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: Prof. Volodymyr (Volodya) Vovk

Department of Computer Science
Royal Holloway, University of London

March 31, 2023

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 Where other published and unpublished source materials have been used, these have been acknowledged.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: 21,537 words

Student Name: Shing Him Yip

Date of Submission: 31-03-2023

Signature:

Table of Contents

Abstract

The use of computers has become increasingly crucial in the modern finance environment. From sending financial reports to conducting meetings, technology plays a significant role in this field. This project focuses specifically on finance strategies, which are essential for the planned development of the Finance function. By building on insights from a business context, stakeholder expectations, and own performance and capabilities, a finance strategy aims to focus on opportunities that create value.

One critical aspect of investing is controlling risk, as noted by Warren Buffett. This project explores the use of Value at Risk (VaR) as a strategy for quantifying 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 can be calculated across various confidence levels. The project involves developing a program for computing VaR and testing its performance through back test.

The project's main objective is to explore the effectiveness of VaR as a risk management strategy and to develop a program that can accurately compute VaR across various confidence levels. This will be achieved by conducting back test on the program and comparing its results to actual historical data to ensure its accuracy. The program will also be evaluated based on its ability to provide insights into potential investment opportunities and the level of risk associated with each.

To sum up t the goal of this project is to contribute to the development of effective risk management methods in finance and to give insights into the possible benefits of employing VaR as a tool for financial risk management. This project will give significant insights into the effectiveness of VaR as a risk management approach and its possible uses in the financial industry by building and testing a programme that accurately computes VaR.

Project Specification

Your project specification goes here.

Objectives:

- Develop a proof-of-concept program for computing Value at Risk (VaR) for a portfolio consisting of 1 or 2 stocks using model-building and historical simulation methods.
- Analyse different ways of computing VaR and back test them to evaluate their statistical significance.
- Extend the program to allow derivatives (such as options) in the portfolio and use Monte Carlo simulation.
- Allow an arbitrary number of stocks in the portfolio using eigen- or Cholesky decomposition.
- Design and implement the final program with full object-oriented design and modern software engineering principles, including a graphical user interface.
- Conduct computational experiments with different data sets, methods, and parameters.

Background:
Estimating the riskiness of a portfolio is a complex and important problem in finance. Value at Risk (VaR) is a widely used tool for measuring financial risk. This project aims to develop a program for computing VaR for portfolios using model-building and historical simulation methods, and to extend the program to allow derivatives and Monte Carlo simulation. The final program will have a full object-oriented design and modern software engineering principles.

Deliverables:

- Proof-of-concept program for computing VaR for a portfolio consisting of 1 or 2 stocks using model-building and historical simulation methods.
- Analysis of different ways of computing VaR and their statistical significance.
- Extended program that allows derivatives and Monte Carlo simulation, and allows an arbitrary number of stocks in the portfolio using eigen- or Cholesky decomposition.
- Final program with full object-oriented design and modern software engineering principles, including a graphical user interface.
- Report detailing the theory behind the algorithms, implementation issues, software engineering process, and computational experiments with different data sets, methods, and parameters.

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

The aim of this project is to develop a program that simplifies the process of calculating Value at Risk (VaR) for investment portfolios. The program will allow investors and traders to easily evaluate the risk of their investment decisions, which is crucial for successful investment strategies.

The following goals was achieved in the first term of the project:

- Implement the historical simulation method for calculating VaR in single stocks using a proof-of-concept program.
- Implement the historical simulation method for calculating Conditional VaR (CVaR) in single stocks using the same program.
- Implement the historical simulation method for calculating VaR in portfolios using the same program.
- Implement the historical simulation method for calculating CVaR in portfolios using the same program.
- Implement the model building method for calculating VaR in single stocks using the same program.
- Back test the different methods for computing VaR and analyse the statistical significance of the results.

The following goals was achieved in the final term of the project:

- Extend the program to allow for derivatives such as options in the portfolio using Monte Carlo simulation and historical data.
- Allow an arbitrary number of stocks in the portfolio by using Cholesky decomposition.
- Implement a full object-oriented design with a complete implementation life cycle using modern software engineering principles.
- The program have a graphical user interface to provide an intuitive user experience.

1.3 Survey of Related Literature

This section provides an extensive overview of the key literature sources that have informed the development and understanding of the financial risk assessment tool. These sources have been instrumental in shaping the tool's theoretical foundation and practical implementation.

Hull, J. C. (2022). Options, Futures, and Other Derivatives (11th ed.). Pearson Educational Limited.

In this comprehensive book presents a detailed overview of the different financial instruments, such as options, futures, and other derivatives, as well as their pricing and risk management. The book covers essential concepts, including the estimation of Value at Risk (VaR) and other risk measures, which are directly relevant to the development of the financial risk assessment tool. This book makes a foundation for me to understand the underlying financial theories and quantitative methods used in the tool. The book also delves into various risk management strategies, hedging techniques, and the role of regulation in the financial industry. This book is an indispensable resource for building a solid theoretical understanding of the financial risk assessment tool's underlying principles.

Jorion, P. (2007). Value at risk: The new benchmark for managing financial risk. McGraw-Hill.

In this influential book, it provided an in-depth analysis of the Value at Risk (VaR) concept and its application in managing financial risk. This book presents various methodologies for estimating VaR, such as the historical simulation approach, the model building method , and the Monte Carlo simulation. The book also discusses back testing and model validation, which are crucial for ensuring the reliability of the financial risk assessment tool. This book help me to understand the practical implementation of VaR methodologies and their application in the risk assessment tool. Furthermore, the book explores the limitations and potential pitfalls of VaR, offering valuable insights into the challenges that practitioners may face when employing this risk measure in real-world scenarios.

Dowd, K., & Hutchinson, M. (2010). Measuring Market Risk (2nd ed.). Wiley.

Kevin Dowd and Martin Hutchinson's book focuses on the measurement and management of market risk. The authors provide a thorough examination of different risk measures, including Value at Risk (VaR), Conditional Value at Risk (CVaR), and stress testing. They also discuss various estimation methods, model validation techniques, and regulatory frameworks for market risk. This book offers valuable insights into the practical aspects of measuring and managing market risk, which can be applied in the development of the financial risk assessment tool. Additionally, the authors emphasize the importance of tail risk and the potential consequences of extreme market events, highlighting the need for robust risk management practices that account for such occurrences.

These three books serve as essential resources for understanding the theoretical and practical aspects of financial risk management, as well as the methodologies and best practices for developing a robust financial risk assessment tool. By examining these sources, we ensure that our tool is grounded in well-established concepts and methods in the field of finance and risk management. This comprehensive literature review helps to guarantee that the financial risk assessment tool adheres to industry standards and remains adaptable to the evolving needs of its users.

1.4 Milestones Summary (timeline)

1.4.1 Term 1

Week 1:
During this week, I began studying Value at Risk (VaR) and its underlying calculations. I focused on understanding the fundamental concepts and applications of VaR in financial risk management. I also identified and reviewed relevant literature and resources that would help me deepen my understanding of the topic and provide guidance for the project. I explored various risk assessment methods, their advantages and disadvantages, and how they have evolved over time.

Week 2:
In this week, I started planning the first term of the project and began learning how to use Python packages (yfinance, NumPy, pandas, and matplotlib) essential for data analysis in the financial domain. I researched the functionality and capabilities of these packages, studied their documentation, and explored sample use cases to gain hands-on experience. This week's focus was to develop a solid foundation for the data analysis and visualization components of the project.

Week 3:
This week, I started implementing simple calculations using Jupyter notebook and the Python packages learned in Week 2. I successfully initialized market data using the yfinance package and began experimenting with data manipulation and visualization using pandas and matplotlib. The goal was to become proficient in working with financial data, which would be crucial for the subsequent development of the financial risk assessment tool.

Week 4:
During this week, I successfully implemented a proof-of-concept program for computing Value at Risk (VaR) using the historical simulation method for both single stock and portfolio. I focused on understanding the underlying mechanics of the historical simulation method and its suitability for different types of financial assets. The proof-of-concept program allowed me to test my understanding of the method and served as a starting point for further development.

Week 5:
In this week, I implemented a proof-of-concept program for computing Conditional Value at Risk (CVaR) using the historical simulation method for both single stock and portfolio. I explored the differences between VaR and CVaR and learned about the importance of considering tail risk in financial risk management.

Week 6:
During this week, I enhanced the functions for calculating VaR and CVaR to accept different variables, allowing for more flexibility in the risk assessment process. I refined the calculation logic and ensured the functions' reliability and accuracy, which is crucial for the tool's effectiveness in risk management applications.

Week 7:
This week, I successfully implemented a proof-of-concept program for computing VaR using the model-building method for single stock. I compared the results with those

obtained from the historical simulation method and evaluated the strengths and weaknesses of the model-building approach. This week's focus was to expand the project's scope by exploring alternative methodologies for risk assessment.

Week 8:
During this week, I refactored the code for computing VaR using the historical simulation method and the data initialization process. I aimed to improve the code's readability, maintainability, and efficiency to ensure the project's long-term sustainability and ease of use for future developers and users.

Week 9:
This week, I worked on implementing a proof-of-concept program for computing VaR using the model-building method for portfolio. I extended the single stock model-building approach to handle portfolios and assessed the accuracy and efficiency of this method for multi-asset risk management.

Week 10:
During this week, I continued working on the model-building method for portfolio risk assessment. Simultaneously, I implemented back test functionality for the historical simulation method for single stock, which allowed me to validate the performance and reliability of the risk assessment tool.

Week 11:
In this week, I prepared for the midterm submission and presentation. I consolidated my progress, documented the work completed thus far, and prepared a comprehensive report and presentation for the midterm evaluation. I focused on showcasing the project's objectives, methodology, and results, highlighting the challenges and lessons learned during the development process. I also prepared to address any questions or concerns that might arise during the presentation.

Week 12:
In the final week of Term 1, I reviewed and reflected on my progress, evaluated the project's current state, and identified areas for improvement and further development. I also gathered and incorporated feedback from the midterm presentation to refine the project's direction and goals. I created a plan for the upcoming term, outlining the tasks and milestones to be achieved, including the implementation of additional risk assessment methods, improving the user interface, and enhancing the overall user experience.

1.4.2 Term 2

Week 1:
During this week, I upgraded the programs to API and began my research on Monte Carlo Simulation. I delved into Brownian motion, drift, and Cholesky decomposition to better understand the principles and methodology behind the simulation.

Week 2:
I implemented several functions related to Monte Carlo Simulation, including logarithmic_returns, compute_drift, and predict_daily_price. I also developed a function to combine the predicted prices into a dataframe and calculate the VaR using the historical simulation method.

Week 3:
I continued my work on Monte Carlo Simulation by implementing a variation using Cholesky decomposition. Additionally, I created test cases for the new functions and started looking into derivatives as another aspect of the project.

Week 4:
After meeting with my professor and receiving valuable feedback, I updated the project report and revised the formulas and styles within. I then added back tests for historical simulation and model building method s for single stocks and portfolios.

Week 5:
I continued working on back tests for the model building method s and resolved some importing bugs with Django. I also restructured the code to improve its organization and added driver code for Monte Carlo simulation methods for single stocks.

Week 6:
During this week, I implemented functions to check user input for stock lists and weights, and updated the app.py functions to accept user input. I upgraded the software to an API and refactored the code to remove dead code.

Week 7:
I began working on the frontend of the project using React, creating several pages and components such as navBar, SideBar, and a form component for user input. I also implemented a function to fetch the API with given parameters.

Week 8:
I focused on developing frontend components, including a graph component to display closing prices and updating the sideBar. I then created a new page in the project to display the calculated VaR in different methods and with option prices.

Week 9:
This week, I implemented various graphs to showcase closing prices and returns. I added a selection box for users to choose different methods for calculating VaR and started working on the VaR for option prices.

Week 10:
I continued working on the VaR for option prices, creating a class called cal_option_price for calculating option prices. I implemented the Black-Scholes formula and developed functions for cal_sigma, cal_d1_d2, and black_scholes.

Week 11:
I implemented the Option VaR class and tried to integrate historical simulation and Monte Carlo Simulation with the Option VaR. I then converted the functions to an API and started designing the interface for the OptionVaR.

Week 12:
In the final week of Term 2, I integrated the OptionVaR API with the frontend, developed the OptionSideBar component, and refactored the code to improve its overall structure and organization. This concluded the second term of the project with significant progress in both backend calculations and frontend user interface development.

Chapter 2: **Background Research**

2.1 Value at Risk[12]

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.

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.

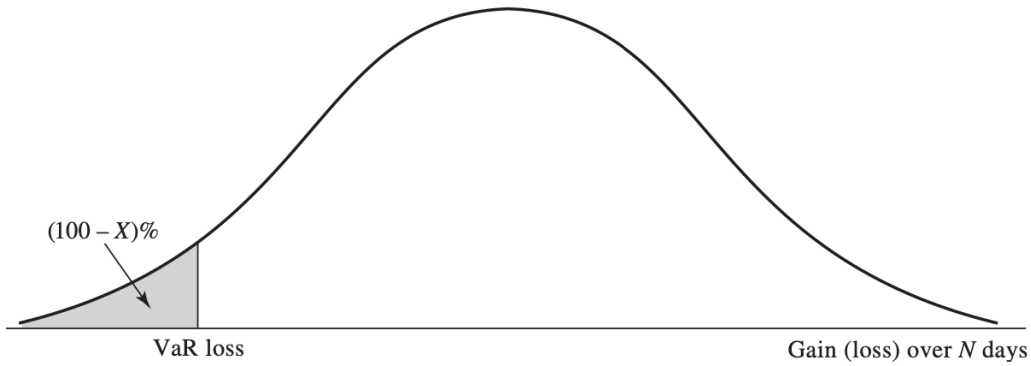


Figure 1 Calculation of VaR from the probability distribution of the change in the portfolio value; confidence level is X%. Gains in portfolio value are positive; losses are negative [3]

[1] Investopedia. (2022). Value at Risk (VaR). Retrieved from https://www.investopedia.com/terms/v/var.asp
[2] Jorion, P. (2007). Value at risk: The new benchmark for managing financial risk. McGraw-Hill.
[3] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 496

2.1.1 Mathematical definition[4]

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.

2.2 Conditional Value at Risk (CVaR) [5]

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.

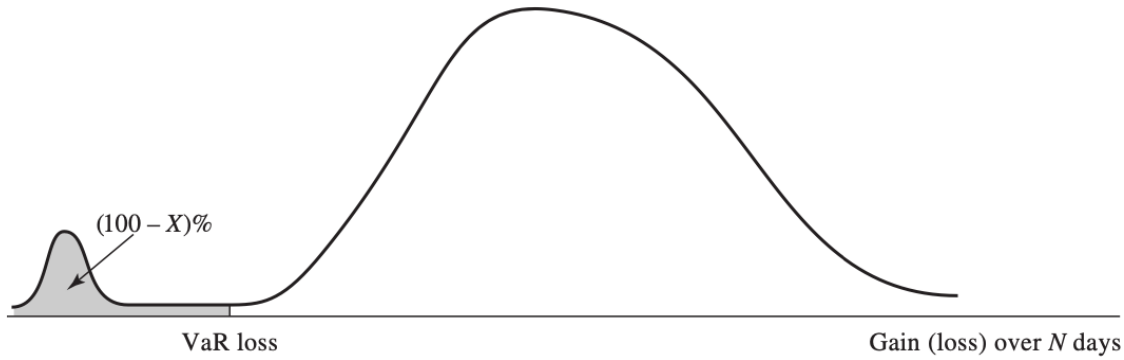


Figure 2 Alternative situation to Figure 1. VaR is the same, but the potential loss is larger.

[4] 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.

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

2.2.1 Mathematical definition[6]

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 points on the distribution where the analyst sets the VaR breakpoint
VaR = the agreed-upon VaR level

2.3 Time Horizon[7]

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.

2.4 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.

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

2.5 Historical Simulation [8]

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.)

[6] Chen, J. (2019). Conditional Value at Risk (CVaR). [online] Investopedia.
Available at: https://www.investopedia.com/terms/c/conditional_value_at_risk.asp.
[7] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 496
[8] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 497

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 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.

2.5.1 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. Therefore 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 day's closing prices 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 scenarios 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: sorted table from largest loss to profit.

Since we are interested in 99%VaR it means we need to that the take 1% of the total day, it means $500 \times 1\% = 5$ 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:

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

2.5.2 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: 501 day's closing prices for TSM, GOOGL, TSLA, MSFT, AAPL .

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) and calculated their returns.

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: 501 day's returns for TSM, GOOGL, TSLA, MSFT, AAPL .

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: sorted portfolio table from largest loss to profit.

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:

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

2.6 Model Building method [9]

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.

2.6.1 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 :

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

[9] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 501

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.

2.6.2 Model Building method – Single Asset [10]

Let said we have 10K in a share 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%.

From the standard normal cumulative distribution function, we know that $N^{-1}(0.01) = 2.326$. This means that there is 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:

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

2.6.3 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: returns table

[10] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p .502

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

$$cov(R_1, R_2) = \frac{1}{n} \sum_{i=1}^{n} r1_i \times r2_i$$

where:

$cov(R_1, R_2)$ is the covariance between the returns of stock 1 and stock 2
R_1 is the returns for stock 1
R_2 is the returns for stock 2
n is the total number of observations (i.e., time periods)
$r1_i$ is the return for stock 1 in period i
$r2_i$ is the return for stock 2 in period i

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

Table 7: covariances table

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

$$\sigma = \sqrt{\sum_{i,k=1}^{K} a_i a_k cov(R_i, R_k)}$$

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.

2.7 Monte Carlo simulation[11]

Monte Carlo simulation is a method of statistical modelling that uses random sampling to simulate possible outcomes of a system. It is commonly used in finance to estimate the potential losses in a portfolio of financial assets. Before I show case how to use it , let us discuss the theory behind.

