfolio: {"AMD", "CDNS", "FLT", "YUM", "IQV", "LW", "MA", "NOC", "V", "ZTS"}.
The back test result is shown in Figure 6.2.
For the second time, the best portfolio obtained is:
Portfolio: {"ALK", "AVGO", "HD", "LMT", "MA", "SBUX", "TXN", "URI", "V", "ZTS"}.
The back test result is shown in Figure 6.3.
Since in month 1, the portfolio return does not beat return of SPY, so the previous 1 month data is included as the training data. The new best portfolio obtained is:
Portfolio:{ "AWK","CBOE", "CPRT", "DG", "FLT", "HII", "KMX", "MA", "SBUX", "V"}.
The back test result is shown in Figure 6.4.



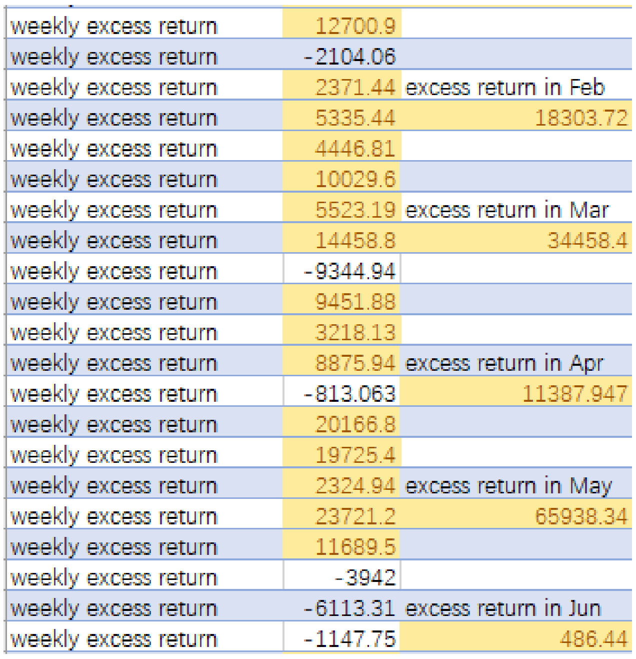
weekly excess return	62.0625	
weekly excess return	586.813	
weekly excess return	-11687.3	
weekly excess return	15665.6	excess return in Jan
weekly excess return	38542.4	43169.5755
weekly excess return	-4574.56	
weekly excess return	9072.81	
weekly excess return	6960.63	excess return in Feb
weekly excess return	1246.81	12705.69
weekly excess return	-5973	
weekly excess return	19467	
weekly excess return	24413.4	excess return in Mar
weekly excess return	1929.81	39837.21
weekly excess return	182.938	
weekly excess return	-3966.94	
weekly excess return	-4773	
weekly excess return	12137.8	excess return in Apr
weekly excess return	-34.25	3546.548
weekly excess return	24142.3	
weekly excess return	19831.2	
weekly excess return	-2843.63	excess return in May
weekly excess return	19006.1	60135.97
weekly excess return	21317.6	
weekly excess return	-24026.5	
weekly excess return	-2691.75	excess return in Jun
weekly excess return	8913.06	3512.41

Figure 6.2: Back Test Result for the first time



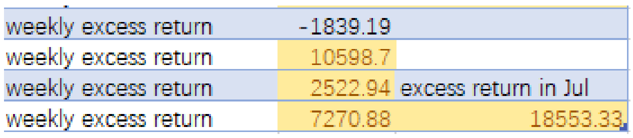
weekly excess return	-14453.1	
weekly excess return	7233.5	
weekly excess return	-17607.3	
weekly excess return	19301.8	excess return in Jan
weekly excess return	1859	-3666.1
weekly excess return	11140.6	
weekly excess return	7304.19	
weekly excess return	-4888	excess return in Feb
weekly excess return	-6124.44	7432.35
weekly excess return	-2593.44	
weekly excess return	12577.9	
weekly excess return	10035.7	excess return in Mar
weekly excess return	6837.56	26857.72
weekly excess return	1741.13	
weekly excess return	11766.6	
weekly excess return	463.875	
weekly excess return	4802.13	excess return in Apr
weekly excess return	19.25	18792.985
weekly excess return	8910.5	
weekly excess return	7201.75	
weekly excess return	-673.438	excess return in May
weekly excess return	14265.6	29704.412
weekly excess return	16597.8	
weekly excess return	-5641.38	
weekly excess return	1828.69	excess return in Jun
weekly excess return	8019.81	20804.92

Figure 6.3: Back Test Result for the second time



weekly excess return	12700.9	
weekly excess return	-2104.06	
weekly excess return	2371.44	excess return in Feb
weekly excess return	5335.44	18303.72
weekly excess return	4446.81	
weekly excess return	10029.6	
weekly excess return	5523.19	excess return in Mar
weekly excess return	14458.8	34458.4
weekly excess return	-9344.94	
weekly excess return	9451.88	
weekly excess return	3218.13	
weekly excess return	8875.94	excess return in Apr
weekly excess return	-813.063	11387.947
weekly excess return	20166.8	
weekly excess return	19725.4	
weekly excess return	2324.94	excess return in May
weekly excess return	23721.2	65938.34
weekly excess return	11689.5	
weekly excess return	-3942	
weekly excess return	-6113.31	excess return in Jun
weekly excess return	-1147.75	486.44

Figure 6.4: Back Test Result for the second time



weekly excess return	-1839.19	
weekly excess return	10598.7	
weekly excess return	2522.94	excess return in Jul
weekly excess return	7270.88	18553.33

Figure 6.5: Probation Test Result for the first time

6.4 Probation Test Result

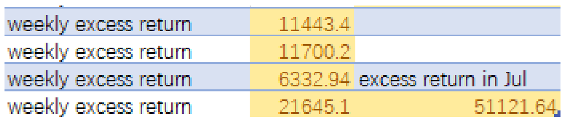
The process for probation test is same as the back test described above. We do the probation test from July,1, 2019 to July, 30, 2019.

6.4.1 First Running Probation Test

For the first running, we have the best portfolio beat SPY so we can do the probation test. Portfolio: {"AMD", "CDNS", "FLT", "YUM", "IQV", "LW", "MA", "NOC", "V", "ZTS"}. The result is shown in Figure 6.5.

6.4.2 Second Running Probation Test

In the second running, we have the back test beat SPY in month 2 to 6. So probation test is then performed.
Portfolio:{ "AWK","CBOE", "CPRT", "DG", "FLT", "HII", "KMX", "MA", "SBUX", "V"}. The result is shown in Figure 6.6.



weekly excess return	11443.4	
weekly excess return	11700.2	
weekly excess return	6332.94	excess return in Jul
weekly excess return	21645.1	51121.64

Figure 6.6: Probation Test Result for the second time

Chapter 7

Conclusion And Future Work

7.1 Conclusion

This capstone project applies GA to find the best performance portfolio by using C++. We finally applied fitness function which combines information ratio, MA50Day, MA200Day and ROA on portfolios. We run the GA two times and get two best portfolio for each time. The results we obtained showed that the best portfolios obtained can at least beat SPY in continuous 5 months averagely and averagely beat SPY in 6 months in back test. Both portfolios obtained can beat SPY in July averagely.

7.2 Future Work

The future work is required in this capstone project as the existence of the limitations and space for improvement.

1. The fitness function applied to portfolio should be improved in the future as the back test result shows that we cannot beat the market for 6 months continuously.

2. This capstone project did not consider shorting stocks which is unrealistic. Hence it is better to including shorting stocks in the future work.

3. In the part for implementing genetic algorithm, the random selection of stocks are used for generating the first generation. We can try to form the first generation by select one stock from each industry in the future's work. Additionally, the best portfolio obtained is highly determined by the initial population we selected. Hence, we may only get the local optimum portfolio. Moreover, after many generations, chromosomes in populations tends to be identical. These problems should be considered further.

4. When we do the back test and probation test, it is assumed that the transaction and commission fee is 0. We can add those fee to have a more realistic test in future.

Bibliography

J. D. Bermúdez, J.V. Segura and E. Vercher: *A multi-objective genetic algorithm for cardinality constrained fuzzy portfolio selection*, Fuzzy Sets and Systems 188 (2012)16C26.

C. Caldwell and V. S. Johnston: *Tracking criminal suspect through "face-space" with a genetic algorithm*, Belew and Booker (eds.), pp. 416-421.

A. E. Drake and R. E. Marks: *Genetic Algorithms in Economics and Finance: Forecasting Stock Market Prices and Foreign Exchange: A Review.*

M. Droettboom, et al: *Understanding JSON Schema.*

C. C. Lin and Y.T. Liu: *Genetic algorithms for portfolio selection problems with minimum transaction lots*, European Journal of Operational Research 185 (2008) 393C404.

D.E. Goldberg: *Genetic algorithms in search, optimization, and machine learning*, New York: Addison-Wesley.

D.E. Goldberg: *Genetic and Evolutionary Algorithms come of age*, Communications of the ACM, Vol. 37, No. 3, March, pp. 113-119.

A.S. Gyamerah: *Heuristic Crossover For Porfolio Selection.*

S.E. Haupt and R.L. Haupt: *Genetic Algorithms and Their Applications In Environmental Science*

