

# Creating Software for Value at Risk

*Interim Report*

BSc Final Year Project

## **Author**

Benjamin Shearlock

## **Supervisor**

Dr. Volodya Vovk

Department of Computer Science  
Royal Holloway, University of London

## Declaration

I have read and understood the Universities regulations on plagiarism and I hereby declare that all work submitted in this project is my own, except where explicitly stated otherwise, such as the references that have been cited.

- Word Count: 0
- Name: Benjamin Charles Shearlock
- Submission Date: 08/12/2023

# Contents

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Chapter 1 - Introduction</b>	<b>5</b>
2.1	What is VaR? . . . . .	5
2.2	Aims and Goals . . . . .	6
2.3	Milestone Plan . . . . .	7
2.3.1	Timescale . . . . .	8
2.3.2	Timeline . . . . .	10
<b>3</b>	<b>Chapter 2 — Understanding VaR</b>	<b>12</b>
3.1	Historical Simulation . . . . .	12
3.2	Model Building (Variance-Covariance) . . . . .	13
3.3	Coding VaR Methods . . . . .	14
3.3.1	Command Line — Historical Simulation . . . . .	14
3.3.2	Command Line — Variance-Covariance Model Building . . . . .	15
3.4	Back-Testing . . . . .	17
3.5	Combining Implementations . . . . .	19
<b>4</b>	<b>Chapter 3 — Graphical User Interface</b>	<b>22</b>
4.1	What is Kivy? . . . . .	22
4.2	Conceptualisation of Initial Design Ideas . . . . .	23
4.3	Initial GUI Creation . . . . .	24
<b>5</b>	<b>Chapter 4 - Software Engineering</b>	<b>24</b>
5.1	Design Patterns . . . . .	24
5.2	Testing . . . . .	24
5.3	Documentation . . . . .	24
<b>6</b>	<b>Chapter 5 - Evaluation &amp; Scope</b>	<b>24</b>
6.1	Current Work . . . . .	24
6.2	Future Work . . . . .	24
<b>7</b>	<b>Bibliography</b>	<b>24</b>
<b>8</b>	<b>Literature Review</b>	<b>26</b>
<b>9</b>	<b>Appendix - Diary</b>	<b>26</b>

# 1 Abstract

In the realm of financial risk management, understanding and evaluating the level of risk associated with any investment or portfolio is of extremely high importance. Perhaps the most universally regarded metric used for this purpose is Value at Risk (VaR). VaR provides a quantitative estimate of the potential losses that a portfolio may incur over a certain period of time (specified time horizon) at a given confidence level.

Original widespread use of VaR came about in the early 1990s, the concept first being introduced by J.P. Morgan in 1994, since it helped provide an estimate of the maximum loss an investor is willing to accept for any given investment. Its historical roots can be traced back to the financial industry's increasing need for a standardized and comprehensive measure of risk following the 1987 stock market crash, so they sought a comprehensive way to assess risk in complex portfolios [3]. Mathematically, VaR is expressed as follows:

$$VaR(N, X) = -\text{Percentile}(L, 1 - X) \quad (1)$$

Where:

$N$  : Time horizon (in days)

$X$  : Confidence level (in percentage)

$L$  : Loss distribution over  $N$  days

This formula captures the loss at the  $(100 - X)$ th percentile of the loss distribution over the specified time horizon [6].

VaR can be mathematically computed through various methods, each with its own strengths and limitations. The most common approaches include the historical simulation method, parametric method, model building method and Monte Carlo simulation [1], to list a few. For historical simulation, past data is used to estimate future risk by examining the historical returns of an asset or portfolio, while model building employs mathematical models to predict portfolio performance. The choice of algorithm depends on data availability, computational resources, and specific requirements.

To implement VaR calculations, various pieces of software are essential. In this project, I will be utilising Visual Studio Code (VSCode) as the integrated development environment (IDE) and Python for its rich ecosystem of libraries. Python libraries like NumPy, Matplotlib, and Kivy will be invaluable for data manipulation, visualisation, and user interface design [9].

My inaugural proof of concept program will be created to visually demonstrate VaR calculations for two initial methods, these being historical simulation and model building techniques to estimate VaR for a sample portfolio. Historical simulation would involve collecting historical data and computing VaR based on past performance (can involve acquiring stock data through an API), while model building would use a predefined model to forecast future losses. Visualisation tools like Matplotlib can help in presenting the results graphically, enabling me to generate informative charts and graphs. NumPy will facilitate data manipulation and efficient mathematical operations [10]. Additionally, Kivy, a Python framework for developing multi-touch applications, will be used to create the interfaces for the visual representation of VaR, as well as giving it the option to possibly be viewed on other devices.

Later on into the project, if there is enough time, I think it may be worth exploring some more advanced topics like the variance-covariance of returns, specifically employing GARCH (Generalized Autoregressive Conditional Heteroscedasticity) models. GARCH models provide a more nuanced understanding of volatility and can enhance the accuracy of VaR estimates [6].

The objective of my project is to gain a deeper understanding of VaR, develop a functional program to calculate and visualise it, and potentially extend the research to incorporate more advanced risk management techniques if possible. This project is a stepping stone towards a deeper understanding of the risk side of finances and will contribute to enhancing the knowledge and skills necessary for effective financial decision-making, since this is not a topic that I have delved much into before, but I have always been very interested in learning more about it. This gives me a fantastic opportunity to learn about the financial sector, as well as create something that is applicable and useful to real life.

