OLLSCOIL TEICNEOLAÍOCHTA
BHAILE ÁTHA CLIATH
TECHNOLOGICAL
UNIVERSITY DUBLIN

Sample Dissertation #1

(Front Matter removed)

Table of Contents

Table of Figures

Figures

Code Snippets

Chapter 1 - Introduction

Project Overview

The purpose of this project was to design and develop a web application for investors to help them research potential stocks and analyse their portfolios. The project attempts to bridge the gaps seen in traditional solutions by aggregating multiple sources of information into one platform and introducing some innovative techniques such as portfolio optimisation and machine learning enhanced social media sentiment analysis.

The project consists of multiple components all geared towards aiding the user in their investment decisions. Stockify provides information on over 5500 companies trading on the New York and Nasdaq stock exchanges. For each company it provides information on the business which includes brief business description, website link, the businesses industry and the names of key managerial staff such as the CEO. Stockify also provides in-depth financial information on each company to allow investors to establish and evaluate the financial status of the company in question. This is provided with financial ratios and breakdowns, the latest and historical pricing as well as financial reports going back as far as 10 years. Along with financial and business information regarding each company Stockify also provides social media and news feeds so users can see the latest happenings and opinions of other investors on the market.

Building on top of the social feeds provided by the platform Stockify also offers real time sentiment analysis on the latest tweets to enhance investors understanding and to quickly relay the feeling of the market toward any stock on the platform. To achieve this Stockify employs an automated cluster computing backend that uses the resources of the Google Cloud platform to train a sentiment analyser on several hundred thousand tweets every day. This is done to keep up with the financial lexicon and to provide the most accurate predictions.

Once investors have concluded their research and have made additions to their portfolios, Stockify offers portfolio analysis. The portfolio analysis attempts to help users tailor their portfolios in a way that makes sense for them and allows them to achieve their financial goals.

Through the techniques mentioned above it is intended that the project will help investors of all means and experience to better their investment decisions and meet their financial goals be it investing for retirement or for a rainy day.

Project Objectives

The objective of the project is to design and implement an easy to use web application that will allow users to enhance their investment decisions by providing an aggregation of previously existing information sources and innovative techniques. For the project to achieve this it needs to provide an easily navigable website for all users. It must also provide useful information on all the stocks on the Nasdaq and NYSE stock exchanges. The project should also provide sentiment analysis on the social feeds. Once decisions have been made on the part of the user their portfolios must be analysed and the system must return useful information regarding their decisions.

In order to implement a system that can provide these features the following steps must be completed.

1. Gain an understanding of the financial market with regards to stocks and investing in public companies.
2. Research and gather the information necessary for the platform, such as financial and business information API's.
3. Research pre-existing solutions and techniques.
4. Research the most suitable technology necessary to implement a web based system.
5. Design and implement a system that incorporates all the necessary features.
6. Deploy the system for use on the web.
7. Test and evaluate the system once built.

Project Challenges

Introduction

As with every project there will be a significant amount of challenges that need to be dealt with. Below the potential risks and issues associated with the project will be outlined along with the precautions that can be put in place to mitigate the potential negative effect of each.

Data Access

As this project requires a large volume of information from multiple sources to be aggregated before analysing the data or providing the data to users, it may very well be an issue. This project will be interacting with multiple API's regularly and as a result they need to not only be reliable but they must also provide the correct information as well as an appropriate amount of information upon which the system can operate.

Financial information and data feeds are notoriously expensive so sourcing a free alternative or a relatively cheap data source was a priority.

Hardware limitations

As this project will be processing a large amount of data, transforming it and using it for input to intensive processes, hardware issues may become a concern. Attention needs to be paid to the resources necessary to run such operations and if such operations are feasible to run for the duration of the project. Using resources on cloud providers can become quite expensive. If large resources are allocated for extended periods, finding the right balance between performance and cost will be a key aspect of the project.

Technical Approach

Learning new technologies can be time consuming and makes forecasting objectives more difficult. It may prove more challenging than originally anticipated to develop certain aspects of the system. Certainly new technologies will need to be learned. However aligning the technologies employed for the project with what is taught through other modules should help to accelerate the learning process. Identifying new areas early will be key as time must be given to accommodate for the learning process.

Time Horizon

The main concern regarding time is the lack of it. Although the duration of an academic year is provided to develop the project, energy must also be devoted to other modules. Unforeseen time issues may occur leading to certain features being dropped. It is important to prioritise the most necessary features.

Conclusion

Unfortunately, as mentioned above there are quite a number of potential issues that may take place in the coming year. The main concerns to note are data, technology and time. The project essentially hinges on the presence of actionable data. The lack of which will disrupt the project entirely. A great deal of research is necessary to ensure this issue is minimalised. New and familiar technology also provide some issues that may be unforecastable. Often times projects run into problems with technology not working as intended or multiple technologies not being interoperable. No doubt issues of a technical nature will arise but providing adequate time to deal with them is key to the success of this project.

Report Structure

Research

This section of the report will cover all the background research conducted for the project. This will include research into financial markets and their behaviour, market forces that drive stock prices up and down, risk calculation and mitigation. This section will also include information on technology necessary to develop and host the web application such as languages, frameworks, architecture and hosting solutions. Consideration will also be given to pre-existing technologies that relate to or operate in a similar domain as the project.

Design

This section of the report will focus on design choices regarding the project. In this section design methodologies will be explored along with project design and the evolution from prototype to deployed application.

System Validation

This section of the report will discuss the testing and evaluation of the project. Software testing approaches will be discussed as well as usability and user evaluation.

Project Plan

This section of the report will examine the project plan, more importantly it will discuss the change in plans over the duration of the project and the reasoning behind it. It will also discuss possible future additions and use cases.

Conclusion

This section of the report will serve as a reflection on the whole project. It will explore the process of development for the project and any changes that would be made by the author if they had to undertake the project again.

Chapter 2 – Research

Introduction

This section of the report will cover research associated with developing and designing the project. It is critical for the project that research is performed as it is essential in determining the viability of the project. Research regarding the stock market, it's behaviour and the forces behind it needs to be gathered. This research will provide a baseline understanding of how the product should help its users and to allow the author to better tailor a solution.

An examination of the current investment research process undertaken by the author will be conducted to better outline the aggregation necessary. This will help determine necessary features for the project.

Building on top of the aggregation, some of the data-oriented features such as sentiment analysis will be examined. These features should be examined to determine feasibility and usefulness regarding investment decisions.

The research process should also explore portfolio analysis and financial management techniques that can help the end user achieve their financial goal. To this end the report will explore risk mitigation, maximising returns and portfolio diversification. It must also be considered how these techniques can be implemented in a software solution and provided to the users. The solution must be useable in a way that isn't too technical for beginners but also provides sufficient information for more advanced investors.

Attention must also be paid to technical solutions and architecture of the system. Researching and selecting the correct technologies is essential in making this project viable. Therefor the second half of this section will be providing deep dive into the technical aspects of the project.

Background Research

Introduction

Before a system is developed to assist users in their research some background research must be completed to understand why the user may want to invest. This research will outline some of the reasons why stock markets exist and why people chose to invest in them.

What is the stock market ?

A stock market refers to a collection of exchanges where shares of public companies can be bought and sold by investors (1). Companies can also issue shares for sale to the public through these exchanges. Two of the largest stock markets at this moment in time is the New York Stock Exchange and The Nasdaq stock exchange (2). The previous two exchanges deal mainly with American stocks and the majority of the largest American based companies are listed with either of them. Other countries have their own stock markets such as the Euronext exchange that covers most European countries (3).

What are stocks / shares ?

Shares or stocks are the units of ownership in public companies that are sold on the stock market. Shares are sold to the public through the exchanges for multiple reasons which will be explored in the upcoming sections. The most common type of shares issued are common stock. Common stock can appreciate and provide dividends for stockholders but they also provide voting rights with regards to the business (4). This means that stockholders have the right to vote on issues pertaining to the company such as a change in the board of directors, mergers and acquisitions of other companies (5).

What is the purpose of the stock market ?

There are three main reasons why stock markets are consistently used. Firstly, Companies who wish to raise capital for operations may opt to go public and sell shares in the business. If a company wishes to go public, they will go through an initial public offering or IPO. The company is divided into shares and a portion of those shares are offered to the public to purchase (6).

The second reason is to allow investors to easily purchase and sell shares in companies. Investors purchase shares in the hope that they will appreciate in value due to the success of the company or the shares offer a dividend to the stockholder.

The third reason is regulation. Stock markets provide both investors and companies a secure and regulated marketplace in which they can participate. Government bodies such as the Securities and Exchange Commission or SEC monitor market activity for foul play (7). An example of this would be insider trading where individuals working in the company's trade shares based on news not released or known by the public (8). A recent example of this was Martha Stewart, a celebrity chef who served 5 months in prison after she sold all of her shares of ImClone (9). The shares were sold two days prior to a 16% decline in the stock's price.

Why do investors purchase shares / stocks?

The reasoning behind purchasing stocks over other assets such as government bonds is the possibility of a higher return. An investor can expect a 2.99% return on a 10-year bond from the US government (10). This is more return than an investor would receive from bank interest due to low interest rates. However, the investors return will quickly be eaten up by inflation which as of October 12th, 2018 is 2.5 percent (11,12). Stocks although riskier, provide more of a return overall and are more effective at hedging inflation due to the possibility of a greater return and dividends (13).

Why do stocks have a higher return than other assets?

Stock prices are influenced by a multitude of variables such as the financial climate, supply, demand and management. These can all have an effect on the performance of the underlying companies which dictates the price (14). It is these factors that are not present or not as influential on other assets such as bank interest rates that dictate the risk and return provided by the stock market.

Conclusion

Having looked at the stock market and the reasons behind investing the requirements for the system can be better defined. The stock market provides a regulated market place where investors may invest to better their financial standing and hedge against inflation. Risk and return play a massive role in the market and determine how well or poorly an investment will perform. Focusing on managing risk and improving returns will be key to investor success and research will be conducted on the subject in the following sections of this report.

Stock Research Process

Introduction

There are many platforms that allow investors to research securities and assets on the internet. Many platforms provide useful in-depth analysis on thousands of stocks. However, due to the nature of the stock market and the profits associated with it, the majority of online resources that provide such analysis are paid for services that require memberships.

For amateur investors such as the author of this paper the solutions provided are too expensive or provide too narrow of an insight upon which an investor is to confidently invest their money.

As a result, many investors utilise an array of online resources to determine the viability of potential investments. These sources range from services that provide news and analysis to social media feeds. By manually aggregating these feeds investors can gain insight into company's financial health, business practices, industry news and market sentiment.

To highlight the aggregation necessary to provide such a platform, a typical investor research process will be outlined below.

Research process

Finviz

Figure 1 : Finviz page for First Solar Inc

An investor looking for stocks to enhance their portfolio might begin their research on a security screener such as Finviz. Screeners provide users the ability to define criteria such as market capitalisation, financial ratios and industry which is used to return stocks matching those criteria.

Once the investor finds a potential stock within the results of their search, they can view financial figures and ratios of the stock to determine if it fits their investment strategy.

SEC Edgar search

If the investors believe the stock found during their research on Finviz may be a fit for their portfolio they would typically read through the company's filings. These filings are provided on the Securities and Exchange Commission's website.

The SEC provides a database of company financial filings which each company is mandated to release on an annual and quarterly basis. Form 10-Ks and 10-Qs are filed by every company (with some exceptions for foreign based companies which file 20-Fs) outlining important information. Information such as the company's financial statements and possible risk factors. The filings also include a description of company operations and management's discussion of the company's financial condition.

If the investor believes that the stock could be a viable investment after reading the company filings, they may opt to view other investors opinions to gauge market sentiment toward the stock.

Figure 2 : SEC filings for First Solar Inc

Seeking-Alpha

Seeking Alpha is a platform that provides news articles and analysis from its users. The site is used to publish in depth critiques and analysis of individual stocks and the stock market. The authors of these articles vary in their association with the stock market, ranging from amateur to professional investors.

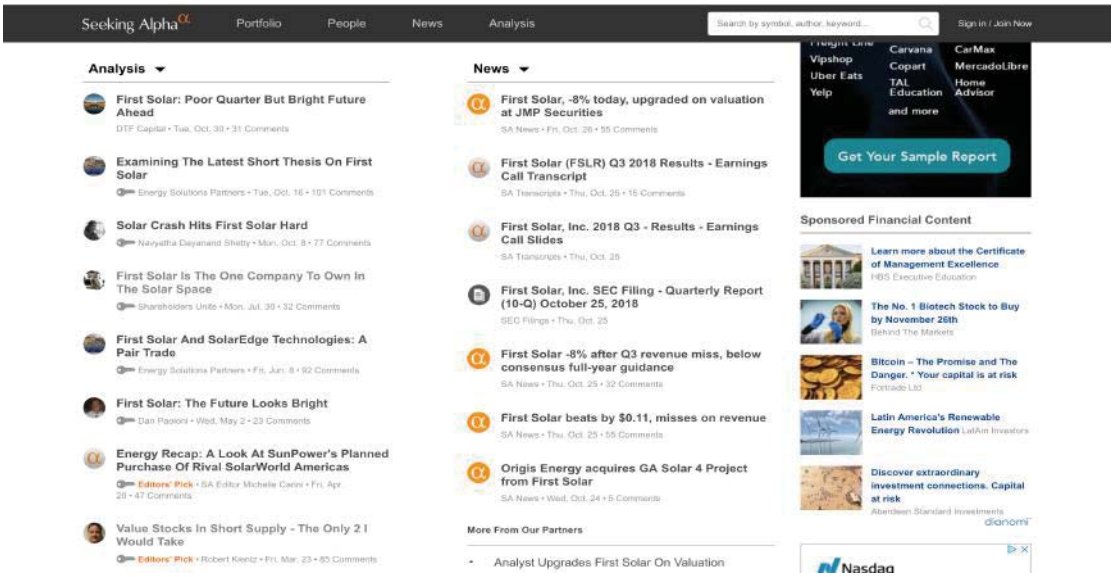

Figure 3 : Seeking Alpha articles and news relating to First Solar Inc.

Twitter & Stock twits

Aside from Seeking Alpha other social media platforms provide user opinions on the stock market. Two prominent sources being StockTwits and Twitter. Both platforms provide nearly identical data with character limited statements. StockTwits provides a sentiment value that denotes the user's sentiment toward a given stock. Investors wishing to publish their sentiment can choose between either bullish or bearish. A bullish investor believes the stock in question will perform well, whereas a bearish investor believes the stock will perform poorly.

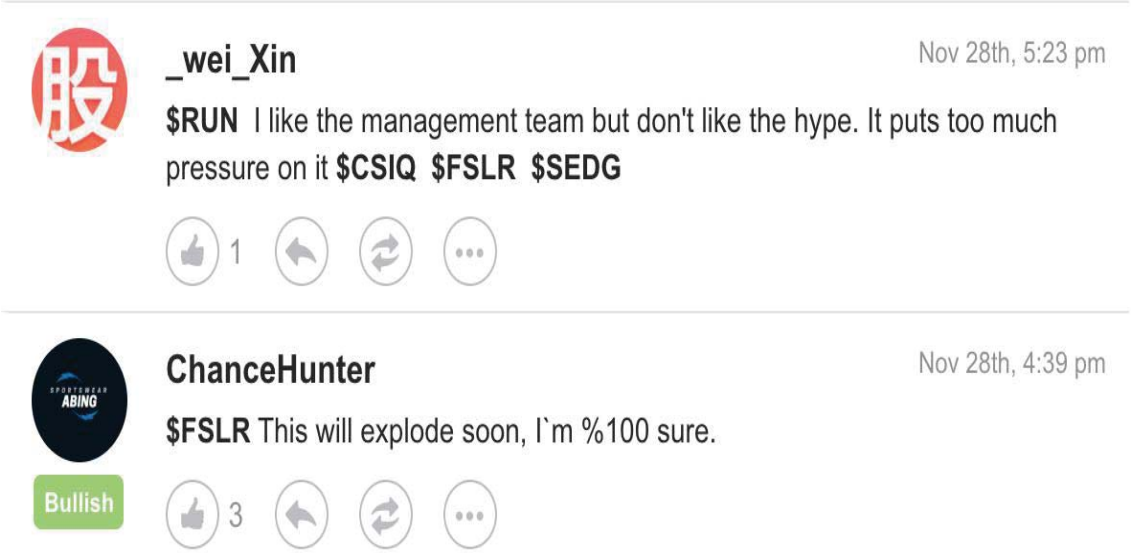

Figure 4 : StockTwits posts related to First Solar Inc.

Conclusion

The section above highlights the amount and types of sources investors must visit to gather enough information to form an opinion on a company. Each site provides useful information however, using any one alone would probably not be enough to make a confident decision. Many aspects of the company are reflected in the above data such as the financial information / standing of the company and the sentiment of the market surrounding the stock. Both of these have a huge influence on the performance of the company in question. If this sort of data can be gathered and enhanced with other technologies and techniques, it could very well improve the scope of the investors research while saving them time.

Existing Solutions

Introduction

The stock market is quite a lucrative venture for willing investors. As a result, a multitude of companies have designed and implemented platforms to help inform investors on potential investments.

Prior to reading the analysis of existing solutions some distinctions must be made between methods of analysis. When evaluating stocks there are two approaches generally taken, these are technical and fundamental security analysis.

Fundamental Analysis

Fundamental analysis aims to find the intrinsic value of a company. To do so investors who apply fundamental analysis look at economic factors. These include the company's industry or sector, financials such as revenues and expenses and the company's growth prospects (15,16).

This technique typically assumes that in the long run a stock price will correct itself and that the investor can make gains by purchasing under-valued stocks and waiting for the market to correct itself (15,16).

Investors using fundamental analysis have longer time horizons and are willing to wait longer for a return (15,16).

Typically used by value investors, some notable value investors include Warren Buffet and Benjamin Graham (17).

Technical Analysis

Technical analysis uses historical stock price movements to predict future price movements. A technical minded trader would use charts to analyse past stock pricing and volume to discern patterns in the pricing in order predict future trends (15,18).

People who utilise this sort of analysis are usually referred to as traders due to the short time horizon on their investments (15,18).

Both forms of analysis will be discussed further as part of further research into existing solutions. However, the Stockify platform will be tooled for fundamental analysis.

Free / Freemium Services

Finviz

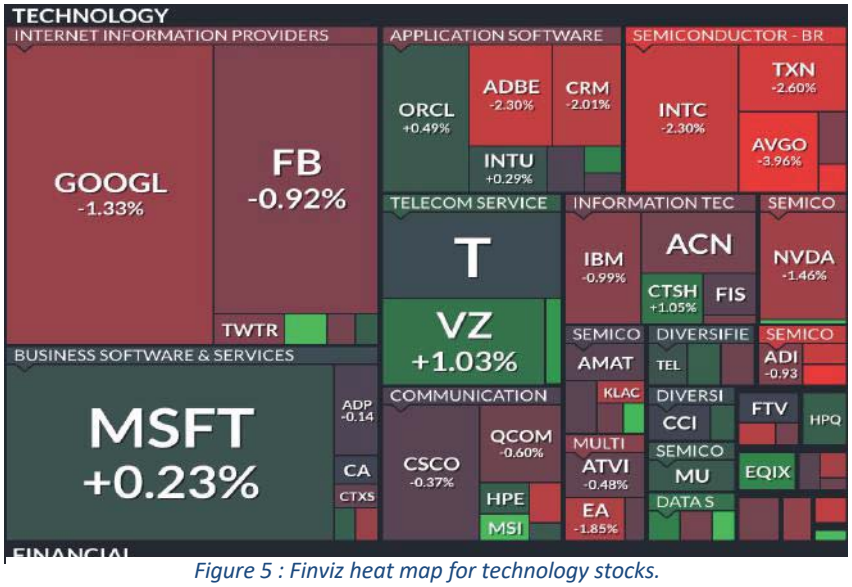

Figure 5 : Finviz heat map for technology stocks.

Finviz is a stock screener that allows users to define criteria by which they can search and filter stocks. The site also offers insight on useful topics such as the day's biggest price movements, group performance, insider trading and news. Finviz is tooled for both fundamental and technically minded traders as it provides a multitude of different search criteria and graphs for both houses of investors.

Figure 6 : Finviz pricing graph

Many of Finviz's features provided inspiration for the Stockify project. The site provides pricing graphs and useful financial calculations that are of great use to the investor. These features are provided for every stock on the site and they serve to highlight the financial status of every company well. This saves the investor from having to pour through financial filings. Many of the financial calculations that are found on the Finviz site such as price to earnings and price to sales ratios are implemented in the project also.

Seeking Alpha

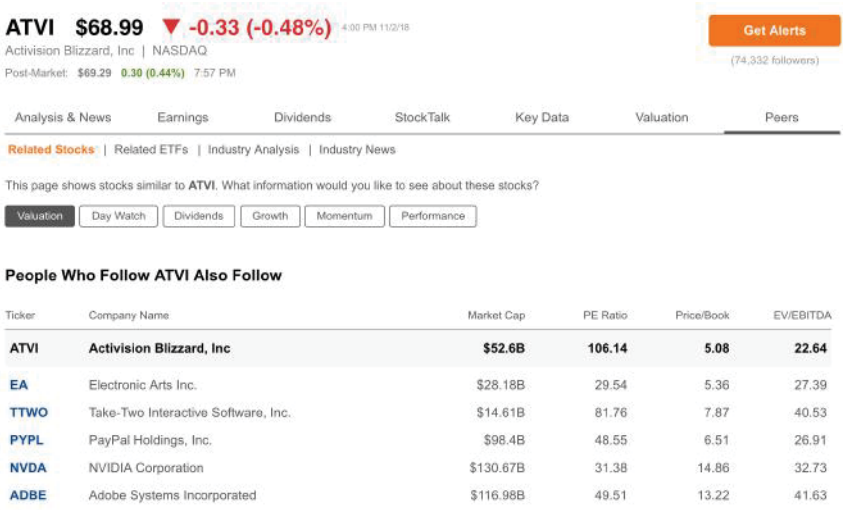

Figure 7 : Seeking Alpha information on Activision Blizzard.

Seeking alpha is crowd sourced content service tailored for the financial world and the stock market. Seeking Alpha provides news articles and analysis for companies on the stock market. Many of the analysis articles are written by users of the site that vary in their association with the stock market, articles are written by amateur and professional investors alike. Seeking Alpha also provides pertinent financial information to fundamental investors such as key financial data and the company in questions peers / competitors.

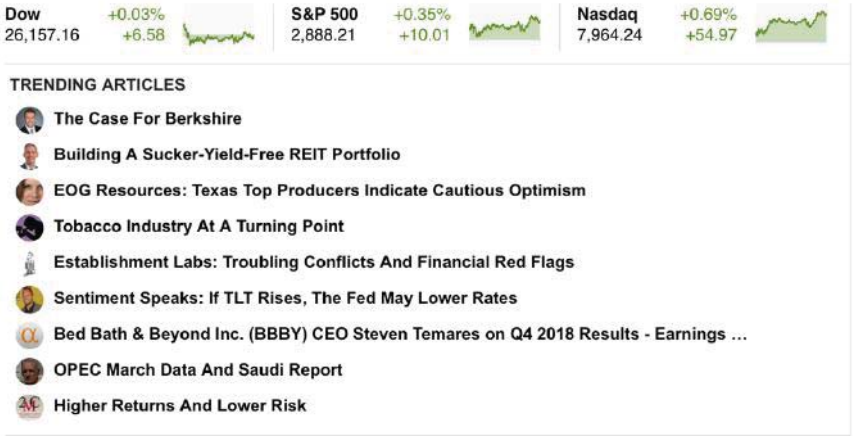

Figure 8 : Seeking Alpha trending news articles

The news and analysis aspect of Seeking Alpha's platform served as an inspiration for this project. The news articles and opinions expressed by the platform's users provide a great amount of detail and research. Accessing these articles provides the user's with up to date news on any company which is important when events can cause the price of companies to rise or fall. Keeping informed on the latest news and research is very important in the financial world and it is reason why the Seeking Alpha RSS feed was integrated in to the project.

Paid Services

StockFluence

StockFluence is a web platform for providing sentiment analysis on individual securities. StockFluence monitors social media channels such as Twitter and uses proprietary algorithms to analyse market sentiment.

StockFluence provides measures such as strength, reach and passion to determine the markets feelings toward a particular stock. The platform also provides a five-day forecast for the previously mentioned measurements. You can also view sentiment statistics for the previous month for free on the website. The rest of the site's functionality requires a subscription.

HedgeChatter

HedgeChatter provides sentiment analysis for individual securities in a similar vain to StockFluece. HedgeChatter utilizes millions of social media posts to determine sentiment toward a given stock.

They recognise Social data as the new alternative data for financial intelligence and use it to inform decisions and provide alerts on 7,600 US equities. This alternative data analysis provides real-time actionable market intelligence for investors and trading firms.

Conclusion

There are a number of solutions available currently to help investors research potential investments. No doubt these platforms have great products and some even implement features that Stockify has such as sentiment analysis. However, upon researching none of the solutions provide the range of information necessary. Products such as StockFluence and HedgeChatter implement innovative ideas but lack other areas of information or incur significant costs. Again, some of the solutions above offer an entirely different approach to what this project is aimed to achieve. This platform is to help inform users to make their own decisions rather than make decisions for the users. Having researched existing solutions and their scope, the concept of Stockify provides a solution that can fill a niche in the solutions market. This will be done by incorporating useful information sources and infusing data with modern computing techniques.

Evaluating existing solutions has highlighted some points that will be added to the requirements for the system. Namely the visual aspect of the platforms especially Finviz and how it relays information visually through charts and graphs.

Financial Research

Introduction

The intention for Stockify has been to not only provide users with aggregated information. The project also utilises computing to help users optimise their financial decisions and recommend steps for constructing the portfolio that is right for them and their goals.

Securities Risk Calculation and mitigation

Quantifying Risk

Consider two stocks whose average annual return is 15% over 4 years.

Stock 1 has a very stable return over time with a return of 14% in year 1, 16% return in year 2, 13% return in year 3 and 17% return in year 4. As an investor you can confidently say that you expect to earn a stable return of roughly between 13 – 17% per year.

Stock 2 returns differ more significantly with a return of -35% in year 1, +65% in year 2, -45% in year 3, +75% in year 4. Although stock 2 has the same average return as stock 1 its returns are very unstable. It is difficult to predict what the stock is going to do next. If you were to hold stock 2 for two years you would have lost a significant amount of money (19).

A volatile market can return both a loss and a gain, however sane investors would be more at ease knowing that they will be making a consistent rate of return (19).

In order to reduce risk, it must be quantified. Statistical measures can be used such as variance and standard deviation which measure the dispersion or spread of the data points around the mean value (19).

These metrics will be used to explore and mitigate risk in the following points.

Risk Mitigation

Non-Diversifiable vs Diversifiable risk

In order to mitigate risk some distinctions should be made in types of risk present the stock market.

Non-diversifiable risk

Non-diversifiable risk affects the whole market and unfortunately can't be mitigated by diversification. It is caused by events such as recession or disaster and every stock will suffer to a certain degree as a result (20,21).

We can expect that all stocks in the market can be influenced by the same factors. For example, the state of the economy.

In a healthy economy which favours the market it is expected that stock prices will increase. During a recession when the economy is down, and consumers are spending less we can assume that stocks will suffer somewhat.

However, some sectors suffer less than others in tough economies. For example, when the economy is performing poorly during a recession, it can be expected that consumers will have less discretionary income which they would use to buy luxury items. In this case a car or phone manufacturer will suffer more compared to a retail food store. An example of this is Walmart as consumers will hold off on purchasing a new phone, but they still have to buy food (22).

Diversifiable risk

Diversifiable risk is the risk associated with events surrounding a given company, industry or sector. It can be mitigated by diversifying your portfolio by picking stocks in various industries or countries and reducing the correlation present between the stocks in your portfolio (20,23,24).

This fact should be kept in investors' minds when considering additions to their portfolio. If an investor was to purchase shares in two technology companies, they are leaving themselves open to risk. If there was an event that effected the technology industry such as a silicon shortage both stocks would be heavily impacted. If the investor had purchased a share in a technology company and a share in a retail company, only one of their shares would have been negatively affected. The retail share would have aided in hedging the loses caused.