T. Hombashi: *SimpleSQLite Documentation.*

M. Hughes: *Improving products and processes- Nature's way*,Industrial Management & Data Systems, 6, pp. 22-25.

H. Markowitz: *Portfolio selection.*The Journal of Finance, 7(1): 77-91.

R. Salmanzadeh: *Using libcurl in Visual Studio.*

Appendix A

C++ code

A.1 Data Retrieving and saving algorithm

```
#pragma once
#ifndef Database_h
#define Database_h
#include "Stock.h"
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include "curl_easy.h"
#include "curl_form.h"
#include "curl_ios.h"
#include "curl_exception.h"
#include "json/json.h"
#include <sqlite3.h>
#include <cstdlib>
using namespace std;
int DisplayFundamentalData(const char *sql_select, sqlite3 *db, Stock* myStock, int num, vector<float> & dividend_vector,
vector<float> &PE_vector,vector<float> &profit_vector, vector<float> &ROA_vector, vector<float> &ROE_vector)
{
string PE;
float High_52Week = 0, Low_52Week = 0, MA_50Day = 0, MA_200Day = 0, DividendYield = 0, PERatio = 0, beta = 0,
ProfitMargin=0, ReturnOnAssetsTTM=0, ReturnOnEquityTTM=0;
int rc = 0;
char *error = NULL;
// Display MyTable
//cout << "Retrieving values in a table ..." << endl;
char **results = NULL;
int rows, columns;
// A result table is memory data structure created by the sqlite3_get_table() interface.
// A result table records the complete query results from one or more queries.
sqlite3_get_table(db, sql_select, &results, &rows, &columns, &error);
if (rc)
{
cerr << "Error executing SQLite3 query: " << sqlite3_errmsg(db) << endl << endl;
sqlite3_free(error);
system("pause");
return -1;
}
// Display Table
int rowCtr = num + 1;
for (int colCtr = 0; colCtr < columns; ++colCtr)
{
// Determine Cell Position
int cellPosition = (rowCtr * columns) + colCtr;
if (colCtr == 2)
High_52Week = atof(results[cellPosition]);
else if (colCtr == 3)
Low_52Week = atof(results[cellPosition]);
else if (colCtr == 4)
MA_50Day = atof(results[cellPosition]);
else if (colCtr == 5)
MA_200Day = atof(results[cellPosition]);
else if (colCtr == 6)
DividendYield = atof(results[cellPosition]);
else if (colCtr == 7)
PERatio = atof(results[cellPosition]);
else if (colCtr == 8)
beta = atof(results[cellPosition]);
else if (colCtr == 9)
ProfitMargin = atof(results[cellPosition]);
else if (colCtr == 10)
ReturnOnAssetsTTM = atof(results[cellPosition]);
else if (colCtr == 11)
ReturnOnEquityTTM = atof(results[cellPosition]);
//ProfitMargin,ReturnOnAssetsTTM,ReturnOnEquityTTM
}
dividend_vector.push_back(DividendYield);
```

```
if (PERatio == 100000)
PERatio = -1;
if (MA_50Day == 0)
MA_50Day = 1;
if (High_52Week == 0)
High_52Week = 1000;
if (MA_200Day == 0)
MA_200Day = 1;
if (ProfitMargin == 100000)
ProfitMargin = 0;
if (ReturnOnAssetsTTM == 100000)
ReturnOnAssetsTTM = 0;
if (ReturnOnEquityTTM == 100000)
ReturnOnEquityTTM = 0;
PE_vector.push_back(PERatio);
profit_vector.push_back(ProfitMargin);
ROA_vector.push_back(ReturnOnAssetsTTM);
ROE_vector.push_back(ReturnOnEquityTTM);
Fundamental_Data* aFundamental_Data=new Fundamental_Data(High_52Week, Low_52Week, MA_50Day, MA_200Day, DividendYield, PERatio,
beta, ProfitMargin, ReturnOnAssetsTTM, ReturnOnEquityTTM);
myStock->addFundamental(aFundamental_Data);
//cout << (myStock->Get_fundamentals())->get_MA_50days() << endl;
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}
int DisplaySP500Ticker(const char *sql_select, sqlite3 *db, vector<string>& ticker_list)
{
string ticker;
int rc = 0;
char *error = NULL;
// Display MyTable
//cout << "Retrieving values in a table ..." << endl;
char **results = NULL;
int rows, columns;
// A result table is memory data structure created by the sqlite3_get_table() interface.
// A result table records the complete query results from one or more queries.
sqlite3_get_table(db, sql_select, &results, &rows, &columns, &error);
if (rc)
{
cerr << "Error executing SQLite3 query: " << sqlite3_errmsg(db) << endl << endl;
sqlite3_free(error);
system("pause");
return -1;
}
// Display Table
for (int rowCtr = 1; rowCtr <= rows; ++rowCtr)
{
// Determine Cell Position
int cellPosition = (rowCtr * columns) + 1;
ticker = (results[cellPosition]);
ticker_list.push_back(ticker);

// End Line
//cout << endl;

}
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}
int DisplayNewSP500Ticker(const char *sql_select, sqlite3 *db, vector<string>& ticker_list, vector<float>&weight_list,
vector<float> &share_held_list, vector<map<string, vector<string>>> &sector_category)
{
string ticker, sector;
float weight = 0;
float share_held = 0;
int rc = 0;
char *error = NULL;
// Display MyTable
//cout << "Retrieving values in a table ..." << endl;
char **results = NULL;
int rows, columns;
// A result table is memory data structure created by the sqlite3_get_table() interface.
// A result table records the complete query results from one or more queries.
sqlite3_get_table(db, sql_select, &results, &rows, &columns, &error);
if (rc)
{
cerr << "Error executing SQLite3 query: " << sqlite3_errmsg(db) << endl << endl;
sqlite3_free(error);
system("pause");
return -1;
}
// Display Table
for (int rowCtr = 1; rowCtr <= rows; ++rowCtr)
{
// Determine Cell Position
//int cellPosition = (rowCtr * columns) + 1;
//ticker = (results[cellPosition]);
//ticker_list.push_back(ticker);
for (int colCtr = 0; colCtr < columns; ++colCtr)
{
// Determine Cell Position
int cellPosition = (rowCtr * columns) + colCtr;
if (colCtr == 1)
ticker = results[cellPosition];
else if (colCtr == 3)
sector = results[cellPosition];
else if (colCtr == 4)
weight = atof(results[cellPosition]);
else if (colCtr == 5)
share_held = atof(results[cellPosition]);

}
if (ticker == "BRK.B")
ticker = "BRK_B";
if (ticker == "BF.B")
ticker = "BF_B";
if (ticker == "FOXA")
continue;
if (ticker == "LIN")
continue;
if (ticker == "DOW")
continue;
if (ticker == "CTVA")
continue;
if (ticker == "CPRI")
continue;
if (ticker == "AMCR")
continue;
ticker_list.push_back(ticker);
weight_list.push_back(weight);
share_held_list.push_back(share_held);
if (sector == "Communication Services")
sector_category[0]["Communication_Services"].push_back(ticker);
else if (sector == "Consumer Discretionary")
sector_category[1]["Consumer_Discretionary"].push_back(ticker);
else if (sector == "Consumer Staples")
sector_category[2]["Consumer_Staples"].push_back(ticker);
else if (sector == "Energy")
sector_category[3]["Energy"].push_back(ticker);
else if (sector == "Financials")
sector_category[4]["Financials"].push_back(ticker);
else if (sector == "Health Care")
sector_category[5]["Health_Care"].push_back(ticker);
else if (sector == "Industrials")
sector_category[6]["Industrials"].push_back(ticker);
else if (sector == "Information Technology")
sector_category[7]["Information_Technology"].push_back(ticker);
else if (sector == "Materials")
sector_category[8]["Materials"].push_back(ticker);
else if (sector == "Real Estate")
sector_category[9]["Real_Estate"].push_back(ticker);
else if (sector == "Utilities")
sector_category[10]["Utilities"].push_back(ticker);
else if (sector == "Unassigned")
sector_category[11]["Unassigned"].push_back(ticker);
}
//for (vector<vector<string>>::iterator it = sector_category.begin(); it != sector_category.end(); ++it)
// cout << *it << endl;
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}
int DisplayRiskFreeRateData(const char *sql_select, sqlite3 *db, map<string, float> &RiskFreeRate,
vector<float>& RiskFreeRate_vector)
{
string date;
float adjusted_close = 0;
int rc = 0;
char *error = NULL;
// Display MyTable
//cout << "Retrieving values in a table ..." << endl;
char **results = NULL;
int rows, columns;
// A result table is memory data structure created by the sqlite3_get_table() interface.
// A result table records the complete query results from one or more queries.
sqlite3_get_table(db, sql_select, &results, &rows, &columns, &error);
if (rc)
{
cerr << "Error executing SQLite3 query: " << sqlite3_errmsg(db) << endl << endl;
sqlite3_free(error);
system("pause");
return -1;
}
// Display Table
for (int rowCtr = 0; rowCtr <= rows; ++rowCtr)
{
for (int colCtr = 0; colCtr < columns; ++colCtr)
{
// Determine Cell Position
int cellPosition = (rowCtr * columns) + colCtr;
if (colCtr == 2)
date = (results[cellPosition]);
else if (colCtr == 7)
adjusted_close = atof(results[cellPosition]);
}
RiskFreeRate[date] = adjusted_close;
RiskFreeRate_vector.push_back(pow(1+adjusted_close*0.01,1/252.0)-1);
// End Line
//cout << endl;
}
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}
int DisplayTradeData(const char *sql_select, sqlite3 *db, Stock* myStock)
{
int volume = 0;
string date;
```

```
float open = 0, high = 0, low = 0, close = 0, adjusted_close = 0, daily_return = 0;
int rc = 0;
vector<Trade> trades;
char *error = NULL;
// Display MyTable
//cout << "Retrieving values in a table ..." << endl;
char **results = NULL;
int rows, columns;
// A result table is memory data structure created by the sqlite3_get_table() interface.
// A result table records the complete query results from one or more queries.
sqlite3_get_table(db, sql_select, &results, &rows, &columns, &error);
if (rc)
{
cerr << "Error executing SQLite3 query: " << sqlite3_errmsg(db) << endl << endl;
sqlite3_free(error);
system("pause");
return -1;
}
// Display Table
for (int rowCtr = 1; rowCtr <= rows; ++rowCtr)
{
for (int colCtr = 0; colCtr < columns; ++colCtr)
{
// Determine Cell Position
int cellPosition = (rowCtr * columns) + colCtr;
if (colCtr == 2)
{
date = (results[cellPosition]);
}
else if (colCtr == 3)
{
open = atof(results[cellPosition]);
}
else if (colCtr == 4)
high = atof(results[cellPosition]);
else if (colCtr == 5)
low = atof(results[cellPosition]);
else if (colCtr == 6)
close = atof(results[cellPosition]);
else if (colCtr == 7)
adjusted_close = atof(results[cellPosition]);
else if (colCtr == 8)
volume = atoi(results[cellPosition]);
else if (colCtr == 9)
daily_return = atof(results[cellPosition]);

}
// End Line
//cout << endl;
Trade* aTrade=new Trade(date, open, high, low, close, adjusted_close, volume, daily_return);
//cout << *aTrade << endl;
myStock->addTrade(aTrade);
}
//myStock.addTrade(trades);
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}
int PopulateFundamentalDataTable(const Json::Value & root, string symbol, sqlite3 *db, int num)
{
string PE;
string date;
float High_52Week = 0, Low_52Week = 0, MA_50Day = 0, MA_200Day = 0, DividendYield = 0, PERatio = 0, ProfitMargin=0, ReturnOnAssetsTTM=0, ReturnOnEquityTTM=0;
float beta = 0;
int volume;
Stock myStock(symbol);
int count = num;
for (Json::Value::const_iterator itr = root.begin(); itr != root.end(); itr++)
{
// cout << *itr << endl;
if (itr.key().asString() == "Technicals")
for (Json::Value::const_iterator inner = (*itr).begin(); inner != (*itr).end(); inner++)
{
//cout << inner.key() << ": " << *inner << endl;
if (inner.key().asString() == "52WeekHigh")
{
if ((inner->asString()).empty())
{
High_52Week = 100000;
}
else
High_52Week = (float)atof((inner->asCString()));
}
else if (inner.key().asString() == "52WeekLow")
{
if ((inner->asString()).empty())
{
Low_52Week = 100000;
}
else
Low_52Week = (float)atof((inner->asCString()));
}
else if (inner.key().asString() == "Beta")
{
if ((inner->asString()).empty())
{
beta = 100000;
}
else
beta = (float)atof((inner->asCString()));
}
else if (inner.key().asString() == "50DayMA")
{
```

```
if ((inner->asString()).empty())
{
MA_50Day = 1;
}
else
MA_50Day = (float)atof((inner->asCString()));
}
else if (inner.key().asString() == "200DayMA")
{
if ((inner->asString()).empty())
{
MA_200Day = 1;
}
else
MA_200Day = (float)atof((inner->asCString()));
}

}
else if (itr.key().asString() == "Highlights")
for (Json::Value::const_iterator inner_1 = (*itr).begin(); inner_1 != (*itr).end(); inner_1++)
{
if ((inner_1.key().asString() == "DividendYield") || (inner_1.key().asString() == "Yield"))
{
if ((inner_1->asString()).empty())
{
DividendYield = 100000;
}
else
DividendYield = (float)atof((inner_1->asCString()));

}
else if (inner_1.key().asString() == "PERatio")
{

if ((inner_1->asString()).empty())
{
PERatio = 100000;
}
else
PERatio = (float)atof((inner_1->asCString()));
}
else if (inner_1.key().asString() == "ProfitMargin")
{

if ((inner_1->asString()).empty())
{
ProfitMargin = 0;
}
else
ProfitMargin = (float)atof((inner_1->asCString()));
}
else if (inner_1.key().asString() == "ReturnOnAssetsTTM")
{

if ((inner_1->asString()).empty())
{
ReturnOnAssetsTTM = 0;
}
else
ReturnOnAssetsTTM = (float)atof((inner_1->asCString()));
}
else if (inner_1.key().asString() == "ReturnOnEquityTTM")
{

if ((inner_1->asString()).empty())
{
ReturnOnEquityTTM = 0;
}
else
ReturnOnEquityTTM = (float)atof((inner_1->asCString()));
}

}
}
//dividend_vector.push_back(DividendYield);

// Execute SQL
char stock_fundamental_insert_table[512];
sprintf_s(stock_fundamental_insert_table, "INSERT INTO stock_fundamental (id, symbol,High_52Week, Low_52Week, MA_50Day,
MA_200Day, DividendYield, PERatio,Beta,ProfitMargin,ReturnOnAssetsTTM,ReturnOnEquityTTM) VALUES
(%d, \"%s\", %f, %f, %f, %f, %f, %f, %f,%f,%f,%f)", count, symbol.c_str(), High_52Week, Low_52Week, MA_50Day,
MA_200Day, DividendYield, PERatio, beta, ProfitMargin, ReturnOnAssetsTTM, ReturnOnEquityTTM);
if (InsertTable(stock_fundamental_insert_table, db) == -1)
return -1;


//cout << myStock;
return 0;
}
int PopulateStockTable(const Json::Value & root, string symbol, sqlite3 *db, int num)
{
string date;
float open, high, low, close, adjusted_close, volume,daily_return = 0;
    //unsigned int volume;
Stock myStock(symbol);
int count = num;
cout << count;
for (Json::Value::const_iterator itr = root.begin(); itr != root.end(); itr++)
{
cout << *itr << endl;
for (Json::Value::const_iterator inner = (*itr).begin(); inner != (*itr).end(); inner++)
{
//cout << inner.key() << ": " << *inner << endl;
if (inner.key().asString() == "adjusted_close")
{

adjusted_close = (float)(inner->asDouble());
//cout << "adjusted_close" << endl;
}
else if (inner.key().asString() == "close")
```

```
{
close = (float)(inner->asDouble());
//cout << "close" << endl;
}
else if (inner.key() == "date")
{
date = inner->asString();
//cout << "date" << endl;
}
else if (inner.key().asString() == "high")
{
high = (float)(inner->asDouble());
    //cout << "high" << endl;
      }
else if (inner.key().asString() == "low")
{
low = (float)(inner->asDouble());
//cout << low << endl;
}
else if (inner.key() == "open")
{
open = (float)(inner->asDouble());
//cout << open << endl;
}
else if (inner.key().asString() == "volume")
{
volume = (float)(inner->asDouble());//(inner->asInt());
//cout << volume << endl;
}
else
{
cout << "Invalid json field" << endl;
system("pause");
return -1;
}
}
volume = (int)volume;
//U[i] = fundamentals.get_DividendYield() + (v3[i].get_adjusted_close() - v2[i].get_adjusted_close()) / v2[i].get_adjusted_close();
Trade* aTrade=new Trade(date, open, high, low, close, adjusted_close, volume);
myStock.addTrade(aTrade);

daily_return = myStock.daily_return(count);
count++;
cout << count<<endl;
// Execute SQL
char stockDB_insert_table[512];
sprintf_s(stockDB_insert_table, "INSERT INTO %s (id, symbol, date, open, high, low, close, adjusted_close, volume,daily_return)
VALUES(%d, \"%s\", \"%s\", %f, %f, %f, %f, %f, %f,%f)", symbol.c_str(), count, symbol.c_str(), date.c_str(), open,
high, low, close, adjusted_close, volume, daily_return);
if (InsertTable(stockDB_insert_table, db) == -1)
return -1;
}
//cout << myStock;
return 0;
}
int PopulateSP500Table(const Json::Value & root, sqlite3 *db, vector <string> &SP_ticker)
{
int count = 0;
string name, symbol, sector;
for (Json::Value::const_iterator itr = root.begin(); itr != root.end(); itr++)
{
cout << *itr << endl;
for (Json::Value::const_iterator inner = (*itr).begin(); inner != (*itr).end(); inner++)
{
//cout << inner.key() << ": " << *inner << endl;
if (inner.key().asString() == "Name")
name = inner->asString();
else if (inner.key().asString() == "Sector")
sector = inner->asString();
else if (inner.key() == "Symbol")
{
symbol = inner->asString();
SP_ticker.push_back(symbol);
}
else
{
cout << "Invalid json field" << endl;
system("pause");
return -1;
}
}
count++;
// Execute SQL
char sp500_insert_table[512];
sprintf_s(sp500_insert_table, "INSERT INTO SP500 (id, symbol, name, sector) VALUES(%d, \"%s\", \"%s\", \"%s\")",
count, symbol.c_str(), name.c_str(), sector.c_str());
if (InsertTable(sp500_insert_table, db) == -1)
return -1;
}
return 0;
}
int GetTickerData(sqlite3 *db)//, vector<string> & stock_ticker, vector<float> &stock_weight)
{
int count = -1;
fstream file;
file.open("SPYAllHoldings.csv");
string Name, ticker, sector, weight, share_held;
//float weight, share_held;
getline(file, Name);
while (file.good()) {
//Stock information;
getline(file, Name, ',');
if (Name == "") continue;
```

```
getline(file, ticker, ',');
getline(file, weight, ',');
getline(file, sector, ',');
//cout << sector << endl;
getline(file, share_held);
//cout << share_held << endl;
count += 1;
// Execute SQL
char New_sp500_insert_table[512];
sprintf_s(New_sp500_insert_table, "INSERT INTO New_SP500 (id, symbol, name, sector,weight,share_held)
VALUES(%d, \"%s\", \"%s\", \"%s\", %f, %f)", count, ticker.c_str(), Name.c_str(), sector.c_str(), atof(weight.c_str()),
atof(share_held.c_str()));
if (InsertTable(New_sp500_insert_table, db) == -1)
return -1;
}
file.close();
return 0;
}
int PopulateRiskFreeRateTable(const Json::Value & root, sqlite3 *db)
{
string date;
float open, high, low, close, adjusted_close;
int count = 0;
for (Json::Value::const_iterator itr = root.begin(); itr != root.end(); itr++)
{
cout << *itr << endl;
for (Json::Value::const_iterator inner = (*itr).begin(); inner != (*itr).end(); inner++)
{
//cout << inner.key() << ": " << *inner << endl;
if (inner.key().asString() == "adjusted_close")
adjusted_close = (float)(inner->asDouble());
else if (inner.key().asString() == "close")
close = (float)(inner->asDouble());
else if (inner.key() == "date")
date = inner->asString();
else if (inner.key().asString() == "high")
high = (float)(inner->asDouble());
else if (inner.key().asString() == "low")
low = (float)(inner->asDouble());
else if (inner.key() == "open")
open = (float)(inner->asDouble());
}
count++;
// Execute SQL
char RiskFreeRate_insert_table[512];  //(id, symbol, date, open, high, low, close, adjusted_close, volume)
VALUES(%d, \"%s\", \"%s\", %f, %f, %f, %f, %f, %d)", symbol.c_str(), count, symbol.c_str(), date.c_str(), open,
high, low, close, adjusted_close, volume);
sprintf_s(RiskFreeRate_insert_table, "INSERT INTO RiskFreeRate (id, rf, date, open, high, low, close, adjusted_close)
VALUES(%d, \"%s\", \"%s\", %f, %f, %f, %f, %f)", count, "rf", date.c_str(), open, high, low, close, adjusted_close);
if (InsertTable(RiskFreeRate_insert_table, db) == -1)
return -1;
}
return 0;
}
```

