

# *FinSpotter: Sentiment Analysis in Financial News*

MSc Applied Computing  
Project Report (2019-2020)



*Jacob Ellis*

*Matriculation number: 140014972*

*Word count: 11133*

---

## Declaration

I declare that the program and achievements described in this dissertation have been carried out honestly, and with integrity in partial fulfilment of the MSc Applied Computing degree 2019/2020.

**Jacob Ellis**

Any queries regarding the veracity of this statement can be sent to:

J.Lawrence @dundee.ac.uk

## Acknowledgements

*As with my last dissertation, this is dedicated to both my Grannies, without whom none of my academic success would have been possible.*

The project has benefitted from the guidance offered from John Lawrence, and so a special thanks to him.

# Table of Contents

<u>Page 1</u>	Abstract
<u>Page 2</u>	Literature Review
<u>Page 5</u>	Technologies
<u>Page 10</u>	Software Architecture
<u>Page 13</u>	User Interface
<u>Page 17</u>	Testing Accuracy
<u>Page 21</u>	Conclusions
<u>Page 22</u>	References
<u>Page 24</u>	Appendices

# ‘FinSpotter’: Sentiment Analysis in Financial News

*Abstract:* The aim of this project has been to take information from four prominent financial forecasting blogs, and to perform sentiment analyses on companies which feature in these, indicating whether a company is being discussed positively or negatively. This program (FinSpotter) is designed to measure the aggregate opinions of a group of experts, across multiple articles and combine these scores into a final output. The reports will indicate whether general sentiment toward a company is favorable, or unfavorable – with a numerical sentiment score. Four blogs have been identified as candidates for this process, from these, a total of 100 articles are sampled. The input data will be the text body from each article, which is then used for analyses. Output will be in the form of a report which will list the number of occurrences of a named entity, which blogs have mentioned a company, and a sentiment score – ranging from (-1) to (+1). This is designed as a proof of concept, and accuracy may be at less than would be satisfactory in a commercial product. There will be three major phases in this program. Firstly, the program will search and retrieve the relevant inputs with a web crawler and import them as simple text files (one for each article). Secondly, Named Entity Recognition (NER) is used to distinguish company names from the text. Thirdly a sentiment analysis will be performed, using company names to determine the relevant sentences to process. This project been designed as a means of learning the basic elements of Natural Language Processing (NLP) and does not provide any novel contributions to the field of Natural Language Financial Forecasting (NLFF). A potential application for sentiment analysis on financial texts could be to identify areas for potential investment in the stock market.

*Disclaimer:* FinSpotter does not claim to inform financial decisions and should not influence any decisions in relation to buying and selling stocks.

## Introduction

Natural Language Processing (NLP) has progressed significantly in recent years thanks to developments in technology and deep learning. The internet has become the primary method for communication and publication, and with this, there have been challenges in how to record and interpret data. Quantitative data is relatively straightforward to process and categorise with traditional methods, and data scientists have largely been able to keep up with the growth in numerical data that has come with internet prevalence. Qualitative data is comparatively difficult to process, textual data is not a natural format for computers to work with and so a wide range of techniques have had to be developed to help parse text. In the last two decades this area of study has seen significant growth, as the constraints of computational power have shifted further back. There are a wide range of applications for NLP in various industries, but this project focusses on one area – sentiment analysis applied to financial news articles. The main aim is to be able to extract company names from text and to process the sentences in which that company is mentioned. In theory, this data should be able to point to areas for potential investment, or at least direct attention to companies which are worthy of further investigation. ‘FinSpotter’ has three main phases in the back-end process – data collection (web-crawler), Named Entity Recognition (identifying company titles), and data processing (sentiment analysis).

A literature review is provided at the beginning of this report to provide some context for how the aims and objectives for this project have developed. Following from this is a discussion on the technologies

used in each of the main phases, along with some explanations behind the choices made, and any alternatives that were considered. After explaining the constituent components of FinSpotter, there is a section titled ‘Software Architecture’. This provides a description of the design patterns used, how data changes along this pipeline, and the interaction between modules at each stage. The section titled ‘User Interface’ is dedicated to the front-end aspects of the program. The primary focus for this project has been on the back-end functions, namely collecting and processing textual data in one consolidated package. Despite this, there are aspects of the UI design where user experience has been factored in. The ‘Testing’ section of this report analyses the human data collected for evaluating the accuracy of FinSpotter. There have been considerable issues in completing these evaluations due to the specialised language used in financial news. Both the participants, and FinSpotter performed poorly at this. In the final section of this report, conclusions are drawn with a roadmap for hypothetical areas of improvement.

## Literature Review

One of the fundamental techniques used in NLP is sentiment analysis, and it is this which forms a basis for the FinSpotter program. This section will provide a summary of the preliminary reading carried out for this project, highlighting major developments in the field, and describing any relevance to the project topic. Reading was carried out prior to deciding firmly on a topic, and so some of the literature reviewed has helped to shape the direction and aims of the project. The primary focus for this literature review will be on research which uses NLP techniques on financial texts.

### General NLP texts

Lawrence and Reed (2019) provide a summary of developments in the field of argumentation mining. While the intended audience is one that understands computational linguistics, this has served as a strong starting point to base initial readings. A short history of major developments in Natural Language Processing is provided, and distinctions are made between the generalised applications of NLP and the more specific aims of argumentation mining. Argument mining research looks at ways of identifying argumentative structure (or reasoning) digitally. There are a range of methods for achieving this and this paper goes some way to describing these. Included in this is a description of the foundational techniques with which mining has developed, and sentiment analysis features as one of these techniques. This paper was reviewed before deciding on a formal project topic and gave insights into what was possible using NLP. I had strongly considered attempting a project which sought to identify *Logos*, *Pathos*, and *Ethos* in political communications, but had to remain realistic in what was possible with the time allotted. After reviewing this summary on argument mining, I made the decision to pursue the sentiment analysis of financial texts as the project goal. This work was useful for highlighting areas for further study and introduced further general texts that are less relevant to the aims of this project, but which helped improve my understanding of NLP.

### Sentiment Analysis on Financial Texts

Once an objective for this project had been decided, further reading into applications of NLP on financial texts was a logical next step. Fortunately, I came across a paper titled – *Natural language based financial forecasting: a survey* (Xing et.al. 2017) which provided a helpful overview of this field of study. As with the previous paper, the terminology and expertise were not intended for novice

researchers. While much of the terminology was far beyond my initial levels of comprehension, after considerable study, I was able to gain some traction on the direction this project would take. NLP techniques have been applied to financial market prediction under the research heading of Natural Language Based Financial Forecasting (NLFF). Techniques in computational linguistics have developed to the point that it is now possible to capture sentiment and semantics in a more nuanced way than was previously possible; despite this, classifying financial text still presents a major challenge. There is an inherent complexity in doing this, as financial language is specialised, and it is difficult to create a complete comprehensive lexicon for this domain. This work pointed my studies toward literature which was more specified to my project topic. Most notably: *The Stock Sonar* (2011) and *Identifying polarity in financial texts for sentiment analysis: a corpus-based approach* (2015), reviews for which will feature below.

### Lexicon Based Sentiment Analysis

In 2015, Moreno-Ortiz and Fernandez-Cruz published their methodology for integrating domain-specific lexicons into a generic sentiment analyser (Moreno-Ortiz & Fernandez-Cruz, 2015). The general premise is that it would be useful to be able to ‘plug-in’ various lexical resources as needed, and this would provide a standardised sentiment classifier that could be used for difference research disciplines. Their approach seems based in the knowledge-based area of sentiment analysis and relies heavily on a comprehensive set of lexical resources, rather than a set of algorithms. In their methodology, the researchers used a specialised language corpus made using the ‘Mag-Finance’ and ‘News-Money’ sections of the Corpus of Contemporary American English (COCA). This pointed me to further resources for tailoring sentiment analysis toward a financial domain, rather than using a generic lexicon. Words such as ‘Growth’ or ‘Expansion’ tend to have a positive meaning when used in a financial context, though are entirely context dependant when used in a general text. This shows that context is an important factor to consider and highlights a major area of refinement which could happen for this project. For my own project, a generalised sentiment analysis tool named “VADER” will be used which can accept alterations to its standard lexicon. This is one major area of improvement which could have been implemented if there had been more time to complete this project.