2.7.1 Random Walk

In finance and economics, a random walk is a simple framework for modelling the evolution of asset prices and other economic variables over time. The path travelled by a variable that changes over time and the future value is unpredictable, such as the stock price, is referred to as its path. This means that future values of the variable cannot be predicted with certainty, as they move in a sequence of steps over time that are random and independent of the previous ones.

one example of a random walk is a coin toss. Assume we start with balance of zero, every time we toss a coin if it is head then add one, otherwise subtract one .Since the value of the balance will follow a random sequence of coin tosses, each toss being independent of the previous toss, the result can be considered a random walk.[12]

The concept is that the price of a stock can randomly rises or fall over time, with both possibilities being equally probable. Therefore, the stock price at any point can be seen as a random variable, the stock price's movement over time can be represented by a sequence of random steps or increments. So, it can be treated as random walk.

$$S_t = S_{t-1} + \epsilon_t$$

where:

S_t is the stock price at time t
S_{t-1} is the stock price at the previous time step
ϵ_t is a random variable representing the change in stock price from t-1 to t

2.7.2 Wiener Process[13]

The Wiener process, also known as Brownian motion, is a continuous-time stochastic process that is widely used in finance and economics to model random variables that evolve continuously over time. It is characterized by two properties: independence of increments and Gaussian distribution. The Wiener process is named after the mathematician Norbert Wiener, who first introduced it in 1923.

In mathematical terms, the Wiener process is defined as a continuous-time stochastic process $W(t), t \geq 0$with the following properties:

1. $W(0) = 0$
2. The increments $W(t2) - W(t1)$ are independent for any time t1 < t2

[11] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 511
[12] CS3930 Computational Finance class 4 PowerPoint
[13] Textbook 304 Wiener process

3. The increments $W(t2) - W(t1)$ are normally distributed with mean 0 and variance (t2 - t1)

In other words, the Wiener process is a random walk with infinitely small steps that are normally distributed. The variance of the increments increases with the length of the time interval, which means that the process becomes more volatile over time.

The following equation represents the Wiener process:

$$dW(t) = N(0, dt)$$

where $dW(t)$ is the increment of the process over an infinitely small time interval dt, and $N(0, dt)$ is a normally distributed random variable with mean 0 and variance dt.

2.7.3 Cholesky decomposition[14]

In mathematical optimization and statistics, Cholesky decomposition is a matrix factorization of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose, which is useful for efficient numerical solutions, such as Monte Carlo simulations, and for generating correlated random variables.

The Cholesky decomposition of a positive-definite matrix A can be written as:

$$A = LL^*$$

Where L is a lower triangular matrix with real and positive diagonal entries, and L* is its conjugate transpose. The decomposition is unique for positive-definite matrices, which are symmetric and have all positive eigenvalues.

One important application of Cholesky decomposition is in Monte Carlo simulations, which are widely used for financial risk management and option pricing. In Monte Carlo simulations, we generate a large number of random samples from a multivariate distribution with a specified covariance matrix, which is often positive-definite. The Cholesky decomposition allows us to efficiently transform independent, normally distributed random variables into correlated random variables with the specified covariance matrix.

2.7.4 Monte Carlo simulation[15]

Monte Carlo simulation is a technique used to generate numerous potential outcomes for a given system or process by using random sampling. It is often used in finance to simulate the behaviour of financial assets such as stocks, options, or portfolios. Monte Carlo simulation is a statistical method that can help estimate the range of possible outcomes and their associated probabilities. It can be used to calculate various financial metrics such as value at risk (VaR), expected return, and portfolio risk.

The Monte Carlo simulation is based on the principle of generating many random sample paths for a given financial asset's price. Each sample path is constructed by generating random variables for each of the inputs that drive the process. For example, for a stock

[14] Jorion, P. (2006). Value at Risk: The New Benchmark for Managing Financial Risk (3rd ed.). McGraw-Hill.
[15] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 469 , 511

price simulation, we would generate random variables for the expected return and volatility of the stock. Once the inputs have been determined, we simulate the stock price movement using the Wiener process, which is a continuous-time stochastic process that is also a type of Brownian motion.

Using Monte Carlo simulation, we can generate many possible stock price paths. Based on these paths, we can calculate the statistical properties of the resulting distribution of prices, such as the mean, standard deviation, and VaR. Monte Carlo simulation provides a powerful tool to estimate the range of possible outcomes for a given financial asset or portfolio.

In the following section, we will illustrate how Monte Carlo simulation can be applied to predict stock prices for a single stock. For the purpose of this example, we will use the stock of Apple Inc., which is commonly referred to by its ticker symbol 'AAPL'.

Step 1: Collect historical stock price data and calculate logarithmic returns for each stock.

Scenario number	Date	AAPL Closing	log return
1	2021-03-25 00:00:00-04:00	119.1556854	
2	2021-03-26 00:00:00-04:00	119.7683182	0.00514138
	⋮		
498	2023-03-16 00:00:00-04:00	155.8500061	0.01869404
499	2023-03-17 00:00:00-04:00	155	-0.005454
500	2023-03-20 00:00:00-04:00	157.3999939	0.01548383

Table 8: calculate the logarithmic returns

Step 2: Compute the drift (expected return) and standard deviation for each stock using the historical data.

$$drift = u - (0.5 \times var)$$

Where:
u is the mean return of the stock (typically calculated using historical data)
var is the variance of the stock's returns

$$drift = 0.000571 - (0.5 \times 0.000356)$$
$$drift = 0.000393$$

Step 3: Generate random variables for each stock using cumulative distribution function. This step assumes that the stocks are independent.

$$Z = norm.ppf(U)$$

Where:
U is a uniformly distributed random number between 0 and 1

Step 4: Calculate the predicted daily returns using the drift and standard deviation for each stock.

$$R = \mu + \sigma * Z$$

Where:

μ is the drift
σ is the standard deviation ($\sqrt{(var)}$)
Z is the random variable generated in step 3.

Step 5: Simulate the stock prices using the predicted daily returns and the most recent stock price.

Day \ Iterations	0	1	2	...	997	998	999
0	158.475006	158.475006	158.475006	...	158.475006	158.475006	158.475006
1	164.373126	157.324095	157.753675		153.364918	161.177674	154.531318
		⋮				⋮	
497	148.733376	181.161799	159.819651		352.932769	250.734484	188.732894
498	152.456903	181.778016	165.117405	...	350.069022	250.755293	185.136395
499	153.013279	184.806859	164.133817		348.059189	243.704491	182.083997
500	151.685277	185.541095	161.059877		352.448248	248.517901	181.811294

Table 9: table of simulation

If we want to predict stock prices for multiple stocks, i.e., a portfolio, there are two methods to do it.

Method 1: Monte Carlo Simulation with Independent Stocks

We assume that each stock in the portfolio has a separate random walk and generate simulations for each stock individually. We can then combine the results to get simulations for the overall portfolio. The following steps can be taken:

Step 1: Collect historical stock price data and calculate logarithmic returns for each stock.
Step 2: Compute the drift (expected return) and standard deviation for each stock using the historical data.
Step 3: Generate random variables for each stock using cumulative distribution function. This step assumes that the stocks are independent.
Step 4: Calculate the predicted daily returns using the drift and standard deviation for each stock.
Step 5: Simulate the stock prices using the predicted daily returns and the most recent stock price.

(The first 5 step is same with the single stock)

Step 6: Repeat Steps 1-5 for each stock in the portfolio separately, generating simulations for each stock individually.

Step 7: Concatenate the simulation results for each stock from Step 6 to obtain the simulation result for the portfolio. This can be done by adding the simulated stock prices for each stock on the corresponding date. At the end of this step, we will have 1000 tables (if we do 1000 iterations) with simulated prices for the entire portfolio.

	TSM	GOOGL	TSLA	MSFT	AAPL
0	94.6600037	105.560097	197.854996	278.975006	161.130005
1	97.0890978	107.313856	198.980869	282.964512	162.841016
			⋮		
497	134.529485	87.3191885	95.8811076	333.517002	206.498532
498	137.467336	88.9855642	97.4497122	337.452625	210.631171
499	135.883797	86.2051221	95.767473	329.624666	206.807902
500	132.612548	84.1184788	91.1817691	325.285487	202.660824

Table 10: combine 5 different stock simulation

Method 2: Monte Carlo Simulation with Cholesky Decomposition
We take into account the correlations between the stocks in the portfolio. We can do this by using the Cholesky decomposition of the covariance matrix to generate correlated random variables and then simulating the prices of each stock using these variables. The following steps can be taken:

The first 2 step is same, so let start at Step 3
Step 3: Calculate the covariance matrix for the logarithmic returns.

	TSM	GOOGL	TSLA	MSFT	AAPL
TSM	0.00048221	0.00025766	0.00039879	0.00022384	0.00024525
GOOGL	0.00025766	0.00043827	0.00035918	0.00031801	0.00028901
TSLA	0.00039879	0.00035918	0.00147227	0.00034294	0.0004027
MSFT	0.00022384	0.00031801	0.00034294	0.0003525	0.0002722
AAPL	0.00024525	0.00028901	0.0004027	0.0002722	0.00035614

Table 11: covariance matrix for the logarithmic returns.

Step 4: Perform Cholesky decomposition on the covariance matrix to obtain a lower triangular matrix.

0.02194790840	0	0	0	0
0.01172016079	0.01734395329	0	0	0
0.01814834081	0.00844612582	0.03273777441	0	0
0.01017112928	0.01144333951	0.00187587853	0.01067630013	0
0.01115481223	0.00911422601	0.00376106388	0.00440169712	0.01071734917

Table 12: lower triangular matrix

Step 5: Generate uncorrelated random variables and use the Cholesky decomposition matrix to transform them into correlated random variables.

-0.0171203	-0.0020785		-0.0231774	-0.0182764	-0.0194324	-0.0254976
-0.0055269	0.0015568		0.0042548	-0.0173976	-0.0150730	-0.0077189
-0.0109461	-0.0239878	⋯	-0.0481057	-0.0483844	-0.0454906	0.0010758
-0.0260129	0.0273971		-0.0163112	-0.0117386	-0.0195111	-0.0143122
-0.0118941	-0.0075111		0.0165358	-0.0019080	-0.0171641	-0.0024085

Table 13: table for correlated random variables.

Step 6: Calculate the predicted daily returns using the drift and correlated random variables.

1.0130765	0.9946762		0.9964177	0.9919025	1.0274476	0.9898305
1.0045734	0.9878057		0.9926966	1.0044542	0.9942437	0.9973309
0.9690590	1.0126855	. . .	0.9910841	0.9639545	1.0510070	0.9336895
0.9860603	0.9985201		0.9882376	1.0128240	0.9756909	0.9880241
0.9837859	1.0009511		1.0069531	0.9934275	1.0091926	0.9765988

Table 14: table of daily returns

Step 7: Simulate the stock prices using the predicted daily returns and the most recent stock price.

Step 8: Generating simulations for each stock by using the correlated random variables in Step 6. Like(Table 9)

Step 9: Concatenate the simulation results for each stock from Step 8 to obtain the simulation result for the portfolio. This can be done by adding the simulated stock prices for each stock on the corresponding date. At the end of this step, we will have 1000 tables (if we do 1000 iterations) with simulated prices for the entire portfolio.(Like Table 10)

Overall, Cholesky Decomposition may provide more accurate simulations for the portfolio since it considers the interdependence of the stocks.

2.8 Monte Carlo simulation for VaR[16]

Monte Carlo simulation for VaR involves using the simulated stock prices to calculate the potential losses in a portfolio at a given confidence level. This can be done using the same Historical Simulation approach discussed earlier.

Suppose we want to compute the 1-day 99% VaR for a portfolio having 5 stocks (AAPL, GOOGL, MSFT, TSLA, and TSM) with the following weights: 20%, 15%, 30%, 15%, and 20%. We can perform a Monte Carlo simulation to generate the stock price paths for each stock in the portfolio, as described in the previous section.

Next, we can use Historical Simulation to calculate the potential losses for the portfolio at the 99th percentile level. This involves calculating the portfolio value for each simulated stock price path and sorting the resulting portfolio returns from lowest to highest. The 99th percentile portfolio return represents the 1-day 99% VaR for the portfolio.

Finally, we can use the same method to determine the portfolio's 5-day 99% VaR by multiplying the 1-day VaR by the square root of 5. This provides an estimate of the potential losses that the portfolio could experience over a 5-day period at the 99% confidence level, based on the simulated stock price paths generated by the Monte Carlo simulation.

[16] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 511

2.9 Option[17]

An option represents a financial agreement involving two participants, in which the purchaser (holder) acquires the privilege, without being required, to either purchase or sell a particular asset at a pre-established cost (strike price) during a specified time period. The provider (writer) of the option must fulfil the obligation to buy or sell the underlying asset at the strike price if the purchaser decides to utilize the option. Furthermore, the contract's designated date is referred to as the expiration or maturity date, while the predetermined price is known as the exercise or strike price.

Since the option is a financial instruments, it can be traded on different kind of underlying assets, including stocks, bonds, currencies, cryptocurrencies or even commodities. Option letting investor to achieve various objectives, such as hedging risk, speculating on price movements, and generating income.

There are two type of option, American or European, and this distinction has nothing to do with their geographic location. American options can be exercised at any time up to the expiration date, while European options can only be exercised on the expiration date itself. Most options traded on exchanges are American, but European options are generally easier to analyse than American options.

2.9.1 Option Type

There are two main types of options: call options and put options. A call option gives the holder the right to buy the underlying asset at the strike price, while a put option gives the holder the right to sell the underlying asset at the strike price. Both call and put options have an expiration date, after which the option is no longer valid. I will explain in more details in call and put option. For make it simple we will talk about the European option only

Call Option

Call options give the holder the right to buy an underlying asset at a predetermined price (strike price) on the expiration date. Consider an investor who buys a European call option with a strike price of $100 to purchase 100 shares of a certain stock. If the stock price is above $100 on the expiration date, investor will exercise the option. For example, the stock price(market price) is $115. By exercising the option, the investor is able to buy 100 shares for $100 per share. If the shares are sold immediately, the investor makes a gain of $15 per share, or $1,500, ignoring transaction costs. When the initial cost of the option is taken into account, the net profit to the investor is $1,000. However, if the stock price on the expiration date is less than $100, the investor will choose not to exercise and will lose the initial investment.

[17] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 213

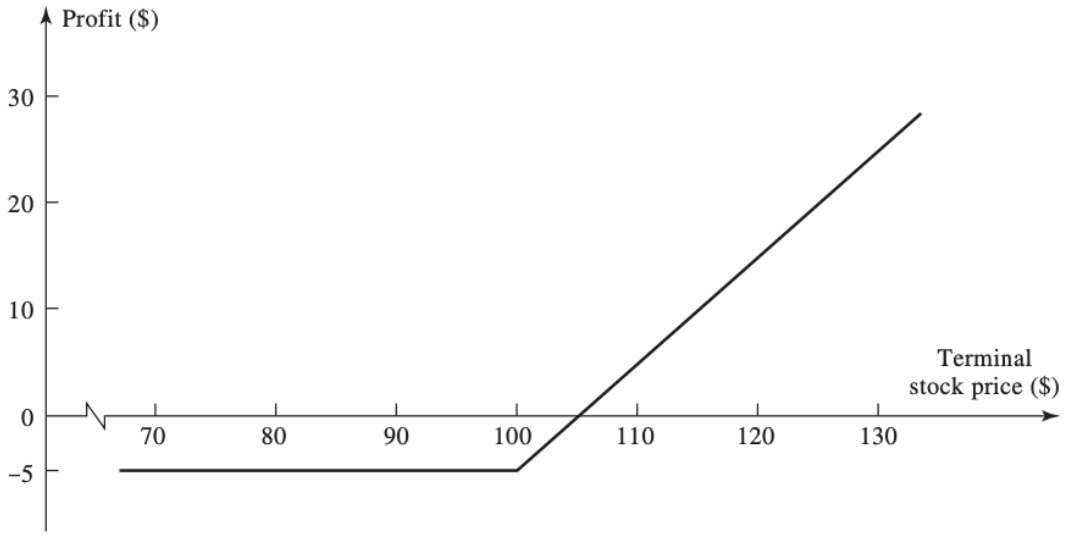


Figure 3: Profit from buying a European call option on one share of a stock. Where option price = $5, strike price = $100[18]

Put Option

Put options allow the holder to sell an underlying asset at a predetermined price (strike price) on the expiration date. Consider an investor who buys a European put option with a strike price of $70 to sell 100 shares of a certain stock. If the stock price falls below $70 by the expiration date, the investor can exercise the option and sell the 100 shares for $70 each, realizing a gain of $15 per share, or $1,500. When the initial cost of the option is taken into account, the investor's net profit is $800. However, if the final stock price is above $70, the put option will expire worthless, and the investor will lose the initial investment of $700.

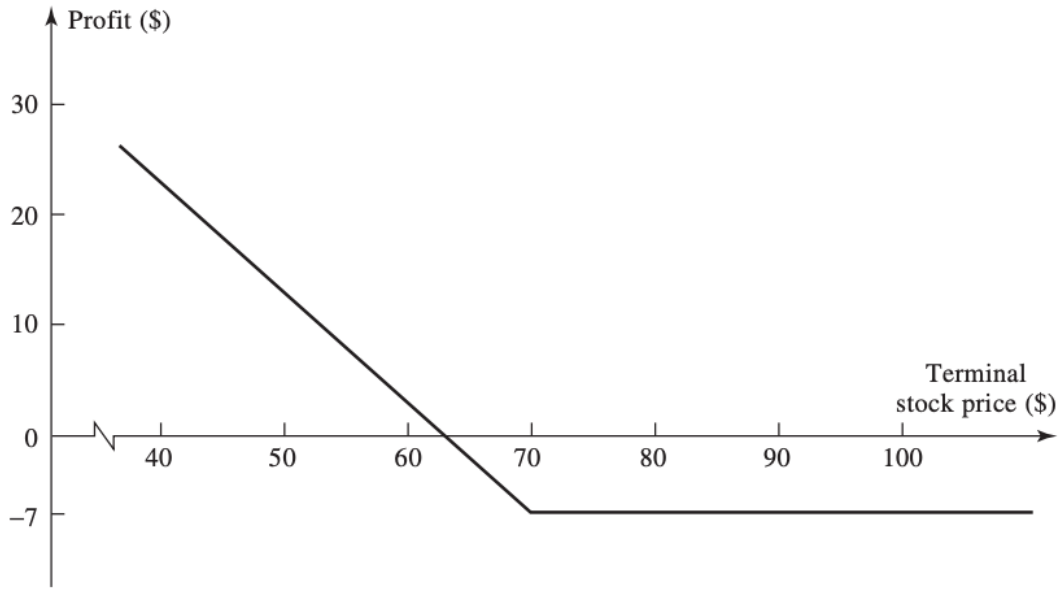


Figure 4: Profit from buying a European put option on one share of a stock. Option price is $7 , Strike Price is $70[19]

Early Exercise

[18] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 214
[19] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 215

European options can only be exercised at the expiration date. However American options, which can be exercised at any time before the expiration date. In the case of European options, the holder must wait until the expiration date to exercise the option.