A.2 Trade Class

```
#pragma once
#ifndef Trade_h
#define Trade_h
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include "Utility.h"
#define NUM_OF_STOCKS 505
using namespace std;
class Trade
{
private:
string date;
float open;
float high;
float low;
float close;
float adjusted_close;
int volume;
float daily_return;
public:
Trade(string date_, float open_, float high_, float low_, float close_, float adjusted_close_, int volume_) :
date(date_), open(open_), high(high_), low(low_), close(close_), adjusted_close(adjusted_close_), volume(volume_)
{}
Trade(string date_, float open_, float high_, float low_, float close_, float adjusted_close_, int volume_, float daily_return_) :
date(date_), open(open_), high(high_), low(low_), close(close_), adjusted_close(adjusted_close_), volume(volume_), daily_return(daily_return_)
{}
~Trade() {}
friend ostream & operator << (ostream & out, const Trade & t)
{
out << "Date: " << t.date << " Open: " << t.open << " High: " << t.high << " Low: " << t.low << " Close: " << t.close
 << " Adjusted_Close: " << t.adjusted_close << " Volume: " << t.volume << " daily_return: " << t.daily_return << endl;
return out;
}
float get_adjusted_close()
{
return adjusted_close;
}
float get_open()
{
return open;
}
float get_close()
{
return close;
}
string get_date()
{
return date;
}
float get_daily_return()
{
return daily_return;
}
};
#endif
```

A.3 Fundamental Data Class

```
#pragma once
#ifndef Fundamental_data_h
#define Fundamental_data_h
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include "Utility.h"
#define NUM_OF_STOCKS 505
using namespace std;
class Fundamental_Data
{
private:
float PERatio;
float DividendYield;
float high_52weeks;
float low_52weeks;
float MA_50days;
float MA_200days;
float beta;
float ProfitMargin;
float ReturnOnAssetsTTM;
float ReturnOnEquityTTM;
public:
Fundamental_Data()
{}
Fundamental_Data(float high_52weeks_, float low_52weeks_, float MA_50days_, float MA_200days_, float DividendYield_, float PERatio_, float beta_,
float ProfitMargin_, float ReturnOnAssetsTTM_, float ReturnOnEquityTTM_):high_52weeks(high_52weeks_), low_52weeks(low_52weeks_), MA_50days(MA_50days_),
MA_200days(MA_200days_), DividendYield(DividendYield_), PERatio(PERatio_), beta(beta_), ProfitMargin( ProfitMargin_),ReturnOnAssetsTTM(ReturnOnAssetsTTM_),
ReturnOnEquityTTM(ReturnOnEquityTTM_)
{}
~Fundamental_Data() {}
float get_DividendYield()
{
return DividendYield;
}
float get_beta()
{
return beta;
}
float get_ProfitMargin()
{
return ProfitMargin;
}
float get_ReturnOnAssetsTTM()
{
return ReturnOnAssetsTTM;
}
float get_ReturnOnEquityTTM()
{
return ReturnOnEquityTTM;
}
float get_high_52weeks()
{
return  high_52weeks;
}
float get_MA_50days()
{
return MA_50days;
}
float get_MA_200days()
{
return MA_200days;
}
float get_PERatio()
{
return PERatio;
}
};
#endif
```

