

# 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

Yuying Fan

May 2019

## Acknowledgements

I would like to express my gratitude to Professor Song Tang for his patience and insights. During the whole summer, he continues to check my progress and gives me guidance, which makes me feel encouraged and hopeful. I really appreciate the time he has spent on both the meeting and our questions. Also, I am very grateful that I have so many hardworking classmates and friends, with whom I could discuss my difficulty. I would also like to thank my department, the FRE department at NYU Tandon, which provides me with the opportunity to do the research. This project really makes me learn a lot and have the opportunity to develop my own program in Finance field.

Yuying Fan

May 2019

## ABSTRACT

This capstone project is to design a C++ program to use Artificial Intelligence Genetic Algorithm to select and optimize a SP500 investment portfolio. We try to find the best portfolio that could beat SPY both in the back testing and probation testing. In the program, we retrieve and parse market data, write into database, create the population, implement fitness function, do the crossover and mutation, and finally get the best portfolio after testing and comparing to the index. Nowadays it is popular to combine computer techniques with financial problems, such as selecting the best portfolio that could beat the index. In our project, we apply libcurl, json, and sqlite3, which are widely used software packages and applications. We build this platform using GA technique in order to help selecting portfolio by using past trading and fundamental data. It will improve the possibility to make money by using the strategy we have developed.

# Contents

<b><i>Introduction</i></b> .....	<b>1</b>
<b><i>Background</i></b> .....	<b>4</b>
<b>2.1 Portfolio Performance Measurement</b> .....	<b>4</b>
2.1.1 Sharpe Ratio .....	4
2.1.2 Fundamental Data.....	6
<b>2.2 Genetic Algorithm Techniques</b> .....	<b>7</b>
2.2.1 Generations.....	7
2.2.2 Parents Selection, Crossover and Mutation .....	8
<b><i>Data</i></b> .....	<b>10</b>
<b>3.1 Data Information</b> .....	<b>10</b>
3.1.1 Data Sources .....	10
3.1.2 Data Retrieval .....	12
<b>3.2 Database Design</b> .....	<b>15</b>
<b><i>Genetic Algorithm</i></b> .....	<b>16</b>
<b>4.1 Whole Logic</b> .....	<b>16</b>
4.1.1 Parameters .....	16
4.1.2 Steps .....	17
<b>4.2 Fitness Function</b> .....	<b>18</b>
<b>4.3 Crossover and Mutation</b> .....	<b>20</b>
<b>4.4 Different Classes</b> .....	<b>20</b>
4.4.1 Fundamental Class.....	20
4.4.2 Trade Class .....	21
4.4.3 Stock Class .....	21
4.4.4 Portfolio Class .....	21
<b><i>Back Testing and Probation Testing</i></b> .....	<b>23</b>
<b>5.1 Back Testing</b> .....	<b>23</b>
<b>5.2 Probation Testing</b> .....	<b>24</b>
<b><i>Performance Analysis</i></b> .....	<b>25</b>
<b>6.1 Improvements Among Generations</b> .....	<b>25</b>

<b>6.2 Returns Comparison.....</b>	<b>26</b>
<b><i>Conclusions and Future Work.....</i></b>	<b>29</b>
<b><i>Appendix: Code .....</i></b>	<b>31</b>
<b>A.1 Fundamental class.....</b>	<b>31</b>
<b>A.2 Trade class .....</b>	<b>32</b>
<b>A.3 Stock class .....</b>	<b>33</b>
<b>A.4 Portfolio class.....</b>	<b>35</b>
<b>A.5 Utility functions .....</b>	<b>39</b>
<b>A.6 GA functions .....</b>	<b>43</b>
<b>A.7 Retrieve data.....</b>	<b>45</b>
<b>A.8 Database .....</b>	<b>48</b>
<b>A.9 Main function .....</b>	<b>61</b>

# Chapter 1

## Introduction

This capstone project does the following several things coded in C++: it first sets market data retrieval using Unicorn data feed and parses market data in json format to store market data in SQLite database; then implements class trade, fundamental, stock, portfolio; develops Genetic Algorithm logic and designs fitness function; creates crossover and mutation methods; finally back tests performance of the selected portfolio using 2019 data and completes the probation testing. The program has a menu in order to complete different purposes. The program design is also shown in flow chart Figure 1.1.

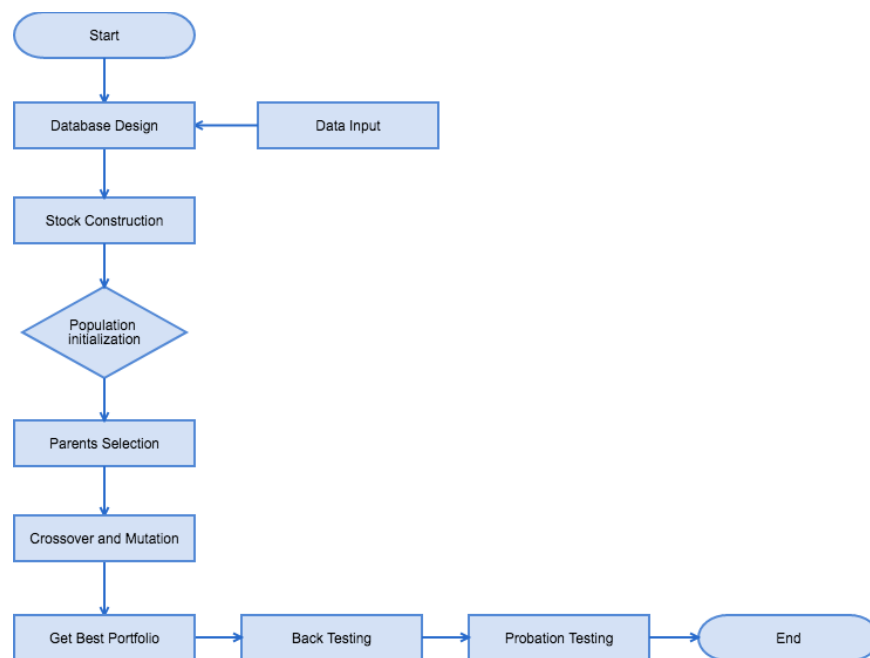


Figure 1.1 program design

The program is used to select best performance portfolio using genetic algorithms. I first build up the first generation's 100 portfolio, which consists of a population,

by randomly selecting 10 stocks from S&P500. Then I implement fitness function by assigning each portfolio a score according to their performance. After scoring, I perform selection on the portfolios, deciding parents for creating child portfolio and getting the next generation. I use the best 70% portfolios to do the crossover, mutation, and make their 70 children substitute the worst 70% portfolios. The best 30% portfolios will be kept through generations. In this way, I will get the next generation, which also includes 100 portfolios. During the mutation, I mutate a few of portfolios by randomly adding or removing a small number of stocks. The mutation rate I have used is 3%.

The fitness function is really important for the selection of best portfolio. I combine several portfolio performance measurements to create the score. I not only use the Sharpe ratio, which evaluates the portfolio manager on the basis of both rate of return and diversification, but also add the fundamental data, such as PE ratio, beta and the moving average.

In order to analysis the performance of the best portfolio I have selected, I first use back testing to compare the return with the SPY. I use the data before 2018.12 to do the GA selection, so I could use the first half year data of 2019 for the back testing. Typically, the best portfolio after 50 generations could averagely beat SPY. Then, I do the probation testing using the most recent data: July 2019. Finally, I could find the best portfolio that beats SPY averagely in the month, which means I could make money by holding the selected portfolio and short SPY.

The project process is organized in the following order. In Chapter 2: Background, we provide some backgrounds in portfolio performance measurements and genetic algorithm techniques we used. Chapter 3: Data, we describe how we retrieve, parse and store data into SQLite database. In Chapter 4: Genetic Algorithm, we describe our selection logic, how we choose the parents, crossover, and do the mutation. In

Chapter 5: Back Testing and Probation Testing, we explain how we measure the performance of our selected portfolio and how we modify our fitness function. In Chapter 6: Performance Analysis, we show performance of the best portfolio and the comparison with SPY. At last, we have Chapter 7: Conclusion and appendix where source codes will be provided.



# Chapter 2

## Background

### 2.1 Portfolio Performance Measurement

We need to combine different performance measurements in order to create the fitness function. During the portfolio selection, our objective is to maximize the return and minimize the risk. Therefore, we need to find indexes to measure returns and risk.

#### 2.1.1 Sharpe Ratio

There are several different performance measurements that we could use, such as Treynor Measure and Sharpe ratio. Treynor Measure is also called reward-to-volatility ratio. Typically, the higher the Treynor measure, the better the portfolio. The formula is  $(\text{Portfolio Return} - \text{Risk-Free Rate}) / \text{Beta}$ . However, this performance measure should really only be used by investors who hold diversified portfolios. It assumes that no diversifiable risk is present in the portfolio since it used beta instead of the total return measurement: sigma.

The total risk has two components: (1) Systematic Risk of a portfolio is defined as the inherent risk in the portfolio which cannot be diversified. It is measured as beta, relative to the market as a whole. (2) Unsystematic risk: This is the component of risk that can be diversified away.

Therefore, I decided to choose Sharpe ratio as one of the performance measurements. Unlike the Treynor measure, the Sharpe ratio evaluates the portfolio manager on the basis of both rate of return and diversification (as it considers total portfolio risk as measured by standard deviation in its denominator). Therefore, the Sharpe ratio more accurately takes into account the risks of the portfolio. The formula of Sharpe ratio is:

$$(\text{Portfolio Return} - \text{Risk-Free Rate}) / \text{Standard Deviation}$$

There are 3 parts we need to know in order to calculate Sharpe ratio.

- (1) Portfolio Return: The return of a portfolio is derived from the weighted average returns of the assets in the portfolio. For a portfolio with n number of assets, the portfolio returns are:

$$r_P = w_1r_1 + w_2r_2 + \dots + w_nr_n$$

- (2) Portfolio Risk: Total risk of the portfolio can be determined by its volatility, which is nothing but the standard deviation of its returns over a period of time. For n-period returns of a portfolio, volatility is:

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

- (3) Risk Free Rate: The yearly risk free rate have been retrieved, parsed and stored in table RiskFreeRate in my database. The only thing I need to do is to change it from yearly to daily, since the returns of stocks are daily.

In this way, I could calculate Sharpe ratio using the formula. First, I change the risk free rate from yearly to daily. Then, I calculate excess returns of portfolio for each day. Since I have already calculated sigma of the portfolio, I could get the Sharpe ratio for each day. After calling the function calculate\_average, I could get the Sharpe ratio for my portfolio. The code is shown in figure 2.1.

```

void set_sharpe()
{
    Map daily_rf_map;
    daily_rf_map.clear();
    for (Map::iterator itr=stocks[0].risk_free_rate.begin(); itr!=stocks[0].risk_free_rate.end(); itr++)
    {
        daily_rf_map[itr->first]=pow(1+itr->second, 1/252.0)-1;
    }
    Map excess_return=(p_returns-daily_rf_map);
    Map sharpe_ratio_map=excess_return/(p_sigma/sqrt(252));
    Sharpe= calculate_average(sharpe_ratio_map);
    cout<<"Sharpe:"<<Sharpe<<endl;
}

```

Figure 2.1 Set Sharpe ratio

## 2.1.2 Fundamental Data

I have retrieved fundamental data for each stock and save them in table Fundamentals in my database.

After searching on the internet for reference and after back testing, I found that for the PE ratio, both too big and too small are not so good. Then I set highest PE score for the portfolio whose PE is in the range (10, 30), and the lowest when PE is higher than 30.

For the moving average, I think if the 50-day moving average is higher than 200-day moving average, it could reflect that the price is increasing or tends to increase, because the short time average price is higher than the long time average. Therefore, I added 50MA/200MA to my fitness function. From back testing, I found that this strategy's result is pretty good.

## 2.2 Genetic Algorithm Techniques

### 2.2.1 Generations

Before the development of artificial intelligence, people are using different optimization algorithms to select the portfolio. For example, Harry Markowitz 1959 developed a quantitative model, also called mean-variance model. It could either maximize the return for a given level of risk or minimize the risk for a given level of return. However, the application of the mean-variance model has many limitations considering the bounding constraints. In order to satisfy the limitations imposed, some constrained optimization algorithms like Linear Programming has been developed. But they still have their drawbacks in portfolio selection, since these algorithms are based on linear assumptions that could be used to solve quadratic objective functions with a single objective. However, sometimes the objective function is not quadratic and may have more than one objective. Like in the portfolio selection process, we need not only to maximize the returns, but also minimize the risk at the same time. Therefore, people are trying to use genetic algorithm to solve this problem. The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. It can deal with nonlinear optimization problems with non-smooth and even non-continuous objective, and continuous and/or integer variables. Some applications of genetic algorithms to complex problems in financial markets include forecasting returns, portfolio optimization, trading rule discovery, and optimization of trading rules.

In order to do the GA selection, we need to have a population to store our chromosome, so we first create a random population, all with zero fitness. Then we

set a flag to show whether a solution is found. If not, we go into a while loop where we create a new population. In the while loop, we first test and update the fitness of each chromosome in the population. Then we check and see whether we have found any solution according to fitness score. If yes, break; If not, we sort all the chromosomes in a population in ascending order. After this, we create a new population by selecting two parents at a time and creating offspring by applying crossover and mutation. Do this until the desired number of offspring have been created. Then we could use the children to replace the old chromosome. Finally, we need to check whether the generations we have created have exceed the max generation allowed. If so, we need to stop and print the best portfolio we have got so far.

### **2.2.2 Parents Selection, Crossover and Mutation**

There are typically two ways to select the parents. The first way is roulette wheel selection. It is quite similar to a roulette wheel, where the likelihood that an individual is chosen is proportional to its fitness score. Since the fitness score is better to be higher, the size of each individual's slice of the roulette wheel is positively proportional to its fitness score. Once the "slices" have been determined, a number is generated at random. The individual with the range of numbers that contains this randomly generated number will be one parent. This continues until the desired number of parents is found. This method also allows for parents with low fitness score to go to the next generation. The second popular way is elitism. Elitism involves copying a small proportion of the fittest candidates, unchanged, into the next generation. This can sometimes have a dramatic impact on performance by ensuring that the algorithm does not waste time re-discovering previously discarded partial solutions. Candidate solutions that are preserved unchanged through elitism

remain eligible for selection as parents when breeding the remainder of the next generation.

```
void Crossover(Portfolio &offspring1, Portfolio &offspring2)
{
    //dependent on the crossover rate
    // if (RANDOM_NUM < CROSSOVER_RATE)
    {
        //create a random crossover point
        srand((int)time(NULL));
        int crossover = (int)(RANDOM_NUM * CHROMO_LENGTH);

        vector <Stock> stocks1=offspring1.stocks;
        vector <Stock> stocks2=offspring2.stocks;
        vector <Stock> new_stock1;
        vector <Stock> new_stock2;
        for (int i=0; i<crossover; i++)
        {
            new_stock1.push_back(stocks1[i]);
            new_stock2.push_back(stocks2[i]);
        }
        for (int j=crossover; j<10; j++)
        {
            new_stock1.push_back(stocks2[j]);
            new_stock2.push_back(stocks1[j]);
        }
        Portfolio t1(new_stock1);
        Portfolio t2(new_stock2);
        offspring1 = t1; offspring2 = t2;
    }
}
```

Figure 2.2 crossover function

After selecting the parents, the chromosomes that have been chosen as parents will be recombined to form children for the next generation. This process is called crossover. As shown in figure 2.2, I first create a random crossover point. Then I put the first part of the first parent and the second part of the second parent together to create the first child. The second child is created similarly.

