Final Year Project Report

Full Unit – Final Report

Value at Risk

Shing Him Yip

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Supervisor Name

Department of Computer Science

Royal Holloway, University of London

December 07, 2022

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Shing Him Yip

Date of Submission:

Signature:

Table of Contents

Table of Contents

Abstract **4**

Project Specification **5**

Chapter 1: Introduction **6**

1.1 The Problem *6*

1.2 Aims and Goals of the Project *6*

1.3 Survey of Related Literature *6*

1.4 Milestones Summary (timeline) *6*

Chapter 2: Value at Risk (VaR) **8**

2.1 What is Value at Risk *8*

2.2 Value at Risk (VaR) *8*

2.2.1 Mathematical definition 9

2.3 Conditional Value at Risk (CVaR) *9*

2.3.1 Mathematical definition 10

2.4 Time Horizon *10*

2.5 Historical Returns *10*

2.6 Historical Simulation *10*

2.6.1 Historical Simulation – Single Asset 11

2.6.2 Historical Simulation – Portfolio 12

2.7 Model Building method *13*

2.7.1 Daily Volatilities 13

2.7.2 Model Building method – Single Asset 13

2.7.3 Model Building method – Portfolio 14

Chapter 3: Software engineering **16**

3.1 Methodology *16*

3.1.1 Test-driven development (TDD) 16

3.1.2 TDD stages and cycle 16

3.2 Testing *16*

3.2.1 Unit test 16

3.2.2 High level test 17

3.2.3 Back test 17

3.2.4 Simple statistics test 17

3.3 Documentation *18*

3.3.1 Stock_data package 18

3.3.2 Calculating_VaR package 18

data_initialise class 18

Historical_Simulation class 19

parametric_method class 20

3.4 UML *21*

Chapter 4: Proof-of-concepts Development **22**

4.1 Data initialise *22*

4.2 Historical Simulation method *24*
4.2.1 VaR calculation 24
4.2.2 CVaR calculation 25

4.3 Model building method *28*
4.3.1 VaR calculation 28

Chapter 5: Back testing **33**

Chapter 6: User menu ..... Error! Bookmark not defined.

Appendix **36**

Bibliography **37**

Abstract

Nowadays in the modern finance environment, the computer is indispensable. There is a lot of usage in this field, such as sending financial reports and strategies or simply holding meetings. In this project, I want to talk about strategies specifically. By definition, a finance strategy is an approach for the planned development of the Finance function based on a clearly defined vision, strategy, and roadmap. It helps to build on insights from a business context, stakeholder expectations, and own performance & capabilities to focus on opportunities that create value.

So, in this case the strategy that I want to talk about most is values at risk. As Buffett said controlling risk is an important part of investing. What is Value at risk (VaR), by definition It is a statistic used to try and quantify the level of financial risk within a firm or portfolio over a specified time frame. VaR provides an estimate of the maximum loss from a given position or portfolio over a period of time, and you can calculate it across various confidence levels. So, what I will do in the project is I will try to make a program for computing the risk, then check its performance using back testing.

Project Specification

Your project specification goes here.

Chapter 1: **Introduction**

1.1 The Problem

The capacity to create higher returns while limiting risk is essential for a successful investment. Risk is a significant component that investors and traders evaluate when making investment decisions, and it is frequently the key determinant in accepting or rejecting an asset or security. However, while deciding whether to buy or sell an investment, the first worry for any investor is the greatest amount they may lose or the entire value at risk.
During the project, I will try to make an app that can simplify the mathematical formula to some simple button to help user to value their own investment portfolio risk easily.

1.2 Aims and Goals of the Project

There are several aim need to achieved in the first term

- Implement the historical simulation for calculating VaR in single stock
- Implement the historical simulation for calculating CVaR in single Stock
- Implement the historical simulation for calculating VaR for portfolio
- Implement the historical simulation for calculating CVaR in portfolio
- Implement the model building method for calculating VaR in single stock
- Do back testing on the result

1.3 Survey of Related Literature

Please see Bibliography section

1.4 Milestones Summary (timeline)

Week 1:
During this week I studying about what is Value at Risk, and how to do the calculation then find some related resources about the topic.

Week 2:
In this week, I started to plan what I need to do for the first term , then I try to learn about how to use python packages (yfinance , NumPy ,pandas ,matplotlib), because I need to use those package for doing data analysing .

Week 3 :
In this week I try to use Jupyter notebook to implement the simple calculation and try the package that I learn in Week 2 .Also in this week, I have successfully initialization the data I get from the market by using yfinance package.

Week 4:

During this week I have successfully implemented a simple proof-of-concept program for computing Value at Risk using the historical simulation method for both single stock and portfolio.

Week 5:
During this week I have successfully implemented a simple proof-of-concept program for computing Conditional Value at Risk using the historical simulation method for both single stock and portfolio.

Week 6:
During this week I have successfully implemented an upgrade to the function for calculating Conditional Value at Risk and the function for calculating Value at Risk to accept different variables for calculating The CVaR and VaR

Week 7 :
During this week I have I have successfully implemented a simple proof-of-concept program for computing Value at Risk using the model building method for single stock.

Week 8 :
During this week I refactor the code for computing the VaR by using historical simulation method and refactor the code for data initialising .

Week 9 :
During this week , I try to implement a simple proof-of-concept program for computing Value at Risk using the model building method for portfolio.

Week 10:
During this week I am continuing to work on the model-building method for the portfolio. In the meantime implement the backtesting on the historical simulation method for a single stock.

Week 11:
Prepare for the midterm submission and the presentation

Chapter 2: **Value at Risk (VaR)**

2.1 What is Value at Risk

Value at risk is a simple way to describe the magnitude of the likely losses on the portfolio. Based on simplified assumptions used in the calculation, VaR aggregates all risks in a portfolio into a single number that applies to the board, is reported to regulators, or is disclosed in annual reports. By the definition, it is an attempt to provide a single number summarizing the total risk in a portfolio of financial assets. But in simple way to explain it is a single, summary, statistical measure of possible portfolio losses from the market that have a "normal" market movement.

2.2 Value at Risk (VaR)

When using the Value at Risk measure, people will make the statement as follow
I am **X** percent certain there will not be a loss of more than **V** dollars in the next **N** days. we are interested in 3 variables which are

1. V it is the VaR of the portfolio
2. N is the time horizon (N days), normally will be 1 – 5 days
3. X is the confidence level (X %), we normally will put 99, 95 ,90

For example, we will say I am **5** percent certain there will not be a loss of more than **1000** dollars in the next **2** days.

VaR is the loss corresponding to the (100 - x) percentile of the distribution of the gain in the value of the portfolio over the upcoming N days when N days is the time horizon and X% is the confidence level. (Remark: When we look at the probability distribution of losses, the gains are negative losses and VaR is related to the right tail of the distribution. When we look at the probability distribution of returns, losses are negative returns and VaR is related to the left tail of the distribution.) For example, When N = 2 and X = 99 VaR is the first percentile of the distribution of gain in the value of the portfolio over 2 days.

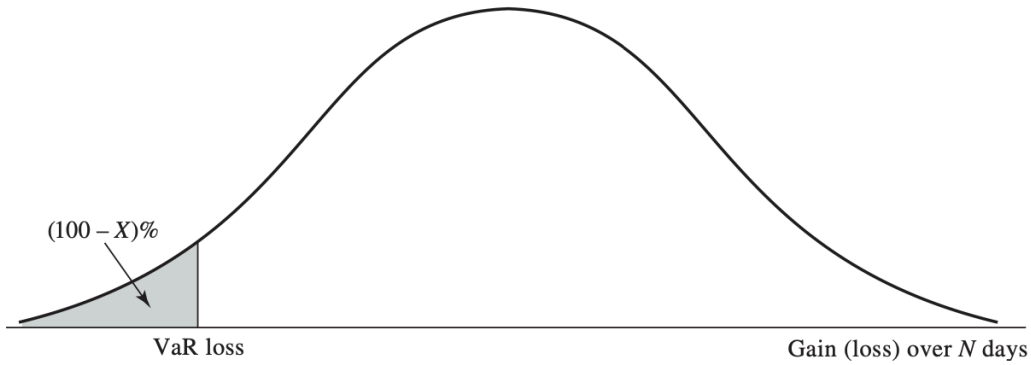


Figure 1 Calculation of VaR from the probability distribution of the change in the portfolio value; confidence level is X% .Gains in portfolio value are positive ; losses are negative [1]

[1] Options, Futures, and Other Derivatives p.496

2.2.1 Mathematical definition[2]

Mathematically we can define as follow :
Let X be the distribution of profit and loss. (Profit is positive and loss is negative) .The VaR at level $\alpha \in (0,1)$ is the smallest number y such that the probability that $Y := -X$ does not exceed y is at least $1-\alpha$ .Mathematically, $VaR_\alpha(X)$ is the $1-\alpha$ quantile of Y

$$VaR_\alpha(X) = -inf\ \{x \in \mathbb{R} : F_X(x) > \alpha\} = F_Y^{-1}(1-\alpha)$$

This is the most general definition of VaR, and the two identities are equivalent (in fact, for any real random variable X, its cumulative distribution function F_X is well-defined). However, this formula cannot be used directly for calculations unless we assume that X has some parametric distribution.