Sohangir et. al (2018) looks at measuring financial information collected from social media platforms and blogs. From reviewing NLFF focused papers, the consensus seems to be that machine learning models produce more accurate sentiment analysis results, though this paper argues that a lexicon-based approach may be the better option. Experiments are produced using their ‘StockTwits’ dataset, and the results show the accuracy score of ‘VADER’ to be the highest. It should be noted that the F-score values were higher for machine learning models. Some bold conclusions are drawn at the end, where the paper states that ‘VADER’ outperformed machine learning methods in extracting sentiment from financial social media posts and that this model was faster. The findings from this gave me some confidence in using ‘VADER’ for the sentiment analysis element of the project. The methods used for carrying out these experiments seem useful, particularly the annotated ‘StockTwits’ data set, and so understanding this helped to shape the experiment design for this project.

### Machine Learning Based Sentiment Analysis

While searching for ‘state of the art’ techniques for classifying sentiment analysis in financial texts, I came across an impressive master’s thesis titled: *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models* (Araci, 2019). It begins with some recognition that financial sentiment

analysis is difficult due to the specialised language used, and that generalised models do not fit this use case efficiently. Instead of using a knowledge-based approach, Araci uses a pre-trained language model which requires less human annotated data to refine. Based on the BERT language model, (Horev, 2018), this research introduces FinBERT, a variant which is tailored to deal with financial NLP tasks. FinBERT is a very impressive development (particularly for a master's thesis) in NLFF and claims to beat every metric on two financial sentiment datasets by using a machine learning approach. Some drawbacks of historical methods are discussed to begin with; methods using neural nets require a large corpus of labelled data and are costly in terms of human effort. Machine learning models which are trained on general corpora do not work effectively, and so specialised training sets were used. The standards set in this paper are incredibly high, and though I have considered trying to implement this methodology, it is beyond my abilities with the time given for completing the project. Despite this, it has highlighted the current state of the art for NLFF well and serves as the model that I would take to improve this project if time allowed for it.

### Hybrid Sentiment Analysis Methods

*The Stock Sonar* (2011) is a stock sentiment analysis tool produced by the Hebrew University of Jerusalem (Feldman et. al, 2011). Its aims were to combine previous techniques in NLFF and claims to have improved on the accuracy of these approaches (at the time it was written). It uses a combination of sentiment dictionaries, phrase level compositional patterns, and predicate semantic events to deliver an output which shows the most discussed stocks for that day. The program uses stock specific RSS feeds to aggregate an input text corpus for the program. Once this feedstock data has been collected, sentiment analyses can be carried out using what they term CARE extraction (CRF Assisted Relation Extraction). This approach uses elements of both Machine-Learning and a knowledge-based approach in its design which is beyond the complexity possible with this project, though it has been useful to see the components used and the functions needed to successfully parse financial texts. The dictionary – based element of the sentiment analyser uses the domain specific *McDonald – Loughran* Lexicon. Seeing this introduced the possibility of using a pre-defined lexicon in my own program. The issue of attention or focus for sentiment analysis is an issue that I became aware of after having read this paper. A basic approach to attention has been taken in FinSpotter: only the sentences in which a company is mentioned are parsed for sentiment scores.

### Initial planning

After reviewing the various approaches used for sentiment analysis, VADER was chosen as the library to proceed with. VADER uses a 'bag of words' approach combined with some use of context pattern matching. This means that it is primarily a lexicon-based approach to sentiment analysis. For the initial stages of designing this program, a Gantt chart was produced to chart some guidelines for the progress of development. The original Gantt chart is included in the additional files folder associated with this report. A development diary is also available in this folder to show that there were differences in the expected development path, and the route taken. There were areas of the program which took longer than expected, particularly the NER functions, and the User Interface (UI). As development began, unforeseen issues and extra features had to be factored in. Something that helped to guide the direction for the program was to consider the elements of the UI that I wanted to see in the final product. This allowed me to plan my design around which types of data would be needed at each phase of the process.

# Technologies

## Overview

There are three key themes which are common to all sentiment analysis programs, those are: data collection, data cleansing, and data analysis. This provided some direction for the tasks which would be needed to develop FinSpotter. Python was chosen as the programming language to proceed with. After this some key decisions were made as to which libraries to use for each task. The 'Requests' library, coupled with 'BeautifulSoup' is used for collecting textual data from each article. This is then written into a text file, which can be processed. 'Spacy' was chosen as the tool for working with this text data and is self-described as an 'industrial method' for parsing text. Once text is in a 'digestible' format, Named Entity Recognition is used (with 'Spacy') to identify general organisation entities. This is then refined by using 'FuzzyWuzzy' to match these organisation entities against a list of 2,000 international companies (collected from a series of excel spreadsheets). Once suitable companies have been matched, sentences which feature these entities are collected for processing. Lastly, for the data analysis stage, the VADER library was chosen as it was at a suitable level of complexity for my project. The User Interface (UI) elements of FinSpotter have been created using the Kivy, Python framework, and are designed with practicality in mind. This section hopes to provide a discussion about the technologies used, and the rationale behind using them.

## Programming Language

The first important decision was in which programming language to use. There were two obvious choices, JavaScript, and Python. JavaScript is well suited to the task with tools like Node.js. though I have a limited understanding of how to work with APIs and JavaScript syntax. I was hesitant to begin trying to learn this method. After consulting with my tutor, it was agreed that Python is the more obvious choice. Python tends to be the main programming language for NLP in academia, and I had some limited knowledge of the syntax used before beginning this project. There are a great number of tutorials available for Python NLP tasks and so this seemed like the better option to begin programming with some momentum. My understanding of Python was limited before undertaking this project and so some time was needed to familiarise myself with the syntax by using a course from Udemy (Portilla, 2020). With this I covered the fundamentals, such as loops, functions, and classes. There were also slightly more advanced elements, such as lambda functions and how to read/write from an external text file. Input/output operations with text files has been useful, as the articles for the program were planned to be saved as simple .txt files.

## Web crawler

Once I had become reasonably confident with basic Python syntax, the next task was to review which libraries were available to perform the tasks that I needed. The first task identified was in collecting the textual data with a 'web crawler'. There are two popular tools for doing this – 'Scrapy' and 'BeautifulSoup'. Scrapy is designed for large scale, mass data mining operations with thousands of website requests and involved a complicated process for setting up and use. After two days of working with Scrapy- it was clear that it would be more efficient to create the prototype with BeautifulSoup. This allowed for much faster progress as this library is more user-friendly, and results were immediately available. One of the drawbacks is that it does not support concurrent web-crawling requests, though this is less of an issue for a project of this scale, which anticipates just 100 webpage requests. It was at this stage that I had to review my knowledge of HTML and CSS by refreshing my understanding of tags, classes, and IDs. The 'Requests' library is used to retrieve the raw HTML file for a given website, from there BeautifulSoup is used to turn this into a special object, which can then be navigated through.



Using methods such as ‘find\_all(“p”, “\_class = article”)’ it is possible to extract the textual data from a webpage, and to write this to a .txt file.

```
else: # If file does not already exist
    html = URLList[count][1]
    req = Request(html, headers={'User-Agent': 'Mozilla/5.0'}) # Defines request parameters
    webpage = urlopen(req).read()
    count += 1
    # print(html)
    MotleyPage = BeautifulSoup(webpage, 'html.parser') # BeautifulSoup object from HTML file
    article = MotleyPage.find('article', class_='large-8 columns clearfix') # Find particular section of webpage
    full_content = article.find('section',
                                {'id': "full_content"}) # Creates object for pulling content from section
    inner_content = full_content.find_all('p') # pulls just text from full_content object
    limit = len(inner_content) - 11 # Removes boilerplate membership text
    cursor = 0
    print("HERE")
```

Figure 1: An example of the web crawler sections for each blog

## Named Entity Recognition

Once I had successfully written information from a webpage to a .txt file, I was acutely aware that I did not know how to parse this information. I spent some time looking into subjects like tokenisation, lemmatisation, and Named Entity Recognition (NER). This helped identify how to approach the next phase of the program – data cleansing. ‘Spacy’ is an impressive tool for NLP and comes bundled with an extensive vocabulary of tagged words and pattern matching functions. I began with an overview of the Spacy library functionality and had some practice with it. The Spacy module converts a string of text into a ‘doc’ object, which allows for methods and operations to be performed on the text. The ‘doc’ object splits text into tokens which are then automatically assigned with ‘Parts of Speech’ (POS) tags and dependencies. These are used to represent the type of words present. Some examples of POS tagging include - proper nouns, verbs, and numbers. NER can take place once tags have been applied and Spacy allows this to be refined even further by identifying the type of entity a proper noun may be. Spacy can also separate text into a series of sentences, which has been useful for the sentiment analysis phase.