## 2 Chapter 1 - Introduction

### 2.1 What is VaR?

Value at Risk (VaR) is a statistical value used to quantify the level of financial risk within a portfolio of stocks/derivatives over a specified time horizon. It estimates the maximum potential loss that an investment portfolio could incur with a given probability, known as the confidence level, under normal market conditions. The VaR metric is typically expressed in monetary terms and is often used to assess the risk of a portfolio of financial instruments. From a banks perspective, they would say:

“With **X% confidence (1-X% Risk Level)**, we expect to not lose more than **V** over the next **N day(s)** on our entire **portfolio** of derivatives, worth **Z**”

Where:

**X** : Confidence Level

**V** : Value at Risk

**N** : Time Horizon

**Z** : Portfolio Value

This is incredibly useful as it allows for a simple and easy to understand metric to be used to assess the risk of a portfolio, which is extremely important in the financial sector, taking all the underlying complexities of any form of portfolio and being able to conform its financial risk into a single, universally used, number.

To include actual numbers, to easier explain how it would look, a bank would say:

“With **95% confidence (5% Risk Level)**, we expect to not lose more than **£3,000,000** over the next **1 day** on our entire portfolio of derivatives, worth **£100,000,000**”

This is essential to help financial institutions understand the level of risk associated with their portfolios, as well as being able to compare the risk of different portfolios, which is why it is such a widely used metric in the financial sector, limiting the level of risk that a person/organisation is exposed to.

The concept of VaR has its roots in the late 20th century, gaining prominence in the finance industry during the 1990s. It emerged from the need for more sophisticated risk management tools in the wake of financial market liberalisation and the increasing complexity of financial instruments. The widespread adoption of VaR was catalysed by the 1998 financial crisis, where it played a significant role in risk assessment and regulatory frameworks.

VaR is significant for several reasons:

- **Risk Measurement:** VaR provides a quantitative measure of potential losses in a portfolio.
- **Decision Making:** VaR aids financial managers in making informed decisions about risk tolerance and capital allocation.
- **Market Risk Management:** VaR is used to monitor and mitigate market risks, contributing to the overall stability of financial systems.

VaR has become a cornerstone in risk management for global financial markets. It is used by banks, investment firms, asset managers, and corporates to measure and control the level of risk exposure in their financial portfolios. VaR's adoption is partly driven by regulatory requirements, such as Basel Accords, which mandate financial institutions to calculate and maintain adequate capital reserves based on their risk exposure. NEED A BASEL ACCORDS REFERENCE??

Developing software for VaR calculation offers numerous advantages:

- **Accessibility:** It democratizes the access to sophisticated risk management tools.
- **Efficiency:** Automated VaR calculations save time and reduce the potential for human error.
- **Customization:** Software can be tailored to specific needs and types of portfolios.
- **Real-Time Analysis:** It enables rapid and up-to-date risk assessments.

VaR has become an essential tool in modern financial risk management. The development of software for VaR calculations aligns with the need for efficient, accurate, and accessible risk management tools in today's fast-paced financial markets. This is what I want to help contribute to within this project.

## 2.2 Aims and Goals

My aims and goals for this project are as follows:

1. **Comprehensive Understanding of VaR:** To touch on VaR's theoretical underpinnings and why it has such a pivotal role in modern financial risk management. This includes commenting on its applications across different financial groups, market conditions, and regulatory environments.
2. **Methodological Exploration:** To investigate various computational methods for calculating VaR, including historical simulation, model building (variance-covariance) approach, and Monte Carlo Simulation, assessing their efficacy in different market scenarios. This will first be explored through command line programs, then later on through a GUI.
3. **Technical Implementation:** Development of a comprehensive program for calculating and presenting VaR. This entails creating a user-friendly interface, ensuring accurate and efficient computation, and integrating various methods of VaR calculation to provide a comprehensive tool.
4. **Application in Diverse Financial Contexts:** To ensure the software's adaptability and applicability in different financial settings. This includes testing the software with various data sets, portfolio types, and market conditions, aiming to make it a versatile tool for different financial entities.
5. **Industry Standard Tool Development:** Creating a software application that not only performs standard VaR calculations but also offers other stock-oriented features, such as advanced data visualization. The goal is to make the tool align with industry standards, ensuring it is suitable for professional financial risk management and has been developed whilst adhering to industry principles in regard to software engineering.

The ambition of this thesis is to present a thorough understanding of VaR, culminating in the development of a robust, adaptable, and industry-relevant tool for financial risk assessment. It aims to be acceptable as an industry standard tool,

## 2.3 Milestone Plan

Due to unfortunate circumstances, I had rough delays to the start of this Project as well as inconsistent health concerns, but that should not effect my overall final deliverable. In the first term, I want to research and create a working program to compute Value at Risk for small portfolios, that has a serviceable GUI that can be expanded on later. I will also make sure to have amply researched about back-testing and how to incorporate it into my program in some capacity. For the second term, I will research and implement applying Value at Risk for a portfolio of derivatives, as well as looking into using the Monte Carlo simulation and allowing for the computation of all this with as many stocks as necessary. I will finalise the GUI and plan to look into completing some of the extensions provided for the project, this will depend on the overall developmental scope of the project at the time, but these are the options I would be willing to explore:

- **Computing Conditional VaR (Expected Shortfall):** Beyond the standard VaR metric, exploring Conditional VaR could be considered to provide a more comprehensive risk assessment, as it accounts for the severity of losses in the tail of the distribution.
- **Parameter Selection for EWMA and GARCH(1,1) Models:** A detailed analysis of parameter selection for the Exponentially Weighted Moving Average and GARCH(1,1) models could be explored. The choice of parameters greatly influences model performance, and overall it would enhance the robustness of the risk assessment.
- **Empirical Study of Approaches to Historical Simulation for n-day VaR:** A comparative empirical study could be undertaken to examine different approaches to extending historical simulation from 1-day to n-day VaR, which can provide deeper insights into the performance of the methods.
- **Complementing Back-Testing by Stress Testing:** The robustness of the VaR model could be assessed not only through back-testing but also by applying stress testing techniques, allowing for the evaluation of the model's performance under extreme market conditions.
- **Computing Monetary Measures of Risk Different from VaR:** Other monetary risk metrics, which could complement or provide alternatives to VaR, such as Tail Value at Risk, could be considered to present a more complete picture of the risk landscape and enhance the risk management process.