2.3 Conditional Value at Risk (CVaR)

Conditional Value at Risk (CVaR), commonly referred to as the expected shortfall or Mean Excess Loss, it estimates the amount of tail risk present in an investment portfolio. A weighted average of the "extreme" losses in the tail of the distribution of potential returns, above the value at risk (VaR) cut-off point, is used to calculate CVaR. Conditional value at risk is used in portfolio optimization for effective risk management.

Generally speaking, the VaR may be enough for risk management in a portfolio that contains an investment if it shows stability over time. However, because VaR is unconcerned with anything that goes above its own threshold, it is more likely that VaR will not accurately reflect risk for investments that are more volatile.

The Conditional Value at Risk (CVaR) model seeks to address the shortcomings of the VaR model, a statistical technique used to assess the level of financial risk in a company or portfolio over a specified time horizon. CVaR is the expected loss when the worst-case threshold is exceeded, whereas VaR represents the worst-case loss in terms of probability and time horizon. CVaR, in other words, quantifies the expected loss above the VaR breakpoint.

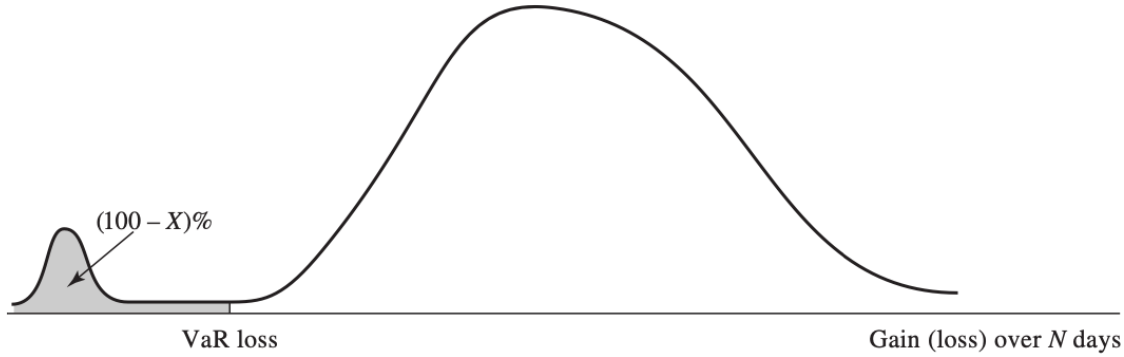


Figure 2 Alternative situation to Figure 1. VaR is the same, but the potential loss is larger.

[2] Artzner, P., Pasteur, L., Strasbourg, F., Delbaen, T., Hochschule, Zürich, J.-M., Société Eber and Générale,P. (1996). COHERENT MEASURES OF RISK. [online] Available at: https://people.math.ethz.ch/~delbaen/ftp/preprints/CoherentMF.pdf.

2.3.1 Mathematical definition[3]

Since the CVaR value is derived from the calculation of the VaR itself, the assumptions on which the VaR is based, such as the shape of the return distribution, the cut-off levels used, the periodicity of the data, and assumptions about stochastic volatility, can affect the value of the CVaR. Once VaR is calculated, calculating CVaR is straightforward. It is the average of values beyond VaR:

$$CVaR = \frac{1}{1-c}\int_{-1}^{VaR} xp\,(x)\,dx$$

$p(x)dx$ = the probability density of getting a return with value "x"
c = the cut-off point on the distribution where the analyst sets the VaR breakpoint
VaR = the agreed-upon VaR level

2.4 Time Horizon[4]

For calculating the VaR we need two parameters: Time horizon N (days) and the confidence level X . In practise, when estimating the VaR for market risk, analysts almost always start with N = 1. This is due to the fact that there is frequently insufficient data to directly estimate the behaviour of market variables over time periods longer than one day. The assumption is :

$$\text{N-day VaR} = \text{1 -day VaR} \times \sqrt{N}$$

When the changes in portfolio value over consecutive days have independent identical normal distributions with a mean of 0, the formula is completely correct. In other cases, it is a rough estimate.

2.5 Historical Returns

To explain it mathematically, suppose today is Day n, the closing price is C,and the returns of a stock variable on Day i will be defined as r . In the historical simulation approach, the i th scenario of the market returns will be

$$\text{Returns under } i \text{ th scenario } r_i = \frac{C_{i+1}-C_i}{C_i}$$

2.6 Historical Simulation

Historical simulation is one of the easy and popular way of calculating VaR It entails using past data to predict what will happen in the future. Let said we want to calculate the VaR for a portfolio using a one-day time horizon with 99% confidence level and 501 days of data. (We normally will put 99, 95 ,90 to the confidence level and 501 is a popular choice for the number of days of data used because this provides 500 alternative scenarios for what can happen between today and tomorrow.)

Data will be collected by the movements in the market variables over the most recent 501 days. Day 0 represents the first day for which we have data, Day 1 represents the second day, and so on. Scenario 1 is the same as Day 0 and Day 1, Scenario 2 is the same as Day 1

[3] Chen, J. (2019). Conditional Value at Risk (CVaR). [online] Investopedia. Available at: https://www.investopedia.com/terms/c/conditional_value_at_risk.asp.

[4] Options, Futures, and Other Derivatives p.496

and Day 2, and so on. The dollar changes in the portfolio's value between today and tomorrow is calculated for each scenario. This defines a probability distribution for the value of our portfolio's daily loss (gains are negative losses). The distribution's 99th percentile can be estimated as the fifth-highest loss. The loss when we reach the 99th percentile is the estimate of VaR If the changes in market variables over the last 501 days are representative of what will happen between today and tomorrow, we are 99% certain that we will not take a loss greater than the VaR estimate.

2.6.1 Historical Simulation – Single Asset

Let said we have 10K investing in Apple. We are interested in the loss level over 5 days that we are 99% certain will not be exceeded in a 1-day time horizon. So N = 5 and X = 99. First, we need to have a table listing all the historical data

Day	Date	Price
0	2020-12-01 00:00:00-05:00	121.264305
1	2020-12-02 00:00:00-05:00	121.620033
2	2020-12-03 00:00:00-05:00	121.481689
3	2020-12-04 00:00:00-05:00	120.799866
⋮		
498	2022-11-22 00:00:00-05:00	150.179993
499	2022-11-23 00:00:00-05:00	151.070007
500	2022-11-25 00:00:00-05:00	148.110001

Table 1 : 501 days closing price for Apple

Then we calculated their return , and use the above table to create a scenario table

Scenario number	Date	Close	returns
1	2020-12-02 00:00:00-05:00	121.6200409	0.00293356
2	2020-12-03 00:00:00-05:00	121.4816895	-0.0011376
3	2020-12-04 00:00:00-05:00	120.7998734	-0.0056125
⋮			
499	2022-11-23 00:00:00-05:00	151.0700073	0.00592632
500	2022-11-25 00:00:00-05:00	148.1100006	-0.0195936

Table 2 : 500 scenario for Apple

Then we sorted the table 2 , from largest loss to profit

Scenario number	Date	Close	returns
448	2022-09-13 00:00:00-04:00	153.5852203	-0.0586795
368	2022-05-18 00:00:00-04:00	140.3917847	-0.0564192
359	2022-05-05 00:00:00-04:00	156.0639801	-0.0557161
363	2022-05-11 00:00:00-04:00	146.0545044	-0.0518412
460	2022-09-29 00:00:00-04:00	142.2440338	-0.049119
485	2022-11-03 00:00:00-04:00	138.6500092	-0.0424049
⋮			
481	2022-10-28 00:00:00-04:00	155.4820862	0.07555254
490	2022-11-10 00:00:00-05:00	146.8699951	0.08897457

Table 3

Since we are interesting in 99%VaR it mean we need to that the take 1% of the total day , it mean $500 \times 1\% = 5$ days . we will take the fifth scenario, in this case will be 460 , So the VaR will be

$$10000 \times |-0.049119| = \$491.19$$

So for N day VaR it will calculate by $\sqrt{N}$ times the 1-day VaR , so the 5-days VaR for apple is :

$$\sqrt{5} \times 491.19 = \$1098.33$$

2.6.2 Historical Simulation – Portfolio

Let said we have 10K investing in the Portfolio which contains 5 stocks which is TSM ,TSLA GOOGL,MSFT and AAPL with weights of 20%,15%,15% 30% and 20 % . We are interested in the loss level over 5 days that we are 99% certain will not be exceeded in 1 day time horizon. So N = 5 and X = 99.

For Portfolio we basically do the same things, but this time we need to have the portfolio weight(Portfolio) columns ,so here we need to do is get the closing price .

Day	Date	TSM	GOOGL	TSLA	MSFT	AAPL
0	2020-12-02 00:00:00-05:00	96.04611206	91.24849701	189.6066742	211.5969849	121.6200333
1	2020-12-03 00:00:00-05:00	96.00751495	91.09200287	197.793335	210.4868011	121.4816971
2	2020-12-04 00:00:00-05:00	100.0890503	91.18800354	199.6799927	210.6046906	120.7998734
3	2020-12-07 00:00:00-05:00	102.6556854	90.85150146	213.9199982	210.5359344	122.2820816
			⋮			
498	2022-11-23 00:00:00-05:00	81.97000122	98.45999908	183.1999969	247.5800018	151.0700073
499	2022-11-25 00:00:00-05:00	81.40000153	97.45999908	182.8600006	247.4900055	148.1100006
500	2022-11-28 00:00:00-05:00	80.70999908	96.79000092	184.1999969	245.0899963	145.9949951

Table 4

Then replace the closing price with returns and use dot product to create a new columns Portfolio with the given weight 20%,15%,15% 30% and 20% (which is the combined returns with all stock provide).