2.9.2 Calculate Option Price

There are two main ways to calculate the price of an option: using Binomial Tree and using the Black-Scholes formula.

Binomial trees[20]

Binomial trees

One method for pricing options is to use a binomial tree model. The binomial tree is a mathematical tool used to calculate the probabilities of future stock prices and to price options based on these probabilities. The tree consists of a series of nodes, where each node represents a possible stock price at a future time. Starting from the current stock price, the tree branches out to two possible future prices: one up and one down. The branching continues until the final nodes are reached, which represent the possible stock prices at the expiration date of the option.

To compute the option price using a binomial tree, we must first compute the probability of the stock price moving up or down at each node. The probability are determined by the volatility of the stock price and the period before the option expires. Therefore, using probabilities, we can compute the predicted value of the choice at each node.

To calculate the option price using a binomial tree model, the expected value at each node is computed as the discounted value of the future payoffs of the option. For a call option, the payoff at each node is the maximum difference between the stock price and the strike price, or zero. Mathematically, the call option payoff at a node is:

$$Payoff_{call} = \max(S_t - K, 0)$$

where S_t is the stock price at a given node and K is the strike price of the option. For a put option, the payoff at each node is the maximum of the difference between the strike price and the stock price, or zero. Mathematically, the put option payoff at a node is:

$$Payoff_{put} = \max(K - S_t, 0)$$

The expected value is then calculated as the weighted average of the payoffs, using the probabilities of the stock price moving up or down at each node.

To determine the option price at the current time using the binomial tree model, we work backwards through the tree after calculating the expected values at each node. The expected value at each node is then discounted using the risk-free interest rate and the time to expiration of the option. This allows us to determine the option price at the current time based on the expected values at the expiration of the option.

The final result is the option price at the current time, which represents the fair value of the option based on the assumptions used to construct the binomial tree. Deviations from these assumptions can affect the accuracy of the calculated option price.

[20] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.p. 274

The mathematical equation for pricing a European call option using a binomial tree is as follows:[21][22]

$$C_0 = \frac{1}{(1+\tau)^n}\sum_{i=0}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left(S_0u^id^{n-i}\right) - K$$

$$P_0 = \frac{1}{(1+r)^n}\sum_{i=0}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left(K - S_0u^id^{n-i}\right)$$

where:

C_0 is the current price of the call option
P_0 is the current price of the put option
S_0 is the current stock price
K is the strike price
r is the risk-free interest rate
n is the number of periods in the binomial tree
p is the probability of the stock price moving up at each node
u is the factor by which the stock price moves up at each node
d is the factor by which the stock price moves down at each node
$\binom{n}{i}$ is the binomial coefficient, which represents the number of ways to choose i up movements out of n total movements.

Black-Scholes Formula[23]

The Black-Scholes model is a widely used mathematical model for pricing options contracts. The model takes several factors to calculate the option price, such as the current stock price (S_0), the strike price(K), the time to maturity(T), the risk-free interest rate(r), and the volatility of the underlying asset (σ).

The model considers several factors that affect the value of an option, including the current stock price, the strike price, the time to expiration, the risk-free interest rate, and the volatility of the underlying asset.

The Black-Scholes model assumes that the price of the underlying asset follows a lognormal distribution, which means that it has a long-term average growth rate and a certain degree of randomness. The model also assumes that there are no dividends paid out during the life of the option.

The formula for the Black-Scholes model is given as:[24]

$$C(S_t, t) = S_tN(d_1) - Ke^{-r(T-t)}N(d_2)$$
$$P(S_t, t) = Ke^{-r(T-t)}N(-d_2) - S_tN(-d_1)$$

[21] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited. P.321
[22] McDonald, R. L. (2014). Derivatives Markets (3rd ed.). Pearson Education Limited.
[23] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited. P.321
[24] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited. P.321

Where

$C(S_t, t)$ is the price of a call option at time t
$P(S_t, t)$ is the price of a call option at time t
S_t is the current stock price
K is the strike price
r is the risk-free interest rate
$T - t$ is the time to expiration
$N(d)$ is the cumulative standard normal distribution.
The value of $N(d)$ can be found in standard statistical tables or calculated using a computer program.

The values of d_1 and d_2 are calculated as follows:

$$d_1 = \frac{\ln(S_t/K) + (r + \sigma^2/2)(T - t)}{\sigma\sqrt{T - t}}$$

$$d_2 = d_1 - \sigma\sqrt{T - t}$$

where σ is the volatility of the underlying asset.

here's an example using the Black-Scholes formula to price a call option:

Suppose we want to price a European call option on a stock with a current price of $100. The option has a strike price of $120, an expiration date in 6 months, and a volatility of 20%. The risk-free interest rate is 5%.

First, we calculate the value of d1 and d2 using the Black-Scholes formula:

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T)}{\sigma\sqrt{T}} = \frac{\ln\left(\frac{100}{120}\right) + \left(0.05 + \frac{0.2^2}{2}\right)\left(\frac{6}{12}\right)}{0.2\sqrt{\frac{6}{12}}} \approx -1.04172$$

$$d_2 = d_1 - \sigma\sqrt{T} = -1.04172 - 0.2\sqrt{\frac{3}{12}} \approx -1.18314$$

Next, we calculate the value of N(d1) and N(d2) using a standard normal distribution table or calculator. For our example, we have:

$$N(d_1) \approx 0.14877$$

$$N(d_2) \approx 0.11838$$

Finally, we use the Black-Scholes formula to calculate the price of the call option:

$$C(S, t) = S_t N(d_1) - Ke^{-r(T-t)}N(d_2) = 100(0.14877) - 120e^{-0.05\cdot\frac{6}{12}}(0.11838) \approx 1.02262$$

So the price of the call option is approximately $1.02.

If we use the Black-Scholes formula to calculate the price of the put option:

$$P(S,t) = Ke^{-r(T-t)}N(-d_2) - S_tN(-d_1)$$
$$= 120e^{-0.05\left(\frac{6}{12}\right)}N(-0.11838) - 100N(-0.14877) \approx 18.05980$$

So the price of the call option is approximately $18.06.

2.10 Option VaR[25]

There are several methods to calculate option VaR, including historical simulation and Monte Carlo simulation. Both methods involve simulating the future prices of the underlying asset, based on historical data, or assumed probability distributions, and using these simulations to calculate the potential losses of the option at a specified confidence level.

In both historical and Monte Carlo simulations, the Black-Scholes formula can be used to calculate the option price at each simulated price of the underlying asset, based on the specified option parameters such as the strike price, time to expiration, and volatility.

2.10.1 Historical Simulation to Calculate Option VaR

Historical simulation is a widely-used method for calculating the VaR of financial instruments, including options. The method involves using historical data to simulate the potential future price movements of the underlying asset, and then using these simulations to estimate the potential losses of the option at a specified confidence level.

To apply historical simulation to options, we first need to determine the historical daily returns of the underlying asset over a specified time horizon, such as a month or a year. These returns can be calculated using the formula at section 2.4 Historical Returns.

Once the historical returns are calculated, we randomly select a set of returns from the historical data, with replacement, and use these returns to calculate the future prices of the underlying asset. This process is repeated multiple times, generating a distribution of potential future prices of the asset.

To calculate the VaR of the option at a specified confidence level, we then calculate the potential losses of the option at each simulated price, using the option's payoff function. For example, for a put option with a strike price of 120, we would use the formula:

$$Payoff = \max(120 - S_T, 0)$$

where S_T is the simulated price of the underlying asset at the option's expiration date.

Once we have calculated the potential losses at each simulated price, we can use the distribution of potential losses to estimate the VaR at the specified confidence level.

[25] Hull, J. C. (2018). Options, Futures, and Other Derivatives (10th ed.). Pearson Education Limited.

For example, let's consider a European put option on a stock with a current price of $100. The option has a strike price of $120, an expiration date in 6 months, and a volatility of 20%. We can use historical simulation to estimate the VaR of the option at a 95% confidence level.

Assuming a time horizon of 6 months, we calculate the historical daily returns of the underlying asset over the past year and use these returns to simulate the future price movements of the asset. We randomly select a set of 120 daily returns, with replacement, and use these returns to generate 1,000 simulated prices of the underlying asset at the option's expiration date.

Using the simulated prices, we can calculate the potential losses of the option at each price, using the put option's payoff function. We then use the distribution of potential losses to estimate the VaR at a 95% confidence level.

We can see that historical simulation provides a straightforward and simple approach for calculating option VaR, but it may not capture extreme market events that have not occurred in the historical data. Therefore, historical simulation should be used in conjunction with other methods, such as Monte Carlo simulation, to provide a more comprehensive estimate of the option's risk.

2.10.2 Monte Carlo Simulation to Calculate Option VaR

Monte Carlo simulation is another widely used method for calculating the VaR of financial instruments, including options. Unlike historical simulation, Monte Carlo simulation does not rely on past data to simulate future price movements. Instead, it uses probability distributions and random number generators to simulate future price movements.

To apply Monte Carlo simulation to options, we first need to determine the probability distribution of the underlying asset's future prices. This can be done using historical data, market information, or other assumptions. One commonly used probability distribution for stock prices is the lognormal distribution, which assumes that the price changes are proportional to the current price.

Once the probability distribution is determined, we can use random number generators to simulate a large number of potential future prices of the underlying asset. For example, we can simulate 1,000 potential future prices of the underlying asset at the option's expiration date, based on the lognormal distribution.

To calculate the VaR of the option at a specified confidence level, we then calculate the potential losses of the option at each simulated price, using the option's payoff function. For example, for a call option with a strike price of 80, we would use the formula:

$$Payoff = \max(S_T - 80, 0)$$

where S_T is the simulated price of the underlying asset at the option's expiration date.

Once we have calculated the potential losses at each simulated price, we can use the distribution of potential losses to estimate the VaR at the specified confidence level.

For example, let's consider a European call option on a stock with a current price of $100. The option has a strike price of $80, an expiration date in 3 months, and a volatility of 25%. We can use Monte Carlo simulation to estimate the VaR of the option at a 95% confidence level.

Assuming a time horizon of 3 months, we can use the lognormal distribution to simulate 1,000 potential future prices of the underlying asset at the option's expiration date. Using these simulated prices, we can calculate the potential losses of the option at each price, using the call option's payoff function. We then use the distribution of potential losses to estimate the VaR at a 95% confidence level.

Monte Carlo simulation is a more flexible method than historical simulation, as it allows for the use of non-normal distributions and the incorporation of more complex risk factors. However, it is computationally more intensive and requires careful selection of the underlying asset's probability distribution and correlation with other risk factors.

Chapter 3: **Software engineering**

3.1 Methodology

The methodology section explains the approach used for software development. There are different ways to develop software, and in this report, we will use Test-driven development (TDD) as the primary method.

3.1.1 Test-driven development (TDD)[26]

Test-driven development (TDD) is a software development process that relies on software requirements being converted into test cases before the software is fully developed. It involves tracking all software development by repeatedly testing the software against all test cases. This approach is different from traditional methods, where software is developed first and then test cases are created later.

The primary idea behind TDD is to write tests first and let them drive the development of the code. In other words, before writing code, the developer should consider what the code will do and then write a test that uses methods that haven't even been written yet. This approach helps to ensure that the code meets the requirements specified in the test cases.

3.1.2 TDD stages and cycle [27]

The TDD process involves a cycle of stages that repeat until the software meets the desired requirements. The stages are as follows:

1. Write a test: In this stage, the developer writes a test case that specifies the behaviour of the code.
2. Compile the test: In this stage, the developer compiles the test case to ensure that it is syntactically correct. It may not compile because the code has not been written yet.
3. Write just enough code to get the test to compile: In this stage, the developer writes the minimum amount of code required to make the test case compile.
4. Run the test and see it fail: In this stage, the developer runs the test and expects it to fail because the code has not yet been implemented.
5. Write just enough code to pass the test: In this stage, the developer writes the minimum amount of code required to pass the test.
6. Run all the tests a few times to enjoy seeing your code pass: In this stage, the developer runs all the tests to ensure that the code meets all the requirements.
7. Refactor: In this stage, the developer improves the code's design and structure without changing its behaviour.
8. Repeat from step 1: Once the code meets the requirements, the cycle starts again for the next feature or requirement.

[26] Beck, K. (2003). Test-driven development: By example. Addison-Wesley Professional.
[27] CS2800 Topic 2 power point

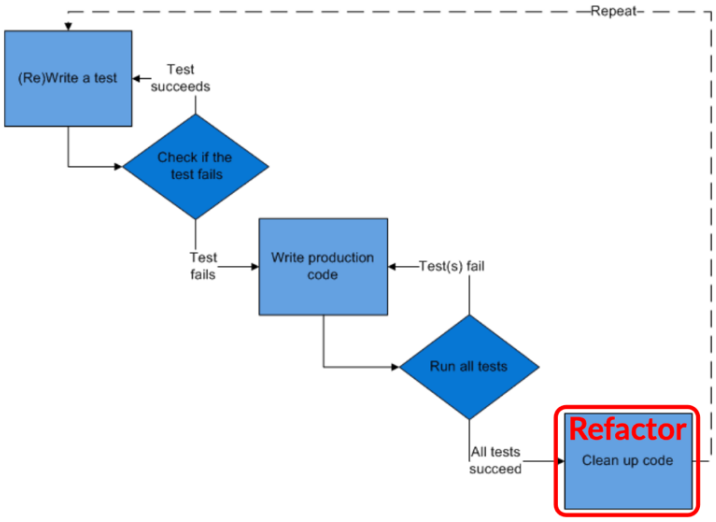


Figure 5: TDD cycle

3.2 Testing

Testing is an essential aspect of software development, and in this project, several types of testing were conducted to ensure the accuracy and reliability of the software. The testing methods used in this project include unit testing, high-level testing, back testing, and statistical testing.

3.2.1 Unit test

Unit testing is a software testing method used to determine if individual units of code are fit for use. It involves testing sets of one or more computer program modules, together with associated control data, usage procedures, and operating procedures, to ensure they meet the required specifications. Unit testing allows developers to validate that each unit of the software performs as expected, which helps to build a robust and reliable system.

In this project, the Test-driven development (TDD) methodology was used to develop the software. With TDD, every function will have at least one test case associated with it. This approach ensures that the code is developed incrementally and tested continuously throughout the development process, which helps to catch errors early and improve the overall quality of the software. The TDD methodology emphasizes writing tests before implementing the actual code, ensuring that the software meets the desired specifications and requirements. [28]

A total of seven-unit test files were created to test different aspects of the software:

1. test_cal_option.py
2. test_Historical_Simulation.py
3. test_option_VaR.py
4. test_stock_data.py
5. test_data_initialise.py

[28] Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 75

6. test_Monte_Carlo_simulation.py
7. test_parametric_method.py

Each of these test files focuses on a specific part of the software, ensuring that all individual components are thoroughly tested. The test files contain multiple test functions that examine the correctness, reliability, and performance of the code. For instance, the test functions may check if the output of a function is as expected or if the function can handle various input types and edge cases. They also verify that the functions can handle exceptions and errors gracefully.

The test files were designed to cover a wide range of scenarios, including testing single stocks, portfolios, and different confidence levels. This comprehensive testing ensures that the software can handle various use cases and provides accurate results under diverse conditions. By following the TDD methodology, the development process was streamlined, and potential issues were identified and resolved early on.

Some examples of the test functions used in this project include:

test_list_to_string_with_space:
This test checks do the list_to_string_with_space function converts a list of strings into a single string separated by spaces.

test_Get_the_time_frame:
This test verifies if the Get_the_time_frame function returns the correct date object when given a specific number of days.

test_create_stock_object:
This test confirms if the create_stock_object function creates a valid stock object when provided with a list of stock symbols.

test_check_stock_exists:
This test evaluates if the check_stock_exists function can correctly identify valid and invalid stock symbols.

test_check_weight:
This test assesses whether the check_weight function can ensure the sum of portfolio weights equals one and that the lengths of the stock symbol list and weight list are the same.

The use of unit testing in this project helped to maintain high-quality code and allowed for easier identification and resolution of potential issues. By writing tests before implementing the actual code, the development process was driven by the desired specifications and requirements, resulting in a more robust and reliable software.

Due to the comprehensive unit testing conducted, the software has exhibited a remarkable degree of accuracy and dependability, effectively reducing the likelihood of unforeseen issues or errors during its operation. The adoption of the test-driven development methodology not only enhanced the software's overall quality but also simplified future

maintenance and modifications. This is because the test suite acts as a safeguard for identifying regressions when updates are implemented.[29]

3.2.2 High level test

In order to test the accuracy of our VaR estimates, we conducted high level testing using two different methods: historical simulation and model building. We also conducted two Monte Carlo simulations to test the effectiveness of our approach. For each method, we set an initial investment of $10,000, a 95% VaR, and a 1-time horizon with 501 days of data.

The results of our testing are shown in the table below:

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	$ 307.35	$ 314.17
Model Build 1	$ 302.66	$ 300.78
Model Build 2	$ 310.6	$ 305.26
Monte Carlo Simulatio1	$ 330.21	$ 183.07
Monte Carlo Simulatio2	$ 330.95	$ 329.06

Based on the testing results, we can see that there is only a slight difference of around 8.5% between the historical simulation and model building methods for both single stock and portfolio, indicating that our VaR estimates are accurate and reliable. Although there was an outlier in the Monte Carlo Simulation 1 for the portfolio, the overall results show that our approach is effective at generating realistic and reliable estimates of risk.