The project's milestones have been structured to ensure a systematic and efficient approach to the development of the Value at Risk software. This plan is divided into distinct phases, each with specific objectives and deliverables.

1. **Initial Research and Planning (Completed):** This phase involved a thorough investigation into the concept of Value at Risk, its calculation methods, and the requirements for the software development.
2. **Software Design and Development (Ongoing):** Currently, the focus is on designing the software and developing the core functionalities of the VaR calculation tool. This includes building the initial graphical user interface and implementing various VaR models into it.
3. **Expanded GUI, Testing and Refinement:** After the development phase, there will be a complete redesign/refinement of the Graphical Interface, followed by rigorous testing that will be conducted to ensure the accuracy and reliability of the software. This phase will also involve refining the user interface and the overall user experience.
4. **Final Evaluation and Documentation:** The final phase involves a comprehensive evaluation of the software against the project's objectives and preparing detailed documentation as well as informational videos.

### 2.3.1 Timescale

The entire project is structured over two academic terms, allowing ample time for each phase while ensuring a steady progression towards the project's completion. This timescale is designed to balance the initial development and subsequent refinement and testing phases effectively, ensuring that each aspect of the software is given due attention.

Concurrent with the development, ongoing documentation is a critical aspect of this project. Documenting the process as it unfolds serves multiple purposes: it ensures a clear record of the development process and aids in identifying and resolving issues more efficiently. This approach to documentation not only enhances the quality and maintainability of the software but also ensures that the project's progress is well-documented and aligns with the overall objectives and milestones.



### 2.3.2 Timeline

Weeks 1–3 .....	<b>Project Research</b> <ul style="list-style-type: none"><li>• Research the fundamentals of Value at Risk (VaR)</li><li>• Research best coding language to use (Python)</li><li>• Familiarize myself with LaTeX and prepare IDE &amp; Git for Project specified use</li></ul>
Week 4 .....	<b>Finalize Plan and Start Coding</b> <ul style="list-style-type: none"><li>• Complete Project Plan</li><li>• Continue researching VaR and Python</li><li>• Begin project coding</li></ul>
Week 5–7 .....	<b>Coding and Data Preparation</b> <ul style="list-style-type: none"><li>• Continue to work on the VaR program (No GUI)</li><li>• Start collecting and organizing sample data for small portfolios so it can be used by the program</li><li>• Finalizing understanding of the two computational methods needed, this being model-building and historical simulation</li></ul>
Week 8 .....	<b>Back-Testing Research &amp; Implementation</b> <ul style="list-style-type: none"><li>• Investigate methods and techniques for VaR back-testing</li><li>• Start integrating back-testing into the project</li></ul>
Week 9–10 .....	<b>GUI Development</b> <ul style="list-style-type: none"><li>• Initiate the development of the GUI</li><li>• Ensure the GUI is robust for its current task as well as expandable for future enhancements</li></ul>
Week 11 .....	<b>Interim Report and Presentation Preparation</b> <ul style="list-style-type: none"><li>• Fine-tune programs and report so they are at a satisfactory level, will also allow for easier preparation for the interim presentation</li><li>• Prepare for the interim presentation</li></ul>

Weeks 1–2 .....	<b>Reflection and Research</b> <ul style="list-style-type: none"> <li>• Spend time to reflect on the progress of the project so far, make any changes that I think are warranted after having the winter break time to think about</li> <li>• Research the Monte Carlo simulation method for VaR, as well as how I could start implementing derivatives as portfolios into the project</li> </ul>
Week 3–4 .....	<b>Start Implementing New Features</b> <ul style="list-style-type: none"> <li>• Start implementing the Monte Carlo simulation method and continue derivative implementation</li> <li>• Start researching Eigen &amp; Cholesky decomposition to allow for however many stocks are needed within a portfolio</li> </ul>
Week 5–7 .....	<b>GUI Finalisation</b> <ul style="list-style-type: none"> <li>• Decide on the final visual product I want to represent with the GUI and start implementing it (if progress on this needs to continue into the next period, then it will be done so)</li> <li>• Set the program up to work portably/allowing it to work on mobile OS's as well as different desktop OS's</li> </ul>
Week 8–9 .....	<b>Extend Project Scope (if time permits)</b> <ul style="list-style-type: none"> <li>• Explore additional features or enhancements for the project, possibly decided upon at the start of Term 2</li> <li>• Implement as many as can be appropriately managed, with all additional time spent within this period being used to ensure the project is at its most refined state</li> </ul>
Week 10–11 .....	<b>Perfect Final Report</b> <ul style="list-style-type: none"> <li>• Make sure the program has been achieved to the best of its ability</li> <li>• Finalise and perfect the final report</li> </ul>

## 3 Chapter 2 — Understanding VaR

### 3.1 Historical Simulation

Historical Simulation is a method of estimating Value at Risk (VaR). It relies on historical market data to predict future risks, making it a straightforward yet powerful method for calculating VaR. The process of computing VaR through Historical Simulation involves several steps, as outlined below:

1. **Data Collection:** Historical price data of the asset is collected for a specified time period. This can be done using an API or through a CSV file.
2. **Calculate Percentage Differences:** The daily returns are calculated, this being done by comparing the closing price of the current day to the closing price of the previous day by dividing one by the other, the taking away 1. This is done for each day in the data set.
3. **Sort Returns:** The calculated returns are sorted in ascending order.
4. **Determine the VaR Threshold:** A percentile is chosen based on the confidence level (e.g. 95%). This is used to work out  $100\% - X$  percentile (e.g. 5th), this being the risk level of the portfolio and  $X$  being the confidence level. This is the VaR threshold.
5. **VaR Estimation:** The Value at Risk is estimated as the value at the chosen percentile. For example, if you have 500 days of data, the 5th percentile would be the 25th value in the sorted list of returns, then multiplied by the portfolio value, simulating the worst percentage decrease that your portfolio could encounter based on the last  $Z$  days of data.
6. **Correct Negative Number:** Since it takes the 5th percentile, it will be a negative percentage difference multiplied by the portfolio, since we need to represent VaR as a positive value of potential loss, it is multiplied by  $-1$  to make it positive.