A.4 Stock Class

```
#pragma once
#ifndef Stock_h
#define Stock_h
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include "Utility.h"
#include "Trade.h"
```

```
#include "Fundamental_data.h"
#define NUM_OF_STOCKS 505
using namespace std;
class Stock
{
private:
string symbol;
vector<Trade*> trades;
float weight;
float share_held;
Fundamental_Data* fundamentals;
map<string, float> RiskFreeRate;
map<string, float> daily_return_map;
vector<float> daily_return_vector;
vector<float> RiskFreeRate_vector;
vector<float> ROC_vector;
public:
Stock()
{}
Stock(string symbol_, float weight_) :symbol(symbol_), weight(weight_)
{}
Stock(string symbol_) :symbol(symbol_)
{}
~Stock() {}
void addFundamental(Fundamental_Data * aFundamental_Data)
{
fundamentals = aFundamental_Data;
}
void addTrade(Trade *aTrade)
{
trades.push_back(aTrade);
}
void clear_trade()
{
trades.clear();
}
void addWeight(float weight_)
{
weight = weight_;
}
void addRiskFreeRate(vector<float> & RiskFreeRate_vector_)
{
RiskFreeRate_vector = RiskFreeRate_vector_;
}
void addShareHeld(float share_held_)
{
share_held = share_held_;
}
vector<Trade*> get_trades()
{
return trades;
}
string get_symbol()
{
return symbol;
}
map<string, float> Get_RiskFreeRate()
{
return RiskFreeRate;
}
Fundamental_Data* Get_fundamentals()
{
return fundamentals;
}
friend ostream & operator << (ostream & out, const Stock & s)
{
out << "Symbol: " << s.symbol << endl;
for (vector<Trade*>::const_iterator itr = s.trades.begin(); itr != s.trades.end(); itr++)
out << *(*itr);
return out;
}
float get_weight()
{
return weight;
}
float get_share_held()
{
return share_held;
}
vector<float>get_RiskFreeRate_vector()
{
return RiskFreeRate_vector;
}
float daily_return(int num)
{
if (num == 0)
{
return 0;
}
float daily_return = (trades[num]->get_adjusted_close() - trades[num - 1]->get_adjusted_close()) / trades[num - 1]->get_adjusted_close();
return daily_return;
}
void cal_daily_return_vector()
{
daily_return_vector.clear();
for (int i = 1; i < trades.size(); i++)
{
daily_return_vector.push_back(trades[i]->get_daily_return());// trades[i].get_daily_return();
}
}
vector<float> cal_daily_close_vector()
{
vector<float> daily_close_vector(0);
int size = trades.size();
for (int i = 1; i < 253; i++)
{
daily_close_vector.push_back(trades[size-i]->get_close());// trades[i].get_daily_return();
}
return daily_close_vector;
}
float cal_MA()
{
const int NUMBERS_SIZE = ROC_vector.size();
int windowSize = 6;
double sum = 0.0;
float movingAverage = 0.0;
for (int i = 0; i <= (ROC_vector.size() - windowSize); i++)
{
sum = 0.0;
for (int j = i; j < i + windowSize; j++)
sum += ROC_vector[j];
movingAverage = sum / windowSize;
}
return movingAverage;
}
void cal_ROC_vector()
{
   ROC_vector.clear();
   vector<float> daily_close_vector=cal_daily_close_vector();
   for(vector<float>::iterator it=daily_close_vector.begin()+13; it!=daily_close_vector.end();++it)
      ROC_vector.push_back((*it-*(it-12))*100/(*(it-12)));
}
float MAROC()
{
cal_ROC_vector();
float MA_for_ROC = cal_MA();
vector<float>MA_ROC;
for (vector<float>::iterator it = ROC_vector.begin(); it != ROC_vector.end(); ++it)
{
MA_ROC.push_back((*it)/ MA_for_ROC);
}
return vector_average(MA_ROC);
}
void cal_daily_return_map()
{
for (int i = 1; i < trades.size(); i++)
{
daily_return_map[trades[i]->get_date()] = trades[i]->get_daily_return();
}
}
map<string, float> get_daily_return_map()
{
return daily_return_map;
}
vector< float> get_daily_return_vector()
{
return daily_return_vector;
}
vector<float> get_close_price_vector()
{
vector<float> close_price_vector(0);
for (vector<Trade*> ::iterator it = trades.end() - 125; it != trades.end(); ++it)
{
close_price_vector.push_back((*it)->get_close());
}
return close_price_vector;
}
static bool compBySize(Stock* x, Stock* y)
{
return ((x->get_trades()).size() < (y->get_trades()).size());
}
};
#endif
```