Scenario number	Date	TSM	GOOGL	TSLA	MSFT	AAPL	Portfolio
1	2020-12-03 00:00:00-05:00	-0.000401781	-0.001715033	0.043177071	-0.005246905	-0.001137383	0.0043374
2	2020-12-04 00:00:00-05:00	0.042512582	0.001053887	0.00953853	0.000560152	-0.005612626	0.0091369
3	2020-12-07 00:00:00-05:00	0.025643592	-0.003690201	0.071314133	-0.00032647	0.012270013	0.01762837
				⋮			
499	2022-11-25 00:00:00-05:00	-0.00695376	-0.010156409	-0.001855875	-0.000363504	-0.019593609	-0.0072204
500	2022-11-28 00:00:00-05:00	-0.010810871	-0.008670236	0.016542677	-0.012808591	-0.017148031	-0.0082535

Table 5

Then we sort Table 5 , from the largest loss to profit

Scenario number	Date	TSM	GOOGL	TSLA	MSFT	AAPL	Portfolio
358	2022-05-05 00:00:00-04:00	-0.040004201	-0.047075525	-0.083286153	-0.043554757	-0.055716186	-0.0517648
447	2022-09-13 00:00:00-04:00	-0.040657382	-0.058993333	-0.040371881	-0.054978336	-0.058679524	-0.0512657
351	2022-04-26 00:00:00-04:00	-0.036055669	-0.035945822	-0.121841219	-0.037403734	-0.037328058	-0.0495659
465	2022-10-07 00:00:00-04:00	-0.061869517	-0.027016347	-0.063242755	-0.050852865	-0.036718733	-0.0485124
367	2022-05-18 00:00:00-04:00	-0.029897211	-0.039266625	-0.068013798	-0.045529703	-0.056419178	-0.0470143
360	2022-05-09 00:00:00-04:00	-0.047255263	-0.027953299	-0.090729488	-0.036945477	-0.033189158	-0.0449749
				⋮			
414	2022-07-27 00:00:00-04:00	0.038024174	0.076556857	0.061655037	0.066851944	0.034234672	0.05523914
489	2022-11-10 00:00:00-05:00	0.089846098	0.075813133	0.073934372	0.082268032	0.088974571	0.08290667

Table 6

Since we are interested in 99%VaR it means we need to that 1% of the total day , it means $500 \times 1\% = 5$ days . we will take the fifth scenario, in this case , it will be 367, So the VaR will be :

$$10000 \times |-0.0470143| = \$470.14$$

So for N day VaR it will calculate by $\sqrt{N}$ times the 1-day VaR , so the 5-days VaR for apple is :

$$\sqrt{5} \times 470.14 = \$1051.27$$

2.7 Model Building method

The model-building method is the primary alternative to historical simulation. Before delving into the method, we need to define clearly the volatility unit first.

2.7.1 Daily Volatilities

In option pricing, time is normally defined in years, and an asset's volatility is usually expressed as a "volatility per year." When utilising the model-building approach to calculate VaR for market risk, time is typically measured in days, and an asset's volatility is typically expressed as a "volatility per day."

Let's define the volatility per year of a certain asset be σ_{year} and the equivalent volatility per day of the asset as σ_{day}. Assume a year has 252 trading days , the daily volatility will be :

$$\sigma_{day} = \frac{\sigma_{year}}{\sqrt{252}}$$

So the daily volatility will be around 6% of annual volatility. In calculating VaR, we assume that the daily volatility of a stock is equal to the standard deviation of the percentage change within a day.

2.7.2 Model Building method – Single Asset [5]

Let said we have 10K in a shares of Apple . We are interested in the loss level over 5 days that we are 99% certain will not be exceeded in 1 day time horizon. So N = 5 and X = 99 , Assuming the daily volatility of Apple is 2% (32 % in a year) the standard deviation of daily change will be 2 % of 10K which is $ 200.

In the model-building methods, it is common to assume that the predicted change in a market variable over the time period under consideration is zero. Although this is not precisely correct, it is a valid assumption. When compared to the standard deviation of the change, the expected change in the price of a market variable over a short time period is often tiny. For example, Apple is predicted to yield 20% every year. The expected return during a one-day period is 0.20/252, or around 0.08%, with a standard deviation of 2%. The predicted return over a 10-day period is 0.08 x 10, or about 0.8%, while the standard deviation of the return is $2\sqrt{5}$, or about 4.5%.

[5] Options, Futures, and Other Derivatives p.497

From the standard normal cumulative distribution function we know that $N^{-1}(0.01) = 2.326$. This means that there is a only 1% chance that the value of a normally distributed variable will change by more than 2.326 standard deviations. It also means that we are 99% certain that the value of a normally distributed variable will not decrease by more than 2.326 standard deviations. So let's put the number together

$$2.326 \times 200 = \$465.2$$

So for N day VaR it will calculate by $\sqrt{N}$ times the 1-day VaR , so the 5-days VaR for apple is :

$$\sqrt{5} \times 465.2 = \$1040.2$$

2.7.3 Model Building method – Portfolio

Let said we have 10K investing in the Portfolio which contains 5 stocks which is TSM ,TSLA GOOGL,MSFT and AAPL with weights 20%,15%,15% 30% and 20 % . We are interested in the loss level over 5 days that we are 99% certain will not be exceeded in 1 day time horizon. So N = 5 and X = 99.

By the model-building methods, we also need to gather all the data (see Table 4) , then we calculate their returns

Scenario number	Date	TSM	GOOGL	TSLA	MSFT	AAPL
1	2020-12-03 00:00:00-05:00	-0.000401781	-0.001715033	0.043177071	-0.005246905	-0.001137383
2	2020-12-04 00:00:00-05:00	0.042512582	0.001053887	0.00953853	0.000560152	-0.005612626
3	2020-12-07 00:00:00-05:00	0.025643592	-0.003690201	0.071314133	-0.00032647	0.012270013
499	2022-11-25 00:00:00-05:00	-0.00695376	-0.010156409	-0.001855875	-0.000363504	-0.019593609
500	2022-11-28 00:00:00-05:00	-0.010810871	-0.008670236	0.016542677	-0.012808591	-0.017148031

Table 6

Then we need to calculate the covariance for each stock, by using the formula below:

$$cov(R1, R2) = \frac{1}{n} \sum_{i=1}^{n} r1_i \times r2_i$$

In the formula R1 and R2 is representing all the returns for Stock 1 and Stock 2 , n is how many days we have in the data , $r1$ and $r2$ will be the daily return . In this example, we have 5 stocks, so we will have 25 covariances

	TSM	GOOGL	TSLA	MSFT	AAPL
TSM	0.000530	0.000254	0.000407	0.000214	0.000246
GOOGL	0.000254	0.000401	0.000342	0.000292	0.000273
TSLA	0.000407	0.000342	0.001435	0.000333	0.000409
MSFT	0.000214	0.000292	0.000333	0.000324	0.000265
AAPL	0.000246	0.000273	0.000409	0.000265	0.000373

Then we need to calculate the standard deviation σ for the portfolio. With n stock the formula will be :

$$\sigma = \sqrt{\sum_{i,k}^{K} a_i a_k \, cov(Ri, Rk)}$$

In this formula a is the amount of money to put into this stock

In our example, we have initial investment 10K , we need to extract the amount with the weight So $a_1 = 10000 \times 20\% = 2000$, $a_2 = 10000 \times 15\% = 1500$ …

Then to calculate the σ we need to

$$\sigma = \sqrt{2000 \times 2000 \times 0.000530 + 2000 \times 1500 \times 0.000254 \ldots 2000 \times 2000 \times 0.000373}$$

$$\sigma = 183.42$$

Then VaR can be calculated by Percentile point function, so we give the function our confidence level which is $\frac{95}{100}$ and standard deviation 183.43. As a result the VaR will be $ 301.79

Chapter 3: **Software engineering**

3.1 Methodology

There are several ways to develop a software. In this report, I will mainly use Test-driven development (TDD) as the main method for developing the software.

3.1.1 Test-driven development (TDD)

Test-driven development (TDD) by definition is a software development process relying on software requirements being converted to test cases before the software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is as opposed to software being developed first and test cases being created later. What this means is that before we write code, we consider what the code will do, and then we write a test that uses methods you haven't even written yet. Using this methoåd, the tests indicate intent and drive the code development.