The operation of mutation allows new individual to be created. In the mutation function, I first generate a random number between 0 and 1, if the random number is smaller than the mutation rate, which is 0.03, then the portfolio will go through mutation. We will randomly replace 3 stocks with 3 randomly selected new stock from SP500. The alternate portfolio is then copied in to the next generation of the population.

# Chapter 3

## Data

### 3.1 Data Information

#### 3.1.1 Data Sources

##### 1. S&P 500 company information

S&P500 constituent data in excel format, which includes name of company, symbol, market weight, sector and shares held, shown in figure 3.1. The market weights here are used to decide the stock weights in a particular portfolio.

Name	Identifier	Weight	Sector	Shares Held
Microsoft Corporation	MSFT	4.256837	Information Technology	85841100.000
Apple Inc.	AAPL	3.539644	Information Technology	48965524.000
Amazon.com Inc.	AMZN	3.321625	Consumer Discretionary	4632684.000
Facebook Inc. Class A	FB	1.940564	Communication Services	26914088.000
Berkshire Hathaway Inc. Class B	BRK.B	1.657294	Financials	21697908.000
Johnson & Johnson	JNJ	1.493156	Health Care	29742706.000
JPMorgan Chase & Co.	JPM	1.485670	Financials	36339690.000
Alphabet Inc. Class C	GOOG	1.407548	Communication Services	3433223.000
Alphabet Inc. Class A	GOOGL	1.375127	Communication Services	3354523.000
Exxon Mobil Corporation	XOM	1.317375	Energy	47398000.000
Visa Inc. Class A	V	1.261569	Information Technology	19480548.000
Procter & Gamble Company	PG	1.151588	Consumer Staples	28099044.000
Bank of America Corp	BAC	1.042077	Financials	99057680.000

Figure 3.1 SP500 company information

##### 2. market data

Stock market data from eodhistoricaldata are in json format, shown in figure 3.2 includes symbol, date, open, high, low, close, adjusted close, volume from 2008-01-01 to recent, where only date and adjusted close price are used in our genetic algorithm program.

### 3. stock fundamental data

```
{ "General": { "Code": "AAPL", "Type": "Common Stock", "Name": "Apple Inc", "Exchange": "NASDAQ", "CurrencyCode": "USD", "CurrencySymbol": "$", "CountryName": "USA", "CountryISO": "US", "ISIN": "US0378331005", "CUSIP": "0378331005", "FiscalYearEnd": "September", "IPODate": "1980-12-12", "InternationalDomestic": "International/Domestic", "GicSector": "Information Technology", "GicGroup": "Technology Hardware & Equipment", "GicIndustry": "Hardware, Storage & Peripherals", "Description": "Apple Inc. designs, manufactures, and markets mobile communication devices, related software, services, accessories, and third-party digital content and applications. The company offers iPhone, iPad, and Mac, a line of desktop and portable personal computers, as well as iOS, macOS, watchOS, and tvOS operating systems for customers to purchase and download, or stream music and TV shows; rent or purchase movies; and download free apps, contacts, calendars, mail, documents, and others. In addition, the company offers AppleCare support service contracts for consumers' TVs and enables them to access digital content directly for streaming video, playing music and podcasts, as well as AirPods, Beats products, HomePod, iPod touch, and other Apple-branded and third-party accessories. The company also provides education, enterprise, and government customers worldwide. It sells and delivers digital content and applications through its App Store, Book Store, and Apple Music. The company also sells its products through its retail and online stores, and franchisees, and resellers. Apple Inc. was founded in 1977 and is headquartered in Cupertino, California.", "Website": "http://www.apple.com", "WebURL": "http://www.apple.com", "LogoURL": "\\img\\logos\\US\\aapl.png", "FullTimeEmployees": 100000, "UpdatedAt": "2019-06-14T12:00:00Z", "MarketCapitalization": 873738207232, "MarketCapitalizationMln": 873738.2072, "EBITDA": 76544999424, "PERatio": 35.21, "EPS": 21.86, "DividendShare": 3.08, "DividendYield": 0.0148, "EarningsShare": 11.777, "EPSEstimateCurrentQuarter": null, "EPSEstimateNextQuarter": null, "MostRecentQuarter": "2019-06-01", "LastFiscalYear": "2018-09-30", "LastFiscalPeriod": "Q3", "LastFiscalReportDate": "2018-10-01", "LastFiscalReportPeriod": "Q3", "LastFiscalReportType": "Annual", "LastFiscalReportSize": 1000000, "LastFiscalReportUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportTitle": "Form 10-K Annual Report", "LastFiscalReportText": "Form 10-K Annual Report", "LastFiscalReportHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentType": "Form 10-K", "LastFiscalReportDocumentSize": 1000000, "LastFiscalReportDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentAudio": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentVideo": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocument": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentType": "Form 10-K", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentSize": 1000000, "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentUrl": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentText": "Form 10-K Annual Report", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentHtml": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentPdf": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentDocumentImage": "https://investor.apple.com/sec-filings/default.aspx?filenum=1447747", "LastFiscalReportDocumentDocumentDocumentDocumentDocument
```

4. risk free rate

Risk free rate could be retrieved from the `eodhistoricaldata`. I store risk free rate in a map with date as key, so that I could calculate sharpe ratio for each stock.



### 3.1.2 Data Retrieval

We first put SP500 constituent data into database, then retrieve data for each ticker from SP500 and also SP500 index price.

Function `PopulateNewSP500Table` is used to read data from csv. The `getline` function could read data in csv cell by cell and line by line. In this way, I could retrieve symbol, name, sector, weight, shares and store them into the table `NEWSP500` in my database using the function `InsertTable`. I will use the symbol to retrieve price information for each stock and use weight to set the stock weights in portfolio.

This function is shown in figure 3.4. And my table `NEWSP500` is shown in figure 3.5.

```

601 int PopulateNewSP500Table(sqlite3 *db, vector<string> &symbols, vector<string> &weights)
602 {
603     fstream fin;
604     fin.open("new_SP500.csv", ios::in);
605     string title, name, symbol, weight, sector, shares;
606     getline(fin, title);
607     cout<<title<<endl;
608     int count=0;
609     while (fin.good())
610     {
611         getline(fin, name, ',');
612         cout<<name<<endl;
613         getline(fin, symbol, ',');
614         cout<<symbol<<endl;
615         getline(fin, weight, ',');
616         cout<<weight<<endl;
617         getline(fin, sector, ',');
618         cout<<sector<<endl;
619         getline(fin, shares);
620         cout<<shares<<endl;
621         count++;
622         symbols.push_back(symbol);
623         weights.push_back(weight);
624
625         // Execute SQL
626         char new_sp500_insert_table[512];
627         sprintf(new_sp500_insert_table, "INSERT INTO NEWSP500 (id, symbol, name, sector, weight, shares) VALUES(%d, \"%s\", \"%s\", \"%s\", \"%s\", \"%s\")", count, symbol.c_str(), name.c_str(), sector.c_str(), weight.c_str(), shares.c_str());
628         if (InsertTable(new_sp500_insert_table, db) == -1)
629             return -1;
630     }
631     return 0;
632 }

```





Figure 3.4 Function `PopulateNewSP500Table`

Table: <input type="text" value="NEWSP500"/>						
	id	symbol	name	sector	weight	shares
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	MSFT	Microsoft Corpo...	Information Tec...	4.256837	85841100.000
2	2	AAPL	Apple Inc.	Information Tec...	3.539644	48965524.000
3	3	AMZN	Amazon.com Inc.	Consumer Discr...	3.321625	4632684.000
4	4	FB	Facebook Inc. C...	Communication...	1.940564	26914088.000
5	5	BRK.B	Berkshire Hatha...	Financials	1.657294	21697908.000
6	6	JNJ	Johnson & Johns...	Health Care	1.493156	29742706.000
7	7	JPM	JPMorgan Chase...	Financials	1.485670	36339690.000
8	8	GOOG	Alphabet Inc. Cl...	Communication...	1.407548	3433223.000
9	9	GOOGL	Alphabet Inc. Cl...	Communication...	1.375127	3354523.000
10	10	XOM	Exxon Mobil Co...	Energy	1.317375	47398000.000

Figure 3.5 table NEWSP500

Function `PopulateStockTable` is used to parse Json format and retrieve the date, volume, open, close, high, low, adjusted close data. `Trade` is a class that have private members like adjusted close. `Stock` is also a class with private members like symbol, prop, which stores the weight of each stock, and vector of `Trade`, which stores different dates' price information for each stock. After retrieving the data, I could execute SQL and store each stock's price information in table `Stock_symbol` in my database by using `InsertTable` function. For example, table `STOCK_AAPL` is shown in figure 3.6, and this function is shown in figure 3.7.

Table:



New Record

Delete Record

	symbol	date	open	high	low	close	adjusted_close	volume
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	AAPL	2008-01-02	199.270004	200.259995	192.550003	194.839996	24.3836	38542020
2	AAPL	2008-01-03	195.410004	197.389999	192.690002	194.929993	24.394899	30073780
3	AAPL	2008-01-04	191.449997	193.0	178.889999	180.050003	22.5327	51993904
4	AAPL	2008-01-07	181.25	183.600006	170.229996	177.639999	22.2311	74006846
5	AAPL	2008-01-08	180.139999	182.460007	170.800003	171.25	21.4314	54421984
6	AAPL	2008-01-09	171.300003	179.5	168.300003	179.399994	22.451401	64840673
7	AAPL	2008-01-10	177.580002	181.0	175.410004	178.020004	22.278601	52963350
8	AAPL	2008-01-11	176.0	177.850006	170.0	172.690002	21.611601	44010111
9	AAPL	2008-01-14	177.520004	179.419998	175.169998	178.779999	22.3738	39301774
10	AAPL	2008-01-15	177.720001	179.220001	164.660004	169.039993	21.1548	83937922

Figure 3.6 table STOCK\_AAPL

```

15 int PopulateStockTable(const Json::Value & root, string symbol, sqlite3 *db)
16 {
17     string date;
18     float open = 0, high = 0, low = 0, close = 0, adjusted_close = 0;
19     unsigned int volume = 0;
20     Stock myStock(symbol);
21     int count = 0;
22
23     for (Json::Value::const_iterator itr = root.begin(); itr != root.end(); itr++)
24     {
25         cout << *itr << endl;
26         for (Json::Value::const_iterator inner = (*itr).begin(); inner != (*itr).end(); inner++)
27         {
28             //cout << inner.key() << ": " << *inner << endl;
29
30             if (inner.key().asString() == "adjusted_close")
31                 adjusted_close = inner->asFloat();
32             else if (inner.key().asString() == "close")
33                 close = inner->asFloat();
34             else if (inner.key() == "date")
35                 date = inner->asString();
36             else if (inner.key().asString() == "high")
37                 high = inner->asFloat();
38             else if (inner.key().asString() == "low")
39                 low = inner->asFloat();
40             else if (inner.key() == "open")
41                 open = inner->asFloat();
42             else if (inner.key().asString() == "volume")
43                 volume = inner->asUInt();
44             else
45             {
46                 cout << "Invalid json field" << endl;
47                 return -1;
48             }
49         }
50         Trade aTrade(date, open, high, low, close, adjusted_close, volume);
51         myStock.addTrade(aTrade);
52         count++;
53
54         // Execute SQL

```

Figure 3.7 Function PopulateStockTable

When retrieving data, I found that 5 stocks in the SP500 only have data for 2019. These stocks are “DOW”, “CTVA”, “AMCR”, “FOXA”, and “CPRI”. I delete these stocks, since I don’t have enough data to do the GA and back testing. For the stocks that miss only a few data, like several months, I keep these stocks because I want to keep as many stocks as I could. In this way, several stocks may have less price data and less daily returns. Therefore, I change the data type of daily\_return from vector

to map with date as key in order to match different stocks' returns. Then I could do the calculations easier.

## 3.2 Database Design

Our database design comprises of 511 tables with two entity relationship, as shown in Figure 3.8. First, each of 506 stock table with “symbol” and “date” as primary key and “symbol” as foreign key maps to “symbol” as primary key from NewSP500 table. Second, both table Fundamentals and table NewSP500 has primary key “symbol”.

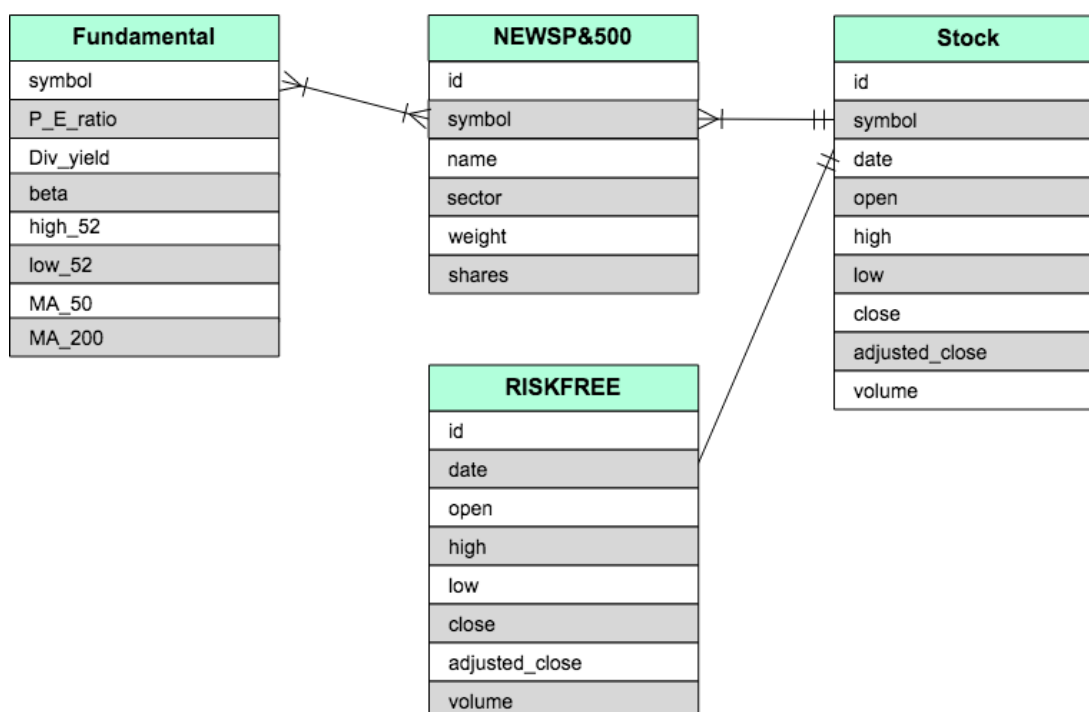


Figure 3.8 Database design

# Chapter 4

## Genetic Algorithm

### 4.1 Whole Logic

#### 4.1.1 Parameters

- Population size: 100
- Max number of generations: 1000
- Probability to have a mutation: 0.03 (that is a probability of 3%)
- Crossover rate: 0.7
- Stock in a portfolio: 10

In this program, the gene is the stock, and the chromosome is portfolio. Population size is 100, which means each population has 100 chromosomes, so each population have 100 portfolios. Stock in a portfolio is 10, so to generate one portfolio, I randomly selected 10 different stocks from SP500. Crossover rate is 0.7, which means 70% of the population, that is 70 portfolios could have their children. The 70 children will then replace the worst 70% population in the generation before. Probability to have a mutation is 0.03, so each offspring has a 3% probability to have a mutation. The max number of generations could not be more than 1000, and typically, I run GA for less than 80 generations to get my best portfolio.

The whole logic could be seen more clearly in the following figure.