A.5 Portfolio Class

```
#pragma once
#ifndef Portfolio_h
#define Portfolio_h
#include "Stock.h"
#include <vector>
#include <map>
#include <numeric>
#include <limits>
#include "Utility.h"
//https://patents.google.com/patent/US6484152B1/en
class Portfolio
{
private:
Stock mySPY;
float fitness;
vector<string> SP_ticker;
vector<Stock*> Stock_list;
vector<string> portfolio_ticker_list;
float total_share_held;
vector<float> portfolio_part_return;
float portfolio_adjuested_beta;
vector<float> average_RiskFreeRate;
vector<float> SPY_average_daily_return;
float sp500_average_dividend;
float sp500_average_PE;
float sp500_average_profit;
float sp500_average_ROA;
float sp500_average_ROE;
map<string, float> portfolio_daily_return;
vector<float> portfolio_return_vector;
public:
```

```
Portfolio()
{}
Portfolio(vector<Stock*> Stock_list_) :Stock_list(Stock_list_)
{}
void addPortfolio_para(Stock &mySPY_, vector<string> &SP_ticker_, vector<string> &portfolio_ticker_list_, float sp500_average_dividend_,
float sp500_average_PE_, float sp500_average_profit_,float sp500_average_ROA_,float sp500_average_ROE_)
{
mySPY = mySPY_;
SP_ticker = SP_ticker_;
portfolio_ticker_list = portfolio_ticker_list_;
sp500_average_dividend = sp500_average_dividend_;
sp500_average_PE =  sp500_average_PE_;
sp500_average_profit = sp500_average_profit_;
sp500_average_ROA = sp500_average_ROA_;
sp500_average_ROE = sp500_average_ROE_;
}
~Portfolio() {}
void update_list(vector<Stock*> Stock_list_, vector<string> portfolio_ticker_list_)
{
Stock_list = Stock_list_;
portfolio_ticker_list = portfolio_ticker_list_;
}
void add_mySPY(Stock SPY)
{
mySPY = SPY;
}
vector<string> get_SP_ticker()
{
return  SP_ticker;
}
vector<Stock*> get_Stock_list()
{
return Stock_list;
}
Stock get_mySPY()
{
return  mySPY;
}
vector<string> get_portfolio_ticker_list()
{
return  portfolio_ticker_list;
}
float get_total_share_held()
{
return total_share_held;
}
void cal_portfolio_return()
{
portfolio_daily_return.clear();
total_share_held_cal();
std::sort(Stock_list.begin(), Stock_list.end(), Stock::compBySize);
for (vector<Stock*>::iterator itr = Stock_list.begin(); itr != Stock_list.end(); itr++)
{
(*itr)->cal_daily_return_map();
}
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
if (it == Stock_list.begin())
{
map<string, float> stock_daily_return_map = (*it)->get_daily_return_map();
for (map<string, float>::iterator itr = stock_daily_return_map.begin(); itr != stock_daily_return_map.end(); itr++)
{
portfolio_daily_return[itr->first] += portfolio_daily_return[itr->first] + (itr->second)*((*it)->get_share_held()) / total_share_held;
}
}
else
{
if (((*it)->get_trades()).size() == (((*(it - 1))->get_trades()).size()))
{
total_share_held -= (*it)->get_share_held();
continue;
}
int len = ((*it)->get_daily_return_vector()).size();
Stock* before = *(it - 1);
int len_1 = before->get_daily_return_vector().size();
map<string, float> stock_daily_return_map = (*it)->get_daily_return_map();
for (map<string, float>::iterator itr = next(stock_daily_return_map.begin(),len_1); itr != stock_daily_return_map.end(); itr++)
{
portfolio_daily_return[itr->first] += portfolio_daily_return[itr->first] + (itr->second)*((*it)->get_share_held()) / total_share_held;
}
}
}

}
map<string,float> get_portfolio_daily_return()
{
return portfolio_daily_return;
}
void cal_portfolio_return_vector()
{
portfolio_return_vector.clear();
for (map<string, float>::iterator it = portfolio_daily_return.begin(); it != portfolio_daily_return.end(); ++it)
{
portfolio_return_vector.push_back(it->second);
}
}
vector<float> get_portfolio_return_vector()
{
return portfolio_return_vector;
}

float portfolio_risk()
{
```

```
vector<float> ri_rmean2;
float return_mean = vector_average(portfolio_return_vector);
for (map<string, float>::iterator it = portfolio_daily_return.begin(); it != portfolio_daily_return.end(); ++it)
{
ri_rmean2.push_back((it->second - return_mean)*(it->second - return_mean));
}
return sqrt(vector_average(ri_rmean2));
}
void cal_portfolio_adjuested_beta()
{
float beta = 0;
total_share_held_cal();
vector<float> RiskFreeRate = mySPY.get_RiskFreeRate_vector();
mySPY.cal_daily_return_vector();
vector<float> SPY_daily_return = mySPY.get_daily_return_vector();
std::sort(Stock_list.begin(), Stock_list.end(), Stock::compBySize);
vector<float> temp_vector;
//int i = 0;
float average_risk_free = 0;
float average_spy = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); ++it)
{
if (it == Stock_list.begin())
{
(*it)->cal_daily_return_vector();
float temp = vector_average((*it)->get_daily_return_vector());
int len = (*it)->get_daily_return_vector().size();
average_risk_free = accumulate(RiskFreeRate.end()-len, RiskFreeRate.end(), 0.0 / len) / len;
average_spy = accumulate(SPY_daily_return.end()-len, SPY_daily_return.end(), 0.0 / len) / len;
beta+= ((*it)->get_share_held())*(temp- average_risk_free) / (average_spy - average_risk_free);
}
else
{
(*it)->cal_daily_return_vector();
float temp = vector_average((*it)->get_daily_return_vector());
if (((*it)->get_trades()).size() == (((*(it - 1))->get_trades()).size()))
{
beta += ((*it)->get_share_held())* (temp - average_risk_free) / (average_spy - average_risk_free);
}
else
{
int len = (*it)->get_daily_return_vector().size();
average_risk_free = accumulate(RiskFreeRate.end() - len, RiskFreeRate.end(), 0.0 / len) / len;
average_spy = accumulate(SPY_daily_return.end() - len, SPY_daily_return.end(), 0.0 / len) / len;
beta += ((*it)->get_share_held())* (temp - average_risk_free) / (average_spy - average_risk_free);
}
}
}
portfolio_adjuested_beta = beta*0.67/ portfolio_part_return.size() + 0.33;
}
float get_portfolio_adjuested_beta()
{
return portfolio_adjuested_beta;
}
float Jensens_Alpha()
{
float Jensens_Alpha = 0;
float portfolio_average = vector_average(portfolio_return_vector);
float risk_free_average = vector_average(mySPY.get_RiskFreeRate_vector());
mySPY.cal_daily_return_vector();
float SPY_average= vector_average(mySPY.get_daily_return_vector());
Jensens_Alpha = portfolio_average - (risk_free_average + get_portfolio_adjuested_beta()*(SPY_average - risk_free_average));
return (Jensens_Alpha );
}
float Sharpe_Ratio()
{
float sharpe_ratio = 0;
float portfolio_average=vector_average(portfolio_return_vector);
float risk_free_average = vector_average(mySPY.get_RiskFreeRate_vector());
float excess_return = portfolio_average - risk_free_average;
    sharpe_ratio = excess_return / portfolio_risk();
cout <<"sharpe ratio" << sharpe_ratio*10 << endl;
return sharpe_ratio*10;
}
float InformationRatio()
{
float InformationRatio = 0;
mySPY.cal_daily_return_vector();
vector<float> SPY_daily_return = mySPY.get_daily_return_vector();
float SPY_average = vector_average(SPY_daily_return);
float portfolio_average = vector_average(portfolio_return_vector);
float TrackingError = 0;
cout << SPY_daily_return.size() << " " << portfolio_return_vector.size() << endl;
vector<float> difference = portfolio_return_vector - SPY_daily_return;
vector<float> difference_2 = difference * difference;
TrackingError= sqrt(vector_average(difference_2));
InformationRatio = vector_average(difference) / TrackingError;
cout << InformationRatio*50/4 << endl;
return InformationRatio*50/4;
}
void total_share_held_cal()
{
total_share_held = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
total_share_held += (*it)->get_share_held();
}
float High52WeekRatio()
{
total_share_held_cal();
float weighted_High52WeekRatio = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
```

```
{
float average_adjusted_price = vector_average((*it)->get_close_price_vector());
float hightWeekRatio = average_adjusted_price / ((*it)->Get_fundamentals())->get_high_52weeks();
weighted_High52WeekRatio += hightWeekRatio * ((*it)->get_share_held() );
}
cout << "finish weighted_High52WeekRatio cal" << weighted_High52WeekRatio / total_share_held << endl;
return weighted_High52WeekRatio / total_share_held;
}
float weightedMA50MA200_ratio()
{
total_share_held_cal();
float weightedMA50MA200_ratio = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
weightedMA50MA200_ratio += ((*it)->get_share_held())*(((*it)->Get_fundamentals())->get_MA_50days()) / (((*it)->Get_fundamentals())->get_MA_200days());
}
return (weightedMA50MA200_ratio/ total_share_held-1)*10;
}
float weighted_dividend_yield()
{
total_share_held_cal();
float weightedDividendYield = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
 weightedDividendYield += (((*it)->Get_fundamentals())->get_DividendYield()- 0.03)*((*it)->get_share_held() )/ 0.03;
}
return weightedDividendYield / total_share_held;
}
float weighted_ROE()
{
total_share_held_cal();
float weightedROE = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
weightedROE += (((*it)->Get_fundamentals())->get_ReturnOnEquityTTM())*((*it)->get_share_held());
cout <<(*it)->get_symbol()<< " weightedROE cal " << (((*it)->Get_fundamentals())->get_ReturnOnEquityTTM()) << endl;
}
return weightedROE*0.1 / total_share_held;
}
float weighted_profit()
{
total_share_held_cal();
float weightedProfit = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
weightedProfit += (((*it)->Get_fundamentals())->get_ProfitMargin())*((*it)->get_share_held());
//cout << " weightedDividendYield cal " << ((*it)->Get_fundamentals())->get_DividendYield() << endl;
}
return weightedProfit / total_share_held/ sp500_average_profit;
}
float weighted_ROA()
{
total_share_held_cal();
float weightedROA = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
weightedROA += (((*it)->Get_fundamentals())->get_ReturnOnAssetsTTM())*((*it)->get_share_held());
}
cout << " weightedROA cal " << weightedROA*5 / total_share_held << endl;
return weightedROA*6/ total_share_held;
}
float weighted_PERatio()
{
float weighted_PERatio = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
float PERatio = ((*it)->Get_fundamentals())->get_PERatio();
if (PERatio < 7)
{
weighted_PERatio += 0 * ((*it)->get_share_held());
}
else if (PERatio > sp500_average_PE)
{
weighted_PERatio += 0 * ((*it)->get_share_held());
}
else
{
weighted_PERatio += 1 * ((*it)->get_share_held());
}
}
cout << "finish weighted_PERatio cal" << weighted_PERatio / total_share_held << endl;
return weighted_PERatio / total_share_held;
}
float weighted_MAROC()
{
total_share_held_cal();
float weighted_MAROC = 0;
for (vector<Stock*>::iterator it = Stock_list.begin(); it != Stock_list.end(); it++)
{
  float MAROC = (*it)->MAROC();
  if(MAROC<1)
            weighted_MAROC = 0.2* ((*it)->get_share_held());
  else if(MAROC < 1.02)
 weighted_MAROC = 0.4* ((*it)->get_share_held());
  else
  weighted_MAROC = 0.6* ((*it)->get_share_held());
}
return weighted_MAROC / total_share_held;
}
void cal_fitness()
{
fitness = 0;
cal_portfolio_return();
```

```
cal_portfolio_return_vector();
fitness = InformationRatio()+ weightedMA50MA200_ratio() + weighted_ROA();
}
float get_fitness()
return fitness;
static bool compByFitness(Portfolio* x, Portfolio* y)
{
return x->get_fitness() < y->get_fitness();
}
};
#endif
```

A.6 Genetic Algorithm Class

```
#pragma once
#ifndef GenA_h
#define GenA_h
#include <string>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <math.h>
#include <vector>
#include <algorithm>
#include <random>
#include "Portfolio.h"
using namespace std;
#define MUTATION_RATE            0.03
#define POP_SIZE                 100      //must be an even number
#define CHROMO_LENGTH            10
#define MAX_ALLOWABLE_GENERATIONS   1000
//returns a float between 0 & 1
#define RANDOM_NUM      ((float)rand()/RAND_MAX)
typedef vector<float> Vector;
//Detail your design of all the classes, your crossover and mutation logic. Specify any unique or improvement to our proposal.
void GetSectorStockTicker(vector<string>&portfolio_ticker_list, vector<vector<string>>&sector_category)
{
for (vector<vector<string>>::iterator it = sector_category.begin(); it != sector_category.end(); ++it)
{
// int randNum = rand()%(max-min + 1) + min
int Rand_Num = rand() % (sector_category.size());
string ticker = (*it)[Rand_Num];
portfolio_ticker_list.push_back(ticker);
}
for (int i = 9; i != portfolio_ticker_list.size() - 1; i++)
{
portfolio_ticker_list.pop_back();
}
}
void GetRandomStockTicker( vector<string>&portfolio_ticker_list, vector <string> &SP_ticker)
{
int length = 10;
std::random_device rd;
std::mt19937 g(rd());
std::shuffle(SP_ticker.begin()+1, SP_ticker.end(), g);
for (vector<string>::iterator it = SP_ticker.begin()+1; it != SP_ticker.begin() + length+1; ++it)
portfolio_ticker_list.push_back(*it);
}
void Mutate(Portfolio &offspring, map<string, Stock*> &sp500_Stock)
{
int num = rand() % 100 + 1;
int len = 10;
Stock mySPY = offspring.get_mySPY();
vector<string>SP_ticker = offspring.get_SP_ticker();
vector<Stock*> new_portfolio_Stock_list;
vector<Stock*> portfolio_Stock_list = offspring.get_Stock_list();
vector<string> portfolio_ticker_list = offspring.get_portfolio_ticker_list();
sort(portfolio_ticker_list.begin(), portfolio_ticker_list.end());
for (int i = 1; i < portfolio_ticker_list.size(); i++)
if (portfolio_ticker_list[i - 1] == portfolio_ticker_list[i])
{
bool btn = true;
while (btn)
{
string ticker_1 = SP_ticker[rand() % (SP_ticker.size() - 1) + 1];
if (std::find(portfolio_ticker_list.begin(), portfolio_ticker_list.end(), ticker_1) != portfolio_ticker_list.end())
{
}
else
{
portfolio_ticker_list[i] = ticker_1;
btn = false;
}
}
}
for (vector<string>::iterator it = portfolio_ticker_list.begin(); it != portfolio_ticker_list.end(); ++it)
{
new_portfolio_Stock_list.push_back(sp500_Stock[*it]);
}
if (num < 4)
{
new_portfolio_Stock_list.clear();
int randNum = rand() % (SP_ticker.size() - 1) + 1;
int randNum_1 = rand() % (SP_ticker.size() - 1) + 1;
string ticker = (SP_ticker[randNum]);
string ticker_ = (SP_ticker[randNum_1]);
int rand_portfolio = rand() % (portfolio_Stock_list.size() - 1) + 1;
portfolio_Stock_list.pop_back();
portfolio_ticker_list.pop_back();
portfolio_Stock_list.pop_back();
portfolio_ticker_list.pop_back();
portfolio_Stock_list.push_back(sp500_Stock[ticker]);
portfolio_ticker_list.push_back(ticker);
portfolio_Stock_list.push_back(sp500_Stock[ticker_]);
portfolio_ticker_list.push_back(ticker_);
for (int i = 1; i < portfolio_ticker_list.size(); i++)
if (portfolio_ticker_list[i - 1] == portfolio_ticker_list[i])
{
bool btn = true;
while (btn)
{
string ticker_1 = SP_ticker[rand() % (SP_ticker.size() - 1) + 1];
if (std::find(portfolio_ticker_list.begin(), portfolio_ticker_list.end(), ticker_1) != portfolio_ticker_list.end())
{
btn = true;
}
else
{
portfolio_ticker_list[i] = ticker_1;
btn = false;
}
}
}
std::random_device rd;
std::mt19937 g(rd());
std::shuffle(portfolio_ticker_list.begin(), portfolio_ticker_list.end(), g);
for (vector<string>::iterator it = portfolio_ticker_list.begin(); it != portfolio_ticker_list.end(); ++it)
{
new_portfolio_Stock_list.push_back(sp500_Stock[*it]);
}
}
//cout << "portoflio list   " << portfolio_ticker_list << endl;
offspring.update_list(new_portfolio_Stock_list, portfolio_ticker_list);
return;
}
void Crossover(Portfolio &offspring1, Portfolio &offspring2)
{
std::random_device rd;
std::mt19937 g(rd());
Stock mySPY = offspring1.get_mySPY();
vector<string> SP_ticker = offspring1.get_SP_ticker();
int crossover = (int)(RANDOM_NUM * CHROMO_LENGTH);
vector<Stock*> portfolio_stock_list_1 = offspring1.get_Stock_list();
vector<Stock*> portfolio_stock_list_2 = offspring2.get_Stock_list();
std::shuffle(portfolio_stock_list_1.begin(), portfolio_stock_list_1.end(), g);
std::shuffle(portfolio_stock_list_2.begin(), portfolio_stock_list_2.end(), g);
vector<Stock*> new_Stock_list_1, new_Stock_list_2;
vector<string> new_ticker_list_1, new_ticker_list_2;
for (vector<Stock*>::iterator it = portfolio_stock_list_1.begin(); it != portfolio_stock_list_1.begin() + crossover; ++it)
{
new_Stock_list_1.push_back(*it);
new_ticker_list_1.push_back((*it)->get_symbol());
}
for (vector<Stock*>::iterator it = portfolio_stock_list_2.begin(); it != portfolio_stock_list_2.begin() + crossover; ++it)
{
new_Stock_list_2.push_back(*it);
new_ticker_list_2.push_back((*it)->get_symbol());
}
for (vector<Stock*>::iterator it = portfolio_stock_list_2.begin() + crossover; it != portfolio_stock_list_2.end(); ++it)
{
new_Stock_list_1.push_back(*it);
new_ticker_list_1.push_back((*it)->get_symbol());
}
for (vector<Stock*>::iterator it = portfolio_stock_list_1.begin() + crossover; it != portfolio_stock_list_1.end(); ++it)
{
new_Stock_list_2.push_back(*it);
new_ticker_list_2.push_back((*it)->get_symbol());
}
offspring1.update_list(new_Stock_list_1, new_ticker_list_1);
offspring2.update_list(new_Stock_list_2, new_ticker_list_2);
}
void New_Population(vector<Portfolio*> &Population)
{
std::sort(Population.begin(), Population.end(),Portfolio::compByFitness);
int j = Population.size() - 1;
for (int i = 0; i < Population.size()*0.2; i += 2)
{
Population[i] = Population[j];
Population[i + 1] = Population[j];
j -= 1;
}
std::sort(Population.begin(), Population.end(), Portfolio::compByFitness);
for (vector<Portfolio*>::iterator it = Population.begin(); it != Population.end(); it++)
{
cout << (*it)->get_fitness();
cout << (*it)->get_portfolio_ticker_list()<<endl;
}
}
#endif
```