As the example above demonstrates in the real world the more context that two companies operate in the more correlation there will be between their stock prices. This is an example of un-systemic or diversifiable risk.

Mitigating Non-diversifiable risk with Beta

Beta is used to gauge the volatility of a stock in relation to the market. Investors can use it to calculate the nature of a stock and how it's performance would change given market conditions changing (25).

$$\beta = \frac{\text{Cov (Stock, Market)}}{\text{Var (Market)}}$$

Figure 9 : Beta formula (26)

Beta is calculated by dividing the covariance between the market and the stock in question, divided by the variance of the market as seen in figure 9 (26). Typically, an index like the Dow Jones industrial average or the S&P 500 is used to represent the market. As the platform deals with American companies it is good to use the S&P 500. The S&P 500 contains the top 500 US companies and is a good representation of the market (27,28).

Once the beta is calculated the stock can be classified into 3 categories based on the value.

Beta > 1 - Is an Aggressive Stock

Beta < 1 – Is a Defensive Stock

Beta = 0 - Has no relationship to the market

When a stock is Aggressive or Defensive it reacts in a particular way to the market. Aggressive stocks receive large returns when the economy is healthy and typically outperform the market. However, when there is a downturn such as a recession these aggressive stocks see a large decline typically more than the market (28).

A Defensive stock will see limited returns when the economy is good and will slightly underperform the market. When a recession hits, they are typically less affected and don't drop as significantly (28).

As discussed in the previous section the companies that sell essentials like Walmart are typically defensive stocks with low betas whereas companies such as Tesla are aggressive stocks with high betas. Low beta stocks can be used by investors to hedge their riskier bets.

Investors armed with metrics such as Beta can use them to optimise their portfolio for their risk appetite. That being said a company might have a healthy Beta value, but its price is down trending overall. This compared to a company with a very high beta but

is trending upward over time (29). Beta needs to be utilised along with fundamental analysis to determine the health of the company and if it will trend upward in the future. This will ultimately determine if the company is a worthy investment.

Portfolio Optimisation

Calculating Markowitz Efficient Frontier

Created by Harry Markowitz in 1952. The Markowitz Portfolio Theorem is an investment strategy based on the idea that average investors who are risk adverse can construct portfolios that optimise their returns based on a given level of risk. Markowitz suggests that through the combination of lowly correlated stocks investors can optimise their return without incurring additional risk. Markowitz was successful in demonstrating that it is not about selecting individual stocks but also determining the right combination of stocks in which you invest (30,31).

Figure 10 : Plot of an efficient frontier (32)

We can compute an efficient frontier for a given portfolio using python to perform a Monte Carlo simulation, as seen in figure 10 (32). This will test thousands of iterations of weights for each stock in the portfolio. For each iteration weights are randomised and expected annual log returns are calculated. The expected annual volatility is then calculated for the portfolio and stored (33).

```
for x in range(1000 * len(portfolio)):
    weights = np.random.random(len(portfolio))
    weights /= np.sum(weights)
    returns = np.sum(weights * log_returns.mean()) * 250
    volatilities = np.sqrt(np.dot(weights.T, np.dot(log_returns.cov() * 250, weights)))
```

Code Snippet 1 : Efficient frontier calculation in the financial API

After the returns and volatility are calculated the Sharpe Ratio must be calculated. The Sharpe ratio is a simple but elegant formula for calculating a risk adjusted return.

The Sharpe ratio assumes that a risk-free asset exists. Unfortunately, no assets are without risks on the stock market. The rate of return for the risk-free asset must be substituted with the rate of return of a 10-year US government bond. A bond typically yields 2.9% a year (10,34).

To calculate the Sharpe ratio of a given portfolio an investor must subtract the rate of return for the risk-free asset (10 year us government bond) from the expected portfolio return and divide the result by the standard deviation of the portfolio.

$$Sharpe\ Ratio = \frac{R_p - R_{rf}}{\sigma_p}$$

R_p = Expected portfolio/asset return

R_{rf} = Risk-free rate of return

σ_p = Portfolio/asset standard deviation

Figure 11 : Sharpe Ratio equation (33)

Calculating the Sharpe Ratio for every iteration of the Monte Carlo simulation allows the system to rank the portfolio weightings. If a portfolio's return increases the Sharpe ratio also increases. If the portfolios risk decreases the Sharpe ratio associated with that portfolio also decreases. Finding the portfolio weighting with the highest Sharpe value will provide the highest return compared to its risk without adding or removing stocks from the portfolio (33).

On the opposite end of the spectrum, very risk adverse investors may not want to expose themselves to that much risk and would much prefer to choose a low volatility portfolio. This can be achieved by choosing the portfolio whose overall volatility is the smallest.

Figure 12 shows the differences in portfolios with the highest Sharpe ratio and lowest volatility or risk. As you can see, the Sharpe portfolio provides over double the annual return (based on the last 5 years) compared to the low volatility portfolio. However, the Sharpe portfolio is nearly twice as volatile or risky. This is also reflected in the weightings of the companies with the Sharpe ratio portfolio prioritising Apple and Nike. Whereas the low volatility portfolio positions half of its stake in Procter and Gamble which has a very low beta value.

Figure 12 : Outcome of MPT analysis performed on a user's portfolio

Conclusion

Using the techniques and formulas mentioned previously, Stockify can provide some interesting and useful information for its users. Users are free to choose whatever variation of stocks from which to construct a portfolio. Stockify will provide analysis on the portfolio in an effort to help optimise either return or risk mitigation.

Sentiment Analysis and the Market

Introduction

Sentiment analysis is a process that attempts to infer or categorise opinions being expressed in a sample of text. Sentiment analysis is typically done by computers using natural language processing and machine learning algorithms (35). The widespread adoption of the internet and the volume of data being generated by users on social media platforms such as twitter has provided an unprecedented data mining opportunity. Sentiment analysis is being employed by various companies to monitor brand reputation, trends and consumer sentiment through social media channels (35). The adoption of the internet and social platforms such as StockTwits and Twitter has provided a platform that anyone can use to voice their opinions on any topic they wish. For the stock market this means there is a wealth of data expressing opinion about various aspects of the market or individual stocks. This can provide a market sentiment forecast in real time. Stockify not only aims to aggregate and present social media data but to also process it and analyse opinions to provide sentiment for all stocks on the platform.

Stock Market Sentiment

In the financial world, market sentiment refers to the opinion of investors in relation to specific stocks or the market itself (36). Stocks and the market in general go through bubbles and panics which can be seen in the myriad of recessions and booms. During booms investors feel that the market is performing well and will continue to do so, this is considered a bullish market. The other end of a bullish market is a bearish one where investors are pessimistic about the market. During bullish runs in the market, prices tend to soar as investors profit. However, Inevitably the market takes a down turn and the market does poorly as a result of fearful investors removing their financial stakes in the market (37).

Applying Sentiment Analysis to The Stock Market

Due to the intangible nature of investors feelings toward the stock market it is challenging to gauge market sentiment through traditional means such as surveys. As mentioned in the introduction new forms of media have sprung up to provide a channel to all investors, not just wall street pundits. The mass of data resulting from these new media platforms could provide clues from a broad pool of investors as to how the market is performing.

The question of why the market behaves like it does has been posed since the beginning. Theorems such as the Efficient market hypothesis states that the price of a stock reflects

all news and information about the company accurately (38). If the efficient market hypothesis is correct then there should be no recessions as the price of the stock should accurately reflect its state at all times (39). From experience this is not the case as multiple recessions have occurred. Clearly other factors are at play and contribute to the success and decline of the market (40).

If market sentiment is the missing link that decides the difference in the businesses efficient market price and its current price. It would be useful to indicate to investors the markets sentiment toward a given stock and to allow them to judge whether or not its current price is justified.

How sentiment analysis can be used in Stockify

StockTwits is a popular platform that allows its users to post opinions of stocks. Users can give their opinion in the form of Bullish or Bearish to better distinguish how they feel. These twits could be used as input to a machine learning algorithm to train a model that can classify bullish and bearish twits. As twitter employs the same character limited statements the posts will be nearly identical in structure and the model could be used to classify tweets also. Once the model is established and deployed company specific tweets and twits can be fed to the algorithm to classify sentiment.

Machine Learning Approach Research

Prior to fully implementing a machine learning algorithm, research and testing was done on some of the possible approaches. This initial research was done in a Jupyter notebook and consisted of data exploration, data sampling techniques and the application of numerous different classification algorithms. The issues encountered and there solutions will be outlined in the following sections.

Data & Data Quality

After StockTwits data had been gathered and stored in a the MongoDB database some data exploration took place to find out the quality of the data. The database contained roughly 250,000 twits, of which only 39,375 had a bullish sentiment and 7,359 had a bearish sentiment. The total amount of useful twits came to 46,734 which is a decent sized dataset. However, there was a significant class imbalance with there being about 5 bullish twits for every bearish twit. This was of concern as the imbalance may effect a classifiers performance significantly to the point of completely ignoring the minority class when predicting.

Identifying Models

The next step in the process was to identify models that would perform well on the imbalanced dataset and provide balanced predictions. As this was a classification problem a supervised learning approach was needed. Simply put, a supervised learning algorithm is used to predict the output y given variables x. It is called supervised as it is being taught by the dataset and it is informing the models decisions (41). In the following sections some supervised learning models that were used will be outlined. As there are a wide variety of classification algorithms available for use, a handful were chosen to be tested. They were chosen as they are simple to comprehend and employ an approach that should be able to perform well on the imbalanced dataset.

K Nearest Neighbours

The K nearest neighbours algorithm or KNN for short operates by placing instances from the dataset in a feature space. When a new instance is being evaluated it places the instances on the feature space and calculates the k nearest neighbours using a distance function. This new instance would be then classified based on majority class of its nearest neighbours (42). It was hoped that this approach would perform well when identifying the bearish classes as its nearest neighbours should also be bearish.

Naïve Bayes

Naïve Bayes describes a collection of algorithms that are based on Bayes Theorem of probability. Essentially the probability of each feature is calculated independently of each other and an overall probability is calculated. The class with the highest probability will be chosen for the prediction (43). It is hoped that the makeup of negative twits is consistent so that the probabilities will accurately reflect the data and balanced decisions will be made.

Decision Trees

Decision Trees are a type of machine learning algorithm typically used in classification tasks. They work by analysing a dataset in the effort of constructing comprehensive questions which can be used to predict a class. A metric called information gain is calculated at each node in the tree to find the best characteristic to split the dataset on (44). Figure 13 shows an example of how decision trees operate.

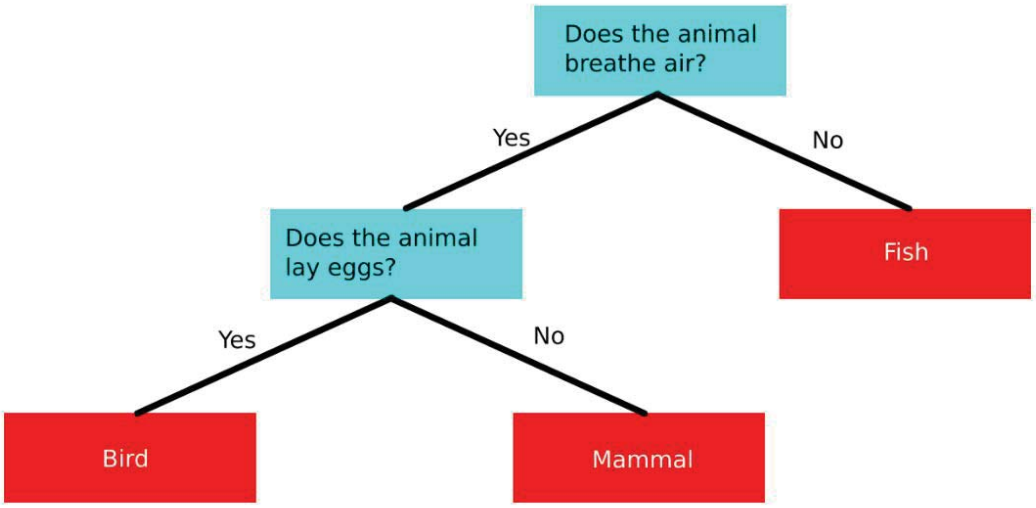

Figure 13 : Example of a Decision Tree (44).

Random Forest

Random Forest classifiers build on top of regular decision trees by using multiple decision trees and combining the output to enhance the classification. Each decision tree will act on a random subset of the data. Instead of calculating the best information gain for all features it attempts to find the highest information gain from a random subset of the features (45). It is hoped the multiple decision trees and randomisation will produce results with a wider variety and prevent biased classifications.

Data Sampling Techniques

After data exploration some research was done to investigate ways of enhancing or balancing the current dataset to help the classifier produce more balanced classifications.

Oversampling

Oversampling in this case would refer to randomly generating more data points for the minority class which is bears. The idea being that if the imbalance is addressed by inflating the data with bearish data points, the classifier will not be biased when making predictions. Synthetic sampling can also be employed to generate bearish data points using a nearest neighbour technique similar to that of KNN (46,47). Generating fake data points may influence the classifier to make decisions on false patterns that do not exist in the actual data.

Undersampling

Undersampling takes an opposite approach to oversampling by randomly removing data points from the majority class which is the bulls (47). This will address the imbalance making both classes the same size. However doing this would significantly decrease the size of the dataset from 46,000 to roughly 15,000. This could lead to important data being missing from the dataset.

Data Scaling

Data scaling is the process of standardising the dataset into a specific range of values. An example of this would be scaling all values to lie between 0 and 1. This should help the algorithms performance as it is not calculating large values.

Algorithm Optimisation

Many of the algorithms chosen are provided by Scikit Learn and as a result accept parameters to customise the algorithm. For example, the number of decision trees used by the Random Forest classifier can be specified in a parameter. In order to find the best parameters automated tuning can be performed.

GridSearchCV

GridSearchCV is a tool provided by Scikit Learn that will test different variations of parameters on a machine learning algorithm. The parameters that yielded the best scores are provided once testing has concluded. This could provide a major boost to the algorithms efficiency and accuracy.

Conclusion

All of the above algorithms were thoroughly tested before one was chosen. All three classifiers were first trained on the dataset without any data sampling techniques applied and there scores recorded using a classification report. All three were then tested on the dataset again without any sampling techniques applied but the data was scaled. The algorithms performed slightly better overall when data sampling was applied to the data.

The algorithms were then all tested on scaled over and under sampled data. First, oversampled data was used. The process of generating the 40,000 data points necessary to address the imbalance took a significant amount of time and hindered the performance of the algorithm. After oversampling, undersampling was tested and it too

made the algorithms perform worse. The imbalance would be left in the dataset as both sampling techniques failed to improve the accuracy of the algorithms.

Test Classification Report

	precision	recall	f1-score	support
bearish	0.76	0.71	0.74	841
bullish	0.94	0.95	0.95	3902
micro avg	0.91	0.91	0.91	4743
macro avg	0.85	0.83	0.84	4743
weighted avg	0.91	0.91	0.91	4743

Figure 14 : Model accuracy report

The algorithm that had performed the best out of all three was the Random Forest Classifier. It had the highest accuracy on the minority class (bears) out of the three algorithms so it was chosen. It scored nearly 10% higher in terms of accuracy on the minority class with a highest score of 76 % for precision and 71% for recall. The recall and precision metrics will be explored in further detail in the testing section of the report.

After the algorithm had been selected a GridSearchCV parameter tuner was fitted to the algorithm to find the parameters that best improved the models accuracy. A Google Compute Engine virtual machine with 64gb of Ram and 16 CPU cores was used to run the GridSearchCV on the algorithm. Attempts to tune the model on modest hardware failed due to resource constraints. The parameter tuning took three hours on account of the large dataframes being used to train the model. The tuner found 200 n_estimators to be the best fit for the model with an accuracy slightly higher than that of the default model parameters. The n_estimators parameter is used to tell the model how many decision trees should be used when making a prediction.

Due to the reasons outlined above Random Forest classifier with 200 decision trees was chosen as the classifier for the sentiment analysis API and machine learning pipeline.

Technologies Researched

Languages

Python

Python has been chosen for Stockify's technical stack as the majority of the features to be implemented into the platform are typically constructed with Python. Popular machine learning frameworks and packages such as TensorFlow and Scikit Learn are available in Python. Other necessary features such as the web scraper will make use of Python packages such as Python-XBRL for XBRL parsing.

As the majority of the features will be built using Python and its libraries it makes sense to encapsulate these features in a Python web infrastructure. Python has several web frameworks such as Django which is a web development framework for developing reliable web applications and APIs.

JavaScript

JavaScript will be implemented in the Stockify front end for a number of reasons. This project will also make use of a charting library called ChartJS which is a package for JavaScript to display information in graphs and charts. This will be used extensively to present pricing and other information to the users.

Packages

Python

Along with Python and its frameworks Stockify will be employing packages for various tasks such as machine learning, data preparation and Google Cloud operations. The most important Python packages will be outlined below.

Sci-Kit Learn

Sci-Kit Learn is a scientific package for Python that provides developers a wide range of tools with regards to machine learning. In Stockifys case it is used extensively for Sentiment Analysis. It offers many useful features such as data cleaning functions, data scaling, accuracy reports and the machine learning algorithms themselves.

NLTK

NLTK is the Natural Language Tool Kit, it provides a plethora of solutions for natural language processing that will be used for the sentiment analysis portion of this project. NLTK provides functions to clean and prepare textual data. For example, it provides a lemmatiser to shorten words to dictionary form to reduce unique words being processed. NLTK also provides functions that remove stop words to further reduce the amount of unique words present in the dataset making it easier for the algorithm to distinguish patterns.

Pandas & NumPy

Pandas and NumPy are packages that provide helpful functions with regards to data preparation and cleaning. In Stockify they are used extensively in the financial API to calculate portfolio optimisations. Pandas allows for manipulation of large datasets easily with its dataframe object and NumPy provides extremely useful matrix maths functions.

Rq

Rq is a Python package that allows developers to leverage a standard Redis cache as a Python job queue that remotely executes Python code. This package will be used in sentiment analysis data gathering to delete virtual machines once data has been collected.

Python-XBRL

Python-XBRL is a Python package that is used to parse XBRL documents. Companies release financial statements in both a PDF report and an XBRL document. The XBRL document is to be computer readable and easily parsed by computers. It is implemented as part of the web scraper.

Pickle

Pickle is a package used to store Python objects in a file. This allows developers to store objects, lists and dictionaries if needed. It is used to store the machine learning classifier and global bag of words in the Pipeline API.

Requests

Requests is a package that allows developers to make calls to the web using HTTP protocols such as GET and POST. It is used extensively when communicating with external and internal API's.

Pymongo

Pymongo is a Python package that allows for MongoDB queries to be executed in Python code. It is used to retrieve and store data in the MongoDB instance.

Google API Client

The Google API Client is a Python package that allows developers to create and deploy Google Cloud resources in Python scripts. It used to create the data collection cluster in the machine learning pipeline.

Feed Parser

Feed Parser is a small Python package for parsing RSS feeds. The package is used by the rssreader.py file in the web application which is responsible for collecting Seeking Alpha articles.

JavaScript & CSS

Bootstrap

Bootstrap is a front-end CSS framework that provides templates for page elements such as divs. Bootstrap is extremely useful for making responsive and neat webpages. Bootstrap is used to layout the entirety of the webpages in the project and it is responsible for making the application responsive to different screen sizes.

ChartJS

ChartJS is a JavaScript library that provides clean looking and interactive charts and graphs. ChartJS offers a wide range of charts and graphs of which the application employs bar charts, donut charts and line graphs to display information such as pricing and sentiment for the user.

Frameworks

Backend – Django, Django REST Framework and Flask

As mentioned previously, the features such as machine learning and web scraping will be built in Python due to the packages available. It is much easier to run Python code in a Django view rather than making a request to an API written in another language such as R. This is why Django was chosen.

Django is a mature framework and provides many useful tools out of the box. An example of this is user authentication. User authentication and sessions are baked in and don't require code to get them up and running. Django also provides an admin page where developers can view, edit and delete entities such users without having to query a database with SQL.

Django and the Django rest framework allow developers to store Python objects easily in MySQL and PostgreSQL with the use of an Object Relational Mapper. These Python objects are further enhanced by the fact that they can be serialized in an API and returned as JSON with very little code, making interaction with JavaScript in the frontend much easier.

Flask will also be implemented to the project. Flask is an API framework similar to Django. Flask is much more stripped back and offers less in the way of features compared to Django. It is used in place of Django for the sentiment analysis API as only a small-scale API is needed. Not much work is being performed by the API and it is not exposed to the web, so security is not as much of an issue. This allowed more time to be spent on perfecting the machine learning and data preparation for the sentiment analysis rather than the hosting.

Frontend Frameworks – Django

For the final implementation of the project Django was chosen. There were multiple reasons behind this. Firstly, the project does not consist of many pages and Django provides great templating and dynamic HTML injecting with Jinja. Jinja allows developers to iterate through python objects and add them as elements to the DOM which is then displayed on the webpage. The second reason was that single page functionality was not necessary as page loading times were reduced with the introduction of a cache. Providing a SPA page with multiple AJAX calls would be pointless as the page would load quickly on its own. Thirdly, flexibility wasn't an issue. JavaScript frameworks such ChartJS work great when implemented with Django and continuous monitoring of data sources like APIs wasn't necessary due to the time horizon of the data in use.

Databases: PostgreSQL vs MongoDB

As a way of comparing NoSQL and SQL a comparison will be made between the two of the more popular implementations. These are MongoDB and PostgreSQL.

MongoDB is a non-relational database that stores information as documents in BSON a binary interpretation of JSON or JavaScript Object Notation (48). Fields are stored as key value pairs inside of collections and are queried using the MongoDB query language which is quite similar to JavaScript. MongoDB does not have forced schemas and is inherently flexible which allows for easier storage of objects (49). The positives for MongoDB are its flexible nature which allows developers to quickly store data due to the lack of a fixed schema and its similarity to JSON. This makes it a comfortable fit for developers (48). It also allows for better scaling as implementations grow due to its built-in sharding features (48).

PostgreSQL is an object-relational database system. It has been in operation since 1986 and is quite mature at this stage. Like other relational database systems, it enforces a strongly typed schema and sports ACID compliance. However, PostgreSQL supports JSONB and arrays (50). PostgreSQL utilises schemas but also supports JSONB and allows for storage of unstructured data while maintaining its overall structure (50). PostgreSQL has a lower memory foot print than some NoSQL implementations such as MongoDB when highly complex relationships exist between the data (51).

Both PostgreSQL and MongoDB are utilised for Stockify due to the following reasons. The flexible nature and the ability of MongoDB to scale will be useful for storing large amounts of data quickly when gathering data for sentiment analysis. For the Web application PostgreSQL will be the main database. It is hoped that in the long run a strictly enforced schema and the ability to support JSONB means that a schema can be defined while maintaining the option to store unstructured JSONB data. This may be useful when storing user related and general data.

Hosting Platform

The final project hosting implementation will mainly be using Google Cloud. The main reason for choosing Google Cloud was the amount of credits provided. Certain solutions would not be possible on other platforms as it would be too expensive. Google Clouds credits allowed for the implementation of cluster computing for data aggregation and the use of a powerful virtual machine to best train the sentiment analysis algorithm. Google Cloud also provides a great API that allows developers to create virtual machines and assets. The API comes with great documentation and code examples from Google itself. Google Clouds range of solutions are also excellent, and the project makes use of quite a few of them such as the Cloud Scheduler, Compute engines and VPC networking.

The main database for the web application will be hosted on AWS. The reason for this is that AWS was experimented with in the prototyping phase and efforts to port the database to another solution such as Google Cloud SQL has failed. This was due to differences in the versioning of PostgreSQL.

Docker

Docker is a tool to make application deployment easier by hosting applications in containers. Containers allow developers to specify requirements such as language versions and libraries inside the container and ship it out as one cohesive unit. Each docker container is similar to a virtual machine but differs slightly as it allows applications within the container to utilise the underlying Linux kernel. Containers are shipped with the necessary additions such as the libraries and packages mentioned before (52,53). This means that every time an application needs to be setup on a new machine, system administrators do not need to manually install each package. All necessary additions are in place before getting the server online as they are already present in the container.

For Stockify, Docker is used to create containers with the python libraries and version necessary to make the Django web application and API run on Google Cloud. Docker is also used in the data collection cluster of the machine learning pipeline.

Dockerhub

Dockerhub is an online repository offered by Docker. It provides a similar service as GitHub for Docker containers. When a Docker container is built the developer has an option to push it to Dockerhub from here the container can be pulled again from anywhere and ran. This makes deployment much easier and it is used to deploy code to the nodes of the data collection cluster in the machine learning pipeline.

Caching with Redis

Redis is an in-memory key-value data store that is used in Stockify to cache API data (54). The reason a cache is necessary is due to the API limits in place by providers such as Twitter and StockTwits. If these API's are over used by the application, access will be blocked. Redis was chosen as the cache for many reasons, firstly it is very quick as it was built in C and it runs in memory (55), so response time is kept to a minimum. Redis supports useful packages such as Rq which is used in the sentiment analysis pipeline to delete virtual machines. Redis has proven quite adaptable also as it not only supports Python job queues but also allows for the storage of full JSON objects in memory. This means that data does not have to be converted by the backend before being cached.

Continuous Integration with Jenkins

Continuous Integration is a process by which code is compiled and tested prior to being pushed to a code repository (56). For Stockify, automated testing is performed when any changes are made to the codebase. Jenkins was chosen because it is non-proprietary and free for anyone to use.

Data Sources

Introduction

As highlighted in the previous sections, multiple sources are necessary to be aggregated to provide the information necessary for the platform. Time was spent assessing data sources such as API's and RSS feeds to determine how and where this data will be collated from. The data sources that were chosen for the project will be highlighted below.

Intrinio API

When it came to collect financial information about companies such as pricing and ratios, the intention was to build a web scraper to collect this information. The web scraping approach turned out to be unviable, this will be addressed in the development section of the report. After some research into financial APIs it was decided that Intrinio would be the data source chosen. Intrinio is paid API that provides a wide range of information on the stock market such as stock prices, financial ratios and financial filings. Intrinio offers a student pricing plan that made it affordable for use in this project. From researching other data sources Intrinio appeared to be the only one that provided the necessary data at an affordable price. It provides stock pricing, financial ratios and financial filing links for the web application.

StockTwits and Twitter

As outlined previously in the stock research process, time is spent gathering opinions of others from platforms such as Twitter and StockTwits prior to making decisions when investing. Seeing other opinions not only opens investors to different viewpoints but gives an indication of the market sentiment surrounding the stock in question. It was important that social media feeds were present in some fashion in the project. The decision was made to access data from Twitter and StockTwits. The reasoning behind Twitter and StockTwits was simple. Both platforms provide character limited posts that quickly convey an opinion making it easier to present multiple posts on a single page. Presenting reddit or forum posts would not work as they tend to be quite large and would take up significant space. Both platforms are massively popular in their own

respects, therefor there wouldn't be a shortage of useful data to work with. StockTwits allows users to indicate an opinion either bullish or bearish which was used for implementing sentiment analysis. Both share a similar post structure which will allow the algorithm to analyse content from both platforms easily.

Seeking Alpha

Seeking Alpha as outlined in the existing solutions section is a platform that allows users to post analysis and news on stocks. The content on the platform is written by staff and investors using the site. Often it provides extremely interesting insights to aspects of the market. As new information and keeping up to date with events on the market is very important it was necessary to have some source of news present in the project. Seeking Alpha was chosen as it provides news and user research/analysis in its feed making for an insightful and varied feed for users. Other platforms were considered such as Intrinios news feed which mainly consisted of Yahoo News and a few other sources. The problem was that these feeds contained multiple duplicates and articles referring to the same event or information making a significant amount of the data on the site redundant.