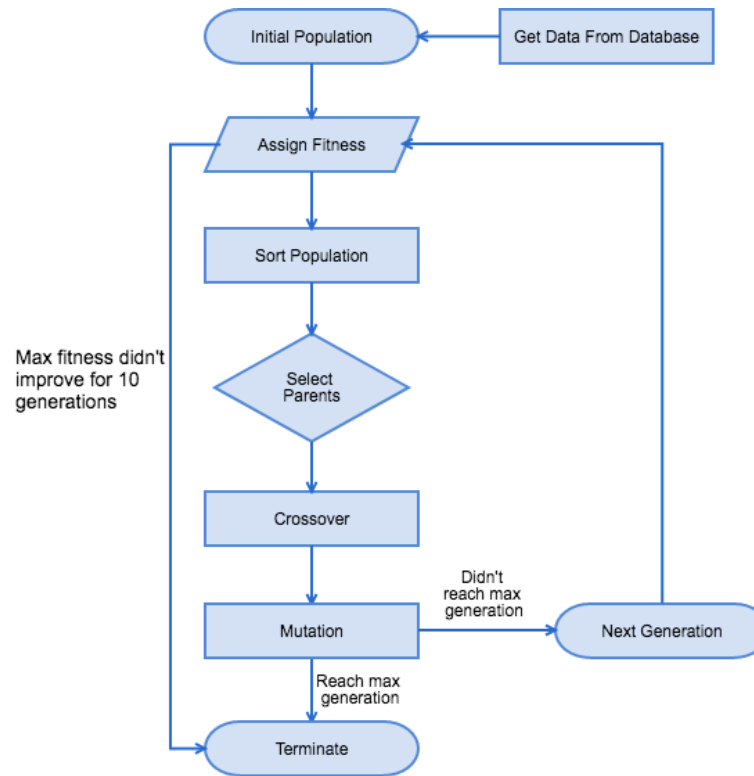


Figure 4.1 GA logic

### 4.1.2 Steps

1. I get the data I need from my database, and store into some data structure. I get the symbols from NEWSP500 table using function GetSymbols. After that, I get fundamental data using stock\_fundamentals and price information using function stock\_prices for each stock. After calling the member function of class Stock named calculate\_returns, I store all the stocks information in a vector of Stock named total\_stocks. Also, I get the weights I stored in table NEWSP500 and set the weights for each stock. For GA part, I only use the data from 2008-01-01 to 2018-12-31 and save the recent data for future testing.

2. Population is a vector of Portfolio and it has 100 portfolios. For each portfolio, I get 10 random numbers and select these 10 stocks from total\_stocks. After

calculating weights for each portfolio using the prop of each stock, I get the first generation with initial fitness as 0.0.

3. I set the fitness score for each portfolio in the first generation using Assignfitness, which is a public member of class Portfolio, and sort the population according to fitness score.

4. I use Roulette to choose 2 parents at a time from the best 70%, let them crossover and generate 2 children. Therefore, the 70 best portfolios could generate 70 offspring. After doing the mutation, they will replace the worst 70% population in the generation before. In this way, the population size will always be 100 and the best 30% could go to the next generation.

5. When the highest fitness score in a population doesn't increase for 10 generations, or when number of generations is bigger than the max allowable generations, the GA will stop and print the best portfolio. Also, I write some code to store the best portfolio into table BestPortfolio in my database.

## 4.2 Fitness Function

```
void Assignfitness()
{
    set_portfolio_returns();
    set_portfolio_sigma();
    set_P_E();
    set_sharpe();
    set_p_beta();
    set_p_div();
    set_ma();
    float pe_score=0.0;
    if (p_pe<30.0&& p_pe>10)
        pe_score=5;
    else if (p_pe>=30)
        pe_score=-5;
    fitness=60*Sharpe+20*p_beta+30*p_div+8*pe_score+8*ma_50/ma_200;
    cout<<"fitness:"<<fitness<<endl;
}
```

Figure 4.2 fitness function

Assignfitness is a member function of class Portfolio. In this function, I first set portfolio returns. This function is shown in figure 4.3, I get the individual returns for each stock and calculate the weighted average return for portfolio. Since each day have one return, I use the function calculate\_average to get the average return. Then I calculate sigma for portfolio, as shown in figure 4.4.

```
void set_portfolio_returns()
{
    p_returns.clear();
    for (int i=0; i<stocks.size(); i++)
    {
        map<string,float> individual_returns=stocks[i].get_returns();
        for (map<string,float>::iterator itr=individual_returns.begin(); itr!=individual_returns.end();itr++)
        {
            p_returns[itr->first]=p_returns[itr->first]+itr->second*weights[i];
        }
    }
    average_return=calculate_average(p_returns);
}
.. .. .
```

Figure 4.3 portfolio returns

```
void set_portfolio_sigma()
{
    float average=calculate_average(p_returns);
    map<string,float> dif2;
    for (map<string,float>::iterator itr=p_returns.begin(); itr!=p_returns.end();itr++)
    {
        dif2[itr->first]=(itr->second-average)*(itr->second-average);
    }
    p_sigma=sqrt(calculate_average(dif2));
    cout<<"sigma:"<<p_sigma<<endl;
}
-- .. .. .
```

Figure 4.4 portfolio sigma

After getting return and sigma for the portfolio, I could use the formula I have shown in Chapter 2 to calculate Sharpe ratio. Also, I set the fundamental data like PE ratio for portfolio by getting the fundamental data for each stock and calculate weighted average. I found that for PE ratio, either too big or too small is not good, so I set PE score according to PE ratio. If  $p\_pe < 30.0$  &  $p\_pe > 10$ , I set PE score as 5. If  $p\_pe$  is bigger or equal to 30, I set the score as -5. Otherwise, the score will be 0.



I have tried different fitness functions to compare their results through back testing, and find that it could be better to include moving average to fitness function and set it as  $60 * \text{Sharpe} + 20 * p\_beta + 30 * p\_div + 8 * pe\_score + 6 * ma\_50 / ma\_200$ .

## 4.3 Crossover and Mutation

During my crossover, I first generate a random number between 0 and the chromosome length, which is 10. In this way, I could decide at which point I will cut the chromosome. Then, I will cut the parent chromosomes into two parts from that point. After cutting, I will combine the first part of the first parent chromosome with the second part of the second parent chromosome. Similarly, I could combine the second part of the first parent chromosome with the first part of the second parent. Finally, I could get the two children chromosomes, which will substitute parent chromosomes in the future.

During my mutation, I mutate a few of the portfolios by randomly adding or removing a small number of stocks. For each portfolio, if the random number between 0 and 1 is smaller than 0.03, it will go through mutation. Then the stocks in that portfolio will be replaced by randomly selected new stocks from SP500.

After crossover and mutation, I will check whether the portfolio has duplicated stocks. If so, I will replace the duplicated one with another randomly selected stock.

## 4.4 Different Classes

### 4.4.1 Fundamental Class

The fundamental class is used to aggregate the fundamental data I may use in the future. This class only contains basic fundamental information for stocks, so its

private members include PE ratio, dividend yield, beta, high and low 52 weeks, 50-day and 200-day moving average. The structure of this class is in figure 4.5. The public members are used to get access to private members and overload the cout for fundamental class.

#### **4.4.2 Trade Class**

The trade class is used to put the price information together. It helps to aggregate each day's different prices for stocks. The private members include date, volume and different prices. For each stock, we will have a vector of Trade, which will contain different days' price information for that stock. The public members are only used to access private members and print price information.

#### **4.4.3 Stock Class**

The stock class could contain all information for a stock, including its symbol, weight, fundamental information, price information, and daily returns. The fundamental information is stored in class Fundamental, and the price information is stored in a vector of Trade, which contains price data for each date. The public member calculate\_returns is used to calculate each day's return using price data. The returns will be stored in a map with date as its key.

#### **4.4.4 Portfolio Class**

The portfolio class contains information for a portfolio, including the 10 stocks it contains, each stock's weight in this portfolio, each day's portfolio return, portfolio sigma, Sharpe ratio, PE ratio, beta, dividend yield, moving average and fitness. The

weights of each stock in the portfolio is calculated using the weights information stored in class Stock. Portfolio returns are stored in a map, and it is the weighted average of each stock's daily returns. Portfolio sigma and Sharpe ratio are calculated using the formula I have introduced. Other portfolio's fundamental data like beta are weighted average of the 10 stocks' fundamental data. The fitness score is  $60 * \text{Sharpe} + 20 * p\_beta + 8 * pe\_score + 8 * ma\_50 / ma\_200$ .

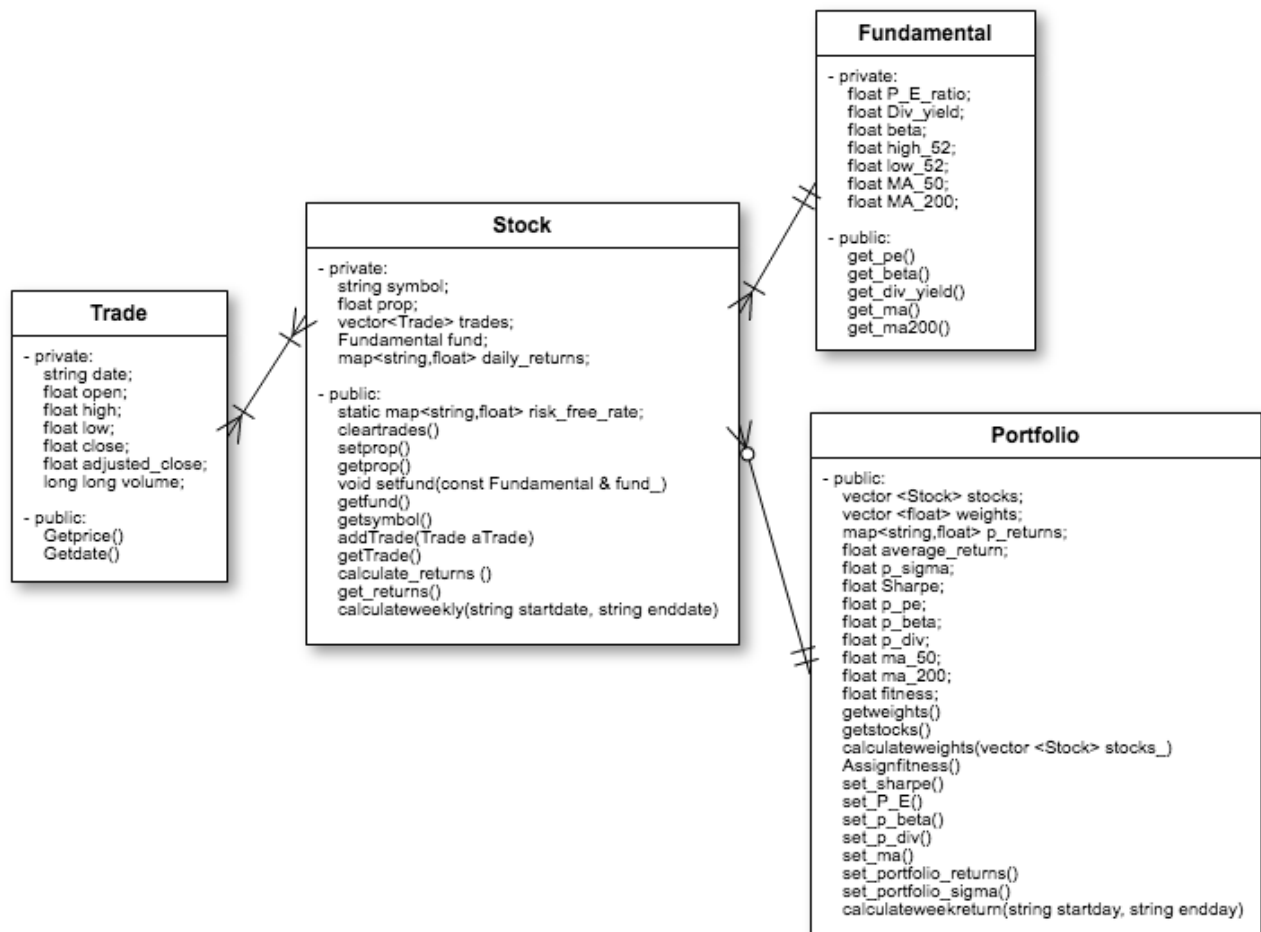


Figure 4.5 classes relationship

## Chapter 5

# Back Testing and Probation Testing

### 5.1 Back Testing

For back testing, start Jan 1, 2019, every Monday, I purchase the same amount for my best portfolio(s) as well as the index SPY, then sell both of them on Friday. Repeat it for a month. For each week, it is similar to compare weekly return of my best portfolio and SPY. For each month, I just need to add the weekly returns.

I use a pointer named pBest to point to the location of the best portfolio I have got through GA selection. In the backtesting part, I need to use the data from 2019-01-01 to 2019-06-31 to test the result. Therefore, I retrieve the data in that time period from my database use function stock\_prices for each of the 10 stocks in my best portfolio. Then I use the new period's data to replace the old trade data for each stock.

In the class Portfolio, I have a member function called calculateweekreturn. Once I give it one start date and one end date, it could calculate this period's return for a particular portfolio. As shown in figure 5.1, it first gets each stock's adjusted close price for both start day and end day. Then it could calculate weighted average price for the portfolio for these two days. Finally, it is easy to get the weekly return.

In this way, I calculate the weekly, monthly and 6 months accumulated return for my best portfolio and write into a csv file. Then I could compare with SPY and see whether I could beat SPY.

```

float calculateweekreturn(string startday, string endday)
{
    float weeklyreturn;
    float sum1=0.0;
    float sum2=0.0;
    float temp1=0.0;
    float temp2=0.0;
    for (int i=0; i<stocks.size(); i++)
    {
        for (int j = 0; j<stocks[i].getTrade().size(); j++)
        {
            if (stocks[i].getTrade()[j].Getdate()==startday)
            {
                temp1=stocks[i].getTrade()[j].Getprice();
                sum1=sum1+temp1*weights[i];
            }
            if(stocks[i].getTrade()[j].Getdate()==endday)
            {
                temp2=stocks[i].getTrade()[j].Getprice();
                sum2=sum2+temp2*weights[i];
            }
        }
    }
    weeklyreturn=(sum2-sum1)/sum1;
    return weeklyreturn;
}

```

Figure 5.1 calculate weekly return

## 5.2 Probation Testing

Probation testing is similar to back testing, except that the dates used are different. Probation testing uses the data between July 1<sup>st</sup>, 2019 to July 31<sup>st</sup>, 2019, which is the most recent data. In the one month, we also buy same amount best portfolio and SPY on every Monday, then sell it on Friday. We could compare the money we could make in each week. Also, we could sum them up and compare the money we could make in the whole month averagely.

## Chapter 6

### Performance Analysis

#### 6.1 Improvements Among Generations

In each generation, I printed the highest fitness score for the population and write into a file. Then I could create a chart to see the improvement clearly and make sure that the fitness score has some improvement among generations. The highest fitness score in each generation is in figure 6.1. It could show the growth path of my highest fitness score. We could see that the fitness score increases among generations. And after about 30 generations, the highest fitness score increases slower.

After trying lots of times, I found that typically, the fitness score stops improving after around 70 generations. Therefore, in order to save running time, I set the max allowable generation to 80. Also, I write some code to make sure the GA could stop if the highest fitness score stays the same for 10 generations.

