



UNIVERSITY OF LINCOLN

Portfolio Optimisation using Modern Portfolio Theory & Python

Graeme Hutchison

HUT14470469

Project Supervisor: Dr Bowei Chen

BSc(Hons) Computer Science

University of Lincoln, School of Computer Science

27th April 2017

Acknowledgements

I would like to thank my project supervisor Dr Bowei Chen for his support and guidance throughout this project as well as all my friends who supported me.

Abstract

This project focuses on the implementation of a portfolio optimisation system in Python that makes use of concepts such as Markowitz's mean-variance model found within Modern Portfolio Theory, in an attempt to generate effective trading strategies. Relevant research into existing work in this area is done to evaluate and assess the effectiveness of features within Modern Portfolio Theory. Full details on the design, implementation and testing of the artefact are thoroughly explained throughout this report along with evaluation of the artefact's effectiveness at achieving the project aim.

Contents

1. Introduction.....	6
1.1 Background	6
1.2 Aims & Objectives.....	7
Aim & Rationale.....	7
Objectives	7
2. Literature Review	8
2.1 Modern Portfolio Theory	8
Background	8
The Markowitz Model.....	8
Model Inputs.....	11
Diversification	12
Portfolio Performance Evaluation.....	13
3. Methodology	14
3.1 Project Management	15
Time Planning.....	15
Risk Management	15
Development Tools	15
4. Software Development.....	20
Requirements	20
4.2 Retrieving and Handling Data	21
Design.....	21
Implementation.....	22
Testing	25
4.3 Stock Analysis and Visualisation.....	26
Design.....	26
Implementation.....	26
4.4 Portfolio Creation & Optimisation	30
Design.....	30
Implementation.....	30
4.5 Creation of the GUI.....	38
Design.....	38
Implementation.....	39
Testing	43
4.6 Artefact Evaluation	44

5. Reflective Analysis.....	47
5.1 Reflection.....	47
5.2 Conclusion.....	48
5.3 Future Work	48
7. References	50
8. Appendix A: Methodology	54
1. Risk Matrix	54
9. Appendix B: Software Development	55
1. Retrieval System Test Cases.....	55
2. Artefact Test Cases.....	56

Figures

Figure 1 - Efficient Frontier (Finance Train, 2017)	10
Figure 2 - The effects of Diversification in different markets (Solnik, 1995).....	13
Figure 3 - Graph showing the popularity of languages found in Data Science job listings. (Muenchen, 2017)	16
Figure 4 - Stackoverflow's developer survey - Most popular Version Control (Stackoverflow, 2015)	18
Figure 5 - Evidenced Use of GitHub throughout project: Commit distribution	19
Figure 6 - Iterative version of SDLC. (Powell-Morse, 2016)	20
Figure 7 - Flowchart detailing process of Data Retrieval.....	22
Figure 8 - Code example of retrieving and compiling each stock's adjusted close price using Quandl's API.....	24
Figure 9 - Setting up the Cron job.	25
Figure 10 - Conceptual stock class design	26
Figure 11 - Generated time series of returns	27
Figure 12 - Example use of NumPy's built-in functions.....	28
Figure 13 - Example of generated adjusted closing price graph	29
Figure 14 - Example of generated daily returns graph.....	29
Figure 15 - Portfolio class diagram	30
Figure 16 - Compiled returns for stocks in portfolio	31
Figure 17 - NumPy method for covariance	32
Figure 18 - 4 asset covariance matrix.....	32
Figure 19 - Optimisation using CVXOPT	34
Figure 20 - Generated Efficient Frontier	34
Figure 21 - Generated Minimum Variance Portfolio	35
Figure 22 - Portfolio with a Risk Aversion level of 5	36
Figure 23 - Portfolio Optimisation using a Risk Aversion level.....	37
Figure 24 - Conceptual GUI class diagram	39
Figure 25 - Qt Designer interface	40
Figure 26 - Main Window.....	41
Figure 27 - Stock Chooser window.....	41
Figure 28 - Stock Performance	42
Figure 29 - Stock Analysis window	43
Figure 30 - Monte Carlo Simulation for feasible set	44
Figure 31 - Evaluation portfolio	45
Figure 32 - Portfolio performance	46

1. Introduction

1.1 Background

This section aims to set the scene of the project, providing context and introduction to the problem at hand. An investment is defined as “The Action or process of investing money for profit” (Oxford Dictionary, 2017). There are lots of different options that investors are faced with when it comes to choosing what to invest in. There are stocks, bonds, options, futures and countless more to consider. For this project, the focus is going to be on stocks. A stock is a portion of shares issued by a company (Oxford Dictionary, 2017). When investing into stocks, or anything else for that matter, the general goal is going to be that when it comes time to sell, they have risen in value enough that the investor has made a profit. However, making that profit almost never comes free. Each investment typically has an “Expected Return” and a “Risk” aspect associated with it. These can be defined in a number of ways. In the context of this project, “Expected Return” is defined as the anticipated profit or loss on an investment characterised by a historical average. “Risk” is defined as the volatility of an investment’s return, which is effectively the likelihood that an investment’s actual return will be different to the expected return. This is characterised as variance. The best possible investment would be one that had a very high return with no risk attached. Sadly, this is never the case as almost every investment is guaranteed to have some kind of inherent risk (Thangavelu, 2015). To achieve those higher returns, investors have to be prepared to take on the extra risk involved in doing so.

It would be unusual for investors to only invest in one asset at a time. Instead, what is created is a portfolio. A portfolio is “a range of investments held by a person or organisation” (Oxford Dictionary, 2017). Due to the insanely large amount of asset options that are available for investors to choose from, this introduces a small part of the problem that this project addresses. How do you decide what assets to invest in? Traditionally when creating portfolios, investors would select assets using anecdotal analysis or statistics of each asset individually. This changed when Harry Markowitz’s paper “Portfolio Selection” was released in March 1952 compiled within The Journal of Finance. Markowitz put forward among others, the idea of looking at how assets move and interact with the portfolio as a whole rather than on an individual basis. Instead of looking at an asset’s individual variance, it instead considers the risk of the portfolio to be determined by the covariance between each asset. In finance, covariance is a measure of how much the returns of two assets move together, contextualising the risk of a specific asset with the rest of the portfolio (Investopedia, 2017). One of Markowitz’s biggest ideas is that investors can potentially decrease the amount of risk they are having to take for the same return through means of diversification (Shipway, 2009). By spreading investments out over a larger number of diverse assets, the risk associated is generally going to be lower than if the investments were made in only one or two assets. This is due to each asset reacting differently to certain scenarios. With a large and diverse portfolio, there are going to be scenarios in which some assets will pay off, and some that won’t (Elton and Gruber, 1997).

The age old question of how to distribute budget between each of asset in a portfolio forms the problem that this project aims to provide a solution for. Modern Portfolio Theory (MPT) is a combination of Markowitz’s “Portfolio Selection” theory and William Sharpe’s later contributions in 1964 with his Capital Asset Pricing Model (CAPM). It is an investment framework that provides guidance for the creation of portfolios based on the maximisation of expected returns for a given risk (Mangram, 2013). Markowitz said that for every level of risk, there exists what he described as an “efficient portfolio”. This being a portfolio that provides the maximum return for that specific amount of risk. The Mean-Variance analysis techniques featured within Modern Portfolio Theory provide insight into mathematical ways of identifying

and calculating efficient portfolios through distribution of budget, optimising the trade-off between risk and return. Despite its reputation, original forms of Markowitz's optimisation model aren't extensively used for larger portfolios due to the computational complexity involved in optimising them (Konno and Yamazaki, 1991). Along with the larger advancements of computing into the financial world, optimisations like these are becoming easier to do, due to strides in both software and hardware. It is this area in which the project will take place; using mean-variance analysis to craft and optimise select portfolios to achieve the desired goals be it; minimum variance, maximum return or somewhere in between.

1.2 Aims & Objectives

Aim & Rationale

The aim and rationale of this project is to create a proprietary portfolio optimisation system that uses concepts of Modern Portfolio Theory to create investment strategies. It should be able to retrieve relevant and up-to date stock data providing visualisation and descriptive analysis to the user. It will allow the user to craft a portfolio using their choice of stocks from the available selection. Using concepts from Modern Portfolio Theory such as Markowitz's Mean-Variance model, this portfolio can then be optimised in order to achieve the user's desired goals. The statistics of the portfolio's potential performance will be displayed by the system so the user can then decide whether to proceed with that portfolio as their investment strategy.

Objectives

The objectives of this project are to:

- Use research to acquire and develop a greater understanding of Modern Portfolio Theory to help pinpoint requirements and guide implementation.
- Find and consider various options for a reliable online source of relevant financial data.
- Implement solutions using Python to achieve the following:
 - A retrieval system to fetch up-to-date price data for a specified list of stocks.
 - Perform statistical analysis on the data to provide the user with descriptive summaries of each stock as well as graphic visualisation.
 - Allow the user to craft custom portfolios manually choosing from the available selection.
 - Perform mean-variance optimisation to achieve the user's desired portfolio goal.
 - Provide the user with summarising statistics of their optimised portfolio.
- Create a full graphical user interface to contain and situate the functionality of the solutions that have been created to form an interactive desktop application.

2. Literature Review

This chapter illustrates and explains some of the underlying concepts of this project through the review of relevant academic work. Description and analysis of their work is performed, furthered by presenting a comparison of their findings with work of a similar nature.

2.1 Modern Portfolio Theory

Background

The original founder of Modern Portfolio Theory (MPT): Harry Markowitz had his paper “Portfolio Selection” first published in the Journal of Finance, March 1952. His work and theories provided completely new and revolutionary insight into many areas of finance, totally changing the traditional methods of managing investments which awarded him a Nobel Prize in Economic Sciences in 1990. (Elton and Gruber, 1997). He characterised and formulated the problem of financial portfolio selection as a more than just a consideration of return, but as a combination of both; risk and return, suggesting that there is an unavoidable trade-off to be made. MPT assumes there to be a positive relationship between risk and return for each investment, meaning that to obtain a higher expected return, there has to be a higher risk involved. This introduced the concept of mean-variance analysis into portfolio theory, called the Markowitz Model. It explains that for every level of given risk in a portfolio, there is what Markowitz describes as an “efficient portfolio”, or one portfolio that provides the greatest possible return for that level of risk. This laid the foundation for other theories such as the “Capital Asset Price Model” which extended the analysis of determining what assets should be included into a portfolio (Elton and Gruber, 1997).

The Markowitz Model

The main idea of the Markowitz Model is to try describe and optimise the relationship between risk and return through the selection and weighting of assets in a portfolio. A portfolio that is optimised would be considered efficient, as it would have the highest return for a given risk level or vice versa. When calculating these efficient portfolios, Markowitz makes fairly reasonable assumptions about how investors make decisions. The main important assumption is that investors are risk averse, meaning that, given a choice of two portfolios with the same expected return but different risk, an investor will always choose the one with the least risk.

Markowitz modelled the return of an asset as a random variable following a normal distribution. The average or mean of those returns can be typically taken as the expected return of the asset, with the dispersion of the returns being characterised as the risk of the asset. As a portfolio is made up of a weighted combination of risky assets, the expected return of a portfolio over a single period can be described as the weighted sum of all the asset’s expected returns expressed as the following:

$$E(R_p) = x_1E(R_1) + x_2E(R_2) + \dots + x_iE(R_i) \quad (1)$$

Or

$$E(R_p) = \sum_{i=1}^n x_iE(R_i) \quad (2)$$

where

$E(R_p)$ = the expected return of a portfolio over time period p

$E(R_n)$ = the expected return of asset n over the period

x_i = the weight of asset i

i = the number of assets in the portfolio

As Markowitz modelled the return of an asset as a random variable, the variance can be used to describe the dispersion of those returns around an expected value which he argued provides an intuitive measure of an asset's risk. Markowitz theorised that to calculate the risk of a portfolio it is not enough to look at weighted sum of each asset's risk individually like before with expected return, instead the covariance between each asset must be considered as well. Covariance measures the degree of which each asset's returns will move with one another, with a positive covariance meaning that they move together (Investopedia, 2017). The covariance of two assets describes a similar phenomenon to correlation except that it is not scaled between -1 and 1. Markowitz calculates the variance or risk of a general portfolio using the following:

$$\sigma_p^2 = \sum_i \sum_j x_i x_j \sigma_{ij} \quad (3)$$

where

σ_{ij} = covariance of periodic returns on two assets, i and j .

Each portfolio will have its own specific risk and expected return values which are calculated from its assets and their weightings. These asset weightings are what form the basis on how optimisations can be performed to maximise and minimise expected return and risk, respectively. The feasible set is what is described as the collection of all possible portfolios that can be created from that specific selection of assets. This collection includes all of the efficient portfolios, which form the upper bound of the feasible set as they provide the highest return for a given level of risk. Figure 1 shows the Efficient Frontier for a portfolio containing only risky assets (Elton and Gruber, 1997). No portfolios within the feasible set can lie above the Efficient Frontier, which demonstrates how any efficient portfolio that forms it, dominates the inefficient portfolios below it.

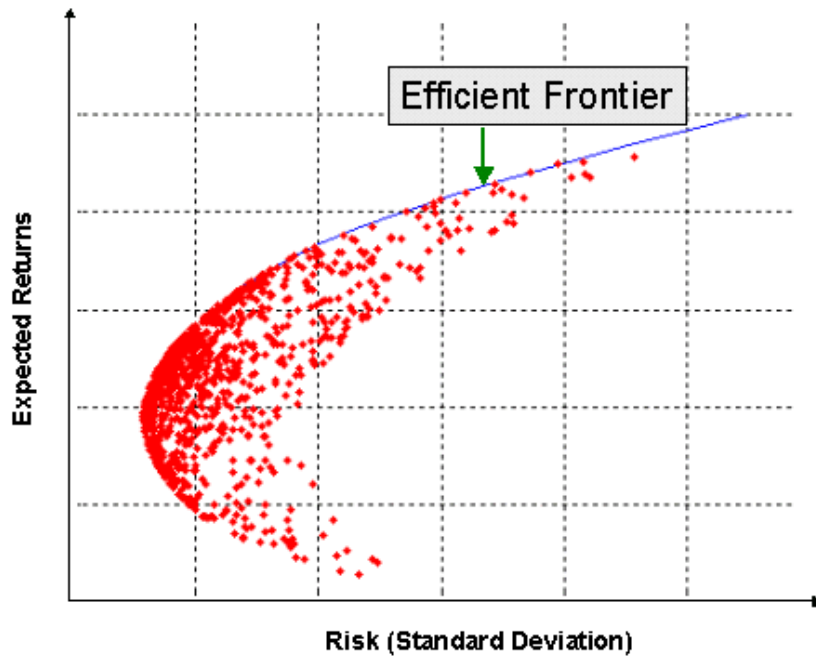


Figure 1 - Efficient Frontier (Finance Train, 2017)

Under the mean-variance framework, efficient portfolios can be calculated two different ways. Either by minimising the risk of a portfolio for a given return, or by maximising return for a given risk. Given a portfolio of two risky assets i and j , using the Markowitz model, optimisation can be done to find the lowest risk for a given return. Constraints for this model are that the sum of the asset weights must be 1 and that the return of the portfolio must be equal to the desired return. This problem is described as the following:

$$\text{Minimise } \sigma_p^2 = \sum_i \sum_j x_i x_j \sigma_{ij} \quad (4)$$

$$\text{Subject to } \sum_{i=1}^n x_i E(R_i) = \mu \quad (4.1)$$

$$\sum_{i=1}^n x_i = 1 \quad (4.2)$$

Markowitz showed that techniques of calculus can be used to solve this problem through introduction of Lagrange multipliers λ_1 and λ_2 . These are additional variables that capture the effects of the two constraints respectively. With the addition of these variables, the Lagrangian is formed:

$$L = \sum_i \sum_j x_i x_j \sigma_{ij} + \lambda_1 \left(\sum_{i=1}^n x_i E(R_i) - \mu \right) + \lambda_2 \left(\sum_{i=1}^n x_i - 1 \right) \quad (5)$$

The partial derivatives of the equation can be taken and set equal to 0, giving a series of linear equations that can be solved to find the portfolio weights.

$$\frac{\delta L}{\delta x_i} = 2x_i\sigma_{ii} + 2x_j\sigma_{ij} + \lambda_1 E(R_i) + \lambda_2 = 0 \quad (5.1)$$

$$\frac{\delta L}{\delta x_j} = 2x_j\sigma_{jj} + 2x_i\sigma_{ij} + \lambda_1 E(R_j) + \lambda_2 = 0 \quad (5.2)$$

$$\frac{\delta L}{\delta \lambda_1} = x_i E(R_i) + x_j E(R_j) - \mu = 0 \quad (5.3)$$

$$\frac{\delta L}{\delta \lambda_2} = x_i + x_j - 1 = 0 \quad (5.4)$$

These linear equations are described in matrix form by the following:

$$\begin{bmatrix} 2\sigma_{ii} & 2\sigma_{ij} & E(R_i) & 1 \\ 2\sigma_{ji} & 2\sigma_{jj} & E(R_j) & 1 \\ 1 & 1 & 0 & 0 \\ E(R_i) & E(R_j) & 0 & 0 \end{bmatrix} \begin{bmatrix} x_i \\ x_j \\ \lambda_2 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \mu \end{bmatrix} \quad (6)$$

Or

$$Cx = k \quad (6.1)$$

These matrices will can then be solved by computing the inverse matrix:

$$C^{-1}Cx = C^{-1}k \quad (6.2)$$

$$Ix = C^{-1}k \quad (6.3)$$

resulting in:

$$x_i = a + b_1\mu \quad (6.4)$$

$$x_j = a + b_2\mu \quad (6.5)$$

where a and b are constants.

This will give values for the weights of assets i and j in terms of μ which was the set expected return values that the minimum variance was being calculated for. Once that is achieved, μ can then be varied between feasible limits in order to obtain values of the efficient frontier. There are different goals that investors can have which will require the setup of new and different constraints, leading to more complex equations.