I began by practicing with Spacy through following online tutorials (Joshi, 2020) and then moved to integrating it with the web crawling module which outputs .txt files. When ‘Spacy’ detects an organisation, it tags the word with the label ‘ORG’. This was initially used as the primary method for detecting company names but there were some major issues in relying on this. Certain companies such as ‘Tesla’ were recognised with ‘Works-of-art’ or ‘Person’ NER tags, and so I widened the scope to include these for processing. It is possible to enter words into the ‘Spacy’ lexicon and specify a tag for them, though this did not seem like an efficient approach. Another issue with this method of NER is that ‘ORG’ tags are applied to organisations, regardless of whether they are companies or not. Words like ‘United Nations’ or ‘Government’ also get labelled as ‘ORG’ entities. Despite this major flaw, I proceeded to move on to developing the other areas of the program and re-addressed this problem at a later stage. To simplify the process, only sentences in which a company name is mentioned will be passed on for sentiment analysis. This ensures that sentences are relevant, without having to create a complex attention mechanism. If there was more time allowed for the project, this is an area that could be improved.

To solve the issues with Spacy identifying non-companies as ORG entities, I decided to combine the Spacy entity recognition, with an algorithm for matching company names from a list of companies. On the first pass, the program uses Spacy to identify candidate entities by labelling them as 'ORG's. After this, these entity names are compared with an extensive list of company names (2000 present), collected from a series of excel spreadsheets. The spreadsheets used to collect these names hold data for: The Wilshire 5000, The fortune 500, and the Forbes 500 (see 'create\_company\_list.py' in project files). To extract this data, standard excel extraction library 'xlrd' has been used to populate a dataframe (from the 'Pandas' library). The name column in this data frame is then converted to a list which makes subsequent operations easier. The names themselves were legal titles, with suffixes such as 'Ltd.' Or 'Corp.'. This meant that some cleansing of these names was needed. Thankfully, a Python module exists called 'Cleanco' which is designed for this very purpose, though I found that there were many instances where suffixes remained. By combining Cleanco methods with a series of 'Regular Expressions', I finally managed to clean this data thoroughly. It has been useful to include a spread of companies from various regions of the globe, as the blogs sampled often recommend stocks which are not necessarily tied to their published location.

```
def clean_names(names_in):
    clean_names = []
    terms = prepare_terms()
    for name in names_in:
        if (type(name)!='str'):
            name = str(name)
            name = re.sub("Inc.", "", name) # Used to remove legal suffixes
            name = re.sub("Ltd.", "", name)
            name = re.sub("Corp.", "", name)
            name = re.sub("Co.", "", name)
            name = re.sub('[^A-Za-z0-9]+', '', name)
            clean_names.append(baseName(name, terms, prefix=False, middle=False, suffix=True))
    return clean_names
```

Figure 2: Code used to cleanse the company names into a standard form

Once this lengthy list of company titles had been collected and cleaned, it was possible to match candidate entities against this list of verified companies. To account for spelling variance or partial mentions, the 'FuzzyWuzzy' module is used. FuzzyWuzzy can process two strings and return a score for how similar the two strings are as a percentage, using something called the 'Levenshtein' metric (Baber, 2018). The basic principle is that it measures the number of alterations needed to match the two strings perfectly and represents this as a percentage similarity score. Using this it was possible to discard entities which were less than 90% like any of the company titles. This has allowed me to reliably identify company names, even in cases where the company name is more than one word. I had previously considered analysis of proper nouns from the text, but this was not a viable solution, due to the fact many company names are comprised of more than one word. Combining the Spacy NER tools with this method of string matching has created a robust hybrid method, though despite this, it may exclude abbreviated company titles, or partial mentions which match at less than 90%. It is possible to extend this list to include companies from different regions, but for the purposes of this project, the focus has been on UK and US companies. The 'Pickle' module is used to serialise this list for future use.

## Sentiment Analysis Component

The next phase for the program is sentiment analysis, and to gain a better understanding of the process, I revised topics such as word vectorisation and how the ‘bag of words’ approach to sentiment analysis works. There are various methods for sentiment analysis, which can be grouped into two general categories – lexicon based, and machine learning based. The state of the art in -sentiment analysis tends to be driven forward by continually improving on machine learning methods. Pre-trained models such as BERT and GPT-3 are powerful tools which seem to set the benchmark for accurate NLP going into the future. I considered the viability of implementing these for FinSpotter. I was particularly interested in trying to follow the methodology used with FinBERT (Araci, 2009), but it became clear that this was beyond my skillset and would not be possible to implement in the time given for this project. This highlights an area for improvement in this program, which would increase the accuracy of sentiment analysis considerably. Once this had been accepted, I began to research lexicon-based methods and modules. ‘TextBlob’ was one possible library considered, but it seems to have been overtaken in popularity by the ‘VADER’ module. There is some evidence to show that ‘VADER’ outperformed ‘TextBlob’ on financial texts, and so this seemed like the obvious choice (Sohangir et al., 2018).

‘VADER’ stands for ‘Valence Aware Dictionary and sEntiment Reasoner’ and is both a lexicon - based and rule-based sentiment analysis tool. Initially I thought that VADER was simply a ‘bag of words’ model, but I have since learned that it is slightly more complex and that it does consider some context in its calculations. VADER uses some rule-based negation, which involves reversing the sentiment of a set of words, depending on words such as ‘not’ or ‘isn’t’. Another strength of VADER is that it is relatively easy to use, compared with deep learning techniques such as BERT. VADER takes in a sentence as input and outputs four separate scores – negative, neutral, positive, and compound. The compound score is calculated from the others and offers the best way of reviewing a sentence in a single numerical value. It is represented as a float number between -1 (negative) and +1 (positive). FinSpotter provides two averages in analysis, the ‘Macro’ average, and the ‘Micro’ average. For the Macro average - the sum of all compound scores for all sentences (across all articles) related to a company is divided by the number of sentences present. For the Micro average, the mean average for each individual article is divided by the number of articles which feature that company.

## User Interface

Designing the user interface has been one of the more challenging aspects of this project. I began with trying to design a web interface using the ‘Flask’ python framework. I spent some time learning through tutorials and practicing with small projects, and eventually managed to produce a skeleton interface which could interact with my program but did not display outputs on screen. Issues arose in trying to make the interface look more professional, and I attempted to use a bootstrap template to speed this up. Eventually I had to acknowledge that my understanding of JavaScript was not sufficient to complete the interface to a standard that I would be happy with. After this I moved on to building the interface using a python module called ‘Kivy’. Kivy is a popular open source user interface module. It is highly recommended in various forums since it is easy to learn and has a lot of useful features. I completed a few beginner tutorials, and this allowed for work on the interface to begin. An interesting feature of Kivy is in that it has a dedicated design language which makes arranging a UI screen straightforward. A drawback of this would be that things can get complicated when passing objects between Python and the Kivy language.

There are two ways to build an interface using this module, first it is possible to write each widget in python and add it to a '.kv' file for structure; or secondly, it is possible to write everything within the '.kv' file using the Kivy design language (see 'pageDesign.kv' in project files). This language is declarative and functions much in the way that CSS does for webpages. Because my program is quite complex in the information it displays to the user, a mixture of these two techniques has been used in the final product. Kivy has many similar features to other mobile UI design technologies, for instance layouts such as 'GridLayout', 'FloatLayout' and 'BoxLayout' are available. One of the major benefits to using 'Kivy' over other modules is that it produces a program that is compatible on all platforms (Windows, Mac, Android, Linux). While this has not been a specific design goal for this project, it is a nice additional feature. For my project, I have designed this interface with Windows in mind, and I have used the original default dimensions for the program window. Proportions may differ if changed to full screen mode, but the program will certainly still be operable.