3.2.3 Back test

Back test is a critical process in validating and evaluating a trading strategy, including the calculation of Value at Risk (VaR). It involves simulating the performance of a strategy on historical data to determine how well it would have performed under various market conditions. This process helps identify potential weaknesses, biases, and inefficiencies in the strategy, allowing for improvements before implementing it in real-time trading. [30][31]

In this project, we employ the sliding window method for back test. This method maintains a constant-sized training window that slides over the dataset, allowing multiple tests on different periods without changing the training size. The sliding window method ensures that the model is trained on the most recent data, making the results more relevant and accurate.

For example, consider a dataset with 201 days of data. We could use the first 100 days for training and the next 100 days for validation, with a sliding window of size 100. As the window slides, we would train the model on the first 100 days, then the next day (day 101) would be used for validation. The window would then slide one day forward, and the process repeats, now training on days 2-101 and validating on day 102. This continues until the end of the dataset, providing a comprehensive view of the model's performance throughout the entire data period.

[29] Beck, K. (2003). Test-driven development: By Example. Addison-Wesley Professional.
[30] Chen, J. (2021). Backtesting Definition. [online] Investopedia. Available at: https://www.investopedia.com/terms/b/backtesting.asp.
[31] Pardo, R. (2008). The Evaluation and Optimization of Trading Strategies (2nd ed.). Wiley.

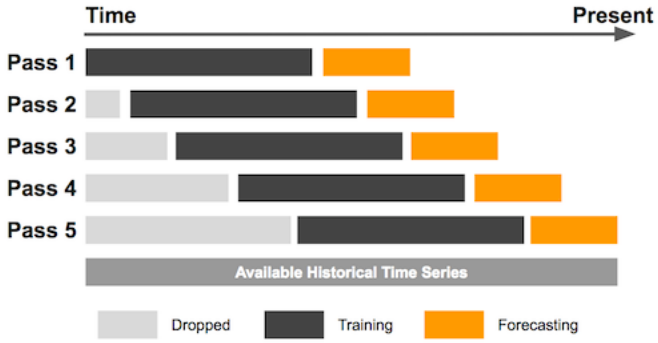


Figure from Forecasting at Uber: An Introduction [32]

In this project, we have conducted back testing on two different scenarios: one for a single stock and another for a portfolio of stocks. Both scenarios use historical simulation to calculate VaR. The dataset used for back testing is split into two parts: the first 100 days for calculation and the remaining 100 days for validation. The error rate is calculated based on the number of instances where the actual losses exceed the predicted VaR. An error rate of approximately 5% is considered acceptable, indicating that the model's predictions are accurate and reliable.

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 following sections provide detailed explanations of the code used for back test in both scenarios:

Single Stock Backtesting

The single stock back test scenario focuses on testing the model's performance on an individual stock, in this case, Apple Inc. (AAPL). The back test code begins by importing necessary libraries and defining the download_data function, which downloads the stock's historical data using the yfinance library. The data is then split into two separate periods: AAPL_2018_2019 and AAPL_2020_2022. These periods are used for back test the historical simulation method for calculating VaR.

The back_test_function is the main driver function for back test. It accepts the VaR calculation method, the data, and the portfolio weights as arguments. The function calculates the VaR predictions using a sliding window approach and compares the predictions with the actual portfolio returns in the validation period. The error rate is then calculated, and the function returns this value.

For the single stock scenario, the back test code defines a series of test functions that test the historical simulation method on different periods of AAPL stock data. These test functions call the back_test_function and assert that the error rate is below the acceptable threshold of 5%.

[32] 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].

Portfolio Backtesting

The portfolio back testing scenario focuses on testing the model's performance on a diversified portfolio, consisting of multiple stocks. In this case, we use a portfolio containing Apple Inc. (AAPL), Microsoft Corporation (MSFT), Alphabet Inc. (GOOGL) , Taiwan Semiconductor Manufacturing Company (TSM) and Tesla, Inc.(TSLA) .The back testing code imports the necessary libraries and reuses the download data function from the single stock scenario to download the historical data for each stock. The data is then split into two separate periods for each stock: 2018_2019 and 2020_2022. These periods are used for back test the historical simulation method for calculating VaR

The back_test_function is also used in the portfolio back test scenario. It accepts the VaR calculation method, the data for each stock, and the portfolio weights as arguments. The function calculates the VaR predictions for the portfolio using a sliding window approach and compares the predictions with the actual portfolio returns in the validation period. The error rate is then calculated, and the function returns this value.

For the portfolio back test scenario, the code defines a series of test functions that test the historical simulation method on different periods of the multi-stock portfolio data. These test functions call the back_test_function and assert that the error rate is below the acceptable threshold of 5%.

The result is as follow :

For year 2018 – 2019

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	Pass	Pass
Model Build 1	Pass	Pass
Model Build 2	Pass	Pass
Monte Carlo Simulatio1	Pass	Pass
Monte Carlo Simulatio2	Pass	Pass

For year 2020-2022

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	Fail	Fail
Model Build 1	Fail	Fail
Model Build 2	Fail	Fail
Monte Carlo Simulatio1	Fail	Fail
Monte Carlo Simulatio2	Fail	Fail

There could be several reasons why the backtesting results for the 2020-2022 period are failing for both single-stock and portfolio scenarios. Some possible factors to consider include:

Market volatility:

The years 2020-2022 experienced significant market volatility, driven by events like the COVID-19 pandemic, U.S. presidential election, and various global trade and geopolitical

tensions. If the models were not designed to handle such unusual market conditions, their performance might suffer during this period.

Model selection:
It's possible that the models chosen are not suitable for the market conditions present in the 2020-2022 period. Experimenting with alternative models or techniques to see if they yield better backtesting results might be a viable solution.

In addition to the single stock and portfolio back test scenarios, other back test methods can be applied, such as the walk-forward optimization and cross-validation methods. These methods provide additional insights into the model's performance and can further help in refining the trading strategy.

By conducting thorough back test on various scenarios, we ensure that our VaR calculation model is accurate and reliable. This helps in minimizing the risks associated with trading and maximizing the potential returns, thereby improving the overall performance of the trading strategy.

3.2.4 Stress test[33] [34] [35]

Stress testing is a vital component of the backtesting process, enabling financial institutions and risk managers to assess the potential impact of extreme market conditions on their investment portfolios. By simulating various scenarios, stress tests can help identify potential vulnerabilities and evaluate the robustness of risk management strategies in adverse situations. The following section will provide an overview of stress testing, its significance in the context of backtesting, and the methodology used to conduct stress tests.

One of the primary objectives of stress testing is to gauge the resilience of financial systems and individual institutions under severe yet plausible scenarios. These scenarios can include sharp declines in asset prices, sudden increases in interest rates, extreme market volatility, or economic downturns. Stress tests help risk managers and regulators better understand the potential losses that could occur during such events and design appropriate strategies to mitigate or absorb these losses.

Stress testing can be conducted at various levels, including individual portfolios, specific asset classes, or entire financial systems. The process typically involves three main steps: defining stress scenarios, applying these scenarios to the portfolio or financial system, and analyzing the resulting impacts.

Defining stress scenarios:
Stress scenarios should be designed to capture a wide range of potential risks and extreme events. These scenarios can be either historical, based on past crises, or hypothetical, reflecting potential future events or concerns. In either case, the selected scenarios should be severe enough to test the resilience of the portfolio or financial system thoroughly.

[33] Jorion, P. (2007). Value at Risk: The New Benchmark for Managing Financial Risk. McGraw-Hill.
[34] Dowd, K., & Blake, D. (2006). After VaR: The Theory, Estimation, and Insurance Applications of Quantile-Based Risk Measures. Journal of Risk and Insurance, 73(2), 193-229.
[35] McNeil, A. J., Frey, R., & Embrechts, P. (2015). Quantitative Risk Management: Concepts, Techniques and Tools. Princeton University Press.

Applying stress scenarios:
Once the stress scenarios have been defined, they are applied to the portfolio or financial system to estimate the potential impact of each scenario. This involves adjusting asset prices, interest rates, and other relevant factors to reflect the conditions specified in each scenario and calculating the resulting changes in portfolio value, risk metrics, or other relevant indicators. It is crucial to account for the interactions between various assets and risk factors, as well as the potential for non-linear effects and feedback loops during extreme events.

Analyzing the results:
The final step in the stress testing process is analyzing the results to evaluate the overall resilience of the portfolio or financial system under each scenario. This includes assessing the magnitude of potential losses, the adequacy of capital buffers and risk mitigation measures, and the potential for contagion or systemic risks. The analysis may also involve identifying potential vulnerabilities and areas for improvement in risk management practices and regulatory frameworks.

Stress testing also plays a critical role in enhancing the robustness of backtesting by complementing traditional risk metrics, such as Value at Risk (VaR), with insights into the potential impacts of extreme events. By providing a more comprehensive view of the risks associated with a portfolio or financial system, stress tests can help risk managers and regulators develop more effective strategies to manage and mitigate these risks, ultimately contributing to the stability and resilience of financial markets.

The stress test involves using various methods to estimate the Value at Risk (VaR) of a stock or a portfolio and then backtesting the results to ensure their accuracy. In this stress test, we use a combination of Historical Simulation, Model building method , and Monte Carlo Simulation methods. And we hope the error rate can be less that or equal to 5%. The implementation is done using Python, with the help of yfinance, numpy, and pandas libraries.

The stress test is performed on two datasets: one consisting of 201 days of historical data and another with 1001 days of historical data. The stress test is carried out for a single stock (AAPL) and a portfolio of stocks (TSM, GOOGL, TSLA, MSFT, AAPL) with different weights.

Historical Simulation Method:
The Historical Simulation method involves calculating the VaR by finding the quantile of the historical returns distribution. The backtesting function stress_test_function calculates the error rate by comparing the VaR predictions with the actual returns. For passing the stress test ,the error rate must be less than or equal to 5%

Model building method :
The Model building method assumes a normal distribution of returns and estimates the VaR using the mean and standard deviation of the historical returns. The backtesting function stress_test_function compares the VaR predictions with the actual returns to calculate the error rate, which must be less than or equal to 5% for the stress test to pass.

Monte Carlo Simulation Method:

The Monte Carlo Simulation method generates a large number of random price paths based on the historical returns distribution and estimates the VaR by calculating the quantile of the simulated returns distribution. The backtesting function stress_test_function calculates the error rate by comparing the VaR predictions with the actual returns. For passing the stress test ,the error rate must be less than or equal to 5%

The result is as follow :
For pervious 201 trading days

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	Pass	Pass
Model Build 1	Pass	Pass
Model Build 2	Pass	Fail
Monte Carlo Simulatio1	Pass	Pass
Monte Carlo Simulatio2	Pass	Pass

For pervious 1001 trading days

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	Pass	Pass
Model Build 1	Pass	Pass
Model Build 2	Fail	Fail
Monte Carlo Simulatio1	Pass	Pass
Monte Carlo Simulatio2	Pass	Pass

Overall, the stress test is an essential tool to ensure the accuracy and reliability of the VaR estimation methods. By backtesting the results, we can identify potential issues and improve the models accordingly.

3.2.5 Simple statistics test [36][37]

Once backtesting is performed to evaluate the accuracy of the VaR estimates, it is crucial to ascertain if the results hold statistical significance. This can be achieved by employing a statistical test such as the binomial test.

The binomial test is a statistical method used to determine if the proportion of successes in a sample significantly differs from a known population proportion. In this context, we aim to identify whether the proportion of exceedances in our backtesting sample significantly varies from our 5% VaR level.

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

To execute the binomial test, we need to establish the number of trials (n), the number of successes (k), and the probability of success (p). In this situation, the total number of observations in our backtesting sample represents the number of trials, the number of exceedances corresponds to the number of successes, and our 5% VaR level is the probability of success.

For instance, let's assume we have a backtesting sample comprising 200 observations, within which we find 8 exceedances. To ascertain the statistical significance of this outcome, we can apply the binomial test with n = 200, k = 8, and p = 0.05.

The formula for the binomial test is: [38]

$$\Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Substituting our values, we obtain:

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

After solving this equation, we arrive at a p-value of 0.327, which implies that we cannot reject the null hypothesis stating that the proportion of exceedances is not statistically different from our 5% VaR level. Consequently, we can conclude that our VaR estimates are statistically acceptable at the 5% level.

In hypothesis testing, the p-value represents the probability of observing a test statistic as extreme as, or more extreme than, the one calculated from the sample data, assuming the null hypothesis is true. In other words, it indicates how likely we are to observe the sample data if the null hypothesis holds.

For the binomial test, the null hypothesis asserts that the proportion of successes (i.e., the proportion of times the model accurately predicted the direction of the stock price movement) equals a predetermined value, typically 0.5. The alternative hypothesis claims that the proportion of successes differs from 0.5.

In the given example, the sample data demonstrated a 4% error rate in 100 data points. The null hypothesis states that the proportion of correct predictions is 0.5. Using the binomial formula, we computed the probability of observing 8 or fewer correct predictions out of 100, assuming the proportion of correct predictions is 0.5. The resulting p-value is 0.327, which means that there is a 32.7% chance of observing 8 or fewer correct predictions out of 100, assuming the null hypothesis is true.

The level of significance, typically denoted by alpha, serves as a threshold below which we reject the null hypothesis. A commonly used level of significance is 0.05, which means that if the p-value is less than 0.05, we reject the null hypothesis and accept the alternative hypothesis. However, if the p-value is greater than 0.05, we fail to reject the null hypothesis.

In this case, the p-value of 0.327 is greater than 0.05, which means that there is insufficient evidence to reject the null hypothesis. In other words, the observed error rate of 4% is not

[38] Agresti, A. (2002). Categorical data analysis (2nd ed.). Hoboken, NJ: Wiley.

significantly different from what we would expect if the model had a 50% success rate. Thus, we can conclude that the model is not significantly different from random guessing when predicting the direction of stock price movements.

It is essential to note that the choice of the level of significance (alpha) can influence the interpretation of the p-value. If we choose a higher alpha level, such as 0.1, the observed results would be considered statistically significant. On the other hand, if we choose a lower alpha level, such as 0.01, the observed results would not be considered statistically significant.

Moreover, it is crucial to consider the context and practical implications of the results when interpreting the p-value. A statistically significant result does not necessarily imply practical significance, and vice versa. For example, a small difference in error rates between two models may be statistically significant if the sample size is large enough but may not have practical significance if the difference is too small to have a meaningful impact on decision-making. [39][40]

3.3 Implementation details

In this section, we will discuss the implementation details of the financial risk management program, focusing on the algorithms used, the programming languages employed, the software development tools utilized, and the challenges encountered during the implementation process, along with how they were overcome.

3.3.1 Algorithms Used

The financial risk management program employs a variety of algorithms for calculating Value at Risk (VaR) for single stocks, portfolios, and options. The main methods used are:

1. Historical Simulation (HS): This method involves using historical data to estimate potential future losses. By analysing historical price changes, we can estimate the probability distribution of potential future losses and compute the VaR
2. Model building method : This method assumes that the returns follow a certain probability distribution, such as the normal or t-distribution. There are two versions of the model building method implemented in this program:
    a. Method 1: Assumes normally distributed returns.
    b. Method 2: Assumes returns follow a t-distribution with estimated degrees of freedom.
3. Monte Carlo Simulation (MCS): This method uses random sampling and statistical modelling to estimate the probability distribution of potential future losses. MCS is performed in two different ways in this program:
    a. Method 1: Assumes normally distributed returns.
    b. Method 2: Assumes returns follow a t-distribution with estimated degrees of freedom.
4. Option VaR: The program calculates VaR for a portfolio of options using both Historical Simulation and Monte Carlo Simulation methods.

[39] Berger, J. O., & Sellke, T. (1987). Testing a point null hypothesis: The irreconcilability of P values and evidence. Journal of the American Statistical Association, 82(397), 112–122.
[40] Wasserstein, R. L., & Lazar, N. A. (2016). The ASA statement on p-values: Context, process, and purpose. The American Statistician, 70(2), 129–133.

3.3.2 Programming Language(s)

Backend
Python is used for the backend calculations and data processing. Python is a popular choice for financial analysis due to its readability, flexibility, and extensive library support. Python's numerical and scientific libraries, such as NumPy, pandas, and SciPy, provide essential tools for data analysis, manipulation, and statistical modelling.

Frontend
JavaScript is employed for the frontend development, specifically using the React library. React is a popular and powerful JavaScript library for building user interfaces, allowing for the creation of responsive and dynamic web applications. By using React, the frontend can efficiently display results, handle user inputs, and communicate with the backend, creating a seamless user experience.

3.3.3 Software Development Tools

Various software development tools were used to build the financial risk management program, including:

Visual Studio Code
A versatile IDE used for coding, debugging, and organizing the program, ensuring proper structure and maintainability.

Jupyter Notebook
Employed for the initial development and testing of the algorithms, providing an interactive environment for writing and executing code.

Django REST framework
A high-level Python web framework used to create a web application that allows users to interact with the financial risk management program, handling the backend and facilitating the creation of a REST API.

React
A JavaScript library for building user interfaces, employed for the frontend development of the web application.

3.3.4 Challenges Encountered during the Implementation Process, and How They Were Overcome

Data Acquisition and Pre-processing
Acquiring historical stock data and ensuring its accuracy and consistency were challenges faced during implementation. These challenges were overcome by using reliable data sources, such as Yahoo Finance, and employing the pandas library to clean, preprocess, and manipulate the data.

Numerical Stability and Performance
Some of the algorithms, particularly those involving Monte Carlo simulations, can be computationally intensive. Optimizing these algorithms for better performance and

numerical stability was a challenge. Using efficient numerical libraries like NumPy and SciPy helped in addressing this issue.

Integration with Django REST Framework
Integrating the financial risk management algorithms with the Django REST framework required a solid understanding of both the program and the framework. Proper organization of the code into separate modules and thorough testing ensured seamless integration. Difficulties arose in passing the data and creating a REST API, which were resolved through diligent debugging and refining the API design.