Model Inputs

A common way to calculate inputs for the Markowitz model is to look at historical data as a means to estimate values for sample mean for expected return and variance for risk. There is contention about the accuracy and effectiveness of using historical data as inputs for the mean-variance optimisation model. Fabozzi et al (2002) discuss the effectiveness of using historical data as an input, stressing that there is a strong level of importance in assessing the economic

conditions when using that period to estimate return and variance. It goes on to suggest that there are reasons for the use of historical data, including arguments about the length of the histories, as well as the political and financial state of that market playing a major role in assessing the viability of the historical data. They argue that only after an economy has a long record of good performance and stability before should they be considered for use as an estimation. To contrast this point, Michaud (1989) created a similar implementation of a mean-variance optimiser using historical data. He theorised that the poor performance of the model was due to the use of historical data. Going on to suggest that a trait of the optimiser was causing it to maximise the error contained within the data due to what he believed was because the estimations made ignored what he described as the “inherent multivariate nature” of the problem. Sharpe (1999) argues that a lot of performance measures are justified in their use of historic data in some applications due to the likelihood of predicted relationships that can be extrapolated in theory but not always in practice.

Furthering the discussion of effective inputs for the Markowitz model, Konno and Yamazaki (1991) question the use of standard deviation of historical returns as a measure of risk. They note that an investor’s perception of risk is not symmetrical around the mean of the returns. Arguing that investors aren’t as concerned when there is large positive deviation from the mean compared to small negative deviations and going further to say that market returns are generally not distributed symmetrically. Ways to classify these return fluctuations have been devised through the use of a newer estimator called “Downside Risk”, which is the estimation of an asset’s potential to decline in value (Investopedia, 2017). Rockafellar and Uryasev (2000) assess the use of alternatives to standard deviation in optimisation practices, recommending the use of more intricate methods to characterise risk, such as Value-at-Risk (VaR), or even Conditional Value-at-Risk (CVaR). Both of which are based on the use of standard deviation of normal distributions and contain undesirable attributes described as a “lack of subadditivity and convexity” making optimisations more computationally difficult as well as an increase in implementation technicality.

Diversification

The concept of diversification is one of the most important parts of Markowitz’s work. Even today, the concept of diversification is commonly used in portfolio management, as it has such an intuitive appeal as well as an empirical backing. Diversification is the process of spreading investments over a wide range of investment types and areas in an attempt to reduce unsystematic risk (Markowitz, 1952). There are two types of risk that the financial world. There is systematic and unsystematic risk, both of which are a component of the overall risk that is present within assets and portfolios. Systematic risk can be defined as the risk that is inherent to an entire market, meaning that it is something that all assets that are traded within that market are subject to. An example of Systematic Risk would be the effects applied on the whole market by events such as The Great Recession, not the event itself. Events that have these affects are what is known as Systemic Risks (Investopedia, 2017). Unsystematic Risk is the part of an investment’s risk that is specific to that investment, influenced by events attributable to the investment itself or the sub-group it is in. Not the entire market system like Systematic Risk (Investopedia, 2017).

The effectiveness of Diversification at reducing unsystematic risk is generally undisputed. Solnik (1995) analysed the effects of diversification on markets outside of the US finding that it was almost just as effective in European markets. However, he noted that the proportionality of the number of assets compared to the amount risk reduced was not consistent and it was impossible

to completely eliminate risk after a certain point because of what can be attributed to the assets moving together. He suggested that it is possible to reduce risk past this certain point through further diversification extending the holding of assets across multiple markets rather than just one.

Figure 1. United States

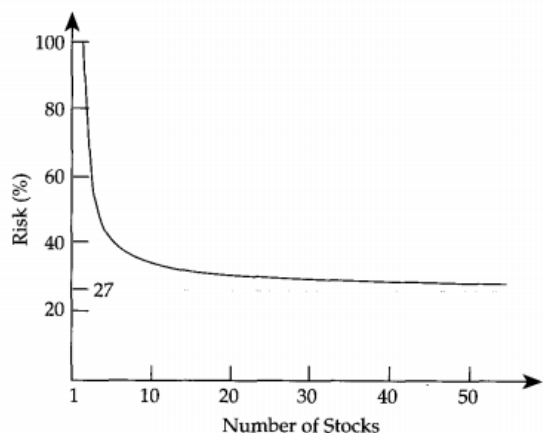


Figure 2. United Kingdom

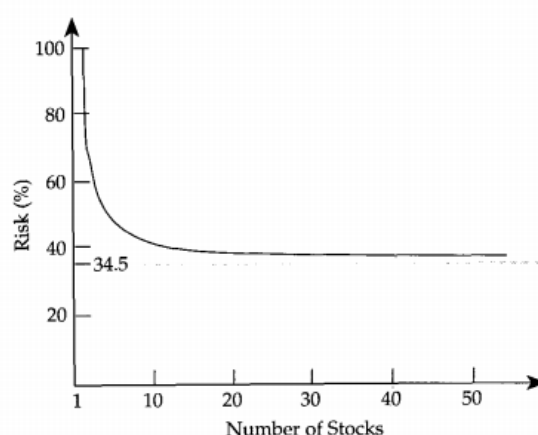


Figure 2 - The effects of Diversification in different markets (Solnik, 1995)

Raffestin (2014) builds on the summary provided by Solnik (1995), investigating the effectiveness of Diversification at reducing the impact caused by systemic risks, theorising that Diversification can make investors safer individually but can actually increase the effects caused by a systemic risk due to common asset holdings between each investor. He described the connections made by these common asset holdings as “endogenous covariance” that can actually propagate the effects of systemic risk through the system. He found again, that investors can minimise their risk by spreading wealth across not only a greater amount of assets, but more distant assets, potentially increases the overall stability of a market. This further reinforces the importance of strong Diversification in a portfolio.

Portfolio Performance Evaluation

There are a number of different metrics to measure the overall performance of a portfolio. One of the oldest and most influential performance metrics is the Sharpe Ratio developed by William Sharpe in 1966 (Sharpe, 1994). It calculates the overall reward to risk ratio of an asset or portfolio through the use of a benchmark, typically described as a “risk-free asset”. The choice of risk-free asset weighs in heavily to the effectiveness of the Sharpe Ratio. It’s debated in practice whether there are assets that are totally risk free. However, in theory index funds and other assets such as government bonds are typical candidates when it comes to selecting a risk free asset.

Lediot and Wolf (2008) test the overall performance and robustness of the Sharpe Ratio as a performance metric, finding that the use of it in fairly stable conditions can actually be quite effective. However, they state that as the Sharpe Ratio generally requires the use of historical data as an input, it can produce misleading conclusions. Primarily occurring when there are tails heavier than the normal distribution of the data, noting that it shouldn’t really be used because these tails are common within financial returns. Christie (2005) largely tends to agree with Lediot and Wolf’s (2008) conclusion suggesting that the biggest limitation of the Sharpe Ratio is that its inputs; expected return and risk are measured with error, leading the Sharpe Ratio itself to contain error.

3. Methodology

Every project will usually have some kind of project management and software development methodology that it will follow. For very small projects, methodologies aren't really much of a concern, as they generally don't last long enough or aren't intense enough to utilise the full benefit of following an established software development methodology. However, as the project grows in size, it becomes more and more important to be following a methodology as it can make time and cost more foreseeable, as well as boosting efficiency by sticking to a thought-out schedule (Awad, 2005). This makes it important to consider a range of methodologies for both project management and software development. Before choosing the most optimal methodology, relevant analysis would need to be done for this particular project.

All of the software development methodologies out there each have their own strengths and weaknesses that can become more or less apparent depending on the type of project. One of the most influential concepts that affects the viability of a software development methodology is the initial state of the software's requirements (Balaji and Murugaiyan, 2012). The way the requirements were set out at the beginning was very uncertain. The artefact being produced from the project was always going to be making use of various concepts within Modern Portfolio Theory (MPT). As MPT is a well-established theory, all concepts featured within it are already laid out and are not going to change. This makes it seem possible to create a full list of requirements from what had been researched, and proceeding with a traditional sequentially structured methodology such as Waterfall. However, due to the sheer amount of concepts situated within MPT, it is extremely hard at the beginning of the project to fully gauge how many of these concepts were realistically going to be implemented, and in general, how far the artefact could go. The project supervisor acted not far from what could be considered a type of client to the project. The direction and requirements that were dictated by it were often the result of in-depth discussion with the project supervisor. This set a slight feature-driven-development tone to the project as the outcomes of these discussions provided both general direction and priority. This uncertainty alone would rule out the use of a sequential methodology like Waterfall. It tends to benefit from having very well defined requirements, as well as a strong time and cost forecast which would be unable to accommodate this type of project (Balaji and Murugaiyan, 2012).

Going off the exclusion of rigid methodologies such as Waterfall, the logical choice is something a lot more flexible that could adapt to the initial lack of knowledge in the chosen field, making determination of detailed requirements and calculating an accurate time plan for the project impractical. This lent itself to choosing a more adaptable methodology that could account for these unknown details, providing a framework for both software development and project management. In terms of project management, some of the more intricate and established methodologies such as PRINCE2 seemed a little too process heavy and intensive for a solo project. Something such as PRINCE2 is more suitable and effective for a larger team, where allocating people and resources is more of a complex problem. It would be unnecessary and perhaps detrimental to try incorporate this large methodology for a small project like this (Matos and Lopes, 2013).

An Agile methodology would be most fitting for a project of this specification. It provides multiple opportunities to assess the overall direction the project is heading in, which accommodates the uncertainty of this project's final goal. The iterative and incremental nature of Agile makes it extremely appealing as there are regular intervals in which new requirements can be gathered and work done previously can be built on and improved (Agile Methodology,

2008). While Agile itself is not a methodology, there are methodologies that use Agile principles. One of the most popular ones is SCRUM (SCRUM, 2017). One of the problems with using SCRUM, is that it is so heavily focused on team communication and feedback, that a lot of its benefits are wasted on a one-man project. For this project, the chosen methodology is Agile in nature, primarily combining aspects of Extreme Programming (XP) and Feature-Driven Development (FDD). XP accommodates the heavy software development aspect of the project as well as the presence of changing requirements and FDD relates to the prioritising of features to include in the final release of the artefact.

3.1 Project Management

Time Planning

Time planning is something that is essential to any project's success. A Gantt chart is a commonly used time planning tool that provides good visualisation of how time is going to be spent over the course of the project (Investopedia, 2017). However, since the project follows an Agile methodology it is difficult and often useless to try and plan every part of the project accordingly. The project is prone to changing requirements and deadlines throughout, making the process of accurately planning out each stage on a weekly basis a troublesome task. The number or length of iterations that will be present in the development process were unknown so the creation of a Gantt chart to try and visualise this is difficult. Instead, with cooperation of the project supervisor, temporary deadlines were set as the project went on to ensure completion.

Risk Management

As the project contained multiple software based components, it was important to consider some of the different risks that were involved. An essence of project management that could be utilised to summarise and handle this risk is what is called a Risk Matrix. A Risk Matrix can be defined as a structured approach to identifying risk, defining which risks are most dangerous to the project, what impact they might have, and how they can be dealt with (Garvey and Lansdowne, 1998).

A Risk Matrix has been formulated to provide evaluation and insight into potential risks that may be present within the project, see Appendix A. 1. Risks are defined and given likelihood values to describe how possible they are as well as an impact value to portray how much that particular risk would affect the project if it were to occur. Management and mitigation steps are also provided to give an idea of how each risk can either be dealt with or avoided.

Development Tools

This section delves into the main tools and applications that were used throughout the whole process of creating the project's final artefact. It explains what the tools were used for, how they were chosen and if applicable, what were some possible alternatives. Tools that were only used for more specific areas of project development are discussed during the development section of the report.

Programming Language

One of the most important tools utilised in a software development project is the programming language that it is created in. There are programming languages that are much more suited to specific tasks than others and it's helpful to pick a suitable one so as to not make things purposely difficult or inefficient. When deciding what programming language is going to be used, it is

important to classify what kind of problems are needing to be solved using language. A brief overview of the tasks in this project involve a series of data analysis and handling techniques including fairly comprehensive mathematical procedures. Finally, the creation of a graphical user interface (GUI) is also necessary, which really requires some kind of object-oriented principles.

Based on the data handling and mathematical function requirements alone, some potential candidate languages would be Python (Python Software Foundation, 2017) and R (R Core Team). R is one of the most popular languages used in data science and provides a large majority of the utilities needed for the project. The R environment contains an integrated suite of facilities for data manipulation, calculation and graphical display (R Core Team, 2017). However, R unfortunately does not contain the object oriented features that would be necessary to create a GUI. Python on the other hand, was not initially designed as a data analysis tool. However, due to its general applicability, several high quality modules have been developed to handle and manipulate data, providing similar functionality to R. This has made Python extremely popular in the Data Science world and a real competitor to R (Muenchen, 2017). This makes Python the ideal choice for this project due to its versatility as it can perform everything necessary to assure its completion. There are quite a few different distributions of Python, each coming with their own pre-packaged modules. Since this project is fairly heavy on mathematical and data handling processes, Anaconda (Continuum Analytics, 2017) was the chosen distribution as it already contained a lot of the necessary Python modules with it, making it the logical choice.

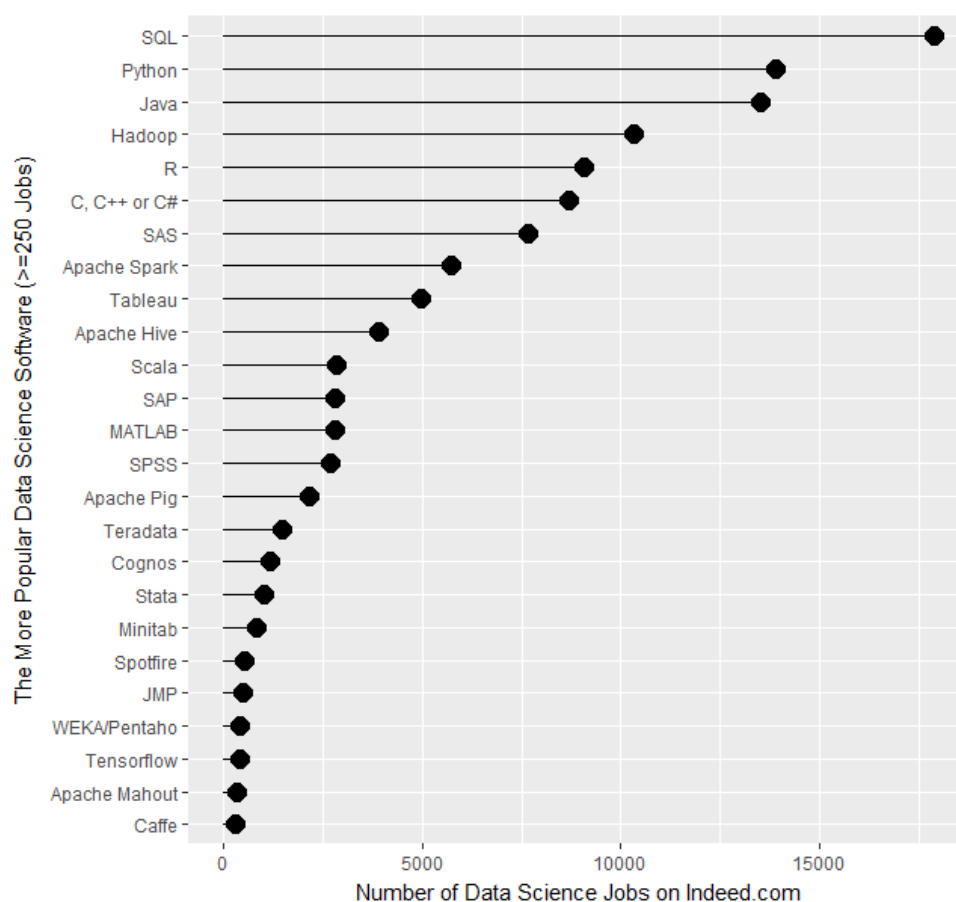


Figure 3 - Graph showing the popularity of languages found in Data Science job listings. (Muenchen, 2017)

Integrated Development Environments (IDE)

IDEs can be fairly important when it comes to developing software. They package most of the fundamental tools that a developer is going to need into a single graphical user interface. This streamlines the process of writing code as it provides many quality of life improvements that can often be overlooked in terms of importance. Spyder (Spyder Developer Community, 2017) is a cross-platform IDE that is used for a lot of Python projects involving data processing and mathematics. Some of its main features that were utilised during the development process were things such as its integrated console. It has an interactive shell called IPython that provides support for data visualisation and also contains a flexible interpreter providing a quick and built-in way to run the program in development, making debugging and general program execution a lot faster. It provided instant, on-screen console output pertaining to the program as well as detailing bugs were they to occur.

Code completion also provides a huge boost to productivity, especially within object oriented projects. It helps by providing suggestions of relevant attributes pertaining to that specific class or object, as well as displaying information about that function's parameters. This can become extremely useful when the project contains multiple instances of a complex class by making it easier to see how that class is set up.

A minor but important feature none the less, is syntax highlighting. Code can get difficult to read and understand when the size of the project begins to ramp up. Spyder contains settings that allow full customisation of syntax colours, providing easy ways to differentiate between various elements within code, such as functions, classes and data types. This greatly increases the overall clarity of the code by making its structure much more visually defined, making it easier to find errors stemming from specific areas of the code.

Version Control

With any project it is necessary to have some kind of centralised storage for all work created that it accessible from anywhere. Some storage systems are a lot more advanced and contain more effective forms of Version Control. A Version Control System (VCS) is a system that records all changes made to a set of files that are located within that storage system. It provides date and time specifications, as well as exact character changes that provides the user with the ability keep track of changes and recall older versions if necessary (Chacon and Straub, 2014). The benefits of using a VCS become more noticeable and apparent when doing a larger and more complex project. It becomes even more useful in a project such as this because of its iterative nature and use of Agile methodologies.