### Bar Chart Plotting

The 'Matplotlib' module has been used to create the bar charts from the top 10 companies, and bottom 10 companies (ranked in order of sentiment). This is a standard module and so seemed the obvious choice. Once a bar chart has been created, Kivy can render this to the UI as an image, rather than by using a custom widget. This streamlined the process and allowed for time to be allocated elsewhere.

# Software Architecture

*“Clean code reads like well written prose” – Robert C. Martin*

## Overview

After becoming confident with the constituent parts of the program - extracting textual data from blogs, identifying named entities, and calculating a sentiment score; integration and the overall design of the program was the next stage of development. There were three general iterations of refinement which took place. The first iteration missed some key class attributes and processes, which were needed to output the desired comprehensive set of data. The second iterations addressed this, but program run-time was unacceptably long, which forced yet another re-design. The third iteration incorporated both lessons and makes extensive use of saved objects to speed up report delivery times. An E-R diagram for the first and third iterations is available in the appendices for reference (Appendix 1 & 2).

## Initial design

An Entity-Relationship (E-R) diagram helped to show relationships, classes, and data structures. The first draft of the E-R diagram had some major issues which were only discovered once the implementation of the program began, namely, there was only one ‘ORG’ class. This forced a re-design to address the practical needs of the program. By considering which types of output and information I wanted to see in the final User Interface (UI), this helped to inform which types of data would be needed in each class of the program. For the final form of FinSpotter, the user is presented with a list of all the companies identified from the articles sampled (ordered by sentiment), along with a ‘Macro’ and ‘Micro’ sentiment score for each. There are options to present this data as a bar chart, though it was impractical to represent all companies found, and so instead two bar charts are used – one for the top ten companies, and one for the bottom ten companies. There is also an option to view the sentences parsed for a company, as well as links for visiting the original articles. This allows the user to see the base text for the companies discussed so that they can exercise their own judgement on sentiments. There had been plans to allow the user to save and display previous reports, but this feature fell outside the core functionality and time did not allow for it.

In the initial designs for this program there was a single class used to hold information at every stage of the program. This proved to be clunky, as there were distinct uses for different data formats at each stage in the process. Initially the ‘ORG’ class only held an aggregate sentiment score, and the title of the organisation being discussed (Appendix 1). This produced sentiment scores which were outside of the range -1 to +1. Normalisation of the compound score needs to take place so that the score remains within the range -1 and +1, and so a count of the sentences parsed is important. After identifying the types of output which were required by the UI, it was clear that this class would not capture enough data. To resolve this, the ‘ORG’ class was split into three separate classes. The second iteration did perform, but it used a rather ‘ham-fisted’ approach to data cleansing and analysis. A series of list objects, at different levels of abstraction (article level, consolidation level, output level) were created to produce an overall average for all sentences parsed. This approach used a series of nested loops, only output an average for all sentences in all the articles sampled (‘Macro’ average), and crucially – required a full analysis each time it was run.

## Object oriented lists

The third iteration has combined the lessons learned in previous versions, and now has a more practical class design, combined with making use of serialised ('pickled') object lists. This has reduced run time considerably, and now offers all the information needed for outputs. There are three primary classes used to hold information at different stages of analysis (see 'ORG\_classes.py' in program files). Firstly, there is an 'Article\_company' class which is used to hold all mentions of a company within a single article. Secondly, an 'Article\_obj' is used to hold all the information for an article and includes a list of 'Article\_company' objects. Lastly, a class called 'Full\_company' is used to merge information from all articles into single objects for each company, which can then be used for displaying outputs. In the UI section of the program, it is a list of 'Full\_company' objects which forms the basis for all data displayed on each of these screens (Appendix 2).

## Article level objects

Each article text has a list of company entities identified, using the NER mechanism, which informs the creation of a list of 'Article\_company' objects. The attributes for the 'Article\_company' class are 'company\_name', 'present\_name', 'sentence\_list', 'art\_vader' (see 'ORG\_classes.py' in program files). Once a company has been identified and verified, an object is created using a constructor; then 'Spacy' is used to separate the text into sentences, and all sentences in which that entity is mentioned are added to a list of sentences. VADER scores for each of these sentences are calculated and saved as a list of tuples - (sentence, VADER compound score) to the 'sentence\_list' attribute. Finally, the mean sentiment score for all sentences in an article is saved with the attribute 'art\_vader', which allows for the calculation of averages later in the program. Each 'Article\_obj' will have a list of these objects once parsed. The 'Article\_obj' class holds the following self-explanatory attributes – 'title', 'URL', 'company\_list'. It is this object which is serialised so that subsequent analysis is faster if the article has already been parsed. A directory titled 'Article\_objects' holds these 'pickled' objects, and it is here that checks are made against incoming .txt files to determine whether a new object needs to be created or not (see 'Populate\_list\_functions.py', line: 47 in program files).

A key design consideration has been in how to deal with new articles when the program is run again. If a new set of articles appear on a website, then titles for these articles are compared with existing .txt files in the web crawler function, and only the articles which are not already present are parsed (see 'Motley\_functions.py', line: 38 in program files).

```
count = 0
for url in URLlist: # Iterates over all URLs in list for writing to .txt
    title = URLlist[count][0]
    title = re.sub('[^A-Za-z0-9]+', '_', title) # Strip special characters for saving file name
    if (os.path.isfile(f'\\TextFiles\\{title}.txt')):
        #Checks to see if file already exists before sending requests
        # print(f"{title}")
        # print("FOUND")
        count +=1
        continue
    else: # If file does not already exist
        html = URLlist[count][1]
```

*Figure 3: Code snippet showing the process used to determine existing texts*

This avoids having to run the web crawler in its entirety every time the function runs and therefore reduces the time it takes to analyse these blogs. In the second phase, data analysis – the analysis function

checks to see if there are any new .txt files which do not match the list of serialised items in the 'Article\_objects' directory (see 'Populate\_list\_functions.py', line 47 in program files). Again, analysis will only be carried out for new items, which saves runtime. In cases where both directories do not contain any matches, there is a considerably longer wait time, while these folders need to be updated. This is something that has been difficult to avoid using Python, which is slow due to dynamic typing. It is possible that my own design for this process could have been made more efficient by using multithreading, though this is something I only learned about when creating the UI elements of the program. Another area where speeds could have been improved would be by using a different method for data collection, such as an RSS feed or an online API.

## Consolidation

The 'Full\_company' class attributes are: 'company\_name', 'sentence\_list', 'URL\_list', 'micro\_average', 'macro\_average', and 'temp\_average'. It is this list which persists between articles and collects data for the entire sample. The 'Full\_company' list merges all references to a single company (removing duplicates), creates a full list of all sentences mentioned, and holds a list of the article URLs. The 'Full\_merge' function has some complex logic which checks to see if the company already features in the full list, and if it does not, it creates a new object to append to the list (see 'Populate\_list\_functions.py', line: 121 in program files). If a company does feature in this list already, then information is added for that specific entry. Once a list of partially completed 'Full\_company' objects has been populated with a complete set of sentences, VADER sentiment analysis can be performed for both averages. The 'macro\_average' attribute is calculated using the sum of all valence scores from the 'sentence\_list' tuples, divided by the number of sentences present across all articles. The 'micro\_average' attribute is calculated by using the sum of all article level valence scores, divided by the number of articles that company appears in. Consolidation of all company objects is complete after the 'Full\_merge' function.

Some issues arose while trying to work with lists of objects, which themselves have list attributes; using lists as class attributes in Python is different to the C++ arrays that I had been familiar with. A full day was spent trying to get a list attribute to work as an instance attribute, rather than as a class wide – static variable. Eventually I found that Python list attributes default to class wide attributes unless they are re-assigned with a specific syntax (see 'Populate\_list\_functions.py', line 109 in program files). Another issue was in differences between the quoted company reference and the matched standardised company title (i.e spelling differences, partial names etc.). This was solved by saving both the quoted reference, and the title that FuzzyWuzzy matched it with as a tuple for further comparisons. Doing this allowed for differences to be accounted for when merging. Once a fully merged list has been produced, the UI elements can interact with this and display them in a 'recycle view' layout.