A.7 Back Test And Probation Test

```
#pragma once
#ifndef BackTest_h
#define BackTest_h
#include "Portfolio.h"
#include "Stock.h"
#include<iostream>
#include <time.h>
#include <ctime>
#include <sstream>
#include <iomanip>
```

```
#pragma warning(disable : 4996)
using namespace std;
float Test(Portfolio* best_Portfolio)
{
int cash = 1000000;
Stock mySPY = best_Portfolio->get_mySPY();
vector<Stock*> stock_list = best_Portfolio->get_Stock_list();
vector<float> weight_list;
best_Portfolio->total_share_held_cal();
for (vector<Stock*>::iterator it= stock_list.begin();it!= stock_list.end();it++)
{
weight_list.push_back(((*it)->get_share_held())/ (best_Portfolio->get_total_share_held()));
}
vector<Trade*> SPY_trades = mySPY.get_trades();
vector<int>  all_day;
vector<float> adjusted_close_vector;
vector<float> open_vector;
vector<float> SPY_adjusted_close_vector;
vector<float> SPY_open_vector;
vector<int> month_vector_stock;
int action = 0;
for (vector<Trade*>::iterator it = SPY_trades.begin(); it != SPY_trades.end()-1;++it)
{
string date = (*it)->get_date();
std::tm t = {};
std::istringstream ss(date);
ss >> std::get_time(&t, "%Y-%m-%d");
std::mktime(&t);
int get_month = t.tm_mon+1;
int day = t.tm_wday;
string date_1 = (*(it +1))->get_date();
std::tm t_1 = {};
std::istringstream ss_1(date_1);
ss_1 >> std::get_time(&t_1, "%Y-%m-%d");
std::mktime(&t_1);
int get_month_1 = t_1.tm_mon + 1;
int position = std::distance(SPY_trades.begin(), it);
if (get_month<=8)// month)
{
for (vector<Stock*>::iterator itr = stock_list.begin(); itr != stock_list.end(); itr++)
{
vector<Trade*> myTrade = (*itr)->get_trades();
vector <float> portfolio_price;
// 3 4 5 1 2 3 4 5 1 2 3 4
int first_day = t.tm_wday;//5
int next_day = t_1.tm_wday;//1endl
if ((first_day > next_day))
{
action = -1;
float adjusted_close = (*myTrade[position]).get_close();
float SPY_adjusted_close = (*SPY_trades[position]).get_close();
adjusted_close_vector.push_back(adjusted_close);
SPY_adjusted_close_vector.push_back(SPY_adjusted_close);
}
else if (first_day < next_day && action == 0)// 1<2
{
//cout << day << endl;
float open = (*myTrade[position]).get_open();
float SPY_open = (*SPY_trades[position]).get_open();
open_vector.push_back(open);
SPY_open_vector.push_back(SPY_open);
month_vector_stock.push_back(get_month);
}
else
{
continue;
}
}
action += 1;
}
}
vector<int> share_list;
vector<float> excess_return_list(0);
vector<int> month_vector;
for (int i = 0; i !=adjusted_close_vector.size(); i = i + 10)
{
month_vector.push_back(month_vector_stock[i]);
share_list.clear();
float left_cash = cash;
for (int j = 0; j != stock_list.size(); j++)
{
int share_1 = (int)((cash *weight_list[j]) / open_vector[i + j]);
share_list.push_back(share_1);
left_cash = left_cash - share_1 * open_vector[i + j];
}
float portfolio_total_return = 0;
float cash_invested = cash - left_cash;
float share_for_SPY = cash_invested / SPY_open_vector[i];
for (int k=0;k!= share_list.size();k++)
{
portfolio_total_return += share_list[k] * adjusted_close_vector[i + k];
}
float SPY_total_return = share_for_SPY * SPY_adjusted_close_vector[i];
float excess_return = portfolio_total_return - SPY_total_return;
excess_return_list.push_back(excess_return);
cout << excess_return  << endl;
}
int month = month_vector[0];
while (month< month_vector[month_vector.size()-1]+1)
{
float month_excess_return = 0;
for (int i = 0; i != excess_return_list.size(); i += 1)
{
if (month_vector[i] == month)
{
month_excess_return += excess_return_list[i];
}
}
cout << "The month " << month << " has excess return " << month_excess_return<<endl;
month += 1;
}
return vector_average(excess_return_list);

}
#endif
```