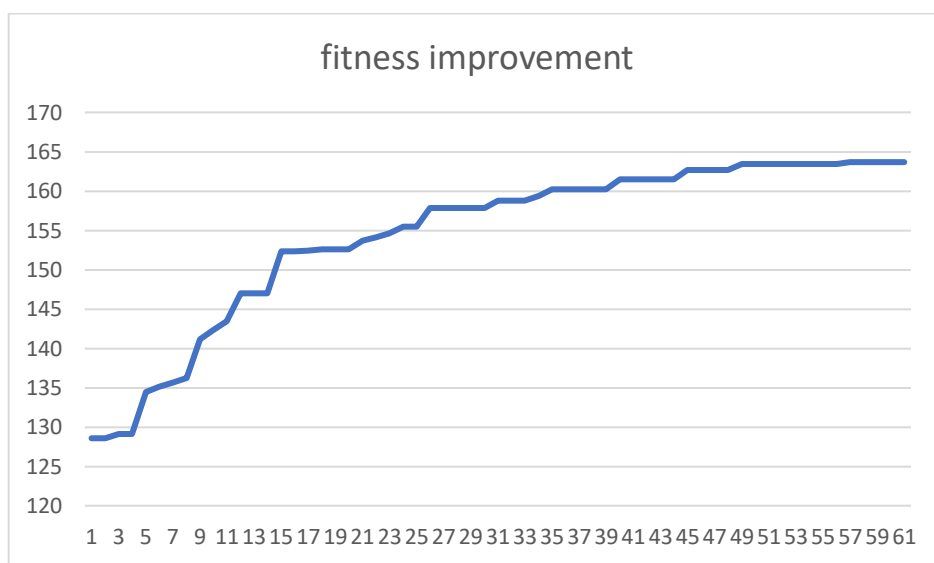


Figure 6.1 fitness improvement

## 6.2 Returns Comparison

After getting the best portfolio, I write the portfolio's composite stocks and weights to my database in order to keep a record. Figure 6.2 is one of my best portfolios, and it is stored in the table named BestPortfolio in my database.

Table: BestPortfolio

	id	stockname	weight
	Filter	Filter	Filter
1	1	AGN	0.131963
2	2	APTV	0.052295
3	3	AVGO	0.281986
4	4	DLTR	0.067801
5	5	EW	0.104924
6	6	IT	0.038882
7	7	LYB	0.066329
8	8	PSX	0.108022
9	9	SHW	0.097829
10	10	ULTA	0.049969

Figure 6.2 table best portfolio

After getting the best portfolio, I do the back testing to see the performance of the portfolio and compare it to SPY. First, I get the weekly return of my best portfolio and SPY, write them into a csv file, and create a chart of the weekly returns. Figure 6.3 shows the weekly returns of both my best portfolio and SPY.

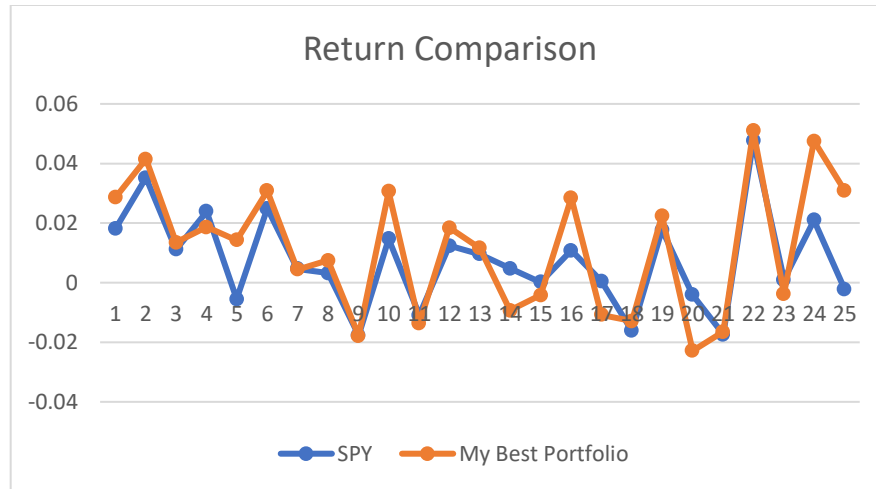


Figure 6.3 return comparison with SPY

Although it could not beat SPY for every week, it beats SPY 6 months averagely. We could see it from the following figure 6.4, which shows the monthly PnL of my best portfolio and the index. Also, I calculate the 6 months total PnL for them by summing up the monthly PnL. We could see that my best portfolio beats SPY averagely for the 6 months.

	My Best Portfolio	SPY	
first month	102155	88275.2	beat
second month	56982.9	27326.8	beat
third month	17584.1	-1504.08	beat
fourth month	26355.7	25320.5	beat
fifth month	-40641	-19382.2	
sixth month	125691	67404	beat
6 months total	288127	187440.22	beat

Figure 6.4 monthly PnL

After finishing the back testing, I do the probation testing using the July 2019 data. Similarly, I calculate the weekly returns and sum them up to get the monthly return. The following figure 6.5 could show the weekly PnL of my best portfolio and SPY,



and also the whole month's profit. We could see that my best portfolio beats SPY averagely.

	My Best Portfolio	SPY	
first week	19105.1	9470.3	beat
second week	19402.6	12903.4	beat
third week	-12105.5	-11903.5	
fourth week	1218.34	13796.6	
Total profit	27620.5	24266.8	beat

Figure 6.5 weekly PnL for July

## Chapter 7

### Conclusions and Future Work

This capstone project builds a C++ program that help us select the best portfolio that could beat SPY averagely during back testing and probation testing. Therefore, we could use this program to select the best portfolio, buy same amount of our portfolio and SPY on every Monday and sell them on Fridays. Finally, we could make money by executing this strategy. This strategy may not ensure that we could make money (beat SPY) every week consistently, but we could make money averagely for the six months in back testing and beat SPY averagely in July during probation testing.

This GA selection program is a good tool to select the best portfolio. However, it still has some limitations. First, we could not make sure that every time, the selected portfolio could beat SPY consistently. I think it is because that the stock market is not stable and 100 percent predictable. There are many factors that could affect the performance of a particular stock, like political factors, economics factors and even individuals' confidence. Also, the stock price could fluctuate from time to time. Even the price of big companies' stock, like Amazon or Apple could not increase consistently. Second, our strategy only includes the "buy" strategy, but not the "sell". But in real time, we could not only buy stocks, but also could short sell some of them. Third, we didn't consider the transaction cost or the taxes, that we must pay in the real world.

There are also ways to improve our program and make it have better results. First, there are some other factors that could be added to fitness function. I currently choose some of the important fundamental information and combines them with

Sharpe ratio. We could use other performance measurements like alpha instead of Sharpe ratio, or include more measurements into fitness function. Also, the fitness function has some parameters that could be modified and optimized. Second, we currently use the market share as the weights for the stocks, but there are other ways to decide the stocks' weights in a portfolio. Third, we could further diversify the portfolio by making sure that each stock is from a different industry. Also, we have 10 stocks in each portfolio now, but we could use GA for a larger number of portfolio compositions. Fourth, we could include the short sell strategy in our program, since they could be used in the real world. Fifth, we could consider the costs of transaction and the taxes that may need to pay.

# Appendix A

## Appendix: Code

### A.1 Fundamental class

```

#ifndef Fundamental_hpp
#define Fundamental_hpp

#include <iostream>
using namespace std;

class Fundamental
{
private:
    float P_E_ratio;
    float Div_yield;
    float beta;
    float high_52;
    float low_52;
    float MA_50;
    float MA_200;
public:
    Fundamental(float P_E_ratio_, float Div_yield_, float beta_, float high_52_, float low_52_, float
MA_50_, float MA_200_) : P_E_ratio(P_E_ratio_), Div_yield(Div_yield_), beta(beta_),
high_52(high_52_), low_52(low_52_), MA_50(MA_50_), MA_200(MA_200_) {}
    Fundamental() {}
    ~Fundamental() {}
    float get_pe()
    {
        return P_E_ratio;
    }
    float get_beta()
    {
        return beta;
    }
    float get_div_yield()
    {
        return Div_yield;
    }
    float get_ma()
    {
        return MA_50;
    }
    float get_ma200()

```

```

    {
        return MA_200;
    }
    friend ostream & operator << (ostream & out, const Fundamental & f)
    {
        out << "P/E Ratio:" << f.P_E_ratio << " Dividend Yield:" << f.Div_yield << " Beta:" << f.beta << "
High 52 Weeks:" << f.high_52 << " Low 52 Weeks:" << f.low_52 << " MA 50 Days:" << f.MA_50 << "
MA 200 Days:" << f.MA_200 << endl;
        return out;
    }
};

#endif /* Fundamental_hpp */

```

## A.2 Trade class

```

#ifndef Trade_hpp
#define Trade_hpp

#include <iostream>
using namespace std;

class Trade
{
private:
    string date;
    float open;
    float high;
    float low;
    float close;
    float adjusted_close;
    long long volume;
public:
    Trade(string date_, float open_, float high_, float low_, float close_, float adjusted_close_, long long
volume_) :
        date(date_), open(open_), high(high_), low(low_), close(close_), adjusted_close(adjusted_close_),
volume(volume_)
    {}
    ~Trade() {}
    float Getprice()
    {
        return adjusted_close;
    }
    string Getdate()
    {

```

```

        return date;
    }
    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 << endl;
        return out;
    }
};

#endif /* Trade_hpp */

```

## A.3 Stock class

```

#ifndef Stock_hpp
#define Stock_hpp

#include <iostream>
#include <vector>
#include <map>
#include "Fundamental.hpp"
#include "Trade.hpp"
using namespace std;

class Stock
{
private:
    string symbol;
    float prop;
    vector<Trade> trades;
    Fundamental fund;
    map<string,float> daily_returns;

public:
    static map<string,float> risk_free_rate;
    Stock(string symbol_) :symbol(symbol_)
    {}
    ~Stock() {}
    void cleartrades()
    {
        trades.clear();
    }
    void setprop(float prop_)
    {
        prop=prop_;
    }
}

```

```

float getprop()
{
    return prop;
}
void setfund(const Fundamental & fund_)
{
    fund=fund_;
}
Fundamental getfund()
{
    //cout<<fund<<endl;
    return fund;
}
string getsymbol()
{
    return symbol;
}
void addTrade(Trade aTrade)
{
    trades.push_back(aTrade);
}
vector <Trade> getTrade()
{
    return trades;
}
void calculate_returns ()
{
    daily_returns.clear();
    for (vector<Trade>::iterator itr = trades.begin()+1; itr != trades.end(); itr++)
    {
        float temp = (*(itr)).Getprice()/(*(itr-1)).Getprice()-1;
        daily_returns[(*itr).Getdate()]=temp;
    }
}
map<string,float> get_returns()
{
    return daily_returns;
}
float calculateweekly(string startdate, string enddate)
{
    float temp1=0.0;
    float temp2=0.0;
    for (int j = 0;j<trades.size(); j++)
    {
        if (trades[j].Getdate()==startdate)
        {
            temp1=trades[j].Getprice();
        }
        if(trades[j].Getdate()==enddate)
        {
            temp2=trades[j].Getprice();
        }
    }
}

```

```

    }
}
return (temp2-temp1)/temp1;
}
friend ostream & operator << (ostream & out, const Stock & s)
{
    out << "Symbol: " << s.symbol << endl;
    out << "Fundamental data: " << endl << s.fund << endl;
    for (vector<Trade>::const_iterator itr = s.trades.begin(); itr != s.trades.end(); itr++)
        out << *itr;
    return out;
}
};

#endif /* Stock_hpp */

```

## A.4 Portfolio class

```

#ifndef Portfolio_hpp
#define Portfolio_hpp

#include <iostream>
#include <vector>
#include <map>
#include <math.h>
#include "Fundamental.hpp"
#include "Trade.hpp"
#include "Stock.hpp"
#include "Utilities.hpp"
using namespace std;

class Portfolio
{
public:
    vector <Stock> stocks;
    vector <float> weights;
    map<string,float> p_returns;
    float average_return;
    float p_sigma;
    float Sharpe;
    float p_pe;
    float p_beta;
    float p_div;
    float ma_50;
    float ma_200;

```



```

float fitness;
public:
Portfolio() {}
Portfolio(vector<Stock> stocks_): stocks(stocks_)
{}
~Portfolio() {}
vector<float> getweights()
{
    return weights;
}
vector<Stock> getstocks()
{
    return stocks;
}
void calculateweights(vector<Stock> stocks_)
{
    float sum=0.0;
    for (int i=0; i<stocks_.size(); i++)
    {
        sum=sum+stocks_[i].getprop();
    }
    weights.clear();
    for (int j=0; j<stocks_.size(); j++)
    {
        weights.push_back(stocks_[j].getprop()/sum);
    }
}
void Assignfitness()
{
    set_portfolio_returns();
    set_portfolio_sigma();
    set_P_E();
    set_sharpe();
    set_p_beta();
    set_p_div();
    set_ma();
    float pe_score=0.0;
    if (p_pe<30.0&& p_pe>10)
        pe_score=5;
    else if (p_pe>=30)
        pe_score=-5;
    //fitness=100.0*(0.6*Sharpe+0.2*p_beta+0.2*p_div);
    //fitness=60*Sharpe+40*p_beta+20*p_div;
    //better: fitness=60*Sharpe+20*p_beta+30*p_div+8*pe_score
    fitness=60*Sharpe+20*p_beta+30*p_div+8*pe_score+8*ma_50/ma_200;
    //fitness=60*Sharpe+20*p_beta+30*p_div+8*pe_score+8*ma_50/ma_200;
    //fitness=60*Sharpe+30*p_beta+30*p_div+8*pe_score+10*ma_50/ma_200;
    cout<<"fitness:"<<fitness<<endl;
}
void set_sharpe()
{

```

```

Map daily_rf_map;
daily_rf_map.clear();
for (Map::iterator itr=stocks[0].risk_free_rate.begin(); itr!=stocks[0].risk_free_rate.end(); itr++)
{
    daily_rf_map[itr->first]=pow(1+itr->second, 1/252.0)-1;
}
Map excess_return=(p_returns-daily_rf_map);
Map sharpe_ratio_map=excess_return/(p_sigma/sqrt(252));
Sharpe= calculate_average(sharpe_ratio_map);
cout<<"Sharpe:"<<Sharpe<<endl;
}
void set_P_E()
{
    p_pe=0.0;
    for (int i=0; i<stocks.size(); i++)
    {
        p_pe=p_pe+stocks[i].getfund().get_pe()*weights[i];
    }
    cout<<"P_E:"<<p_pe<<endl;
}
void set_p_beta()
{
    p_beta=0.0;
    for (int i=0; i<stocks.size(); i++)
    {
        p_beta=p_beta+stocks[i].getfund().get_beta()*weights[i];
    }
    cout<<"beta:"<<p_beta<<endl;
}
void set_p_div()
{
    p_div=0.0;
    for (int i=0; i<stocks.size(); i++)
    {
        p_div=p_div+stocks[i].getfund().get_div_yield()*weights[i];
    }
    cout<<"div:"<<p_div<<endl;
}
void set_ma()
{
    ma_50=0.0;
    ma_200=0.0;
    for (int i=0; i<stocks.size(); i++)
    {
        ma_50=ma_50+stocks[i].getfund().get_ma()*weights[i];
        ma_200=ma_200+stocks[i].getfund().get_ma200()*weights[i];
    }
    cout<<"MA:"<<ma_50<<endl;
    cout<<"MA200:"<<ma_200<<endl;
}
void set_portfolio_returns()