Financial Modelling Prep & Clearbit

Financial Modelling Prep and Clearbit are used in tandem to collect information on companies through a Python script. Financial Modelling Prep is an API that provides information on companies such as their CEO, website, business sector and industry. Clearbit is an API that provides company logos using the company's URL. FMP and Clearbit were chosen because they were the only free solution that provided business descriptions and logos. FMP went offline for a significant period during the early stages of the project. A decision was made to write a script to collect information for all companies and to store it in a PostgreSQL database before the API went down again or permanently.

Pandas Datareader

Pandas Datareader is a Python package that provides stock pricing in the form of Pandas dataframes. It is used extensively in the financial API. The package returns the information in a Pandas dataframe which makes it easier to manipulate when performing transformations on the data. Pandas Datareader could have provided the pricing for the website. However, there have been issues with the package in the past not supplying data for a period of time so Intrinio was used in its place.

Conclusion

Having researched the basics of the stock market and why it is used. The important aspects can be identified and focused on such as risk mitigation and maximising returns.

Examining a current example of the investment process has highlighted the aggregation necessary to provide such a wide range of information to users. It has also served to point out data sources for the project. The project must find a way to not only aggregate the data successfully but to also present it in a useful manor. The data should also be enhanced with machine learning and other techniques to improve the insights it provides to the user.

Evaluating existing solutions provided insight on options available on the market and where this project can fit amongst them. It also served to highlight the importance of presentation. The project should implement some sort of visual aid when presenting information to the users such as charts and graphs.

Identifying the lack of portfolio optimisation and sentiment analysis amongst existing solutions lead to their inclusion in this project. Their implementation is key to this project as they will set Stockify apart from other solutions and provide interesting insights on the collected data.

Chapter 3 – Design

Introduction

Having concluded background and technical research a design must be put in place for the project. The designs covered in this chapter of the report will define the structure of both the frontend and the backend of the platform. The designs will consist of multiple iterations of prototypes for the front end as well as other documents such as use case, architectural and database diagrams. A design methodology must also be considered and implemented prior to design taking place.

Methodology

Agile Methodology

Agile is an umbrella term for multiple software development methodologies. It promotes a highly iterative process for application development. It allows for parties involved in development to receive and provide feedback early and often. Each implementation of Agile incorporates continuous development, testing and integration, ensuring that quality is guaranteed at every stage of an applications development.

When compared to traditional frameworks such as Waterfall, Agile can be considered quite minimal as it does not enforce strict rules or practices and is meant to be adaptive to the development processes of the team employing the framework (57).

The most popular implementation or subset of the agile methodology is the SCRUM Framework.

SCRUM Framework

The Scrum process is a framework based on the Agile methodology and aims to allow developers to continuously iterate through design development and gather feedback in a specified time frame (57,58).

Scrum Process

The development process typically consists of two-week periods known as sprints.

Before each sprint the team gets together to have a sprint planning session where the product owner and scrum master allocate stories or jobs to team members.

The team will work on their tasks for roughly a week. Every day they have a stand-up or scrum meeting where the team get together and discuss how they are doing in regard to their task.

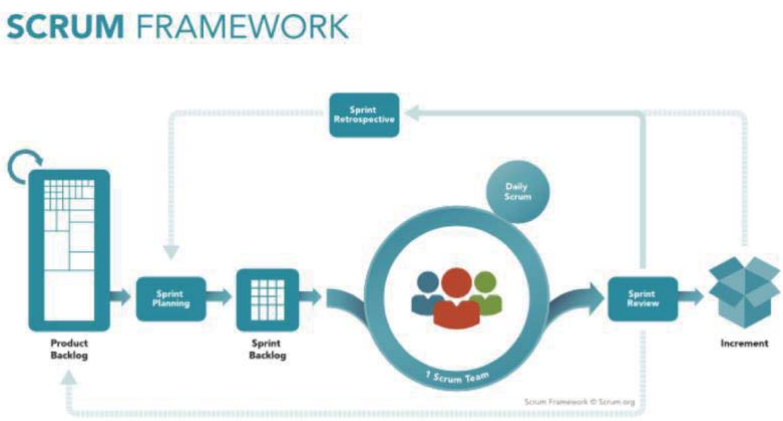

Figure 15 : Scrum Process (58)

After a week the team will get together to have a grooming session to discuss how tasks are progressing and to see if they need to refine the timeline for certain stories or if they can take on additional tasks for the rest of the sprint.

The team will continue to have daily stand ups until the end of the sprint when they will have a sprint review to go over what happened during the sprint. Team members typically present tasks they were working on and other teammates provide feedback.

After the sprint review the team will get together with the scrum master and have a retrospective where they discuss what went well and what didn't go to plan. They also discuss any obstacles that could have been avoided or limited to allow the developers to be as efficient as they can.

After the sprint has ended the next one begins, and the process continues the same as before until the product or project is deemed feature complete.

Why Agile / Scrum was chosen for this project.

There are a few reasons why scrum was chosen for Stockifys development. Firstly, the Author has prior experience being a part of a Scrum team during work placement. The Author found the approach to be quite effective and allowed them to achieve and maintain efficiency in completing their tasks.

The second reason for choosing Agile Scrum is flexibility. Flexibility is built into any agile based frameworks core. This allows for features to be revisited at any stage to address issues or implement design changes. Due to the nature of this project this may very well be useful as issues arrive during development.

Finally, Scrum fits well with the final year project schedule. The typical two week Scrum period can changed to weekly sprint periods where each sprint concludes with a meeting with the supervisor. The supervisor meeting will be the sprint review where feedback will be given and issues can be discussed.

Feature List

Now that research has been completed and a design methodology has been chosen the finalised list of features for the platform should be outlined.

Features

- *Sentiment Analysis*
  - The platform should provide sentiment analysis on the latest tweets and twits for every stock.

- *Stock information*
  - Information should be provided for each stock, this will include multiple data sources such as

    - Pricing
    - Company information e.g. Name, logo and CEO
    - Financial ratios
    - Financial filings
    - News feeds
    - Social feeds

- *Information visualisation with charts*
  - Information such as pricing should be presented numerically and in a chart or graph.

- *Portfolio Optimisation*
  - The platform should provide analysis on users portfolios if they decide to create one.

- *User portfolios*
  - Users should be able to create a portfolio and add any stock on the platform to their portfolio.

- *User registration and login*
  - Users should have the ability to create an account on the platform and be able to log in and out of the account at will.

Use Case Diagram

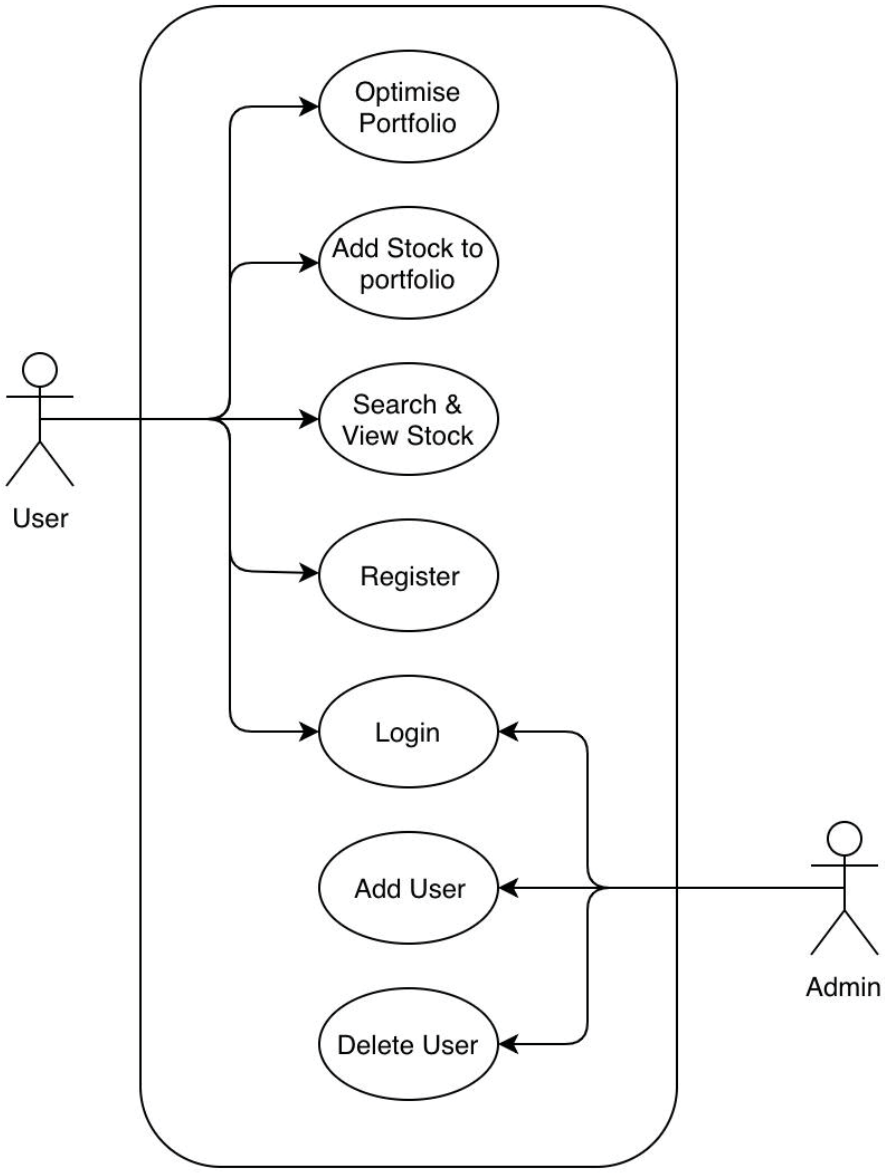

Figure 16 : Stockify use case diagram

Frontend Design

Introduction

This section will outline the two iterations in the frontend design prior to final implementation. The first is a low fidelity drawn prototype that provides an idea of how the pages may be structured. The second iteration was medium/high fidelity prototype used in the interim submission. This served to show how the requirements can be presented in a real web page.

Low Fidelity – Iteration 1

The first prototype for the platform was drawn on paper in an attempt to design a page structure. It serves as a crude mock-up of the final application. These mock-ups were used to provide a sense of direction when implementing the interim prototype in actual code. The two mock-ups represent the home page where portfolio analysis is provided and the stock page for individual stocks are visible in figure 17.

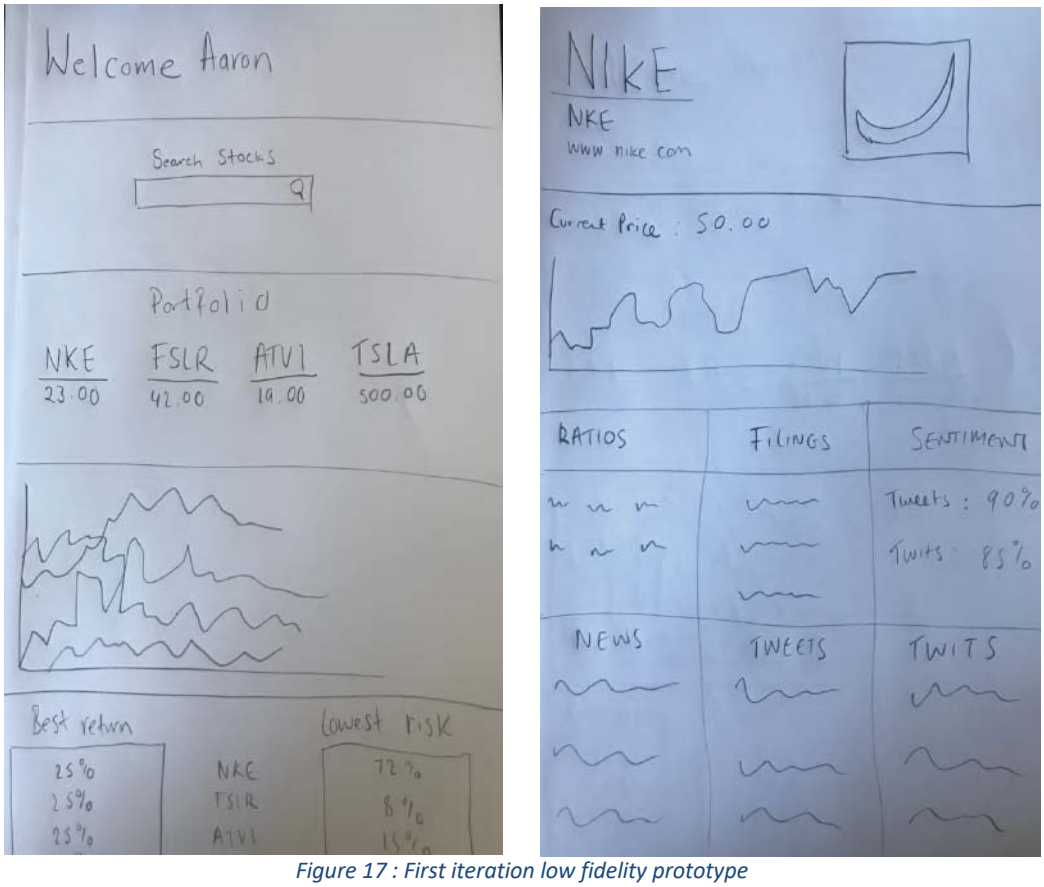

Figure 17 : First iteration low fidelity prototype

High Fidelity – Iteration 2

The second iteration of the design was implemented fully in HTML and CSS. It was designed using the previous iteration as a guide however some changes were made due to difficulties placing elements correctly in HTML. Changes were also made to enhance the look and feel of the design and make the site easier to navigate.

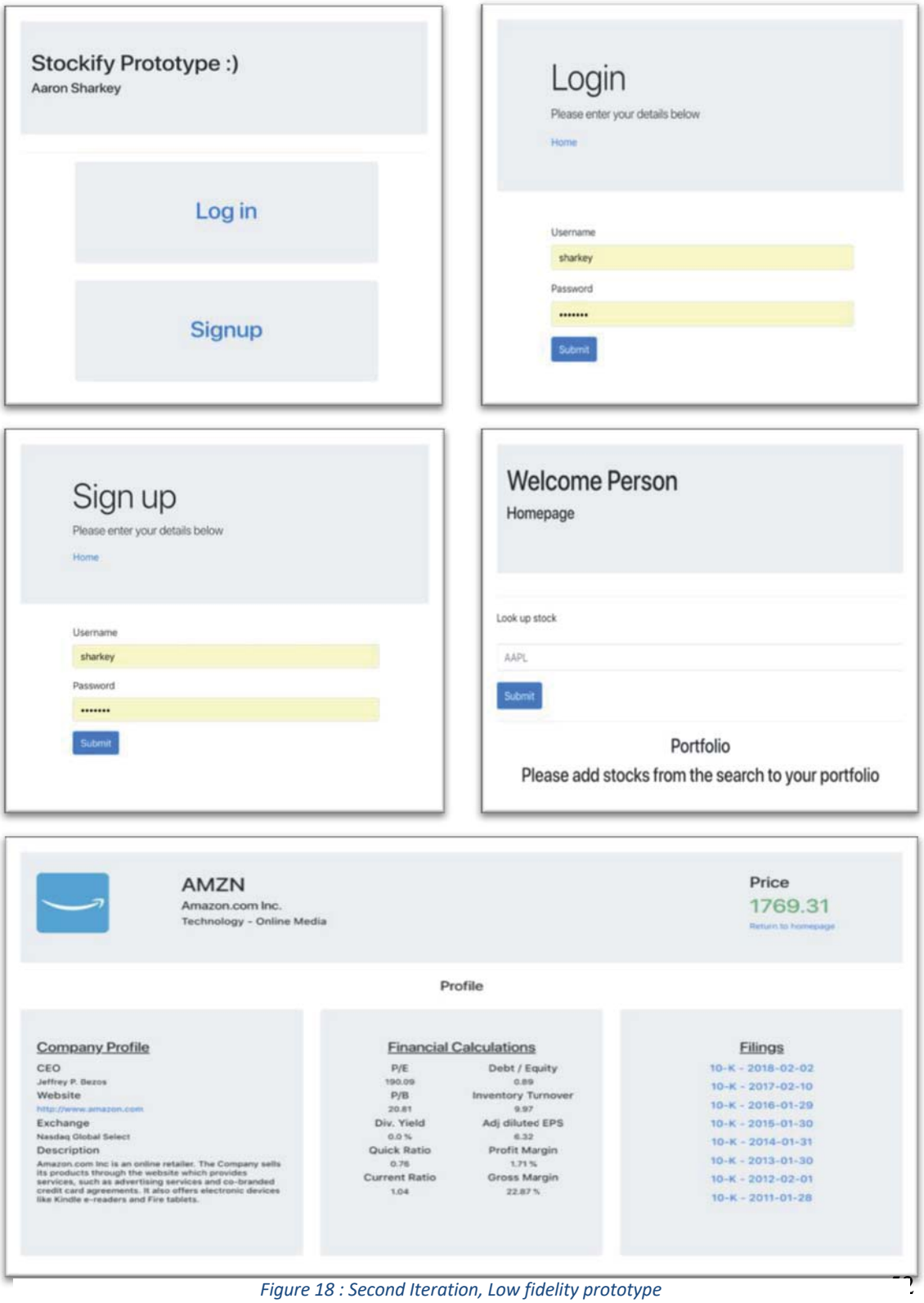

Figure 18 : Second Iteration, Low fidelity prototype

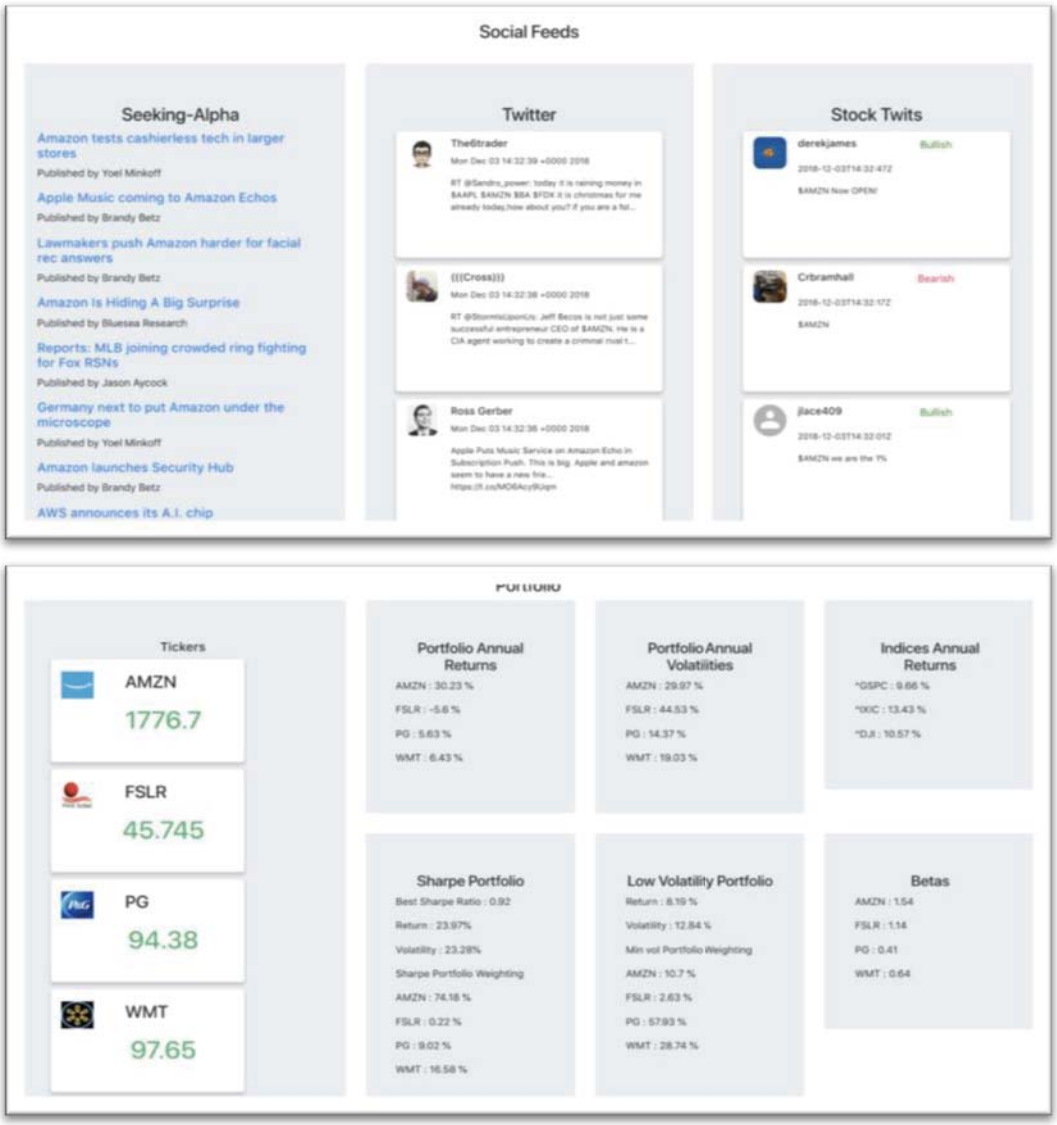

Figure 19 : More of the second iteration high fidelity prototype

It was also at this stage in the development that potential users were invited to use the platform and provide feedback on the design. The platform functioned fully as a website at this stage. However, as it was a prototype and some features weren't completed on time such as the sentiment analysis and charts.

Overall feedback was good from the test users. Users did note the lack of charts and graphs. Some felt that the information on the platform would be much easier to consume if it was visualised. Other users noted that the addition of a search bar on the stock page would be useful as users must navigate to the homepage to search for stocks again.

Project Architecture

Introduction

The following section of the report will outline the technical architecture of the project. This will outline the web platform design which consists of the web application, backend API's and external data source access. This section will also outline the machine learning pipeline architecture and examine the Cloud Scheduler, Pipeline API and external data source access.

Stockify Web Platform

The web platform will follow a three tier application consisting of a presentation, logic and data tier. This approach was chosen to enforce separation of concerns meaning that if changes are made to one tier it shouldn't affect other tiers. The system will be further split up into micro services in the logic tier. This further serves to enforce separation of concerns and allows for the web application to still function if certain aspects of the logic tier fail. If for example the sentiment analysis microservice goes down the frontend will still operate just without up to date sentiment data.

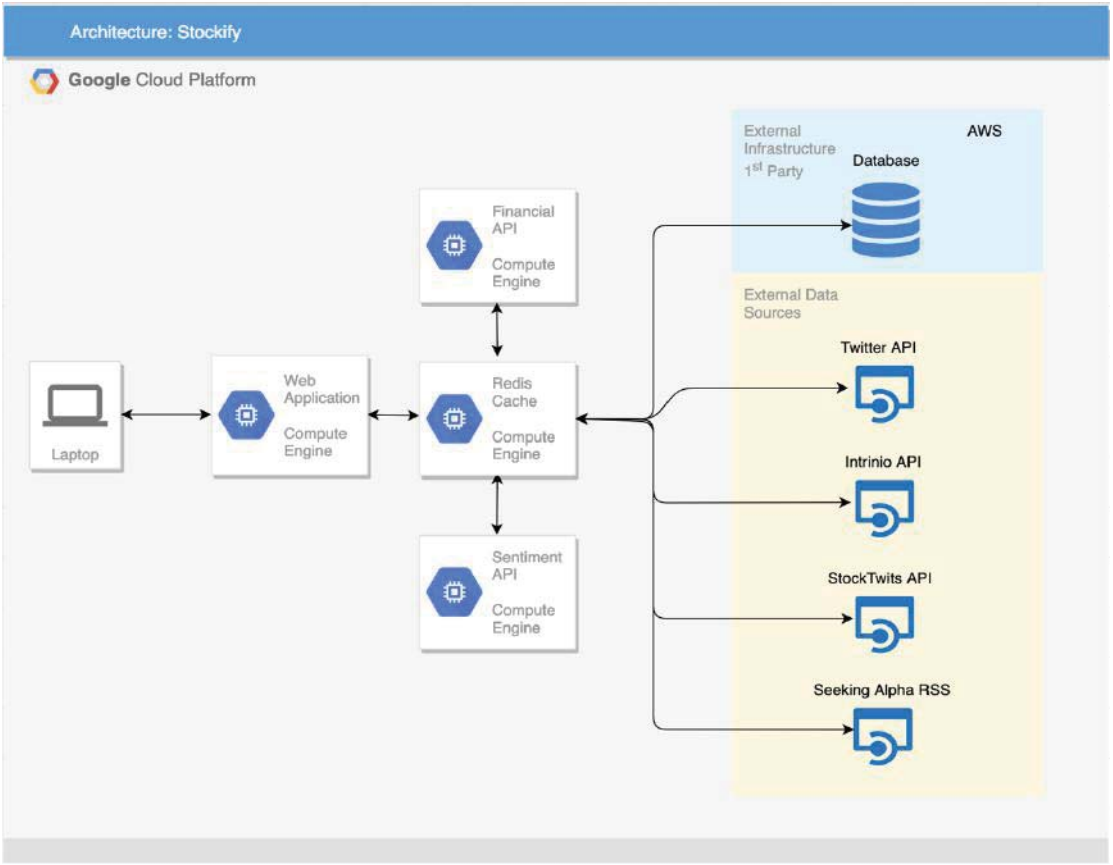

Figure 20 : Web platform architecture

Presentation tier

Web Application

The presentation tier will consist of the web application which the user will interact with. Here the user will be able to use all the features the platform provides. The web application will be responsible for collecting the data from the logic tier and certain external data sources. Once data is collected, the web application will process it and present the data on a webpage for the user.

Logic tier

Financial API

The financial API will be responsible for performing portfolio analysis on user portfolios. It will accept data in from a HTTP request containing JSON. The API will collect information regarding the portfolio in the request and perform analysis on the data before returning the results to the web application.

Sentiment Analysis API

The sentiment analysis API will be responsible for performing sentiment analysis on a given company. It too will accept data from a HTTP request containing JSON. When the API has received a request it will collect social media data about the company. Once the data is collected it will be cleaned and analysed by the machine learning algorithm and the results will be returned to the web application.

Redis Cache

The Redis cache will also serve an important role for the platform. It will store information from the external APIs temporarily to ensure API limits are not exceeded and response time is minimised. The web application, financial API and sentiment API will query the cache for data before querying certain external data sources.

Data Tier

Database

The database contains information for the website such as user information, user portfolios and company information. The web application will pull information from here to present to the users on the webpage.

External APIs

Both the presentation tier and the logic tier will be extensively using external APIs to gather, analyse and present data.

Example Web Platform Workflow - Sequence Diagram

Below is an example of the work performed to load a stock page. It outlines the work required to view a stock page and the requests / processing necessary to aggregate and present all the data. This sequence diagram assumes the cache is empty.