# User Interface

It was useful to consider the final UI elements and what sort of functionality I would like to see from the end-product. This helped to decide what types of information were needed at each phase of the program. After some thought, four key features for the UI output were identified.

- A list of companies with a compound sentiment score for all sentences where that company is mentioned (both ‘Micro’, and ‘Macro’ scores).
- An option to view all sentences that mention a company along with their valence score.
- The title of articles in which the company is mentioned and the option to see these articles, with links for visiting each webpage.
- Option to visualise this data in a Bar chart.

As discussed in the previous section, initial attempts to create the interface used the Flask web framework, but a decision was made to proceed using the Kivy UI module due to a lack of understanding with JavaScript. While I have enjoyed this process, there have been some major challenges in trying to create a UI to my own standards.

The design philosophy behind the FinSpotter Graphical User Interface (GUI) is, for the most part, utilitarian. The primary focus for this program has been to complete the objectives of the project in the back end. The front end is designed with basic functionality and practicality in mind. Despite this, there are some aspects where considerable effort has been made to display information in a user-friendly format. The first foundational widget needed for any app with multiple screens is the ‘WindowManager’. This allows for screens to be added and directs navigation with buttons between screens. Additionally, it allows for passing information between screens if this is required. The first screen of ‘FinSpotter’ displays a message explaining the reasons and goals behind the creation of the program. This is a very straightforward design, with a text label, and two buttons for navigation (Appendix 3). The second screen provides information on how the program operates, along with the list of blogs where the sample articles have been sourced (Appendix 4). From here there is a button titled ‘Begin Analysis’, which triggers the ‘execute\_func()’ function (see ‘main.py’, line 91 in program files), and proceeds to the loading screen.

```
self.set_screen()

def executeFunc(self):
    self.manager.current = 'Loading' # Here is where I have tried to move to loading screen while func runs
    t1 = threading.Thread(target=self.startAnalysis) # Here is where I have tried to thread the function
    t1.start()

@mainthread
def set_screen(self):
    self.manager.current = 'output'
```

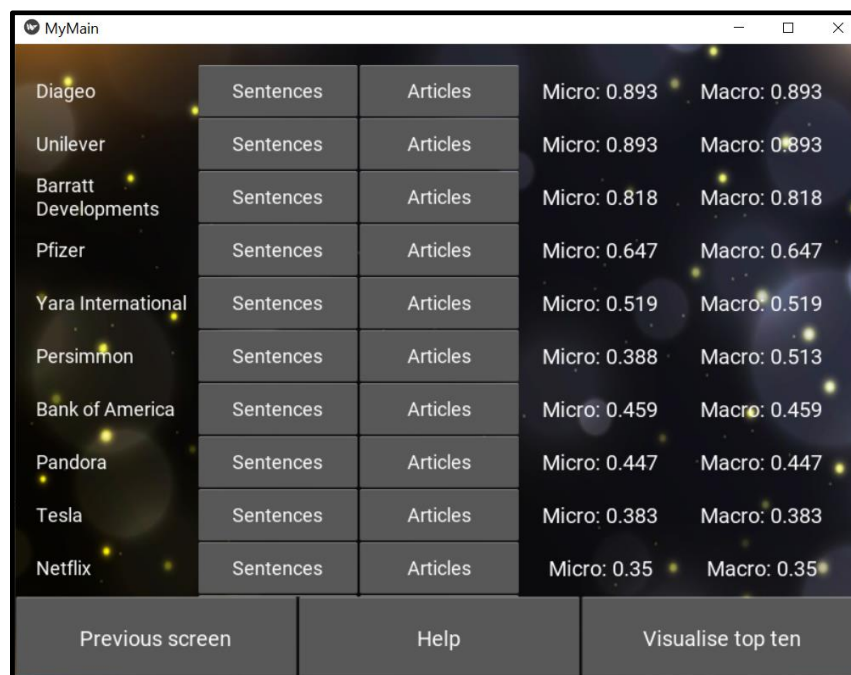
Figure 4: The code which runs the backend functions in an alternative thread

Once the back-end functions are complete, the ‘OutputWindow’ screen is brought into focus. This screen holds the most complicated components in the user interface and has taken the most amount of time to implement. From this screen, there are options to visualise this data in bar charts (top ten companies and bottom ten companies).



## The Output Screen

This screen is where most of the time has been spent in trying to display data in a user-friendly format (Appendix 5). To display lists of any kind in Kivy, a widget called recycle view is required. This is an updated version of the scroll view modules which are popular in many mobile development kits. Recycle view displays rows, which can be custom widgets with extra functionality as in this case. Each row of the initial recycle view represents a ‘Full\_company’ object, and attributes such as company name, and micro/macro valence scores are displayed at the first screen. Each row has two buttons with – ‘Sentences’ and ‘Articles’. These buttons bring up a popup screen, which itself has a small nested recycle view widget to display information. The ‘Sentences’ popup window shows a list of all the sentences parsed, and the individual score for each of these (Appendix 6). This is so that the user can exercise some manual judgement over the accuracy of VADER scores. The ‘Articles’ popup shows a list of all the articles in which that company has been matched (Appendix 7). The title of the article is given, along with a link so that the user can visit that article if they wish. The idea behind both additional information buttons is that this can help the user determine for themselves which companies look promising, without having to rely too heavily on the VADER score. A help button is provided which provides an explanation behind the information displayed on this screen (Appendix 8).



Company	Sentences	Articles	Micro	Macro
Diageo			0.893	0.893
Unilever			0.893	0.893
Barratt Developments			0.818	0.818
Pfizer			0.647	0.647
Yara International			0.519	0.519
Persimmon			0.388	0.513
Bank of America			0.459	0.459
Pandora			0.447	0.447
Tesla			0.383	0.383
Netflix			0.35	0.35

Previous screen	Help	Visualise top ten
-----------------	------	-------------------

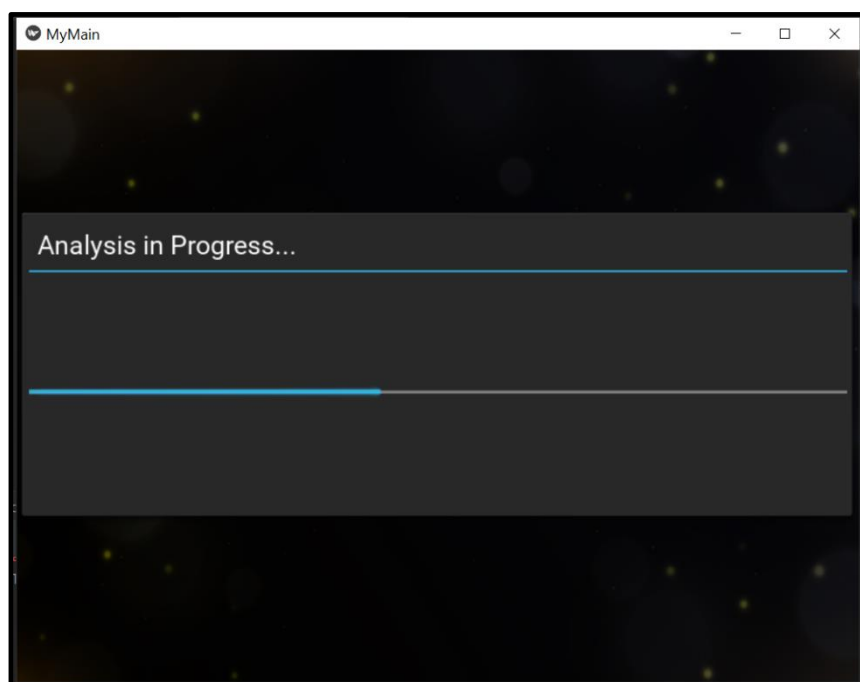
Figure 5: The main output window from FinSpotter

The Kivy design module is easy to learn the basics with, though difficult to master past that point. In realising my vision for a UI, things did get quite complex. After establishing the initial needs for the UI early, I decided to take a practical approach to displaying these features. In some stages I sought help from the official Kivy discord channel to ask for some support. Thankfully, they were incredibly helpful and tolerant of my inexperience. The main aspects where they were able to help me were in passing data between the various widgets. Getting data passed between widgets proved to be no small task, the solution has involved using a system of Kivy properties (Kivy, N/A). A second issue has been in passing data from the ‘All\_blogs()’ function to the main recycle view at the appropriate time. Initially this was included in the constructor for the ‘RV’ class, though this meant that the function was triggered

immediately on starting the program. This caused the program to pause on a white screen for some time before displaying the welcome page, which was not very user friendly. Now the 'All\_blogs()' function is called with the same button which transitions between the second page and the loading screen.