A.8 Main Function

```
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>

#include "curl_easy.h"
#include "curl_form.h"
#include "curl_ios.h"
#include "curl_exception.h"

#include "json/json.h"
#include <sqlite3.h>

#include "Database.h"
#include "Stock.h"
#include "GenA.h"
#include "Portfolio.h"
#include "Utility.h"
#include "AddMissing.h"
#include "BackTest.h"
#include <cstdlib>
#define NUM_OF_STOCKS 505
#include <time.h>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <sstream>

#include <stdio.h>
//boost::gregorian::from_simple_string();
#include <sstream>
#define CROSSOVER_RATE            0.7
#define MUTATION_RATE             0.03
#define POP_SIZE                  100     //must be an even number
#define CHROMO_LENGTH             10
//#define GENE_LENGTH               4
#define MAX_ALLOWABLE_GENERATIONS   1000
#pragma warning(disable : 4996)
using namespace std;

int main(void)
{
map<string, int> mymap{ {"1",1},{"2",2 } };
map<string, int> mymap_1{ {"1",1},{"2",2 },{"3",3},{"4",4} };
for (map<string, int>::iterator it = next(mymap_1.begin(), 2); it != mymap_1.end(); it++)
{
cout << (it)->second << endl;
}
vector<string> best_Portfolio_ticker;
vector<Stock*> best_Portfolio_Stock;
Stock* mySPY = nullptr;
Portfolio* best_Portfolio=nullptr;
string api_token = "5ba84ea974ab42.45160048";
vector <string> SP_ticker;
SP_ticker.push_back("SPY");
const char * stockDB_name = "Stocks.db";
int count = 0;
sqlite3 * stockDB = NULL;
map <string, Stock*> sp500_Stock;
//vector<string> ticker_list;
vector<float> weight_list;
vector<float> share_held_list;
vector<float> RiskFreeRate_vector;
vector<float> dividend_vector;
vector<float> PE_vector;
vector<float> profit_vector;
vector<float> ROE_vector;
vector<float> ROA_vector;

//vector<float>
float sp500_dividend = 0;
float sp500_PE = 0;
float sp500_profit = 0;
float sp500_ROE = 0;
float sp500_ROA = 0;
weight_list.push_back(1);
share_held_list.push_back(3186981124);
map<string, vector<string>> Communication_Services, Consumer_Discretionary, Consumer_Staples, Energy, Financials, Health_Care,
Industrials,
Information_Technology, Materials, Real_Estate, Utilities, Unassigned;
vector<map<string, vector<string>>> sector_category{ Communication_Services,Consumer_Discretionary,Consumer_Staples, Energy,
Financials,Health_Care,Industrials,Information_Technology,Information_Technology,Materials,
Real_Estate,Utilities, Unassigned };

if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
map<string, float> RiskFreeRate;

string sp500_select_table = "SELECT * FROM New_SP500;";
//test
DisplayNewSP500Ticker(sp500_select_table.c_str(), stockDB, SP_ticker, weight_list, share_held_list, sector_category);

CloseDatabase(stockDB);
bool a = true;
int attempt = 0;
```

```
while (a)
{
string str;
cout << endl;
cout << "Pick a choice from the list:" << endl << endl
<< "1.Retrieve SP500 ticker and save to database" << endl
<< "2.Retrieve new SP500 ticker and save to database" << endl
<< "3.Retrieve trade data and save to database" << endl
<< "4.Retrieve fundamental data and save to database" << endl
<< "5.Retrieve risk free rate and save to database" << endl
<< "6.Get stock data from database" << endl
<< "7.Portfolio Selection by genetic algorithm" << endl
<< "8.Backtest"<<endl
<< "9.Exit your program" << endl << endl;
cin >> str;
int answer;
answer = std::stoi(str);
switch (answer)
{
case 1:
{
cout << endl << "Retrieve SP500 ticker and save to database..." << endl << endl;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
std::string sp500_drop_table = "DROP TABLE IF EXISTS SP500;";
if (DropTable(sp500_drop_table.c_str(), stockDB) == -1)
return -1;
string sp500_create_table = "CREATE TABLE SP500 (id INT PRIMARY KEY NOT NULL, symbol CHAR(20) NOT NULL, name CHAR(20) NOT NULL,
sector CHAR(20) NOT NULL);";
if (CreateTable(sp500_create_table.c_str(), stockDB) == -1)
return -1;
string sp500_data_request = "https://pkgstore.datahub.io/core/s-and-p-500-companies/constituents_json/data/64dd3e9582b936b0352fdd826ecd3c95/constituents_json.json";
//string sp500_data_request = "https://datahub.io/core/s-and-p-500-companies/r/0.html";
Json::Value sp500_root;   // will contains the root value after parsing.
if (RetrieveMarketData(sp500_data_request, sp500_root) == -1)
return -1;
if (PopulateSP500Table(sp500_root, stockDB, SP_ticker) == -1)
return -1;
cout << sp500_root << endl;
string sp500_select_table = "SELECT * FROM SP500;";
if (DisplayTable(sp500_select_table.c_str(), stockDB) == -1)
return -1;
CloseDatabase(stockDB);
cout << "Retrieve successfully! Thank you for your patience!" << endl << endl;
attempt++;
break;
}
case 2:
{
cout << endl << "Retrieve new SP500 ticker and save to database..." << endl << endl;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
std::string New_sp500_drop_table = "DROP TABLE IF EXISTS New_SP500;";
if (DropTable(New_sp500_drop_table.c_str(), stockDB) == -1)
return -1;
string New_sp500_create_table = "CREATE TABLE New_SP500 (id INT PRIMARY KEY NOT NULL, symbol CHAR(20) NOT NULL,
name CHAR(20) NOT NULL, sector CHAR(20) NOT NULL,weight REAL NOT NULL, share_held REAL NOT NULL);";
if (CreateTable(New_sp500_create_table.c_str(), stockDB) == -1)
return -1;
if (GetTickerData(stockDB) == -1)
return -1;
string New_sp500_select_table = "SELECT * FROM New_SP500;";
if (DisplayTable(New_sp500_select_table.c_str(), stockDB) == -1)
return -1;
CloseDatabase(stockDB);
cout << "Retrieve successfully! Thank you for your patience!" << endl << endl;
attempt++;
break;
}
case 3:
{
cout << endl << "Retrieve trade data and save to database" << endl << endl;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
string stock_url_common = "https://eodhistoricaldata.com/api/eod/";
string api_token = "5ba84ea974ab42.45160048";
string stock_start_date = "2008-01-01";
string stock_end_date = "2019-08-01";
vector<string> temp_stock_ticker{ "A"};
for (vector<string> ::iterator it = temp_stock_ticker.begin(); it != temp_stock_ticker.end(); it++)
{
cout << *it << endl;
string stockDB_symbol = *it;
if (*it == "BRK_B")
*it = "BRK-B";
if (*it == "BF_B")
*it = "BF-B";
if (*it == "LUK")
continue;
if (*it == "MON")
continue;
//https://eodhistoricaldata.com/api/eod/BRK.B.US?from=2008-01-01&to=2019-06-06&api_token=5ba84ea974ab42.45160048&period=d&fmt=json
string stockDB_data_request = stock_url_common + *it + ".US?" +
"from=" + stock_start_date + "&to=" + stock_end_date + "&api_token=" + api_token + "&period=d&fmt=json";
if (stockDB_symbol == "ALL")
stockDB_symbol = "Stock_ALL";
int pos = stockDB_symbol.find('-');
if (pos != std::string::npos)
stockDB_symbol.replace(pos, 1, 1, '_');

std::string stockDB_drop_table = "DROP TABLE IF EXISTS " + stockDB_symbol + ";";
if (DropTable(stockDB_drop_table.c_str(), stockDB) == -1)
return -1;
string stockDB_create_table = "CREATE TABLE " + stockDB_symbol
+ "(id INT PRIMARY KEY NOT NULL,"
+ "symbol CHAR(20) NOT NULL,"
+ "date CHAR(20) NOT NULL,"
+ "open REAL NOT NULL,"
+ "high REAL NOT NULL,"
+ "low REAL NOT NULL,"
+ "close REAL NOT NULL,"
+ "adjusted_close REAL NOT NULL,"
+ "volume REAL NOT NULL,"
+ "daily_return REAL NOT NULL);";
if (CreateTable(stockDB_create_table.c_str(), stockDB) == -1)
return -1;
Json::Value stockDB_root;   // will contains the root value after parsing.
if (RetrieveMarketData(stockDB_data_request, stockDB_root) == -1)
return -1;
if (PopulateStockTable(stockDB_root, stockDB_symbol, stockDB, 0) == -1)
return -1;
string stockDB_select_table = "SELECT * FROM " + stockDB_symbol + ";";
if (DisplayTable(stockDB_select_table.c_str(), stockDB) == -1)
return -1;
}
CloseDatabase(stockDB);
break;
}
case 4:
{
cout << endl << "Retrieve fundamental data and save to database..." << endl << endl;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
//stock fundamental data
//https://eodhistoricaldata.com/api/fundamentals/AAPL.US?api_token=5ba84ea974ab42.45160048&fmt=json
string stock_Fundamental_url_common = "https://eodhistoricaldata.com/api/fundamentals/";
std::string stock_fundamental_drop_table = "DROP TABLE IF EXISTS stock_fundamental;";
if (DropTable(stock_fundamental_drop_table.c_str(), stockDB) == -1)
return -1;
string stock_fundamental_create_table = "CREATE TABLE stock_fundamental (id INT PRIMARY KEY NOT NULL, symbol CHAR(20)
NOT NULL, High_52Week REAL NOT NULL,Low_52Week REAL NOT NULL,MA_50Day REAL NOT NULL,MA_200Day REAL NOT NULL,DividendYield REAL NOT NULL,
PERatio REAL NOT NULL,Beta REAL NOT NULL,ProfitMargin REAL NOT NULL,ReturnOnAssetsTTM REAL NOT NULL,ReturnOnEquityTTM REAL NOT NULL); ";
if (CreateTable(stock_fundamental_create_table.c_str(), stockDB) == -1)
return -1;

for (vector<string> ::iterator it = SP_ticker.begin(); it != SP_ticker.end(); it++)//SP_ticker.end()
{
if (*it == "BRK_B")
*it = "BRK-B";
if (*it == "BF_B")
*it = "BF-B";
if (*it == "LUK")
continue;
if (*it == "MON")
continue;
Stock myStock(*it);
string stockDB_symbol = *it;
if (stockDB_symbol == "ALL")
stockDB_symbol = "Stock_ALL";


// creat database for fundamental data
string FundamentalDB_data_request = stock_Fundamental_url_common + *it + ".US?" +
"&api_token=" + api_token;
Json::Value stockFD_root;   // will contains the root value after parsing.
if (RetrieveMarketData(FundamentalDB_data_request, stockFD_root) == -1)
return -1;
cout << count << endl;
if (PopulateFundamentalDataTable(stockFD_root, *it, stockDB, count) == -1)
return -1;
cout << stockFD_root << endl;
string stock_fundamental_select_table = "SELECT * FROM stock_fundamental;";
if (DisplayTable(stock_fundamental_select_table.c_str(), stockDB) == -1)
return -1;
count++;
}
CloseDatabase(stockDB);
cout << "Retrieve successfully! Thank you for your patience!" << endl << endl;
attempt++;

break;
}
case 5:
{
cout << endl << "Retrieve risk free rate and save to database..." << endl << endl;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
std::string RiskFreeRate_drop_table = "DROP TABLE IF EXISTS RiskFreeRate;";
if (DropTable(RiskFreeRate_drop_table.c_str(), stockDB) == -1)
return -1;
string RiskFreeRate_create_table = "CREATE TABLE RiskFreeRate (id INT PRIMARY KEY NOT NULL, rf CHAR(20) NOT NULL,
date CHAR(20) NOT NULL, open REAL NOT NULL, high REAL NOT NULL, low REAL NOT NULL, close REAL NOT NULL, adjusted_close REAL NOT NULL);";
if (CreateTable(RiskFreeRate_create_table.c_str(), stockDB) == -1)
return -1;
string RiskFreeRate_data_request = "https://eodhistoricaldata.com/api/eod/TNX.INDX?from=2008-01-01&to=2019-06-06&api_token=5ba84ea974ab42.45160048&period=d&fmt=json"
Json::Value RiskFreeRate_root;   // will contains the root value after parsing.
if (RetrieveMarketData(RiskFreeRate_data_request, RiskFreeRate_root) == -1)
return -1;
if (PopulateRiskFreeRateTable(RiskFreeRate_root, stockDB) == -1)
return -1;
cout << RiskFreeRate_root << endl;
```

```
string RiskFreeRate_select_table = "SELECT * FROM RiskFreeRate;";
if (DisplayTable(RiskFreeRate_select_table.c_str(), stockDB) == -1)
return -1;
CloseDatabase(stockDB);
cout << "Retrieve successfully! Thank you for your patience!" << endl << endl;
break;
}
case 6:
{
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
for (vector<string> ::iterator it = SP_ticker.begin(); it != SP_ticker.end(); it++)
{
//cout << *it << endl;
if (*it == "BRK.B")
*it = "BRKB";
if (*it == "BF.B")
*it = "BF_B";
if (*it == "ALL")
*it = "Stock_ALL";
int position = std::distance(SP_ticker.begin(), it);//std::max_element(SP_ticker.begin(), SP_ticker.end()));
Stock *myStock = new Stock(*it);
myStock->addWeight(weight_list[position]);
myStock->addShareHeld(share_held_list[position]);
string stockDB_symbol = *it;
string stockDB_select_table = "SELECT * FROM " + stockDB_symbol + " WHERE strftime(\'%Y-%m-%d\', date)
between \"2008-01-02\" and \"2018-12-31\";";
if (DisplayTradeData(stockDB_select_table.c_str(), stockDB, myStock) == -1)
return -1;
string stock_fundamental_select_table = "SELECT * FROM stock_fundamental;";
int num = std::distance(SP_ticker.begin(), it);
if (DisplayFundamentalData(stock_fundamental_select_table.c_str(), stockDB, myStock, num, dividend_vector,  PE_vector, profit_vector,ROA_vector,ROE_vector) == -1)
return -1;
string RiskFreeRate_select_table = "SELECT * FROM RiskFreeRate WHERE strftime(\'%Y-%m-%d\', date)
between \"2008-01-02\" and \"2018-12-31\";";
if (DisplayRiskFreeRateData(RiskFreeRate_select_table.c_str(), stockDB, RiskFreeRate, RiskFreeRate_vector) == -1)
return -1;
myStock->addRiskFreeRate(RiskFreeRate_vector);
sp500_Stock[*it] = myStock;

}
sp500_dividend = vector_average(dividend_vector);
sp500_PE = vector_average(PE_vector);
sp500_profit = vector_average(profit_vector);
sp500_ROA = vector_average(ROA_vector);
sp500_ROE = vector_average(ROE_vector);
cout << "dividend average " << sp500_dividend << endl;
cout << "dividend average " << sp500_PE << endl;
cout << "sp500_profit average " << sp500_profit << endl;
cout << "sp500_ROA average " << sp500_ROA << endl;
cout << "sp500_ROE average " << sp500_ROE << endl;
break;
}
case 7:
{
//fill_missing_return(sp500_Stock);
cout << "Portfolio selection" << endl;
Stock* mySPY = sp500_Stock["SPY"];
//seed the random number generator
srand((int)time(NULL));
//just loop endlessly until user gets bored :0)
while (true)
{
//storage for our population of chromosomes.
vector<Portfolio*> Population;
    for (int i = 0; i < POP_SIZE; i++)
{
vector<string>portfolio_ticker_list(0);
vector<Stock*>portfolio_stock_list(0);
//GetSectorStockTicker(portfolio_ticker_list, sector_category);
GetRandomStockTicker(portfolio_ticker_list, SP_ticker);
for (vector<string>::iterator it = portfolio_ticker_list.begin(); it != portfolio_ticker_list.end(); ++it)
{
portfolio_stock_list.push_back(sp500_Stock[*it]);
}
Portfolio *myPortfolio = new Portfolio(portfolio_stock_list);
myPortfolio->addPortfolio_para(*mySPY, SP_ticker, portfolio_ticker_list, sp500_dividend,sp500_PE, sp500_profit,
sp500_ROA, sp500_ROE);
Population.push_back(myPortfolio);
}
int GenerationsRequiredToFindASolution = 0;
//we will set this flag if a solution has been found
bool bFound = false;
//enter the main GA loop
while (!bFound)
{
cout << "==============================" << endl;
cout << "The " << GenerationsRequiredToFindASolution << " generation" << endl;
cout << "==============================" << endl;
// test and update the fitness of every chromosome in the
// population
    for (int i = 0; i < POP_SIZE; i++)
{
Population[i]->cal_fitness();
}
std::sort(Population.begin(), Population.end(), Portfolio::compByFitness);
vector<Portfolio*> temp;
vector<Portfolio*> new_pop;
vector<Portfolio> no_change;
```

```
New_Population(Population);
std::random_device rd;
std::mt19937 g(rd());
for (vector<Portfolio*>::iterator it = Population.begin(); it != Population.begin() + Population.size()*0.95; it++)
{
temp.push_back(*it);
}
for (vector<Portfolio*>::iterator it = Population.begin() + Population.size()*0.95; it != Population.end(); it++)
{
no_change.push_back(*(*it));
}
std::shuffle(temp.begin(), temp.end(), g);
for (vector<Portfolio*>::iterator it = temp.begin()+1; it < temp.end()-1; it = it + 2)
{
Portfolio* offspring1 = *it;
Portfolio* offspring2 = *(it-1);
Crossover(*offspring1, *offspring2);
Mutate(*offspring1, sp500_Stock);
Mutate(*offspring2, sp500_Stock);
new_pop.push_back(offspring1);
new_pop.push_back(offspring2);

}
for (int i=0; i!= new_pop.size();i++)
{
Population[i] = new_pop[i];
}
for (int i = 0; i != no_change.size(); i++)
{
vector<string> ticker_list=no_change[i].get_portfolio_ticker_list();
Portfolio *myPortfolio = new Portfolio(no_change[i].get_Stock_list());
myPortfolio->addPortfolio_para(*mySPY, SP_ticker, ticker_list, sp500_dividend,sp500_PE, sp500_profit,
 sp500_ROA, sp500_ROE);
Population[Population.size() - 1 - i] = myPortfolio;//&no_change[i];
}
cout << "finished children generation================================" << endl;
++GenerationsRequiredToFindASolution;
if (GenerationsRequiredToFindASolution > MAX_ALLOWABLE_GENERATIONS)
{
cout << Population[99]->get_portfolio_ticker_list();
best_Portfolio = Population[99];
bFound = true;
}
}
break;
cout << "\n\n\n";
}//end while
break;
}
case 8:
{
    mySPY = sp500_Stock["SPY"];
best_Portfolio_ticker = best_Portfolio->get_portfolio_ticker_list();
best_Portfolio_Stock = best_Portfolio->get_Stock_list();
vector<Stock*> new_best_Portfolio_Stock;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
for (vector<Stock*> ::iterator it = best_Portfolio_Stock.begin(); it != best_Portfolio_Stock.end(); it++)
{
cout << (*it)->get_symbol() << endl;
string stockDB_symbol = (*it)->get_symbol();
(*it)->clear_trade();
string stockDB_select_table = "SELECT * FROM " + stockDB_symbol + " WHERE strftime(\'%Y-%m-%d\', date)
between \"2019-01-02\" and \"2019-07-01\";";
if (DisplayTradeData(stockDB_select_table.c_str(), stockDB, *it) == -1)
return -1;
new_best_Portfolio_Stock.push_back(*it);//sp500_Stock.push_back(myStock);
}
best_Portfolio->update_list(new_best_Portfolio_Stock, best_Portfolio_ticker);

Stock* mySPY = sp500_Stock["SPY"];
mySPY->clear_trade();
string stockDB_symbol = "SPY";
string stockDB_select_table = "SELECT * FROM " + stockDB_symbol + " WHERE strftime(\'%Y-%m-%d\', date)
between \"2019-01-02\" and \"2019-07-01\";";
if (DisplayTradeData(stockDB_select_table.c_str(), stockDB, mySPY) == -1)
return -1;
best_Portfolio->add_mySPY(*mySPY);
int month = 1;
float excess_return = 1;
int cash = 1000000;
excess_return = Test(best_Portfolio);
break;
}
case 9:
{
vector<Stock*> new_best_Portfolio_Stock;
if (OpenDatabase(stockDB_name, stockDB) == -1)
return -1;
for (vector<Stock*> ::iterator it = best_Portfolio_Stock.begin(); it != best_Portfolio_Stock.end(); it++)
{
string stockDB_symbol = (*it)->get_symbol();
cout << (*it)->get_symbol() << endl;
(*it)->clear_trade();
string stockDB_select_table = "SELECT * FROM " + stockDB_symbol + " WHERE strftime(\'%Y-%m-%d\', date)
```

```
between \"2019-07-01\" and \"2019-07-30\";";
if (DisplayTradeData(stockDB_select_table.c_str(), stockDB, *it) == -1)
return -1;
new_best_Portfolio_Stock.push_back(*it);//sp500_Stock.push_back(myStock);

}
best_Portfolio->update_list(new_best_Portfolio_Stock, best_Portfolio_ticker);

Stock* mySPY = sp500_Stock["SPY"];
mySPY->clear_trade();
string stockDB_symbol = "SPY";
string stockDB_select_table = "SELECT * FROM " + stockDB_symbol + " WHERE strftime(\'%Y-%m-%d\', date)
 between \"2019-07-01\" and \"2019-07-30\";";
if (DisplayTradeData(stockDB_select_table.c_str(), stockDB, mySPY) == -1)
return -1;
best_Portfolio->add_mySPY(*mySPY);
////SPY haven't change!!!
int month = 7;
float excess_return = 1;
int cash = 1000000;
excess_return = Test(best_Portfolio);
break;
}
case 10:
{
a = false;
cout << endl << "successfully exit!" << endl << endl;
exit(0);
break;
default:
cout << endl << "Bad choice! Please try again: " << endl;
}
}
}
system("pause");
return 0;
}
```