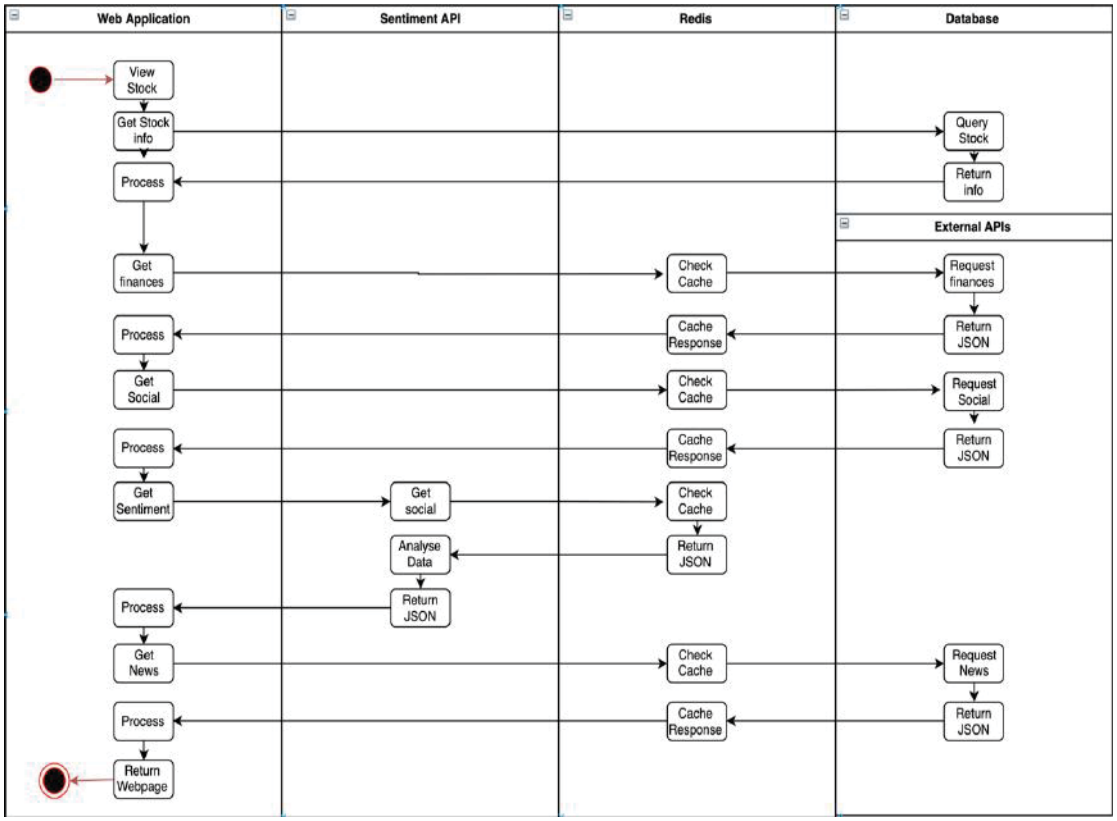

Figure 21 : Example workflow for the web application

Example Web Platform Workflow – Flow chart Diagram

Below is a use case diagram that outlines the actions available to the user on the system. It shows an example of the typical user flow on the platform.

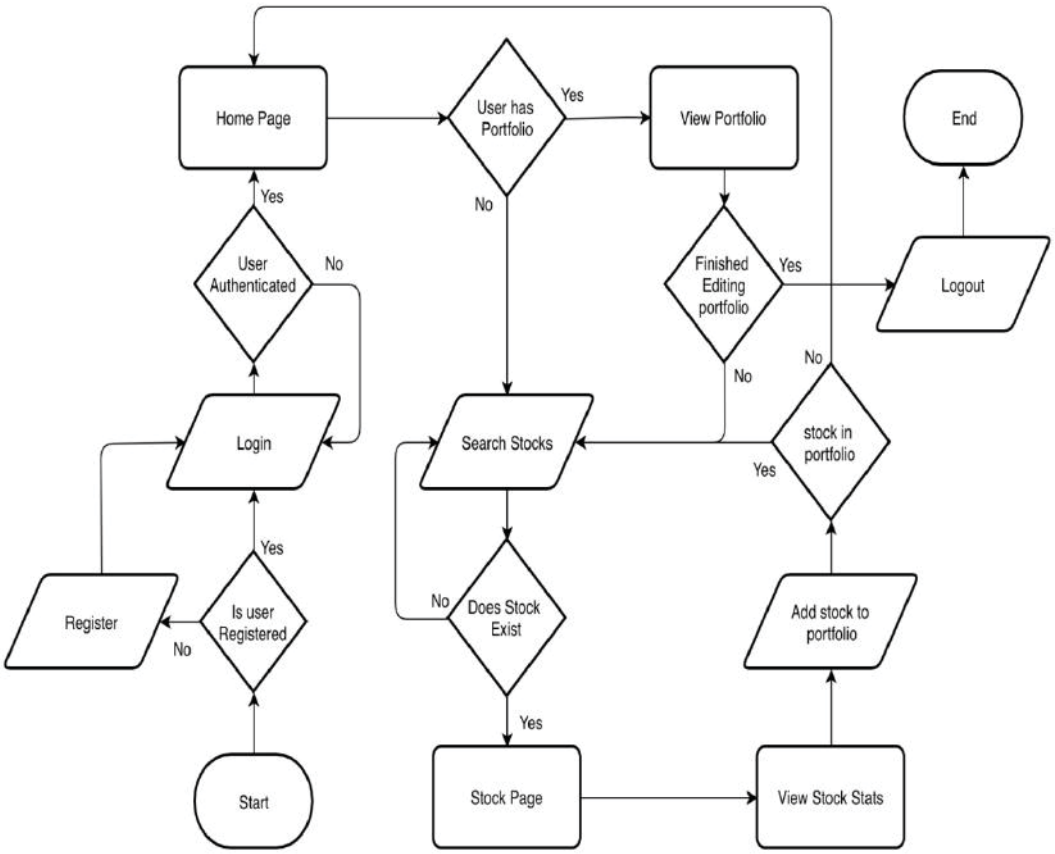

Figure 22 : Web application flowchart

Machine Learning Pipeline

The machine learning pipeline conforms to a customised master-slave architecture. The Google Cloud Scheduler pings the Pipeline API to begin a job. The Pipeline API will act as master to a cluster of Google Compute Engine virtual machines who will request and store data from StockTwits. Once data collection has finished the Pipeline API will gather all the data within the database, clean and train an algorithm on the data, store the trained algorithm and email the results to the author.

The reasoning behind this architecture is data access limitations. The StockTwits API enforces a 200 call per hour limit on unauthorised calls to the StockTwits API. This limit is enforced by checking the external IP address of the caller and see if the caller has reached their limit. If a server was to collect 200 twits for all the companies in the database on its own it would take roughly 230 days to complete. Unfortunately the time horizon of this project and the data needs do not allow for gradual collection so cluster computing will be employed. By spawning several google cloud computers, each with a separate external IP address to collect twits, 30 twits for each company can be collected in roughly 15 minutes. Measures will be taken to ensure that the API is not being abused. For example the jobs will be ran during the night when the markets are closed and the pool of compute engines will be limited in size.

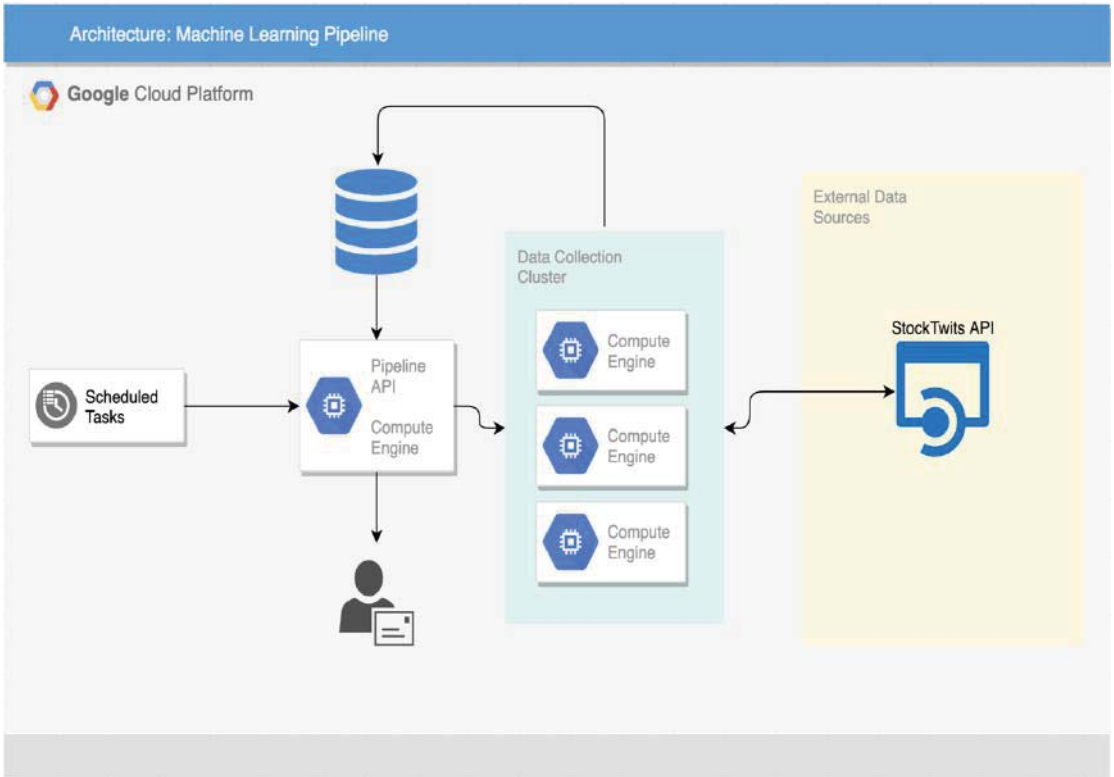

Figure 23 : Machine Learning Pipeline architecture

Task Scheduler Tier

Google Cloud Scheduler

Google Cloud Scheduler is a service offered by Google to allow developers to schedule a time for a job to begin. The developer provides a time in CRON format, for example "0 2 * * *" will run a job at 2am every day. When the time criteria is met the scheduler sends a get request to a URL specified by the developer. Google Cloud Scheduler is used to start data collection jobs on the Pipeline API at specified times.

Master Tier

Pipeline API

The Pipeline API will be the master with regards to this design. It will be responsible for spawning a cluster of multiple Compute Engine virtual machines to collect data. Once data has been collected by the cluster, the Pipeline API will gather the Stocktwits posts from the database, clean the data and train an algorithm on the data. Once the algorithm is trained it will be tested with a test set and the testing results will be emailed to the author.

Slave Tier

Data Collection Cluster

The cluster will be responsible for collecting data from the StockTwits API, processing the data and storing it in the database for use by the Pipeline API. Once each virtual machine is spawned it will download a Docker container. The container will contain a Python script and once ran will request data from the API and store it. Once a virtual machine has finished collecting it will queue a job to destroy itself using a Rq queue on the Redis cache.

Data Tier

StockTwits API

The StockTwits API will provide all the Bullish and Bearish labelled twits necessary to train a sentiment classifier.

Database

The database will be responsible for storing all the collected StockTwits data. It will accept multiple connections from the cluster when a data gathering job is in progress.

Example Pipeline Workflow – Sequence Diagram

The Sequence Diagram below outlines the typical flow of a machine learning pipeline operation. This diagram assumes the Data Collection Cluster section is following the path of a single node of the cluster. It is also assumed that this is the last collection node spawned and is responsible for starting the ML trainer job on the Pipeline API.

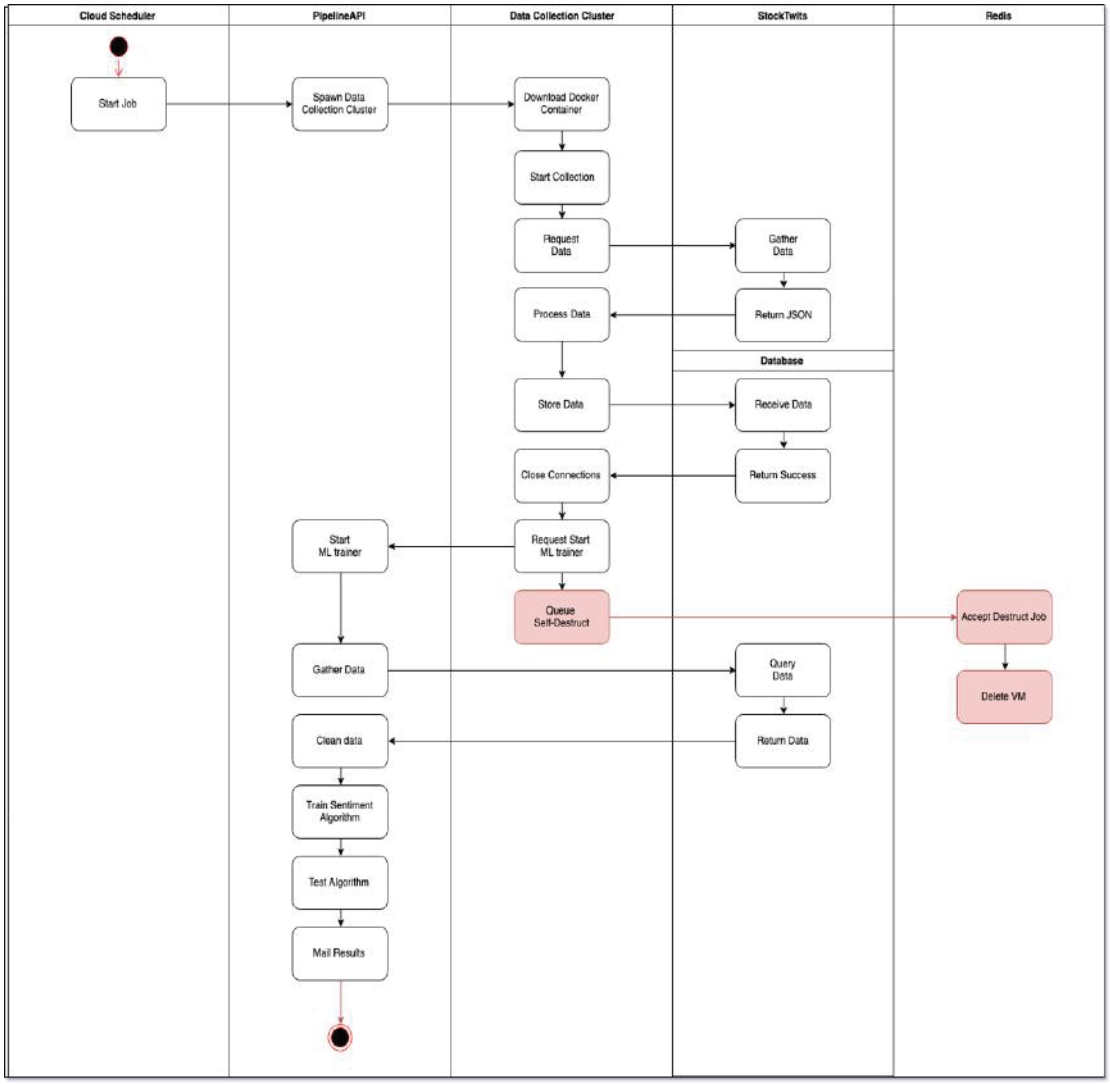

Figure 24 : Sequence diagram outlining the Machine Learning Pipeline workflow

Database Design

Introduction

The project will be employing two databases, the first is a PostgreSQL database for the web application. The second database will be a MongoDB database for the Machine Learning Pipeline. Both of the database designs will be outlined in the following sections.

Web Application – PostgreSQL

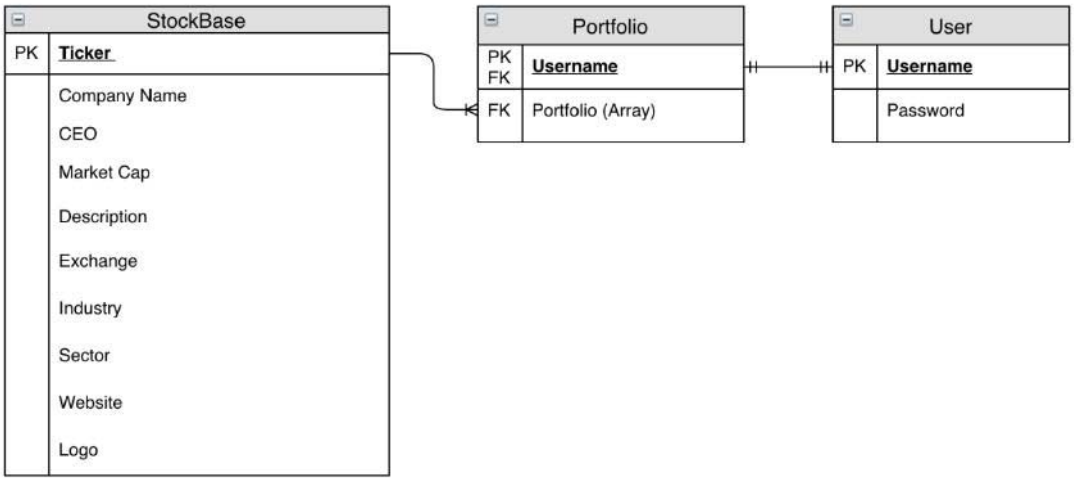

Figure 25 : ERD outlining the web applications database design

The layout for the database was designed around both the features specified for the project and the data available from the external APIs. From the requirements discovered during research it was essential that users could register an account and be able to log in and out. To accommodate this a user table was added that featured a username and a password. Then the portfolio was added, each user can only have one portfolio so a portfolio table was created with a one to one relationship with the user table. The Portfolio table employs the Username field from the User table as a primary / foreign key. To store the stocks in the portfolio it was decided to use a PostgreSQL array as this would cut down on the amount of tables necessary. When a user adds a stock to their portfolio it is appended to the Portfolio array.

During the research phase the stock information API called FMP was found to be unreliable. It was decided that a script would be written to collect business information for each stock and store it in a database. The table was designed to accommodate all the data returned by the API for each stock so it was modelled after the API response.

Machine Learning Pipeline - MongoDB

In order to train the sentiment analysis algorithm a large amount of StockTwits will need to be collected. This is the reason why MongoDB is employed for this section of the report. As MongoDB does not have a rigid schema or tables its design was much more straightforward. Similar to the previous database design, choices were made based on the data returned from external data sources.

As MongoDB documents are inherently JSON it made sense to store the JSON that was returned from the StockTwits API. Below in figure 26 is an example of a StockTwits JSON response. All documents in the collection have the same structure. This was done to minimise processing and to speed up data collection.


```
{
    "_id" : ObjectId("5c51f8b19a6aa000011036d8"),
    "id" : 151689444,
    "body" : "$MFT / BlackRock MuniYield Investment Quality Fund files form N-CEN/A https://fintel.io/filings/us/mft",
    "created_at" : "2019-01-25T17:07:45Z",
    "user" : {
        "id" : 127776,
        "username" : "risenhoover",
        "name" : "Wilton",
        "avatar_url" : "https://avatars.stocktwits.com/production/127776/thumb-1384448707.png",
        "avatar_url_ssl" : "https://avatars.stocktwits.com/production/127776/thumb-1384448707.png",
        "join_date" : "2012-01-05",
        "official" : false,
        "identity" : "User",
        "classification" : [

        ],
        "followers" : 1556,
        "following" : 27,
        "ideas" : 252237,
        "watchlist_stocks_count" : 1,
        "like_count" : 1022
    },
    "source" : {
        "id" : 2883,
        "title" : "Fintel.io",
        "url" : "https://fintel.io"
    },
    "symbols" : [
        {
            "id" : 6282,
            "symbol" : "MFT",
            "title" : "BlackRock MuniYield Insured Investment Fund",
            "aliases" : [

            ],
            "is_following" : false,
            "watchlist_count" : 17
        }
    ],
    "mentioned_users" : [

    ],
    "entities" : {
```


Figure 26 : Example of a StockTwit JSON

Chapter 4 – Architecture & Development

Introduction

This chapter of the report will outline the entire development of the project. Development began in September 2018 and has been ongoing since. Features were typically broken down in to units of work which could be completed in a 1 week sprint. Each aspect of the projects development will be examined in the order that they were developed.

Architectural Overview

As outlined in the previous chapter the project consists of two separate architectures, the web platform and the machine learning pipeline.

The web platform was implemented as designed and follows a three tier architecture. The Django web application acts as a presentation tier or frontend where the users interact with web pages. The web application communicates with the two backend APIs, the financial API which is a Django API and the Sentiment Analysis API which is a Flask API. The web application also pulls data from the PostgreSQL database and various external data sources such as API's. The web application checks the Redis cache prior to querying the external APIs for certain limited resources such as Twitter and StockTwits.

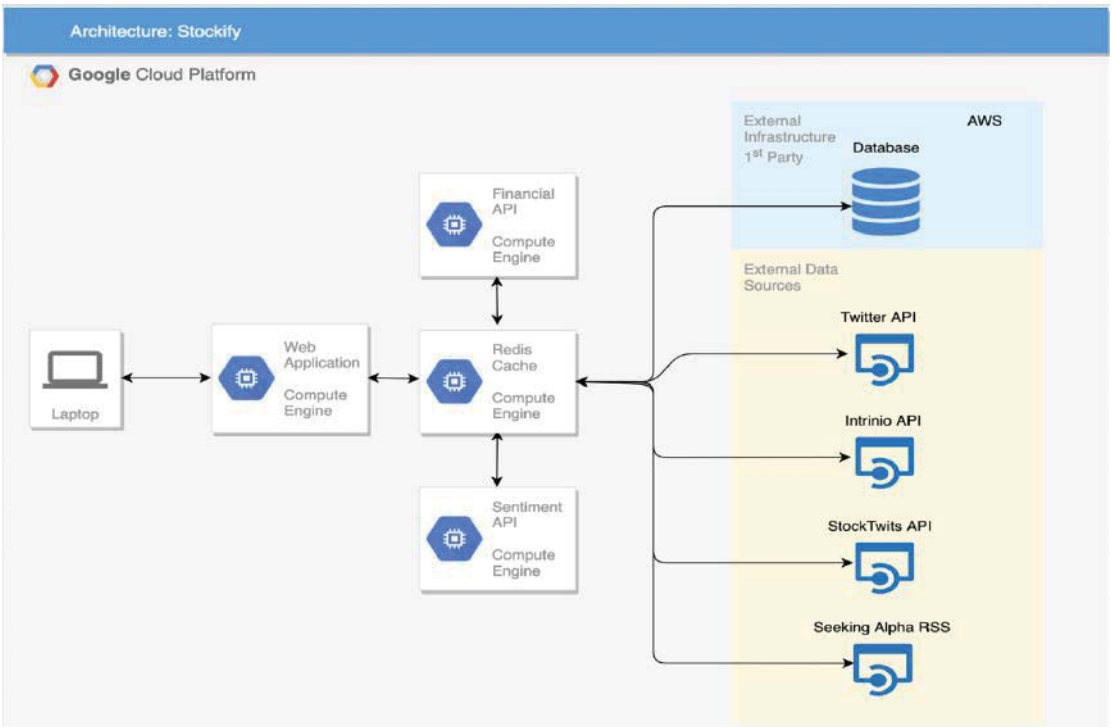

Figure 27 : Final web platform architecture

The machine learning pipeline was also implemented as designed in the previous chapter. It follows the customised master slave architecture put forward during the design process. The Google Cloud Scheduler Service sends a GET request to the Pipeline API when it is time to start a job. The Pipeline API is a Flask API that spawns a cluster of Google Compute Engine Virtual Machines to collect StockTwits data. Each virtual machine in the cluster downloads a Docker container and requests twits for 200 companies before storing it in the MongoDB database. The virtual machine will then queue a deletion job on the Redis cache queue to destroy itself. The last node created in the cluster is responsible for sending a request to the Pipeline API once it has stored the data and before it starts to self-destruct. The Pipeline API will then gather all StockTwits data from the MongoDB database. Once the data is gathered the API will clean the data, train a Random Forest Classifier on the data, test the predictions for accuracy and email a report to the Author.

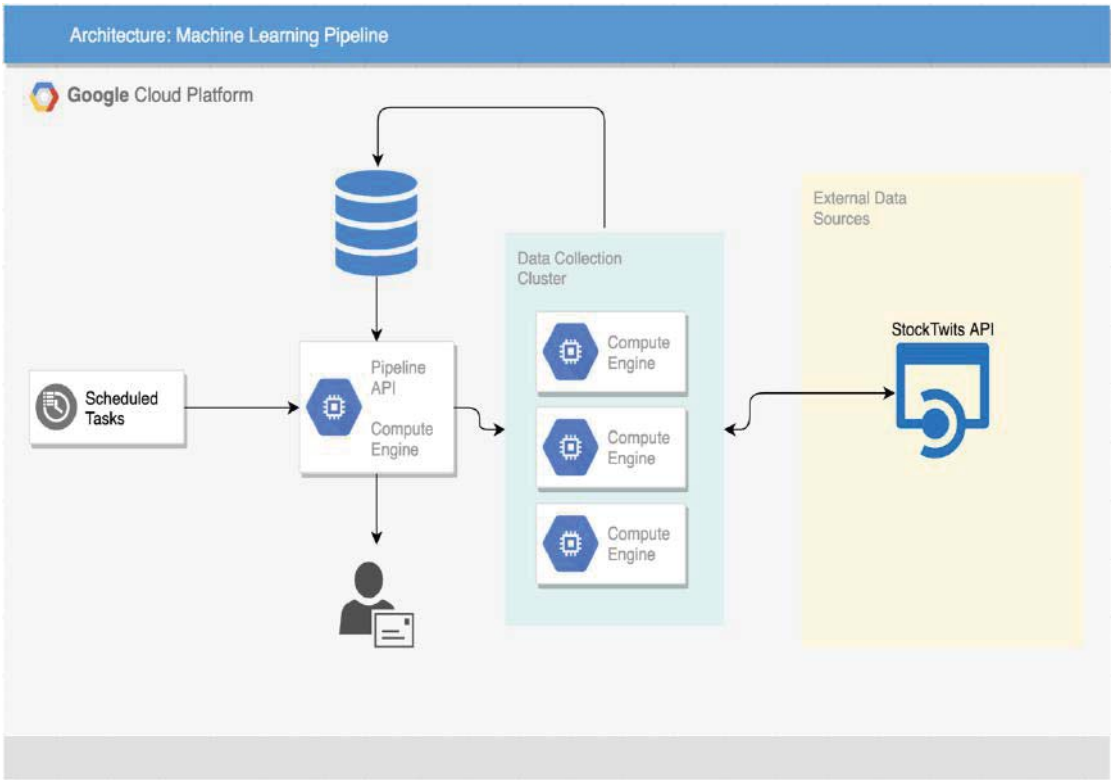

Figure 28 : Final Machine Learning Pipeline architecture

Web Platform Development

Web Scraper Development

Development began back in September on the web scraper that would be responsible for gathering financial data on companies. It worked by gathering XBRL financial filings from the securities and exchange commission's website. It was decided that a web scraper would be built due to the lack of API's that provided company financial ratios or key financial data for free.

A scraper was successfully built. It scraped the human readable HTML and XBRL financial filings for all companies on the Nasdaq and New York Stock exchanges by scraping the SEC EDGAR Database search. This was done by downloading a CSV with all the stocks listed on Nasdaq and New York Stock exchanges and looping through each stock making requests to the EDGAR database for filings.

XBRL

Once the XBRL and HTML reports had been scraped parsing began on the XBRL reports to extract financial data programmatically. XBRL stands for extensible business reporting language and it is used to store company report information in XML by defining a schema to allow machines to parse the data. Each report must follow the US Generally accepting accounting principles schema or US-GAAP for short. The US-GAAP schema defines terms that can be used to refer to reported financial figures within the reports. For example if a company wished to report gross profit, they should specify it under US-GAAP GrossProfit, however this is not always the case unfortunately (59,60).

The scraper was intended to scrape various financial figures to derive ratios that would give investors insight to the business such as price / earnings and price / sales. For this the scraper needed to be able to access figures available in every financial report such as gross profit and revenues. Although every filing should employ the US-GAAP schema there were no rules to prevent companies also implementing their own schema on top of US-GAAP. This is typically done to define more specific tags to their financial figures. This made data collection impossible due to a large percentage of companies choosing to employ tags from their own schema to label figures in their report. These custom tags varied wildly from company to company. Ultimately the decision was made to pay for the Intrinio API as it offered an affordable student plan and it provided the data necessary for the platform on demand.

Pre Development Set up

PostgreSQL setup

Before full development of the web platform began, a PostgreSQL database was hosted on Amazon web services RDS platform. This database serves the web application and is connected to PgAdmin for backup and maintenance.

Redis Setup

Prior to development the Redis server was also set up. It runs on a Google Compute Engine virtual machine. The Redis server needed to be updated with a module called RedisJson in order to support JSON object storage.

Company information Script Development

The web application was initially intended to call the FinancialModellingPrep API for company information at run time. However, over the course of testing the API was unreliable and it went down for several hours. The unreliability coupled with the fact that the data provided is unlikely to change frequently drove development in the direction of building a script to collect it and store it in the database.

First a script called nasdaq_companies.py was created to download the list of companies from Nasdaq and the New York Stock exchanges. It is called nasdaq_companies.py as it was only supposed to collect Nasdaq stocks however a New York stock exchange data source was found and added to the script. The script downloads two CSV files, one is a list of companies on the Nasdaq exchange and the other is a list of companies on the New York Stock exchange. Once the CSVs were downloaded, they were parsed and merged into one large Pandas dataframe.