```

```

{
    p_returns.clear();
    for (int i=0; i<stocks.size(); i++)
    {
        map<string,float> individual_returns=stocks[i].get_returns();
        for (map<string,float>::iterator itr=individual_returns.begin(); itr!=individual_returns.end();itr++)
        {
            p_returns[itr->first]=p_returns[itr->first]+itr->second*weights[i];
        }
    }
    average_return=calculate_average(p_returns);
}
void set_portfolio_sigma()
{
    float average=calculate_average(p_returns);
    map<string,float> dif2;
    for (map<string,float>::iterator itr=p_returns.begin(); itr!=p_returns.end();itr++)
    {
        dif2[itr->first]=(itr->second-average)*(itr->second-average);
    }
    p_sigma=sqrt(calculate_average(dif2));
    cout<<"sigma:"<<p_sigma<<endl;
}
float calculateweekreturn(string startday, string endday)
{
    float weeklyreturn;
    float sum1=0.0;
    float sum2=0.0;
    float temp1=0.0;
    float temp2=0.0;
    for (int i=0; i<stocks.size(); i++)
    {
        for (int j = 0;j<stocks[i].getTrade().size(); j++)
        {
            if (stocks[i].getTrade()[j].Getdate()==startday)
            {
                temp1=stocks[i].getTrade()[j].Getprice();
                sum1=sum1+temp1*weights[i];
            }
            if(stocks[i].getTrade()[j].Getdate()==endday)
            {
                temp2=stocks[i].getTrade()[j].Getprice();
                sum2=sum2+temp2*weights[i];
            }
        }
    }
    weeklyreturn=(sum2-sum1)/sum1;
    return weeklyreturn;
}
friend ostream & operator << (ostream & out, Portfolio & p)
{

```

```

    for (vector<Stock>::iterator itr = p.stocks.begin(); itr != p.stocks.end(); itr++)
        out<< "Symbols: "<<itr->getsymbol()<<" ";
    for (vector<float>::iterator itr2 = p.weights.begin(); itr2!= p.weights.end(); itr2++)
        out<< "Weights: "<<*itr2<<" ";
    cout<<endl;
    return out;
}
};

#endif /* Portfolio_hpp */

```

## A.5 Utility functions

```

#ifndef Utilities_hpp
#define Utilities_hpp

#include <iostream>
#include <vector>
#include <algorithm> // std::sort
#include <map>
#include <math.h>
#include "Stock.hpp"
#define NUM_OF_STOCKS 500
#define SP500_NUM_OF_STOCKS 500
using namespace std;

typedef vector<float> Vector;
typedef vector<Vector> Matrix;
typedef map<string,float> Map;

struct less_than_symbol
{
    inline bool operator() (Stock &s1, Stock &s2)
    {
        return (s1.getsymbol() < s2.getsymbol());
    }
};

float calculate_expectation(vector<float> v);
Vector operator + (const Vector &a, const Vector &b);
Vector operator - (const Vector &a, const Vector &b);
Vector operator / (const Vector &a, const float &b);
Map operator - (Map &a, Map &b);

```

```

Map operator / (Map &a, const float &b);
float calculate_average(map<string,float> mymap);
vector <Stock> GetRandomStocks(int l, vector <Stock> total_stocks_, vector <int> & random_number);
void checkduplicate(vector <Stock> &s, vector <Stock> total_stocks_);
float calculatemothly(map<string, float> weekly);

```

```

#endif /* Utilities_hpp */

```

```

#include "Utilities.hpp"
#include <algorithm> // std::sort
#include <vector>

```

```

float calculate_expectation(vector <float> v)
{
    float sum=0;
    for (int i=0; i<v.size(); i++)
    {
        sum=sum+v[i];
    }
    return sum/v.size();
}

```

```

//vector calculation overloading
Vector operator + (const Vector &a, const Vector &b)
{
    unsigned long size_a=a.size();
    unsigned long size_b=b.size();
    unsigned long size;
    if (size_a<size_b)
        size=size_a;
    else
        size=size_b;
    Vector result(size);
    for (unsigned long i=size-1; i>=0; i--)
    {
        result[i]=a[i]+b[i];
    }
    return result;
}

```

```

Vector operator - (const Vector &a, const Vector &b)
{
    unsigned long size_a=a.size();
    unsigned long size_b=b.size();
    unsigned long size;
    if (size_a<size_b)
        size=size_a;
    else
        size=size_b;

```

```

Vector result(size);
for (unsigned long i=size-1; i>=0; i--)
{
    result[i]=a[i]-b[i];
}
return result;
}

```

```

Vector operator / (const Vector &a, const float &b)
{
    unsigned long size=a.size();
    Vector result(size);
    for (unsigned long i=0; i<size; i++)
    {
        result[i]=a[i]/b;
    }
    return result;
}

```

```

Map operator - (Map &a, Map &b)
{
    Map result;
    for (Map::iterator itr=a.begin(); itr!=a.end();itr++)
    {
        result[itr->first]=a[itr->first]-b[itr->first];
    }
    return result;
}

```

```

Map operator / (Map &a, const float &b)
{
    Map result;
    for (Map::iterator itr=a.begin(); itr!=a.end();itr++)
    {
        result[itr->first]=a[itr->first]/b;
    }
    return result;
}

```

```

float calculate_average(map<string,float> mymap)
{
    float temp=0;
    int count=0;
    for (map<string,float>::iterator itr=mymap.begin(); itr!=mymap.end();itr++)
    {
        temp=temp+itr->second;
        count++;
    }
    return temp/count;
}

```

```

}

vector <Stock> GetRandomStocks(int l, vector <Stock> total_stocks_, vector <int> & random_number)
{
    vector <Stock> temp;
    random_number.push_back(rand()%SP500_NUM_OF_STOCKS);
    for (int j=1;j<l;j++)
    {
        int a=rand()%SP500_NUM_OF_STOCKS;
        bool flag=true;
        while (flag)
        {
            if (std::find(random_number.begin(), random_number.end(), a)!=random_number.end())
            {
                a=rand()%SP500_NUM_OF_STOCKS;
            }
            else
                flag=false;
        }
        random_number.push_back(a);
    }
    for (int i=0; i<random_number.size(); i++)
    {
        temp.push_back(total_stocks_[random_number[i]]);
    }
    return temp;
}

void checkduplicate(vector <Stock> &s, vector <Stock> total_stocks_)
{
    bool flag=true;
    while(flag)
    {
        flag=false;
        std::sort(s.begin(), s.end(),less_than_symbol());
        for (int i=1; i<s.size(); i++)
        {
            if (s[i].getsymbol()==s[i-1].getsymbol())
            {
                vector <int> random_number_;
                vector <Stock> add_in = GetRandomStocks(1, total_stocks_, random_number_);
                s.reserve(s.size()+add_in.size()-1);
                s.erase(s.begin()+i-1);
                s.insert(s.end(), add_in.begin(), add_in.end());
                flag=true;
            }
        }
    }
}

```

```
float calculatemothly(map<string, float> weekly)
{
    float sum=0.0;
    for (map<string, float> ::iterator itr=weekly.begin(); itr!=weekly.end(); itr++)
    {
        sum=sum+itr->second;
    }
    return sum;
}
```

## A.6 GA functions

```
#ifndef GA_hpp
#define GA_hpp

#include <string>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <math.h>
#include <vector>
#include "Portfolio.hpp"

#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       1
#define MAX_ALLOWABLE_GENERATIONS  80
#define RANDOM_NUM        ((float)rand()/RAND_MAX)
using namespace std;

struct less_than_fitness
{
    inline bool operator() (const Portfolio & chromo1, const Portfolio & chromo2)
    {
        return (chromo1.fitness < chromo2.fitness);
    }
};

/////////////////////////////////prototypes/////////////////////////////////

Portfolio Roulette(int total_fitness, vector <Portfolio> & Population);
```



```

void Mutate(Portfolio &p_, vector <Stock> total_stocks_);
void Crossover(Portfolio &offspring1, Portfolio &offspring2);

#endif /* GA_hpp */

#include <string>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <math.h>
#include <vector>
#include "GA.hpp"

//-----Mutate-----
//
// Mutates a chromosome's bits dependent on the MUTATION_RATE
//-----

void Mutate(Portfolio &p_, vector <Stock> total_stocks_)
{
    if (RANDOM_NUM < MUTATION_RATE)
    {
        for (int j=0; j<3; j++)
        {
            int ran_ = rand()%CHROMO_LENGTH;
            int random_ = rand()%SP500_NUM_OF_STOCKS;
            p_.stocks[ran_]=total_stocks_[random_];
        }
        p_.calculateweights(p_.stocks);
        cout<<"the child:"<<p_<<endl;
        return;
    }
}

//----- Crossover -----
//
// Dependent on the CROSSOVER_RATE this function selects a random point along the
// length of the chromosomes and swaps all the bits after that point.
//-----

void Crossover(Portfolio &offspring1, Portfolio &offspring2)
{
    //dependent on the crossover rate
    // if (RANDOM_NUM < CROSSOVER_RATE)
    {
        //create a random crossover point
        //srand((int)time(NULL));
        int crossover = (int)(RANDOM_NUM * CHROMO_LENGTH);

        vector <Stock> stocks1=offspring1.stocks;
        vector <Stock> stocks2=offspring2.stocks;
    }
}

```

```

    vector <Stock> new_stock1;
    vector <Stock> new_stock2;
    for (int i=0; i<crossover; i++)
    {
        new_stock1.push_back(stocks1[i]);
        new_stock2.push_back(stocks2[i]);
    }
    for (int j=crossover; j<10; j++)
    {
        new_stock1.push_back(stocks2[j]);
        new_stock2.push_back(stocks1[j]);
    }
    Portfolio t1(new_stock1);
    Portfolio t2(new_stock2);
    offspring1 = t1; offspring2 = t2;
}
}

//select the parents use the best 70 populations, not the worse 70

//-----Roulette-----
//
// selects a chromosome from the population via roulette wheel selection
//-----
//string Roulette(int total_fitness, chromo_typ* Population)
Portfolio Roulette(int total_fitness, vector <Portfolio> & Population)
{
    float Slice = (float)(RANDOM_NUM * total_fitness);

    //go through the chromosomes adding up the fitness so far
    float FitnessSoFar = 0.0f;

    for (int i = 0; i<Population.size(); i++)
    {
        FitnessSoFar += Population[i].fitness;

        if (FitnessSoFar >= Slice)

            return Population[i];
    }
    return Population[0];
}

```

## A.7 Retrieve data

```

#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include <sqlite3.h>

#include "json/json.h"
#include "curl/curl.h"
#include "retrievedata.hpp"

using namespace std;

//writing call back function for storing fetched values in memory
static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    ((std::string*)userp)->append((char*)contents, size * nmemb);
    return size * nmemb;
}

int RetrieveMarketData(string url_request, Json::Value & root)
{
    std::string readBuffer;

    //global initiliation of curl before calling a function
    curl_global_init(CURL_GLOBAL_ALL);

    //creating session handle
    CURL * myHandle;

    // We'll store the result of CURL's webpage retrieval, for simple error checking.
    CURLcode result;

    // notice the lack of major error-checking, for brevity
    myHandle = curl_easy_init();

    curl_easy_setopt(myHandle, CURLOPT_URL, url_request.c_str());
    //curl_easy_setopt(myHandle, CURLOPT_URL,
    "https://eodhistoricaldata.com/api/eod/AAPL.US?from=2018-01-05&to=2019-02-10&api_token=5ba84ea974ab42.45160048&period=d&fmt=json");

    //adding a user agent
    curl_easy_setopt(myHandle, CURLOPT_USERAGENT, "Mozilla/5.0 (Windows; U; Windows NT
6.1; rv:2.2) Gecko/20110201");
    curl_easy_setopt(myHandle, CURLOPT_SSL_VERIFYPEER, 0);
    curl_easy_setopt(myHandle, CURLOPT_SSL_VERIFYHOST, 0);
    curl_easy_setopt(myHandle, CURLOPT_VERBOSE, 1);

```

```

// send all data to this function
curl_easy_setopt(myHandle, CURLOPT_WRITEFUNCTION, WriteCallback);

// we pass our 'chunk' struct to the callback function */
curl_easy_setopt(myHandle, CURLOPT_WRITEDATA, &readBuffer);

//perform a blocking file transfer
result = curl_easy_perform(myHandle);

// check for errors
if (result != CURLE_OK) {
    fprintf(stderr, "curl_easy_perform() failed: %s\n",
        curl_easy_strerror(result));
}
else {
    cout << readBuffer << endl;

    //json parsing
    Json::CharReaderBuilder builder;
    Json::CharReader * reader = builder.newCharReader();
    string errors;

    bool parsingSuccessful = reader->parse(readBuffer.c_str(), readBuffer.c_str() + readBuffer.size(),
    &root, &errors);
    if (not parsingSuccessful)
    {
        // Report failures and their locations
        // in the document.
        cout << "Failed to parse JSON" << std::endl
        << readBuffer
        << errors << endl;
        system("pause");
        return -1;
    }
    std::cout << "\nSuccess parsing json\n" << root << endl;
}

//End a libcurl easy handle. This function must be the last function to call for an easy session
curl_easy_cleanup(myHandle);
return 0;
}

```

## A.8 Database

```

#ifndef Database_hpp
#define Database_hpp

#include <iostream>
#include "Stock.hpp"
using namespace std;

int OpenDatabase(const char * name, sqlite3 * & db);
void CloseDatabase(sqlite3 *db);
int DropTable(const char * sql_drop_table, sqlite3 *db);
int CreateTable(const char *sql_create_table, sqlite3 *db);
int InsertTable(const char *sql_insert, sqlite3 *db);
int DisplayTable(const char *sql_select, sqlite3 *db);
int SelectValue(const char *sql_select, sqlite3 *db, vector<float> &fundinfo, int count);
int SelectTrade(const char *sql_select, sqlite3 *db, vector<vector<string>> &priceinfo);
int SelectRiskFree(const char *sql_select, sqlite3 *db, vector<vector<string>> &priceinfo);
int GetWeights(const char *sql_select, sqlite3 *db, vector<float> &weights);
int GetSymbols(const char *sql_select, sqlite3 *db, vector<string> &symbols);
int PopulateFundamentalTable(const Json::Value & root, string symbol, sqlite3 *db);
int PopulateRiskFreeTable(const Json::Value & root, sqlite3 *db);
int PopulateStockTable(const Json::Value & root, string symbol, sqlite3 *db);
int PopulateSP500Table(const Json::Value & root, sqlite3 *db, vector<string> &symbols);
int PopulateNewSP500Table(sqlite3 *db, vector<string> &symbols, vector<string> &weights);

#endif /* Database_hpp */

#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include <sqlite3.h>

#include "json/json.h"
#include "curl/curl.h"
#include "Database.hpp"

using namespace std;

int OpenDatabase(const char * name, sqlite3 * & db)
{
    int rc = 0;

```

```

//char *error = NULL;
// Open Database
cout << "Opening database: " << name << endl;
rc = sqlite3_open(name, &db);
if (rc)
{
    cerr << "Error opening SQLite3 database: " << sqlite3_errmsg(db) << endl;
    sqlite3_close(db);
    system("pause");
    return -1;
}
cout << "Opened database: " << name << endl;
return 0;
}

void CloseDatabase(sqlite3 *db)
{
    cout << "Closing a database ..." << endl;
    sqlite3_close(db);
    cout << "Closed a database" << endl << endl;
}

int DropTable(const char * sql_drop_table, sqlite3 *db)
{
    // Drop the table if exists
    if (sqlite3_exec(db, sql_drop_table, 0, 0, 0) != SQLITE_OK) { // or == -- same effect
        std::cout << "SQLite can't drop sessions table" << std::endl;
        sqlite3_close(db);
        system("pause");
        return -1;
    }
    return 0;
}

int CreateTable(const char *sql_create_table, sqlite3 *db)
{
    int rc = 0;
    char *error = NULL;
    // Create the table
    cout << "Creating a table..." << endl;
    rc = sqlite3_exec(db, sql_create_table, NULL, NULL, &error);
    if (rc)
    {
        cerr << "Error executing SQLite3 statement: " << sqlite3_errmsg(db) << endl << endl;
        sqlite3_free(error);
        system("pause");
        return -1;
    }
    cout << "Created a table." << endl;
}