As you can see, it takes a very literal approach, assuming that, since you've got all this historical data, then it is likely that the future will be similar to the data of the past. It is an empirical method, meaning it is based on historical data, and as such is a simple and easy to understand, but it does have its limitations, such as the fact that it does not take into account any changes in the market, such as a financial crisis, which could have a huge impact on the market.

Refer to Figure 1, it expresses that:

- The bell-shaped curve represents the probability distribution of gains and losses for the asset or portfolio over the specified period.
- The left tail of the distribution marks the area of losses, where we can observe the VaR loss at a specific confidence level, say  $(100 - X)\%$ . This tail area signifies the worst losses incurred in the historical period.
- The point where the left tail cuts the horizontal axis (gain/loss over  $N$  days) corresponds to the VaR at the chosen confidence level. For example, if  $X$  is 95, then the VaR would represent the maximum expected loss over the  $N$  days period that only 5% of the time is expected to be exceeded.

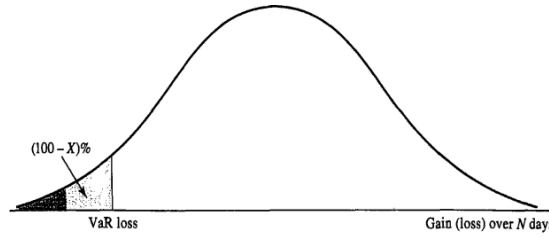


Figure 1: The Value at Risk (VaR) is calculated from the probability distribution of the variations in the portfolio's value, where gains are represented as positive values and losses as negative. The confidence level for this calculation is set at X%. [4]

- The graph assumes that the past can be a predictor of future risk, thus the frequency and magnitude of historical losses are directly used to estimate the VaR.

In the future, I could also use extreme value theory (EVT) to smooth out the distribution, which would allow for a more accurate estimation of the VaR, as it would allow for a better estimation of the tail of the distribution, which is where the worst losses are. [6]

### 3.2 Model Building (Variance-Covariance)

The Variance-Covariance method, used in this analysis for Value at Risk (VaR) calculation, is based on the assumption of normally distributed market returns. The key steps in the model are as follows:

1. **Data Collection:** Historical price data of the asset is collected for a specified time period once again. This can be done using an API or through a CSV file.
2. **Calculate Daily Returns:** Daily returns are computed as the percentage change in the adjusted closing prices of the stock.
3. **Statistical Analysis:** The mean and standard deviation of the daily returns are calculated to represent the average return and the volatility of the stock, respectively.
4. **VaR Calculation:** The Value at Risk is calculated using the formula:

$$VaR = -P \times (\text{norm.ppf}(rl, \text{mean}, \text{sD}) + 1) \quad (2)$$

where:

- $P$  represents the total value of the portfolio.
- `norm.ppf` is a function from the `scipy.stats` library that calculates the inverse of the cumulative distribution function (CDF) of the normal distribution, also known as the Percent Point Function (PPF).
- $rl$  is the risk level, which corresponds to the confidence level for VaR. For example, a 95% confidence level would use an  $rl$  of 0.05 (5% risk level).
- `mean` and `sD` are the mean and standard deviation of the historical returns, respectively.

The Value at Risk (VaR) calculation in the Variance-Covariance method is executed using a specific equation that combines statistical measures with the portfolio value to estimate the potential loss over a specified time horizon. The equation is as follows:

1. The `norm.ppf` function takes the risk level  $rl$ , mean, and standard deviation of returns as inputs and outputs a Z-score. This score corresponds to the point on the normal distribution curve where the cumulative probability is equal to the risk level.
2. The equation then multiplies this score with the portfolio value  $P$  and subtracts from  $P$ . This represents the potential loss in value of the portfolio at the given confidence level.
3. The addition of 1 in the formula adjusts for the fact that the `norm.ppf` function returns a negative value for typical VaR confidence levels. This is because the function calculates the Z-score for the left tail of the distribution, while VaR is the loss at the right tail. The addition of 1 converts the negative value to a positive one.

Thus, the equation calculates the VaR as the maximum expected loss over a certain period, given normal market conditions and a specified confidence level. The Variance-Covariance method is a simple yet effective approach to VaR calculation, but it does have its limitations, such as the fact that it assumes that the returns are normally distributed, which is not always the case.

### 3.3 Coding VaR Methods

To validate that I could apply the knowledge I had gained from my research, I decided to code the two methods of VaR calculation that I had researched, this being the historical simulation and model building methods. I have decided to use Python (3.11.1) as my coding language, as it is a language that I am familiar with and has a rich ecosystem of libraries that I can use to help me with the project. I also decided to use command line to display the results, as it utilises python's inbuilt `print()` command to easily display variables, allowing for a quick and easy way to display the results of the VaR calculations, as well as any tests run as well.

#### 3.3.1 Command Line — Historical Simulation

I decided to get the stock data I wanted to use for this directly from Yahoo Finance's online website, since it would allow me to specify the days I wanted to use, as well as the stock I wanted to use. I decided to use Nike (NKE) as my stock, as it is a company that I am familiar with and I know has been around for a long time, so it would have a lot of historical data to use. I decided to use the last 500 days of data, as it would allow for a good amount of historical stock data coverage, but not too much that it would take a long time to run.