VI. SOURCE CONTROL

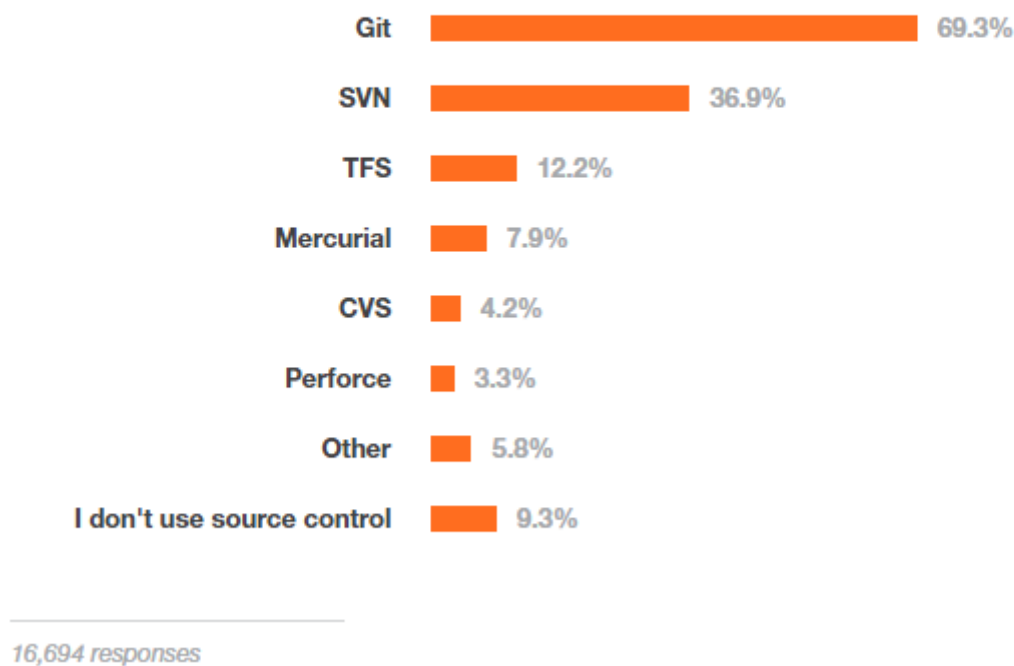


Figure 4 - Stackoverflow's developer survey - Most popular Version Control (Stackoverflow, 2015)

In 2015, Stackoverflow ran a survey for developers asking them a series of questions. One of the questions was their chosen VCS. Figure 4 shows the results from that survey which had almost seventeen thousand responses. The results show that roughly seventy percent of those developers used some version of Git (Hamano, 2017). This provided a solid indicator of which VCS is most popular and likely the most effective.

Git and an extension of that being GitHub (GitHub, 2017) was used for this project. University students get free access to a series of private online GitHub storage repositories that are ideal for a project like this as they have practically unlimited storage space. GitHub was used extensively throughout the project from the very beginning. Project work was often not done in the same place and usually on different systems. The work stored in the repository ranged from code used in development to diagrams and word documents as it was logical to keep everything in one place. The centralised nature of a system like GitHub provided ways to pull up-to-date versions of the project to any system and then push any changes that were made during that session back to the online repository. This made working on the project very flexible as it was extremely easy to access the repository from almost any location should it have been necessary. In conjunction with the push & pull system that GitHub incorporates, the amount of information that is available pertaining to each change made is extensive. This can help keep track of progress made during the project, as well as retaining old solutions that may have otherwise been discarded should they prove useful again.

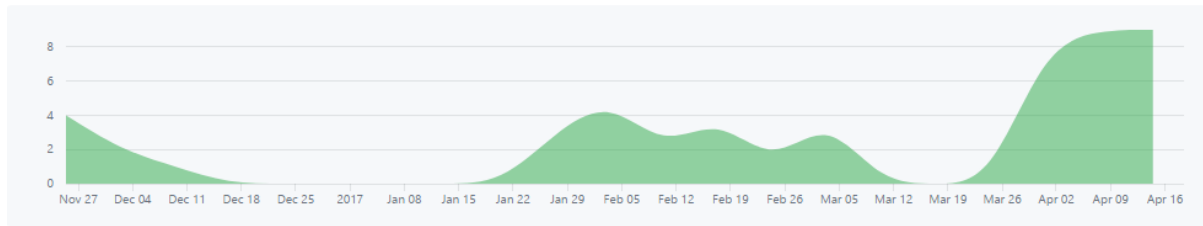


Figure 5 - Evidenced Use of GitHub throughout project: Commit distribution

Qt Designer

Creating the visual design and layout for Graphical User Interfaces (GUI) through pure coding practices can become an extremely time consuming process unless the exact look is solidified from the outset and not many changes are made. For projects where the visuals of the GUI aren't the main aspect of the development, many developers will opt to use other software to accelerate and streamline the creation process. Qt (The Qt Company, 2017) itself is an application framework that runs on several different software and hardware platforms without requiring much code adaptation. Qt Designer is one of those applications that use this framework. It provides a toolkit containing useful features such as prebuilt widgets and layout structures to help users facilitate their application functionality in a timely manner (Blanchette and Summerfield, 2006). When it came designing and creating the layout of the GUI for this project, Qt Designer was used to handle almost all of the design tasks. It was extremely useful in the creation of the GUI wrapper as it saved a lot of time by automatically generating code for the layout designed within Qt Designer. Meaning that all the visuals of the application were instantly setup and all that needed to be done were tasks such as connecting inputs to specific functions and handling the outputs.

4. Software Development

This section details the full development process for the artefact created during this project. It does not follow the traditional Software Development Life Cycle (SDLC) due to the Agile iterative methodology that has been adapted to this project. The development in this project is done over multiple iterations that repeat the typical stages found in the SDLC. Figure 6, shown below roughly outlines the overall process used in the development of this artefact. Requirements were gathered before each iteration to figure out what was needing to be done. Suitable design was then generated based on those requirements which would then be implemented. Relevant testing to a varying degree would be performed at the end of each iteration to ensure that the result of that iteration was working as intended before starting the next. Some of the later iterations rely on and utilise functionality produced in the earlier iterations which makes it imperative that the dependencies found within the earlier iterations are fully complete. More in-depth testing will be done for the completion of the final artefact, once everything has been brought together. The project supervisor provided significant input when it came to gathering requirements and assessing the output from each iteration to ensure that it was time to begin the next iteration.

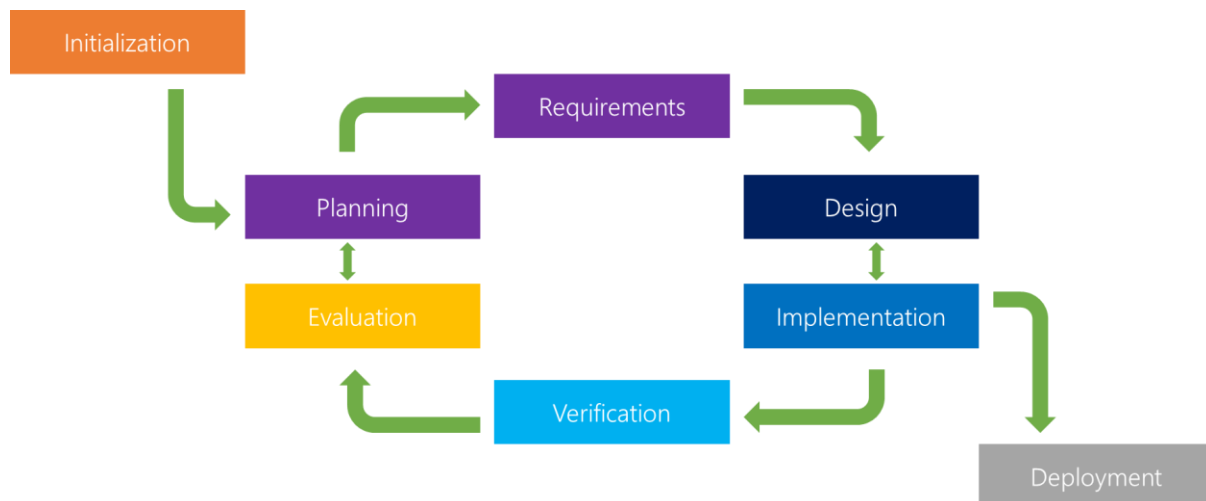


Figure 6 - Iterative version of SDLC. (Powell-Morse, 2016)

Requirements

The project artefact as described in the Aims & Objectives section of the report is a portfolio management system that can provide descriptive statistics and visualisation of an asset's performance as well as provide optimised asset allocation suggestions for a portfolio. Upon meeting the supervisor of the project, possible directions and solutions were discussed on what kind of analysis was to be done, as well as possible models that could be implemented as a means of portfolio optimisation. As a primary direction of the project, the Markowitz model as detailed by Harry Markowitz, was suggested as a model to start with, which could then be built and expanded on later if time prohibited it. This made logical sense as the Markowitz model is seen as one of the most intuitive models as well as being considered the foundation of portfolio theory as it's known today. Following the details of the Markowitz model is ideal as a starting solution to the project aim as it provides a means of asset analysis as well as portfolio optimisation techniques. There are various areas of the Markowitz model to implement which have been separated into steps and divided between development iterations. Each step generally requires

the completion of the previous step in order to be implemented. Requirements that are specific to the implementation of certain step are discussed and explained in greater detail at the beginning of that iteration. The overall development focuses on the implementation of these stages of the Markowitz model to achieve the project aim.

4.2 Retrieving and Handling Data

As per the requirements of the project, the acquisition of relevant, up-to-date financial data is of utmost importance. After discussion with the project supervisor and further research into previous implementations of the Markowitz model, it was found that historical data pertaining to an asset's price is often used as a means to generate estimations for input values that are used for analysis and optimisation later on in the Markowitz Model. Furthermore, historical data is often used by investors to gather descriptive statistics that can provide guidance and indication as to what stocks are good investments. As the project specifically focuses on the use of stocks as the chosen asset class, appropriate stock price data will need to be acquired and stored appropriately.

Design

During the design phase different solutions were considered about how best to approach the process of retrieving data. The initial idea was that the data retrieval and storage process was going to be ran in conjunction and separated to the final portfolio analysis tool. As the data had to be kept up-to-date to ensure relevancy, the idea of automatic retrieval on a daily basis seemed appealing because new stock pricing data becomes available every day and it seemed relatively hands-off for the user once it is set up. This reinforced the idea of saving the daily retrieved data to a database hosted on a cloud server. Local storage would not suffice as it lacks accessibility for other systems as well as the continuous need for the user's system to be available so data can be stored. The crux of continuous availability also applied to the process of running the automatic retrieval program periodically. It made sense to host the automatic retrieval system on a cloud server as well, so it could be constantly active. The use of a database in conjunction with the ability to pull directly from the data source also acts as a backup for if one of them were to become unavailable. Figure 7 shows a conceptual design of the automatic retrieval system in relation to the online financial data source and the created database. Considerations would have to be made about services were available for cloud hosting, for both: the retrieval system and the database.

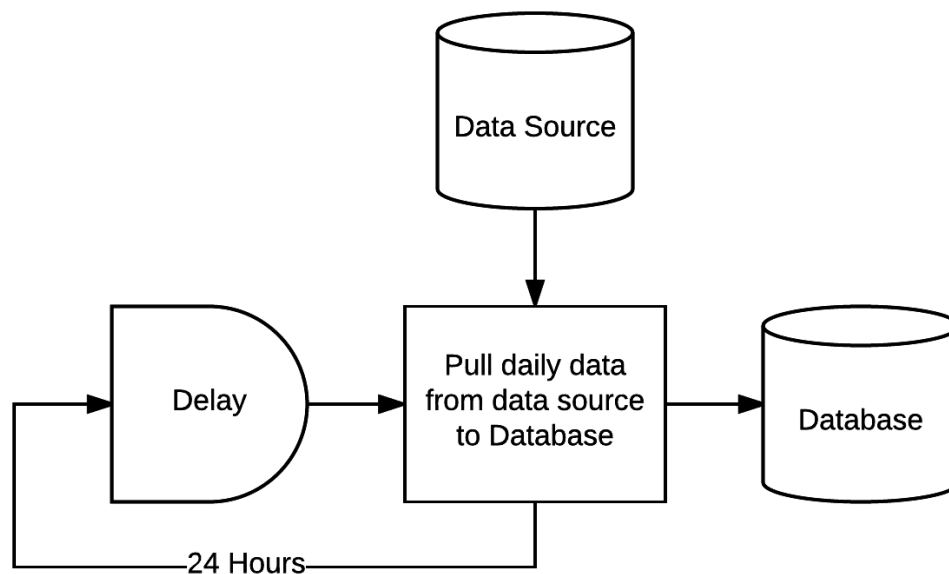


Figure 7 - Flowchart detailing process of Data Retrieval

Implementation

Going into the implementation phase, implementation goals can be generated for this iteration based off of the initial requirements of the project and the ideas and conclusions arrived at during the design phase. The implementation goals for this particular iteration can be summarised as:

- Find a suitable online source for daily stock pricing data
- Create a database to store the data and find a suitable cloud service to host it.
- Create a Python script that can be run every day to retrieve the pricing data for that particular day and store it in the database, as well as a service to host it.
- Create separate script to then be able to access that data for future use in the artefact.

Quandl

When looking for what to use as a source of stock pricing data, there are several different things to consider. The most important things are the legitimacy and accuracy of the data, especially when applied in practice. During development, there isn't as much of an issue due to their being no real world consequences of using inaccurate data when testing the application. However, it makes sense to ensure the quality of the data at the beginning anyway, even before deployment. Some of the most commonly used online sources of data for individuals and small companies are websites such as Yahoo! Finance (Yahoo!, 2017) or Quandl (Quandl, 2017). Both of these provide free access to a huge amount of different datasets that are updated every day for almost every different asset class. Both websites allow users to easily retrieve data from the desired dataset using the website's own API. Quandl generally stood out more compared to Yahoo! Finance due to the detailed documentation and instructions they provided for using their API, as well as direct code examples of how to implement it using a number of different languages, most important of which was Python. Quandl have their own Python module that can be easily downloaded and imported, giving access to the functions necessary for making the API calls. The

API call function takes several parameters to specify details about the call: name of the dataset, ticker for the stock and start/end date. An example demonstrating the simplicity of calling the Quandl API and storing the data in a variable using Python as per the Quandl Documentation (Quandl, 2017):

```
data = quandl.get_table('WIKI/PRICES')
```

MySQL Database

The data storage solution that was chosen for this project involved creating a MySQL database that would be hosted on a cloud server. MySQL is the world's most popular open-source relational database management system (MySQL, 2017). There are several different services that can be used to create databases, but MySQL seemed the most appropriate due to having previous experience using it, its prestige and the wide range of services that provided cloud hosted instances of it. One of the most popular cloud based services providers is Amazon Web Services (Amazon, 2017). They provide various cloud-based services including instances of MySQL databases and virtual machines making them an obvious choice for hosting both the database and the automatic retrieval system (Butler, 2014). They have the Amazon RDS service for relational databases and the Amazon EC2 service for instances of virtual machines. Both of these services come under Amazon Web Services' (AWS) free tier providing a certain amount of usage per month, making the use of AWS ideal for a project of this size.

Once both of the AWS cloud instances had been established, the database would then have to be configured so it contained the correct tables for storing the relevant data. Instead of using dozens of SQL queries to set it up, MySQL have their own visual database design tool called MySQL Workbench (Oracle Corporation, 2017) which combines the design, development and management aspects of creating the database into one IDE. This was used throughout the process of creating the database as it provided a centralised way for to be accomplished. This made it a lot easier to create and configure essential parts of the database such as the schema and tables including the handling of connection permissions. It also provided a means of visualising the data stored within the tables making it much easier to test and ensure that the data was being retrieved and handled properly.

Retrieval System

Once the database had been set up and a suitable data source had been selected, the next step was to create the Python script that would pull data from the data source and then push it into the database. There are many different pieces of information that can be retrieved pertaining to each stock, but there is only one which is going to be of use to the project, that is the "Closing" price. A stock's closing price is the final price that it ends on after a day of trading, giving the most recent price valuation until trading begins again the next day. However, closing price as it stands is not particularly useful for analysis due to certain corporate actions such as stock splits, dividends and rights offerings that can appear as bizarre changes in price (Investopedia, 2017). Fortunately, Quandl also provides what is called the "Adjusted" closing price which is the same as the closing price except that the price is amended to include the effects of any corporate actions that may have taken place. It is the main type of historical stock data that is used to calculate historical returns which are used for analysis and optimisations such as those in the Markowitz model (Investopedia, 2017). Initially a list of 15 or so stocks were picked out that were mainly to do with large tech firms. However, after research into the Markowitz model and the effectiveness

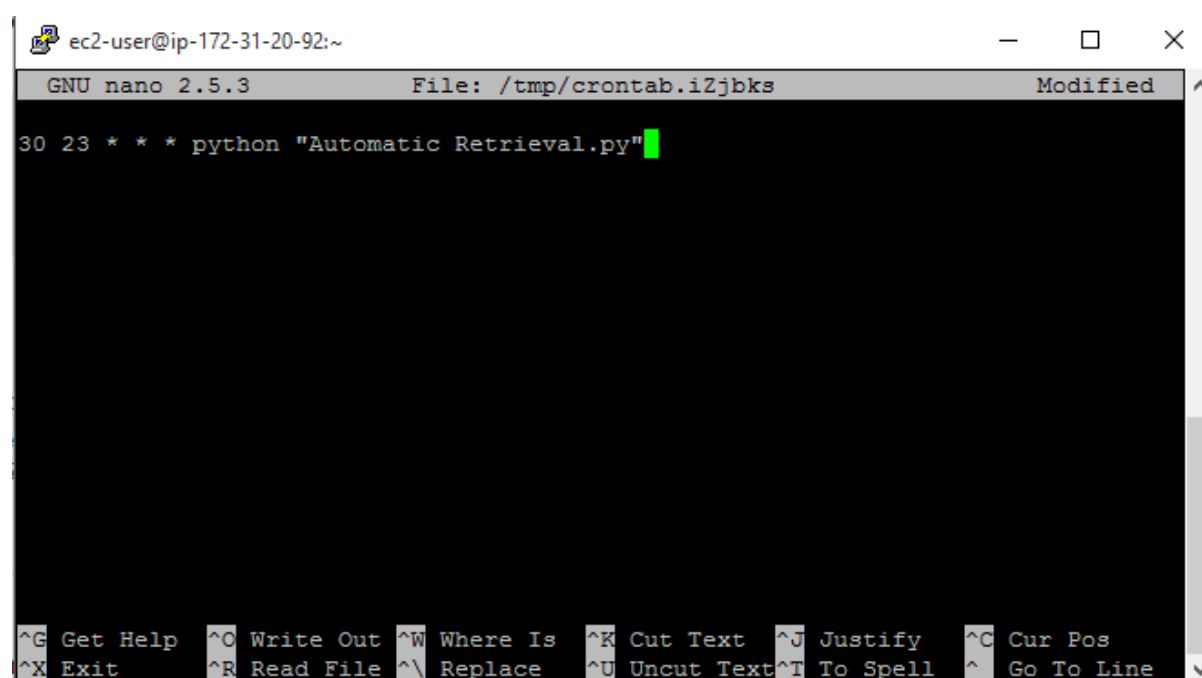
of diversification, this was changed to include roughly 25+ different stocks. Solnik's (1995) research into diversification provided a good visualisation of how effective diversification can be. Visualised in Figure 2, it roughly shows that for US stock markets in which this project takes place, the bulk effect of diversification caps out at around 20 or so stocks. Instead of just increasing the number of stocks, the selection was also made more varied by including stocks of companies outside the tech sector to further increase the potential effectiveness of diversification.