3.1.2 TDD stages and cycle [6]

1. Write a test.
2. Compile the test. It may not compile. The code's not written.
3. Write just enough code to get the test to compile.
4. Run the test and see it fail.
5. Write just enough code to pass the test
6. Run all the tests a few times to enjoy seeing your code pass.
7. Refactor.
8. Repeat from step 1

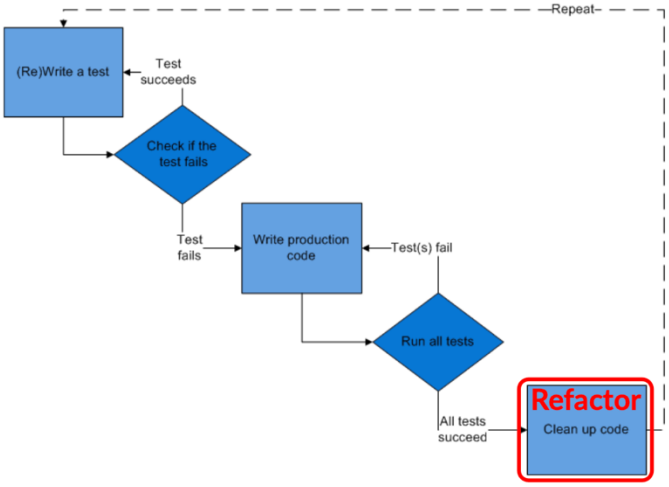


Figure 3 TDD cycle

3.2 Testing

In this project I many use 4 type of testing , unit test while doing TDD , back test to test my model ,and high level testing to check the output

3.2.1 Unit test

unit testing by definition , is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for us. [7]

[6] CS2800 Topic 2 power point
[7] Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 75

Since we are using TDD method to develop the software every function will at least have one test case with it .

3.2.2 High level test

For the high level testing I have compared the model building method and historical simulation method with Initial Investment = 10000 , 95% VaR and 1 Time Horizon , with 501 day date.
The result show as follows

	Single stock (TSLA)	Portfolio (TSM,GOOGL, TSLA ,MSFT ,AAPL)
Historical Simulation	$ 655.21	$ 326.71
Model Build 1	$ 617.61	$ 299.85
Model Build 2	$ 623.26	$ 303.93

So here we can see for the single stock there is only 5.4% and for the portfolio is only 7.8% difference between the 2 methods, which is very close, In this case, I believe the prediction is accurate.

3.2.3 Back test

Please see chapter 5

3.2.4 Simple statistics test [89]

To see why 5% is an acceptable number, we can use the binomial test to check why it is acceptable

The formula :

$$\Pr(k; n, p) = \Pr(X = k) = \frac{n!}{(n-k)! \times k!} p^k (1-P)^{n-k}$$

So using the result in the back testing , we have 4% error rate in 100 data mean k = 4 , n = 100 and p = 0.05 .

$$\Pr(8; 200, 0.05) = \Pr(X = 8) = \frac{200!}{(200-8)! \times 8!} 0.05^8 (1-0.05)^{200-8}$$

$$\Pr(8; 200, 0.05) = 0.3270245$$

Since the p-value is 3 , which mean it is acceptable .

[8] Wikipedia Contributor (2022). Binomial test. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Binomial_test.
[9] PyShark (2021). Binomial Distribution and Binomial Test in Python. [online] PyShark. Available at: https://pyshark.com/binomial-distribution-and-binomial-test-in-python/.

3.3 Documentation

3.3.1 Stock_data package

1. Get_the_single_stock_historical_data_in_the_given_time (stock_ticket, period)
   Return the single stock historical data in the given time
   Parameters: stock_ticket : str
   The stock ticket that user want to get the historical data

   period: int
   how many day data the user want

   Returns: pandas. Dataframe
   the historical data in the given time

2. Get_the_stock_portfolio_historical_data_in_the_given_time (stock_list, period)
   Get the portfolio closing historical data in the given time
   Parameters: stock_ticket : str
   The list of stock ticket that user want to get the historical data

   period: int
   how many date data the user want

   Returns: pandas. dataframe
   The historical data in the given time

3. Get_the_time_frame(how_many_day)
   Return the date from today to the day given

   Parameters: how_many_day: int
   how many day user want to get

   Returns: date
   return the date from today to the previous given days

4. create_stock_object(stock_list)
   Create a stock Object in the yfinance package
   Parameters: stock_list: list
   A list has all the stock ticket user want to get the historical data

   Returns: Tickers
   The tickers object from yfinance

5. list_to_string_with_space(list)
   transfer a list to a string with space
   Parameters : list

   Returns : String
   concatenate the element in the string

3.3.2 Calculating_VaR package

data_initialise class

1. Calculating_daily_portfolio_Returns()

Calculated the daily returns for the given stock and closing price

Returns: pandas. Dataframe
a dataframe that have the daily returns for given stock and closing price while creating the object

2. add_Portfolio_columns_to_df(Stock_historical_data_df_with_returns)
Using the given weights and historical returns data to create a new column which call portfolio column
Parameters: Stock_historical_data_df_with_returns
historical returns dataframe

Returns : pandas.dataframe
A dataframe that added a new column call portfolio into the original dataframe

3. portfolioPerformance(weights, meanReturns, covMatrix, Time)
Calculating the portfolio Performance
Parameters : weights: int
the weights for the portfolio

meanReturns : float
the mean for the returns dataframe

covMatrix :pandas.dataframe
this is the Covariance matrix for the stock in the list

Time: int
time horizon

Returns : tuple (returns ,std)
returns :float
predicted returns

std : float
standard deviation

Historical_Simulation class

1. Calculating_VaR_by_Historical_Simulation(Stock_historical_data_df_with_Portfolio_returns,confidence_level)
Read in a pandas series of returns then output the percentile of the distribution at the given confidence level

Parameters: Stock_historical_data_df_with_returns : pandas.Series
This is a pandas Series that have the Stock returns

confidence_level : int
This is the confidence level for predicting the VaR

Returns: float
percentile of the distribution at the given confidence level

2. Calculating_CVaR_by_Historical_Simulation(Stock_historical_data_df_with_Portfolio_returns,confidence_level)
Read in a pandas series of returns then output the percentile of the distribution at the given confidence level

Parameters: Stock_historical_data_df_with_returns: pandas.Series
This is a pandas Series that have the Stock returns

confidence_level: int
This is the confidence level for predicting the CVaR

Returns: float
percentile of the distribution at the given confidence level

3. quantile_to_VaR(quantile, Time,InitialInvestment)
Return the VaR from given quantile
Parameters: quantile : float
The given quantile for VaR or CVaR

Time : int
Time horizon for the calculation

InitialInvestment: float
Initial Investment for your portfolio or stock

Returns : float
The VaR for given portfolio or stock ,confidence level ,Time ,Initial Investment

parametric_method class

1. Calculating_VaR_by_parametric_method(Stock_historical_data_df_with_returns, confidence_level, Time, InitialInvestment)
Read in a pandas series of returns then Calculated the VaR at the given confidence level
Parameters: Stock_historical_data_df_with_returns : pandas.dataframe
This is a pandas dataframe that have the Stock returns

confidence_level: int
This is the confidence level for predicting the CVaR

Time : int
Time horizon for the calculation

InitialInvestment: float
Initial Investment for your portfolio or stock

Returns : float
The VaR for given portfolio or stock ,confidence level ,Time ,Initial Investment
2. Calculating_VaR_by_parametric_method_portfolio(Stock_historical_data_df_with_returns , confidence_level, Time, InitialInvestment)
Read in a pandas series of returns then Calculated the VaR at the given confidence level

Parameters: Stock_historical_data_df_with_returns : pandas.dataframe
This is a pandas dataframe that have the Stock returns

confidence_level: int
This is the confidence level for predicting the CVaR

Time : int
Time horizon for the calculation

InitialInvestment: float
Initial Investment for your portfolio or stock

Returns : float
The VaR for given portfolio or stock, confidence level ,Time ,Initial Investment

3.

3.4 UML

Chapter 4: **Proof-of-concepts Development**

To Implement the concepts of VaR, I made a python program for calculating the VaR for real time data. During the development, I used TDD method as the main develop methodology.

4.1 Data initialise

To Calculate VaR in a real time situation, the First step we need do the data initialise first , As a result I have created a package mainly for this purpose (Get_the_stock_data.py) . In this package , it has a function Get_the_stock_portfolio_historical_data_in_the_given_time , which accepts the stock list and the period of time as input.

```python
def Get_the_stock_portfolio_historical_data_in_the_given_time(stock_list:list[str],period:int) -> Type[DataFrame] | Any:
    """
    Get the portfolio closing historical data in the given time

    Parameters
    ----------
    stock_ticket : list[str]
        The list of stock ticket that user want to get the historical data
    period : int
        how many day data the user want

    Returns
    -------
    pd.dataframe
        the historical data in the given time
    """

1   portfolio_stock_ticket = create_stock_object(stock_list)

    # single_closing_price_df = portfolio_stock_ticket.tickers[str(stock_list[0])].history(period=f'{period}d')['Close']
    # print(single_closing_price_df.head())

    portfolio_closing_price_df = pd.DataFrame
2   for ticket_symbol in stock_list:

        single_closing_price_df = portfolio_stock_ticket.tickers[ticket_symbol].history(period=f'{period}d')[['Close']]


3       if ticket_symbol == stock_list[0]:
            # print('1st')
            portfolio_closing_price_df = single_closing_price_df.copy()
            portfolio_closing_price_df = portfolio_closing_price_df.rename({'Close' : ticket_symbol},axis='columns')

        else:
            # print(ticket_symbol)
            portfolio_closing_price_df = pd.concat([portfolio_closing_price_df,single_closing_price_df],axis=1)
            portfolio_closing_price_df = portfolio_closing_price_df.rename({'Close' : ticket_symbol},axis='columns')


    # print(portfolio_closing_price_df)

4   return portfolio_closing_price_df
```

Figure 4 Get_the_stock_portfolio_historical_data_in_the_given_time function

In step 1, I have created the portfolio object by using the create_stock_object function, see appendix 1, then we will have a yfinance ticket.