To start, I imported the libraries I would need, this being `numpy` and `csv`. I then created an empty array called `closes`, which would be used to store the closing prices of the stock. I then opened the CSV file that I had downloaded from Yahoo Finance, and used a `for` loop to iterate through each row of the CSV file, adding the closing price of the stock to the `closes` array. `diffs` array, this being stored in the 6th column of the CSV file.

```

closes = np.array([])
with open('NKE.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        closes = np.append(closes, float(row[5]))

```

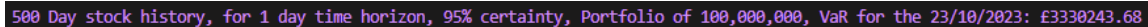
I then created an empty array called `diffs`, which would be used to store the percentage differences of the closing prices of the stock, followed by a `for` loop to iterate through each element of the `closes` array, calculating said percentage difference between the current element and the previous element, then adding it to the `diffs` array. Finally, I used the `np.percentile()` function to calculate the 5th percentile of the `diffs` array, which would be the VaR for the portfolio (this meaning I didn't need to manually order it to find the 25th worst case result, I can just use this useful function). I multiplied this by the portfolio value, which in this case was 100,000,000, to get the VaR for the portfolio and then multiplied this by -1, as VaR is always represented as a positive value, to get the final VaR value.

```

diffs = np.array([])
for i in range(1, len(closes)):
    diffs = np.append(diffs, (closes[i]/closes[i-1] - 1))
print("500 Day stock history, for 1 day time horizon, 95% certainty,
      Portfolio of 100,000,000, VaR for the 23/10/2023: £" +
      str(round(np.percentile(diffs, 5)*100000000*-1, 3)))

```

This results in this output, which appears to be a reasonable VaR result.



```

500 Day stock history, for 1 day time horizon, 95% certainty, Portfolio of 100,000,000, VaR for the 23/10/2023: £3330243.68

```

Figure 2: Historical Simulation VaR Result

For this project, I did not look up any online VaR results for this stock or any other further stock used, since I believe the methods were robust enough to be reliable. Since I also create multiple methods for calculating VaR, I was able to cross-reference them with each other, with them mostly showing similar results for the same stock, so I am confident that these are accurate VaR predictions.

### 3.3.2 Command Line — Variance-Covariance Model Building

To begin with this method, I discovered that I could import the historical stock price data conveniently by using the `yfinance` package, which provides a convenient way to download financial market data from Yahoo Finance directly into Python. I selected Nike (NKE) as the target stock so I could compare it to the previous result given by Historical Simulation. A time span of the last 500 trading days was once again chosen to balance between a sufficient data sample size and computational efficiency, as well as for comparative reasons.



The first step was to import the necessary libraries and download the stock data using the `yf.download()` function, one of these additional libraries being `scipy`, which is used for scientific and technical computing, due to it allowing the user to manipulate and visualize data with a wide range of high-level commands, one of such being the `norm.ppf()` function, which is used to calculate the inverse of the cumulative distribution function (CDF) of the normal distribution, also known as the Percent Point Function (PPF) that I will need to utilise in a future step. I then discovered that you can use the inbuilt `pct_change()` function to calculate the daily returns/percentage changes for the whole data set, which is a much more efficient way of doing it than the method I used for Historical Simulation. I also used the `dropna()` function to remove any missing values from the data set, in case using the API rather than the CSV could cause this issue.

```
from scipy.stats import norm
import yfinance as yf

stock = yf.download('NKE', dt.datetime(2021, 10, 26), dt.datetime(2023, 10, 24))
closeDiffs = stock['Adj Close'].pct_change()
```

I then proceeded to define the necessary variables to calculate the VaR, following the formula outlined in Equation 2. The portfolio and risk level were set to 100,000,000 and 0.05 (5%) respectively, since these were the values used in Historical Simulation, then the mean was calculated using the `np.mean` function, followed by the standard deviation using the `np.std` function.

```
portfolio = 100000000
rlPercent = 0.05
mean = np.mean(closeDiffs)
sD = np.std(closeDiffs)
```

Finally, calculating the VaR result takes place with the negative portfolio value multiplied by the `norm.ppf()` function, which takes the risk level, mean, and standard deviation of returns as inputs and outputs a Z-score, which corresponds to the point on the normal distribution curve where the cumulative probability is equal to the risk level, giving us the VaR estimation.

```
print("500 Day stock history, for 1 day time horizon, 95% certainty,
      Portfolio of 100,000,000, VaR for the 23/10/2023: £" +
      str(round(-portfolio*(norm.ppf(rlPercent, mean, sD)), 3)))
```

The output of this computation provides a VaR estimate under the assumption of normally distributed returns and a linear correlation between the assets. This assumption is of course a simplification and might not hold during periods of financial turmoil, which I acknowledge as a limitation of the model, but for this data it is acceptably accurate.

```
500 Day stock history, for 1 day time horizon, 95% certainty, Portfolio of 100,000,000, VaR for the 23/10/2023: £3640086.568
```

Figure 3: Variance-Covariance VaR Result

As you can see, the VaR result is very similar to the one calculated using Historical Simulation, which is a good sign that both methods are accurate. But how do I check the validity of both methods/models?

### 3.4 Back-Testing

So for the methods that I had already coded, I had achieved VaR estimations, but I had no way of validating how accurate they were, which is where back-testing comes in. Back-testing is a way of testing the accuracy of a model by using historical data to see how well it would have predicted the actual results. For example, if I had a model that predicted the weather, I could use back-testing to see how accurate it was by using historical weather data to see how well it has predicted the weather correctly in the past

To achieve this, you need to see how many times your model has previously predicted the correct VaR, or more so, how many times they have predicted the VaR wrong. To do this, let's say you have 500 days of historical data, and you want to see how many times your model has predicted the VaR wrong for a 1 day time horizon, 95% certainty, and a portfolio of 100,000,000. You would first need to take a portion of the data, let's say the first 50 days, and calculate the VaR for it using the model. You would then compare this VaR result to the actual value that the portfolio lost/gained from the 50th day to the 51st day, thus seeing if the model predicted the VaR correctly. This would then continue, keeping the 50 day data span, but moving it along by 1 day each time, so the next data span would be from the 2nd day to the 51st day, then the 3rd day to the 52nd day, and so on, until you reach the end of the data set. This can be seen in Figure 4 below.