The process of retrieving data was done using two similar but separate Python scripts. The first script would retrieve the bulk of the pricing data from a desired starting point. A year's worth of data was chosen for this project. This would act as an initialisation script that would ideally only need to be run one time upon setting up the database. The initialisation script loops through the list of selected stock names, calling the Quandl API for each one retrieving all the data available for that stock. The adjusted closing price column of each stock is renamed to the name of the stock and then appended to an empty container. A Pandas dataframe is used to store and handle all of the data, as it is relatively easy to use while having high performance and a bunch of analytical functions built into it, including one that pushes it to an SQL database (Pandas, 2016). Once the adjusted close price of each stock had been appended this dataframe was pushed to the database that had been setup which would now contain the daily adjusted closing price of each stock in a time series format.

```
allData = pd.DataFrame([])
for stock in availableStockList:
    fullHolder = pd.DataFrame(quandl.get("WIKI/"+stock, trim_start =
start, authtoken = token))
    closeHolder = pd.DataFrame(fullHolder['Adj. Close'])
    closeHolder.columns = [stock]
    if allData.empty:
        allData = closeHolder
    else:
        allData = allData.join(closeHolder, how='outer')
```

Figure 8 - Code example of retrieving and compiling each stock's adjusted close price using Quandl's API

The second Python script was created to automate the retrieval of new data on a daily basis, instead of retrieving a year's worth of data every time. It is set up the same way as the first script that was created, except that it would only retrieve the data for the day that it has been executed on, instead of the whole time period. It then appends the data for that day to the end of the MySQL database. To automate this process, the virtual machine server that was set up using Amazon Web Services was configured to contain a virtual Linux system that came with Python already. The reason for choosing a Linux system is that Linux comes with a utility called Cron that allows the user to schedule commands or scripts to be run automatically at a certain time and date (HostGator, 2017). The Python script that was created to be run automatically was then uploaded onto the virtual server, and then using Cron, was scheduled to be run every day at 11:30pm. To do this, a Cron job was setup using the Crontab interface which can be accessed through the Linux terminal. Cron jobs take time and date parameters followed by the action or script that you want it to run at this time. 23:30pm was chosen because financial data sources like Quandl can take some time to update the current day's data, so a late evening time was chosen to account for the possibility that it may take longer. Once that had been set up, the database will be automatically updated every day with each stock's adjusted closing price.



```

ec2-user@ip-172-31-20-92:~
GNU nano 2.5.3 File: /tmp/crontab.iZjbks Modified
30 23 * * * python "Automatic Retrieval.py"
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

```

Figure 9 - Setting up the Cron job.

Testing

The retrieval system acts as a separate system to the asset allocation system, so separate testing was done for this on its own. As the future parts of the artefact require historical stock pricing data, the most important outcome of this iteration that must be achieved; is a system that retrieves adjusted close price data in the correct format which can then be used for later iterations. The functionality that needs to be verified can be split into two separate components:

- Ensuring that data is retrieved from Quandl for each stock and that it is in the desired format.
- Confirming that the database is being populated with data.

As the testing required for these two parts of the system is a fairly simple matter of verification, a simple black-box test was performed to check that the results from system match the expected results. Appendix A. 1 is a table created to facilitate the results of the black-box testing, providing description of the function being tested, the expected result and the actual result.

4.3 Stock Analysis and Visualisation

Now that pricing data has been collected pertaining to each stock in the list, the next step is to provide descriptive statistics for both the user, and to use later on as an input for the Markowitz model. This will form the beginning of what will eventually become the full Python application. Some of the main statistics of a stock that an investor would look to assess its performance are its expected return and the risk associated with it. A common metric used to describe a stock's expected return is the mean of its periodic returns. This project is using one year's worth of data which is a relatively small amount, so a stock's returns are going to be recorded daily. By assuming they are normally distributed, the risk of an asset can also be derived using daily returns. This is done by calculating the standard deviation or variance of the returns. In this iteration, any visual output pertaining to individual stocks will also be done here.

Design

Due to the objectives of this iteration being fairly linear, there wasn't that much pre-implementation design that was done. As this iteration focuses on analysing, calculating and displaying statistics of stocks individually, it made sense to apply some of Python's object-oriented capabilities and treat each stock as its own object through the creation of a Stock class. This would provide the ability to centralise every piece of information that is relevant to that particular stock. This is useful for a number of reasons: it makes the storage and handling of each piece of data for every stock a lot less overwhelming and messy, while making them a lot easier to access and call. The design idea for the stock class can be seen in Figure 10 through the use of a class diagram. It shows the initial conceptual idea of what attributes and functions would be contained within that class.

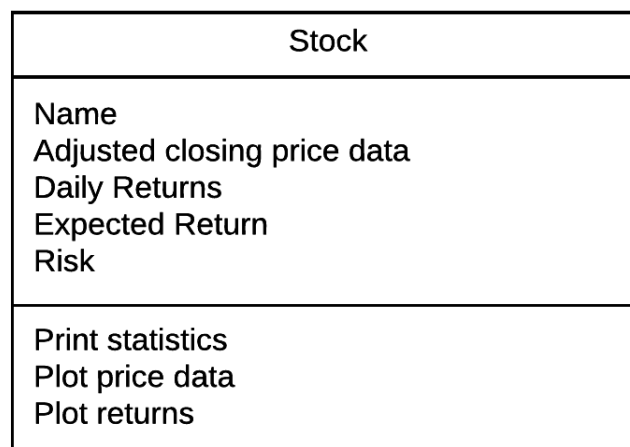


Figure 10 - Conceptual stock class design

Implementation

- Create a stock class to consolidate all data and methods associated with individual stocks.
- Calculate and store daily returns of each stock using its adjusted closing price.
- Calculate estimations for a stock's risk and expected return by calculating the mean and variance of the daily returns.
- Provide appropriate visualisation of data and statistics.

Upon running the application, all the adjusted closing price data is retrieved. The constructor for the stock class that has been created requires a valid name only. The list of stock names is then

iterated through and a Stock object is created for each of them. When they are initialised they are given attributes for the name of the stock and the appropriate closing price data that goes with it. This data is then used to calculate other pieces of information which form the remaining attributes of the object.

Daily Returns

Using the pricing data that was retrieved, a series of historical returns can now be calculated for each stock. The rate of return of an asset can be described as the percentage change in value between two time periods. As the data that has been retrieved in this project is the price valuation of the stock at the end of each trading day, the rate of return is used to describe the change in stock price between each trading day. In this project, a stock's rate of return denoted by R_t is the current day's price P_t minus the previous day's price P_{t-1} divided by the previous day's price. This gives the percentage change in price over time period t .

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \% \Delta P_t \quad (7)$$

The array of daily closing price data is iterated through to calculate the rate of return for each day. Instead of using arithmetic rates of return, what is often used instead is the log returns. Returns are usually assumed to be normally distributed, but when compounding arithmetic returns, it becomes unsymmetrical (Morgan, 2013). To offset this error, log returns are used by taking the natural log of P_t instead. These log returns are stored as an attribute of the stock object in a NumPy array. NumPy is a scientific computing module for Python that contains a more sophisticated array object that can perform a variety of mathematical functions (NumPy, 2017).

ibm	
Date	
2015-01-02	0.000000
2015-01-05	-0.015860
2015-01-06	-0.021802
2015-01-07	-0.006557
2015-01-08	0.021502
2015-01-09	0.004346
2015-01-12	-0.016923
2015-01-13	0.002362
2015-01-14	-0.006462

Figure 11 - Generated time series of returns

Expected Return and Risk

The expected return and risk of investment into a stock can be calculated using the historical returns of said stock. The expected return of an asset can be calculated by taking the arithmetic mean of the returns. In this case, A denotes the expected return of the stock, $a_{1...n}$ denotes each of the daily returns and n denotes the total number of daily returns there are.

$$A = \frac{1}{n} \sum_{i=1}^n a_i = \frac{a_1 + a_2 + \dots + a_n}{n} \quad (8)$$

The variance or standard deviation of the daily returns is a commonly used metric for describing the risk associated with investment in an asset. The variance of a stock is denoted by σ^2 with χ_i

representing each daily return. N denotes the number of daily returns and μ denotes the arithmetic mean of those returns. To achieve the standard deviation, one just has to square-root the variance.

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2 \quad (9)$$

Both of these operations were fairly simple to do through the built-in functionality of NumPy. NumPy contains pre-defined methods for calculating the arithmetic mean, variance and standard deviation. All of these values are then made attributes of the relevant stock object, making them a lot easier to access again for analysis and visualisation.

```
mean = numpy.mean(self.returns)

variance = numpy.var(self.returns)

standardDeviation = numpy.std(self.returns)
```

Figure 12 - Example use of NumPy's built-in functions

Visualisation

Visualisation of assets and their properties is an important part of the overall system as it provides a lot of information which can help the user make investment decisions. To create graphical visualisations of each stock's pricing data and returns, the Matplotlib module was used. Matplotlib is a 2D plotting library which provides easy plotting of data due to its compatibility with a lot of data types such as time-series data which is used here. It has a lot of customisation potential and has good synergy with other modules used such as Pandas, because it handles the Pandas dataframe structure well, allowing multiple columns of data to be plot against the dataframe's index in one function. It also provides an interactive toolbar which can be used to zoom in on or pan around the canvas in which the data that has been plotted (Droettboom, 2017). This is especially useful in this project as the time period for the data is in days which provides a lot of small points on the graphs which can be hard to see. Visual observations were used during development to verify that the functionality of the graphical visualisations were working correctly. Examples of these graphs are shown below:

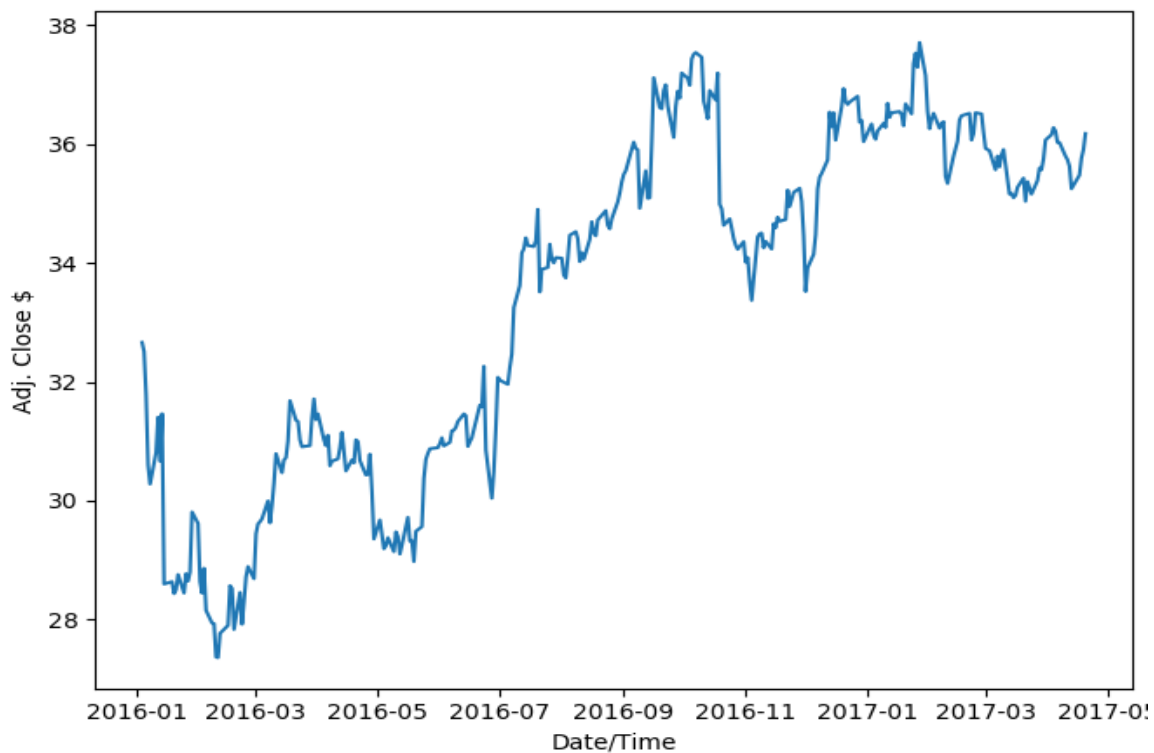
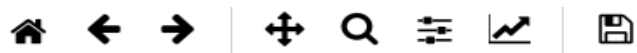


Figure 13 - Example of generated adjusted closing price graph

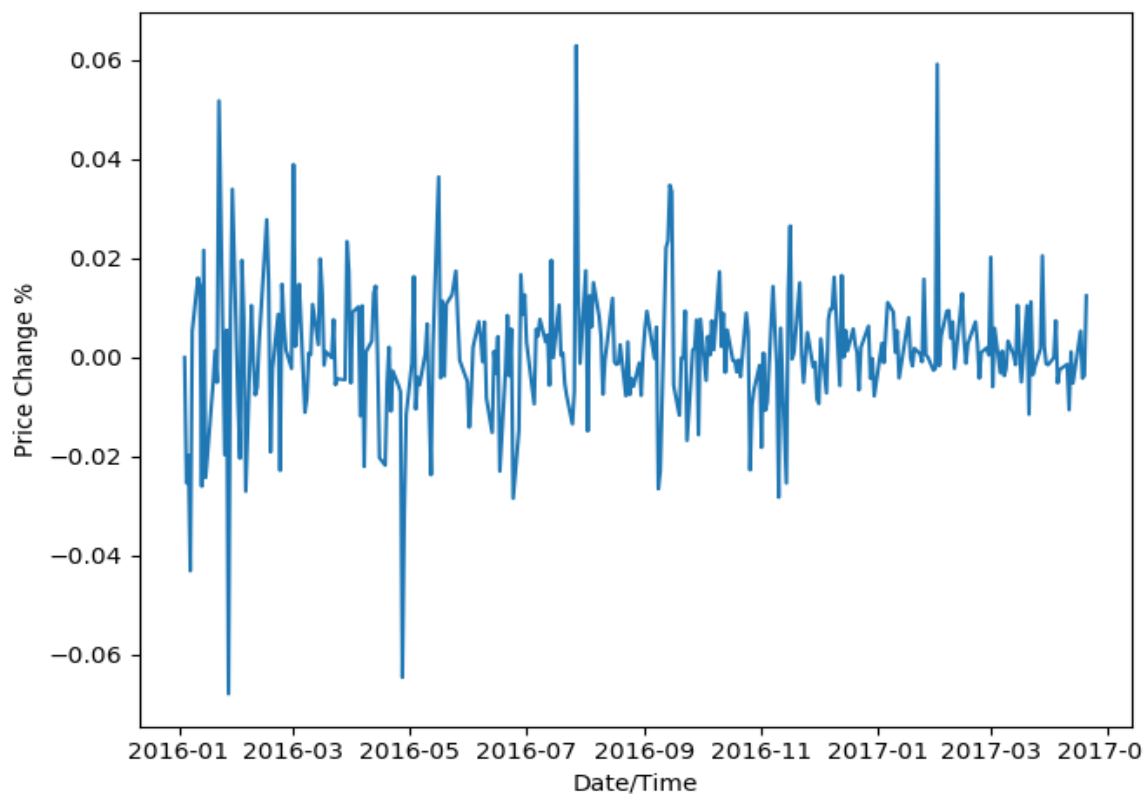
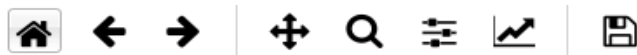


Figure 14 - Example of generated daily returns graph

4.4 Portfolio Creation & Optimisation

This iteration focused on the implementation of the Markowitz portfolio and optimisation model. A portfolio is a grouping of financial assets, or more specifically in this case; stocks. Each portfolio can contain any number of stocks that each have their own weighting. Given a budget that is going to be used to invest, the weighting of an asset in a portfolio corresponds to how much of that budget is going to be invested in that particular stock. The process of choosing the weighting for each asset is called Asset Allocation, which forms the backbone of the problem this project tries to solve using Markowitz's portfolio theory. Like each asset, each portfolio will have its own expected return and risk that are calculated using methods that incorporate an asset's weighting as well as individual risk and return values.

Design

Similar to what was done with each individual stock, creating a new portfolio class meant that every piece of information to do with that specific portfolio would be centralised as an attribute. Each portfolio object could then have its own method for each type of optimisation that could be performed, be it minimum variance, or maximum return. The basis of implementation for calculating each portfolio statistic necessary for performing optimisation is the work laid out by Markowitz himself. Due to the underlying theory being established already, minimal design was needed for this iteration except for the portfolio class structure. The arguments required to initialise the portfolio class were a list of stock names that would be used as the collection of assets that form a portfolio. As each stock object is already initialised when upon retrieval of the stock data, only the name of the stock would need to be passed as an argument to the portfolio object constructor instead of the stock object itself. This makes it so the data needed from each stock object can then be accessed by reference. Each portfolio object would then have its own list of attributes pertaining to relevant statistics that may need to be recalled later. A conceptual example of what the portfolio class could like using a class diagram:

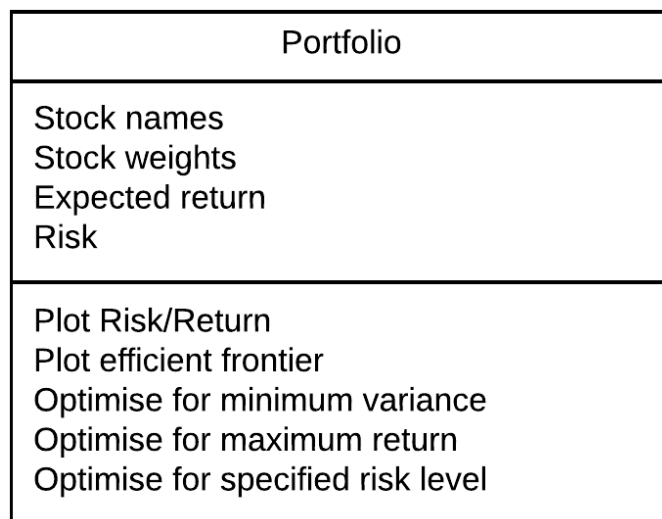


Figure 15 - Portfolio class diagram

Implementation

- Create a portfolio class to consolidate information and all functions that specifically affect that portfolio.