A second script was created called the company_info.py script. This script was responsible for gathering data from the FMP and Clearbit APIs. The company_info.py script calls the nasdaq_companies.py script and gets the list of all companies from the Nasdaq and New York Stock Exchanges. Once it gathers the list it loops through the list and requests information from the FMP API on each company. The FMP API returns business information such as the company's full name, the sector it operates in and its CEO. Among the information provided on each company is its website address which is then used to query the Clearbit API. The Clearbit API provides company logos based on their website domain names. Once company information and logos were gathered, they were all stored in the PostgreSQL database on AWS.

Financial API development

The Financial API was built with Django and hosted on a Google Compute Engine virtual machine. It is also ran inside of a Docker container with the necessary libraries and packages it needs. It was implemented to provide all the calculations outlined in the Financial research chapter of this report. This API was built in Django however only 1 file contains custom code. The explanation of the Django Project structure will be provided in the web application section as it makes more use of the Django resources.

Financial API operation

The Financial API provides seven API endpoints which can be accessed via a HTTP request. Depending on the endpoint it will accept different data and HTTP methods such as GET or POST. Each endpoint returns a JSON response with the content requested.

The Financial API is used to provide information to the users home page about the stocks in their portfolios.

Financial API endpoints

Simple Return

The SimpleReturn endpoint is responsible for providing returns and averaged annualised returns of a portfolio with multiple stocks based on their weights in the portfolio. For example, if a user's stock consisted of 40% Stock 1 and 60% Stock 2 it would calculate the return for each stock and the overall return of the portfolio. The endpoint accepts a HTTP GET request and returns JSON.

Log Return

The logReturn endpoint returns the annualised log returns for an individual stock. This end point is used to calculate and display the returns of each company in a user's portfolio. Log returns are used to ensure the returns follow a normal distribution.

Indices Return

The indicesReturn endpoint functions similarly to the previous 2 endpoints except it provides the annualised average return of market indices. These indices are used to represent the market as they comprise of multiple stocks. This end point is used to determine the performance of stocks in a user's portfolio against the market. Indices like the S&P 500 and the Dow Jones are used.

Volatility

The volatility endpoint returns the annualised and daily volatilities for all the stock tickers specified in the request. Volatility is a measure of risk and shows how unpredictable and extreme the average change in the stock's prices have been. This end point is used to display the risk of each stock in a user's portfolio.

Portfolio Analyser

The portfolio analyser performs a Monte Carlo simulation with 1000 iterations. Each iteration it creates a random weight for each stock in the received portfolio and calculates a Sharpe ratio and an overall volatility. It then returns the portfolio weighting for the portfolios with the highest Sharpe ratio and lowest volatility.

```
# for number of stocks * 1000 iterations
for x in range(1000):
    # generate random weights
    weights = np.random.random(len(portfolio))
    # ensure weights add up to 1
    weights /= np.sum(weights)
    # calculate the weighted annualised returns
    returns = np.sum(weights * log_returns.mean()) * 250
    # calculate the weighted annualised volatilities
    volatilities = np.sqrt(np.dot(weights.T, np.dot(log_returns.cov() * 250, weights)))
    pfolio_returns.append(returns)
    pfolio_volatilities.append(volatilities)
    # Calculate sharp with risk free weight equivalent to rate of 10 year US gov bond ie 2.9 %
    pfolio_sharpes.append((returns - 0.029) / volatilities)
    pfolio_weights.append(weights)
```

Code Snippet 2 : Monte Carlo simulation performed by the Financial API

Beta & Capm

The betaCapm endpoint returns the beta and capital asset model calculations for all the stocks specified in the request. The beta gives an indication of how the stock is performing with regards to the market and the capital asset model provides an estimation of the return the stock should be making. These endpoints are used to provide data for user portfolio.

```
covariance = returns.cov() * 250
tick_market_cov = covariance.iloc[0,1]
market_var = returns['^GSPC'].var() * 250
#beta is the covariance between the stock and the market divided by the variance of the market
beta  = tick_market_cov / market_var
#round the beta value
betas.append(round(beta,2))
# Capm
capms.append(0.025 + beta * 0.05)
```

Code Snippet 3 : Beta and CAPM calculations performed by the Financial API

Pricing

The getPricing endpoint is responsible for returning the historical prices of a stock. This endpoint will also fill in gaps in the data if needed. For example if a company started trading publicly 3 years ago and 5 years' worth of pricing is requested it will fill the 2 years before the company started trading with 0s. The reason this is done is to ensure the charts on the web page function properly.

Financial API utility functions

Many of the above endpoints make use of utility methods that serve an important role in the API. The operation of each will be outlined below.

Simple Return – sret

The Sret function is responsible for calculating the simple return of a stock. It divides every price in the pricing data by the price before and subtracts 1 at the end. It does this simply using the NumPy shift method instead of looping. It also calculates the annual returns by multiplying the mean of the returns by 250 which is the number of business days in a year.

```
# get daily return by dividing current day previous
returns = (ticker_prices / ticker_prices.shift(1)) - 1
# get annual return by multiplying the daily return by the number of business days in a year
annual_returns = returns.mean() * 250
```

Code Snippet 4 :Returns calculation done by the sret function

Log Return – lret

The lret function does the same return calculation as the sret function however it logs the result afterward.

```
# calculates log return, similar to simple but logged
def lret(ticker_prices, ret=False):

    # when the returns for a single stock is needed with no average
    if ret:
        returns = np.log(ticker_prices / ticker_prices.shift(1))

        return returns

    #otherwise average is returned
    ticker_prices['log_return'] = np.log(ticker_prices / ticker_prices.shift(1))
    avg_return = ticker_prices['log_return'].mean() * 250

    return avg_return
```

Code Snippet 5 : The log returns calculation done by the lret function

Volatility – volit

The volit function is responsible for calculating the volatility of each stock passed to the function. It does this by gathering the annual return from lret and calculating the standard deviation of the annual returns using the NumPy std method. The function provides both the daily and annual volatility measures and returns them.

```
# Calculates volatility
def volit(ticker_prices, tickers):

    annual_returns = lret(ticker_prices, True)
    daily_risk = []
    annual_risk = []
    for tick in tickers:
        # volatility is calculated by getting the standard deviation of the daily and annual returns
        daily_risk.append(round((annual_returns[tick].std()) * 100, 2))
        annual_risk.append(round((annual_returns[tick].std() * 250 ** 0.5) * 100, 2))

    daily_risk_dict = dict(zip(tickers, daily_risk))
    annual_risk_dict = dict(zip(tickers, annual_risk))

    return daily_risk_dict, annual_risk_dict
```

Code Snippet 6 : The volatility calculations performed by the volit function

Pricing data – getPrices

The getPrices function is responsible for getting historical pricing data for the stocks. The getPrices function uses pandas datareader as its data source to gather pricing. It can return all pricing information such as volume, opening price and close price or just the close price on its own if specified using the multi parameter. Every endpoint uses this function to gather the pricing information needed. Before data is requested from Pandas Datareader it checks the Redis cache to see if pricing information has been stored recently.

```
try:
    print('trying cache')
    pricing_json = json.loads(rconn.execute_command('JSON.GET', f'{ticker}Pricing'))
    ticker_prices = pd.read_json(pricing_json, typ='series')
    print(f'Success cache - {ticker}')


except Exception as e:

    print('Cache failed')
    ticker_prices = wb.DataReader(ticker, data_source='yahoo', start=sformat_start)
    tp_json = ticker_prices.to_json()

    # This will add the data to the cache and set an expiration on the data
    rconn.execute_command('JSON.SET', f'{ticker}Pricing', '.', json.dumps(tp_json))
    expire_time = calculate_time_delta()
    rconn.execute_command('EXPIRE', f'{ticker}Pricing', int(expire_time))
```

Code Snippet 7 : The data collection logic performed by the getPrices function

If there is no data present in the cache, then it will proceed to request data from Pandas Datareader. Once fresh data has been gathered it will be added to the cache with an expiry time calculated by the calculate_time_delta function.

Cache Expiration – calculate_time_delta

The calculate_time_delta function is responsible for calculating the amount of time in seconds between the current time and when the market closes. The data returned by Pandas Datareader provides the price of the stock from the end of the last trading day. The pricing data is valid until it is refreshed when market closes again. By storing the pricing of the stocks in the cache it will speed up response times as it does not need to use an external data source to gather data. The time until market close (10pm GMT) is calculated by the function and returned in seconds.

Date range – defineDates

The defineDates function is called to determine the date range the data should be collected in. By default, it gets the current date and the same date 5 years ago and returns it. It is used in getPrices to determine the start and end dates of the data collected from Pandas Datareader. The function accepts a parameter to choose how far back to go, for example 3 or 4 years instead of 5.

Docker Hosting

The financial API is hosted inside of a Docker container. This makes moving the codebase to remote servers like Googles Compute Engine virtual machines much easier. Instead of having to install Python and the necessary packages a single command can be issued to Docker to build a container to download all the necessary packages and languages needed. To do this a Dockerfile must be created, the Dockerfile is essentially a list of steps to be carried out when the Docker container is being built. This typically includes downloading updates, languages and packages. Below code snippet 8 shows the Dockerfile used for the financial API container.

```
From python:3.7

RUN apt-get update && \
      apt-get -y install sudo

RUN mkdir -p usr/src/app

COPY requirements.txt /usr/src/app

COPY . /usr/src/app

WORKDIR usr/src/app

RUN pip install --upgrade pip

RUN pip install --upgrade setuptools

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8000

CMD [ "python", "manage.py", "runserver",  "0.0.0.0:8000" ]
```

Code Snippet 8 : Dockerfile used to create a Docker container for the Financial API

This Dockerfile tells docker to first update the container and then copy all the code and necessary files to the working directory in the container. The requirements.txt file lists all the necessary Python packages for the API to run. It is used by the Python package installer called Pip to install all the necessary packages.

Once the packages have been installed with pip the Docker container will then expose port 8000 so that the API can be accessible via that port. The CMD command at the end is executed when the Docker container is ran. This will start the API.

Web Application Development

The web application is what ties all the data feeds together, processes them and presents the results to the end user. The Web application was developed in Django and is hosted on a Google Compute Engine virtual machine. The web application is also hosted in a Docker container.

Web Application Operation

The web application is responsible for serving web pages to the users. Here users can register and login, create portfolios and research stocks. The web application makes calls to the various APIs and data sources to gather data which it then processes and constructs a webpage with. All the Major components will be outlined in the following sections.

Django Structure

Before proceeding any further it is important to note the structure of a Django project. A Django web application consists of a few layers. The main directory of a Django application is called the project. The project folder consists of multiple subfolders, these are called apps (templates and static are the exception). These are used to separate the code for different parts of the project. In this project folder is the manage.py file. This file is responsible for operations such as running the project. For example to start the website the "python manage.py runserver" command must be issued. The Dockerfile and requirements.txt files are used to build and run the Docker container when the website is deployed. The following sections will explain the structure by examining the database setup and apps.

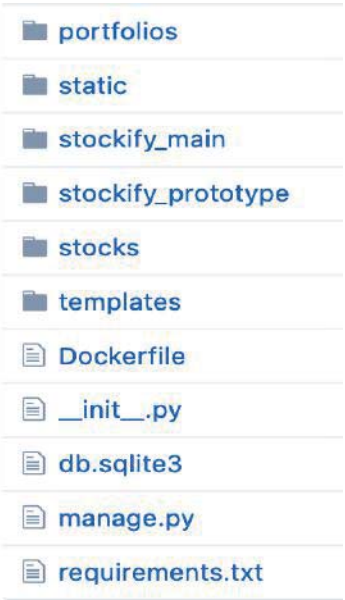

Figure 29 : Django project structure

Django Database

The web application is connected to the AWS database that was set up before development began. This is the database that houses all of the stock and user information. The database connection is created when the Django server is ran. All the database credentials and connection details are specified in the servers settings.py file. Settings.py can be found in the stockify_prototype app.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'stproto',
        'HOST': 'stockify-prototype.csmyllyw1yns.eu-west-1.rds.amazonaws.com',
        'PORT': '5432',
        'USER': 'sharkey',
        'PASSWORD': '34ruanbeg34'
    }
}
```

Code Snippet 9 : Database connection used on the Django web application

Once Django is connected to the database, Django will take care of creating database tables. Database tables are defined in the models.py files in each app. To create a database table a Python class is used to represent each table like the class in code snippet 10.

```
# Create your models here.
class StockBase(models.Model):
    ticker = models.CharField(max_length=7, primary_key= True)
    companyName = models.CharField(max_length=200)
    ceo = models.CharField(max_length=100)
    marketCap = models.BigIntegerField()
    description = models.CharField(max_length=1000)
    exchange = models.CharField(max_length=100)
    industry = models.CharField(max_length=500)
    sector = models.CharField(max_length=500)
    website = models.CharField(max_length=255)
    logo = models.CharField(max_length=200, default=None)
```

Code Snippet 10 : The StockBase database table defined in Django

When models are defined they are added to the database by running "Python manage.py migrate". This command tells Django to convert the classes in the models.py to database tables on the database.

Once the database and its tables are established, the database is interacted with through objects. The Django object relational mapper allows developers to perform database operations through an object rather than making SQL queries. Below in code snippet 11 is an example of a user being created using the object mapper.

```
User.objects.create_user(username=uname, password=password, email=None)
```

Code Snippet 11 : User creation via the Django ORM

Django Apps

This section of the report will be breaking down each of the apps in the web application. Each is responsible for a different set of tasks. This was done to separate concerns and make code more maintainable.

Stockify_prototype App

This Django project was started in October 2018 and was intended to be a prototype for the project before moving to a JavaScript frontend. The author felt that a lot of work was done to construct the prototype and that it would be completely lost if the project moved to JavaScript, so the prototype codebase was used. This is why the project is called Stockify Prototype as Django does not provide a way to rename projects.

As the overall project is called Stockify prototype a Django app is auto-generated when the project is created with the same name. This app is responsible for storing configuration information for the project. All the files are auto-generated by Django.

The stockify_prototype app hosts the settings.py file. Settings.py is what stores all the configuration for the project. Currently it stores the database connection details, time zone and the location of HTML templates used in the templates folder.

The app also hosts the main urls.py file. Urls.py is used to create URL routes for the website. For example if the user wishes to login they would visit www.stockify.ie/login. The base routes for the project are added here and can be expanded on by the urls.py files in the other apps.

Stockify_main App

The stockify main app is responsible for providing the main pages of the application. This includes the index, about, signup and user home pages.

Admin.py & Apps.py

The stockify_main app has a few more files as it is actively used to run the website. Admin.py and apps.py are auto-generated and not used by the developer. Admin.py is used to create database tables for admins which is not used in this project. Apps.py is used to register this app with the configurations in stockify_prototype/settings.py file. This lets the project track the apps that are in use.

FinancialApiWrapper.py

The financialApiWrapper.py file contains code that is used to request data from the financial API. As this app is responsible for the home page which provides portfolio information it gathers data from the financial API. The financialApiWrapper is used extensively in views.py which is where the majority of the apps code resides. This file provides a class that has a methods that return information for each endpoint of the Financial API. It uses the requests package to make calls to retrieve and gather data. This makes the views.py code neater and allows for focus on data processing.

Models.py

Models.py is used to create database tables however this app does not make use of a custom database table as Django automatically created a user's table on the database with the use of the authentication library.

Tests.py

Tests.py is what holds the tests for this specific application. This will be covered in more depth in the testing section of the report.

Urls.py

Urls.py is similar to the file in stockify_prototype. It provides URL routes for the app. The URL's in this urls.py extend the URL's specified in the stockify_prototype/urls.py file. If a user wishes to signup they would visit www.stockify.com/signup and the signup view in views.py will provide the webpage and handle the logic behind signing up.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('home/', views.home, name = 'home'),
    path('signup/', views.signup, name='signup'),
    path('about/', views.about, name='about')

]
```

Code Snippet 12 : URL routes in use by the stockify_main app

Views.py

Views.py is where the majority of the code is executed. It consists of multiple functions or views. When a user makes a request to the website or visits a page their request is forwarded to one of these views using the URL in the request. For example if a user visited the signup page their request will be sent to the signup view in views.py to be processed. These views take HTTP requests, process them, gather information and return the correct webpage and data. The operation of each view will be outlined below.

Index

Index is responsible for returning the index page of the website when a user visits the base URL of the website. The index page is provided when someone visits www.stockify.sytes.net. The index page provides links to the login, signup and about pages. The index HTML page is stored in the templates folder and is rendered when someone visits the index page.

Figure 30 : The final implementation of the index page

The index page is a simple webpage written in HTML and CSS. The text on the page is animated and bounces onto the screen to give it some flare this is done using cssanimate on the page elements.

About

The about view is responsible for rendering the about page which provides information on Stockify. It showcases what the platform can do and how it can help investors. The about page is rendered when a user visits www.stockify.sytes.net/about.

Figure 31 : The final implementation of the about page

Signup

Signup is responsible for handling user registration and rendering the signup page. When a user visits www.stockify.sytes.net/signup they will be brought to this page. On this page the user can create a username and specify a password.

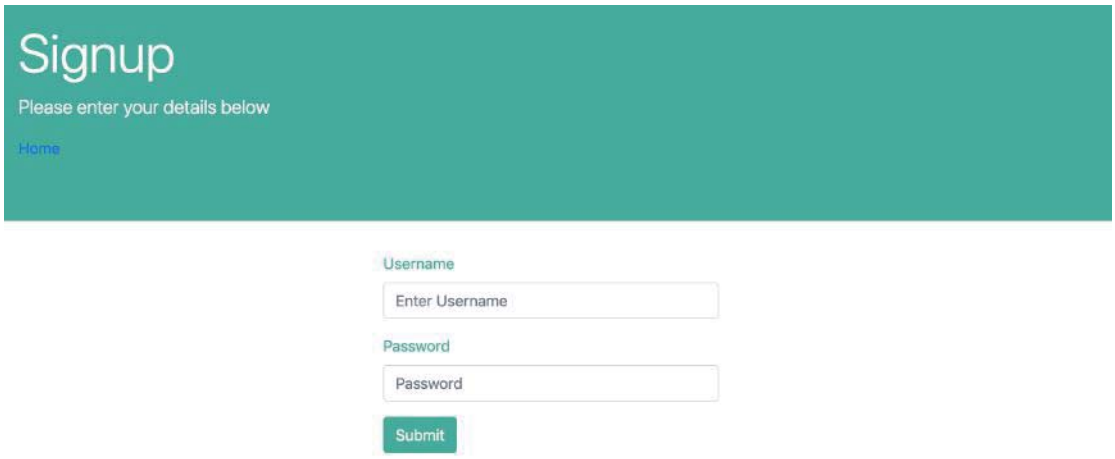

Figure 32 : The final implementation of the signup page

Session variables are used to relay messages to the user. For example if a username is already in use a message in red will appear on the page informing the user of the issue. This feature is important as it helps inform users with minimal IT experience to understand why the system is rejecting their request.

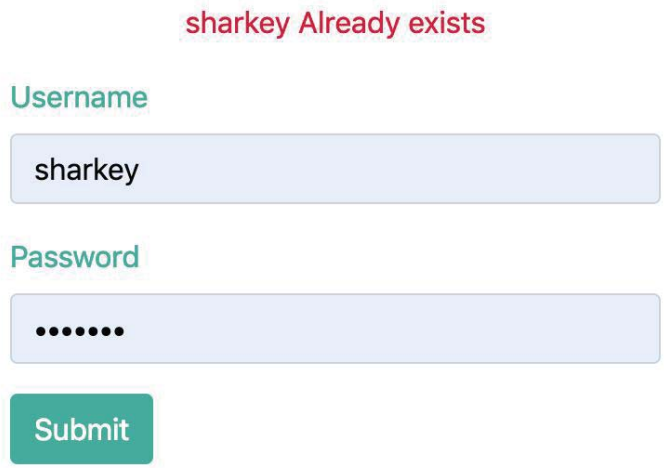

Figure 33 : An example of a session message error

When a user submits their account information the view will check if the information is in a correct format before adding the user to the system. If the users request was successful they will be redirected to the login in page. A success message will added to the session variables and be displayed in green on the login page to let the user know they were successful.

Login & Logout

The views.py file is not responsible for user log in and log out. Django provides an authentication library to handle user logic. It is responsible for creating a user's database table and maintaining it. This library also takes care of the user login and logout by providing and removing authentication tokens for the users. The login and logout HTML pages send the users information to the library and the library handles the logic. The users information such as passwords are hashed and stored securely on the database.

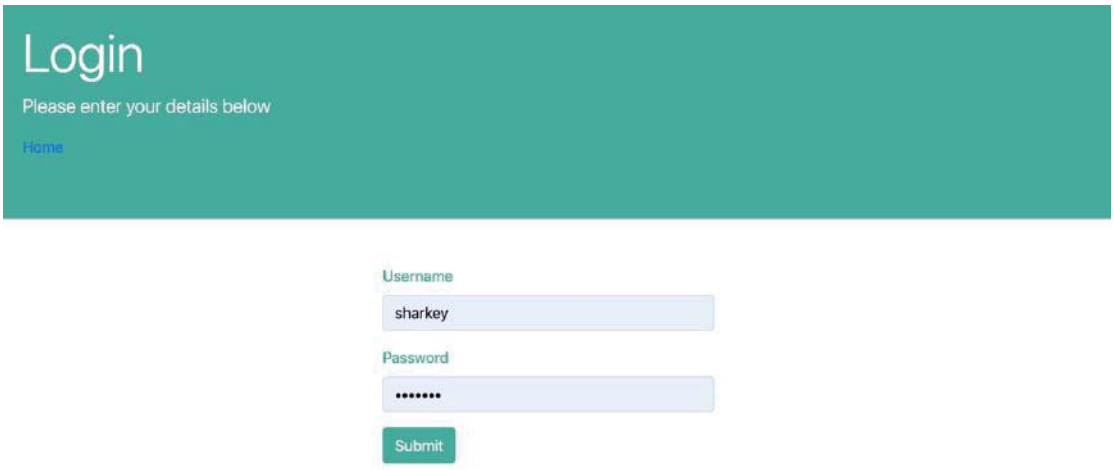

Figure 34 : Final implementation of the login page

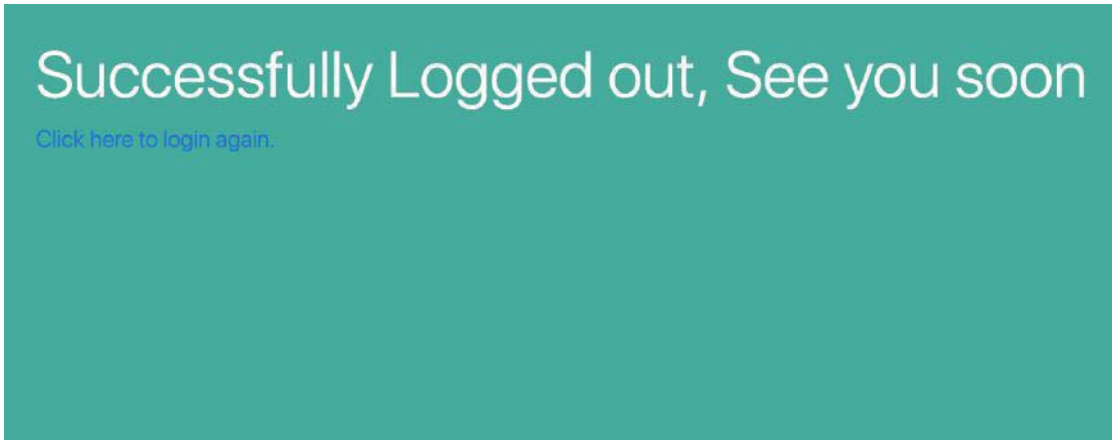

Figure 35 : The final implementation of the log out page

Home page

The home page is responsible for displaying information on the users portfolio and allows them to search for stocks. When a user logs in they are greeted by their name and a search bar to lookup stocks on the platform.

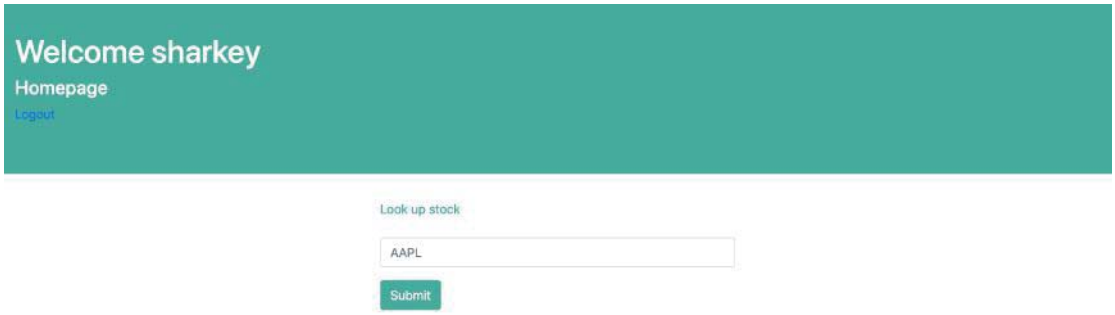

Figure 36 : Top section of the user's home page

The search bar is a HTML form that triggers a JavaScript function when clicked. When the user enters a value such as AAPL in the search and hits enter the function will take the value and go to the stock page for that stock.

```
// This is the function for searching the database for the ticker entered in the search bar
function searchStock(){
    ticker = document.getElementById('search_stock').value
    console.log(ticker)
    window.open('/stocks/' + ticker,'_self')
}
```

Code Snippet 13 : JavaScript search function

The home page is also protected by CSRF tokens. When a user is logged in they are provided a CSRF token by which to authenticate themselves. There is a check in the HTML code using Jinja that examines the session variables to see if the user possesses a token. If the user is not logged in attempting to access the home page will return a message asking them to login and providing a link to the login page.

```
{% if auth_check %}
    <div class="jumbotron" id="jumbo1" style="background-color: #41b3a3;">
        <div class="col-mid-3">
            <h1 id="header1" style="color: white;">Welcome {{username}}</h1>
            <h4 style="color: white;">Homepage </h4>
            <a href="/accounts/logout">Logout</a>
        </div>
    </div>
```

Code Snippet 14 : Jinja authentication check on the web page

The home page is also used to display information about the current users portfolio of stocks. The user must have at least 2 stocks added to their portfolio before information is displayed on the home page. If the user does not have a portfolio the portfolio section of the home page will provide a message telling the user to add stocks to their portfolio.

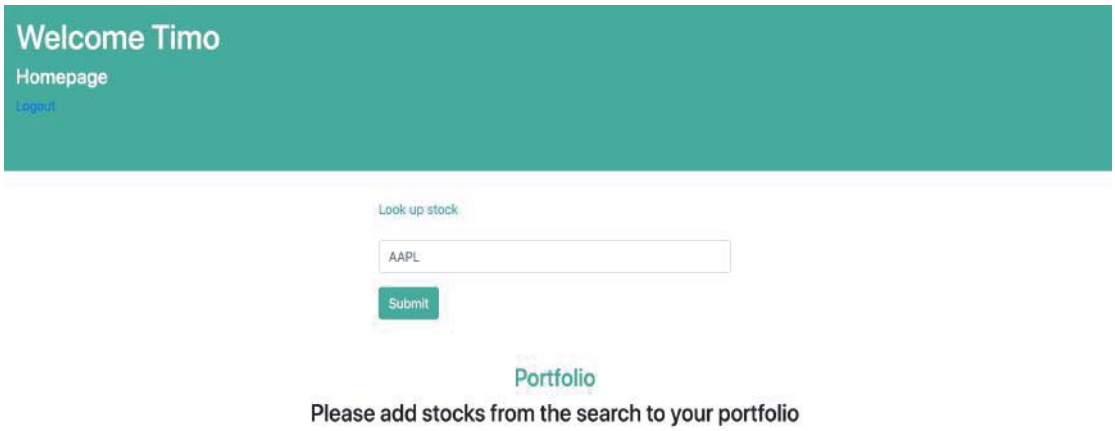

Figure 37 : User without a portfolio's home page

If the user does have a portfolio, a wide variety of information will be provided. The portfolio section will provide the tickers in the users portfolio on the left side of the page. It will provide the ticker for each company, the logo and its last closing price. The closing price can be either green or red. The price will be green if the stock's closing price was higher than the opening price for that day. The price will be red if the stock's closing price is less than the opening price for the day.