## Loading Screen

After solving the issue where backend functions were triggered as the program started, a similar problem arose between the switching of screens. As the user pressed 'Analyse blogs' there was a considerable 30-40 second pause while everything processed, and the data loaded into the 'output' screen. This proved to be a major challenge to solve, as the Kivy UI framework is run exclusively on the main thread (based on an OpenGL foundation), which means that processes generally must run sequentially unless using a multithreaded approach. Roughly ten hours were spent trying to solve this issue. I began by studying how to use a multithreaded approach in Python. After gaining some confidence with this, I moved to looking at examples of multithreaded processes using Kivy. Finally, I was able to create an alternative thread to run the backend functions (see 'main.py', line: 93 in program files). This solved the first part of the issue. Next, I wanted to display a loading screen to the user while these functions executed (Appendix 9). This initially seemed straightforward and changing to the loading screen was fine. Changing away from the loading screen once the backend functions were complete was a greater challenge. This involved more hours of learning how the screen manager methods worked and how best to implement them for my use case (Kivy, N/A).



*Figure 6: Loading screen for FinSpotter*

## Visualisation screens

One of the features identified at the start of the UI design phase was to include some visualisation functions for the program. One issue with this is that there were many companies identified in reports once the 100-article sample size was implemented. This meant that visualising all companies mentioned would be impractical. Instead, now FinSpotter offers two forms of bar graph, one with the top ten companies (Appendix 10), and one for the bottom ten companies (Appendix 11) ranked by sentiment score. The 'Matplotlib' library has been used to produce these charts, and it saves these as images before being displayed on their respective screens. The steps used to produce these images and display them to the user were straight forward, apart from one major flaw. The program would not update the saved image at runtime, and instead it would display the saved image from the previous time the program had been run. This was because the constructors used to create these Kivy objects would source the image as the program began and would not update once the Matplotlib functions had saved an up to date image. Again, because of my inexperience with Kivy, this involved a long period of trial and error before I was able to address this issue. Eventually this was completed by using an update function in a custom image widget (see 'main.py', line: 269 in program files).

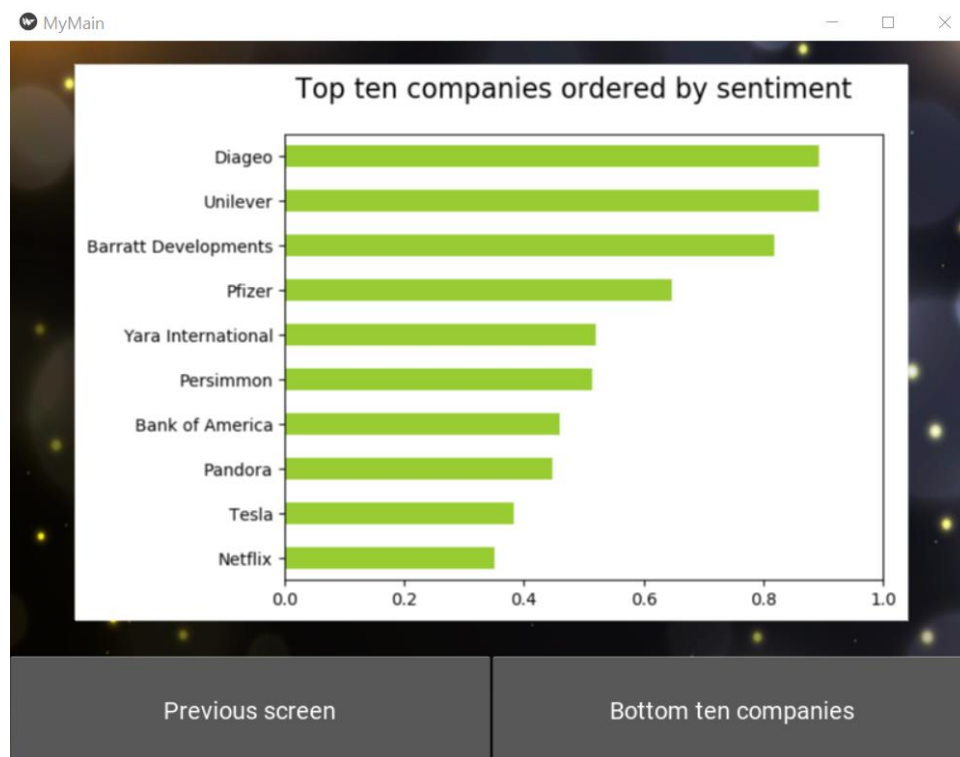


Figure 7: Bar chart showing top ten companies ranked by sentiment

## UI conclusions

As discussed, the front-end implementation for FinSpotter is designed with practicality in mind. There are areas where this could have been improved if UI design had been a greater point of focus for the project. In its current form, the FinSpotter UI is up to the minimum standard that I personally would be happy to present. Some areas for potential improvement would be in the design of buttons, the layout of the recycle view rows, as well as a more interesting animation on the loading screen. In general, the UI does perform all the actions expected of it and so can be considered a success.

# Testing accuracy

## Overview

To test the accuracy of VADER sentiment analysis, a survey has been created to collect data from human participants. This has been completed by using the JISC survey system (JISC, 2020). The design of the survey is relatively straight forward: participants are asked to rate a series of 48 sample sentences, which have come from the blogs that FinSpotter analyses. The categories for rating are 'Positive', 'Negative', and 'Neutral'. An example sentence would be 'Tesco has made major gains', and participants are instructed to rate the sentiment of this sentence, in relation to the company mentioned. There have been some unforeseen issues with the collection of this data. Participants were sampled using an 'opportunity' sampling method, meaning that none of them had any experience with finance or financial terminology. This meant that the results for a few of the questions have very mixed results and could have ambiguous interpretations. Another issue has been in FinSpotter's ability to analyse sentiment reliably. Often this issue was due to the lack of robust attention mechanism, and an over reliance on sentences in which a company was mentioned. This section will provide a discussion on these issues and some suggestions for an improved experimental design are offered.

## Experiment design

This survey has been designed in a very simple way. Initially there is a screen which describes the objectives of this survey, and shows the participant briefing information, along with the participant consent form. There are options here to confirm that they are happy to participate, so that this element is easier for the participant than filling out and sending back a participant consent form. Following this is another screen re-iterating instruction for what is expected of the participant. The next page from this is where the questions begin. There are 48 questions in total, asking participants to rate a series of sentences as either 'Positive', 'Negative', or 'Neutral'. Each sentence has been taken from the output of FinsPotter so that sentiment scores are already present for comparisons. When choosing the sentences to include, I tried to include polarised sentences so that participants would have an easier time deciding. However, this did mean that there was some selection bias being introduced into this element of the process. The idea behind this design has been so that the data collected from these can be measured against the output sentiment scores from VADER, and an assessment of accuracy carried out.

## Participant sampling

There have been some major issues with the quality of responses seen from the sample group. 'Opportunity' sampling has been the method used for collecting responses, meaning that these are general, uninformed responses that have been measured. This has posed problems due to the specialised terminology used frequently in financial news. Participants have reported that they were unsure of whether a sentiment was positive or negative, because they lack any understanding of finance. This has resulted in an increased number of neutral responses than was anticipated. Additionally, some questions where there were two companies mentioned were difficult for participants to classify in relation to one single company, which could be considered a flaw in experiment design. If this survey were to be run again, I would look to sample a more informed group of participants, either finance students, or from finance forums online. The number of participants may also be of some issue, as I have collected only 10 responses. In one question, there is a huge disparity in the intended meaning of a sentence, and the interpreted sentiment recorded by participants. The sentence "Reckitt Benckiser's share price has also

gone gangbusters in 2020” is intended as a positive statement when read in the context of the article. Contrary to this, only 1 respondent rated this positive, 2 rated this neutral, and 4 rated this negative. Despite these issues, the results will be used to measure the accuracy of the FinSpotter VADER element in lieu of an alternative.