- Calculate risk and return for a portfolio
- Calculate and plot the Efficient Frontier
- Perform various optimisations of portfolio weights:
 - Minimum Variance
 - Maximum Return
 - Specified Risk Tolerance

To construct a portfolio object, a list of stock names needs to be provided to the constructor. Upon construction, equal weightings which sum to 1 are given to each of stocks in the portfolio. A new dataframe is created to store the compiled returns of each stock included in the portfolio. This will be used to estimate covariance between the assets which is necessary input to the Markowitz optimisation model.

	googl	ibm	aapl	msft
Date				
2015-01-02	0.000000	0.000000	0.000000	0.000000
2015-01-05	-0.019238	-0.015860	-0.028576	-0.009346
2015-01-06	-0.024989	-0.021802	0.000094	-0.014678
2015-01-07	-0.002945	-0.006557	0.013925	0.012625
2015-01-08	0.003478	0.021502	0.037703	0.028994
2015-01-09	-0.012286	0.004346	0.001072	-0.008441
2015-01-12	-0.007336	-0.016923	-0.024949	-0.012581
2015-01-13	0.009491	0.002362	0.008840	-0.005271
2015-01-14	0.008197	-0.006462	-0.003818	-0.008667

Figure 16 - Compiled returns for stocks in portfolio

Expected Return for a Portfolio

The mathematical model of Markowitz's Portfolio Theory estimates the expected return of portfolio in an intuitive way. The Markowitz model calculates the expected return using the weight and expected return of each asset in a portfolio. In this case, the expected return of the assets is estimated using arithmetic mean of its historical returns that were calculated earlier. The expected return of a portfolio $E(R_p)$ is calculated using the following, where x_i and $E(R_i)$ denote the asset's weight and expected return, respectively.

$$E(R_p) = \sum_i x_i E(R_i) \quad (10)$$

As an example, the expected return of a portfolio containing two assets, A and B can be described as:

$$E(R_p) = x_A E(R_A) + x_B E(R_B) \quad (11)$$

However, when the number of assets gets a lot higher, the length of the formula can become excessive. It can be described in a much neater and concise way using matrix notation instead of standard. This also makes it a lot easier to implement using Python as all the asset values can be stored in matrices, allowing the use of matrix functions such as those from NumPy to perform

the necessary calculations. For a portfolio that contains three assets; A, B and C, the expected return can be described using the following, where x and R are the weight and return of each asset.

$$E(R_p) = (x_A, x_B, x_C) \cdot \begin{pmatrix} R_A \\ R_B \\ R_C \end{pmatrix} \quad (12)$$

This is also described as the dot product of the stock weights and the corresponding returns. Upon portfolio creation, the weights and expected return of each asset are already contained within an array, making this procedure simple to do using NumPy's dot product method.

Risk for a Portfolio

If we consider the same three asset portfolio, the variance or risk of that portfolio can be described by the following, where x and Σ denote the weights of each asset and the covariance between each asset. For example, the covariance of asset A and B is denoted by σ_{AB}^2 . Each asset will have its own covariance with every other asset, which make up a covariance matrix.

$$\sigma_{p,x}^2 = x^T \Sigma x = (x_A, x_B, x_C) \cdot \begin{pmatrix} \sigma_A^2 & \sigma_{AB} & \sigma_{AC} \\ \sigma_{AB} & \sigma_B^2 & \sigma_{BC} \\ \sigma_{AC} & \sigma_{BC} & \sigma_C^2 \end{pmatrix} \begin{pmatrix} x_A \\ x_B \\ x_C \end{pmatrix} \quad (13)$$

The covariance matrix for each portfolio was calculated using one of NumPy's methods as seen below, to which the required input argument was a list of returns such as those shown in Figure 16. The covariance matrix for that particular portfolio was then stored as an attribute of the portfolio object.

```
self.covarianceMatrix = numpy.cov(self.returns)
```

Figure 17 - NumPy method for covariance

```
[ 2.13e-04  7.29e-05  9.10e-05  1.26e-04]
[ 7.29e-05  1.57e-04  7.72e-05  9.24e-05]
[ 9.10e-05  7.72e-05  2.28e-04  1.13e-04]
[ 1.26e-04  9.24e-05  1.13e-04  2.29e-04]
```

Figure 18 - 4 asset covariance matrix

Efficient Frontier

There are almost a limitless amount of different weighting combinations that sum to 1 that can be given to the assets in the portfolio. Each of these different weightings will produce a different risk and expected return for the portfolio. One way of constructing the efficient frontier is to model it as a minimisation problem, with the idea being to minimise the amount of risk for a

series of given expected returns, as detailed in Section 2.1. Similar to Equation 4, the same objective function of portfolio risk is created except that there are n assets and matrix notation is used. This is described by:

$$\mathbf{x}^T \Sigma \mathbf{x}$$

Any optimisation problem will also have a series of constraints that the optimiser will have to adhere to. The same constraints in Equation 4 are used here to plot the Efficient Frontier, however an additional constraint is added to ensure that all the asset weights are positive. Something that the original Markowitz model allowed was the act of short selling stocks. Short selling is the process of selling a borrowed asset in hope that it will decline in price and can be rebought later for a much cheaper price (Investopedia, 2017). This system that has been created doesn't facilitate short selling, so the constraint that keeps the asset weights positive is introduced to stop it. Given that, the three constraints can be the following:

$$\sum_i x_i = 1 \qquad E(R_p) = \mu \qquad x_i \geq 0$$

The whole minimisation problem can be described as:

$$\text{Minimise: } \mathbf{x}^T \Sigma \mathbf{x} \qquad (14)$$

$$\text{Subject to: } E(R_p) = \mu \quad (14.1)$$

$$\sum_{i=1}^n x_i = 1 \quad (14.2)$$

$$x_i \geq 0 \quad (14.3)$$

This is what is described as a quadratic program, which is an optimisation problem that contains a quadratic objective function and linear constraints. With the introduction of the new constraint, the lagrange multipliers used to represent that constraint can become very excessive, so a Python module called CVXOPT was used. It is a software package for convex optimisation that can be used to model minimisation problems such as these. It provides an easy creation process for arguments as well having good performance for solving the problems (Andersen and Vandenberghe, 2016). To use the CVXOPT module, the problem has to be in the correct quadratic form accounting for the use of lagrange multipliers according to the documentation. The inequality constraints are described by $G\mathbf{x} \leq \mathbf{h}$ and the equality constraints are described by $A\mathbf{x} = \mathbf{b}$.

$$\text{Minimise: } \mathbf{x}^T P \mathbf{x} + \mathbf{q}^T \mathbf{x} \quad (15)$$

$$\text{Subject to: } G\mathbf{x} \leq \mathbf{h} \quad (15.1)$$

$$A\mathbf{x} = \mathbf{b} \quad (15.2)$$

When first calculating the efficient frontier using Python, an array of expected return values is produced to act as the expected return constraint that the risk is going to be minimised for. Roughly 200 of them are generated and distributed appropriately in order to form a line of portfolios that will make up the efficient frontier. The constraints that were stated earlier are transformed into matrix form for input into the optimiser. All 200 of the desired expected returns

are iterated through and use as a constraint in the optimiser to return 200 different asset weighting combinations. These 200 portfolios are efficient as they provide the lowest amount of risk for their specified returns. This method is then used for any portfolio to graphically visualise its Efficient Frontier.

```
def efficientFrontier(self):
    n = len(self.returns)
    returns = numpy.asmatrix(self.returns)
    N = 200
    mus = [10**(5.0 * t/N - 1.0) for t in range(N)]

    P = self.covarianceMatrix
    q = cv.matrix(numpy.mean(returns, axis = 1))
    G = -cv.matrix(numpy.eye(n))
    h = cv.matrix(0.0, (n,1))
    A = cv.matrix(1.0, (1,n))
    b = cv.matrix(1.0)

    portfolios = [cv.solvers.qp(mu*P, -q, G, h, A, b)['x'] for mu in mus]

    returns = [blas.dot(q, x) for x in portfolios]
    risks = [numpy.sqrt(blas.dot(x, P*x)) for x in portfolios]

    return risks, returns
```

Figure 19 - Optimisation using CVXOPT

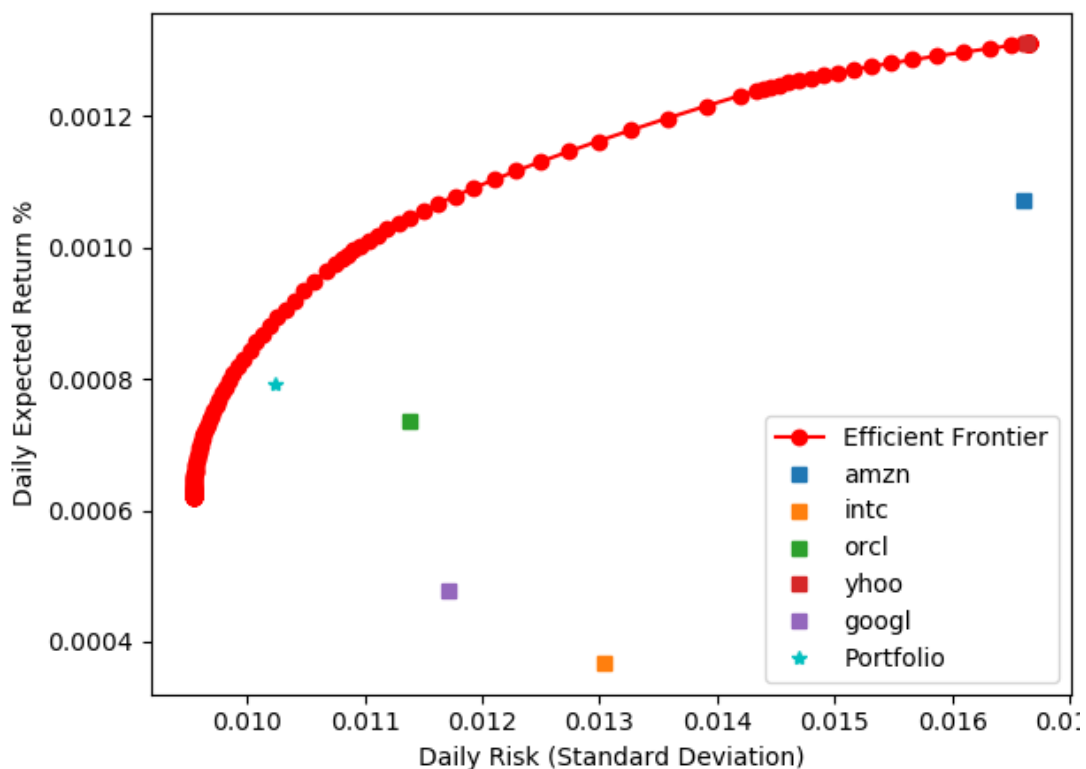


Figure 20 - Generated Efficient Frontier

Minimum Variance

All investors have a different goal when creating a portfolio. Some are more averse to risk and prefer a portfolio with the lowest risk possible. Others might prefer one that gives a medium amount of return and risk, and some might want the portfolio with a very high return and risk. To cater to these preferences, different types of optimisations were implemented in order to achieve certain goals. The same method that was used to create the efficient frontier can be used to achieve other optimisation goals, such as minimum variance. The only differences are the constraints that are being used in the optimisation. To perform find the minimum variance portfolio, a new problem minimisation problem was set up with one different constraint from before. The portfolio is no longer being optimised based on a given return, and instead is just being minimised globally, so the initial previous constraint is removed, as shown by:

$$\text{Minimise: } \mathbf{x}^T \Sigma \mathbf{x} \quad (16)$$

$$\text{Subject to: } \sum_{i=1}^n x_i = 1 \quad (16.1)$$

$$x_i \geq 0 \quad (16.2)$$

The result of this is one set of asset weights which corresponds to the portfolio with the lowest variance possible for that set of assets. Figure 21 shown below depicts a portfolio marker for the minimum variance point.

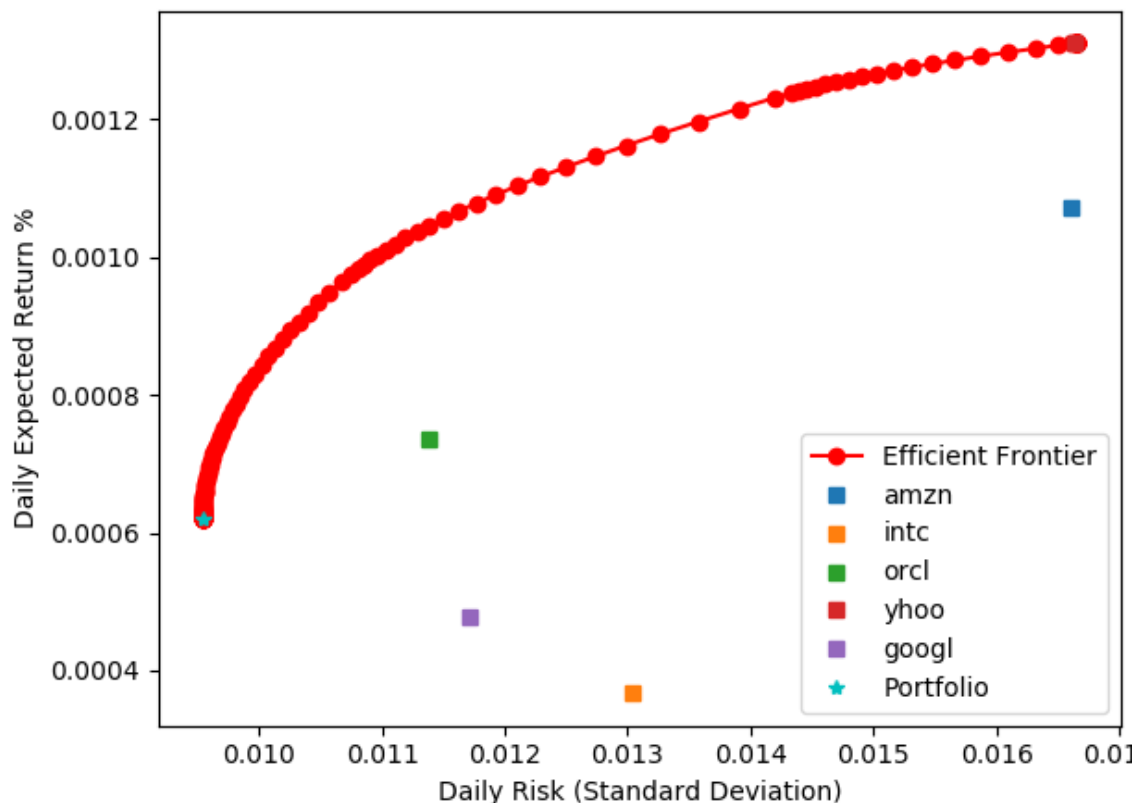


Figure 21 - Generated Minimum Variance Portfolio

Maximum Return and Specific Risk Aversion

A slightly different approach was taken to obtain a maximum return portfolio as well as one that factors in an investor's risk aversion. When plotting the Efficient Frontier for a portfolio, a series of efficient portfolios are generated and their weights are stored in a list. The list is sorted in an ascending order based on the generated expected returns from before, so the portfolio weights with the lowest risk are stored at the beginning and the portfolios with the highest risk are stored at the end. Using these portfolios, a custom utility function was created that attempts to quantify an investor's happiness denoted by H with a certain portfolio's risk and return based on their given level of risk aversion denoted by a :

$$H = E(R_p) - 0.5a(x^T \Sigma x)^2 \quad (17)$$

This is used to generate a H value for every portfolio that was generated when creating the Efficient Frontier. These values are then put in their own list in the order that they were created. The index value of the maximum H value is taken and used to choose a portfolio from the 200 generated ones that reflects the investor's risk aversion level. The H value will scale according to the possible risk levels of that specific portfolio. If a portfolio only contains assets that are extremely risky, then it will be more reluctant to choose one of the higher risk portfolios. Figure 22 shows a portfolio generated using a risk aversion level of 5.