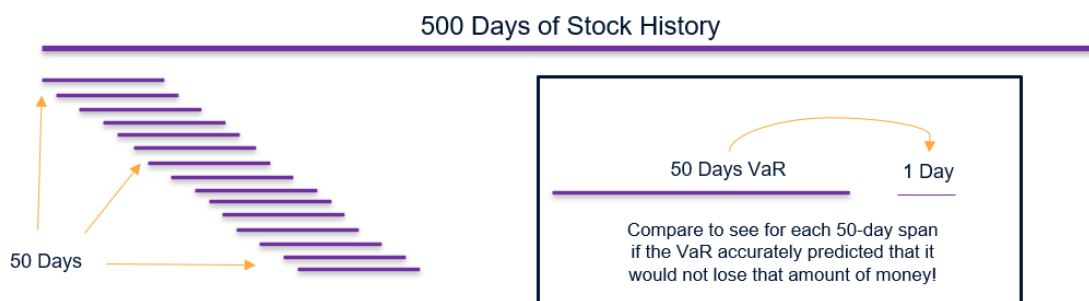


Figure 4: Back-Testing Diagram (Source: My Presentation Powerpoint)

Every time one of these spans creates a VaR that fails to accurately predict the next days loss, you would make note of it and add it to a counting total. This total can then be utilised with p-values. A p-value is a statistical measure that helps to determine the significance of the results obtained from a hypothesis test. In the context of back-testing, the null hypothesis typically states that the model's predictions are correct, and the alternative hypothesis suggests that the model's predictions are not accurate. A low p-value indicates that the observed data is unlikely under the null hypothesis, leading to its rejection in favour of the alternative hypothesis. So for our VaR back-testing, a low p-value would suggest that the model's performance is not as good as predicted, potentially failing to capture the risk adequately.

To do this within Python whilst leading on from our previous two coded methods, I added the following code to the end on both models. It creates a variable called `count`, which is used to count the number of times the model has predicted the VaR incorrectly, and a variable called `adjust`, which is used to adjust the data span to be used for the VaR calculation, this being 10% of the data set. It then creates a `for` loop, which iterates through each day of the data set, calculating the VaR for the data span, then comparing it to the actual value that the portfolio lost/gained from the last day of the data span to the next day, thus seeing if the model predicted the VaR correctly. This will repeat for the length of the entire historical data history, minus the span `adjust` and minus 1 as well, since you would need to compare 449–499 day VaR to day 449–500’s return. The next day return is also multiplied by -1, so it accounts so that the losses are positive like the VaR estimations, making it easy to compare if one is incorrectly larger than the other. If the VaR was predicted incorrectly, the `count` variable is increased by 1.

```
count = 0
adjust = int(len(stock)/10)
for i in range(1, len(stock) - adjust - 1):
    backTest = stock['Adj Close'].pct_change()[i:i+adjust]
    if Historical Simulation:
        VaR = np.percentile(backTest, rlPercent)*portfolio*-1
    else: #Model Building
        VaR = -portfolio*(norm.ppf(rlPercent, mean, sD))
    nextDay = stock['Adj Close'].pct_change()[i+adjust:i+adjust+1].values[0]
    nextDay = nextDay*portfolio*-1 #Done to fit within Latex Hbox
    if nextDay > VaR:
        count += 1
```

Next, I compute the p-value using the cumulative distribution function (CDF) of the binomial distribution, which in turn calculates the probability of observing a certain number of VaR exceedances (or fewer) given the expected number of exceedances under the model’s assumptions.

```
pValue = binom.cdf((len(stock)-adjust)-count, len(stock)-adjust, 1-rlPercent/100)
```

The parameters for the `binom.cdf` function are as follows:

- The number of successes, which is the total number of days minus the adjustment for the data window and the count of exceedances.
- The number of trials, which is the length of the stock data minus the adjustment for the data window.
- The probability of success on each trial, which is 1 minus the risk level percentage divided by 100, reflecting the confidence level of the VaR estimate.

The calculated p-value is then compared to the risk level percentage to determine the statistical significance. If the p-value is greater than the risk level percentage, the model passes the back-test, indicating that the number of VaR exceedances is within acceptable limits of the model’s predictions. Conversely, a p-value lower than the risk level percentage suggests that the model’s poor predictive performance is statistical significance, and it fails the back-test.[11]

```
Back Test: PASSED with 10.0% statistical significance level (p-value)
Back Test: PASSED with 19.0% statistical significance level (p-value)
```

Figure 5: Results using Tesco stock data, for Historical Simulation followed by Model Building.

```
if pValue > rlPercent/100:
    print("Back Test: PASSED with " + str(round(pValue*100, 0)) +
          "% statistical significance level (p-value)")
else:
    print("Back Test: FAILED with " + str(round(pValue*100, 0)) +
          "% statistical significance level (p-value)")
```

By utilizing p-values, we can assign a level of confidence to our back-testing results, supporting the validation process of our VaR model. It enables us to make informed decisions about the model's reliability and whether it can be trusted for making future risk assessments. It help provide a quantitative method to validate the accuracy of VaR models, ensuring that risk managers and financial analysts can rely on the model's predictions to make critical decisions. In the future, along a set of portfolio's with multiple stocks within, I could perform multiple back test on multiple stocks using both VaR, and form a statistical conclusion on how many time each model fails the p-value back-testing, thus seeing which model is more reliable and should possibly be used in further graphical builds, although I have not touched on Monet Carlo simulation yet, which could be far more accurate then either of these two methods, so I will have to see how my research into it goes.[11]

### 3.5 Combining Implementations

Before I decided to embark on the creation of any of my graphical elements, I thought it would be good to have some way to test my VaR models on different single stocks, so I could see a range of results, and better gauge the consistency levels of my back-tests, as well as also being able to simple see more VaR results then just what I had seen so far. I created a fully functioning command line program that allows the user to select a stock from the FTSE100 list (I chose this as I do not have a wide knowledge of stock lists, but since I knew of this one already and all the stocks on it are significant, it would be good to implement), enter a given portfolio value that they would have invested within this one particular stock, and then select a time horizon and confidence level for the VaR calculation. It would then give them 3 options for how much historical data they would want to use, either 100, 500 or they could input their own dates, and it would finally ask what model they would like to compute the VaR estimation with, either Historical Simulation or Model Building. It would compute the VaR and then display the result, as well as display the p-value back-testing result.