**30** 'Reckitt Benckiser's share price has also gone gangbusters in 2020.'

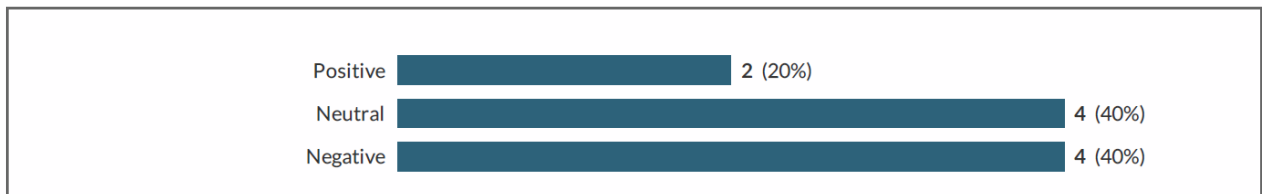


Figure 8: Odd interpretation of gangbusters question

## Problems with VADER

In addition to the issues presented within the human participation element of this survey, there have been issues with the results obtained from VADER scoring. Complex sentences with more than one company, and in which one company may be praised in comparisons with another company, are rated with a blanket score for both companies. This is an issue rooted in the ‘bag of words’ method used by VADER and may also be a result of the overly simplistic attention mechanism employed for this program. Another problem seems to be present for analysing sentences with complex negation rules, VADER does not handle this reliably. A common observation in most Natural Language Financial Forecasting research has been that terminology is the most important factor in producing accurate results. This is another weakness of the model employed with FinSpotter; there is a strong reliance on a generic sentiment classifier, where a domain specific tool would have been more appropriate.

## Results and Analysis

The JISC survey system provides a very user-friendly interface for analysing the results of a survey. A copy of the aggregated responses will be included in a PDF document in the additional files folder for this project. Despite the flaws discussed with this data, this section will provide a comparison of the results obtained by the survey – and the results produced by FinSpotter. These comparisons have been compiled into a table which can be found in the appendices (Appendix 12). Due to the sizing of sentences, I have opted to only include the question number in the comparison charts. These should be used to reference the data available in the survey results PDF. The data shown in the results table indicates a strong sense of uncertainty among participants. For the following discussion, VADER scores of more than 0.1 are positive, and scores less than -0.1 are termed negative, with those in between being neutral. There are several questions which show very different responses, and so these will be discussed, and some speculative reasons offered.

In 34/48 questions, participant answers matched with VADER sentiment classification. On reflection, these results may have been inflated by the fact that sentences were specifically chosen to be polarised to tailor to the sample audience. This will have worked in favour of VADER’s ‘bag of words’ approach,

which calculates sentiment using the polarity scores of individual words. In practice, and with a wider sample, it is highly possible that the ratio of correct answers would be significantly less. There are a lot of neutral responses which seem to indicate that there was some indecision in responses. Indeed, this was confirmed with some participants contacting me to mention that they do not understand financial terminology and found it difficult to decide on responses. Neutral responses can be seen throughout. Correct VADER evaluations seemed to correlate with the intensity of polarisation. This was also the case with strongly neutral sentences. Incorrect VADER scores seemed to follow in cases where there were two or more companies discussed, where complex negation was used, or in cases where the language used was ambiguous to participants.

In the case of question 17, VADER rated this as positive, whereas participants viewed it as neutral. It is not immediately clear why this might be, but one possible reason may be in the word ‘likes’ or ‘hotels’ being perceived as positive with the ‘bag of words’ approach.

**17** 'These include the likes of easyJet, IAG, CCL, TUI, and the InterContinental Hotels Group.'

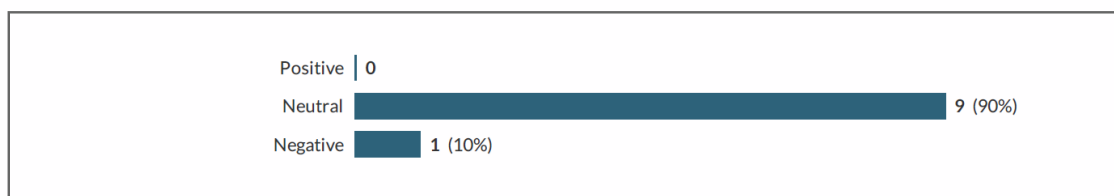


Figure 9: Question 17 from the survey PDF

Question 13 shows an example of VADER classing a sentence as negative, where participants classed this sentence as positive. Again, the methods used by VADER are likely to have been influenced by the polarity for the word ‘crisis’. Outcomes like this could possibly have benefitted from an intra-sentence attention mechanism, where words that appear closer in proximity to the mention of a company title would be weighted heavier. Another possible fix would be to extend the default lexicon of VADER, and to manually add terms like ‘financial crisis’ with a custom valence score. This would be quite a time-consuming process but could offer VADER a better understanding of terminology.

**13** 'Diageo's share price has increased by more than 300% since the financial crisis of 2009.'

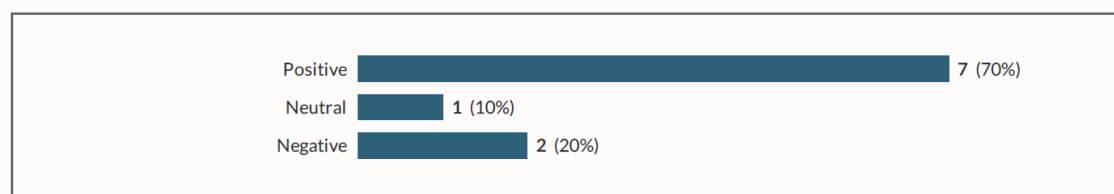


Figure 10: Question 13 from the survey PDF

In question 21, it is not immediately obvious why VADER has classed this as positive, but participants did not view this the same way. Note that answers are spread, showing that there is some ambiguity here.

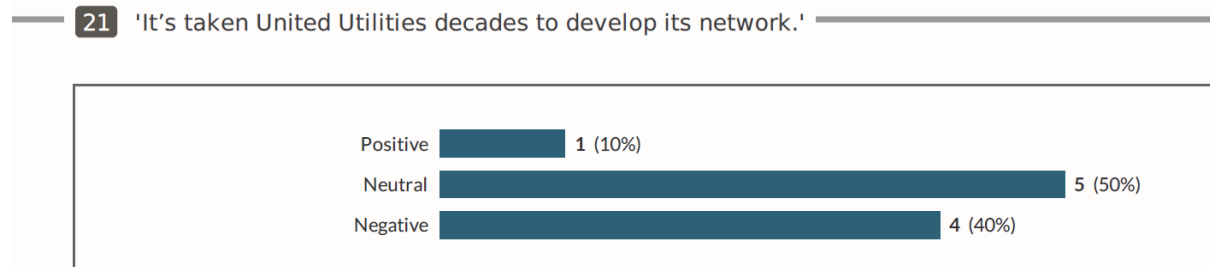


Figure 11: Question 21 from the survey PDF

A final example of different outcomes in comparisons can be seen with question 18. Participants clearly saw this as a negative statement, where VADER inexplicably classed this as positive.

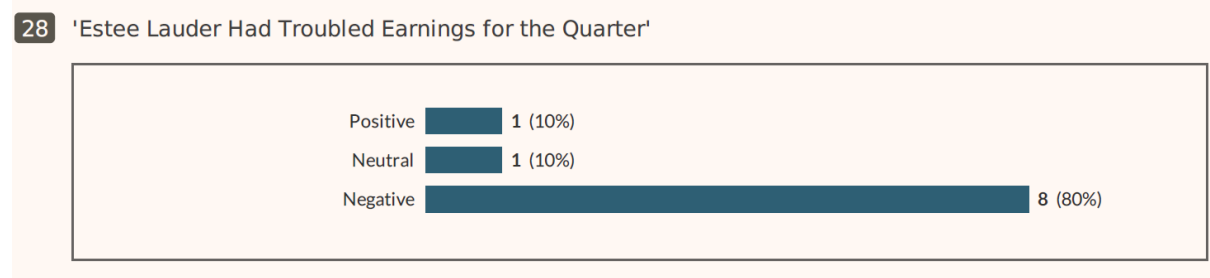


Figure 12: Question 28 from the survey results PDF

### Analysis conclusion

There are three key areas where this experiment design could have been improved. First, by taking care to find an informed sample group which can confidently determine sentiment from financial texts. Second, by using a set of sentences which have not been chosen for their polarised content; and lastly, by increasing the range of available categories to measure the strength of the sentiment contained in a sentence. In hindsight a pre-completed annotated data set, which had been informed by experts would have been a better way to measure the accuracy of FinSpotter. After some analysis of the results from this survey, some areas for improvement in the VADER sentiment mechanism are clear. A larger, domain specific lexicon would improve accuracy in one sense, and a more complex attention mechanism (or weighting system) would help to emphasise the correct sections of a sentence in scoring.