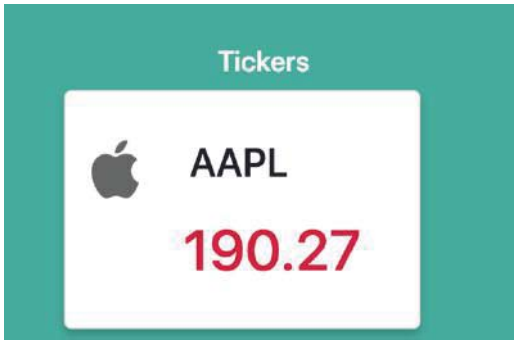

Figure 38 : Portfolio ticker on a user's home page

The stock information such as the name and logo are gathered from the database. To access the stock information from the database the Django ORM is used. When the stock information object is gathered it is put through what's called a Serializer. The serializer essentially converts the stock information object into a JSON string so that it can be more easily parsed on the webpage.

```
#Serialze it ie convert to JSON
serializer = StockBaseSerializer(stock)

# Add to the temp_json structure
temp_json['stock'] = serializer.data
```

Code Snippet 15 : Example of a Stock object being serialized

The price for each ticker is gathered from Intrinio. To access Intrinio the ApiWrapper is called. This is a class created in the stocks app of the project. It is responsible for making the calls to all the external data sources like Intrinio and Twitter. It will be explained in greater detail in the stocks app section.

```
#Instantiate an ApiWrapper class for the current stock
api = ApiWrapper(today_str,today,ticker)

#get the stock pricing
price_info, error = api.getPrice()

temp_json['pricing'] = price_info['data'][0]
```

Code Snippet 16 : ApiWrapper class being used to gather stock prices

This process is done for all stocks in the portfolio. Once the data has been collected it is sent to the webpage to be rendered. In order to generate a dynamic portfolio page for each user the HTML template must be inflated with the data for that users portfolio. Jinja allows for dynamic content by dynamically inflating the HTML with data. The data is sent to the web page template where Jinja can loop through the data and add it to the HTML.

The portfolio section also provides pricing information. Pricing information for each stock is gathered by the view from the getPrices endpoint of the financial API. The reason why getPrices is used over Intrinio is due to some stocks not displaying on the chart properly if data is missing. By using getPrices it takes care of missing data. The financial API is accessed through the financialApiWrapper mentioned before. The wrapper essentially calls the financial API and passes the returned data to the view.

```
pricing = finapi.getPrices(portfolio_tickers)
```

Code Snippet 17 : Collecting stock prices via the FinancialApiWrapper

Once the pricing data is gathered it is sent to the webpage where a JavaScript script takes the data and creates a chart using ChartJS. There is a div in the HTML that acts as an anchor so that when the data is passed to the webpage a new chart is created and inserted into the anchor div on the page. This allows the chart to be reloaded if the browser window size changes. Below is the code used to create the pricing graph.

```
new Chart(ctx1, {
    type: 'line',
    data: data_obj_line,
    options: {
        title: {
          display: true,
          text: 'Closing Prices last 3 years'
        },
        scales: {
            yAxes: [{
              gridLines:{
                  display: false
              },
              ticks:{
                  beginAtZero:false,
                  callback: function(value, index, values) {
                        return '$' + value;
                  }

              }
            }],
            xAxes:[{
                gridLines:{
                    display: false
              },
              ticks:{
                  beginAtZero:false
              }
            }]

        }
  }

});
```

Code Snippet 18 : ChartJS pricing graph JSON configuration

Figure 39 below is the result of creating the graph above. It shows the pricing of Apple and Tesla stock going back to 2014.

Figure 39 : The pricing chart generated for Apple and Tesla

As well as pricing, the page also provides numerous other graphs and charts. The portfolio section provides two donut charts displaying the two portfolios returned from the portfolio analyser endpoint of the financial API. After the pricing is gathered the FinancialApiWrapper is used again to call the portfolio analyser. The API responds with two sets of portfolio weightings. The first set is the Sharpe portfolio, it records the portfolio which returned the best Sharpe ratio during the Monte Carlo simulation. The second set is the minimum volatility portfolio, it records the portfolio which returns the lowest overall volatility or risk. This data is passed to the JavaScript script on the webpage and two donut charts are generated, one for each portfolio. In the images below you can see an example of the two portfolios and weightings.

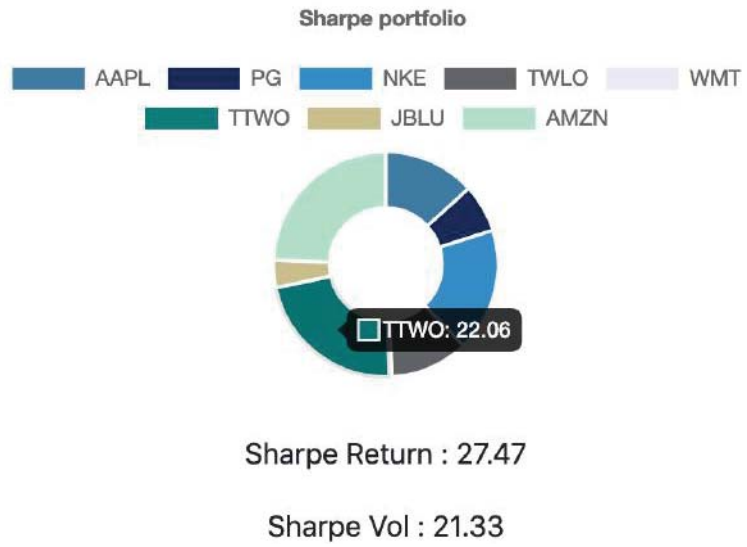

Figure 40 : Example of a Sharpe portfolio donut chart

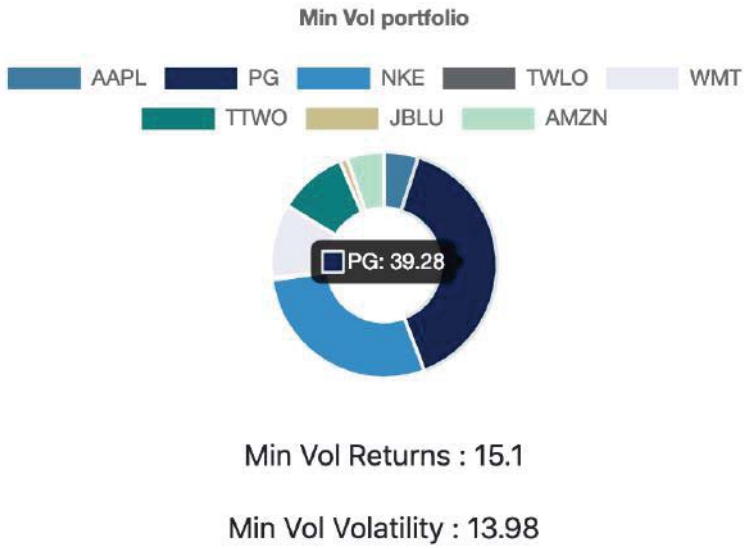

Figure 41 : Example of a minimum volatility portfolio donut chart

The home page also provides three separate bar charts that report on different metrics. The first bar chart displays the annual average return of the stocks in the portfolio. It also includes some of the market indices as well to provide a way to gauge the performance of stocks in the portfolio against the market. The returns are gathered from the financial API via the FinancialApiWrapper and are used to create a bar chart by the JavaScript script in the webpage.

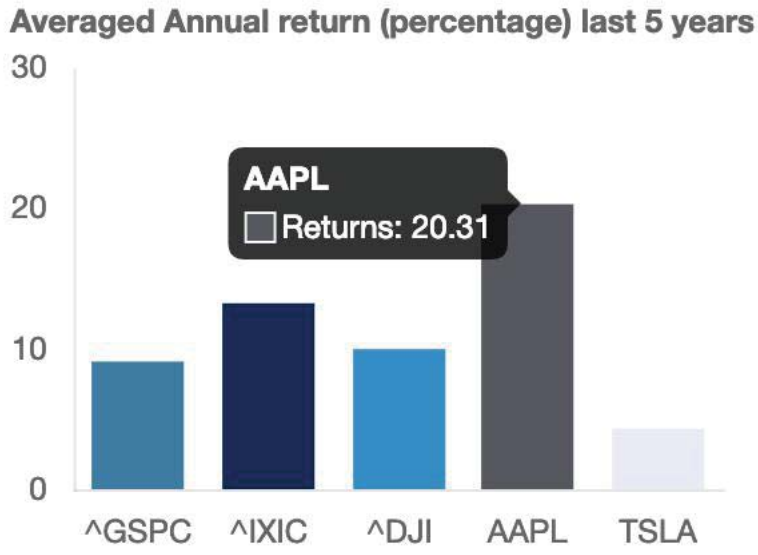

Figure 42 : Example of the annual average returns bar chart

The second bar chart displays the average annual volatility for each of the stocks in the portfolio. This shows the user which of their stocks is the most and least volatile. The volatility information is gathered via the FinancialApiWrapper also and is used to create a ChartJS bar chart.

Figure 43 : Average annual volatility bar chart

The third bar chart displays the average annual Beta for each of the stocks. This is used to see how aggressive or defensive a stock is in relation to the market. This data is gathered from the FinancialApiWrapper also and uses the BetaCapm endpoint of the financial API. The data is then used to produce the bar chart below on the webpage.

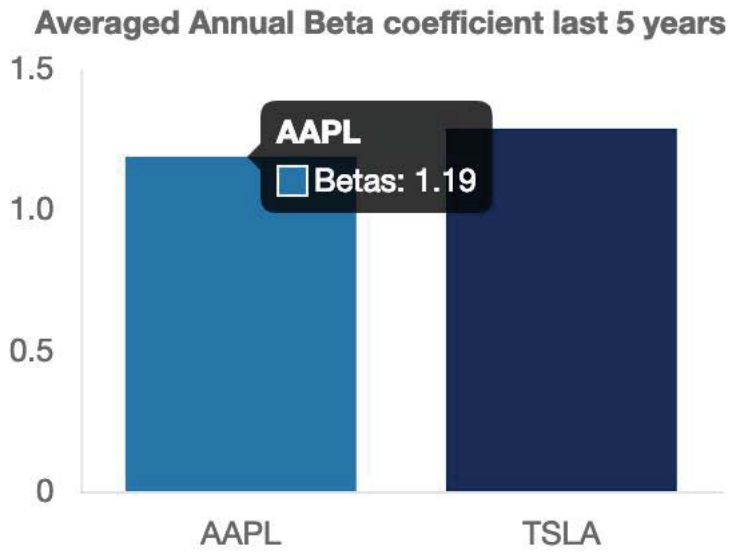

Figure 44 : Example of the beta bar chart

Stocks

The stocks app is responsible for providing the stock page and all of the information associated with each stock. The company_information and nasdaq_companies scripts reside here as they access the database model specified in the models.py file.

Templatetags

The templatetags folder houses the stockify_tags.py file. This file is used by Django to create HTML filters. The code inside of stockify_tags provides the Jinja templating engine the ability to turn values into percentages or to round numbers to 2 decimal places. Certain data returned by the Intrinio API such as the financial ratios are provided as decimals that need to be converted to a percentage. Often times some of the figures in the data are large decimals that need to be rounded also. This simply allows Jinja to round or turn numbers into percentages as it is adding them to the HTML page. The image below shows how the filters are implemented in the HTML code using Jinja syntax.

```
<strong><h5>Div. Yield</h5></strong>
<center><h6>{{calculations.dividendyield | percentize}} % </h6></center>
<strong><h5>Quick Ratio</h5></strong>
<center><h6>{{calculations.quickratio | rounder}}</h6></center>
```

Code Snippet 19 : Example of the template tags in use

Apiwrapper.py

The Apiwrapper file is designed very similarly to the FinancialApiWrapper. It is responsible for gathering data from all the external data sources using the requests library. It is used extensively by the views.py file to gather data for the stocks page. It stores all the links to the external API endpoints and has a function to call each API for the necessary data. Below is two examples of gathering filings and price data from Intrinio.

```
# get financial filings for the company
def getFilings(self):

    #https://api.intrinio.com/companies/filings?identifier=AAPL&report_type=10-K&report_type=10-Q&api_key=OjNkYjdlMTY5Nj
    link = self.intrinio_links['filingsBase'] + f'?identifier={self.ticker}' + '&report_type=10-K' + self.intrinio_key


    filings_data = requests.get(link)

    if filings_data.status_code == 200:

        filings = filings_data.json()
        filings_list = filings['data']
        length = len(filings_list)
        # return 10 or less filings
        if length < 10:
            return filings_list[:length], None

        return filings_list[:8], None

    else:

        return None, {'ERROR':'Filings call failed'}
```

Code Snippet 20 : getFilings function code for collecting links

```
# Get the current price of the company
def getPrice(self):

    link = self.intrinio_links['priceBase'] + f'?identifier={self.ticker}' + self.intrinio_key
    pricing_data = requests.get(link)

    if pricing_data.status_code == 200:

        pricing = pricing_data.json()
        pricing_list = pricing['data']



        return pricing, None

        # return pricing_list[0], None

    else:
```

Code Snippet 21 : getPrice function code to gather pricing data for a stock

Rssreader.py

The rssreader.py file is responsible for gathering data from the Seeking Alpha RSS feed. It is used in views.py to gather the latest news from Seeking Alpha about a stock to display to users. It gathers data from the RSS feed and parses the data using the feedparser Python package. Once article links are parsed from the feed the links are then reconstructed as the article URLs provided by the RSS feed are missing an id field which needs to be added.

```
# requests and parses the feed related to stock passed in
def parseFeed(self):

    link = self.reader_links['seekingAlpha'] + self.ticker + '.xml'
    feed = feedparser.parse(link)

    article_list = feed.entries

    # limit the amount of articles returned to the view
    if len(article_list) < 10:

        length = len(article_list)

    else:
        length = 13

    # call link reconstruct on the article list to fix news URLs
    article_list = self.linkReconstruct(article_list)

    return article_list[:length], article_list
```

Code Snippet 22 : Code used to parse links from the Seeking Alpha RSS feed

Serializer.py

Serializer.py is used by the stocks app and the stockify_main app to translate database objects into JSON. This makes it easier to parse on the webpage with Jinja. It uses a class to define the fields which are to be converted to JSON.

```
#Serializer that converts stock models to JSON
class StockBaseSerializer(serializers.ModelSerializer):
    class Meta:
        model = StockBase
        fields = ('ticker', 'companyName', 'ceo',
                  'marketCap', 'description',
                  'exchange', 'industry',
                  'sector', 'website', 'logo')
```

Code Snippet 23 : The serializer model used to convert Stock objects to JSON

Models.py

Models.py is used to create tables in the database. To create a table a class must be specified in a models.py file. Each app has a models.py file however not all of them use it as they do not need to add tables to the database. The stocksbase table in the database was defined here and Django added it to the database. It lays out all the information stored about each company.

Views.py

The views.py file in the stocks app is responsible for returning the stocks page. If a user was to search a stock in the search bar or visit www.stockify.sytes.net/stocks/nvda they would be provided the stocks page for Nvidia. Views only has one view function which is "getStock" and it is entirely responsible for data wrangling, data processing and web page rendering.

Stocks Page - Top Section

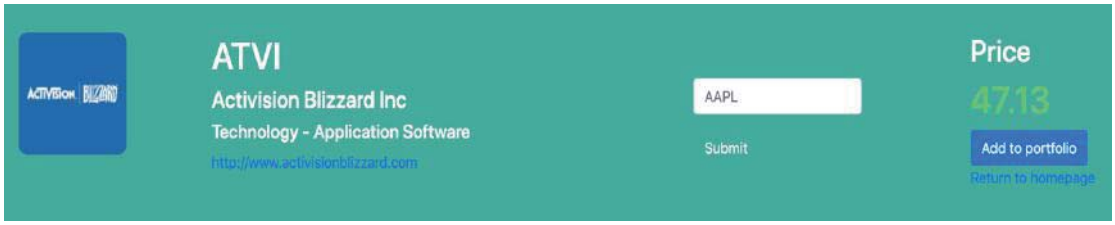

Figure 45 : Stocks page top section (Company not currently in portfolio)

Figure 46 : Stocks page top section (Company already in portfolio)

The first part of the stocks page to be rendered is the top section. Company information is gathered from the database and used to display information about the company such as the logo, name, sector and website. Its current price is also displayed. The price is coloured either green or red depending on the closing price of the stock in the same manner as the prices provided on the home page. The user also has the option to add a stock to their portfolio, if they add the stock a success message will be presented to the user letting them know the operation was successful.

The portfolio table is defined in the portfolios app which will be covered in the next section however it is used in this app when adding to the portfolio. The view gathers the users portfolio via the ORM object and checks it. First it checks if the user has a portfolio to see whether or not it should display the “add to portfolio” button. If the stock is not in the users portfolio the “add to portfolio button” will be added to the page. When this button is pressed a request to add the stock is sent to the views.py in the portfolio app. This is done in the HTML code using Jinja as seen in the image below.

```
{% if portfolio_add %}
    <form action="{% url 'portfolios:add' ticker %}" method="post">
        {% csrf_token %}

        <button type="submit"  class="btn btn-primary">Add to portfolio</button>
    </form>


{% endif %}
```

Code Snippet 24 : HTML form responsible for adding a stock to a portfolio

Stocks Page - Pricing information

The next part of the page to be rendered is the pricing information section. It provides pricing information for the stock as of the last market close on the left hand side. It also provides a chart of the stock price going back 5-6 months. This data is gathered through the ApiWrapper which calls the Intrinio API for the pricing data. Once the data is gathered it is passed to the webpage to be displayed and a chart is generated using the data as seen in figure 47.

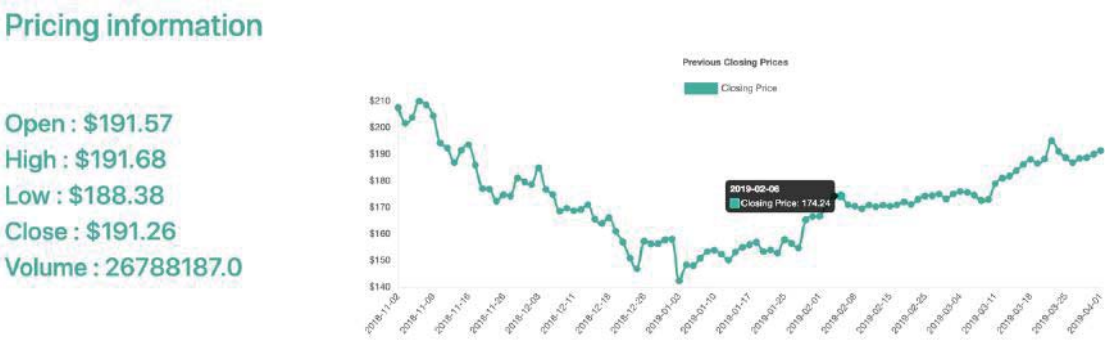

Figure 47 : Pricing information and chart generated for the stocks page

Stocks Page - Sentiment Analysis

The next section of the page contains the sentiment data provided from the Sentiment API. The Sentiment API will be covered in more detail in the machine learning pipeline section of development.

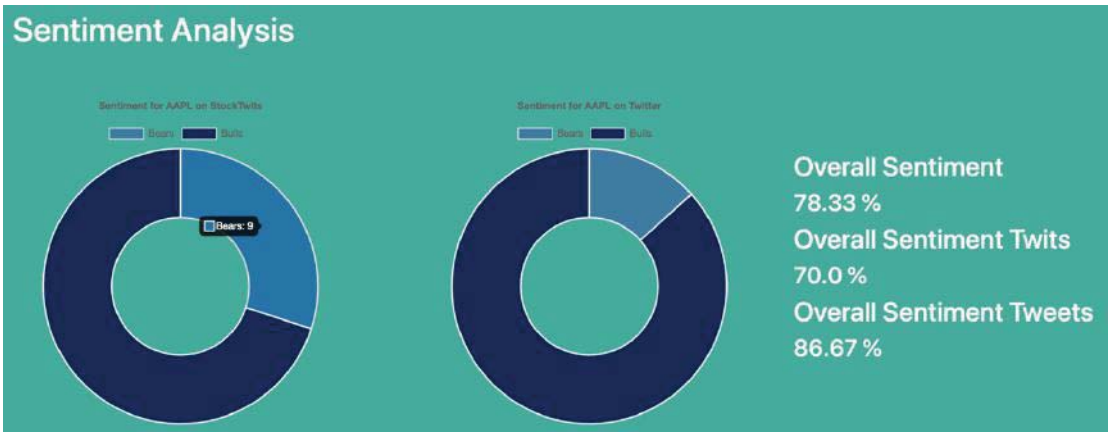

Figure 48 : Sentiment analysis donut charts generated on the stock page

The Sentiment API simply returns the number of bullish and bearish posts it predicts in the last 30 StockTwits and Twitter posts that reference the stock in question. The data returned from the Sentiment API is cached for 15 minutes. The reason for this is when StockTwits and Twitter data are collected they are cached for 15 minutes also. This is to prevent exceeding the API limits. If there are multiple calls for the same stock in under 15 minutes the Sentiment API will be making the predictions on the same data wasting resources and time. It made sense that once a prediction is made for a stock it will be cached for the same amount of time as the data. When the cache expires fresh data and predictions are collected and generated. This is all done on the Sentiment API so the view code simply collects the data from the Sentiment API.

The predictions are gathered using the ApiWrapper and an overall percentage is calculated. The data is then sent to the web page where two donut charts are generated showing the predictions made by the Sentiment API for the latest StockTwits and Twitter posts.

Stocks Page - Financial Information

The financial information section of the page hosts the financial information of the stock. It contains three panes.

Figure 49 : Financial Information panes on the stocks page

The first pane provides company information such as the CEO, website, the exchange it is traded on and a short description of the business. This data was all collected by the company information script. It is gathered by the view using the a database object and is sent to the webpage to be displayed.

The second pane contains financial calculations, these are financial ratios and percentages used to provide insight on the financial health of a company. It provides information on dividend yield, debt coverage and profit. These ratios are used in fundamental analysis to determine the viability of an investment. This data is gathered from Intrinio through the ApiWrapper in the view.

The third pane contains links to the company's financial filings on the SEC website. The filings are mandated yearly and provide an in depth examination of the business that covers multiple aspects such as market risk, financial breakdowns and risk factors. These links are gathered from the Intrinio API through the ApiWrapper in the view.

Stocks Page - Social Feeds

The final section provided by the stocks page is the social feed section. This section follows the same format as the financial information as it includes three panes, each responsible for a separate data source.

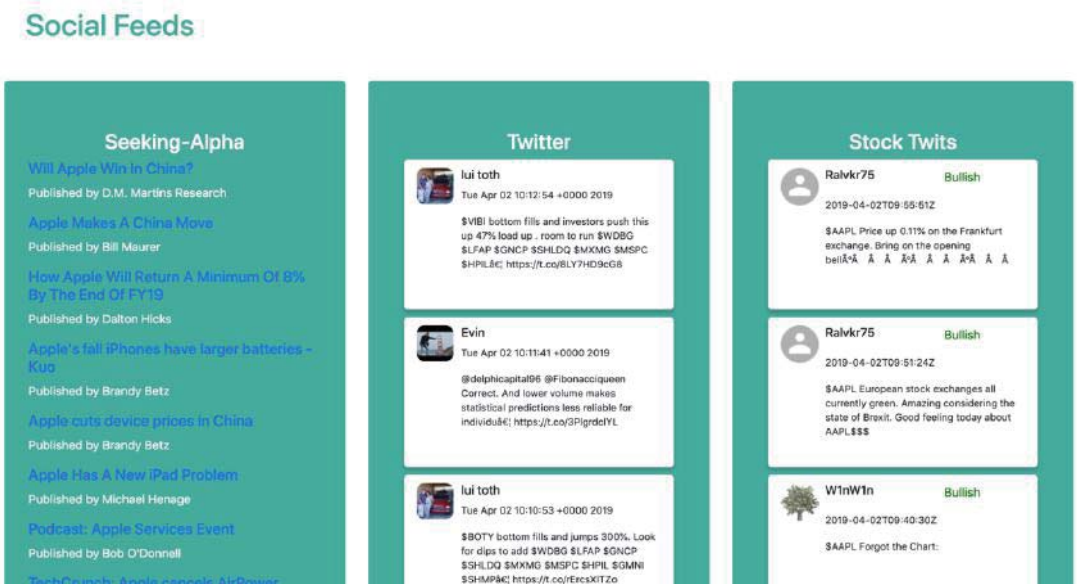

Figure 50 : The social panes on the stocks page

The first pane provides seeking alpha article links. The URL's for articles written on Seeking Alpha that mention the stock are provided here for users to read. These URL's are gathered from the Seeking Alpha RSS feed using the rssreader.py file mentioned in the previous sections. The view calls the rssreader code to gather and reconstruct URL's before passing the data to the webpage to be displayed.

The second and third panes provide a Twitter and StockTwits feed. Thirty of the most recent posts that mention the stock will be collected and presented here. Each post is presented in a CSS card that shows information on the user such as their image and name along with the time that they posted. The StockTwits card supports the option to have an opinion. If a user indicated they were bullish or bearish in their post it will be presented here.

Portfolios

Portfolios is the last app that will be discussed for the web application. It is simply responsible for creating the portfolios table in the database and handling additions to the portfolio. Only two files were modified in this app, these were the models.py and views.py files.

Models.py

The models.py file in portfolios is used to create and define the portfolios database table. It consists of a user field that acts as a primary / foreign key and a portfolio field that is a PostgreSQL list. Django created this table on the database when the project was started.

```
# Create your models here.
class Portfolio(models.Model):
    user = models.CharField(max_length=50, primary_key=True)
    portfolio = ArrayField(models.CharField(max_length=10))
```

Code Snippet 25 : Python class used to represent the portfolio table in the database

Views.py

The views.py file in portfolios is responsible for adding stocks to a user's portfolio. It consists of one view function called "add". When the add stock to portfolio button is clicked on the stocks page a request is sent to this function. This function will first check if the user has a portfolio. If the user does have a portfolio it will check the portfolio in case the stock is already in the portfolio, otherwise it will add the stock to the portfolio. If the user does not yet have a portfolio, one will be created and the stock will be placed in it. The view also constructs session messages to indicate whether or not the operation succeeded or failed. This session message is passed back to the stock page where it will be displayed.

```
# check to see if the stock exists in the portfolio already
if ticker in ticker_list:
    request.session['portfolio_message'] = {'error' : 'Stock already added'}
    request.session.modified = True
    return redirect(f'/stocks/{ticker}')
```

Code Snippet 26 : Session message construction in the portfolio addition view

Docker hosting

The web application uses Docker in a similar manner to the Financial API. A Dockerfile was created to pull in the code and the correct packages to the container before running it. The Dockerfile used for the web application is identical to the financial API Dockerfile. The container is mapped to port 80 when ran so that browsers can access the site using a URL in the browser.

```
From python:3.7

RUN apt-get update && \
      apt-get -y install sudo

RUN mkdir -p usr/src/app

COPY requirements.txt /usr/src/app

COPY . /usr/src/app

WORKDIR usr/src/app

RUN pip install --upgrade pip

RUN pip install --upgrade setuptools

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8000

CMD [ "python", "manage.py", "runserver",  "0.0.0.0:8000" ]
```

Code Snippet 27 : Dockerfile used to host the web application

Machine Learning Pipeline Development

Pipeline API

Work began on the Machine learning pipeline in late January 2019. It was decided that a machine learning pipeline would be constructed to autonomously train a machine learning algorithm. This was done as it was hoped that by gathering a large amount of data gradually over time the machine learning algorithm would be exposed to a larger dataset of posts and would perform better with more data.