Since the code for this is just a large and robust amount of verification and input statements, I will not include it in this report, but I will display a full interaction in figure 6. I used pandas to retrieve the FTSE100 from Wikipedia, pandas being a powerful data manipulation library in Python, allowing for web scraping, reading and writing data, which I utilised it for in this case. Everything else included within the code has been covered within previous pages within this chapter. Full interaction found on the next page:

```

WELCOME TO THE VAR CALCULATOR
-----
Which companies stock would you like to calculate the VaR for?
1 - 3i
2 - Admiral Group
3 - Airtel Africa
*
***
*
40 - Haleon
41 - Halma plc
42 - Hargreaves Lansdown
43 - Hikma Pharmaceuticals
44 - Howdens Joinery
*
***
*
97 - Vodafone Group
98 - Weir Group
99 - Whitbread
100 - WPP plc
Enter the number of the company: 42
Enter the portfolio value (£): 100000000
Enter the risk level percentage (1%, 5%, etc.): 5
Enter the time horizon (Max: 100 days): 1
How many days of historical data would you like to use,
or would you like to choose your own date boundaries?
1. 100 days
2. 500 days
3. Choose your own
Enter the number of your choice: 2
[*****100%*****] 1 of 1 completed
Would you like to calculate VaR using Historical Simulation,
or using Model Building/Variance-Covariance (H/M): m

VaR is: £5,818,845.68
Back Test: PASSED with 32.0% statistical significance level (p-value)

```

Figure 6: Console output of the Single Stock VaR.py program.

I have neglected to mention how the Time Horizon is handled within this program. The best way to show how it is handled would be to quote from reference [6] — *Options, Futures, and Other Derivatives*. by John C. Hull, which states:

VaR has two parameters: the time horizon  $N$ , measured in days, and the confidence level  $X$ . In practice, analysts almost invariably set  $N = 1$  in the first instance. This is because there is not enough data to estimate directly the behavior of market variables over periods of time longer than 1 day. The usual assumption is:

$$N\text{-day VaR} = 1\text{-day VaR} \times \sqrt{N} \quad (3)$$

This formula is exactly true when the changes in the value of the portfolio on successive days have independent identical normal distributions with mean zero. In other cases, it is an approximation.

Since VaR is scaled with time due to volatility. Volatility tends to be proportional to the square root of time.

$$\text{Volatility}(\text{Time}) = \text{Volatility}(\text{One Period}) \times \sqrt{\text{Time Horizon}} \quad (4)$$

Since we are dealing with the other cases frequently throughout the use of the VaR Models used within the programs, I utilise the formula 3 above to accommodate for the VaR value over a given time period, by multiplying the VaR by the square root of the given time horizon, which is why the user is asked to input the time horizon in days, and not in years or months, this providing sufficient estimations so far.

## 4 Chapter 3 — Graphical User Interface

### 4.1 What is Kivy?

Kivy (<https://kivy.org/>) is an open-source Python library for developing multitouch application software with a natural user interface (NUI). It is highly versatile and can run on various operating systems, including Windows, macOS, Linux, Android, and iOS. This cross-platform compatibility is due to Kivy's use of OpenGL ES 2, allowing it to render consistent graphics across all supported platforms.[9] Kivy is also free to use, even for commercial purposes, as it is distributed under the MIT license.

Kivy's compatibility with multiple operating systems makes it an excellent choice for financial software development. The ability to write code once and deploy it on various platforms without modification is particularly advantageous in a financial context, where users may access software from different devices.

- **Cross-Platform:** Kivy apps can run on desktop and mobile devices, enhancing accessibility for users who need to monitor financial markets or run VaR calculations on the go.
- **Pythonic Nature:** Given that Python is a leading language in financial modelling due to its simplicity and the powerful data analysis libraries available, Kivy integrates well within the Python ecosystem, as well as it being the chosen language for this project.
- **Graphics Engine:** Kivy's graphics engine is built over OpenGL ES 2, providing the capability to handle intensive graphical representations, which is essential for visualizing complex financial data that I will want to do in the future.

Developing financial software with Kivy brings several benefits:

- **Rapid Development:** Kivy's straightforward syntax and powerful widgets allow for rapid development of prototypes and software, which is crucial in the fast-paced environment of financial markets.
- **Multitouch Support:** The library's inherent support for multitouch can be leveraged to create interactive financial dashboards, enhancing data exploration and manipulation for important tasks such as risk analysis which we will want to use it for.[9]
- **Customizable UI:** Kivy provides the tools to create a highly customizable user interface (UI), enabling the development of aesthetic financial applications tailored to the specific needs of financial analysts.
- **Community and Support:** Kivy has a growing community and good documentation, which will be highly useful for me as a beginner to the library.

Kivy presents itself as a robust framework for financial software development. Its compatibility with different operating systems, ease of operation within Python, and the capability to create sophisticated UIs make it an appealing choice for developers such as myself to use when creating financial applications, especially when catered for VaR analysis and other risk management tools, which is why I've chosen to use it for this project for the GUI.

The initial design phase of the Value at Risk (VaR) calculation tool will focus on creating a tool that will allow the user to interact with it in an efficient and effective manor, with less of an emphasis on a visually pleasing design, more on practicality for what has been researched and commented on so far. I had an idea in mind for what I wanted to initially be able to create, so I decided to create a design sketch (Figure 7), which reflects an early conceptualization of the interface, emphasizing simplicity and functionality.