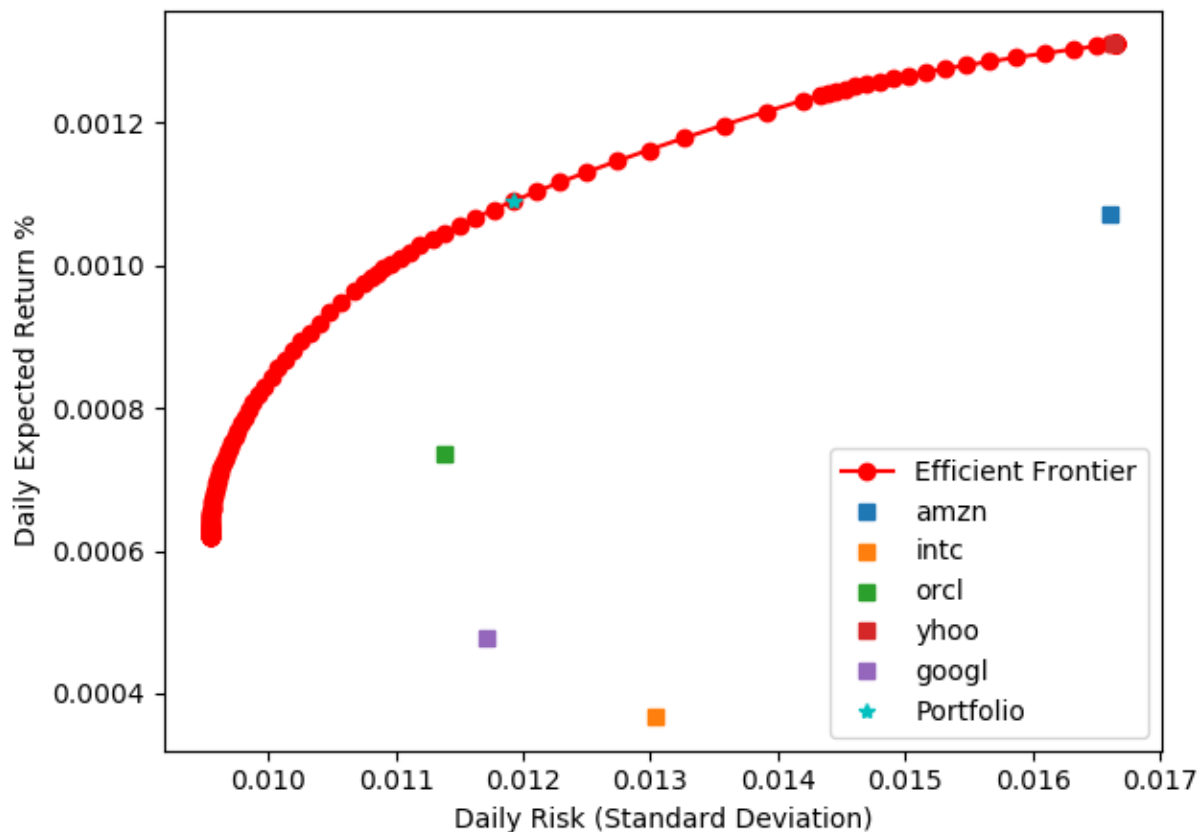


Figure 22 - Portfolio with a Risk Aversion level of 5

The use of the 200 generated portfolios from the efficient frontier is extended to obtaining the portfolio that realistically has the maximum return in an extremely simple way. As the list of portfolios that form the frontier are stored in ascending order according to return, the portfolio that is at the end is chosen to represent the portfolio with maximum return.

```

def personalPort(self, riskAv):
    n = len(self.returns)

    N = 100
    mus = [10**(5.0 * t/N - 1.0) for t in range(N)]

    P = self.covarianceMatrix
    q = self.average
    G = -cv.matrix(numpy.eye(n))
    h = cv.matrix(0.0, (n,1))
    A = cv.matrix(1.0, (1,n))
    b = cv.matrix(1.0)

    portfolios = [cv.solvers.qp(mu*P, -q, G, h, A, b)['x'] for mu in
mus]

    tempReturns = [blas.dot(q, x) for x in portfolios]
    tempRisks = [numpy.sqrt(blas.dot(x, P*x)) for x in portfolios]

    utilitys = []

    for i, n in enumerate(tempReturns):
        utilitys.append((tempReturns[i] - (0.5 * riskAv *
tempRisks[i]**2)))

    maxIndex = utilitys.index(max(utilitys))
    solution = portfolios[maxIndex]
    self.weights = solution

```

Figure 23 - Portfolio Optimisation using a Risk Aversion level

4.5 Creation of the GUI

Up until this point, each iteration focused on implementing functionality in order to perform mean-variance analysis and optimisation. This iteration focused on the creation of a GUI in the form of a desktop application that was going to be used to feature and accommodate all the functionality that has been created in previous iterations.

Design

When it came to designing the graphical user interface, even though the GUI is not the central part of the project, there were still certain user interface design principles that were adhered to when designing the look of the GUI. Porter (2017) states that clarity is the most important part of a GUI. As well as that, there should be one primary action per screen. To take this into consideration, the parts of the application that needed the most input from the user were separated into different windows that would spawn from a main window. This would help prevent the user from becoming overwhelmed as all the functionality that requires the user's input would be in its own separate place. Going further, the visualisation of outputs to the user is the most important part of this system. To focus on this, the overall size and space used for the process of interacting with the system was kept to a minimum. This meant that the graphical and statistical visualisation could take priority as the most visible feature of the application.

To streamline the design process of the GUI, the available functionality of the system was separated into 4 main parts which would form the main components of the system that would then need to be accommodated by the GUI.

- Portfolio visualisation and optimisation
- Portfolio creation via stock selection
- Individual stock analysis and visualisation
- Visualisation of stocks together

A very basic conceptual idea of functionality and features that each window of the GUI will have is shown by the following class diagram. It will make use of the Stock and Portfolio class that were created already to perform and handle all the functions necessary.

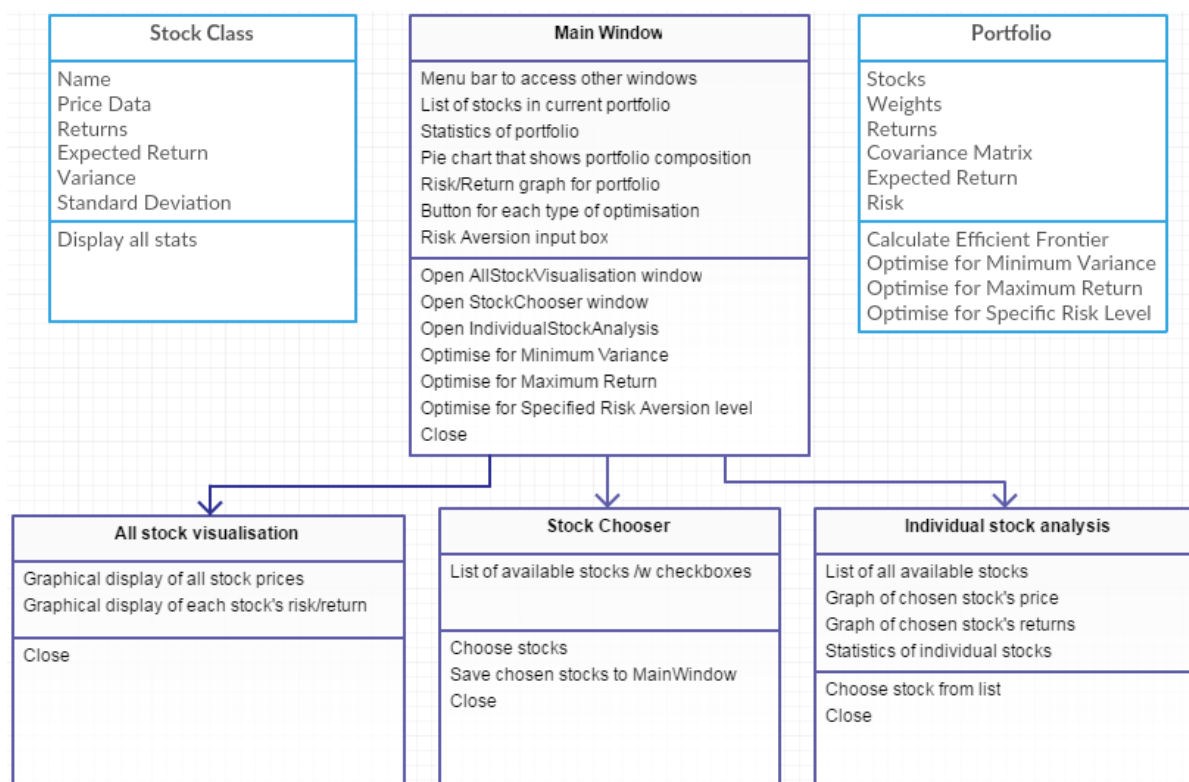


Figure 24 - Conceptual GUI class diagram

Implementation

Writing out the visual layout and general aesthetics of the application through Python code alone, is a very cumbersome task, more so if the full design isn't completely finalised. As mentioned before, Qt Designer (The Qt Company, 2017) was used to streamline the process of creating and laying out each window in the entire application. Qt Designer allows the user to create application windows using Qt Designer's pre-built widgets and containers in a drag and drop fashion. One of the most important widgets used in the application is the Matplotlib widget. It acts as a canvas for displaying graphical outputs. Once all the features have been added and laid out properly within the Qt Designer interface, it can then generate a .ui file which contains code for all the aspects that have just been created. Each separate window that is going to be part of the application will have its own .ui file and class within the application script. The .ui file for each class is imported into the Python application script and assigned to the appropriate class. From here, the objects generated for each part of the interface just needed to be paired to the desired functionality.

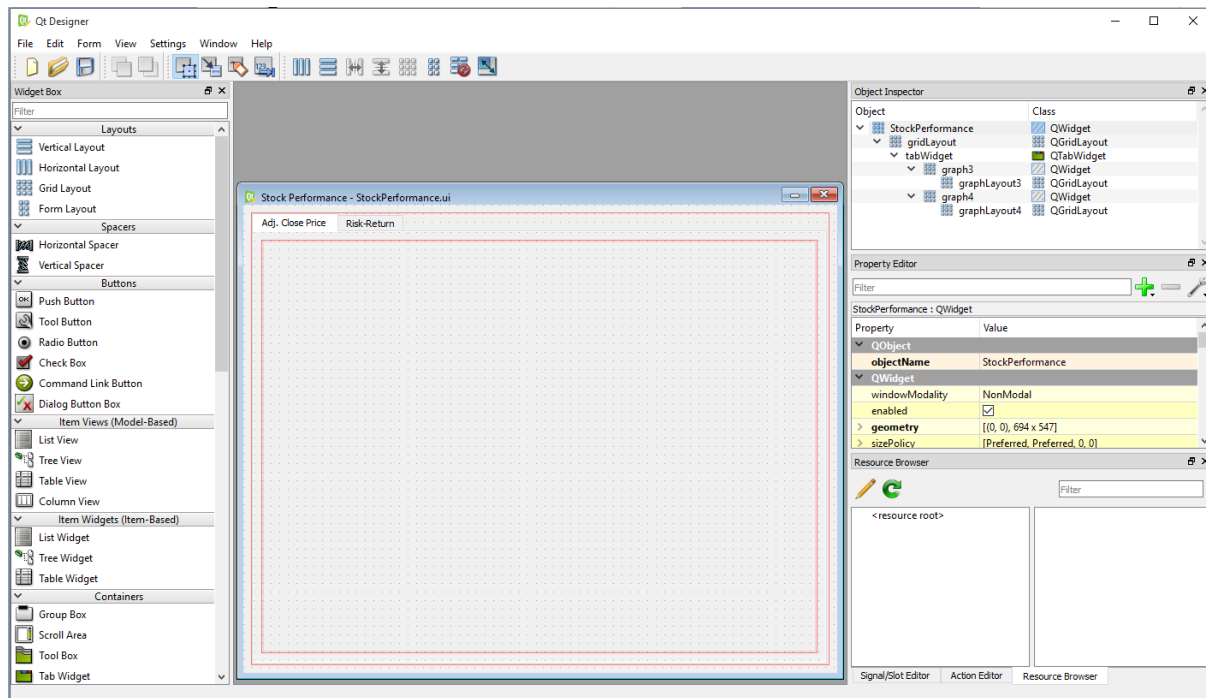


Figure 25 - Qt Designer interface

Main Window

Upon initialisation of the program, the main window is the first to appear. As shown in Figure 24, four different windows are created to house the analysis and optimisation functionality created in the previous iterations. The GUI consists of a main window and three smaller windows that can be opened from the main window. The main window primarily features a Matplotlib canvas that is used to capture the graphical outputs of the portfolio functions such as those detailed in Figure 24. Whenever a new portfolio is created, this canvas is updated to show each individual stock's risk and return as well as the Efficient Frontier generated for that particular portfolio. The risk and return of the portfolio is also displayed, which is updated when an optimisation is performed.

The main window also features a list widget that contains each stock that has been chosen for the portfolio, while also providing numerical values for the portfolio's current risk and return. The weights of each asset are also displayed numerically, accompanied by a pie chart showing the asset composition of the portfolio. Buttons are used to handle the input of the user for what optimisation to perform, as well as a text box that is used to receive input from the user about their risk aversion level. The visuals of these buttons are generated by Qt Designer itself, so pairing the press of the button to the appropriate optimisation function found within the Portfolio class needed to be done manually.

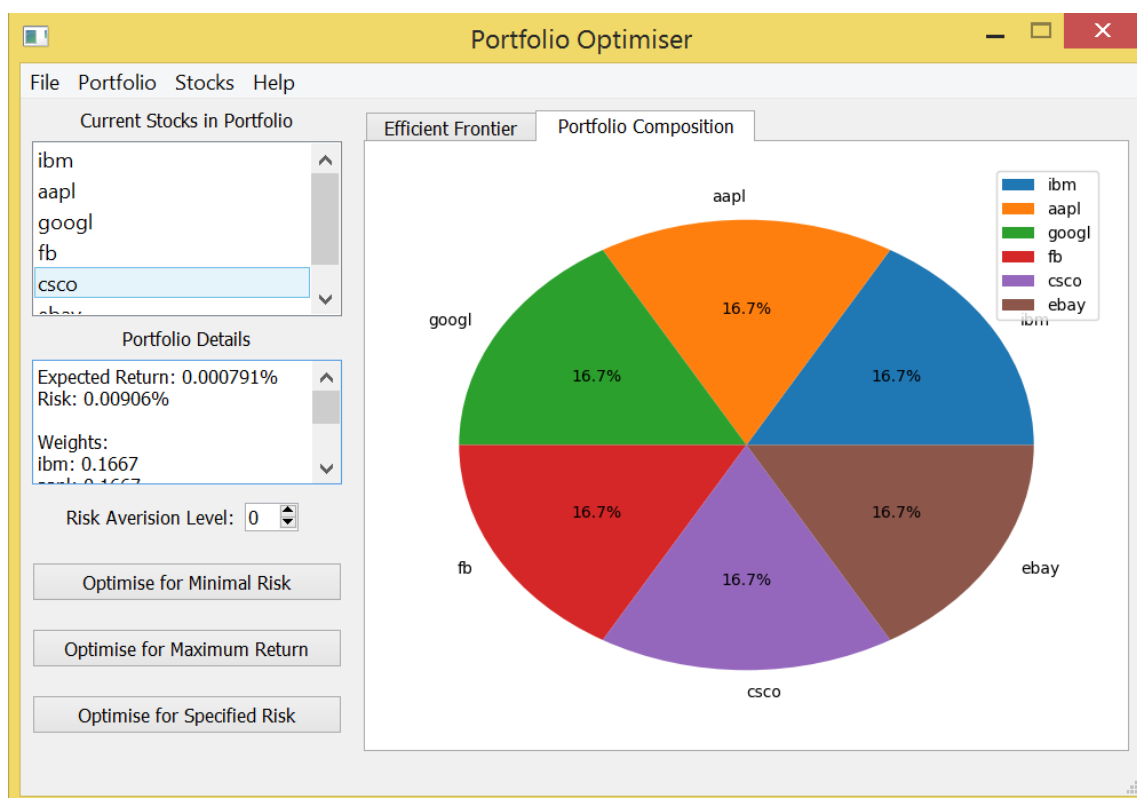


Figure 26 - Main Window

Stock Chooser

To choose the stocks that the portfolio is going to be composed of, a new smaller window was created. It is extremely simple in that its only features are a list widget and a button. The list widget contains all of the available stocks that have data pulled from Quandl. Each of them have a checkbox alongside it so that the user can check the stocks that they would like to include in the portfolio. The button handles the creation of the portfolio. When it is clicked, an instance of the Portfolio class is created using the stocks that have been checked and the Stock Chooser window is closed.

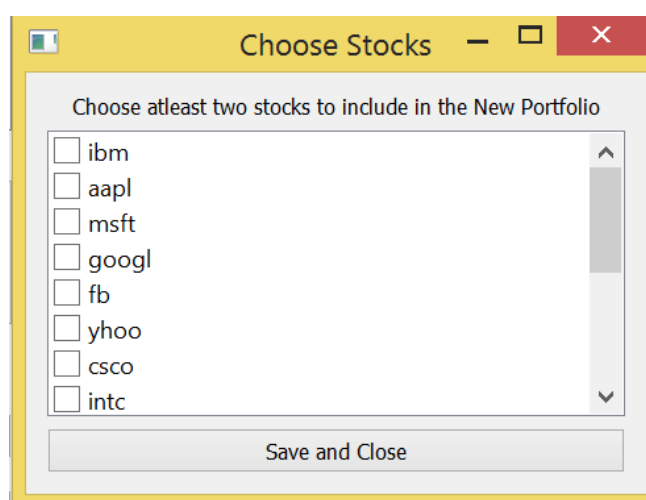


Figure 27 - Stock Chooser window

Stock Performance

To aid the user in choosing stocks for their portfolio, a window was created to visualise the performance of every stock together. Stock data that could be particularly useful when plotted against one another were the closing prices of each one, and the risk and return. To situate this, two more Matplotlib canvases were used to display each of the graphs. As the window didn't provide any extra functionality outside of displaying those graphs, they were displayed upon the opening of the window.