In step 2, I only extract the closing price from the portfolio object, where I used a for loop to go throw all the given stock ticket

In step 3, I created a new pandas dataframe to store all the closing prices with stock tickets as their column name, see the example in Table 4

At step 4, return the pandas dataframe

To see how it works, here is the driver code for single stock and portfolio

```
# * for single stock
US_STOCK_LIST = ["TSLA"]
portfolio_stock_data = Get_the_stock_data.Get_the_stock_portfolio_historical_data_in_the_given_time(US_STOCK_LIST,period)

# * for portfolio
US_STOCK_LIST = ["TSM","GOOGL","TSLA","MSFT","AAPL"]
portfolio_stock_data = Get_the_stock_data.Get_the_stock_portfolio_historical_data_in_the_given_time(US_STOCK_LIST,period)
```

After we get the historical data, we need to prepare for the historical returns for each stock, so we need to use another function I create in Calculating_VaR package.

In the Calculating_VaR package, there are several classes, which are data_initialise, Historical_Simulation, and parametric_method class. Here we will focus on the data_initialise class first and will take about the others later in the report.

To use the function we need to create an object first, to create an object we need two parameters, first, the historical data we get in the previse part, and the weight of the stock in the portfolio, if the user does not put in the weights the default weights will be 1.

```
class data_initialise:
    """
    initialise the raw data
    """

    def __init__(self, Stock_historical_data_df,portfolio_weights=np.array([1])) -> None:
        """Constructor

        Parameters
        ----------
        Stock_historical_data_df : pd:dataframe
            A pandas dataframe which have closing price for specific stock , can be portfolio or single stock
        portfolio_weights : np.array
            A array that contain the weight for different stock , if not given , default is 1
        """

        self.Stock_historical_data_df = Stock_historical_data_df
        self.portfolio_weights = portfolio_weights
```

After we created a new object, we can use the function in the data_initialise class, which is the Calculating_daily_portfolio_Returns function, this function we don't need to provide any input, because we already gave it in the constructor.

```
def Calculating_daily_portfolio_Returns(self) -> Any:
    """
    Calculated the daily returns for the given stock and closing price

    Returns
    -------
    pd.dataframe
        a dataframe that have the daily returns for given stock and closing price

    """
    portfolio_closing_price_df = self.Stock_historical_data_df.copy()
    # print(portfolio_closing_price_df.columns)

    for ticket_symbol in portfolio_closing_price_df.columns:
        portfolio_closing_price_df[str(ticket_symbol)] = portfolio_closing_price_df[str(ticket_symbol)].pct_change()

    portfolio_closing_price_df = portfolio_closing_price_df.dropna()

    return (portfolio_closing_price_df)
```

To calculate the daily returns [10] I have to use the python pandas package function called pct_change (), then I replace the closing price column with the historical returns.

```
def add_Portfolio_columns_to_df(self,Stock_historical_data_df_with_returns:pd.DataFrame) -> DataFrame:
    """
    Using the given weights and historical returns data to create a new column which call portfolio column

    Parameters
    ----------
    Stock_historical_data_df_with_returns: pd.dataframe
        historical returns dataframe


    Returns
    -------
    dataframe
        added a new column call portfolio into the original dataframe

    """
    temp_df = Stock_historical_data_df_with_returns.copy()

    portfolio_weights = self.portfolio_weights

    # print(portfolio_weights)

    if len(portfolio_weights) == 1:
        # print("only one stock")
        temp_df.rename({f'{temp_df.columns[0]}' : 'Portfolio'},axis=1, inplace=True)

    else:
        temp_df['Portfolio'] = temp_df.dot(portfolio_weights)

    # print(temp_df.head())
    return temp_df
```

After we managed to get the historical returns if the user is providing a portfolio we need to calculate the combined returns, where we need to use the dot product to do it. As a result, we need to use the add_Portfolio_columns_to_df function, which accepts the historical returns dataframe that we create in the last part, if the dataframe only has one stock, we just rename the column, otherwise we create a new column , then returns the dataframe . After we finished the process , we now can use it to calculate the VaR

4.2 Historical Simulation method

In this part I will explain the code that calculating VaR and CVaR by using historical simulation method , for the historical simulation , I have created a class call Historical_Simulation in Calculating_VaR package , it also inherits the data_initialise class

4.2.1 VaR calculation [11]

For calculating VaR in the historical method, I used the function called Calculating_VaR_by_Historical_Simulation function, it accepts the historical portfolio returns and confidence level as input, then it will return the percentile of the distribution at the given confidence level here I used the Numpy package function percentile ()[12] to help me to get the quantile.

[10] Chapter 2.5
[11] Chapter 2.6.2
[12] numpy.org. (n.d.). numpy.percentile — NumPy v1.21 Manual. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.percentile.html.

```python
def Calculating_VaR_by_Historical_Simulation(self,Stock_historical_data_df_with_Portfolio_returns:pd.Series,
confidence_level:int) -> floating:
    """
    Read in a pandas series of returns then output the percentile of the distribution at the given confidence level

    Parameters
    ----------
    Stock_historical_data_df_with_returns : pd.Series
        This is a pandas Series that have the Stock returns

    confidence_level:int
        This is the confidence level for predicting the VaR

    Returns
    -------
    float:
        percentile of the distribution at the given confidence level

    """
    if isinstance(Stock_historical_data_df_with_Portfolio_returns,pd.Series):
        # np.sort(Stock_historical_data_df_with_returns.copy())
        return np.percentile(Stock_historical_data_df_with_Portfolio_returns ,confidence_level)
    else:
        raise TypeError("Expected returns to be dataframe ot series")
```

After we have the quantile we need to use the quantile_to_VaR function to get the VaR, the function accepts quantile , Time horizon, and Initial investment as an input

```python
def quantile_to_VaR(self,quantile:float,Time:int,InitialInvestment:float) -> Any:
    """
    Return the VaR from given quantile

    Parameters
    ----------
    quantile : float
        The quantile for VaR or CVaR
    Time : int
        Time horizon for the calculation

    InitialInvestment : float
        Initial Investment for your portfolio or stock

    Returns
    -------
    float
        The VaR for given portfolio or stock  ,confidence level ,Time ,InitialInvestment
    """

    hVaR = quantile*np.sqrt(Time)

    VaR = InitialInvestment*hVaR
    return VaR*-1
```

In this function it scales the quantile with the time horizon, then times the Initial investment, then it will return us the VaR.

4.2.2 CVaR calculation

For the Conditional Value at Risk , we basically do the same thing , but this time we need to take the value below the VaR ,then calculated the mean for it . so we use Calculating_VaR_by_Historical_Simulation function.

```python
def Calculating_CVaR_by_Historical_Simulation(self,Stock_historical_data_df_with_Portfolio_returns:pd.Series,confidence_level:int)
 -> float:
    """
    Read in a pandas series of returns then output the position for CVaR for the given confidence level

    Parameters
    ----------
    Stock_historical_data_df_with_returns : pd.Series
        This is a pandas Series that have the Stock returns

    confidence_level:int
        This is the confidence level for predicting the VaR

    Returns
    -------
    float :
        the quantile for CVaR for the given confidence level

    """
    if isinstance(Stock_historical_data_df_with_Portfolio_returns,pd.Series):
        belowVaR = Stock_historical_data_df_with_Portfolio_returns <= self.Calculating_VaR_by_Historical_Simulation
        (Stock_historical_data_df_with_Portfolio_returns,confidence_level)

        # print(Stock_historical_data_df_with_returns[belowVaR].mean())

        return Stock_historical_data_df_with_Portfolio_returns[belowVaR].mean()

    else:
        raise TypeError("Expected returns to be dataframe ot series")
```

Here we call the Calculating_VaR_by_Historical_Simulation function again to get the quantile we need , then we create a new variable below VaR to store the quantile below the VaR , then we use a pandas mean function [13] to get the mean .After we have the mean we can use the quantile_to_VaR to get the CVaR .