```

```

    return 0;
}

int InsertTable(const char *sql_insert, sqlite3 *db)
{
    int rc = 0;
    char *error = NULL;
    // Execute SQL
    cout << "Inserting a value into a table ..." << endl;
    rc = sqlite3_exec(db, sql_insert, NULL, NULL, &error);
    if (rc)
    {
        cerr << "Error executing SQLite3 statement: " << sqlite3_errmsg(db) << endl << endl;
        sqlite3_free(error);
        system("pause");
        return -1;
    }
    cout << "Inserted a value into the table." << endl;
    return 0;
}

int DisplayTable(const char *sql_select, sqlite3 *db)
{
    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;

            // Display Cell Value

```

```

        cout.width(12);
        cout.setf(ios::left);
        cout << results[cellPosition] << " ";
    }

    // End Line
    cout << endl;

    // Display Separator For Header
    if (0 == rowCtr)
    {
        for (int colCtr = 0; colCtr < columns; ++colCtr)
        {
            cout.width(12);
            cout.setf(ios::left);
            cout << "~~~~~";
        }
        cout << endl;
    }
}

// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}

int SelectValue(const char *sql_select, sqlite3 *db, vector<float> &fundinfo, int count)
{
    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 = count; rowCtr < count+1; ++rowCtr)
    {
        for (int colCtr = 1; colCtr < columns; ++colCtr)
        {

```



```

    // Determine Cell Position
    int cellPosition = (rowCtr * columns) + colCtr;

    // Display Cell Value
    cout.width(12);
    cout.setf(ios::left);
    cout << results[cellPosition] << " ";
    float output = atof(results[cellPosition]);
    fundinfo.push_back(output);
}

// End Line
cout << endl;

// Display Separator For Header
if (0 == rowCtr)
{
    for (int colCtr = 0; colCtr < columns; ++colCtr)
    {
        cout.width(12);
        cout.setf(ios::left);
        cout << "~~~~~";
    }
    cout << endl;
}
}

// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}

int SelectTrade(const char *sql_select, sqlite3 *db, vector <vector <string> > &priceinfo)
{
    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)
{
    vector<string> temp_price;
    for (int colCtr = 2; colCtr < columns; ++colCtr)
    {
        // Determine Cell Position
        int cellPosition = (rowCtr * columns) + colCtr;
        string keai = string(results[cellPosition]);
        temp_price.push_back(keai);
    }

    priceinfo.push_back(temp_price);
}

// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}

int SelectRiskFree(const char *sql_select, sqlite3 *db, vector <vector <string> > &priceinfo)
{
    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)
    {
        vector<string> temp_price;
        for (int colCtr = 1; colCtr < columns; ++colCtr)
        {
            // Determine Cell Position

```

```

        int cellPosition = (rowCtr * columns) + colCtr;
        string keai = string(results[cellPosition]);
        temp_price.push_back(keai);
    }

    priceinfo.push_back(temp_price);

}
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}

int GetWeights(const char *sql_select, sqlite3 *db, vector<float> &weights)
{
    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)
    {
        for (int colCtr = 4; colCtr < 5; ++colCtr)
        {
            // Determine Cell Position
            int cellPosition = (rowCtr * columns) + colCtr;

            // Display Cell Value
            cout.width(12);
            cout.setf(ios::left);
            cout << results[cellPosition] << " ";
            string output = string(results[cellPosition]);
            float output2 = (float)stof(output);
            weights.push_back(output2);
        }
    }
}

```

```

// End Line
cout << endl;

// Display Separator For Header
if (0 == rowCtr)
{
    for (int colCtr = 0; colCtr < columns; ++colCtr)
    {
        cout.width(12);
        cout.setf(ios::left);
        cout << "~~~~~";
    }
    cout << endl;
}
}
// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}

int GetSymbols(const char *sql_select, sqlite3 *db, vector<string> &symbols)
{
    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)
    {
        for (int colCtr = 1; colCtr <= columns; ++colCtr)
        {
            // Determine Cell Position
            int cellPosition = (rowCtr * columns) + colCtr;

```

```

    // Display Cell Value
    cout.width(12);
    cout.setf(ios::left);
    cout << results[cellPosition] << " ";
    string output = string(results[cellPosition]);
    symbols.push_back(output);
}

// End Line
cout << endl;

// Display Separator For Header
if (0 == rowCtr)
{
    for (int colCtr = 0; colCtr < columns; ++colCtr)
    {
        cout.width(12);
        cout.setf(ios::left);
        cout << "~~~~~";
    }
    cout << endl;
}
}

// This function properly releases the value array returned by sqlite3_get_table()
sqlite3_free_table(results);
return 0;
}

int PopulateFundamentalTable(const Json::Value & root, string symbol, sqlite3 *db)
{
    float P_E_ratio = 0, Div_yield = 0, beta = 0, high_52 = 0, low_52 = 0, MA_50 = 0, MA_200 = 0;
    Stock myStock(symbol);
    for (Json::Value::const_iterator itr = root.begin(); itr != root.end(); itr++)
    {
        cout << *itr << endl;
        if (itr.key().asString() == "Highlights")
            for (Json::Value::const_iterator inner = (*itr).begin(); inner != (*itr).end(); inner++)
            {
                if (inner.key().asString() == "PERatio")
                {
                    if (*inner)
                        P_E_ratio = (float)atof(inner->asCString());
                }
                else if (inner.key().asString() == "DividendYield")
                {
                    if (*inner)
                        Div_yield = (float)atof(inner->asCString());
                }
            }
        else if (itr.key().asString() == "Technicals")

```

```

for (Json::Value::const_iterator inner2 = (*itr).begin(); inner2 != (*itr).end(); inner2++)
{
    if (inner2.key().asString() == "Beta")
    {
        if (*inner2)
            beta = (float)atof(inner2->asCString());
    }
    else if (inner2.key().asString() == "52WeekHigh")
    {
        if (*inner2)
            high_52 = (float)atof(inner2->asCString());
    }
    else if (inner2.key().asString() == "52WeekLow")
    {
        if (*inner2)
            low_52 = (float)atof(inner2->asCString());
    }
    else if (inner2.key().asString() == "50DayMA")
    {
        if (*inner2)
            MA_50 = (float)atof(inner2->asCString());
    }
    else if (inner2.key().asString() == "200DayMA")
    {
        if (*inner2)
            MA_200 = (float)atof(inner2->asCString());
    }
}
}
Fundamental aFund(P_E_ratio, Div_yield, beta, high_52, low_52, MA_50, MA_200);
myStock.setfund(aFund);

// Execute SQL
//P_E_ratio = 0, Div_yield = 0, beta = 0, high_52 = 0, low_52 = 0, MA_50 = 0, MA_200 = 0;
char fundamentals_insert_table[512];
sprintf(fundamentals_insert_table, "INSERT INTO FUNDAMENTALS (symbol, P_E_ratio,
Div_yield, beta, high_52, low_52, MA_50, MA_200) VALUES(\"%s\", %f, %f, %f, %f, %f, %f, %f)",
symbol.c_str(), P_E_ratio, Div_yield, beta, high_52, low_52, MA_50, MA_200);
if (InsertTable(fundamentals_insert_table, db) == -1)
    return -1;

cout << myStock << endl;
return 0;
}

int PopulateRiskFreeTable(const Json::Value & root, sqlite3 *db)
{
    string date;
    float open = 0, high = 0, low = 0, close = 0, adjusted_close = 0;
    int volume = 0;

```

```

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 = inner->asFloat();
        else if (inner.key().asString() == "close")
            close = inner->asFloat();
        else if (inner.key() == "date")
            date = inner->asString();
        else if (inner.key().asString() == "high")
            high = inner->asFloat();
        else if (inner.key().asString() == "low")
            low = inner->asFloat();
        else if (inner.key() == "open")
            open = inner->asFloat();
        else if (inner.key().asString() == "volume")
            volume = inner->asInt();
        else
        {
            cout << "Invalid json field" << endl;
            return -1;
        }
    }
    count++;

    // Execute SQL
    char stockDB_insert_table[512];
    sprintf(stockDB_insert_table, "INSERT INTO RISKFREE (id, date, open, high, low, close,
adjusted_close, volume) VALUES(%d, \"%s\", %f, %f, %f, %f, %f, %d)", count, date.c_str(), open, high,
low, close, adjusted_close, volume);
    if (InsertTable(stockDB_insert_table, db) == -1)
        return -1;
}
return 0;
}

int PopulateStockTable(const Json::Value & root, string symbol, sqlite3 *db)
{
    string date;
    float open = 0, high = 0, low = 0, close = 0, adjusted_close = 0;
    long long volume = 0;
    Stock myStock(symbol);
    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 = inner->asFloat();
        else if (inner.key().asString() == "close")
            close = inner->asFloat();
        else if (inner.key() == "date")
            date = inner->asString();
        else if (inner.key().asString() == "high")
            high = inner->asFloat();
        else if (inner.key().asString() == "low")
            low = inner->asFloat();
        else if (inner.key() == "open")
            open = inner->asFloat();
        else if (inner.key().asString() == "volume")
            volume = inner->asUInt();
        else
        {
            cout << "Invalid json field" << endl;
            return -1;
        }
    }
    Trade aTrade(date, open, high, low, close, adjusted_close, volume);
    myStock.addTrade(aTrade);
    count++;

    // Execute SQL
    char stockDB_insert_table[512];
    sprintf(stockDB_insert_table, "INSERT INTO STOCK_%s (id, symbol, date, open, high, low, close,
adjusted_close, volume) VALUES(%d, \"%s\", \"%s\", %f, %f, %f, %f, %f, %lld)", +symbol.c_str(),
count, symbol.c_str(), date.c_str(), open, high, low, close, adjusted_close, volume);
    if (InsertTable(stockDB_insert_table, db) == -1)
        return -1;
}
cout << myStock;
return 0;
}

int PopulateSP500Table(const Json::Value & root, sqlite3 *db, vector <string> &symbols)
{
    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();
    else
    {
        cout << "Invalid json field" << endl;
        system("pause");
        return -1;
    }
}
symbols.push_back(symbol);
count++;

// Execute SQL
char sp500_insert_table[512];
sprintf(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 PopulateNewSP500Table(sqlite3 *db, vector <string> &symbols, vector<string> &weights)
{
    fstream fin;
    fin.open("new_SP500.csv", ios::in);
    string title, name, symbol, weight, sector, shares;
    getline(fin, title);
    cout<<title<<endl;
    int count=0;
    while (fin.good())
    {
        getline(fin, name, ',');
        cout<<name<<endl;
        getline(fin, symbol, ',');
        cout<<symbol<<endl;
        getline(fin, weight, ',');
        cout<<weight<<endl;
        getline(fin, sector, ',');
        cout<<sector<<endl;
        getline(fin, shares);
    }
}

```

```

    cout<<shares<<endl;
    count++;
    symbols.push_back(symbol);
    weights.push_back(weight);

    // Execute SQL
    char new_sp500_insert_table[512];
    sprintf(new_sp500_insert_table, "INSERT INTO NEWSP500 (id, symbol, name, sector, weight,
shares) VALUES(%d, \"%s\", \"%s\", \"%s\", \"%s\", \"%s\")", count, symbol.c_str(), name.c_str(),
sector.c_str(), weight.c_str(), shares.c_str());
    if (InsertTable(new_sp500_insert_table, db) == -1)
        return -1;
    }
    return 0;
}

```

## A.9 Main function

```

#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <stdio.h>
#include <sqlite3.h>
#include <math.h>

#include "json/json.h"
#include "curl/curl.h"
#include "Database.hpp"
#include "retrievedata.hpp"
#include "Fundamental.hpp"
#include "Trade.hpp"
#include "Stock.hpp"
#include "Utilities.hpp"
#include "Portfolio.hpp"
#include "GA.hpp"

using namespace std;

int risk_frees(sqlite3 *db, string stock_select_, Map &risk_free_rate_)
{
    vector <vector <string>> temp;
    if (SelectRiskFree(stock_select_.c_str(), db, temp) == -1)

```

```

        return -1;
    for (int i=0; i<temp.size(); i++)
    {
        string date_=temp[i][0];
        float adjusted_close_=(float)stof(temp[i][5]);
        risk_free_rate_[date_]=adjusted_close_;
    }
    return 0;
}

int stock_prices(sqlite3 *db, string stock_select_, Stock & aStock_)
{
    vector<vector<string>> temp;
    if (SelectTrade(stock_select_.c_str(), db, temp) == -1)
        return -1;
    for (int i=0; i<temp.size(); i++)
    {
        string date_=temp[i][0];
        float open_=(float)stof(temp[i][1]);
        float high_=(float)stof(temp[i][2]);
        float low_=(float)stof(temp[i][3]);
        float close_=(float)stof(temp[i][4]);
        float adjusted_close_=(float)stof(temp[i][5]);
        int volume_=stoi(temp[i][6]);

        Trade aTrade(date_,open_,high_,low_,close_,adjusted_close_,volume_);
        aStock_.addTrade(aTrade);
    }
    return 0;
}

int stock_fundamentals(sqlite3 *db, string fundamental_select_, Stock & aStock_, int count)
{
    vector<float> temp;
    if (SelectValue(fundamental_select_.c_str(), db, temp, count) == -1)
        return -1;
    Fundamental fund(temp[0],temp[1],temp[2],temp[3],temp[4],temp[5],temp[6]);
    aStock_.setfund(fund);
    return 0;
}

//float calculateSPYnum(string startdate, Stock SPYstock)
//{
//    float num=0.1;
//    for (vector<Trade>::iterator itr = SPYstock.getTrade().begin(); itr != SPYstock.getTrade().end();
//itr++)
//    {
//        if ((*itr).Getdate()==startdate)
//        {
//            num=1000000/(*itr).Getprice();

```

```

//    }
// }
// return num;
//}

int main(void)

{
    char answer;
    bool continue_menu = true;
    srand((int)time(NULL));

    const char * stockDB_name = "Stocks.db";
    sqlite3 * stockDB = NULL;
    if (OpenDatabase(stockDB_name, stockDB) == -1)
        return -1;
    cout<<"*****\n";
    cout<<"1 - Store SP500 and stocks price data into database.\n";
    cout<<"2 - Store fundamental data into database.\n";
    cout<<"3 - Store risk free rate into database.\n";
    cout<<"4 - GA selection.\n";
    cout<<"5 - get the weights from csv.\n";
    cout<<"6 - SPY returns for 2019. \n";
    cout<<"7 - backtesting for best portfolio 2019. \n";
    cout<<"8 - prob testing. \n";
    cout<<"0 - Exit the program.\n";
    cout<<"Enter your choice and press return: ";
    Portfolio *pBest = NULL;
    ofstream myfile ("result.csv", ios_base::app);
    //myfile.open ("result.csv");

    while(continue_menu != false)
    {
        cin >> answer;
        switch (answer)
        {
            case '1':
            {
                cout<<"1 - Store SP500 and stocks price data into database.\n";

                std::string new_sp500_drop_table = "DROP TABLE IF EXISTS NEWSP500;";
                if (DropTable(new_sp500_drop_table.c_str(), stockDB) == -1)
                    return -1;

                string new_sp500_create_table = "CREATE TABLE NEWSP500 (id INT PRIMARY KEY
                NOT NULL, symbol CHAR(20) NOT NULL, name CHAR(20) NOT NULL, sector CHAR(20) NOT
                NULL, weight CHAR(20) NOT NULL, shares CHAR(20) NOT NULL);";

                if (CreateTable(new_sp500_create_table.c_str(), stockDB) == -1)
                    return -1;
            }
        }
    }
}