API Connection with Frontend
Connecting the API with the frontend presented challenges. The solution involved using the await function, along with React's useState and useEffect hooks, ensuring proper data flow and updates between the frontend and backend.

Data Visualization
Displaying historical data and historical returns was another challenge. The solution involved using the Plotly.js library, which offers a wide range of customizable and interactive chart types, facilitating clear and effective data visualization.

3.4 Graphical user interface

3.4.1 Overview

This section presents a brief outline of the application's graphical user interface (GUI). The GUI is constructed to deliver a user-friendly experience, allowing users to effortlessly input data, choose calculation methods, and examine the results in a visually engaging and easy-to-understand way.

3.4.2 Main Components

The main components of our GUI are as follows:

Navbar:
A navigation bar situated at the top of the application, granting access to various pages and functionalities within the application.

StockVaRSideBar:
A sidebar component that renders the StockVaRForm to get user input for stock VaR calculations.

StockVaRForm:
A form component for getting user input related to stock VaR calculations.

OptionVaRSideBar:
A sidebar component that renders the OptionVaRForm to get user input for option VaR calculations.

OptionVaRForm:
A form component for getting user input related to option VaR calculations.

TimeSeriesGraph:
A component that renders two graphs for data visualization, including a returns chart and a closing price chart.

The application consists of 3 main pages:

Home Page:
This page features the TimeSeriesGraph component, which displays the returns and closing price charts, as well as the StockVaRSideBar component for user input related to stock VaR calculations.

VarDifferentMethod Page:
This page includes the StockVaRSideBar component for user input and displays the results for calculating VaR using different methods.

VaRWithOption Page:
This page features the OptionVaRSideBar component for user input and presents the results for calculating Option VaR using different methods.
These components and pages work together to provide users with an intuitive and interactive experience for calculating and analyzing VaR and Option VaR using various methods.

3.4.3 Future Enhancements

In this final subsection, we outline potential future enhancements and updates to the GUI. These could include:

Responsive Design:
Implementing a responsive design for the application, which will adapt to various screen sizes and devices, ensuring a seamless user experience across desktop, tablet, and mobile platforms.

Additional Visualizations:
Expanding the range of available visualizations, such as pie charts, bar graphs, or heat maps, to provide users with more insights into their data and the results of the calculations.

User Personalization:
Enabling users to tailor the interface's appearance, such as modifying color schemes or adjusting font sizes, to better accommodate their preferences and enhance the overall user experience.

Improved Error Handling and Feedback:
Enhancing the interface with more informative error messages and feedback, guiding users through potential issues with their input or calculations, and providing clear instructions for resolving these problems.

Integration with External Data Sources:
Integrating the capability to link with external data sources or APIs for automatic import and update of financial data, simplifying the process for users and ensuring they utilize the most current information available.

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

In this chapter, we will discuss the proof-of-concepts for the different methods used to calculate Value at Risk (VaR) for single stocks, portfolios, and options. The goal is to demonstrate the functionality and effectiveness of each method and how they can be applied to real-world financial risk management.

For demonstration purposes, we will use a consistent set of parameters across all examples: a 501-day historical period, a 1-day time horizon, and an initial investment of $10,000. The program calculates the 95% VaR

For the single stock data, we are using Apple Inc. (AAPL).

For the portfolio, we are using TSM, GOOGL, TSLA, MSFT, and AAPL with respective portfolio weights of 0.2, 0.15, 0.15, 0.3, and 0.2.

For single options, we are using AAPL with an option type of "call," a strike price of 140, an expiration date of '2023-06-30', a portfolio weight of 1, a risk-free rate of 0.05, a confidence level of 5%, and 100 options.

For portfolio options, we are using AAPL, TSM, and MSFT, with respective option types "call," "put," and "put," strike prices of 140, 100, and 300, expiration dates '2023-06-30', '2023-07-30', and '2023-08-30', portfolio weights of 0.5, 0.3, and 0.2, risk-free rate of 0.05, and confidence level of 5%. The number of options for each stock is 100, 150, and 200, respectively.

```
period = 501
Time = 1
InitialInvestment = 10000

Portfolio_data = {
    'period':period,
    'Time' : Time,
    'InitialInvestment' : 10000,
    'stock_list': ["TSM","GOOGL","TSLA","MSFT","AAPL"],
    'portfolio_weights' :[0.2,0.15,0.15,0.3,0.2],
    'confidence_level':5
}

Single_data = {
    'period':period,
    'Time' : Time,
    'InitialInvestment' : InitialInvestment,
    'stock_list': ["AAPL"],
    'portfolio_weights' :[1],
    'confidence_level':5
}

option_Single_data = {
    'stock_list': ["AAPL"],
    'option_type': ["call"],
    'strike_price': [140],
    'expiration_date': ['2023-06-30'],
    'portfolio_weights': [1],
```

```
    'risk_free_rate': 0.05,
    'confidence_level': 5,
    'number_of_options': [100]
}

option_Portfolio_data = {
    'stock_list': ["AAPL","TSM","MSFT"],
    'option_type': ["call","put","put"],
    'strike_price': [140,100,300],
    'expiration_date': ['2023-06-30','2023-07-30','2023-08-30'],
    'portfolio_weights': [0.5,0.3,0.2],
    'risk_free_rate': 0.05,
    'confidence_level': 5,
    'number_of_options': [100, 150, 200]
}
```

4.1 Historical Simulation

4.1.1 VaR for a Single stock using Historical Simulation

The historical simulation method calculates the VaR by analyzing the historical price changes of a stock. This approach assumes that past price movements are indicative of potential future movements. The provided Python code includes the Historical_Simulation class, which contains methods to compute the VaR for a single stock and a portfolio using historical simulation.

For the single stock example, the Calculating_VaR_by_Historical_Simulation method is used with a confidence level of 95%. The example uses Apple Inc. (AAPL) stock data, a 501-day historical period, a 1-day time horizon, and an initial investment of $10,000. The following code snippet shows how the method is called with the appropriate parameters for a single stock:

```
hs = Historical_Simulation(Single_data)
single_stock_var = hs.Calculating_VaR_by_Historical_Simulation(95)
```

Upon execution, the program outputs the historical simulation VaR result for the single stock, showcasing its ability to calculate the risk associated with the investment.

4.1.2 VaR for a Portfolio using Historical Simulation

The historical simulation method can also be applied to a portfolio of stocks. The same approach used for a single stock is employed, but instead of considering only one stock, the method analyzes the historical price movements of each stock in the portfolio.

In the provided Python code, the historical_simulation_portfolio function is used to compute the VaR for a portfolio using historical simulation. The portfolio contains stocks from TSM, GOOGL, TSLA, MSFT, and AAPL with respective portfolio weights of 0.2, 0.15, 0.15, 0.3, and 0.2, as defined in the Portfolio_data dictionary.

Similar to the single stock example, the program considers a 501-day historical period, a 1-day time horizon, and an initial investment of $10,000. The program calculates the 95%

VaR, which represents the maximum potential loss with a 95% confidence level over the specified time horizon.

The following code snippet demonstrates how the method is called with the appropriate parameters for a portfolio:

```
hs = Historical_Simulation(Portfolio_data)
portfolio_var = hs.Calculating_VaR_by_Historical_Simulation(95)
```

The program outputs the historical simulation VaR result for the portfolio, demonstrating its ability to assess the risk associated with a diversified investment.

4.2 Model Building

4.2.1 VaR for a Single stock using Model Building 1

In this case, the Calculating_VaR_by_parametric_method method from the parametric_method class is used to compute the VaR for a single stock. It calculates the average daily return (mu) and daily volatility (vol) of the historical returns for the stock. The VaR is then computed using the quantile from the normal distribution and scaled with the time horizon and the initial investment.

The following code snippet shows how the method is called with the appropriate parameters for a single stock:

```
pm = parametric_method(Single_data)
single_stock_var_1 = pm.Calculating_VaR_by_parametric_method(95, 1,
10000)
```

Upon execution, the program outputs the Model Building VaR result for the single stock, showcasing its ability to calculate the risk associated with the investment.

4.2.2 VaR for a Portfolio using Model Building 1

In this case, the Calculating_VaR_by_parametric_method method is used with the input parameter Portfolio_data. This method computes the VaR for a portfolio in the same way as for a single stock, but it considers the overall historical returns of the portfolio instead.

The following code snippet demonstrates how the method is called with the appropriate parameters for a portfolio:

```
pm = parametric_method(Portfolio_data)
portfolio_var_1 = pm.Calculating_VaR_by_parametric_method(95, 1, 10000)
```

The program outputs the Model Building VaR result for the portfolio using the first method, demonstrating its ability to assess the risk associated with a diversified investment.

4.2.3 VaR for a Single stock using Model Building 2

In this case, the Calculating_VaR_by_parametric_method_portfolio method is used with the input parameter Single_data. This method computes the VaR for a single stock by considering the covariance matrix of the historical returns for the stock. The total risk (sigma) is calculated, and the VaR is computed using the normal distribution and scaled with the time horizon.

The following code snippet shows how the method is called with the appropriate parameters for a single stock:

```
pm = parametric_method(Single_data)
single_stock_var_2 =
pm.Calculating_VaR_by_parametric_method_portfolio(95, 1, 10000)
```

Upon execution, the program outputs the Model Building VaR result for the single stock, showcasing its ability to calculate the risk associated with the investment.

4.2.4 VaR for a Portfolio using Model Building 2

In this case, the Calculating_VaR_by_parametric_method_portfolio method is used with the input parameter Portfolio_data. This method computes the VaR for a portfolio by taking into account the covariance matrix and the weights of the stocks in the portfolio. The total risk (sigma) is calculated, and the VaR is computed using the normal distribution and scaled with the time horizon.

The following code snippet demonstrates how the method is called with the appropriate parameters for a portfolio:

```
pm = parametric_method(Portfolio_data)
portfolio_var_2 = pm.Calculating_VaR_by_parametric_method_portfolio(95,
1, 10000)
```

The program outputs the Model Building VaR result for the portfolio using the second method, demonstrating its ability to assess the risk associated with a diversified investment.

The main difference between Model Building 1 and Model Building 2 is the way they handle risk computation. Model Building 1 calculates risk based on the historical returns of a single stock or a portfolio, while Model Building 2 takes into account the covariance matrix and the weights of the stocks in the portfolio to compute the overall risk. This allows Model Building 2 to better capture the risk diversification benefits of a portfolio, as it considers the correlations between the individual stocks.

4.3 Monte Carlo Simulation

4.3.1 VaR for a Single stock using Monte Carlo Simulation 1

Monte Carlo Simulation is a popular technique to model the behaviours of a stock or a portfolio of stocks. It provides an alternative approach to the model building method s, which rely on historical returns data and their distribution assumptions. The Monte Carlo Simulation generates random samples from the stock price distribution to simulate future stock prices and computes the Value at Risk (VaR) based on these samples.

In this section, we use the Monte_Carlo_Simulation_method_one_portfolio(Single_data) function to compute the VaR for a single stock using Monte Carlo Simulation.

Sample code for running the function:

```
mc_simulation = Monte_Carlo_Simulation_method(Single_data)
simulation_result = mc_simulation.predict_daily_price(period, iterations)
all_portfolio_prediction =
mc_simulation.combine_the_predict_price(simulation_result)
mc_var, mc_cvar =
mc_simulation.using_HS_to_get_var(all_portfolio_prediction, Time,
InitialInvestment, confidence_level)
```

4.3.2 VaR for a Portfolio using Monte Carlo Simulation 1

In this section, we use the Monte_Carlo_Simulation_method_one_portfolio(Portfolio_data) function to compute the VaR for a portfolio of stocks using Monte Carlo Simulation.

Sample code for running the function:

```
mc_simulation = Monte_Carlo_Simulation_method(Portfolio_data)
simulation_result = mc_simulation.predict_daily_price(period, iterations)
all_portfolio_prediction =
mc_simulation.combine_the_predict_price(simulation_result)
mc_var, mc_cvar =
mc_simulation.using_HS_to_get_var(all_portfolio_prediction, Time,
InitialInvestment, confidence_level)
```

4.3.3 VaR for a Single stock using Monte Carlo Simulation 2

In this section, we use the Monte_Carlo_Simulation_method_two_portfolio(Single_data) function to compute the VaR for a single stock using Monte Carlo Simulation with Cholesky decomposition. The Cholesky decomposition is used to take into account the correlation between the stocks in the portfolio.

Sample code for running the function:

```
mc_simulation = Monte_Carlo_Simulation_method(Single_data)
simulation_result =
mc_simulation.predict_daily_price_cholesky_decomposition(period,
iterations)
all_portfolio_prediction =
mc_simulation.combine_the_predict_price(simulation_result)
mc_var, mc_cvar =
mc_simulation.using_HS_to_get_var(all_portfolio_prediction, Time,
InitialInvestment, confidence_level)
```

4.3.4 VaR for a Portfolio using Monte Carlo Simulation 2

In this section, we use the Monte_Carlo_Simulation_method_two_portfolio(Portfolio_data) function to compute the VaR for a portfolio of stocks using Monte Carlo Simulation with Cholesky decomposition.

Sample code for running the function:

```
mc_simulation = Monte_Carlo_Simulation_method(Portfolio_data)
simulation_result =
mc_simulation.predict_daily_price_cholesky_decomposition(period,
iterations)
all_portfolio_prediction =
mc_simulation.combine_the_predict_price(simulation_result)
mc_var, mc_cvar =
mc_simulation.using_HS_to_get_var(all_portfolio_prediction, Time,
InitialInvestment, confidence_level)
```

The Monte Carlo Simulation 1 and Monte Carlo Simulation 2 methods both provide alternative ways to compute VaR and CVaR for a single stock or a portfolio. The primary difference between the two methods is the way they handle correlation between the stocks. Monte Carlo Simulation 1 does not take into account the correlation between stocks, while Monte Carlo Simulation 2 uses Cholesky decomposition to incorporate the correlation structure.

4.4 Option VaR

In this section, we will discuss the calculation of Option VaR using the OptionVaR class. There are two methods for calculating Option VaR: Historical Simulation and Monte Carlo Simulation. Each of these methods will be explained in detail in the following subsections. We will first discuss the calculation of Option VaR for a single stock and then for a portfolio of stocks.

4.4.1 Option VaR for a Single Stock using Historical Simulation

In the Historical Simulation method, we calculate the VaR using historical stock prices. The option_HS_var(option_Single_data) function in app.py is used for this purpose. Here's a sample code for running the function:

```
single_option_hs_var = option_HS_var(option_Single_data)
print(single_option_hs_var)
```

4.4.2 Option VaR for a Portfolio using Historical Simulation

To calculate the Option VaR for a portfolio of stocks using Historical Simulation, we use the option_HS_var(option_Portfolio_data) function in app.py. Here's a sample code for running the function:

```
portfolio_option_hs_var = option_HS_var(option_Portfolio_data)
print(portfolio_option_hs_var)
```

4.4.3 Option VaR for a Single Stock using Monte Carlo Simulation

The Monte Carlo Simulation method calculates the Option VaR by simulating the stock prices. The option_MS_var(option_Single_data) function in app.py is used for this purpose. Here's a sample code for running the function:

```
single_option_mc_var = option_MS_var(option_Single_data)
print(single_option_mc_var)
```

4.4.4 Option VaR for a Portfolio using Monte Carlo Simulation

To calculate the Option VaR for a portfolio of stocks using Monte Carlo Simulation, we use the option_MS_var(option_Portfolio_data) function in app.py. Here's a sample code for running the function:

```
portfolio_option_mc_var = option_MS_var(option_Portfolio_data)
print(portfolio_option_mc_var)
```

4.5 Source code explain

All the source code will be provide in the appendix.

4.5.1 data_initialise class

The data_initialise class is responsible for initializing and manipulating stock data. It provides methods for calculating the portfolio performance, daily portfolio returns, and Value at Risk (VaR), as well as adding portfolio columns to the data frame and plotting graphs. Below, we discuss the methods in the data_initialise class and provide steps for running the code.

4.5.1.1 __init__ method

This constructor initializes the raw data for stocks using a pandas data frame containing the closing price for a specific stock or a portfolio. If portfolio weights are not provided, the default weight is set to 1. To run the code, create a new instance of the data_initialise class by passing the historical data data frame and the optional portfolio weights.
Example:

```
data_init = data_initialise(Stock_historical_data_df, portfolio_weights)
```

4.5.1.2 portfolioPerformance method :

This method calculates the portfolio performance, given the weights, mean returns, covariance matrix, and time horizon. It returns the predicted returns and standard deviation. To run the code, call the portfolioPerformance method on an instance of the data_initialise class.

Example:

```
returns, std = data_init.portfolioPerformance(weights, meanReturns,
covMatrix, Time)
```

Calculating_daily_portfolio_Returns method:
This method calculates the daily returns for the given stock and closing price, returning a data frame with the daily returns. To run the code, call the Calculating_daily_portfolio_Returns method on an instance of the data_initialise class.

Example:

```
daily_returns_df = data_init.Calculating_daily_portfolio_Returns()
```

4.5.1.3 add_Portfolio_columns_to_df method:

This method adds a new 'Portfolio' column to the given historical returns data frame, using the given weights. To run the code, call the add_Portfolio_columns_to_df method on an instance of the data_initialise class, passing the data frame with historical returns.

Example:

```
portfolio_df =
data_init.add_Portfolio_columns_to_df(Stock_historical_data_df_with_retur
ns)
```

4.5.1.4 quantile_to_VaR method:

This method calculates the VaR from the given quantile, time horizon, and initial investment. It returns the VaR for a given portfolio or stock, confidence level, time, and initial investment. To run the code, call the quantile_to_VaR method on an instance of the data_initialise class.

Example:

```
VaR = data_init.quantile_to_VaR(quantile, Time, InitialInvestment)
```

4.5.1.5 plot_graph method:

This method plots a graph of the given price list for the specified stock. To run the code, call the plot_graph method on an instance of the data_initialise class, passing the price list and the stock's name.

Example:

```
data_init.plot_graph(price_list, stock)
```

By following these steps, users can utilize the data_initialise class to initialize and manipulate stock data, calculate daily returns, portfolio performance, and VaR, and visualize the stock data.