Driver code for Single stock :

[13] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.mean.html

```python
# * Testing for single stock

US_STOCK_LIST = ["TSLA"]
portfolio_stock_data = Get_the_stock_data.Get_the_stock_portfolio_historical_data_in_the_given_time(US_STOCK_LIST,period)

# print(portfolio_stock_data.head())
portfolio_weights = np.array([1])

TSLA_stock_object = Historical_Simulation(portfolio_stock_data)

TSLA_historical_return_df = (TSLA_stock_object.Calculating_daily_portfolio_Returns())
# print(TSLA_historical_return_df.head())

TSLA_df_with_weights = TSLA_stock_object.add_Portfolio_columns_to_df(TSLA_historical_return_df)
# print(TSLA_df_with_weights.head())

# 100 days Time Horizon
Time = 1
InitialInvestment = 10000

VaR = (TSLA_stock_object.Calculating_VaR_by_Historical_Simulation(TSLA_df_with_weights['Portfolio'],5))

CVaR = (TSLA_stock_object.Calculating_CVaR_by_Historical_Simulation(TSLA_df_with_weights['Portfolio'],5))

# print(f'VaR = {VaR}')

# hVaR = VaR*np.sqrt(Time)
hVaR = TSLA_stock_object.quantile_to_VaR(VaR,Time,InitialInvestment)

# hCVaR = CVaR*np.sqrt(Time)
hCVaR = TSLA_stock_object.quantile_to_VaR(CVaR,Time,InitialInvestment)

# mean_portfolio_historical_return_df = TSLA_historical_return_df.mean()
# covMatrix_portfolio_historical_return_df = TSLA_historical_return_df.cov()

# pRet, pStd = TSLA_stock_object.portfolioPerformance(portfolio_weights, mean_portfolio_historical_return_df,
covMatrix_portfolio_historical_return_df, Time)

print('Historical_Simulation_single_stock')
print(f'For single stock : {US_STOCK_LIST}')

# print('Expected Portfolio Return:      ', round(InitialInvestment*pRet,2))
print('Value at Risk 95th CI    :      ', round(hVaR,2))
print('Conditional VaR 95th CI  :      ', round(hCVaR,2))
```

Expected output :

```
Historical_Simulation_single_stock
For single stock : ['TSLA']
Value at Risk 95th CI    :       655.21
Conditional VaR 95th CI  :       809.28
```

Driver code for portfolio :

```
# * Testing for portfolio

US_STOCK_LIST = ["TSM","GOOGL","TSLA","MSFT","AAPL"]
portfolio_weights = np.array([0.2,0.15,0.15,0.3,0.2])

portfolio_stock_data = Get_the_stock_data.Get_the_stock_portfolio_historical_data_in_the_given_time(US_STOCK_LIST,period)

# print(portfolio_stock_data.head())

portfolio_stock_object = Historical_Simulation(Stock_historical_data_df = portfolio_stock_data,
portfolio_weights=portfolio_weights)

portfolio_historical_return_df = (portfolio_stock_object.Calculating_daily_portfolio_Returns())
# print(portfolio_historical_return_df.head())

portfolio_df_with_weights = portfolio_stock_object.add_Portfolio_columns_to_df(portfolio_historical_return_df)
# print(portfolio_df_with_weights.head())

# 100 days Time Horizon
Time = 1
InitialInvestment = 10000

VaR = (portfolio_stock_object.Calculating_VaR_by_Historical_Simulation(portfolio_df_with_weights['Portfolio'],5))

CVaR = (portfolio_stock_object.Calculating_CVaR_by_Historical_Simulation(portfolio_df_with_weights['Portfolio'],5))

# hVaR = VaR*np.sqrt(Time)
hVaR = portfolio_stock_object.quantile_to_VaR(VaR,Time,InitialInvestment)

# hCVaR = CVaR*np.sqrt(Time)
hCVaR = portfolio_stock_object.quantile_to_VaR(CVaR,Time,InitialInvestment)

# mean_portfolio_historical_return_df = portfolio_historical_return_df.mean()
# covMatrix_portfolio_historical_return_df = portfolio_historical_return_df.cov()

# pRet, pStd = portfolio_stock_object.portfolioPerformance(portfolio_weights, mean_portfolio_historical_return_df,
covMatrix_portfolio_historical_return_df, Time)

print('Historical_Simulation_portfolio')
print(f'For portfolio : {US_STOCK_LIST}')
# print('Expected Portfolio Return:      ', round(InitialInvestment*pRet,2))
print('Value at Risk 95th CI    :      ', round(hVaR,2))
print('Conditional VaR 95th CI  :      ', round(hCVaR,2))
```

Expected output :

```
Historical_Simulation_portfolio
For portfolio : ['TSM', 'GOOGL', 'TSLA', 'MSFT', 'AAPL']
Value at Risk 95th CI    :       326.71
Conditional VaR 95th CI  :       402.32
```

4.3 Model building method

In this part I will explain the code that calculates VaR by using model building method, for the model building method, I have created a class called parametric_method in the Calculating_VaR package , it also inherits the data_initialise class

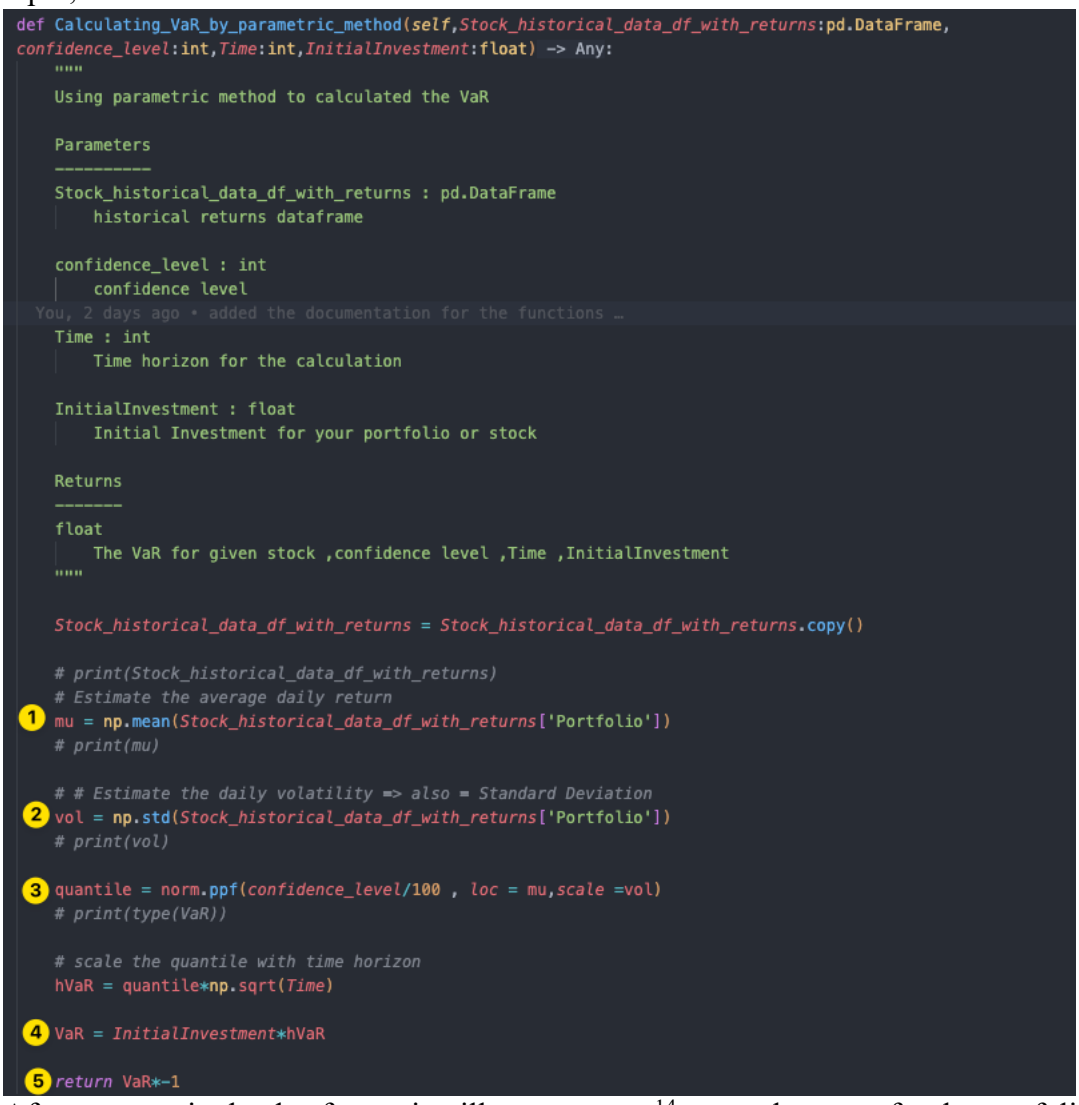
4.3.1 VaR calculation

I have created 2 methods to calculate the VaR by using model building calculation

For the first method, we use the dot product to calculate the total portfolio returns, similar to the historical simulation, then we use the portfolio columns to calculate the mean and standard deviation, then we pass it into the norm.pdf function to get the quantile

In this part, I have to use the function called Calculating VaR by parametric method function to calculate it. it accepts the historical portfolio returns and confidence level as

input, then it will return the VaR for us.



```python
def Calculating_VaR_by_parametric_method(self,Stock_historical_data_df_with_returns:pd.DataFrame,
confidence_level:int,Time:int,InitialInvestment:float) -> Any:
    """
    Using parametric method to calculated the VaR

    Parameters
    ----------
    Stock_historical_data_df_with_returns : pd.DataFrame
        historical returns dataframe

    confidence_level : int
        confidence level
  You, 2 days ago • added the documentation for the functions …
    Time : int
        Time horizon for the calculation

    InitialInvestment : float
        Initial Investment for your portfolio or stock

    Returns
    -------
    float
        The VaR for given stock ,confidence level ,Time ,InitialInvestment
    """

    Stock_historical_data_df_with_returns = Stock_historical_data_df_with_returns.copy()

    # print(Stock_historical_data_df_with_returns)
    # Estimate the average daily return
1   mu = np.mean(Stock_historical_data_df_with_returns['Portfolio'])
    # print(mu)

    # # Estimate the daily volatility => also = Standard Deviation
2   vol = np.std(Stock_historical_data_df_with_returns['Portfolio'])
    # print(vol)

3   quantile = norm.ppf(confidence_level/100 , loc = mu,scale =vol)
    # print(type(VaR))

    # scale the quantile with time horizon
    hVaR = quantile*np.sqrt(Time)

4   VaR = InitialInvestment*hVaR

5   return VaR*-1
```

After we pass in the dataframe, it will use np.mean[14] to get the mean for the portfolio returns , then we calculated the standard deviation by using np.std[15] function . Then we use the norm.ppf (Percent point function) function [16] to the quantile. After we got the quantile we can use the time horizon to scale the quantile, Finally we multiply it with the initial investment will return the VaR for us .