23



A key feature highlighted in the sketch is the inclusion of the layered "current stock" display in the top right, as it lets the users easily see what they have selected, whilst also letting the scroll move underneath it, helping the program look and act dynamically. Additionally, a tick indicator was conceptualized to provide immediate visual feedback when back-testing succeeds or fails, based on statistical analysis (p-values), enhancing the user's understanding of the model's performance, showing evidence of system status. The design also incorporates radio buttons for VaR calculation modes for historical and model-based approaches, allowing for quick toggling between different methods whilst still maintaining the other pre-selected parameters.

This initial design was guided by principles of user centered design (UCD) best practices, aiming to streamline the complex process of risk analysis into a manageable and approachable workflow for end-users. Each element was chosen for its potential to make the software accessible to both novice and experienced users, allowing for ease of user regardless of skill level.

The conceptualisation phase was pivotal in laying the groundwork for the subsequent development of this GUI. It helped me visualise how it needed to look and how the interactions would have to take place, so even if the final result wasn't visually the same, it still provided an excellent user story for how it needed to perform.

### 4.3 Initial GUI Creation

Before my initial creation, I followed a lot of the book [9], *Kivy - Interactive Applications and Games in Python - Second Edition* by Roberto Ulloa, which I have referenced in my bibliography, as it was a very useful book for learning the basics of Kivy, and I would highly recommend it to anyone who wants to learn Kivy.

## 5 Chapter 4 - Software Engineering

### 5.1 Design Patterns

### 5.2 Testing

### 5.3 Documentation

## 6 Chapter 5 - Evaluation & Scope

### 6.1 Current Work

### 6.2 Future Work

## 7 Bibliography

1. Alexander, C. (2008). *\*Market Risk Analysis: Value-at-Risk Models.\** Hoboken, NJ: Wiley. [Online] Available at: <https://ebookcentral-proquest-com.ezproxy01.rhul.ac.uk/lib/rhul/reader.action?docID=416450>. *This book covers various aspects that I want to approach in this project, such as historical simulation, Monte Carlo simulation and various forms of testing that could be highly useful for my project.*

2. Arbuckle, D. (2017). \*Daniel Arbuckle's Mastering Python: Build Powerful Python Applications.\* Birmingham, England: Packt. [Online] Available at: <https://learning.oreilly.com/library/view/daniel-arbuckles-mastering/9781787283695/?ar=>.  
*I chose this reference as it helps with python packaging, being able to turn a python code and all its GUI and libraries into a single executable file, which is something I want to be able to do for this project.*
3. Choudhry, M. (2006). \*An Introduction to Value-at-Risk.\* Chichester: John Wiley & Sons Limited. [Online] Available at: <https://learning.oreilly.com/library/view/an-introduction-to/9780470017579/>.  
*Covers similar topics to the first reference, but seems to go into more detail on specific issues that I may need to address in this project.*
4. Duffie, D. and Pan, J. (2019). \*An Overview of Value at Risk.\* [Online] Available at: <http://web.mit.edu/people/junpan/ddjp.pdf>.  
*A much shorter reference, it is a paper that covers the basics of Value at Risk, which I will be using to help me understand the fundamentals of the topic since I have been able to follow it better than the other references.*
5. Föllmer, H. and Schied, A. (2016). \*Stochastic Finance: An Introduction in Discrete Time.\* Berlin: de Gruyter. [PDF]  
*A recommended reference from my supervisor, it is a book that covers higher level concepts relating to my project but also provides an overview of stochastic finance in general, which I have seen to be useful in understanding the topic.*
6. Hull, J.C. (2008). \*Options, Futures, and Other Derivatives.\* Upper Saddle River, NJ: Prentice Hall. [PDF]  
*This is my main reference for the project, as it is the main textbook suggested. It has a relevant chapter on Value at Risk, which explores many of the different aspects that I will need to look into for this project.*
7. Pritsker, M. (1997). "Evaluating Value at Risk Methodologies: Accuracy versus Computational Time." \*Journal of Financial Services Research\* 12: 201-242. [Online] Available at: <https://doi.org/10.1023/A:1007978820465>.  
*Another short reference, this is a paper that compares the accuracy and computational time of various Value at Risk methodologies, which I will be using to help me understand the different methodologies and how I can apply them to my project in the most efficient manner.*
8. Raman, K. (2015). \*Mastering Python Data Visualization: Generate Effective Results in a Variety of Visually Appealing Charts Using the Plotting Packages in Python.\* Birmingham: Packt Publishing Limited. [Online] Available at: <https://learning.oreilly.com/library/view/mastering-python-data/9781783988327/?ar=>.  
*This reference covers various aspects of data visualisation, which I will be using to help me understand how to best display important information in a visually appealing way for my project.*
9. Ulloa, R. (2015). \*Kivy - Interactive Applications and Games in Python - Second Edition.\* Packt Publishing. [Online] Available at: <https://learning.oreilly.com/library/view/kivy-interactive/9781785286926/?ar=>.  
*This reference is not being used for its content on game development, rather Kivy is a framework that can help create a GUI that works on both desktop operating systems as well as mobile operating systems, which I think I would like to explore the possibility of when developing the application*
10. Weiming, J.M. (2015). \*Mastering Python for Finance: Understand, Design, and Implement State-of-the-Art Mathematical and Statistical Applications Used in Finance with Python.\* Birmingham, England: Packt Publishing. [Online] Available at: <https://learning.oreilly.com/library/view/mastering-python-for/9781784394516/>.

*This reference covers various ways of handling financial data within python, which I will ensure to help me understand how to best handle the data I will be using within my project.*

11. Wasserstein, R.L., & Lazar, N.A. (2016). \*The ASA Statement on p-Values: Context, Process, and Purpose.\* The American Statistician, 70(2), 129-133. [Online] Available at:  
<https://www.tandfonline.com/doi/full/10.1080/00031305.2016.1154108>.

*This reference is crucial for understanding the nuanced interpretation and use of p-values in statistical hypothesis testing, since it is relevant for the back-testing aspect of my project involving the VaR model validation.*

## 8 Literature Review

## 9 Appendix - Diary