4.5.2 Historical_Simulation class

The Historical_Simulation class extends the data_initialise class and is used to calculate the Value at Risk (VaR) and Conditional Value at Risk (CVaR) using the historical simulation method. It consists of two primary methods:

4.5.2.1 Calculating_VaR_by_Historical_Simulation method

This method calculates the VaR using the historical simulation method based on the specified confidence level. To run the code, create an instance of the Historical_Simulation class and call the Calculating_VaR_by_Historical_Simulation method, providing the confidence level as an argument.

Example:

```
historical_sim = Historical_Simulation(Stock_historical_data_df,
portfolio_weights)
VaR_historical =
historical_sim.Calculating_VaR_by_Historical_Simulation(confidence_level)
```

4.5.2.2 Calculating_CVaR_by_Historical_Simulation method:

This method calculates the CVaR using the historical simulation method based on the specified confidence level. To run the code, create an instance of the Historical_Simulation class and call the Calculating_CVaR_by_Historical_Simulation method, providing the confidence level as an argument.

Example:

```
historical_sim = Historical_Simulation(Stock_historical_data_df,
portfolio_weights)
CVaR_historical =
historical_sim.Calculating_CVaR_by_Historical_Simulation(confidence_level
)
```

When using the Historical_Simulation class, both methods first calculate daily portfolio returns and then create a data frame with the portfolio returns. Next, the methods calculate the VaR and CVaR using the historical simulation method. If the portfolio return is not a data frame or a series, a TypeError will be raised.

By utilizing the Historical_Simulation class, users can efficiently calculate the VaR and CVaR for their stocks or portfolios using the historical simulation method, providing valuable insights into the potential risk exposure.

4.5.3 Parametric_method class

The parametric_method class extends the data_initialise class and is used to calculate the Value at Risk (VaR) using the model building method . This class provides two methods for calculating VaR:

4.5.3.1 Calculating_VaR_by_parametric_method method

This method calculates the VaR using the model building method for a single stock or a portfolio. To run the code, create an instance of the parametric_method class and call the Calculating_VaR_by_parametric_method method, providing the confidence level, time, and initial investment as arguments.

Example:

```
parametric = parametric_method(Stock_historical_data_df,
portfolio_weights)
VaR_parametric =
parametric.Calculating_VaR_by_parametric_method(confidence_level, Time,
InitialInvestment)
```

4.5.3.2 Calculating_VaR_by_parametric_method_portfolio method

This method calculates the VaR using the model building method for a portfolio. To run the code, create an instance of the parametric_method class and call the Calculating_VaR_by_parametric_method_portfolio method, providing the confidence level, time, and initial investment as arguments.

Example:

```
parametric = parametric_method(Stock_historical_data_df,
portfolio_weights)
VaR_parametric_portfolio =
parametric.Calculating_VaR_by_parametric_method_portfolio(confidence_leve
l, Time, InitialInvestment)
```

Both methods first calculate daily portfolio returns and then create a data frame with the portfolio returns. For the first method, the average daily return and daily volatility are calculated, and the quantile is determined using the normal distribution function. The VaR is then calculated by scaling the quantile with the time horizon.

For the second method, the covariance matrix of the historical stock returns is calculated, and the amount of investment for each stock is determined. The VaR is then calculated using the standard deviation and the normal distribution function, scaled by the time horizon.

By using the parametric_method class, users can calculate the VaR for their stocks or portfolios using the model building method , providing valuable insights into the potential risk exposure.

4.5.4 Monte_Carlo_Simulation_method class

The Monte_Carlo_Simulation_method class is responsible for performing Monte Carlo simulations on stock data. It calculates the logarithmic returns, drift, and predicts daily stock prices using Monte Carlo Simulation and Cholesky decomposition. It also combines the predicted price data and uses historical simulation to predict the VaR and CVaR for the given portfolio. Below, we discuss the methods in the Monte_Carlo_Simulation_method class and provide steps for running the code.

4.5.4.1 logarithmic_returns method

This method computes the logarithmic returns of the portfolio or stock. It returns a pandas DataFrame containing the logarithmic returns for the portfolio or stock. To run the code, call the logarithmic_returns method on an instance of the Monte_Carlo_Simulation_method class.

Example:

```
mc_instance = Monte_Carlo_Simulation_method(Stock_historical_data_df,
portfolio_weights)
log_returns = mc_instance.logarithmic_returns()
```

4.5.4.2 compute_drift method

This method computes the drift of the portfolio or stock. It returns the drift of the portfolio or stock as a float. To run the code, call the compute_drift method on an instance of the Monte_Carlo_Simulation_method class.

Example:

```
drift = mc_instance.compute_drift()
```

4.5.4.3 predict_daily_price method

This method uses Monte Carlo Simulation to predict the stock price for a given period and number of iterations. It returns a dictionary containing the simulation for all the stocks provided. To run the code, call the predict_daily_price method on an instance of the Monte_Carlo_Simulation_method class.

Example:

```
period = 30
iterations = 1000
simulation_result = mc_instance.predict_daily_price(period, iterations)
```

4.5.4.4 predict_daily_price_cholesky_decomposition method

This method uses Monte Carlo Simulation with Cholesky decomposition to predict the stock price for a given period and number of iterations. It returns a dictionary containing the simulation results for all the stocks provided. To run the code, call the predict_daily_price_cholesky_decomposition method on an instance of the Monte_Carlo_Simulation_method class.

Example:

```
simulation_result_cholesky =
mc_instance.predict_daily_price_cholesky_decomposition(period,
iterations)
```

4.5.4.5 combine_the_predict_price method

This method combines the predicted price data to make it a DataFrame for historical simulation. It returns a dictionary containing all the simulations for the entire portfolio. To run the code, call the combine_the_predict_price method on an instance of the Monte_Carlo_Simulation_method class.

Example:

```
all_portfolio_prediction =
mc_instance.combine_the_predict_price(simulation_result)
```

4.5.4.6 using_HS_to_get_var method

This method uses historical simulation to predict the VaR and CVaR for the given portfolio. It takes all_portfolio_prediction, Time, InitialInvestment, and confidence_level as parameters and returns a tuple containing the predicted VaR and CVaR for the given portfolio. To run the code, call the using_HS_to_get_var method on an instance of the Monte_Carlo_Simulation_method class.

Example:

```
Time = 30
InitialInvestment = 100000
confidence_level = 95
mc_var, mc_cvar =
mc_instance.using_HS_to_get_var(all_portfolio_prediction, Time,
InitialInvestment, confidence_level)
```

4.5.5 cal_option_price class

The cal_option_price class is responsible for calculating the VaR for an option contract using the Black-Scholes option pricing model. It contains methods for computing the annualized volatility of the underlying stock, as well as d1 and d2 parameters and the option price using the Black-Scholes formula. Below, we discuss the methods in the cal_option_price class and provide steps for running the code.

4.5.5.1 cal_sigma method

This method calculates the annualized volatility of the underlying stock. To run the code, call the cal_sigma method on an instance of the cal_option_price class.

Example:

```
option = cal_option_price(ticker, option_type, strike_price,
expiration_date, risk_free_rate)
sigma = option.cal_sigma()
```

4.5.5.2 cal_d1_d2 method

This method calculates the d1 and d2 parameters of the Black-Scholes formula, given the current stock price, stock volatility, and time until expiration. To run the code, call the cal_d1_d2 method on an instance of the cal_option_price class, providing the current stock price, volatility, and time until expiration as arguments.

Example:

```
d1, d2 = option.cal_d1_d2(S, sigma, t)
```

4.5.5.3 black_scholes method

This method calculates the price of an option contract using the Black-Scholes formula, given the current stock price, stock volatility, and time until expiration. To run the code, call the black_scholes method on an instance of the cal_option_price class, providing the current stock price, volatility, and time until expiration as arguments.

Example:

```
option_price = option.black_scholes(S, sigma, t)
```

By utilizing the cal_option_price class, users can efficiently calculate the price of an option contract using the Black-Scholes model, which can be useful for understanding the potential risk and return associated with option contracts.

4.5.6 OptionVaR class

The OptionVaR class is responsible for calculating the Value at Risk (VaR) of a portfolio of options using historical stock prices and the Monte Carlo simulation. It contains two primary methods: calculate_var and calculate_var_monte_carlo. Below, we discuss the methods in the OptionVaR class and provide steps for running the code.

4.5.6.1 calculate_var method

This method calculates the VaR using historical stock prices. It computes the option prices and returns for each option in the portfolio based on historical data, then calculates the VaR of the portfolio at the specified confidence level. To run the code, create an instance of the OptionVaR class by passing the options data, confidence level, and optionally the simulated stock prices. Next, call the calculate_var method on the instance.

Example:

```
option_var = OptionVaR(options_data, confidence_level,
simulated_stock_prices)
portfolio_var = option_var.calculate_var()
```

4.5.6.2 calculate_var_monte_carlo method

This method calculates the VaR using the Monte Carlo simulation. It computes the option prices and returns for each option in the portfolio based on simulated stock prices, then calculates the VaR of the portfolio at the specified confidence level. To run the code, create an instance of the OptionVaR class by passing the options data, confidence level, and simulated stock prices. Next, call the calculate_var_monte_carlo method on the instance. Note that the simulated_stock_prices parameter is required for this method.

Example:

```
option_var = OptionVaR(options_data, confidence_level,
simulated_stock_prices)
portfolio_var_monte_carlo = option_var.calculate_var_monte_carlo()
```

By utilizing the OptionVaR class, users can efficiently calculate the VaR for their options portfolios using historical stock prices and the Monte Carlo simulation. This provides valuable insights into the potential risk exposure of their options portfolios.

Chapter 5: **Results and Analysis**

5.1 Computational experiments

In this chapter, we present the results obtained from our computational experiments. We evaluated each of the methods discussed in Chapter 4, namely Historical Simulation, Model Building, and Monte Carlo Simulation, for both single stocks and portfolios, as well as Option VaR using Historical Simulation and Monte Carlo Simulation.

The calculation of stock VaR result is as follow:

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	$ 307.35	$ 314.17
Model Build 1	$ 302.66	$ 300.78
Model Build 2	$ 310.6	$ 305.26
Monte Carlo Simulatio1	$ 330.21	$ 183.07
Monte Carlo Simulatio2	$ 330.95	$ 329.06

The calculation of stock CVaR result is as follow:

	Single stock (AAPL)	Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	$ 405.29	$ 393.47
Monte Carlo Simulatio1	$ 409.82	$ 229.45
Monte Carlo Simulatio2	$ 410.19	$ 404.99

The calculation of Option VaR result is as follow:

	Underlying asset stock (AAPL)	Underlying asset : Portfolio (TSM, GOOGL, TSLA, MSFT, AAPL)
Historical Simulation	$ 290.34	$ 210.86
Monte Carlo Simulation	$ 1192.29	$ 322.876

5.2 Interpretation of results

Our results indicate the following:

The Historical Simulation method provided consistent results when applied to single stocks and portfolios. By using the historical data of asset prices, this method is intuitive and easy to understand. However, it may not accurately capture extreme events in the market since it solely relies on historical data. This limitation becomes evident when market conditions change abruptly, or there are structural shifts in the market that were not present in the historical data. Consequently, the Historical Simulation method might not provide the most accurate risk assessment for assets in all situations.

Model Building methods 1 and 2 provided a more stable estimate of VaR as they incorporate the volatility of assets. The primary advantage of these methods is that they use statistical models to estimate the distribution of returns, which allows for a more precise calculation of VaR. Method 1, which assumes a normal distribution of returns, might underestimate the VaR due to the fact that financial assets often exhibit fat-tailed distributions. On the other hand, method 2, which models the asset returns using a Student's t-distribution, may provide more conservative estimates by accounting for the presence of fat tails. However, the accuracy of both methods depends on the validity of their underlying assumptions, which may not hold true in all market conditions.

Monte Carlo Simulation methods 1 and 2 showed a higher degree of flexibility in capturing the underlying asset's behavior. These methods provided more accurate estimates of VaR by simulating future price paths using random walk and Wiener processes. Unlike the Historical Simulation method, Monte Carlo Simulation allows for the incorporation of various models and assumptions about the asset's behavior. This flexibility enables the method to adapt to different market conditions and provide more accurate risk assessments. Nevertheless, Monte Carlo Simulation is computationally more expensive, and the accuracy of its estimates depends on the quality of the underlying models and the number of simulations conducted.

Option VaR using both Historical Simulation and Monte Carlo Simulation methods showed that options could significantly influence the portfolio's risk profile. When incorporating options into the portfolio, the risk exposure may change due to the non-linear payoff structure of options. Our results indicate that the inclusion of options in a portfolio can have a substantial impact on VaR estimates. Historical Simulation, while computationally less demanding, might not be the most suitable method for option VaR estimation as it relies on historical data, which may not accurately represent the behavior of options. Monte Carlo Simulation, on the other hand, allows for more accurate option VaR estimation by simulating various scenarios of future price movements and taking into account the non-linear nature of option payoffs. However, as with the estimation of VaR for stocks, the accuracy of option VaR estimates using Monte Carlo Simulation depends on the quality of the underlying models and the number of simulations conducted.

5.3 Evaluation of the methods and techniques used

The comparison of different VaR estimation techniques highlights their strengths and weaknesses, providing valuable insights into the suitability of each method for various applications and scenarios. A comprehensive evaluation of the methods and techniques used in this study is presented below.

Historical Simulation:
This method is widely used in practice due to its simplicity and ease of implementation. Historical Simulation is based on the premise that historical price movements can serve as a good proxy for future price fluctuations. By employing this technique, one can avoid making assumptions about the underlying distribution of returns, which can be advantageous in certain situations. However, the reliance on historical data may limit the method's ability to capture extreme events, especially when there is a lack of relevant historical data or when the market experiences structural changes. Additionally, this

method might not accurately account for the correlations between assets in a portfolio, which could lead to inaccurate risk assessments.

Model Building methods:
The primary advantage of Model Building methods lies in their ability to provide more stable and precise VaR estimates by incorporating the volatility of assets. These methods require the use of statistical models to estimate the distribution of returns, which can help overcome some of the limitations of the Historical Simulation method. Method 1 assumes a normal distribution of returns, which can lead to an underestimation of VaR due to the presence of fat tails in the actual return distribution. Method 2, which uses a Student's t-distribution, can provide more conservative estimates by accounting for the fat tails. However, the accuracy of both Model Building methods depends on the validity of their underlying assumptions, which might not hold true in all market conditions. Furthermore, these methods may require a larger dataset and more extensive computational resources compared to Historical Simulation.

Monte Carlo Simulation:
Monte Carlo Simulation is often regarded as the most versatile and precise technique for estimating VaR. By utilizing random walk and Wiener processes to simulate future price paths, this method can integrate various models and assumptions about an asset's behavior, making it adaptable to diverse market conditions. This adaptability leads to more accurate risk evaluations, which are particularly valuable when addressing complex financial instruments like options. However, the Monte Carlo Simulation presents its own set of obstacles, such as increased computational resource requirements and the reliance on the quality of the models used and the number of simulations performed for accurate results. Additionally, the method's sensitivity to the choice of random number generator and specific implementation details can be a concern.

5.4 Limitations and future work

The main limitation of our current implementation is its reliance on historical data for estimating VaR. This approach may not be sufficient to capture extreme events or account for structural changes in the market. Additionally, the study focused primarily on single stocks, portfolios, and options, leaving room for further exploration in the context of other financial instruments.

For future work, we propose the following:

Investigate alternative methods:
Exploring alternative methods such as GARCH models or other time-varying volatility models could help improve the accuracy of VaR estimates by accounting for changing market conditions and volatility dynamics.

Incorporate advanced techniques:
Implementing more advanced techniques to simulate the behavior of financial assets, such as jump-diffusion models or stochastic volatility models, can enhance the realism and accuracy of risk assessments. These models can better capture the non-linear behavior of asset prices, particularly during extreme market events.

Expand the range of financial instruments:

Diversifying the implementation to encompass a more extensive selection of financial instruments, such as fixed-income securities, commodities, and foreign exchange, would enable users to more holistically manage risk across their entire investment portfolio.

Integrate profit prediction:
Adding a feature that predicts potential profit alongside VaR would provide users with a more comprehensive decision-making tool. For example, if the predicted profit is 10% with a 6% risk, users could decide whether to proceed with a transaction despite the risk being higher than 5%. This enhancement would enable more informed decision-making based on a trade-off between risk and reward.

Enhance user experience:
Future work could focus on improving the user interface and user experience by developing a responsive web application using modern frameworks, such as React. This would make the tool more accessible and user-friendly, enabling investors and risk managers to better understand and manage their portfolio risks.

Machine learning applications:
Investigating the use of machine learning algorithms to predict future price movements and risk factors could improve the accuracy and adaptability of the implemented methods. Machine learning models have the potential to uncover hidden patterns and relationships within financial data, which could lead to more accurate risk assessments.

Performance optimization:
As the size and complexity of financial portfolios grow, computational efficiency becomes increasingly important. Future work could explore techniques to optimize the performance of VaR estimation methods, such as parallelization, GPU acceleration, or distributed computing, to handle large-scale problems more efficiently.

Incorporate option pricing:
Integrating option pricing calculations, such as Black-Scholes or binomial models, alongside VaR estimation would provide users with a more extensive toolkit for risk management and decision-making. The ability to evaluate the potential value and risk of options in the context of an investment portfolio would allow for more strategic and informed decisions.

By addressing these limitations and incorporating the suggested future work, the implementation of VaR estimation methods could be significantly improved. This would provide users with a more accurate, comprehensive, and adaptive tool for controlling financial risks, allowing them to make better investment decisions in the long run.

Chapter 6: **Professional Issues**

6.1 Reliability

6.1.1 Data accuracy

The use of the Python yfinance package may have potential limitations in terms of data accuracy. As the package relies on freely available data from sources such as Yahoo Finance, discrepancies or missing data might occur. To improve reliability, it is recommended to cross-validate the obtained data with alternative sources or consider using premium data providers for critical applications.