A.9 Utility Function

```
#pragma once
#ifndef Utility_h
#define Utility_h
#include <iostream>
#include <vector>
#include <numeric>
#include "Stock.h"
using namespace std;
typedef vector<float> Vector;
typedef vector<Vector> Matrix;
ostream& operator<<(ostream& os, const vector<float>& v)
{
os << " ";
for (unsigned int i = 0; i < v.size(); i++) {
os << v[i];
if (i != v.size())
os << ", ";
}
os << "\n";
return os;
}
ostream& operator<<(ostream& os, const vector<string>& v)
{
os << " ";
for (unsigned int i = 0; i < v.size(); i++) {
os << v[i];
if (i != v.size())
os << ", ";
}
os << "\n";
return os;
}
vector<float> operator*(const vector<float>& v1, float s)
{
long z = v1.size();
vector<float> U(z);
for (long i = 0; i < z; i++) U[i] = s * v1[i];
return U;
}
vector<float> operator+(const vector<float>& v1, const vector<float>& v2)
{
long s = v1.size();
vector<float> U(s);
for (long i = 0; i < s; i++) U[i] = v1[i] + v2[i];
return U;
}
vector<float> operator-(const vector<float>& v1, float k)
{
long s = v1.size();
vector<float> U(s);
for (long i = 0; i < s; i++) U[i] = v1[i] - k;
return U;
}
vector<float> operator-(const vector<float>& v1, const vector<float>& v2)
{
long s = v1.size();
vector<float> U(s);
for (long i = 0; i < s; i++) U[i] = v1[i] - v2[i];
return U;
}
vector<float> operator/(const vector<float>& v1, long s)
{
long z = v1.size();
vector<float> U(z);
for (long i = 0; i < z; i++) U[i] = v1[i] / s;
return U;
}
vector<float> operator*(const vector<float>& v1, const vector<float>& v2)
{
long z = v1.size();
vector<float> U(z);
for (long i = 0; i < z; i++) U[i] = v2[i] * v1[i];
return U;
}

Vector operator*(const Matrix& C, const Vector& V)
{
int d = C.size();
Vector W(d);
for (int j = 0; j < d; j++)
{
W[j] = 0.0;
for (int l = 0; l < d; l++) W[j] = W[j] + C[j][l] * V[l];
}
return W;
}
typedef map<string, float> maps;
maps operator-(maps & m1,  maps &m2)
{
maps myMap;
for (maps::iterator it = m1.begin(); it != m1.end(); it++)
{
myMap[it->first] = m1[it->first] - m2[it->first];
}
return myMap;
}

float vector_average(const vector<float> v)
{
float average = 0;
average=accumulate(v.begin(), v.end(), 0.0 / v.size()) / v.size();
return average;
}
#endif
```

A.10 AddMissing Function

```
#pragma once
#ifndef AddMissing_h
#define AddMissing_h
#include "Stock.h"
#include "Trade.h"
#include <vector>
#include <numeric>
#include <limits>
#include <iostream>
using namespace std;

void fill_missing_return(map<string,Stock*> & sp500_Stock)//need more
{
Stock* mySPY = sp500_Stock["SPY"];
mySPY->cal_daily_return_map();
vector < Trade*> myTrades = mySPY->get_trades();
map<string, float> myTrades_return = mySPY->get_daily_return_map();
map<string,Stock*>  new_sp500_Stock;
for (map<string,Stock*>::iterator it = sp500_Stock.begin(); it != sp500_Stock.end(); ++it)
{
if ((it->second)->get_trades().size() == myTrades.size())
{
continue;
}
Stock* stock = it->second;
string begin_date = (stock->get_trades()[0])->get_date();
string end_date = stock->get_trades()[stock->get_trades().size() - 1]->get_date();
int position_1 = std::distance(std::begin(myTrades_return), myTrades_return.find(begin_date));
stock->cal_daily_return_map();
if(myTrades.end() - myTrades.begin() -position_1 ==(it->second)->get_trades().size())
continue;
for (vector<Trade*>::iterator itr = myTrades.begin() + position_1+1; itr != myTrades.end(); itr++)
{
string date = (*itr)->get_date();
if ((stock->get_daily_return_map()).count(date) == 0)
{
(stock->get_daily_return_map())[(*itr)->get_date()] = (stock->get_daily_return_map())[(*(itr - 1))->get_date()];
}
}
}
cout << "addmissing finished" << endl;
}
#endif
```