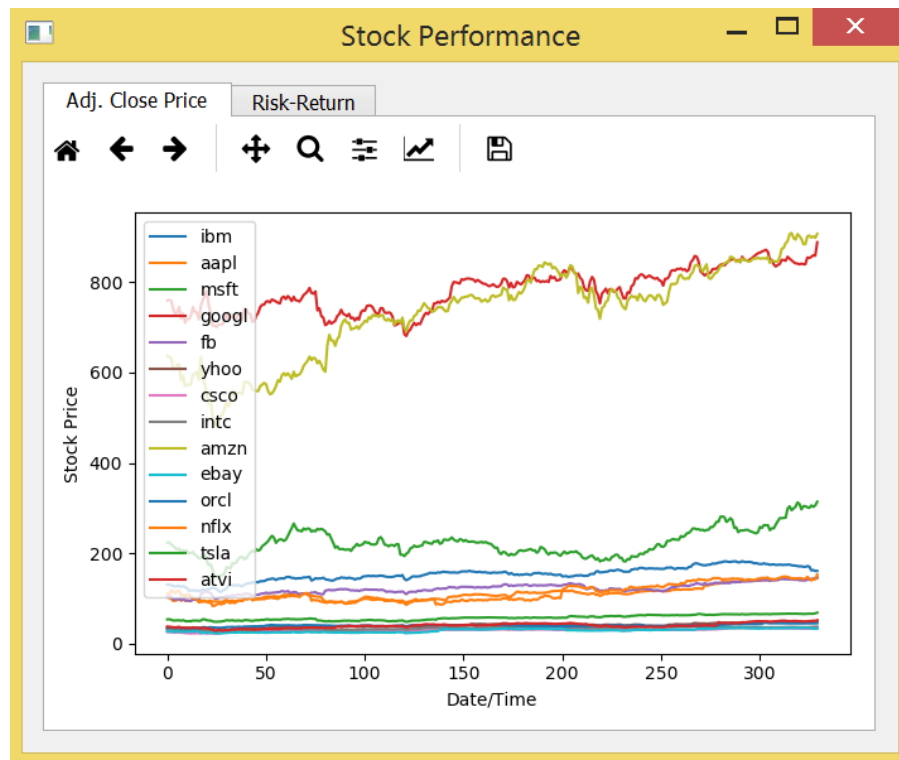


Figure 28 - Stock Performance

Stock Analysis

To provide the user with a more in-depth knowledge about each stock, another window was created to facilitate the graphic and statistical information pertaining to each stock. In conjunction with the Stock class that was created in the previous iterations, a list widget was created to store all the stocks that were available. Functionality was added to the list so that when a stock in the list is selected, its attributes would then be accessed to obtain pricing and return data specific to that stock which was then displayed using a Matplotlib canvas. A text box was also used to contain numeric description of the selected stock's expected return, standard deviation and variance. This allows the user to then assess the performance of each stock before perhaps adding them to a portfolio.

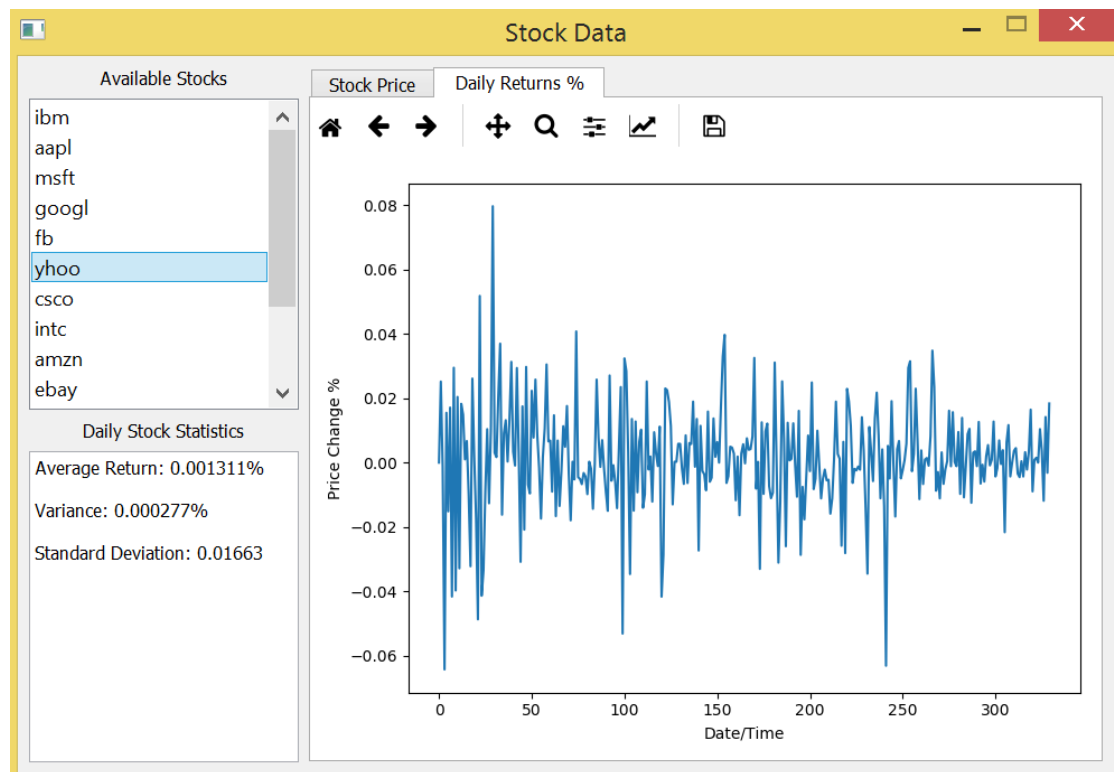


Figure 29 - Stock Analysis window

Testing

During the entire development of the artefact, testing was performed on an expectation and evidence basis to ensure that each component of the system's functionality was producing the correct results. For the sake of the user, it is important that the main functionality of the system has been integrated properly into the encompassing graphical user interface. A common and effective method for testing functionality through the scope of an interface is black-box testing, as it mimics the user's level of knowledge of how the system works (Malaiya, 1995). To apply this to the created artefact, a series of test cases are created to fully test the application. These tests are then performed on the application, comparing the expected result to the actual result. The results of which are show in Appendix B. 1. All of the passed at an acceptable level. Several didn't necessarily produce the correct behaviour but it was behaviour that ultimately didn't affect the artefact negatively.

4.6 Artefact Evaluation

Evaluation of the created artefact is important to determine how effective it is as a solution to the initial problem. There are different ways this artefact can be evaluated. How effective it is at creating investment strategies in the form of optimised portfolios and how effective those portfolios are.

To evaluate at how effective it is at creating portfolios, the comparison between what it is expected to do versus what it actually does can be considered. The artefact can be evidenced meeting project aims through the outputs given during the development process. In regards to generating portfolios correctly, Monte Carlo Simulation can be performed to produce the feasible set that a chosen selection of stocks can take. A total of 100,000 different asset weightings are generated and plotted to visualise the portfolio's feasible set. Combining this result with the Efficient Frontier shown in Figure 30, demonstrates that the creation of a portfolio and its Efficient Frontier works as intended as per the curved line described by Markowitz (1952).

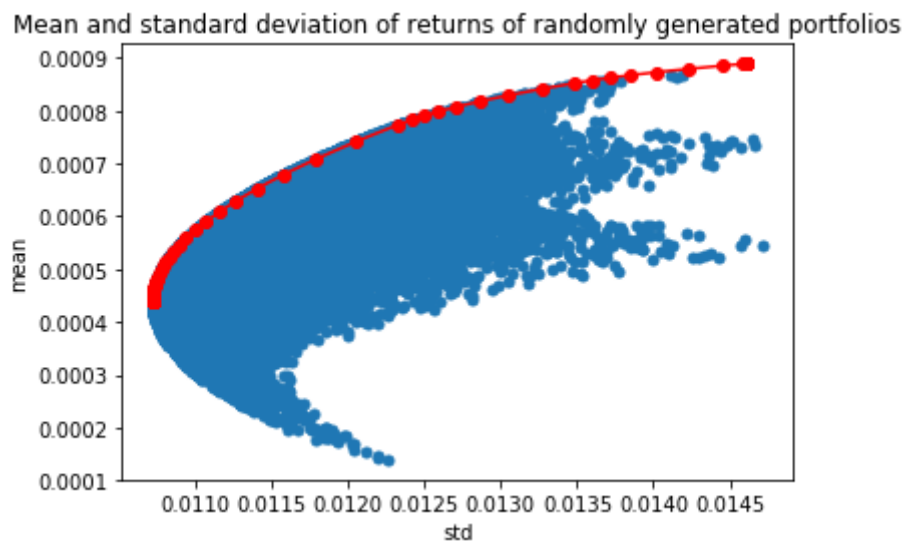


Figure 30 - Monte Carlo Simulation for feasible set

Going further, the functionality of the optimisation procedures that have been implemented can be evidenced through the figures in Section 4.4. Optimisations such as minimum variance optimisation can be shown to work, through the achievement of a portfolio which lies on the minimum variance point as demonstrated in Figure 21. This in conjunction with the other artefact outputs shown in Section 4.4 such as the portfolio in Figure 22 show that the artefact can be an efficient solution to the project aim and provide trading strategies.

Assessing the artefact in a more general sense, the Markowitz model which has been implemented requires a series of prerequisites such as the retrieval and formatting of data before it can perform the mean-variance analysis. The artefact automatically retrieves, handles and displays the acquired data as shown in Figure 11, so the user doesn't have to do this manually. This adds to the solution's effectiveness at achieving the project aim as it forms a more complete package that not only can create investment strategies as per the project aim but also handles the tedious parts such as the process of retrieving and handling data, so that the user doesn't have to manually. The accessibility of the application can also be considered. Potential changes that could be made to the system to further increase accessibility could be a completely different form of user interface. Instead of creating a desktop application to facilitate all the function output,

something such as a webpage would potentially be a better solution. This can provide several advantages over the use of a desktop application as it can be accessed from any system and requires no installation in order to be used.

For financial portfolios, a common way to test the effectiveness of the portfolio is to perform backtesting. Backtesting is the process of testing a created trading strategy in the form of a portfolio on historical data. This simulates the performance of a portfolio over a given time period (Investopedia, 2017). In conjunction with backtesting, the portfolio that is created can also be compared against a benchmark that was present during the chosen time period. This gives an insight into the potential performance that can be expected from a created portfolio. To put this into action, a portfolio containing stocks chosen at random and then optimised for minimum risk is created using the project artefact. The composition of the portfolio is shown by Figure 31:

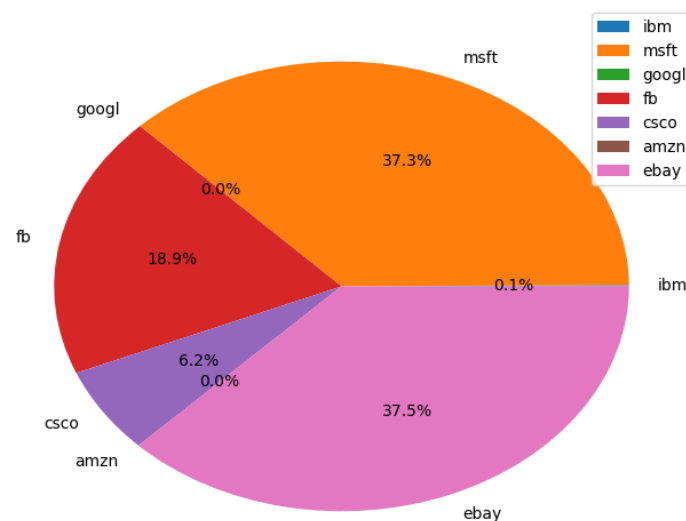


Figure 31 - Evaluation portfolio

It contains a series of stocks with different weightings, some of which are zero, which would effectively exclude them from the portfolio. Using an online portfolio visualiser (Silicon Cloud Technologies, 2017), the portfolio is then backtested to evaluate its performance and growth for over a period of two years from 2015 to 2017. Given an initial investment budget of \$10,000, the return performance of the portfolio can then be compared against an index fund such as the Vanguard 500. Its performance shown below in Figure 32, demonstrates that the portfolio created using the project artefact can be potentially be effective and beat an index fund.

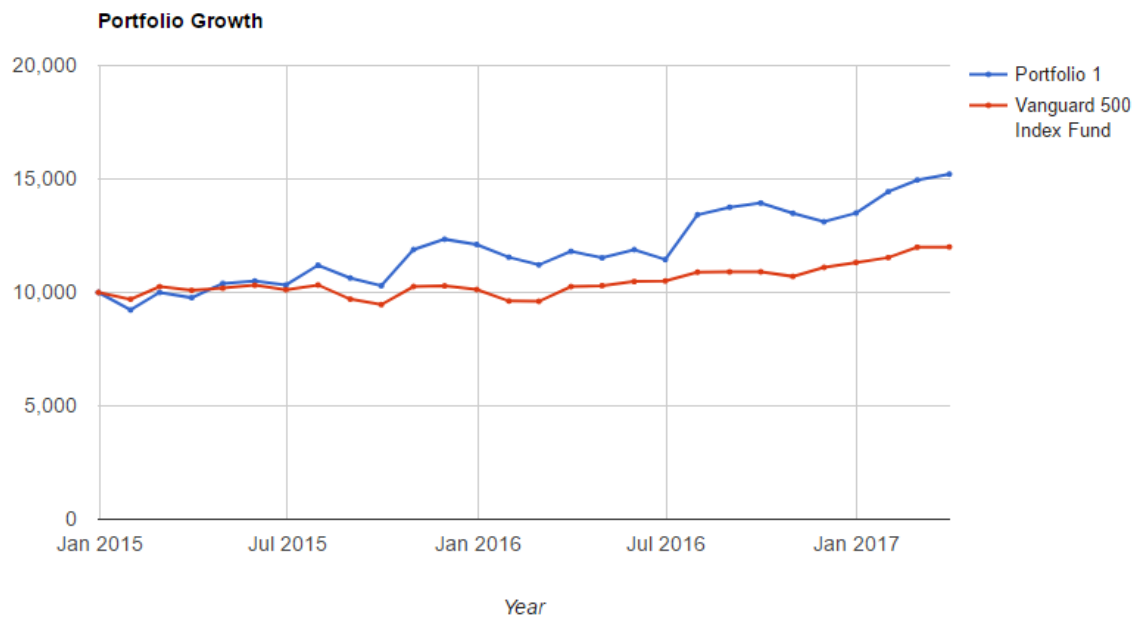


Figure 32 - Portfolio performance

5. Reflective Analysis

With the completion of the project artefact, this chapter will reflect on the project process as a whole. Gibbs's reflective cycle is a popular and well established model for reflection that uses a combination of stages to break down the process of reflective analysis (Gibbs, 1988). It will feature discussion and analysis about things I thought went well with the project as well as things that went bad, providing details about what difficulties were faced and lessons that can be learned from the completion of this project as well as improvements that can be made if it were to be done again.

5.1 Reflection

Before development even begun, a big hurdle initially when starting the project was the knowledge gap surrounding the domain in which the project is set. The aim of the project is to implement a portfolio optimisation system using aspects of Modern Portfolio Theory, which lie within the field of finance. At the beginning of the project, a lot of time was needed to be spent getting familiar with the different concepts that were associated with this particular project. The field of finance is extremely vast and was something that was completely new to me, so realising and estimating the complexity of the required implementation was not easy. In combination with the knowledge gap, I think the initial project goals that were set were too ambitious for my ability, especially in an area that I had no prior knowledge of. This meant that the project goals weren't fully defined and it made planning out the project accurately extremely difficult.

The specific choice and use of tools within the project definitely benefited the project as a whole. The incorporation of things such as version control through the likes of GitHub, made accessing and keeping track of the project much easier and significantly increased workflow. Similar to that, the spontaneous choice to use Qt Designer as a means of creating the GUI was a great help. It vastly increased the speed at which the GUI could be created, which was extremely important at that point in time since it was during the later stages of the project and the deadline was nearing. In terms of the software development methodology that was used to facilitate the project, in retrospect, I think the use of a more traditional, sequential methodology could have been more beneficial to the project and me as a person if the circumstances were right, because it provides a clearer overall plan of the project with stricter deadlines. However, due to the initial circumstances of the project domain being completely new to me and the initial goals being too ambitious, I think the use of the iterative Agile methodology was the correct call for accounting for the uncertainty in the requirements.

Something that I think was done particularly well that was a huge benefit to the project, was the architecture and overall structure of the code. Using the object-oriented principles that are available in Python, I feel I managed to exploit the principles of abstraction and encapsulation in order to accommodate all the necessary functions and data. Having a program structure that was logical and made descriptive sense, made debugging the program a lot more straight forward as well providing an increase in flexibility. In the long run, this made it much easier to integrate the functionality into the GUI, and generally allowed for a much easier time when expanding the program to include new concepts.

Once the project's overall direction had been determined, there weren't any particularly big roadblocks in terms of the progress of the project. One major setback happened at the beginning of the project, a lot of time was spent on the retrieval of data and the implementation of the MySQL database, which was ultimately not included with the final artefact due to maintenance

costs of hosting the MySQL on a cloud server. Exceeding the free usage limits of the cloud service and losing access to the database was something that I anticipated in my risk matrix shown in Appendix A.1 during development and ultimately lead to a lot of time being wasted as the MySQL database was then bypassed, with the data instead being pulled straight from Quandl. This was due to the sheer number of requests being sent to the database during the development process. Had I noticed beforehand, a suitable alternative could have been used while developing and the database could have been saved for the final release.

Although there weren't any major stoppages during development, there were some theoretical concepts that were a challenge to implement using code. The optimisation problem featured within the Markowitz Model isn't very challenging as it stands, but adapting it further to suit specific portfolio goals became a lot more complicated with the introduction of additional constraints. It required the use of mathematical techniques such as lagrange multipliers which is something I had not encountered until that point. To try remedy this, sufficient studying of Markowitz's original theory was done to form a better understanding of how to adapt his work to what was needed.

5.2 Conclusion

Although I would still consider the project to be an overall success as the artefact met the aims of the project, there is a lot that could be improved on in terms of development and also procedure. If I were to do this project again, the most important thing that would be done differently is the amount research that was done into the project domain. I would spend a lot more time at the beginning of the project making sure I had properly grasped the relevant theory before trying to start development. This would not only make implementation of the various models and concepts easier to begin with, but would make the path to achieving the project aim a lot clearer. It would also make it a lot more feasible to create a full project plan to help with time management throughout the project. Not following a properly laid out project plan definitely hindered the potential that the project had. Time was wasted in certain areas of implementation, which ultimately weren't included in the final artefact. However, as mentioned before, the artefact that was created to meet the project aim was done so in a very robust and flexible way. Given more time, and using the knowledge and experience gained from seeing this project through, this functionality of this artefact could easily be expanded.

5.3 Future Work

As previously detailed, this functionality of this artefact can easily be expanded to include a much bigger scope. Based on what the artefact currently contains, a logical and well needed addition to the artefact would be the ability to perform its own backtesting within the application. This gives more insight into the performance of the created portfolio without having to use a third-party system.

The theoretical concepts that the artefact is based off is only the Markowitz Model, the next step for future work would be to include the other parts of Modern Portfolio Theory such as William Sharpe's Capital Asset Pricing Model (CAPM). This would extend the functionality of the artefact to not only optimising the stock selection that the user provides it, but actually calculates

and suggests the most optimal stocks to add to the portfolio using metrics such as the Sharpe Ratio.