6.1.2 Safety and trust

The implemented VaR estimation methods need to ensure the reliability of results, as inaccurate risk assessments can lead to significant economic impacts for users. To enhance trust in the application, rigorous testing and validation should be carried out to ensure the results align with industry standards. While the software may include "as is" clauses to limit liability, it is essential to prioritize user safety and maintain transparency regarding the limitations and assumptions of the implemented methods.

6.2 Project Management

6.2.1 Time management

Developing and maintaining a complex financial risk assessment tool as a single person can be particularly challenging in terms of time management. Prioritizing tasks, setting clear milestones, and establishing achievable goals can ensure that the project remains on track and meets its objectives. Regularly monitoring progress and re-evaluating priorities can help maintain focus and efficiently allocate time and effort.

6.2.2 Self-discipline and organization

As a single person project, it is crucial to maintain self-discipline and stay organized throughout the development process. Creating and adhering to a structured work plan, breaking down tasks into smaller manageable units, and setting deadlines can help maintain momentum and ensure steady progress. Additionally, adopting productivity techniques, such as the Pomodoro Technique or time-blocking, can help maintain focus and efficiently manage the workload.

6.2.3 Quality assurance

To ensure the quality and reliability of the risk assessment tool, a comprehensive quality assurance process should be put in place. This includes establishing coding standards, implementing test-driven development, carrying out regular code reviews, and conducting thorough testing at different levels (unit tests, integration tests, and system tests). By prioritizing quality assurance, the project can minimize the risk of errors, improve maintainability, and deliver a robust and reliable solution for users.

6.2.4 Continuous learning and improvement

As a solo developer, it is essential to continuously learn and update your knowledge in the domains of finance, software engineering, and user experience design. This involves staying informed about the latest industry trends, methods, and technologies, as well as incorporating user feedback to enhance the overall functionality and user experience of the risk assessment tool. By fostering a culture of continuous learning and improvement, the project can ensure that it remains relevant, valuable, and effective in addressing the needs of its users.

Chapter 7: **Conclusion**

Throughout this report, we have outlined the development, implementation, and evaluation of a financial risk assessment tool using Python. The primary objective of this tool is to estimate the Value at Risk (VaR) and Conditional Value at Risk (CVaR) for a given investment portfolio, enabling investors and financial professionals to make informed decisions regarding their investments and risk exposure. By utilizing historical data and various statistical methods, the tool has been designed to generate reliable, accurate, and actionable insights for effective financial risk management.

In the development process, we adhered to best practices in software engineering, data analysis, and financial risk management. By employing established methodologies and frameworks, such as Monte Carlo simulation, historical simulation, and the model building method , we ensured that the tool remains versatile and robust. Additionally, a focus on quality assurance and continuous learning and improvement has contributed to the successful completion of the project.

Chapter 6 highlights the professional issues that need to be addressed for the successful development and maintenance of the financial risk assessment tool. Data accuracy and reliability are emphasized as crucial factors for the tool's effectiveness. By using the Python yfinance package, we acknowledged potential limitations in data accuracy and recommended cross-validation with alternative data sources or premium data providers for critical applications. Furthermore, to enhance trust in the application, rigorous testing and validation should be carried out to ensure the results align with industry standards.

Effective time management, self-discipline, and organization have been crucial to the successful development of the risk assessment tool as a single person project. Prioritizing tasks, setting clear milestones, and establishing achievable goals ensured that the project remained on track and met its objectives. Regular progress monitoring and re-evaluating priorities helped maintain focus and efficiently allocate time and effort.

A comprehensive quality assurance process was put in place to guarantee the quality and reliability of the tool. This included establishing coding standards, implementing test-driven development, carrying out regular code reviews, and conducting thorough testing at different levels. As a result, the risk of errors was minimized, maintainability was improved, and users can rely on a robust and reliable solution for their financial risk assessment needs.

Furthermore, the project emphasized the importance of continuous learning and improvement. As a solo developer, staying informed about the latest industry trends, methods, and technologies, and incorporating user feedback allowed the tool to be enhanced in terms of overall functionality and user experience. By fostering a culture of continuous learning and improvement, the project ensured that it remained relevant, valuable, and effective in addressing the needs of its users.

In conclusion, the development of the financial risk assessment tool represents a significant achievement in helping investors and financial professionals better understand and manage their risk exposure. By leveraging state-of-the-art methodologies, technologies, and best practices, the tool provides accurate, reliable, and actionable insights for risk management.

With a commitment to continuous improvement, the project can adapt and grow to meet the ever-changing needs of its users and the financial markets.

As the financial world continues to evolve, it is crucial that the risk assessment tool remains at the forefront of innovation, incorporating advances in data analysis, risk management, and financial modelling. Future research and development could explore the integration of machine learning and artificial intelligence techniques to enhance the tool's predictive capabilities and provide even more accurate risk assessments. By doing so, the risk assessment tool will continue to empower investors and financial professionals in making informed decisions about their portfolios and risk management strategies.

In summary, this report has provided a comprehensive overview of the development, implementation, and evaluation of a financial risk assessment tool using Python. The tool's success in estimating VaR and CVaR demonstrates its potential to contribute significantly to the field of financial risk management. By maintaining a commitment to best practices, quality assurance, and continuous improvement, the project can continue to serve the needs of its users.

Chapter 8: **Bibliography**

1. Agresti, A. (2002). Categorical data analysis (2nd ed.). Hoboken, NJ: Wiley.

2. Artzner, P., Pasteur, L., Strasbourg, F., Delbaen, T., Hochschule, Zürich, J.-M., Société Eber and Générale, P. (1996). Coherent Measures of Risk. https://people.math.ethz.ch/~delbaen/ftp/preprints/CoherentMF.pdf

3. Astels, D. (2003). Test-Driven Development: A Practical Guide. Prentice Hall Professional.

4. Bailey, D., Borwein, J., & Kapoor, V. (2014). Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-of-Sample Performance. Notices of the American Mathematical Society, 61(5), 458-471.

5. Bâle Committee on Banking Supervision. (1996). Supervisory framework for the use of "backtesting" in conjunction with the internal models approach to market risk capital requirements. https://www.bis.org/publ/bcbs22.pdf

6. Beck, K. (2003). Test-driven development: by example. Addison-Wesley Professional.

7. Bell, F., & Smyl, S. (2018). Forecasting at Uber: An Introduction. Uber Blog. https://www.uber.com/en-GB/blog/forecasting-introduction/

8. Berger, J. O., & Sellke, T. (1987). Testing a point null hypothesis: The irreconcilability of P values and evidence. Journal of the American Statistical Association, 82(397), 112–122.

9. Brown, M. (2018). The Binomial Test: Definition, Examples, and Results. Retrieved from https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/binomial-test-what-is-it/

10. Chen, J. (2019). Conditional Value at Risk (CVaR). Investopedia. https://www.investopedia.com/terms/c/conditional_value_at_risk.asp

11. Chen, J. (2021). Backtesting. Investopedia. https://www.investopedia.com/terms/b/backtesting.asp

12. Chollet, F. (2018). Deep Learning with Python. Shelter Island, NY: Manning Publications.

13. Docs.scipy.org. (n.d.). Scipy.stats.norm – SciPy v1.5.4 Reference Guide. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html

14. Dorota Huizinga and Kolawa, A. (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-Interscience.

15. Dowd, K., & Hutchinson, M. (2010). Measuring Market Risk (2nd ed.). Wiley.

16. Fewster, M., & Graham, D. (1999). Software Test Automation: Effective Use of Test Execution Tools. Addison-Wesley Professional.

17. Freeman, S., & Pryce, N. (2009). Growing object-oriented software, guided by tests. Pearson Education.

18. Giridhar, V. (2016). Test-Driven Development: A Practical Guide. Packt Publishing.

19. Hamill, P. (2004). Unit Test Frameworks: Tools for High-Quality Software Development. O'Reilly Media.

20. Howell, D. C. (2013). Statistical methods for psychology. Wadsworth Cengage Learning.

21. Hull, J. C. (2015). Options, Futures, and Other Derivatives (9th ed.). Harlow, England: Pearson Education Limited.

22. Investopedia. (2022). Value at Risk (VaR). https://www.investopedia.com/terms/v/var.asp

23. Jorion, P. (2007). Value at risk: The new benchmark for managing financial risk. McGraw-Hill.

24. Koskela, L. (2007). Test Driven: TDD and Acceptance TDD for Java Developers. Manning Publications.

25. Langr, J. (2015). Pragmatic Unit Testing in Java 8 with JUnit. Pragmatic Bookshelf.

26. Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley Professional.

27. Numpy.org. (n.d.). Numpy.mean – NumPy v1.22 Manual. https://numpy.org/doc/stable/reference/generated/numpy.mean.html

28. Numpy.org. (n.d.). Numpy.percentile – NumPy v1.21 Manual. https://numpy.org/doc/stable/reference/generated/numpy.percentile.html

29. Numpy.org. (n.d.). Numpy.std – NumPy v1.21 Manual. https://numpy.org/doc/stable/reference/generated/numpy.std.html

30. Pandas.pydata.org. (n.d.). Pandas.DataFrame.mean – pandas 1.3.1 documentation. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.mean.htm

31. Pandas.pydata.org. (n.d.). Pandas.DataFrame.pct_change – pandas 1.3.4 documentation. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pct_change.html

32. Pardo, R. (2008). The Evaluation and Optimization of Trading Strategies (2nd ed.). Wiley.

33. PyShark. (2021). Binomial Distribution and Binomial Test in Python. https://pyshark.com/binomial-distribution-and-binomial-test-in-python/

34. Pytest Documentation. (2022). Getting Started with pytest. Retrieved from https://docs.pytest.org/en/6.2.x/getting-started.html

35. ThoughtWorks. (2022). Test Driven Development. https://www.thoughtworks.com/insights/blog/test-driven-development

Wasserstein, R. L., & Lazar, N. A. (2016). The ASA statement on p-values: Context, process, and purpose. The American Statistician, 70(2), 129–133.

Chapter 9: **Appendix**

9.1 API Document

9.1.1 get_data_view endpoint

Endpoint: /get_data_view

Method: POST

Description: Receives user input for stock data and validates the data.

Request data:
JSON object containing the following data:

```
{
  "stock_list": "AAPL,MSFT",
  "portfolio_weights": "0.5,0.5",
  "period": "501",
  "Time": "1",
  "InitialInvestment": "10000",
  "confidence_level": "5"
}
```

Response: A boolean value indicating if the user input is valid.

9.1.2 graph_data endpoint

Endpoint: /graph_data

Method: POST

Description: Returns either the closing price data or return data for the portfolio stocks.

Request data:
type (string): The type of data requested, either "ClosingPrice" or "Return".
example :

```
{
        "type":"ClosingPrice"
}
```

Response: JSON object containing the requested data.
ClosingPrice respone:

```
{
  "AAPL": {
    "2023-03-17 04:00:00": 155.0,
    "2023-03-20 04:00:00": 157.3999938965,
    ...
  },
```

```
  "TSLA": {
    "2023-03-17 04:00:00": 180.1300048828,
    "2023-03-20 04:00:00": 183.25,
    ...
  }
}
```

Return respone:

```
{
  "AAPL": {
    "2023-03-17 04:00:00": -0.0054540011,
    "2023-03-20 04:00:00": 0.0154838316,
    ...
  },
  "TSLA": {
    "2023-03-17 04:00:00": 0.003440011,
    "2023-03-20 04:00:00": -0.0154356716,
    ...
  }
}
```

9.1.3 simulation endpoint

Endpoint: /simulation

Method: POST

Description: Runs the selected VaR simulation method and returns the calculated VaR.

Request data:

method (string): The method to use for the simulation. Options: "hs", "mb_1", "mb_2", "ms_1", "ms_2".
example :

```
{
        "method":"hs"
}
```

Response: JSON object containing the calculated VaR.
For method "mb_1", "mb_2" response is as follow :

```
{
        "hVaR": 267.56053408332025,
        "InitialInvestment": 10000.0
}
```

For method "hs", "ms_1", "ms_2" response is as follow :

```
{
        "hVaR": 267.56053408332025,
        "hVaR": 208.23633048603963,
        "InitialInvestment": 10000.0
```

```
}
```

9.1.4 option_data endpoint

Endpoint: /option_data

Method: POST

Description: Receives and validates user input for option data.

Request data: JSON object containing the following data:

```
{
  "confidence_level": "5",
  "expiration_date": "2023-12-12,2023-12-24",
  "number_of_options": "10,10",
  "option_type": "call,put",
  "portfolio_weights": "0.5,0.5",
  "risk_free_rate": "0.05",
  "stock_list": "AAPL,MSFT",
  "strike_price": "150,120"
}
```

Response: A boolean value indicating if the user input is valid.

9.1.5 option_var endpoint

Endpoint: /option_var

Method: POST

Description: Calculates the Option VaR using the specified method (either "hs" or "ms").

Request data:

method (string): The method to use for the Option VaR calculation. Options: "hs", "ms".
example:

```
{
        "method":"hs"
}
```

Response: JSON object containing the calculated Option VaR.

```
{
        "VaR": 267.56053408332025,
}
```

9.2 User manual

This user manual will guide you through using the Value-at-Risk (VaR) Web App and API to perform various financial calculations and retrieve historical stock data.

9.2.1 Installation

before you start make sure you have pip, python, npm and node install in your computer

First step make a new folder

```
mkdir value_at_risk
```

then go inside the value_at_risk folder

```
cd value_at_risk
```

then clone the repository , by using

```
git clone https://gitlab.cim.rhul.ac.uk/wjis203/PROJECT.git
```

!!!! if you want to install by step by step you can ignore this part , otherwise you can install by
go inside the repository

```
cd PROJECT
```

then run install.sh

```
sh install.sh
```

to turn on the server

```
sh start_server.sh
```

continue for the installation

Because we want to make make the repository as clean as possible we decided to create the virtual environments out side the repository

```
python -m venv venv
```

After we created the virtual environments , we need to activated it

for linux and mac user

```
. venv/bin/activate
```

for windows user

```
. venv/Scripts/activate
```

After we activated the environment we can go inside the repository

```
cd PROJECT
```

then update the pip and install the package

```
pip install --upgrade pip
pip install -r requirements.txt
```

After that we need to go inside the server folder

```
cd Server
```

After we install the Django package, we need to do the migration to create the database for the server

```
python manage.py makemigrations api && python manage.py makemigrations &&
python manage.py migrate
```

if you want to create a admin account you need to use "createsuperuser" and follow the instruction

```
python manage.py createsuperuser
```

After we setup the backend , we need to setup the frontend , so we go in to the frontend folder

```
cd frontend
```

then we install the package for the frontend

```
npm i
```

then we build the frontend , otherwise django cannot handle it

```
npm run build
```

After all the data and user was created we can start the server now , by using

```
python manage.py runserver 5000
```

then the server will run on local host port 3000

9.2.2 API Usage

Please read the API document

9.2.3 Web App Usage

This user manual provides detailed instructions on how to use the Financial Risk Assessment Web Application. By following the steps outlined in this manual, users will be able to navigate the application, input data, choose calculation methods, and analyze the results of Value at Risk (VaR) and Option VaR calculations.

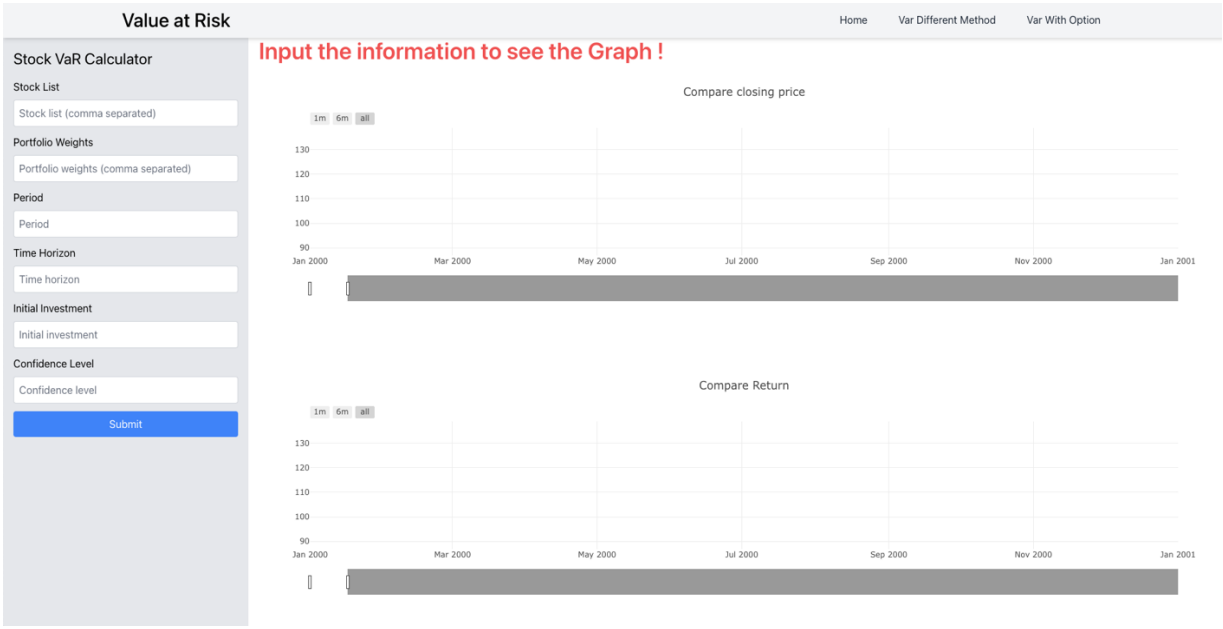
Getting Started

Before start make sure you turn on the django server.
To access the Financial Risk Assessment Web Application, open your preferred web browser and navigate to the application's URL (example:http://localhost:5000/). !

Navigating the Application

Upon entering the application, users will be presented with three main pages: Home Page, VarDifferentMethod Page, and VaRWithOption Page. Use the Navbar located at the top of the application to switch between these pages.