```

```

vector <string> new_symbols;
vector <string> weights;
if (PopulateNewSP500Table(stockDB, new_symbols, weights) == -1)
    return -1;

string new_sp500_select_table = "SELECT * FROM NEWSP500;";
if (DisplayTable(new_sp500_select_table.c_str(), stockDB) == -1)
    return -1;

new_symbols.push_back("SPY");

for (int i=0; i<new_symbols.size(); i++)
{
    string stockDB_symbol = new_symbols[i];
    if (stockDB_symbol=="BRK.B")
        stockDB_symbol="BRK_B";
    if (stockDB_symbol=="BF.B")
        stockDB_symbol="BF_B";
    if (stockDB_symbol=="LUK"||stockDB_symbol=="MON"||stockDB_symbol=="MSI")
        continue;
    if (stockDB_symbol=="CASH_USD")
        stockDB_symbol="TMUS";

    std::string stockDB_drop_table = "DROP TABLE IF EXISTS STOCK_" + stockDB_symbol
+ ";";

    if (DropTable(stockDB_drop_table.c_str(), stockDB) == -1)
        return -1;

    string stockDB_create_table = "CREATE TABLE STOCK_" + 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 INT NOT NULL);";

    if (CreateTable(stockDB_create_table.c_str(), stockDB) == -1)
        return -1;

    if (stockDB_symbol=="BRK_B")
        stockDB_symbol="BRK-B";
    if (stockDB_symbol=="BF_B")
        stockDB_symbol="BF-B";

    //https://eodhistoricaldata.com/api/eod/AGN.US?from=2019-07-01&to=2019-07-
31&api_token=5ba84ea974ab42.45160048&period=d&fmt=json
    string stock_url_common = "https://eodhistoricaldata.com/api/eod/";

```

```

string stock_start_date = "2008-01-01";
string stock_end_date = "2019-07-31";
string api_token = "5ba84ea974ab42.45160048";
string stockDB_data_request = stock_url_common + stockDB_symbol + ".US?" +
    "from=" + stock_start_date + "&to=" + stock_end_date + "&api_token=" + api_token +
    "&period=d&fmt=json";

Json::Value stockDB_root; // will contains the root value after parsing.
if (RetrieveMarketData(stockDB_data_request, stockDB_root) == -1)
    return -1;
if (stockDB_symbol=="BRK-B")
    stockDB_symbol="BRK_B";
if (stockDB_symbol=="BF-B")
    stockDB_symbol="BF_B";
if (PopulateStockTable(stockDB_root, stockDB_symbol, stockDB) == -1)
    return -1;

string stockDB_select_table = "SELECT * FROM STOCK_" + stockDB_symbol + ";";
if (DisplayTable(stockDB_select_table.c_str(), stockDB) == -1)
    return -1;
}
break;
}
case '2':
{
    cout<<"2 - Store fundamental data into database.\n";
    // const char * stockDB_name = "Stocks.db";
    // sqlite3 * stockDB = NULL;
    // if (OpenDatabase(stockDB_name, stockDB) == -1)
    //     return -1;

    std::string new_sp500_drop_table = "DROP TABLE IF EXISTS NEWSP500;";
    if (DropTable(new_sp500_drop_table.c_str(), stockDB) == -1)
        return -1;

    string new_sp500_create_table = "CREATE TABLE NEWSP500 (id INT PRIMARY KEY
    NOT NULL, symbol CHAR(20) NOT NULL, name CHAR(20) NOT NULL, sector CHAR(20) NOT
    NULL, weight CHAR(20) NOT NULL, shares CHAR(20) NOT NULL);";

    if (CreateTable(new_sp500_create_table.c_str(), stockDB) == -1)
        return -1;

    vector <string> new_symbols;
    vector <string> weights;
    if (PopulateNewSP500Table(stockDB, new_symbols, weights) == -1)
        return -1;

    string new_sp500_select_table = "SELECT * FROM NEWSP500;";
    if (DisplayTable(new_sp500_select_table.c_str(), stockDB) == -1)
        return -1;
}

```

```

new_symbols.push_back("SPY");

std::string fundamentals_drop_table = "DROP TABLE IF EXISTS FUNDAMENTALS;";
if (DropTable(fundamentals_drop_table.c_str(), stockDB) == -1)
    return -1;

string fundamentals_create_table = "CREATE TABLE FUNDAMENTALS (symbol
CHAR(20) NOT NULL, P_E_ratio REAL NOT NULL, Div_yield REAL NOT NULL, beta REAL NOT
NULL, high_52 REAL NOT NULL, low_52 REAL NOT NULL, MA_50 REAL NOT NULL, MA_200
REAL NOT NULL);";

if (CreateTable(fundamentals_create_table.c_str(), stockDB) == -1)
    return -1;

for (int i=0; i<new_symbols.size(); i++)
{
    string fundamentals_symbol = new_symbols[i];
    if (fundamentals_symbol=="BRK.B")
        fundamentals_symbol="BRK_B";
    if (fundamentals_symbol=="BF.B")
        fundamentals_symbol="BF_B";

    std::string fundamentals_drop_table = "DROP TABLE IF EXISTS FUNDAMENTALS_" +
fundamentals_symbol + ";";
    if (DropTable(fundamentals_drop_table.c_str(), stockDB) == -1)
        return -1;

    if (fundamentals_symbol=="BRK_B")
        fundamentals_symbol="BRK-B";
    if (fundamentals_symbol=="BF_B")
        fundamentals_symbol="BF-B";

//https://eodhistoricaldata.com/api/fundamentals/AAPL.US?api_token=5ba84ea974ab42.45160048
    string fundamentals_url_common = "https://eodhistoricaldata.com/api/fundamentals/";
    string api_token = "5ba84ea974ab42.45160048";
    string fundamentals_data_request = fundamentals_url_common + fundamentals_symbol +
".US?" +
    "api_token=" + api_token;

    Json::Value stock_fund_root; // will contains the root value after parsing.
    if (RetrieveMarketData(fundamentals_data_request, stock_fund_root) == -1)
        return -1;

    if (fundamentals_symbol=="BRK-B")
        fundamentals_symbol="BRK_B";
    if (fundamentals_symbol=="BF-B")
        fundamentals_symbol="BF_B";

```

```

        //int count = 0;
        if (PopulateFundamentalTable(stock_fund_root, fundamentals_symbol, stockDB) == -1)
            return -1;

    }
    string fundamentals_select_table = "SELECT * FROM FUNDAMENTALS;";
    if (DisplayTable(fundamentals_select_table.c_str(), stockDB) == -1)
        return -1;

//      // Close Database
//      CloseDatabase(stockDB);

    break;
}
case '3':
{
    cout<<"3 - Store risk free rate into database.\n";
//      const char * stockDB_name = "Stocks.db";
//      sqlite3 * stockDB = NULL;
//      if (OpenDatabase(stockDB_name, stockDB) == -1)
//          return -1;

    std::string RiskFree_drop_table = "DROP TABLE IF EXISTS RISKFREE;";
    if (DropTable(RiskFree_drop_table.c_str(), stockDB) == -1)
        return -1;

    string RiskFree_create_table = "CREATE TABLE RISKFREE (id INT PRIMARY KEY 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 INT NOT NULL);";

    if (CreateTable(RiskFree_create_table.c_str(), stockDB) == -1)
        return -1;

    string RiskFree_data_request = "https://eodhistoricaldata.com/api/eod/TNX.INDX?from=2008-
    01-01&to=2019-07-10&api_token=5ba84ea974ab42.45160048&period=d&fmt=json";

    Json::Value sp500_root; // will contains the root value after parsing.
    if (RetrieveMarketData(RiskFree_data_request, sp500_root) == -1)
        return -1;
    if (PopulateRiskFreeTable(sp500_root, stockDB) == -1)
        return -1;

    string RiskFree_select_table = "SELECT * FROM RISKFREE;";
    if (DisplayTable(RiskFree_select_table.c_str(), stockDB) == -1)
        return -1;

    if (risk_frees(stockDB, RiskFree_select_table, Stock::risk_free_rate) == -1)
        return -1;

    Stock astock("AAPL");

```



```

for (Map::iterator itr=astock.risk_free_rate.begin();itr!=astock.risk_free_rate.end();itr++)
{
    cout<<itr->second<<endl;
}
break;
}
case '4':
{
    cout<<"4 - GA selection.\n";
    myfile<<"csv\n";
    vector <Stock> total_stocks;
    //    const char * stockDB_name = "Stocks.db";
    //    sqlite3 * stockDB = NULL;
    //    if (OpenDatabase(stockDB_name, stockDB) == -1)
    //        return -1;
    vector <string> symbols;
    string a = "SELECT * FROM NEWSP500;";
    if (GetSymbols(a.c_str(), stockDB, symbols)==-1)
        return -1;

    int count=1;
    cout<<symbols.size()<<endl;
    for (vector<string>::iterator itr=symbols.begin(); itr!=symbols.end(); itr++)
    {
        if (*itr=="BRK.B")
            *itr="BRK_B";
        if (*itr=="BF.B")
            *itr="BF_B";
        if (*itr=="CASH_USD")
            *itr="TMUS";
        if (*itr=="DOW"||*itr=="CTVA"||*itr=="AMCR"||*itr=="FOXA"||*itr=="CPRI")
        {
            count++;
            continue;
        }
        Stock aStock(*itr);
        string c = "SELECT * FROM FUNDAMENTALS;";
        if (stock_fundamentals(stockDB, c, aStock, count)==-1)
            return -1;
        string stock_select = "SELECT * FROM STOCK_" + *itr + " WHERE
strftime('%Y-%m-%d', date) between \"2008-01-01\" and \"2018-12-31\" and open > 0 and
adjusted_close > 0;";
        if (stock_prices(stockDB, stock_select, aStock)==-1)
            return -1;
        aStock.calculate_returns();
        cout<<*itr<<"!!!!!!!!!!!!!!"<<endl;
        total_stocks.push_back(aStock);
        count++;
    }

    vector<float> ori_weights;

```

```

string b = "SELECT * FROM NEWSP500;";
if (GetWeights(b.c_str(), stockDB, ori_weights)==-1)
    return -1;

int te=0;
for (int i=0; i<ori_weights.size(); i++)
{
    if
(symbols[i]=="DOW"||symbols[i]=="CTVA"||symbols[i]=="AMCR"||symbols[i]=="FOXA"||symbols[i]=
="CPRI")
    {
        te=te+1;
        i=i+1;
    }
    else
        total_stocks[i-te].setprop(ori_weights[i]);
}

//CloseDatabase(stockDB);

vector <Portfolio> Population(POP_SIZE);
for (int i = 0; i<POP_SIZE; i++)
{
    //first generation
    vector <int> random_number_;
    Population[i].stocks = GetRandomStocks(10, total_stocks, random_number_);
    Population[i].calculateweights(Population[i].stocks);
    Population[i].fitness = 0.0f;
    cout<<Population[i];
}

int GenerationsRequiredToFindASolution = 0;
int countchange = 0;
bool bFound = false;
bool bChange;
float MaxFitness = 0.0f;
Portfolio BestP;

//enter the main GA loop
while (!bFound) //find the fitness for the population by adding up
{
    //this is used during roulette wheel sampling
    float TotalFitness = 0.0f;
    bChange = false;

    // test and update the fitness of every chromosome in the population
    for (int i = 0; i<POP_SIZE; i++)
    {
        Population[i].Assignfitness();
        if (Population[i].fitness > MaxFitness)

```

```

    {
        MaxFitness = Population[i].fitness;
        BestP = Population[i];
        bChange = true;
        countchange = 0;
    }
}

if (bChange == false)
    countchange++;

if (countchange == 10)
{
    cout << "\nSolution found in " << GenerationsRequiredToFindASolution << "
generations!" << endl;
    cout << "Result: ";
    cout << BestP;
    pBest = new Portfolio (BestP);
    cout << *pBest;
    cout << "Fitness: " << MaxFitness << endl;
    bFound = true;
    break;
}

//Sort all the chromosomes in a population in ascending order
std::sort(Population.begin(),Population.end(),less_than_fitness());
cout<<"sort completed!"<<"^^^^^^^^^^"<<endl;

myfile<<GenerationsRequiredToFindASolution<<","<<Population[99].fitness<<","<<endl;

//define some temporary storage for the new population we are about to create
vector<Portfolio> temp(POP_SIZE*CROSSOVER_RATE);

int cPop = 0;

vector<Portfolio> goodPopulation;
for (int i=0; i<POP_SIZE*CROSSOVER_RATE; i++)
{
    goodPopulation.push_back(Population[POP_SIZE-i-1]);
}
for (int i=0; i<goodPopulation.size(); i++)
{
    TotalFitness += goodPopulation[i].fitness;
}
cout<<"Prepared!"<<endl;

//loop until we have created POP_SIZE new chromosomes
while (cPop < POP_SIZE*CROSSOVER_RATE)
{
    // we are going to create the new population by grabbing members of the old population

```

```

// two at a time via roulette wheel selection.
Portfolio offspring1 = Roulette(TotalFitness, goodPopulation);
Portfolio offspring2 = Roulette(TotalFitness, goodPopulation);

while (offspring1.average_return==offspring2.average_return)
{
    offspring1 = Roulette(TotalFitness, goodPopulation);
    offspring2 = Roulette(TotalFitness, goodPopulation);
}

//make sure not duplicate
cout<<"will crossover!"<<endl;

//add crossover dependent on the crossover rate
Crossover(offspring1, offspring2);
cout<<"Crossover!"<<endl;

//now mutate dependent on the mutation rate
Mutate(offspring1, total_stocks);
Mutate(offspring2, total_stocks);
cout<<"Mutate!"<<endl;

//add these offspring to the new population. (assigning zero as their
//fitness scores)
offspring1.fitness=0.0f;
offspring2.fitness=0.0f;
vector <Stock> o1(offspring1.getstocks());
vector <Stock> o2(offspring2.getstocks());
checkduplicate(o1, total_stocks);
checkduplicate(o2, total_stocks);
cout<<"change"<<endl;
offspring1=Portfolio(o1);
offspring2=Portfolio(o2);
offspring1.calculateweights(o1);
offspring2.calculateweights(o2);
temp[cPop++] = offspring1;
temp[cPop++] = offspring2;

} //end loop

//copy temp population into main population array
for (int i = 0; i<POP_SIZE*CROSSOVER_RATE; i++)
{
    Population[i] = temp[i]; //use children to replace the old chromosom
}

++GenerationsRequiredToFindASolution;
cout<<"generation: "<<GenerationsRequiredToFindASolution<<endl;

// exit app if no solution found within the maximum allowable number

```

```

// of generations
if (GenerationsRequiredToFindASolution > MAX_ALLOWABLE_GENERATIONS) // if
more than 400, quit
{
    cout << "The best portfolio is:";
    cout<<Population[99]<<endl;
    pBest = new Portfolio (Population[99]);
    cout<<*pBest;
    cout<<"Max fitness:"<<MaxFitness<<endl;

    const char * stockDB_name = "Stocks.db";
    sqlite3 * stockDB = NULL;
    if (OpenDatabase(stockDB_name, stockDB) == -1)
        return -1;
    std::string portfolio_drop_table = "DROP TABLE IF EXISTS BestPortfolio;";
    if (DropTable(portfolio_drop_table.c_str(), stockDB) == -1)
        return -1;

    string portfolio_create_table = "CREATE TABLE BestPortfolio (id INT PRIMARY KEY
NOT NULL, stockname CHAR(20) NOT NULL, weight REAL NOT NULL);";

    if (CreateTable(portfolio_create_table.c_str(), stockDB) == -1)
        return -1;
    int countstock=0;
    for (int p=0; p<Population[99].getstocks().size(); p++)
    {
        countstock++;
        char portfolio_insert_table[512];
        sprintf(portfolio_insert_table, "INSERT INTO BestPortfolio (id, stockname, weight)
VALUES(%d, \"%s\", %f)", countstock, Population[99].getstocks()[p].getsymbol().c_str(),
Population[99].weights[p]);
        if (InsertTable(portfolio_insert_table, stockDB) == -1)
            return -1;
    }

    bFound = true;
}
break;
}
case '5':
{
    cout<<"5 - get the weights from csv.\n";
    // const char * stockDB_name = "Stocks.db";
    // sqlite3 * stockDB = NULL;
    // if (OpenDatabase(stockDB_name, stockDB) == -1)
    //     return -1;

    std::string new_sp500_drop_table = "DROP TABLE IF EXISTS NEWSP500;";
    if (DropTable(new_sp500_drop_table.c_str(), stockDB) == -1)