Another avenue that could be explored would be alternatives to the use of historical data. Machine learning techniques could be employed to potentially predict future stock prices in which returns can be generated for and then analysed using the same mean-variance analysis. This could provide more meaningful analysis for how a portfolio will perform compared to using historical averages as a means of future estimation. Using the same idea of alternatives to metrics typically used in Modern Portfolio Theory, the use of different risk metrics such as VaR or CVaR could be explored, as it is found to be more effective than solely using return variance.

7. References

- Agile Methodology (2008) The Agile Movement. Online. Available from <http://agilemethodology.org/> [accessed 16th April 2017].
- Amazon (2017) Amazon Web Services (AWS) - Cloud Computing Services. Online: Amazon. Available from <https://aws.amazon.com/> [accessed 21st April 2017].
- Andersen, M. and Vandenberghe, L. (2016) CVXOPT: Python Software for Convex Optimisation [app/game/software].Online .
- Awad, M. (2005) A comparison between agile and traditional software development methodologies. University of Western Australia, .
- Balaji, S. and Murugaiyan, M.S. (2012) Waterfall vs. V-model vs. agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2(1) 26-30.
- Blanchette, J. and Summerfield, M. (2006) C++ GUI programming with Qt 4. : Prentice Hall Professional.
- Busse, M., Dacorogna, M. and Kratz, M. (2014) The impact of systemic risk on the diversification benefits of a risk portfolio. Risks, 2(3) 260-276.
- Butler, B. (2014) 10 of the most useful cloud databases . Online: NetworkWorld. Available from <http://www.networkworld.com/article/2162274/cloud-storage/cloud-computing-10-of-the-most-useful-cloud-databases.html> [accessed 21st April 2017].
- Chacon, S. and Straub, B. (2014) Pro git. : Apress.
- Christie, S. (2005) Is the sharpe ratio useful in asset allocation? . Online. Available from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=720801 [accessed 22nd April 2017].
- Continuum Analytics (2017) Anaconda [app/game/software].Austin, Texas : Continuum Analytics.
- Droettboom, M. (2017) Matplotlib [app/game/software]. Version 2.00. Online .
- Elton, E.J. and Gruber, M.J. (1997) Modern portfolio theory, 1950 to date. Journal of Banking & Finance, 21(11) 1743-1759.
- Elton, E.J., Gruber, M.J. and Padberg, M.W. (1978) Simple criteria for optimal portfolio selection: Tracing out the efficient frontier. The Journal of Finance, 33(1) 296-302.
- Fabozzi, F.J., Gupta, F. and Markowitz, H.M. (2002) The legacy of modern portfolio theory. The Journal of Investing, 11(3) 7-22.
- Finance Train (2017) Constructing an Efficient Frontier. Online: Finance Train. Available from <http://financetrain.com/constructing-an-efficient-frontier/> [accessed 24th April 2017].

Garvey, P.R. and Lansdowne, Z.F. (1998) Risk matrix: An approach for identifying, assessing, and ranking program risks. *Air Force Journal of Logistics*, 22(1) 18-21.

Gibbs, G. (1988) The reflective cycle. Kitchen S (1999) an Appraisal of Methods of Reflection and Clinical Supervision. *Br J Theatre Nurs*, 9(7) 313-317.

GitHub (2017) GitHub [app/game/software]. San Francisco, California .

Hamano, J. (2017) Git [app/game/software]. Version 2.12.2. Online : Git.

HostGator (2017) What are Cron Jobs? Online: HostGator. Available from <http://support.hostgator.com/articles/cpanel/what-are-cron-jobs> [accessed 22nd April 2017].

Investopedia Closing Price. Online: Investopedia. Available from <http://www.investopedia.com/terms/c/closingprice.asp> [accessed 22nd April 2017].

Investopedia (2017) Backtesting. Online: Investopedia. Available from <http://www.investopedia.com/terms/b/backtesting.asp> [accessed 27th April 2017].

Investopedia (2017) Covariance. Online: Investopedia. Available from <http://www.investopedia.com/terms/c/covariance.asp> [accessed 13th April 2017].

Investopedia (2017) Creating Gantt Charts. Online: Investopedia. Available from <http://www.investopedia.com/terms/g/gantt-chart.asp> [accessed 17th April 2017].

Investopedia (2017) Downside Risk. Online: Investopedia. Available from <http://www.investopedia.com/terms/d/downsiderisk.asp> [accessed 19th April 2017].

Investopedia (2017) Adjusted Closing Price. Online: Investopedia. Available from http://www.investopedia.com/terms/a/adjusted_closing_price.asp [accessed 22nd April 2017].

Investopedia (2017) Short Selling. Online: Investopedia. Available from <http://www.investopedia.com/terms/s/shortselling.asp> [accessed 25th April 2017].

Investopedia (2017) Systematic Risk. Online: Investopedia. Available from <http://www.investopedia.com/terms/s/systematicrisk.asp> [accessed 18th April 2017].

Investopedia (2017) Unsystematic Risk. Online: Investopedia. Available from <http://www.investopedia.com/video/play/unsystematic-risk/> [accessed 18th April 2017].

Kaplan, P. (1998) Asset allocation models using the markowitz approach. 13th April 2017.

Karatzas, I., Shreve, S.E., Karatzas, I. and Shreve, S.E. (1998) *Methods of mathematical finance*. : Springer.

Konno, H. and Yamazaki, H. (1991) Mean-absolute deviation portfolio optimization model and its applications to tokyo stock market. *Management Science*, 37(5) 519-531.

Ledoit, O. and Wolf, M. (2008) Robust performance hypothesis testing with the sharpe ratio. *Journal of Empirical Finance*, 15(5) 850-859.

Louis Raffestin, (2014) Diversification and systemic risk. *Journal of Banking & Finance*, 46 85.

Malaiya, Y.K. (1995) Antirandom testing: Getting the most out of black-box testing. In: *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on: IEEE*, 86-95.

Mangram, M.E. (2013) a simplified perspective of the markowitz portfolio theory. ♦
VOLUME 7(1) 1.

Markowitz, H. (1952) Portfolio selection. *The Journal of Finance*, 7(1) 77-91.

Matos, S. and Lopes, E. (2013) Prince2 or PMBOK--a question of choice. *Procedia Technology*, 9 787-794.

Michaud, R.O. (1989) The markowitz optimization enigma: Is optimized optimal? *ICFA Continuing Education Series*, 1989(4) 43-54.

Morgan, D. (2013) Arithmetic vs. Logarithmic Rates of Return. Online: DCF NERDS. Available from <http://www.dcfnerds.com/94/arithmetic-vs-logarithmic-rates-of-return/> [accessed 23rd April 2017].

Muenchen, R.A. (2017) The Popularity of Data Science Software.

MySQL (2017) MySQL :: About MySQL. Online: MySQL. Available from <https://www.mysql.com/about/> [accessed 21st April 2017].

NumPy (2017) NumPy [app/game/software]. Version 1.12.0. Online : NumPy.

Oracle Corporation (2017) MySQL Workbench 6.3 [app/game/software]. Version 6.3.9. Online : Amazon.

Oxford English Dictionary (2010) Oxford English Dictionary. 3rd edition. Oxford: Oxford University Press.

Pandas (2016) Python Data Analysis Library [app/game/software]. Version 0.19.2. Online : Pandas.

Porter, J. (2017) Principles of User Interface Design. Online: Bokardo. Available from <http://bokardo.com/principles-of-user-interface-design/> [accessed 25th April 2017].

Powell-Morse, A. (2016) Iterative Model: What Is It And When Should You Use It?. . Available from <https://airbrake.io/blog/sdlc/iterative-model> [accessed 17th April 2017].

Python Software Foundation (2017) Python [app/game/software]. Version 3.6.1. Online : Python Software Foundation.

Quandl (2017) Quandl Financial, Economic and Alternative Data. Toronto, Canada: Quandl. Available from <https://www.quandl.com/> [accessed 20th April 2017].

Quandl (2017) Quandl Python Documentation. Online: Quandl. Available from <https://www.quandl.com/product/WIKIP/WIKI/PRICES-Quandl-End-Of-Day-Stocks-Info> [accessed 20th April 2017].

R Core Team (2017) An Introduction to R. 3.3.3 edition. R.

R Core Team (2017) R [app/game/software]. Version 3.3.3. Online : R Foundation.

Rockafellar, R.T. and Uryasev, S. (2000) Optimization of conditional value-at-risk. *Journal of Risk*, 2 21-42.

SCRUM (2017) Online. Available from <https://www.scrum.org/resources/what-is-scrum> [accessed 17th April 2017].

Sharpe, W.F. (1994) The sharpe ratio. *The Journal of Portfolio Management*, 21(1) 49-58.

Shipway, I. (2009) Modern portfolio theory. *Trusts & Trustees*, 15(2) 66.

Silicon Cloud Technologies (2017) Backtest Portfolio Asset Allocation. Online: Silicon Cloud Technologies. Available from <https://www.portfoliovisualizer.com/about> [accessed 27th April 2017].

Solnik, B.H. (1995) Why not diversify internationally rather than domestically? *Financial Analysts Journal*, 51(1) 89-94.

Spyder Developer Community (2017) Spyder [app/game/software]. Version 3.1.3. Online : MIT.

Stackoverflow (2015) Developer Survey. Online: Stackoverflow. Available from <http://stackoverflow.com/insights/survey/2015> [accessed 16th April 2017].

Thangavelu, P. (2015) Why risk-free investments don't exist.

The Qt Company (2017) Qt. . Available from <https://www.qt.io/> [accessed 17th April 2017].

Yahoo! (2017) Yahoo! Finance. New York City, New York: Yahoo!.. Available from <https://uk.finance.yahoo.com/> [accessed 20th April 2017].

8. Appendix A: Methodology

1. Risk Matrix

Risk	Likelihood	Impact	Risk Quotient	Manage/Mitigation
Chosen data API becomes unavailable.	0.1	6	0.6	Find backup ahead of time, or find alternative.
Underestimate time requirements.	0.2	4	0.8	Make sure to fully understand each project task.
Losing access to online database.	0.2	3	0.6	Use locally stored database for the time being or pull directly from source.
Data provided by API is incorrect.	0.2	1	0.2	Use a different source.
Optimiser produces wrong optimisation.	0.5	8	4	Spend more time working on it to understand why.
Data retrieval system isn't working.	0.2	8	1.6	Manually retrieve data for the time till it is fixed.
Compatibility problems between display and optimisation system	0.5	8	4	Ensure output from optimisation system fits the correct format, or find other way to display it.
GUI doesn't handle user interaction properly.	0.2	2	0.4	Change the way the user can interact with it to a simpler way.

9. Appendix B: Software Development

1. Retrieval System Test Cases

Description	Expected Result	Actual Result																																																				
Verification of data being retrieved and also being in the correct format.	A dataframe containing a column of closing price data for each stock with a date index and name of each stock as column title.	<div>The result shows the first three stocks in the dataframe. It contains the date index and the column name as requested meaning that it has passed the test.</div> <table><thead><tr><th>Date</th><th>ibm</th><th>aapl</th><th>msft</th></tr></thead><tbody><tr><td>2015-01-02</td><td>149.990031</td><td>104.712140</td><td>44.041815</td></tr><tr><td>2015-01-05</td><td>147.629951</td><td>101.762233</td><td>43.632102</td></tr><tr><td>2015-01-06</td><td>144.446157</td><td>101.771811</td><td>42.996340</td></tr><tr><td>2015-01-07</td><td>143.502125</td><td>103.198876</td><td>43.542624</td></tr><tr><td>2015-01-08</td><td>146.621133</td><td>107.164012</td><td>44.823567</td></tr><tr><td>2015-01-09</td><td>147.259742</td><td>107.278943</td><td>44.446819</td></tr><tr><td>2015-01-12</td><td>144.788600</td><td>104.635519</td><td>43.891116</td></tr><tr><td>2015-01-13</td><td>145.131043</td><td>105.564549</td><td>43.660358</td></tr><tr><td>2015-01-14</td><td>144.196266</td><td>105.162289</td><td>43.283610</td></tr><tr><td>2015-01-15</td><td>143.057874</td><td>102.308157</td><td>42.836222</td></tr><tr><td>2015-01-16</td><td>145.436465</td><td>101.513215</td><td>43.552043</td></tr><tr><td>2015-01-20</td><td>145.260616</td><td>104.127906</td><td>43.693323</td></tr></tbody></table>	Date	ibm	aapl	msft	2015-01-02	149.990031	104.712140	44.041815	2015-01-05	147.629951	101.762233	43.632102	2015-01-06	144.446157	101.771811	42.996340	2015-01-07	143.502125	103.198876	43.542624	2015-01-08	146.621133	107.164012	44.823567	2015-01-09	147.259742	107.278943	44.446819	2015-01-12	144.788600	104.635519	43.891116	2015-01-13	145.131043	105.564549	43.660358	2015-01-14	144.196266	105.162289	43.283610	2015-01-15	143.057874	102.308157	42.836222	2015-01-16	145.436465	101.513215	43.552043	2015-01-20	145.260616	104.127906	43.693323
Date	ibm	aapl	msft																																																			
2015-01-02	149.990031	104.712140	44.041815																																																			
2015-01-05	147.629951	101.762233	43.632102																																																			
2015-01-06	144.446157	101.771811	42.996340																																																			
2015-01-07	143.502125	103.198876	43.542624																																																			
2015-01-08	146.621133	107.164012	44.823567																																																			
2015-01-09	147.259742	107.278943	44.446819																																																			
2015-01-12	144.788600	104.635519	43.891116																																																			
2015-01-13	145.131043	105.564549	43.660358																																																			
2015-01-14	144.196266	105.162289	43.283610																																																			
2015-01-15	143.057874	102.308157	42.836222																																																			
2015-01-16	145.436465	101.513215	43.552043																																																			
2015-01-20	145.260616	104.127906	43.693323																																																			
Verification of the database being populated appropriately.	The MySQL database should contain the data compiled from the API call. It should have appropriate headings	<div>This image shows the table featured on the database that was set up, containing all the necessary columns, showing that it is working correctly.</div> <table><thead><tr><th>Date</th><th>ibm</th><th>aapl</th><th>msft</th></tr></thead><tbody><tr><td>2010-01-04 00:00:00</td><td>111.38275978782</td><td>27.727867460078794</td><td>25.558244999518</td></tr><tr><td>2010-01-05 00:00:00</td><td>110.03725268582</td><td>27.775805925385228</td><td>25.566502913896</td></tr><tr><td>2010-01-06 00:00:00</td><td>109.32245203788</td><td>27.333994664047584</td><td>25.409602540717</td></tr><tr><td>2010-01-07 00:00:00</td><td>108.94402816544</td><td>27.283464930346213</td><td>25.1470008635</td></tr><tr><td>2010-01-08 00:00:00</td><td>110.03725268582</td><td>27.464853717992163</td><td>25.31876548256</td></tr><tr><td>2010-01-11 00:00:00</td><td>108.88516222972</td><td>27.222570123065072</td><td>24.996706821823</td></tr><tr><td>2010-01-12 00:00:00</td><td>109.75133242664</td><td>26.91291354986948</td><td>24.831548534266</td></tr></tbody></table>	Date	ibm	aapl	msft	2010-01-04 00:00:00	111.38275978782	27.727867460078794	25.558244999518	2010-01-05 00:00:00	110.03725268582	27.775805925385228	25.566502913896	2010-01-06 00:00:00	109.32245203788	27.333994664047584	25.409602540717	2010-01-07 00:00:00	108.94402816544	27.283464930346213	25.1470008635	2010-01-08 00:00:00	110.03725268582	27.464853717992163	25.31876548256	2010-01-11 00:00:00	108.88516222972	27.222570123065072	24.996706821823	2010-01-12 00:00:00	109.75133242664	26.91291354986948	24.831548534266																				
Date	ibm	aapl	msft																																																			
2010-01-04 00:00:00	111.38275978782	27.727867460078794	25.558244999518																																																			
2010-01-05 00:00:00	110.03725268582	27.775805925385228	25.566502913896																																																			
2010-01-06 00:00:00	109.32245203788	27.333994664047584	25.409602540717																																																			
2010-01-07 00:00:00	108.94402816544	27.283464930346213	25.1470008635																																																			
2010-01-08 00:00:00	110.03725268582	27.464853717992163	25.31876548256																																																			
2010-01-11 00:00:00	108.88516222972	27.222570123065072	24.996706821823																																																			
2010-01-12 00:00:00	109.75133242664	26.91291354986948	24.831548534266																																																			

2. Artefact Test Cases

Description	Expected Result	Actual Result
User creates portfolio using “Stock Chooser” window.	The main window shows all the relevant information pertaining to each stock and plots the graph.	All the main window features are updated with the correct statistics and visuals.
User opens the “Stock Chooser” window after having created a portfolio already.	The “Choose Stocks” window should appear with portfolio’s current stocks checked.	The stocks that are checked previously are no longer checked. This can be accepted as portfolio is rewritten each time
User optimises for Minimal Risk	The portfolio marker on the graph canvas should move to the minimum variance point and the statistics should update.	The portfolio moves to the minimum variance point and statistics change -> risk lowers.
User optimises for Maximum Return	Portfolio marker should move to the right most point of the Efficient Frontier and return/risk increase.	Portfolio marker moves to end of Efficient Frontier and risk/return update properly.
User optimises for specified risk aversion level of 0.	The portfolio marker should again move to the end of the Efficient Frontier.	Portfolio marker moves to the end of the Efficient frontier.
User optimises for a specified risk aversion level of 5 after optimising for a level of 0	The portfolio marker should move to a rough midpoint on the Efficient Frontier. Risk and return should both decrease.	Portfolio marker moves to a varying midpoint depending on the portfolio.
User opens the “All Stock Performance” window.	Two tabs should display showing the price and risk/return graph.	Both price and risk/return graphs display correctly.
User opens the “Stock Analysis” window	A list of stocks should be displayed, with all the other features being blank.	Stocks are displayed and other features are blank.
User clicks stock in the stock list	Numerical and graphical description of that stock should be generated.	Upon clicking, price and return graphs are generated as well as return, standard deviation and variance statistics.