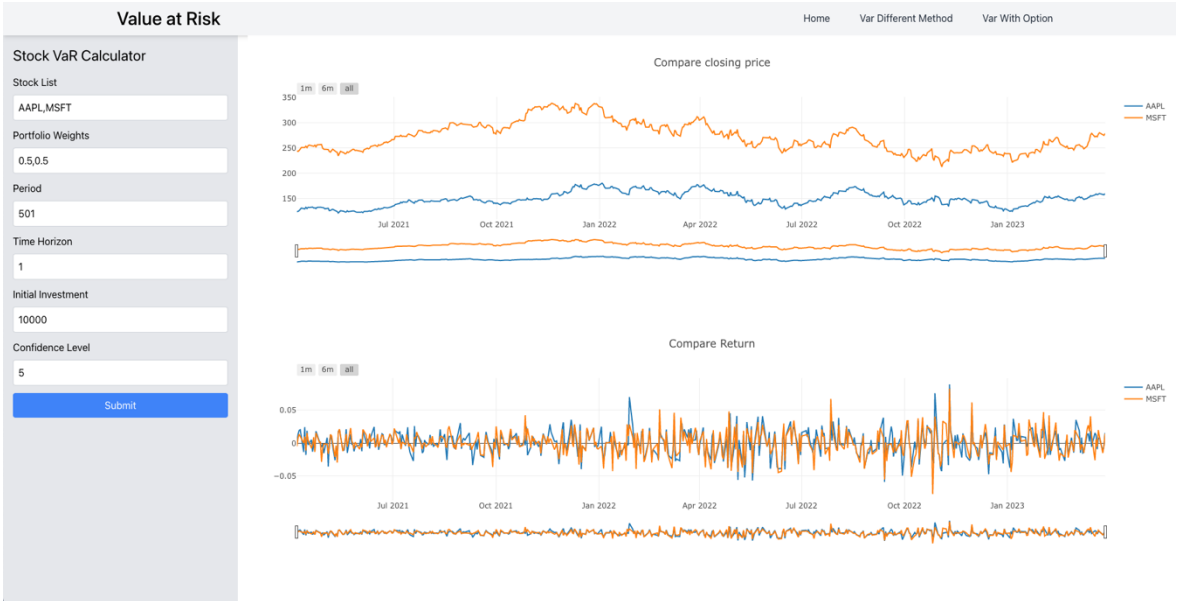
Home Page



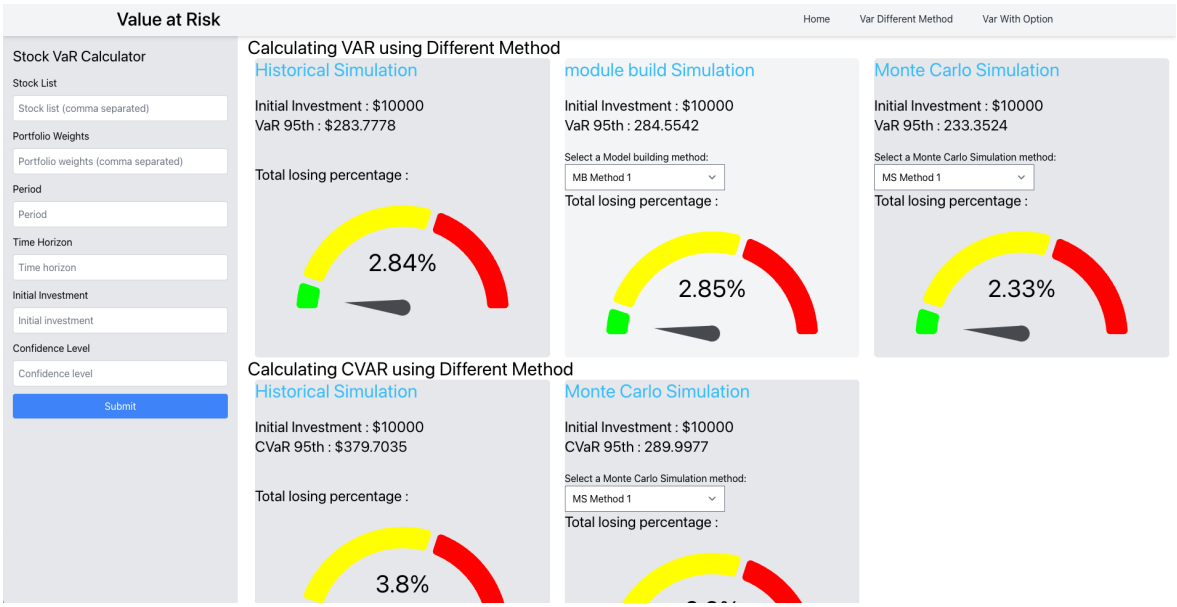
The Home Page displays 2 graphs, which include the returns and closing price charts, and a sidebar for user input related to stock VaR calculations. Enter the following input:

stock_list: AAPL,MSFT
portfolio_weights: 0.5,0.5
period: 501
Time Horizon: 1
Initial Investment: 10000

confidence_level: 5 Then, press the Submit button. After submission, the application will display the graphs with the calculated data:

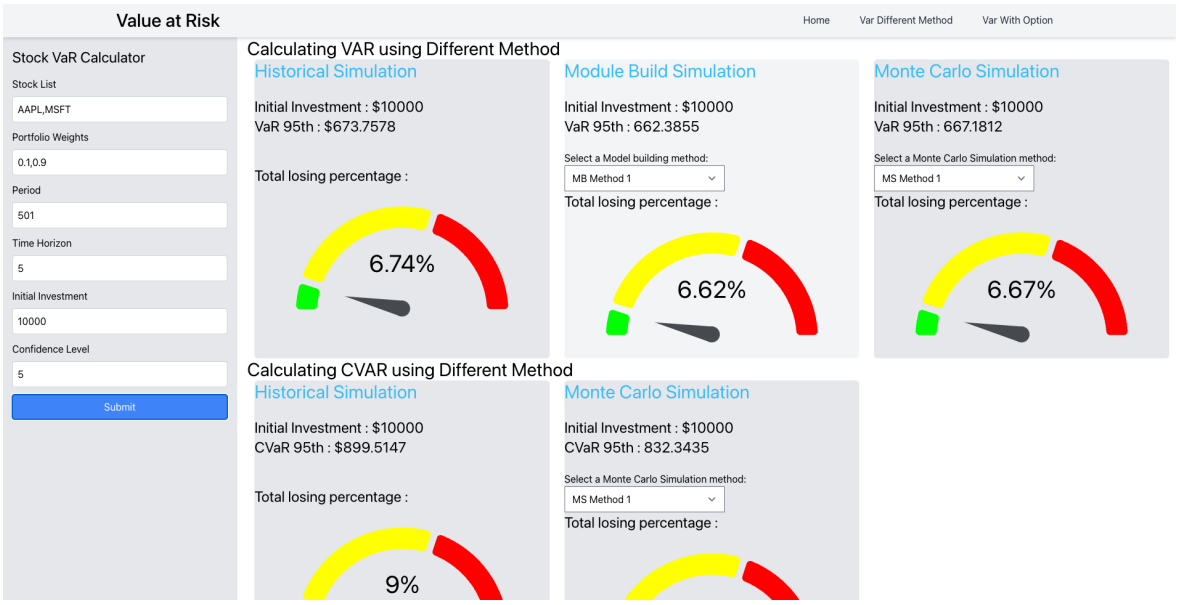


VarDifferentMethod Page

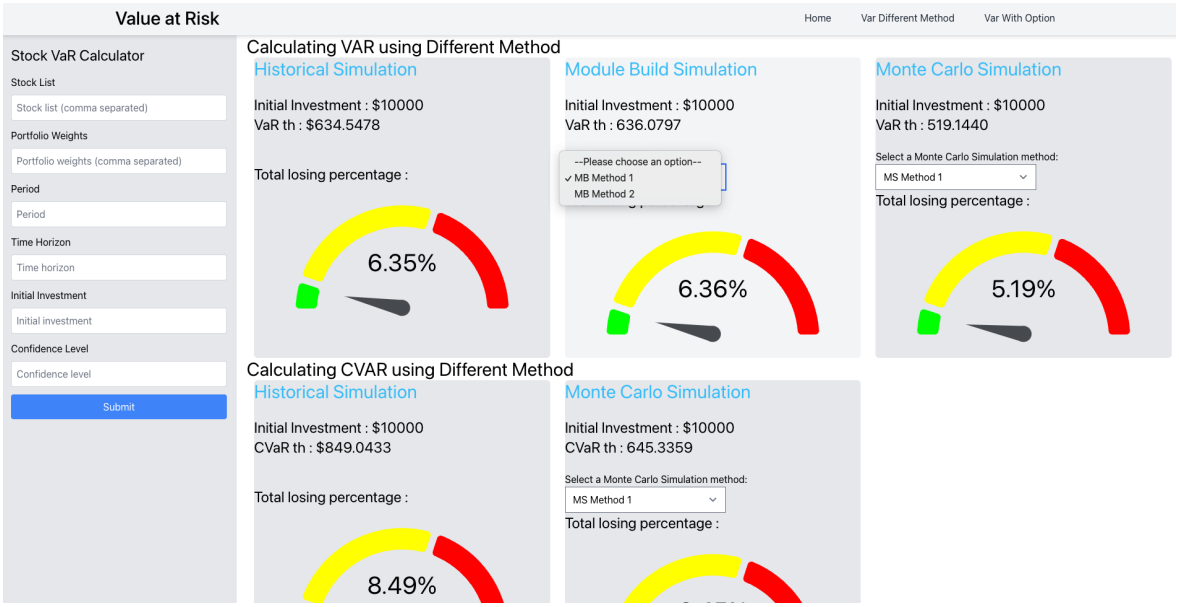


When the user goes to the VarDifferentMethod page, the application will automatically calculate the VaR based on the input provided on the Home page. The VarDifferentMethod Page also includes a sidebar for users to update their input provided on the Home page. To update the input, enter all the data again into the input boxes:

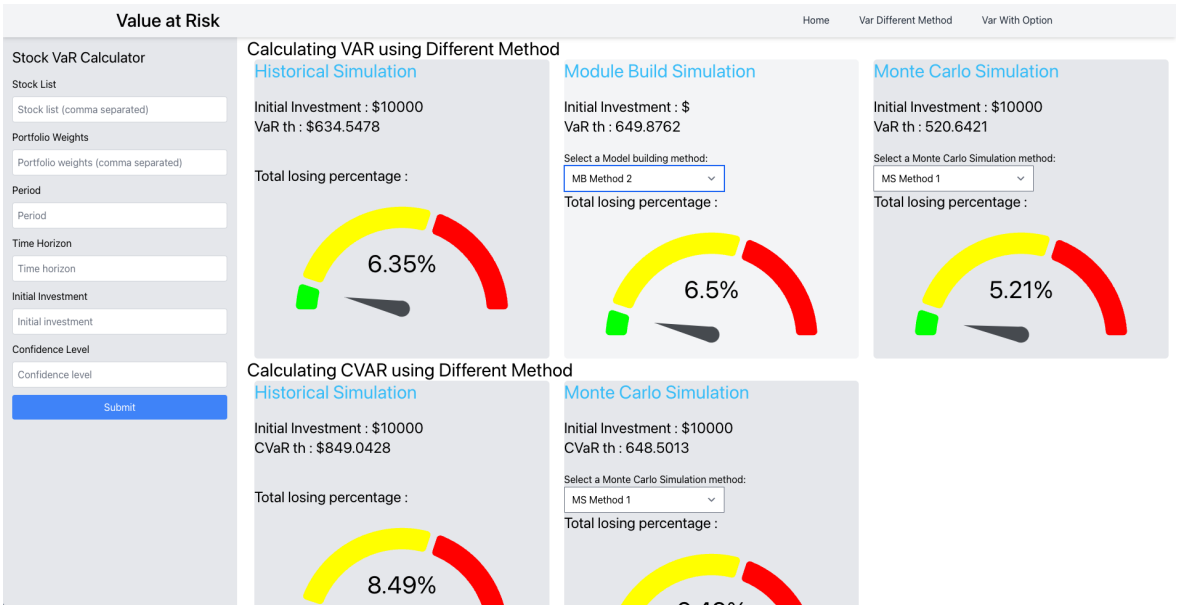
stock_list: AAPL,MSFT
portfolio_weights: 0.5,0.5
period: 501
Time Horizon: 1
Initial Investment: 10000
confidence_level: 5



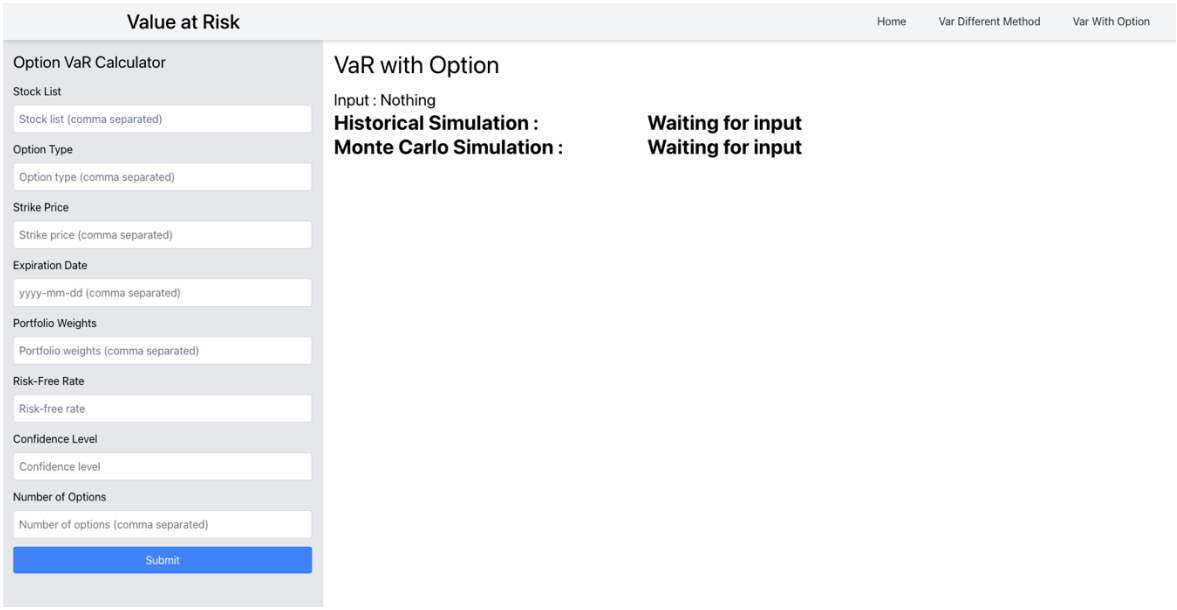
The application will then display the VaR and CVaR calculated by different methods. To calculate VaR using different methods, such as Historical Simulation and Monte Carlo Simulation, follow these steps:



1. Choose the desired VaR calculation methods by selecting the appropriate checkboxes.
2. The application will update the results accordingly.

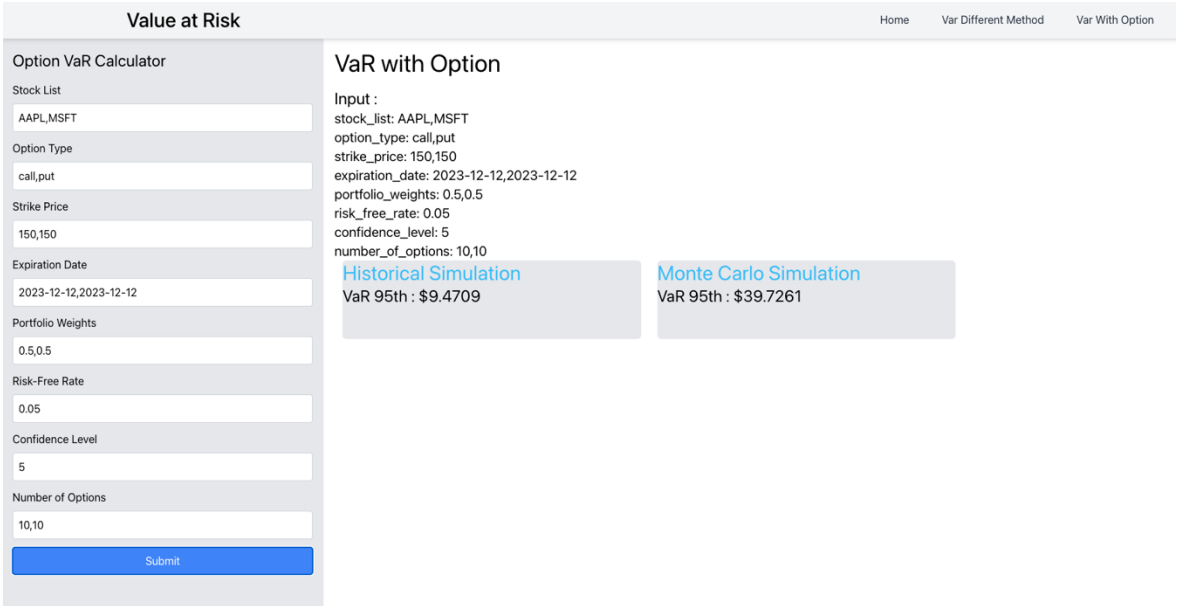


VaRWithOption Page



The VaRWithOption Page has a different sidebar for user input and presents the results for calculating Option VaR using different methods. To calculate Option VaR on the VaRWithOption Page, follow these steps: In the OptionVaRSideBar, enter the required information:

confidence_level: 5
expiration_date: 2023-12-12,2023-12-24
number_of_options: 10,10
option_type: call,put
portfolio_weights: 0.5,0.5
risk_free_rate: 0.05
stock_list: AAPL,MSFT
strike_price: 150,120 The application will then display the Option VaR results. Like this:



Additional Tips and Troubleshooting
To ensure an optimal user experience, consider the following tips and troubleshooting steps:

If the application is not displaying the expected results or is taking too long to load, refresh the web page and enter the input again.

Make sure to input accurate and complete information in the StockVaRSideBar and OptionVaRSideBar components. Incorrect or incomplete data may lead to inaccurate results or errors.

9.3 Youtube video link

https://youtu.be/ua2uV4cL2G8