The Pipeline API is developed in Flask, this was done so that an API could be built as quickly as possible and extra effort could be afforded to the other aspects such as data gathering, cleaning and algorithm training. Unlike Django, Flask does not auto-generate folders and code so all of the code is written by the Author.

Each endpoint when called spawns a process, this allows the API to send a response back the caller straight away. If the API was not to respond straight away the request would timeout and the code would fail to finish. By using a process the code can be executed and not be affected by the request timing out.

The Pipeline API consists of two endpoints. The first is the StockTwits data aggregation endpoint called twit_agg. The second endpoint is the algorithm trainer called the ml_trainer. Each of the end points will be discussed in detail in the following sections.

Pre development Setup

MongoDB

Prior to beginning development a MongoDB instance was setup on a Google Cloud Compute Engine virtual machine. It is used to store all the aggregated StockTwits data.

Redis

Prior to development some additions were made to the Redis cache. A job queue was set up so that virtual machines in the data collection cluster could be deleted when they finish collecting data. Using the Rq package for Python a worker was started on the Redis server that monitors the job queue. When the virtual machines are finished they pass a Python function to the queue that deletes the virtual machine. The worker executes these jobs and deletes the virtual machine. The deletion code must be run remotely as the virtual machines cannot delete themselves. Making use of the pre-existing Redis cache made the most sense so this solution was implemented.

Google Cloud Scheduler

To automate the machine learning pipeline a Google Cloud Scheduler job was created. The cloud scheduler is a service provided by Google that simply sends a HTTP request to the pipeline API to start the data collection job. The time is specified in a cron format that looks like “0 2 * * *”, this will make the scheduler call the pipeline API at 2am. This time was chosen as it is after the market closes.

Twit_agg - Data Aggregation

Twit_agg operation

The twit_agg endpoint is solely responsible for data collection. To do this it creates a cluster of Google Compute Engine virtual machines to collect the data through the Google API Client package. The virtual machines in the cluster download a Docker container and run it. The Docker container contains a simple script to collect the StockTwits data and store it in the MongoDB instance. Each node in the cluster will queue a self-destruction job when it has finished collecting data. The job is queued on the Redis cache using the Rq Python package. The last node to finish collecting data will be responsible for sending a request to the ml_trainer endpoint to start the algorithm training.

Twit_agg implementation

GcpSpawner

When a request is made to the twit_agg endpoint a process is spawned to start the data collection. The process makes use of a gcpSpawner class called from gcpspawner.py, this class is responsible for starting and configuring the virtual machines. When the gcpSpawner class is instantiated a set of parameters are passed to it. These values are used to configure the nodes in the cluster. The parameters include the connection credentials for the MongoDB database and the Redis cache. A list of regions were the virtual machines can be hosted and the number of virtual machines to create are also included.

Once the class is instantiated with the parameters it loops through creating the number of virtual machines specified. As virtual machines are being created, if the region limit is reached the region is switched to one of the other regions in the parameter list and the script continues. To create a virtual machine a JSON object is constructed and sent in a request to google. Below is an example of the virtual machine configuration used.

```
def create_instance(self,compute,counter):
    print('Creating Instance')

    instance_config = {
        'name' : self.base_name + str(counter) ,
        'machineType' : 'zones/' + self.zone +  '/machineTypes/' + self.machine_type,
        'disks' : [
            {
                'boot' : True,
                'autoDelete' : True,
                'initializeParams' : {
                    'sourceImage': self.disk_image,
                }
            }
        ],
```

Code Snippet 28 : Example of the virtual machine configuration object

The JSON object also contains metadata for each virtual machine. Google allows for a few hundred megabytes of data to be stored on their metadata server. This allows the virtual machines to access the metadata when they start up. This is used to store the login information for the MongoDB database and the Redis cache, a start-up script and a few other configurations.

The Virtual Machines

Startup Script

When a virtual machine is created it boots and runs the startup script provided in the metadata. This script is shown below in code snippet 29, it is responsible for updating the virtual machine, installing a logger so logs can be viewed via a Google console and installing Docker. Once Docker is installed the script goes and downloads all the metadata from the google servers and stores them as variables in the script. The script also goes and downloads the testagg docker container from Dockerhub. The testagg container contains the Python script that gathers data from the StockTwits and the Tickers.csv file with all the stock names.

The testagg container is stored publicly on Dockerhub and anyone can download it. Instead of storing the login information in the testagg Python script, it was decided it would be passed to the virtual machine when needed and fed into the container. This is what the last line of the script does as it feeds the metadata to the docker container as environment variables which the script can access easily when ran.

```
#!/bin/bash

mkdir logging
cd logging

curl -sSO https://dl.google.com/cloudagents/install-logging-agent.sh
bash install-logging-agent.sh

apt-get update
apt install apt-transport-https ca-certificates curl software-properties-common -y
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
apt update
apt-cache policy docker-ce
sudo apt install docker-ce -y

mongo_user=$(curl http://metadata/computeMetadata/v1/instance/attributes/mongo-user -H "Metadata-Flavor: Google")
mongo_pwd=$(curl http://metadata/computeMetadata/v1/instance/attributes/mongo-pwd -H "Metadata-Flavor: Google")
mongo_add=$(curl http://metadata/computeMetadata/v1/instance/attributes/mongo-add -H "Metadata-Flavor: Google")
redis_add=$(curl http://metadata/computeMetadata/v1/instance/attributes/redis-add -H "Metadata-Flavor: Google")
google_proj=$(curl http://metadata/computeMetadata/v1/instance/attributes/g-proj -H "Metadata-Flavor: Google")
google_zone=$(curl http://metadata/computeMetadata/v1/instance/attributes/g-zone -H "Metadata-Flavor: Google")
instance_zone=$(curl http://metadata/computeMetadata/v1/instance/attributes/i-name -H "Metadata-Flavor: Google")
index=$(curl http://metadata/computeMetadata/v1/instance/attributes/index -H "Metadata-Flavor: Google")
last=$(curl http://metadata/computeMetadata/v1/instance/attributes/last -H "Metadata-Flavor: Google")

sudo docker pull asharkeydit/testagg
sudo docker run --log-driver=gcplogs -e MUSER=$mongo_user -e MPWD=$mongo_pwd -e MADD=$mongo_add -e RADD=$redis_add -e
```

Code Snippet 29 : Virtual machine startup script

Testagg container

The testagg container consists of 3 files. The agg.py file which is a Python script that gathers the data from StockTwits and stores in it the MongoDB database. There is also the tickers.csv file, that holds all the stocks names and is used in the script to loop through and gather data for each stock. The other file is vmdel.py, this file contains the code the worker on the Redis cache runs when a virtual machine is to be deleted. This container was built using Docker and was hosted on Dockerhub for easy access from the virtual machines.

When the Docker container is ran the agg.py script is started. The first thing it does is gather the metadata variables passed in by the startup script and stores them in a dictionary. The script then opens a connection with the MongoDB database using the credentials provided in the metadata. The script then begins to collect the data. To do this it reads in the tickers.csv file and goes to the index passed in by the metadata. From this index it loops and makes a call for each stock name read from the file. The loop is terminated and the data is written to MongoDB. If this is the last virtual machine that was created. It will make a GET request to the ml_trainer endpoint of the Pipeline API to start the algorithm training. After this the script adds a deletion job to the queue on the Redis cache. It passes the deleteVM function in the vmdel.py file along with the virtual machines information as parameters. This function will be executed on the Redis server by the worker and the virtual machine will be deleted.

Ml_trainer – Machine Learning Algorithm trainer

Ml_trainer Operation

The ml_trainer endpoint is responsible for training a new machine learning algorithm. First, it gathers the data from the MongoDB database. The next step is to thoroughly clean the data before preparing it and assembling it into a bag of words. The bag of words is then used as a input to train a random forest classifier. The predictions produced by the classifier are tested and the results are mailed to the Author.

Ml_trainer Implementation

Similar to the twit_agg endpoint the ml_trainer endpoint spawns a process and calls the MlTrainer class from the trainer.py file. The MongoDB credentials and the Authors email are passed to the MlTrainer class to instantiate it. Once the class is instantiated it gathers twits from the database using the Pymongo package. The twits are gathered into two separate arrays ones for the Bulls and one for the Bears. Once the twits are gathered they are then cleaned.

Data cleaning

```
def twit_sanitiser(self,cursor):

    # Sanitize twits with lemmatizer and removing short words

    logging.info(f'Sanitizing Twits {datetime.datetime.now().strftime("%I:%M%p,%B %d, %Y")} ')

    temp_array = []

    for twit in cursor:

        twit['body'] = self.twit_reger(twit['body'].lower())
        if len(twit['body']) != 0:
            tokens = nltk.word_tokenize(twit['body'])
            cleaned_tokens = []
            for i in range(len(tokens)):
                if len(tokens[i]) < 3:
                    continue
                else:
                    tokens[i] = wordnet_lemmatizer.lemmatize(tokens[i])

                    if tokens[i] in stop_words:
                        continue
                    else:
                        cleaned_tokens.append(tokens[i])

            temp_array.append(cleaned_tokens)
        else:
            continue

    return temp_array
```

Code Snippet 30 : Twit cleaning process

The above code shows the cleaning process performed on each twit. Every twit is put through a regex that strips links, hashtags and emojis. The Twit is then converted from a string into a list of individual words called tokens. This list of tokens is put through a lemmatiser and each token is shortened to its dictionary form. After this the list of tokens is checked for stop words. If any of the tokens are stop words such as “a”, “and” or “but” they are removed.

Bag of words conversion

Once the Twits have been cleaned, a bag of words is created. A bag of words is essentially a dictionary that counts the amount of times each word has appeared. For example, take the sentence “Hello my my friend”. If this sentence was converted to a bag of words it would be a Python dictionary and the words “hello” , “my” and “friend” would be keys of the dictionary. The amount times the word appeared is stored as the value associated with the key. So in this example the dictionary would look like {“hello” : 1, “my” : 2 , “friend” :1}. Every unique word encountered in the Twits is added to the bag of words as a key to create a global bag of words that tracks every word used. To ensure that typos do not affect the bag of words once it is constructed all words that have a value of 10 or less are dropped.

Once the global bag of words is constructed it is copied for each individual twit to create a local bag of words. This local copy has the same keys as the global dictionary but all the values are zeros. The local copy will then be filled with values based on how many words in the individual twit exist as keys in the bag of words. If there are 10 twits in the dataset 10 local bag of words are created and filled. This is done for both the Bullish and Bearish categories. This way a local bag of words represents every twit collected.

Dataframe Construction

```
def construct_df(self, bin_bulls, bin_bears):

    logging.info(f'Constructing DF {datetime.datetime.now().strftime("%I:%M%p,%B %d, %Y")} ')

    pickle.dump(list(self.bow.keys()), open(f'cbks/cbk{datetime.datetime.now():%Y-%m-%d-%H:%M:%S}', 'wb'))

    bear_df = pd.DataFrame(bin_bears, columns=list(self.bow.keys()))

    bear_df['twit_sent'] = 'bearish'

    bull_df = pd.DataFrame(bin_bulls, columns=list(self.bow.keys()))

    bull_df['twit_sent'] = 'bullish'

    df = bull_df.append(bear_df, ignore_index=True)

    return df
```

Code Snippet 31 : Dataframe construction process

Once every twit has been converted to a local bag of words a Pandas dataframe must be constructed as this is the input the machine learning algorithm will accept. First two empty dataframes are created, one for Bulls and one for Bears. Then all the keys in the global bag of words are used as columns for both of the dataframes. After this the values from each local bag of words are added as rows to each of the dataframes. Each row represents one twit, the bearish twits are used to fill the bearish dataframe first and then the same is done for the bullish dataframe. Both data frames then have a column at the end named "twit_sent". This column is used to train the algorithm and tells it if the twit being examined is bullish or bearish. The bullish and bearish dataframes are then merged together to form a single dataframe.

It is also important to note that a pickle file is generated here. This file is responsible for storing the keys from the bag of words as a list. This file will then be transferred to the Sentiment API later so that it does not have to construct a bag of words each time a request comes in. This saves quite a bit of time per request.

Algorithm Training

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

scaler = MinMaxScaler(feature_range=(0, 1))

logging.info(f'Scaling {datetime.datetime.now().strftime("%I:%M%p,%B %d, %Y")} ')

scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Code Snippet 32 : Splitting the dataset into testing and training sets

Now that the dataframe has been constructed the machine learning algorithm can be trained. The first step in this process is to set aside 20% of the dataframe created previously. This will be used to test the algorithm once it is trained. The dataframe is then put through a scaler. This will ensure that all values in the dataframe fit in the same range. This is done to speed up the algorithms performance.

```
clf = RandomForestClassifier(n_estimators=200)

print('Fitting RFC')
logging.info(f'Fitting RFC {datetime.datetime.now().strftime("%I:%M%p,%B %d, %Y")} ')
clf.fit(X_train, y_train)

filename = f'pickles/RFC-model-{datetime.datetime.now():%Y-%m-%d-%H:%M:%S}'

logging.info(f'Saving PKL {filename} {datetime.datetime.now().strftime("%I:%M%p,%B %d, %Y")} ')
print(f'saving file name {filename}')

pickle.dump(clf, open(filename, 'wb'))
```

Code Snippet 33 : Fitting and saving the classifier

The next steps in the process involve training the classifier. A scikit learn RandomForestClassifier is created and trained on the data in the training portion of the dataframe. This trained model is then saved as a pickle file so that it can be transferred to the Sentiment API later. This is done to speed up the Sentiment API. It takes considerable time to train a machine learning classifier on this large of a dataset. By saving it and transferring it to the Sentiment API means the API only has to load it into memory and use it.

Algorithm Testing

Once the algorithm is trained it is tested on the testing portion of the dataframe that was put aside previously. The algorithm will return the results from predicting the test data and this is used to generate a classification report. This report provides metrics by which the algorithm can be evaluated by the Author. It breaks down the precision and recall measures for both Bullish and Bearish categories.

Mailing Results

When the classification report has been created it is emailed to the author using Pythons built in email library. It logs into an email set up for the pipeline called sentwitpipeline@gmail.com. It uses this email address to construct an email which consists of the classification report and sends it to the Authors email.

Figure 51 : Email containing the results of a newly trained algorithm

Sentiment Analysis API development

The Sentiment analysis API was built with Flask and is hosted on a Google Compute Engine virtual machine. It was initially intended that the Pipeline API would have a sentiment analysis endpoint. However it was decided to run the sentiment API separately as resources needed by the Pipeline API were too expensive to afford in the long run.

Sentiment Analysis API Operation

The Sentiment API supports one end point which can be called by the web application. It accepts HTTP requests with a ticker name specified. Once the API receives a request for a stock it will gather the latest StockTwits and Twitter data. Once the data Is gathered it is thoroughly cleaned and put through the machine learning classifier. The predictions made by the classifier are returned to the web application to be displayed.

Sentiment API implementation

The sentiment API employs much of the same code as the pipeline API to perform tasks such as data preparation and machine learning predictions.

The first thing the sentiment API does when it receives a request is to gather the StockTwits data for the stock. As it does this it checks the Redis cache to see if it has already cached sentiment prediction's for this stock already. If it finds prediction's in the cache it will return them. Otherwise it will instantiate the SentimentAnalyser Class which then collects twits and tweets from the respective API's. Once the data is gathered any twits that have a sentiment provided are set aside as a prediction does not need to be made for them.

The next step is to clean both the twits and the tweets. This is done exactly the same as how it is done by the ml_trainer on the pipeline API. The twits and tweets are put through the regex, lemmatised and stop words are removed.

The next step in the process is creating local bag of words for each twit and tweet. To do this the cbk file that stored the keys from the global bag of words on the Pipeline API is loaded into memory. The file contains a Python list of all the words in the global bag of words. An empty bag of words is created using the list of words as keys and filled. This is done for each twit and tweet. These local bag of words are combined into a dataframe to be used as input to the classifier for predictions.

```
def predict(self):

    logging.info(f'Scaling and predicting{datetime.datetime.now().strftime("%I:%M%p,%B %d, %Y")} ')

    scaler = MinMaxScaler(feature_range=(0, 1))

    scaler.fit(self.df)

    self.df = scaler.transform(self.df)

    file = open("RFC_MAIN", 'rb')
    clf = pickle.load(file)

    preds = clf.predict(self.df)

    return preds
```

Figure 52 : Sentiment API function for predicting sentiment

The next step is to make predictions on the twits and tweets. There is no testing involved so the whole dataframe is used as prediction's need to be made for each twit and tweet. Before predictions the data is put through a scaler and scaled. Once the data is scaled the trained classifier that was saved in a pickle file on the Pipeline API is loaded into memory and used to make predictions on the dataframe.

The predictions are then processed and added up before being returned to the web application in JSON form. At this stage the sentiment predictions are stored in the cache for 15 minutes.

```
print('Adding to cache')
rconn.execute_command('JSON.SET', f'{ticker}Sent', '.', json.dumps(ret_json))
rconn.execute_command('EXPIRE', f'{ticker}Sent', 1200)
```

Figure 53 : Example of sentiment caching

Conclusion

The development of this project provided a significant challenge. From beginning development on the web scraper to now has been a long journey. It was quite the learning experience and provided numerous challenges that required innovative solutions. Overcoming issues such as the XBRL parsing in the web scraper, managing data limits and hardware resources were the largest issues faced. Overall development went well and the components constructed followed the intended designs.

Chapter 5 - System Validation

Introduction

This chapter of the report will outline testing and system evaluation. Testing is a necessary part of software development that is essential for finding system halting bugs prior to software releases. Testing was done regularly after each feature was built and a system wide testing phase occurred after development.

System Testing

As the system consists of multiple different aspects such as a web application, multiple APIs and a machine learning algorithm a different approach was used for each.

Web Application Testing

Web application testing was vital to ensure the project functions as designed. As this is the interface through which the users will be interacting with the system it is important that a manual testing approach is put in place. By using a manual testing approach a typical user flow can be replicated and tested for errors. Along with manual testing, usability testing was conducted to ensure that the system is accessible to all users regardless of IT skills. Automated Unit tests are also in place to ensure no regressions are introduced after code changes.

Manual Testing

Manual testing also known as black box testing was used for the project to ensure that the system worked as designed and the necessary requirements were met. Test cases were developed to ensure that all aspects of the system were tested and recorded. Testing took place on multiple devices, mainly a 13 inch laptop, 15 inch laptop and 27 inch monitor to ensure the page elements displayed correctly on multiple screens. Although the project was mainly designed for google chrome other browsers were tested such as Safari and Edge. This phase of testing mainly ensured that the UI elements were displaying correctly with the correct data.

Findings

When the manual test cases were carried out some issues were found in the system. For example if the search bar was left empty when searching for a stock a Django error was caused because a null value was being sent to the backend. Some minor UI issues were flagged also like incorrect button names.

Usability Testing

As this project is user facing it is important to get users to test the system. To do this a group of users were asked to evaluate the website in terms of usability and provide feedback. The group of test users vary in their level of IT skills and their familiarity with the stock market. Each user was provided a list of steps to complete in order to fully navigate the system. A walkthrough script was written and sent to the group to follow. Once the users have finished the walkthrough they were asked to fill out a survey monkey survey to gather information on their experience. The participants were asked about their exposure to the stock market and how they felt using the Stockify website.

The feedback was interesting and quite positive. Majority of the test users had little exposure to the stock market with only 21% of the testers owning shares in a publicly traded company.

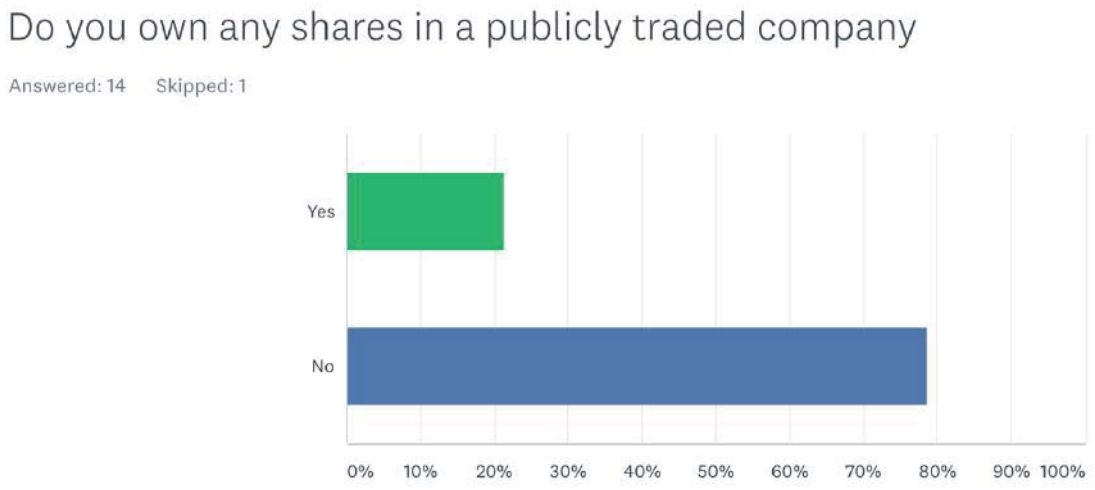

Figure 54 : Survey monkey responses to financial question

The users were then asked about their experience using Stockify. When asked to rate their experience out of 100 the average answer was 88. The users were also asked about page navigation and if the websites features were easy to find. The overall response was very positive with 1 user expressing an opinion that the “return to homepage” link should be more visible. This is valid criticism and attempts were made to enhance it prior to final submission.

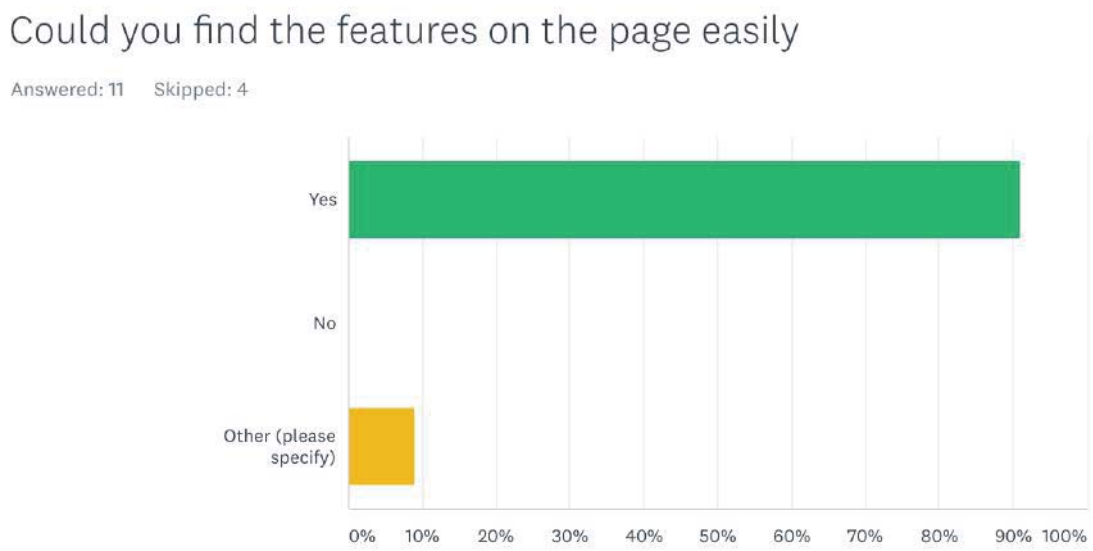

Figure 55 : Survey Monkey responses to usability questions

The users were also asked their opinion on the choice of colour scheme for the site and the overall design. Feedback was overwhelmingly positive in this regard with 100% of the participants liking the design and colour scheme.

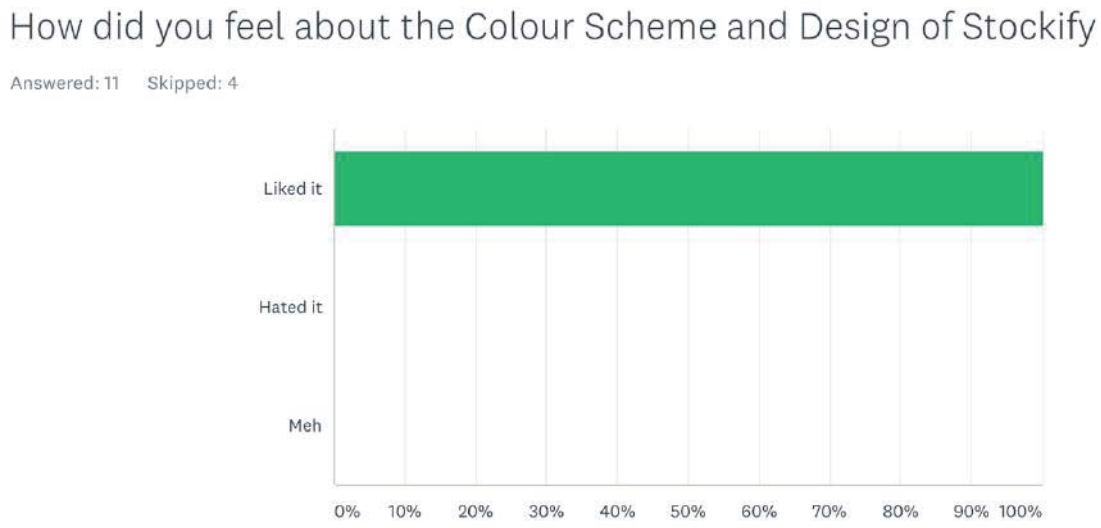

Figure 56 : Survey Monkey responses to design questions

Finally the users were asked about the information on the website. They were asked if they found the information useful and would they visit Stockify to inform their investment decisions in the future. Again responses were extremely positive with all participants agreeing that they found the site useful.

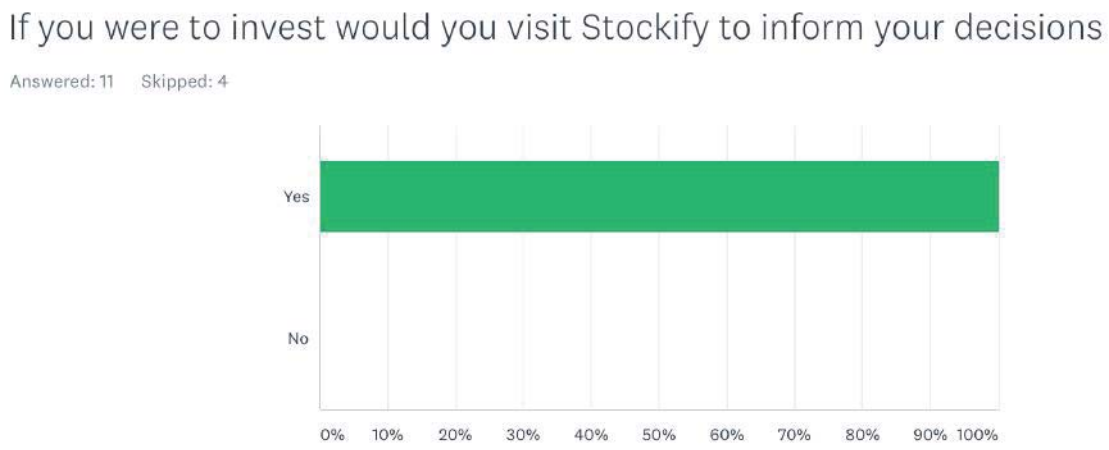

Figure 57 : Survey Monkey responses to informational questions

At the end of the survey there was a text box where users could give a brief opinion on the website in general. Some issues were highlighted and some valid points were made that will be addressed.

Findings

Overall, the usability testing was a success. The majority of the respondents felt that the platform worked well and the information presented on it was of use. Some issues were raised and rightfully so, time was given to better enhance these issues prior to final submission. Some of the feedback mentioned features that may be of use that are missing from the platform. One user remarked on the lack of a search engine and that they could only use the ticker names to search. This feedback served as an indication as to what future work could be done to the platform.