For the second method :
We assume the mean is zero and we used a different method to calculate the standard deviation. I used the formula in Ch 2.7.3 to calculate the standard deviation, other than that everything is the same with the first method.

[14] numpy.org. (n.d.). numpy.mean — NumPy v1.22 Manual. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.mean.html.
[15] numpy.org. (n.d.). numpy.std — NumPy v1.21 Manual. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.std.html.
[16] docs.scipy.org. (n.d.). scipy.stats.norm — SciPy v1.5.4 Reference Guide. [online] Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html.

```python
def Calculating_VaR_by_parametric_method_portfolio(self,Stock_historical_data_df_with_returns:pd.DataFrame,
confidence_level:int,Time:int,InitialInvestment:float) -> Any:
    """
    Using parametric method to calculated the VaR

    Parameters
    ----------
    Stock_historical_data_df_with_returns : pd.DataFrame
        historical returns dataframe

    confidence_level : int
        confidence level

    Time : int
        Time horizon for the calculation

    InitialInvestment : float
        Initial Investment for your portfolio or stock

    Returns
    -------
    float
        The VaR for given portfolio or stock  ,confidence level ,Time ,InitialInvestment
    """

    Stock_historical_data_df_with_returns = Stock_historical_data_df_with_returns.copy()
    portfolio_weights = self.portfolio_weights.copy()

    # cov matrix
    cov_matrix = Stock_historical_data_df_with_returns.copy().cov()
    # print(cov_matrix)

    amount_of_investing_for_each_stock = np.array([i*InitialInvestment for i in portfolio_weights])
    # print(amount_of_investing_for_each_stock)


    i = 0

    temp_list = []
1   for row,cols in cov_matrix.items():
        j = 0
        for col in cols:
            temp_list.append(col*amount_of_investing_for_each_stock[i] * amount_of_investing_for_each_stock[j])
            j+=1
        i+=1


2   temp_list = np.array(temp_list)

3   sigma = np.sqrt(sum((temp_list)))
          You, 1 second ago • Uncommitted changes
    # scale is sigma(standard deviation)
    VaR = norm.ppf(confidence_level/100, scale=sigma) * np.sqrt(Time)
    # print(VaR)

    return VaR*-1
```

Driver code for single stock using first method :

```
US_STOCK_LIST = ["TSLA"]
portfolio_stock_data = Get_the_stock_data.
Get_the_stock_portfolio_historical_data_in_the_given_time(US_STOCK_LIST,period)

TSLA_stock_object = parametric_method(portfolio_stock_data)

TSLA_historical_return_df = (TSLA_stock_object.Calculating_daily_portfolio_Returns())

TSLA_df_with_weights = TSLA_stock_object.add_Portfolio_columns_to_df(TSLA_historical_return_df)
# print(portfolio_df_with_weights.head())

Time = 1
InitialInvestment = 10000

VaR =(TSLA_stock_object.Calculating_VaR_by_parametric_method(TSLA_df_with_weights,5,Time,
InitialInvestment))

portfolio_weights = np.array([1])
# print(f'VaR = {VaR}')
# hVaR = VaR*np.sqrt(Time)

mean_portfolio_historical_return_df = TSLA_historical_return_df.mean()
# print(mean_portfolio_historical_return_df)
covMatrix_portfolio_historical_return_df = TSLA_historical_return_df.cov()


pRet, pStd = TSLA_stock_object.portfolioPerformance(portfolio_weights,
mean_portfolio_historical_return_df, covMatrix_portfolio_historical_return_df, Time)

print('parametric_method_single_stock')
print(f'For single stock : {US_STOCK_LIST}')
# print('Expected Portfolio Return:      ', round(InitialInvestment*pRet,2))
print('Value at Risk 95th CI    :      ', round(VaR,2))
```

Expected output :

```
parametric_method_single_stock
For single stock : ['TSLA']
Value at Risk 95th CI    :       617.66
```

Driver code for portfolio using first method:

```
US_STOCK_LIST = ["TSM","GOOGL","TSLA","MSFT","AAPL"]
portfolio_weights = np.array([0.2,0.15,0.15,0.3,0.2])

portfolio_stock_data = Get_the_stock_data.Get_the_stock_portfolio_historical_data_in_the_given_time
(US_STOCK_LIST,period)

portfolio_stock_object = parametric_method(portfolio_stock_data,portfolio_weights)

portfolio_historical_return_df = (portfolio_stock_object.Calculating_daily_portfolio_Returns())

portfolio_df_with_weights = portfolio_stock_object.add_Portfolio_columns_to_df
(portfolio_historical_return_df)

Time = 1
InitialInvestment = 10000

VaR =(portfolio_stock_object.Calculating_VaR_by_parametric_method(portfolio_df_with_weights,5,Time,
InitialInvestment))

print('first method for portfolio')      You, 1 second ago • Uncommitted changes
print(f'For single stock : {US_STOCK_LIST}')
# print('Expected Portfolio Return:      ', round(InitialInvestment*pRet,2))
print('Value at Risk 95th CI    :      ', round(VaR,2))
```

Expected output :

```
first method for portfolio
For single stock : ['TSM', 'GOOGL', 'TSLA', 'MSFT', 'AAPL']
Value at Risk 95th CI    :       299.85
```

Driver code for single stock using second method :

```
US_STOCK_LIST = ["TSLA"]
portfolio_weights = np.array([1])

portfolio_stock_data = Get_the_stock_data.Get_the_stock_portfolio_historical_data_in_the_given_time
(US_STOCK_LIST,period)

portfolio_stock_object = parametric_method(portfolio_stock_data,portfolio_weights)

portfolio_historical_return_df = (portfolio_stock_object.Calculating_daily_portfolio_Returns())


# portfolio_df_with_weights = portfolio_stock_object.add_Portfolio_columns_to_df
(portfolio_historical_return_df)

Time = 1
InitialInvestment = 10000

VaR =(portfolio_stock_object.Calculating_VaR_by_parametric_method_portfolio
(portfolio_historical_return_df,5,Time,InitialInvestment))

# print(VaR)

# print('using parametric_method portfolio function to test one stock')
print('Model Building method for single stock ')
# a = input('please enter the stock ticket \n')

print(f'For single stock : {US_STOCK_LIST}')
# print('Expected Portfolio Return:      ', round(InitialInvestment*pRet,2))
print('Value at Risk 95th CI    :      ', round(VaR,2))
```

Expected output :

```
Model building method for portfolio
For single stock : ['TSM', 'GOOGL', 'TSLA', 'MSFT', 'AAPL']
Value at Risk 95th CI    :       303.93
```

Driver code for portfolio using second method :

```
US_STOCK_LIST = ["TSM","GOOGL","TSLA","MSFT","AAPL"]
portfolio_weights = np.array([0.2,0.15,0.15,0.3,0.2])

portfolio_stock_data = Get_the_stock_data.
Get_the_stock_portfolio_historical_data_in_the_given_time(US_STOCK_LIST,period)

portfolio_stock_object = parametric_method(portfolio_stock_data,portfolio_weights)

portfolio_historical_return_df = (portfolio_stock_object.Calculating_daily_portfolio_Returns())

# portfolio_df_with_weights = portfolio_stock_object.add_Portfolio_columns_to_df
(portfolio_historical_return_df)

Time = 1
InitialInvestment = 10000

VaR =(portfolio_stock_object.Calculating_VaR_by_parametric_method_portfolio
(portfolio_historical_return_df,5,Time,InitialInvestment))

# print(VaR)

print('parametric_method_portfolio')
print(f'For single stock : {US_STOCK_LIST}')
# print('Expected Portfolio Return:      ', round(InitialInvestment*pRet,2))
print('Value at Risk 95th CI    :      ', round(VaR,2))
```

Expected output :

```
parametric_method_portfolio
For single stock : ['TSM', 'GOOGL', 'TSLA', 'MSFT', 'AAPL']
Value at Risk 95th CI    :       304.09
```

Chapter 5: **Back testing**

Back testing by the definition , is the general method for seeing how well a strategy or model would have done ex-post. Back testing assesses the viability of a trading strategy by discovering how it would play out using historical data. If back testing works, traders and analysts may have the confidence to employ it going forward. [17]
To implement in my code I have try to use the sliding window method to do the test, because I want to keep the training size and do it multiple times

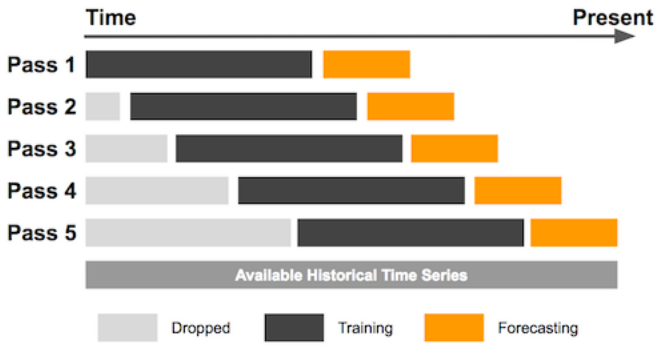


Figure from Forecasting at Uber: An Introduction [18]

So I used 201 days of data to do the test. First, used the first 100 days to do the calculation and used the rest 100 days to check the program working on not, then we calculate the error rate, what I do is if the Loss > VaR then add 1 into the list, if not put 0 in so at the end I can sum it up to see the error rate if the error rate ≈ 5 % mean is acceptable.