```

```

        return -1;

        string new_sp500_create_table = "CREATE TABLE NEWSP500 (id INT PRIMARY KEY
NOT NULL, symbol CHAR(20) NOT NULL, name CHAR(20) NOT NULL, sector CHAR(20) NOT
NULL, weight CHAR(20) NOT NULL, shares CHAR(20) NOT NULL);";

        if (CreateTable(new_sp500_create_table.c_str(), stockDB) == -1)
            return -1;

        vector<string> new_symbols;
        vector<string> weights;
        if (PopulateNewSP500Table(stockDB, new_symbols, weights) == -1)
            return -1;

        string new_sp500_select_table = "SELECT * FROM NEWSP500;";
        if (DisplayTable(new_sp500_select_table.c_str(), stockDB) == -1)
            return -1;
        break;
    }
    case '6':
    {
        cout<<"6 - SPY returns for 2019. \n";
        // const char * stockDB_name = "Stocks.db";
        // sqlite3 * stockDB = NULL;
        // if (OpenDatabase(stockDB_name, stockDB) == -1)
        //     return -1;
        Stock SPY("SPY");
        string c = "SELECT * FROM FUNDAMENTALS;";
        if (stock_fundamentals(stockDB, c, SPY, 506)==-1)
            return -1;
        string stock_select = "SELECT * FROM STOCK_SPY WHERE strftime('%Y-%m-%d', date)
between \"2019-01-01\" and \"2019-07-31\" and open > 0 and adjusted_close > 0;";
        if (stock_prices(stockDB, stock_select, SPY)==-1)
            return -1;
        SPY.calculate_returns();
        vector<string> starts={"2019-01-07", "2019-01-14", "2019-01-22", "2019-01-28"};
        vector<string> ends={"2019-01-11", "2019-01-18", "2019-01-25", "2019-02-01"};
        map<string, float> result3;
        for (int m=0; m<starts.size(); m++)
        {
            result3[starts[m]]=SPY.calculateweekly(starts[m], ends[m]);
            cout<<result3[starts[m]]<<endl;
        }
        cout<<"first month SPY:"<<calculatemothly(result3)<<endl;

        vector<string> starts2={"2019-02-04", "2019-02-11", "2019-02-19", "2019-02-25"};
        vector<string> ends2={"2019-02-08", "2019-02-15", "2019-02-22", "2019-03-01"};
        map<string, float> result4;
        for (int m=0; m<starts2.size(); m++)
        {

```

```

    result4[starts2[m]]=SPY.calculateweekly(starts2[m], ends2[m]);
    cout<<result4[starts2[m]]<<endl;
}
cout<<"second month SPY:"<<calculatemonthly(result4)<<endl;

vector <string> starts3={"2019-03-04", "2019-03-11", "2019-03-18", "2019-03-25"};
vector <string> ends3={"2019-03-08", "2019-03-15", "2019-03-22", "2019-03-29"};
map<string, float> result5;
for (int m=0; m<starts3.size(); m++)
{
    result5[starts3[m]]=SPY.calculateweekly(starts3[m], ends3[m]);
    cout<<result5[starts3[m]]<<endl;
}
cout<<"third month SPY:"<<calculatemonthly(result5)<<endl;

vector <string> starts4={"2019-04-01", "2019-04-08", "2019-04-15", "2019-04-22"};
vector <string> ends4={"2019-04-05", "2019-04-12", "2019-04-18", "2019-04-26"};
map<string, float> result6;
for (int m=0; m<starts4.size(); m++)
{
    result6[starts4[m]]=SPY.calculateweekly(starts4[m], ends4[m]);
    cout<<result6[starts4[m]]<<endl;
}
cout<<"fourth month SPY:"<<calculatemonthly(result6)<<endl;

vector <string> starts5={"2019-04-29", "2019-05-06", "2019-05-13", "2019-05-20", "2019-05-
28"};
vector <string> ends5={"2019-05-03", "2019-05-10", "2019-05-17", "2019-05-24", "2019-05-
31"};
map<string, float> result7;
for (int m=0; m<starts5.size(); m++)
{
    result7[starts5[m]]=SPY.calculateweekly(starts5[m], ends5[m]);
    cout<<result7[starts5[m]]<<endl;
}
cout<<"fifth month SPY:"<<calculatemonthly(result7)<<endl;

vector <string> starts6={"2019-06-03", "2019-06-10", "2019-06-17", "2019-06-24"};
vector <string> ends6={"2019-06-07", "2019-06-14", "2019-06-21", "2019-06-28"};
map<string, float> result8;
for (int m=0; m<starts6.size(); m++)
{
    result8[starts6[m]]=SPY.calculateweekly(starts6[m], ends6[m]);
    cout<<result8[starts6[m]]<<endl;
}
cout<<"sixth month SPY:"<<calculatemonthly(result8)<<endl;

cout<<"six months average
SPY:"<<calculatemonthly(result3)+calculatemonthly(result4)+calculatemonthly(result5)+calculatemonthly(re
sult6)+calculatemonthly(result7)+calculatemonthly(result8)<<endl;

```

```

vector <string> starts7={ "2019-07-01", "2019-07-08", "2019-07-15", "2019-07-22"};
vector <string> ends7={ "2019-07-05", "2019-07-12", "2019-07-19", "2019-07-26"};
map<string, float> result9;
for (int m=0; m<starts7.size(); m++)
{
    result9[starts7[m]]=SPY.calculateweekly(starts7[m], ends7[m]);
    cout<<result9[starts7[m]]<<endl;
}
cout<<"seventh month SPY:"<<calculatemothly(result9)<<endl;
break;
}
case '7':
{
    cout<<"7 - backtesting for best portfolio 2019. \n";
    vector <string> bestsymbols;
    for (int i=0; i<(*pBest).getstocks().size(); i++)
    {
        bestsymbols.push_back((*pBest).getstocks()[i].getsymbol());
    }
    vector<Stock> beststocks;
    beststocks=(*pBest).getstocks();
    for (int j=0; j<bestsymbols.size(); j++)
    {
        beststocks[j].cleartrades();
        string stock_select = "SELECT * FROM STOCK_" + bestsymbols[j] + " WHERE
strftime('%Y-%m-%d', date) between \"2019-01-01\" and \"2019-07-31\" and open > 0 and
adjusted_close > 0;";
        if (stock_prices(stockDB, stock_select, beststocks[j])== -1)
            return -1;
        beststocks[j].calculate_returns();
    }

    //CloseDatabase(stockDB);

    Portfolio testp(beststocks);
    testp.calculateweights(beststocks);
    testp.Assignfitness();
    myfile<<testp<<",\n";

    vector <string> starts={ "2019-01-07", "2019-01-14", "2019-01-22", "2019-01-28"};
    vector <string> ends={ "2019-01-11", "2019-01-18", "2019-01-25", "2019-02-01"};
    map<string, float> result1;
    for (int m=0; m<starts.size(); m++)
    {
        result1[starts[m]]=(testp).calculateweekreturn(starts[m], ends[m]);
        cout<<result1[starts[m]]<<endl;
        myfile<<result1[starts[m]]<<",\n";
        //myfile<<"csv\n";
    }
}

```



```

float first=calculatemothly(result1);
cout<<"first month:"<<first<<endl;

vector <string> starts2={"2019-02-04", "2019-02-11", "2019-02-19", "2019-02-25"};
vector <string> ends2={"2019-02-08", "2019-02-15", "2019-02-22", "2019-03-01"};
map<string, float> result2;
for (int m=0; m<starts2.size(); m++)
{
    result2[starts2[m]]=(testp).calculateweekreturn(starts2[m], ends2[m]);
    cout<<result2[starts2[m]]<<endl;
    myfile<<result2[starts2[m]]<<","<<endl;
}
float second=calculatemothly(result2);
cout<<"second month:"<<second<<endl;

vector <string> starts3={"2019-03-04", "2019-03-11", "2019-03-18", "2019-03-25"};
vector <string> ends3={"2019-03-08", "2019-03-15", "2019-03-22", "2019-03-29"};
map<string, float> result3;
for (int m=0; m<starts3.size(); m++)
{
    result3[starts3[m]]=(testp).calculateweekreturn(starts3[m], ends3[m]);
    cout<<result3[starts3[m]]<<endl;
    myfile<<result3[starts3[m]]<<","<<endl;
}
float third=calculatemothly(result3);
cout<<"third month:"<<third<<endl;

vector <string> starts4={"2019-04-01", "2019-04-08", "2019-04-15", "2019-04-22"};
vector <string> ends4={"2019-04-05", "2019-04-12", "2019-04-18", "2019-04-26"};
map<string, float> result4;
for (int m=0; m<starts4.size(); m++)
{
    result4[starts4[m]]=(testp).calculateweekreturn(starts4[m], ends4[m]);
    cout<<result4[starts4[m]]<<endl;
    myfile<<result4[starts4[m]]<<","<<endl;
}
float fourth=calculatemothly(result4);
cout<<"fourth month:"<<fourth<<endl;

vector <string> starts5={"2019-04-29", "2019-05-06", "2019-05-13", "2019-05-20", "2019-05-
28"};
vector <string> ends5={"2019-05-03", "2019-05-10", "2019-05-17", "2019-05-24", "2019-05-
31"};
map<string, float> result5;
for (int m=0; m<starts5.size(); m++)
{
    result5[starts5[m]]=(testp).calculateweekreturn(starts5[m], ends5[m]);
    cout<<result5[starts5[m]]<<endl;
    myfile<<result5[starts5[m]]<<","<<endl;
}

```

```

float fifth=calculatemothly(result5);
cout<<"fifth month:"<<fifth<<endl;

vector <string> starts6={"2019-06-03", "2019-06-10", "2019-06-17", "2019-06-24"};
vector <string> ends6={"2019-06-07", "2019-06-14", "2019-06-21", "2019-06-28"};
map<string, float> result6;
for (int m=0; m<starts6.size(); m++)
{
    result6[starts6[m]]=(testp).calculateweekreturn(starts6[m], ends6[m]);
    cout<<result6[starts6[m]]<<endl;
    myfile<<result6[starts6[m]]<<"\n";
}
float sixth=calculatemothly(result6);
cout<<"sixth month:"<<sixth<<endl;

cout<<"six months average:"<<first+second+third+fourth+fifth+sixth<<endl;

vector <string> starts7={"2019-07-01", "2019-07-08", "2019-07-15", "2019-07-22"};
vector <string> ends7={"2019-07-05", "2019-07-12", "2019-07-19", "2019-07-26"};
map<string, float> result7;
for (int m=0; m<starts7.size(); m++)
{
    result7[starts7[m]]=(testp).calculateweekreturn(starts7[m], ends7[m]);
    cout<<result7[starts7[m]]<<endl;
    myfile<<result7[starts7[m]]<<"\n";
}
cout<<"seventh month:"<<calculatemothly(result7)<<endl;
break;
}
case '8':
{
    cout<<"8 - prob testing. \n";
    vector <string> bestsymbols;
    for (int i=0; i<(*pBest).getstocks().size(); i++)
    {
        bestsymbols.push_back((*pBest).getstocks()[i].getsymbol());
    }
    vector<Stock> beststocks;
    beststocks=(*pBest).getstocks();
    for (int j=0; j<bestsymbols.size(); j++)
    {
        beststocks[j].cleartrades();
        string stock_select = "SELECT * FROM STOCK_" + bestsymbols[j] + " WHERE
strftime('%Y-%m-%d', date) between \"2019-01-01\" and \"2019-07-31\" and open > 0 and
adjusted_close > 0;";
        if (stock_prices(stockDB, stock_select, beststocks[j])== -1)
            return -1;
        beststocks[j].calculate_returns();
    }
}

```

```

Portfolio testp(beststocks);
testp.calculateweights(beststocks);
testp.Assignfitness();

Stock SPY("SPY");
string c = "SELECT * FROM FUNDAMENTALS;";
if (stock_fundamentals(stockDB, c, SPY, 506)==-1)
    return -1;
string stock_select = "SELECT * FROM STOCK_SPY WHERE strftime('%Y-%m-%d', date)
between \"2019-07-01\" and \"2019-07-31\" and open > 0 and adjusted_close > 0;";
if (stock_prices(stockDB, stock_select, SPY)==-1)
    return -1;
SPY.calculate_returns();

vector <string> starts7={"2019-07-01", "2019-07-08", "2019-07-15", "2019-07-22"};
vector <string> ends7={"2019-07-05", "2019-07-12", "2019-07-19", "2019-07-26"};
map<string, float> result7;
for (int m=0; m<starts7.size(); m++)
{
    result7[starts7[m]]=(testp).calculateweekreturn(starts7[m], ends7[m]);
    cout<<result7[starts7[m]]<<endl;
}
cout<<"seventh month:"<<calculatemothly(result7)<<endl;

map<string, float> result9;
for (int m=0; m<starts7.size(); m++)
{
    result9[starts7[m]]=SPY.calculateweekly(starts7[m], ends7[m]);
    cout<<result9[starts7[m]]<<endl;
}
cout<<"seventh month SPY:"<<calculatemothly(result9)<<endl;
break;
}
case '0':
{
    cout<<"0 - Exit the program.\n";
    // Close Database
    CloseDatabase(stockDB);
    cout<<"finished"<<endl;
    // delete pBest;
    // pBest=NULL;
    myfile.close();
    continue_menu = false;
    exit(1);
}
default:
{
    cout<<"Invalid input!\n";
    break;
}
}

```

```
    }  
}  
CloseDatabase(stockDB);  
  
cout<<"finished"<<endl;  
delete pBest;  
pBest=NULL;  
return 0;  
}
```

# Bibliography

- [1] Chi-Ming Lin and Mitsuo Gen. An effective Decision-based Genetic in Multi-objective Portfolio Optimization Problem, *Applied Mathematical Sciences*, 1(5), (2007), 201-210.
- [2] Luciano Siracusano, Considering smart beta, A real time update, *Journal of indexes*, 2014.
- [3] Slimane Sefiane and Mohamed Benbouziane, Portfolio Selection Using Genetic Algorithm, *Journal of Applied Finance & Banking*, vol.2, no.4, 2012, 143-154.
- [4] C. Aranba and H. Iba, The Mimetic Tree-based Genetic Algorithm and its application to portfolio optimization, *Springer Mimetic Comp.*, 1, (2009), 139-151.
- [5] Tsai CF, Lin YC, Yen DC, Chen YM. Predicting stock returns by classifier ensembles, *Applied Soft Computing*, 2011; 11: 2452–2459.
- [6] Allen F, Karjalainen R. Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*. 1999; 51: 245–271.
- [7] Brian H. McCord, Predicting Stock Trending in a Financial Market with Neural Networks and Genetic Algorithms, Graduate project technical report, summer 2003.
- [8] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.
- [9] R. Lakshman Naik<sup>1</sup>, D. Ramesh, B. Manjula, Dr. A. Govardhan. Prediction of Stock Market Index Using Genetic Algorithm. *Computer Engineering and Intelligent Systems*.

- [10] R. Tsaih, Hsu, Y. and Lai C. C., “ Forecasting S&P 500 Stock index futures with a hybrid AI system”, Decision support Systems, 1998, pp. 161-174.
- [11] L. Davis, Handbook of genetic algorithms, Van Nostrand Reinhold, NY, 1991.