Automated testing

The web application is tested regularly using unit tests on Jenkins. Jenkins is a continuous integration service that checks for changes in the codebase on GitHub. If a change is detected Jenkins will download the code changes and run unit tests for the web application. Unit tests are written in python and are ran through Django. The unit tests are responsible for checking the various web pages have loaded correctly and that users can login. Page elements are also checked to ensure that the correct content is loaded and no errors are returned.

```
def test_index(self):
    response = self.client.get('/')
    self.assertEquals(response.status_code, 200)

def test_about_us(self):
    response = self.client.get('/about/')
    self.assertEquals(response.status_code, 200)

def test_login(self):
    response = self.client.get('/accounts/login/')
    self.assertEquals(response.status_code, 200)

def test_signup(self):
    response = self.client.get('/signup/')
    self.assertEquals(response.status_code, 200)
```

Code Snippet 34 : Python unit tests for the web application

Findings

Having Jenkins continuously monitoring the codebase has to be one of the highlights of the projects. On a few occasions some of the unit tests failed and indicated that some regressions had been introduced to the system. It was a confidence booster knowing that the code is being continually checked and it has genuinely helped during the development of this project.

API Testing

Both the Financial and Sentiment API's play a large role in the web platforms operation. It was essential to thoroughly test the APIs to ensure that each endpoint functions as required. Extensive manual testing was employed on both API's to test their operation before deployment. As there was no visual interface through which the API's can be interacted with, a testing tool was needed. To test the API's, Postman was used to test HTTP requests on all of the endpoints.

Manual Testing With Postman

Postman is a testing tool that provides a GUI through which HTTP messages can be constructed and sent to the API's to test their functionality. Every endpoint developed was thoroughly tested with various sets of test data. This was to ensure that not only did the endpoint work but that it also provided the correct results. Figure 58 below shows an example of a postman test of the portfolioAnalyser endpoint on the Financial API.

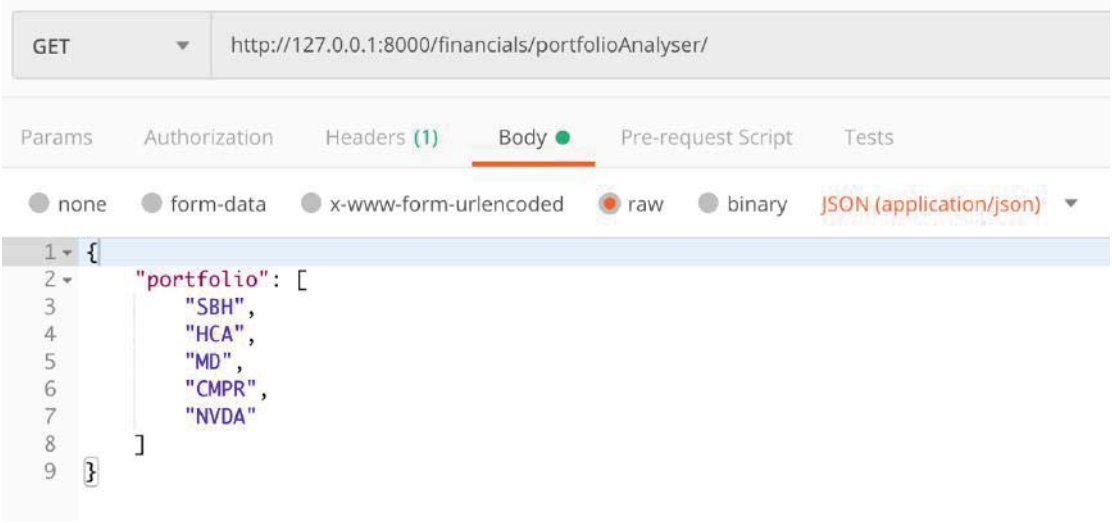

Figure 58 : Postman API test on the Financial API

Findings

Unit tests could have been written for the endpoints to ensure that they return a 200 code and are working. However, it was decided that manual testing would be carried out as the data is changing daily and some of the calculations are easier to verify manually. Postman has been a great aid in this regard and has been helpful in catching some errors in the API logic.

Machine Learning Validation

Ensuring that the sentiment classifier provides an acceptable level of accuracy is very important for the systems operation. To ensure this the algorithm is tested each time it is trained. A hold out cross validation approach was employed to ensure that an adequate amount of testing data was put aside to test the classifiers accuracy.

Cross validation

Cross validation works by splitting the dataset being used in two before training the algorithm. The ratio is typically 20% of the dataset is set aside for testing and the other 80% is used to train the algorithm (61). Once the algorithm is trained on the training data, the test data is used to test the algorithm to see how it performs on unseen data.

Classification Report

The predictions the algorithm makes about the testing data is used to create a classification report which outlines various aspects of the algorithms performance. The report provides numerous metrics for each class which in this projects case is Bulls and Bears. Figure 59 below shows the classification report that is emailed after testing is concluded.

Classification Report of latest ML job

	precision	recall	f1-score	support
bearish	0.76	0.59	0.66	1412
bullish	0.92	0.96	0.94	7383
micro avg	0.90	0.90	0.90	8795
macro avg	0.84	0.78	0.80	8795
weighted avg	0.90	0.90	0.90	8795

Figure 59 : Example classification report

The report provides three scores precision, recall and the f1-score. These scores are used in place of accuracy as they better reflect the decisions made by the classifier.

Recall

Recall provides the number of true positives that were detected by the algorithm. From the example above 59% of bears were identified meaning 41% were missed. It is used to discern how many of the class were actually predicted (62).

Precision

Precision tells us the number of predicted positives that were actually positive. From the example above of all the bears that were predicted only 76% of them were actually bears. It tells us how discerning the algorithm is in making its decision (62).

F1 – Score

The F1 score is a combination of both precision and recall. It is useful when comparing algorithms in terms of predication performance (63) .

Support

Support simply outlines the number of each class that are present in the testing dataset.

Findings

As a new sentiment classifier is being trained regularly it was very important to monitor its performance as the dataset grows. Having the classification report delivered via email was extremely useful. The metrics provided are simple to understand but provide more detail than a simple overall accuracy. As there is a significant imbalance between the two classes in the dataset, monitoring the classifiers accuracy on the minority class was important. Ensuring the algorithm was not overly biased was essential in providing a useful classifier.

System Evaluation

When designing and evaluating the website Nielsen's Heuristics were used to guide some of the design choices that went into the web applications user interface. Below the guidelines will be outlined and the projects user interface will be evaluated according to each one (64).

Nielsen's Heuristics

Visibility of System Status

As there is a great deal of API and database interaction it is essential that the user is aware of the systems status when certain operations are performed. To provide this the web application uses session messages which are displayed on the page to alert the user to the success or failure of an operation.

Match Between System and The Real World

To ensure that a match is made between the system and the real world all messages presented to the users are in plain English without technical jargon. The only place where this may fail is in the use of financial terms. These can be confusing words to amateur investors and may cause some issues for amateurs wishing to use the system.

User Control and Freedom

Control is provided to the user in the form of return links on each page. The design of the website aids freedom and control as there are few webpages the system provides and none of them go two pages deep without an exit or return of some sort.

Consistency

Stockify consists of four unique webpages yet it provides information on over 5000 thousand companies. The web application achieves consistency by not only employing identically styled UI elements but by using a template as the base for each page the user interacts with. This ensures a consistent layout for every company and its information.

Error Prevention

By implementing a micro service architecture certain aspects of the web platform may not be functioning but the web application will continue to function just without the faulty component. An example of this is the Sentiment Analysis API, if the Sentiment

API goes down the web application will still load a Stocks information page. This prevents the user encountering session ending errors.

Recognition Rather Than Recall

The interactive elements on the web application are kept to a minimum, however when they are available for use they are vibrantly coloured and provide a textual description of their role. This limits the recall necessary on the users behalf as the elements explain themselves.

Flexibility and Efficiency of Use

By limiting the number of webpages being displayed by the web application ensures that there is a short travel across the website from one destination to the other. Hosting portfolio analysis and information on the homepage cuts out the time taken to navigate to a separate page making the website flow more efficient.

Aesthetic and Minimalistic Design

In an attempt to adhere to a minimalistic aesthetic there was a great deal of consideration put into designing the stock page. Vital information is directly displayed and more in depth information such as articles and filings are hidden behind a web link as to not clutter the page. The web application also adheres to a colour scheme with a small palette of 5 colours so as to keep the design as simple as possible.

Helping Users Recognize, Diagnose, And Recover from Errors

As mentioned previously, users are presented with messages relating the success or failure of the websites operations. These messages are coloured green or red to indicate their status and are written in plain English to best help the users understand what is going on.

Help and Documentation

Unfortunately the system lacks in this regard. Many of the technical terms featured on the website do not provide an explanation as to what they mean, this burden is left to the user to figure out. The search bar only provides ticker searches meaning that if a user wishes to search for a company but does not know the ticker, they must consult another source to find the ticker before being able to proceed on the web application.

System Demonstration

A demonstration of the system working can be viewed on YouTube https://www.youtube.com/watch?v=Oe9MmmOPFv0.

Conclusion

From the testing and evaluation conducted above the overall results indicate that the project functions well and is suited for use by a broad range of users. Manual testing confirmed that all actions that can be performed on the system are working as intended after development. The usability survey undertaken by users provided extremely positive feedback on the project and its design. Some issues were highlighted by the users and were addressed before submission or will be examined as future work. Evaluating the system using Nielsen's heuristics shows that the application complies with the heuristics and provides good error prevention and minimalistic design but lacks some documentation on its functionality.

Chapter 6 – Project Plan

This chapter of the report outlines the plan that was initially put in place for the project and how the approach may have changed during the course of development. This chapter will also outline future work related to the project.

Project plan

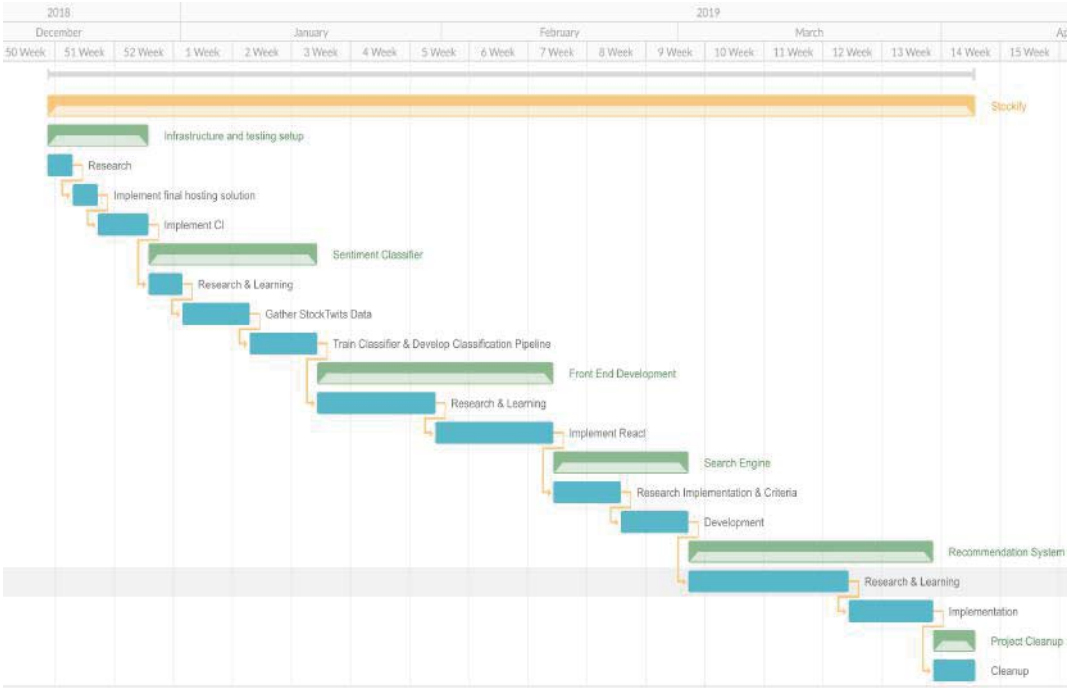

Figure 60 : Gantt chart submitted with interim report

Figure 60 above shows the proposed project plan that was submitted with the interim report. It outlines the numerous other features that were intended for the platform. Unfortunately all of these milestones could not be met. There are a number of reasons for certain features being missing.

Initially it was intended that a JavaScript frontend would replace the Django web application after the prototype. This was changed as the code written for the prototype was viable and it would waste effort starting from scratch again on the frontend. This decision was further reinforced by the time constraints. The sentiment analysis and machine learning pipeline took much longer than forecasted to implement. As a result ensuring that the most necessary features were taken care of first was essential. More importance was placed on perfecting the Sentiment analysis classifier and its pipeline over the less critical parts of the system such as the search engine and recommendation system.

Despite falling short of certain features, the most necessary features to the system were implemented correctly. The three pillars of the project, data aggregation, portfolio analysis and sentiment analysis were addressed as planned and implemented according to the designs.

The agile methodology and time management was essential in developing the main features. The weekly sprints provided much needed feedback while the methodology itself provided the flexibility necessary to address issues as they appeared.

Though the author does not regret aiming to achieve what was set out, it must be said that it proved to be a difficult process only made more troubling by the ambition of its scope. If this project was to be attempted again, ensuring that the scope is not too broad would be a vital factor of design and development.

Future Work

The Stockify project has the potential to be a fully-fledged service. There are many improvements that could be made to the project. Firstly, the features identified in the project plan in the previous section would be valuable additions to the platform. A search engine would make the site more user friendly and stock recommendations could improve investors decisions immensely.

Other features not previously mentioned could be implemented on the platform that would provide useful insight to investors. Large scale stock correlation calculations could help users identify stocks that have very little correlation to their own and adding them to their portfolio would better diversify their investments. This could be done with cluster computing in a similar vain to the data collection cluster employed by the pipeline API.

Providing users the ability to post their own opinions on individual stocks or the market itself would provide insight to our users and another avenue of data upon which insight could be gained with the use of machine learning.

In terms of improving pre-existing features in the project much can be done. It is not possible to store all tweets and twits so as to give insights on how the sentiment surrounding a company has changed over time. If more resources were available, addressing this issue would enhance the platform substantially.

Lastly, much could be done to improve the systems documentation. It was intended for the system to be simple to use when researching investments. Some financial terms are used throughout the system especially in the financial calculations section. No explanation is provided for the user as to what these terms mean. Incorporating some documentation into the GitHub repository or providing a page on the website that

outlines its use in depth would be useful. Adding a link to each financial calculation pointing to Investopedia or a webpage that explains their meaning and use would also be another possible implementation.

Chapter 7 – Conclusion

This chapter of the report provides an overall reflection of the report, it includes how the projects objective was met and the learning experience that was provided.

The main intention behind this project was to provide a solution that would help amateur investors with stock research. It was a combination of passion for investing and computer programming as well as a lack of accessible solutions that lead to the solution outlined in this report. The current manual investment research solution employed by amateur investors such as the author of this paper is slow and cumbersome. It requires manual aggregation from multiple sources to stitch together a cohesive picture of an investments viability. It is hoped that by gathering and enhancing these data sources for thousands of stocks will have in the end helped the investor perform their research and take them one step closer to their financial goals. This project serves to show the interesting results of combining financial data and innovative techniques such as machine learning.

The project has provided an enormous learning opportunity. It has taught the value of adequate research and design prior to development, as the scale and time duration of the project emulates the lifecycle of an industry application. No doubt the lessons learned over the previous number of months will prove invaluable to the author moving forward to industry and in life. This project proved to be an incredibly trying experience, not only technically but both mentally and emotionally. It was a unique experience that had its fair share of ups and downs but provided a platform upon which the knowledge gathered from the education the author has received can be displayed.

Bibliography

1. Chen J. Stock Market [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/terms/s/stockmarket.asp

2. Vilas Shukla. Top 10 Largest Stock Exchanges In The World By Market Capitalization [Internet]. ValueWalk. 2019 [cited 2019 Apr 9]. Available from: https://www.valuewalk.com/2019/02/top-10-largest-stock-exchanges/

3. Chen J. Euronext [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/terms/e/euronext.asp

4. Chen J. Shares [Internet]. Investopedia. [cited 2019 Mar 23]. Available from: https://www.investopedia.com/terms/s/shares.asp

5. Investopedia. Do You Understand the Voting Rights of Common Stock Shareholders? [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/ask/answers/040315/what-can-shareholders-vote.asp

6. Adam Hayes. IPO Basics: What Is An IPO? [Internet]. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/university/ipo/ipo.asp

7. Andrew Beattie. The SEC: A Brief History Of Regulation [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/articles/07/secbeginning.asp

8. Kennon J. What Is Insider Trading and How Does the Practice Break the Law? [Internet]. The Balance. [cited 2019 Apr 9]. Available from: https://www.thebalance.com/what-is-insider-trading-and-why-is-it-illegal-356337

9. Mike Moffat. The ImClone Insider Trading Case: What did Martha Really Do? [Internet]. ThoughtCo. [cited 2019 Mar 23]. Available from: https://www.thoughtco.com/martha-stewarts-insider-trading-case-1146196

10. Bloomberg. United States Rates & Bonds [Internet]. Bloomberg.com. [cited 2018 Dec 2]. Available from: https://www.bloomberg.com/markets/rates-bonds/government-bonds/us

11. Sam Fleming. US Fed raises rates despite trade concerns [Internet]. The Irish Times. [cited 2018 Dec 2]. Available from: https://www.irishtimes.com/business/economy/us-fed-raises-rates-despite-trade-concerns-1.3642606

12. U.S. - projected inflation rate 2008-2023 [Internet]. Statista. [cited 2018 Nov 18]. Available from: https://www.statista.com/statistics/244983/projected-inflation-rate-in-the-united-states/

13. Little K. Why You Should Invest in Stocks [Internet]. The Balance. [cited 2018 Dec 2]. Available from: https://www.thebalance.com/part-one-the-stock-market-doesn-t-care-about-you-3141062

14. Trading Basics- Factors that Influence Share Prices [Internet]. Wall Street. 2018 [cited 2019 Apr 9]. Available from: https://wall-street.com/trading-basics-factors-influence-share-prices/

15. Justin Kuepper. Technical Analysis: Fundamental Vs. Technical Analysis [Internet]. Investopedia. 2006 [cited 2018 Dec 1]. Available from: https://www.investopedia.com/university/technical/techanalysis2.asp

16. WallStreetMojo. Fundamental Analysis vs Technical Analysis | Side by Side Comparison [Internet]. Learn Investment Banking: Financial Modeling Training Courses Online. 2015 [cited 2018 Nov 3]. Available from: https://www.wallstreetmojo.com/fundamental-analysis-vs-technical-analysis/

17. Matthew Cochrane. The Greatest Value Investors of All-Time [Internet]. [cited 2018 Dec 1]. Available from: https://www.fool.com/investing/2018/05/02/the-greatest-value-investors-of-all-time.aspx

18. Daytrading.com. Technical Analysis for Day Trading - Tutorial, Indicators and Examples [Internet]. [cited 2018 Dec 1]. Available from: https://www.daytrading.com/technical-analysis

19. PacktPub. How do we measure a security's risk - Python for Finance: Investment Fundamentals and Data Analytics [Video] [Internet]. [cited 2018 Dec 2]. Available from: https://www.packtpub.com/mapt/video/application_development/9781789618976/81938/81939/how-do-we-measure-a-security%27s-risk

20. Motley Fool. What You Need to Know About Systematic Risk - [Internet]. The Motley Fool. 2017 [cited 2018 Dec 2]. Available from: https://www.fool.com/investing/2017/11/29/what-you-need-to-know-about-systematic-risk.aspx

21. PacktPub. The benefits of portfolio diversification - Python for Finance: Investment Fundamentals and Data Analytics [Video] [Internet]. [cited 2018 Dec 2]. Available from: https://www.packtpub.com/mapt/video/application_development/9781789618976/81938/81941/the-benefits-of-portfolio-diversification

22. Steve Reed, Malik Crawford. How does consumer spending change during boom, recession, and recovery? : Beyond the Numbers: U.S. Bureau of Labor Statistics [Internet]. [cited 2018 Dec 2]. Available from: https://www.bls.gov/opub/btn/volume-3/how-does-consumer-spending-change-during-boom-recession-and-recovery.htm

23. Fontinelle A. Systematic Risk [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/terms/s/systematicrisk.asp

24. Chen J. Unsystematic Risk [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/terms/u/unsystematicrisk.asp

25. Investopedia. How does Beta reflect systematic risk? [Internet]. Investopedia. 2015 [cited 2018 Dec 2]. Available from: https://www.investopedia.com/ask/answers/031715/how-does-beta-reflect-systematic-risk.asp

26. Unicorn Bay. What is the difference between absolute and relative volatility for a security? How it is been calculated? [Internet]. Knowledgebase. 2016 [cited 2018 Dec 3]. Available from: https://unicornbay.com/knowledgebase/what-is-the-difference-between-absolute-and-relative-volatility-for-a-security-how-it-is-been-calculated/

27. Terry Lane. How Do I Get a Stock Beta Value? [Internet]. [cited 2018 Dec 2]. Available from: https://finance.zacks.com/stock-beta-value-8004.html

28. PacktPub. Understanding and calculating a security's Beta - Python for Finance: Investment Fundamentals and Data Analytics [Video] [Internet]. [cited 2018 Dec 2]. Available from: https://www.packtpub.com/mapt/video/application_development/9781789618976/81959/81961/understanding-and-calculating-a-security%27s-beta

29. Will Kenton. Beta [Internet]. Investopedia. 2003 [cited 2018 Nov 18]. Available from: https://www.investopedia.com/terms/b/beta.asp

30. James Chen. Modern Portfolio Theory (MPT) [Internet]. Investopedia. 2012 [cited 2018 Nov 18]. Available from: https://www.investopedia.com/walkthrough/fund-guide/introduction/1/modern-portfolio-theory-mpt.aspx

31. Thune K. How is Modern Portfolio Theory Used With Investing? [Internet]. The Balance. [cited 2018 Dec 2]. Available from: https://www.thebalance.com/what-is-mpt-2466539

32. The Synergy Of Momentum And Value | Market Tamer [Internet]. [cited 2018 Dec 3]. Available from: https://www.markettamer.com/blog/the-synergy-of-momentum-and-value

33. Bernard Brenyah. Efficient Frontier & Portfolio Optimization with Python [Part 2/2] [Internet]. [cited 2018 Nov 18]. Available from: https://medium.com/python-data/efficient-frontier-portfolio-optimization-with-python-part-2-2-2fe23413ad94

34. PacktPub. Introducing the Sharpe ratio and the way it can be applied in practice - Python for Finance: Investment Fundamentals and Data Analytics [Video] [Internet]. [cited 2018 Dec 2]. Available from: https://www.packtpub.com/mapt/video/application_development/9781789618976/81959/81965/introducing-the-sharpe-ratio-and-the-way-it-can-be-applied-in-practice

35. MonkeyLearn. Sentiment Analysis: nearly everything you need to know [Internet]. MonkeyLearn. 12:20 [cited 2019 Apr 9]. Available from: https://monkeylearn.com/sentiment-analysis

36. Smith T. Market Sentiment Definition [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/terms/m/marketsentiment.asp

37. Kramer L. Digging Deeper Into Bull And Bear Markets [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/insights/digging-deeper-bull-and-bear-markets/

38. Kuepper J. Efficient Market Hypothesis (EMH) Definition [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/terms/e/efficientmarkethypothesis.asp

39. Investopedia. Can the Efficient Market Hypothesis explain economic bubbles? [Internet]. Investopedia. [cited 2019 Apr 9]. Available from: https://www.investopedia.com/ask/answers/042015/can-efficient-market-hypothesis-explain-economic-bubbles.asp

40. Gilburt A. Sentiment Speaks: Why Is The Market Not Trading On Fundamentals Lately? [Internet]. Seeking Alpha. 2017 [cited 2019 Mar 23]. Available from: https://seekingalpha.com/article/4096000-sentiment-speaks-market-trading-fundamentals-lately

41. Brownlee J. Supervised and Unsupervised Machine Learning Algorithms [Internet]. Machine Learning Mastery. 2016 [cited 2019 Apr 9]. Available from: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

42. Bronshtein A. A Quick Introduction to K-Nearest Neighbors Algorithm [Internet]. Noteworthy - The Journal Blog. 2017 [cited 2019 Apr 9]. Available from: https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

43. Statsoft. Naive Bayes Classifier [Internet]. [cited 2019 Apr 9]. Available from: http://www.statsoft.com/Textbook/Naive-Bayes-Classifier

44. Bickerton C. A beginner's guide to decision tree classification [Internet]. Towards Data Science. 2018 [cited 2019 Apr 9]. Available from: https://towardsdatascience.com/a-beginners-guide-to-decision-tree-classification-6d3209353ea

45. Donges N. The Random Forest Algorithm [Internet]. Towards Data Science. 2018 [cited 2019 Apr 9]. Available from: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

46. imbalanced-learn. imblearn.over_sampling.SMOTE — imbalanced-learn 0.4.3 documentation [Internet]. [cited 2019 Apr 9]. Available from: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html

47. Marco Altini. Dealing with imbalanced data: undersampling, oversampling and proper cross-validation [Internet]. Marco Altini. [cited 2019 Apr 9]. Available from: http://www.marcoaltini.com/2/post/2015/08/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation.html

48. MongoDB. Comparing PostgreSQL and MongoDB [Internet]. MongoDB. [cited 2018 Oct 6]. Available from: https://www.mongodb.com/compare/mongodb-postgresql

49. MongoDB. MongoDB and MySQL Compared [Internet]. MongoDB. [cited 2018 Nov 18]. Available from: https://www.mongodb.com/compare/mongodb-mysql

50. PostgreSQL. PostgreSQL: About [Internet]. [cited 2018 Dec 2]. Available from: https://www.postgresql.org/about/

51. Compose. Compose Compare: MongoDB vs. PostgreSQL [Internet]. Compose Articles. 2017 [cited 2018 Dec 2]. Available from: https://www.compose.com/articles/compose-compare-mongodb-vs-postgresql/

52. OpenSource.com. What is Docker? [Internet]. Opensource.com. [cited 2018 Nov 29]. Available from: https://opensource.com/resources/what-docker

53. Docker. Why Docker [Internet]. [cited 2018 Dec 2]. Available from: https://www.docker.com/why-docker

54. Redis [Internet]. [cited 2019 Apr 9]. Available from: https://redis.io/

55. Prateek Gogia. Redis: What and Why? – codeburst [Internet]. [cited 2019 Apr 9]. Available from: https://codeburst.io/redis-what-and-why-d52b6829813

56. ThoughtWorks. Continuous integration | ThoughtWorks [Internet]. [cited 2019 Apr 9]. Available from: https://www.thoughtworks.com/continuous-integration

57. Mendix. Agile Framework – A Quick Introduction & Overview of Agile Project Management Methodology [Internet]. Mendix. [cited 2018 Nov 26]. Available from: https://www.mendix.com/agile-framework/

58. Scrum. What is Scrum? [Internet]. Scrum.org. [cited 2018 Dec 3]. Available from: https://www.scrum.org/resources/what-is-scrum

59. Will Kenton. eXtensible Business Reporting Language - XBRL [Internet]. Investopedia. 2006 [cited 2018 Dec 3]. Available from: https://www.investopedia.com/terms/x/xbrl.asp

60. Kenton W. Generally Accepted Accounting Principles (GAAP) [Internet]. Investopedia. [cited 2019 Apr 10]. Available from: https://www.investopedia.com/terms/g/gaap.asp

61. Drakos G. Cross-Validation [Internet]. Towards Data Science. 2018 [cited 2019 Apr 10]. Available from: https://towardsdatascience.com/cross-validation-70289113a072

62. saxena shruti. Precision vs Recall [Internet]. Towards Data Science. 2018 [cited 2019 Apr 10]. Available from: https://towardsdatascience.com/precision-vs-recall-386cf9f89488

63. Jason Brownlee. Classification Accuracy is Not Enough: More Performance Measures You Can Use [Internet]. [cited 2019 Apr 10]. Available from: https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

64. Jakob Nielsen. 10 Heuristics for User Interface Design [Internet]. Nielsen Norman Group. [cited 2019 Apr 10]. Available from: https://www.nngroup.com/articles/ten-usability-heuristics/