The formula:

$$err = \begin{cases} 1 \; if \; Loss > VaR \\ 0 \; if \; not \end{cases} \quad , \quad \frac{1}{k}\sum_{i=1}^{K} err_i \approx 5\%$$

The driver code for single stock in historical simulation with the following assumption VaR 95% , with $ 10000 initial Investment and using 101-day data to make the prediction:

[17] Chen, J. (2021). Backtesting Definition. [online] Investopedia. Available at: https://www.investopedia.com/terms/b/backtesting.asp.
[18] Bell, F. and Smyl, S. (2018). Forecasting at Uber: An Introduction. [online] Uber Blog. Available at: https://www.uber.com/en-GB/blog/forecasting-introduction/ [Accessed 6 Dec. 2022].

```python
confidence_level= 5
InitialInvestment = 10000

# * train set
train_set = TSLA_100day_data.iloc[:101]

# basic case :
def cal_VaR(train_set) -> floating:
    TSLA_stock_object = cal_VaR_by_Historical_Simulation(train_set)

    TSLA_historical_return_df = (TSLA_stock_object.Calculating_daily_portfolio_Returns())

    VaR = (TSLA_stock_object.Calculating_VaR_by_Historical_Simulation(TSLA_historical_return_df
    ['TSLA'],confidence_level))

    return(VaR)

VaR_prediction_list = []

for i in range(0,100):
    train_set = TSLA_100day_data.iloc[i:101+i]

    VaR_prediction_list.append(cal_VaR(train_set))


VaR_prediction_list = (np.array(VaR_prediction_list))
VaR_prediction_list = VaR_prediction_list*-1
VaR_prediction_list = (VaR_prediction_list*InitialInvestment)



# * test set
test_set = TSLA_100day_data.iloc[100:]

# calculating the returns
test_set_return_df = test_set.pct_change()
test_set_return_df = test_set_return_df.dropna()
test_set_return_df = test_set_return_df*-1

# print(test_set_return_df*InitialInvestment)
VaR_test_list = (test_set_return_df*InitialInvestment)["TSLA"]

error_list = []

for i in range(0,100):
    if VaR_test_list[i] >= VaR_prediction_list[i]:
        error_list.append(1)
    else:
        error_list.append(0)      You, 1 second ago • Uncommitted changes

error_rate = (sum(error_list)/len(error_list))
print(f'error rate = {round(error_rate*100,2)}%')
```

The output is 4 %, meaning is less than 5% , which is an acceptable error rate.

For the driver code for portfolio in historical simulation with the following assumption VaR 95% , with $ 10000 initial Investment and using 101 day data to make the prediction:

```python
confidence_level= 5
InitialInvestment = 10000
portfolio_weights = np.array([0.2,0.15,0.15,0.3,0.2])
US_STOCK_LIST = ["TSM","GOOGL","TSLA","MSFT","AAPL"]

# * predict set
train_set = portfolio_stock_data_100day.iloc[:101]
# print(train_set)
  You, yesterday • added the test case for using historical method t…
# basic case :
def cal_VaR(train_set) -> floating:
    portfolio_stock_object = cal_VaR_by_Historical_Simulation(train_set,portfolio_weights)

    portfolio_historical_return_df = (portfolio_stock_object.Calculating_daily_portfolio_Returns())

    portfolio_df_with_weights = portfolio_stock_object.add_Portfolio_columns_to_df
    (portfolio_historical_return_df)

    VaR = (portfolio_stock_object.Calculating_VaR_by_Historical_Simulation(portfolio_df_with_weights
    ['Portfolio'],confidence_level))

    # print(portfolio_df_with_weights)

    return(VaR)

VaR_prediction_list = []

for i in range(0,100):
    train_set = portfolio_stock_data_100day.iloc[i:101+i]

    VaR_prediction_list.append(cal_VaR(train_set))


VaR_prediction_list = (np.array(VaR_prediction_list))
VaR_prediction_list = VaR_prediction_list*-1
VaR_prediction_list = (VaR_prediction_list*InitialInvestment)

# print(VaR_prediction_list)
# print(VaR_prediction_list.shape)

# * test set
test_set = portfolio_stock_data_100day.iloc[100:]
# print(test_set.shape)
# print(test_set)

test_object =cal_VaR_by_Historical_Simulation(test_set,portfolio_weights)

# calculating the returns
test_set_return_df = test_set.pct_change()
test_set_return_df = test_set_return_df.dropna()
test_set_return_df = test_set_return_df*-1

test_set_return_df_df_with_weights = test_object.add_Portfolio_columns_to_df(test_set_return_df)
# print(test_set_return_df_df_with_weights)


VaR_test_list = (test_set_return_df_df_with_weights["Portfolio"]*InitialInvestment)

error_list = []

for i in range(0,100):
    if VaR_test_list[i] >= VaR_prediction_list[i]:
        error_list.append(1)
    else:
        error_list.append(0)

# print(error_list)

error_rate = (sum(error_list)/len(error_list))
print(f'error rate = {round(error_rate*100,5)}%')
```

The output is also 4% which is acceptable .

Appendix

1. create_stock_object function in Stock_data package

```
def create_stock_object(stock_list:list[str]) -> Tickers:
    """Create a stock Object in the yfinance package

    Parameters
    ----------
    stock_list : list[str]
        A list has all the stock ticket user want to get the historical data

    Returns
    -------
    Tickers
        The tickers object from yfinance
    """
    stock_list_str = list_to_string_with_space(stock_list)
    tickers = yf.Tickers(stock_list_str)

    return tickers
```

2. list_to_string_with_space function in Stock_data package

```
def list_to_string_with_space(list:list[str]) -> str:
    """
    transfer a list to a string with space

    Parameters
    ----------
    list : list[str]
        A list

    Returns
    ----------
    str :
        concatenate the element in the string
    """

    my_str = ' '.join(list)
    return(my_str)
```

Bibliography

1. Artzner, P., Pasteur, L., Strasbourg, F., Delbaen, T., Hochschule, Zürich, J.-M., Société Eber and Générale, P. (1996). *COHERENT MEASURES OF RISK*. [online] Available at: https://people.math.ethz.ch/~delbaen/ftp/preprints/CoherentMF.pdf.

2. Basle Committee on Banking Supervision (1996). *SUPERVISORY FRAMEWORK FOR THE USE OF 'BACKTESTING' IN CONJUNCTION WITH THE INTERNAL MODELS APPROACH TO MARKET RISK CAPITAL REQUIREMENTS*. [online] Available at: https://www.bis.org/publ/bcbs22.pdf.

3. Bell, F. and Smyl, S. (2018). *Forecasting at Uber: An Introduction*. [online] Uber Blog. Available at: https://www.uber.com/en-GB/blog/forecasting-introduction/ [Accessed 6 Dec. 2022].

4. Chen, J. (2019). *Conditional Value at Risk (CVaR)*. [online] Investopedia. Available at: https://www.investopedia.com/terms/c/conditional_value_at_risk.asp.

5. Chen, J. (2021). *Backtesting Definition*. [online] Investopedia. Available at: https://www.investopedia.com/terms/b/backtesting.asp.

6. docs.scipy.org. (n.d.). *scipy.stats.norm — SciPy v1.5.4 Reference Guide*. [online] Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html.

7. Dorota Huizinga and Kolawa, A. (2007). *Automated defect prevention : best practices in software management*. Hoboken, N.J.: Wiley-Interscience.

8. Howell, D.C. (2013). *Statistical methods for psychology*. Belmont, Ca: Wadsworth Cengage Learning.

9. Hull, J.C. (2022). *Options, Futures, and Other Derivatives*. 11th ed. Harlow Etc.: Pearson Educational Limited. Copyright.

10. numpy.org. (n.d.). *numpy.mean — NumPy v1.22 Manual*. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.mean.html.

11. numpy.org. (n.d.). *numpy.percentile — NumPy v1.21 Manual*. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.percentile.html.

12. numpy.org. (n.d.). *numpy.std — NumPy v1.21 Manual*. [online] Available at: https://numpy.org/doc/stable/reference/generated/numpy.std.html.

13. pandas.pydata.org. (n.d.). *pandas.DataFrame.mean — pandas 1.3.1 documentation*. [online] Available at: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.mean.html.

14. pandas.pydata.org. (n.d.). *pandas.DataFrame.pct_change — pandas 1.3.4 documentation*. [online] Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pct_change.html.

15. PyShark (2021). *Binomial Distribution and Binomial Test in Python*. [online] PyShark. Available at: https://pyshark.com/binomial-distribution-and-binomial-test-in-python/.

16. Wikipedia Contributor (2020). *Backtesting*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Backtesting.

17. Wikipedia Contributor (2022). *Binomial test*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Binomial_test.

18. Wikipedia Contributors (2018). *Test-driven development*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Test-driven_development.