# Conclusions

In reflecting on the process of creating FinSpotter, this has largely been a positive experience. The program meets all the criteria which I had set out to accomplish 12 weeks ago. Areas for potential improvement are clear and have been documented throughout this report. The FinSpotter program successfully parses 100 articles from various financial news blogs. It has been reliable in its ability to distinguish company titles from text, and to create a series of class objects to house data for these companies. VADER has been an expedient solution to producing sentiment scores, but the sentiment analysis portion of this program could have benefitted significantly with a more complex process. Lastly, the UI functions adequately and displays data in an easily comprehensible format. Overall, I would consider this project a success, in that it has met the objectives set out in June. There are several improvements which could have taken place with a longer time allowance to improve the accuracy of FinSpotter. The area which could have the most impact on accuracy would be in creating an implementation of FinSpotter which uses a BERT or GPT3 sentiment classifier. These can be tailored to account for domain specific terminology. Out with a machine learning approach, VADER would have benefitted from an expanded lexicon model, to account for words in a financial context better.

To improve this project, a more robust experiment design and implementation would have helped to produce reliable results for measurements. This is a significant weakness, though has provided some metric with which to produce an accuracy score (34/48 correct assessments). It should be noted that there is a selection bias, and a participant bias to account for in this assessment score. Another area where this project may have fallen short, would be in the attention mechanism used to identify relevant sentences. FinSpotter uses an overly simplistic method, whereby only the sentences in which a company is present are passed on for sentiment analysis. This excludes a lot of context, and relevant sentences. One way to rectify this would be to consider both the sentence in which a company is mentioned, and the sentence immediately following from this, which would broadly assume some level of relevance based on proximity. Lastly, there are some possible improvements to the speed at which FinSpotter carries out analysis. I only discovered how to use multithreading when completing the UI sections of the program. Had I known this technique earlier in development – it is possible that I could have created a design using concurrent execution, dramatically decreasing the time users must wait for results. In a similar sense, the time it takes to collect data from these blogs could have been made much faster if using an alternative approach. Using an API or RSS feed to collect text data would have been a faster than using BeautifulSoup to parse webpages.

To summarise, here are the main areas for impactful improvements:

- An implementation of FinSpotter with BERT or GPT-3 sentiment classifiers
- A more robust experiment design and implementation.
- A larger sample set of articles, with a faster data collection method (APIs or RSS feeds)
- Create a more attentive attention mechanism for classifying sentences related to companies
- A more extensive lexicon (for VADER) to deal with specialised terminology

With all constraints considered, I can say I am pleased with the final product.



# References

Araci, D. (2019). *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models*. Master's Thesis – University of Amsterdam. (Online) Available at: [https://www.researchgate.net/publication/324957692\\_Financial\\_Sentiment\\_Lexicon\\_Analysis](https://www.researchgate.net/publication/324957692_Financial_Sentiment_Lexicon_Analysis) [Accessed 11/08/2020]

Babar, N. 2018, *The Levenshtein Distance Algorithm*. Big Data Zone (online). Available at: <https://dzone.com/articles/the-levenshtein-algorithm-1> [Accessed 04/09/2020]

Carlile W., Nishant G., Zixuan K., and Ng, V. (2018). *Give me more feedback: Annotating argument persuasiveness and related attributes in student essays*. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne (Online). Available at: <https://www.aclweb.org/anthology/P18-1058/> [Accessed 11/08/2020]

Feldman, R. & Rosenfeld, B. & Bar-Haim, R. & Fresko, M. (2011). *The Stock Sonar - Sentiment Analysis of Stocks Based on a Hybrid Approach*. Proceedings of the National Conference on Artificial Intelligence. 2. (Online) Available at: [https://www.researchgate.net/publication/221016483\\_The\\_Stock\\_Sonar\\_-\\_Sentiment\\_Analysis\\_of\\_Stocks\\_Based\\_on\\_a\\_Hybrid\\_Approach](https://www.researchgate.net/publication/221016483_The_Stock_Sonar_-_Sentiment_Analysis_of_Stocks_Based_on_a_Hybrid_Approach) [Accessed 11/08/2020]

Horev, R. 2018, *BERT Explained: State of the art language model for NLP*. Towards Data Science (online). Available at: <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270> [Accessed 04/09/2020]

JISC, 2020. *Online Surveys* (Online). Available at: <https://www.onlinesurveys.ac.uk/> [Accessed 04/09/2020]

Joshi, P. (2020). *spaCy Tutorial to Learn and Master Natural Language Processing (NLP)*. Available at: <https://www.analyticsvidhya.com/blog/2020/03/spacy-tutorial-learn-natural-language-processing/> [Accessed 04/09/2020]

Kivy, N/A. *Properties*. Kivy Blog (Online). Available at: <https://kivy.org/doc/stable/api-kivy.properties.html> [Accessed 04/09/2020]

Kivy, N/A. *Screen Manager*. Kivy Blog (Online). Available at: <https://kivy.org/doc/stable/api-kivy.ui.screenmanager.html> [Accessed 04/09/2020]

Lawrence J. & Reed, C. (2019). *Argument Mining: A Survey*. Association for Computational Linguistics. (Online) Available at: [https://www.mitpressjournals.org/doi/full/10.1162/coli\\_a\\_00364](https://www.mitpressjournals.org/doi/full/10.1162/coli_a_00364) [Accessed 11/08/2020]

Lukin S., Anand P., Walker M., Whittaker S. (2017). *Argument Strength is in the Eye of the Beholder: Audience Effects in Persuasion*. European Chapter of the Association for Computational Linguistics (EACL 2017). (Online) Available at: <https://arxiv.org/abs/1708.09085> [Accessed 11/08/2020]

Ortiz, A. & Fernández-Cruz, J. (2015). *Identifying Polarity in Financial Texts for Sentiment Analysis: A Corpus-based Approach*. Procedia - Social and Behavioral Sciences. 198. 330-338. 10.1016/j.sbspro.2015.07.451. (Online) Available at: [https://www.researchgate.net/publication/282553639\\_Identifying\\_Polarity\\_in\\_Financial\\_Texts\\_for\\_Sentiment\\_Analysis\\_A\\_Corpus-based\\_Approach](https://www.researchgate.net/publication/282553639_Identifying_Polarity_in_Financial_Texts_for_Sentiment_Analysis_A_Corpus-based_Approach) [Accessed 11/08/2020]

Portilla, J. (2020). *2020 Complete Python Bootcamp: From Zero to Hero in Python*. Udemy Course (Online). Available at: <https://www.udemy.com/share/101W8QBkAad11RRXQ=/> [Accessed 11/08/2020]

Sohangir, S. & Petty, N. & Wang, D. (2018). *Financial Sentiment Lexicon Analysis*. 286-289. 10.1109/ICSC.2018.00052. (Online) Available at: [https://www.researchgate.net/publication/324957692\\_Financial\\_Sentiment\\_Lexicon\\_Analysis](https://www.researchgate.net/publication/324957692_Financial_Sentiment_Lexicon_Analysis) [Accessed 11/08/2020]

Somasundaran S. & Wiebe J. (2010). *Recognizing Stances in Ideological On-Line Debates*. Workshop on Computational Approaches to Analysis and Generation of Emotion in Text. Pages 116-124. (Online) Available at: <https://www.aclweb.org/anthology/W10-0214.pdf> [Accessed 11/08/2020]

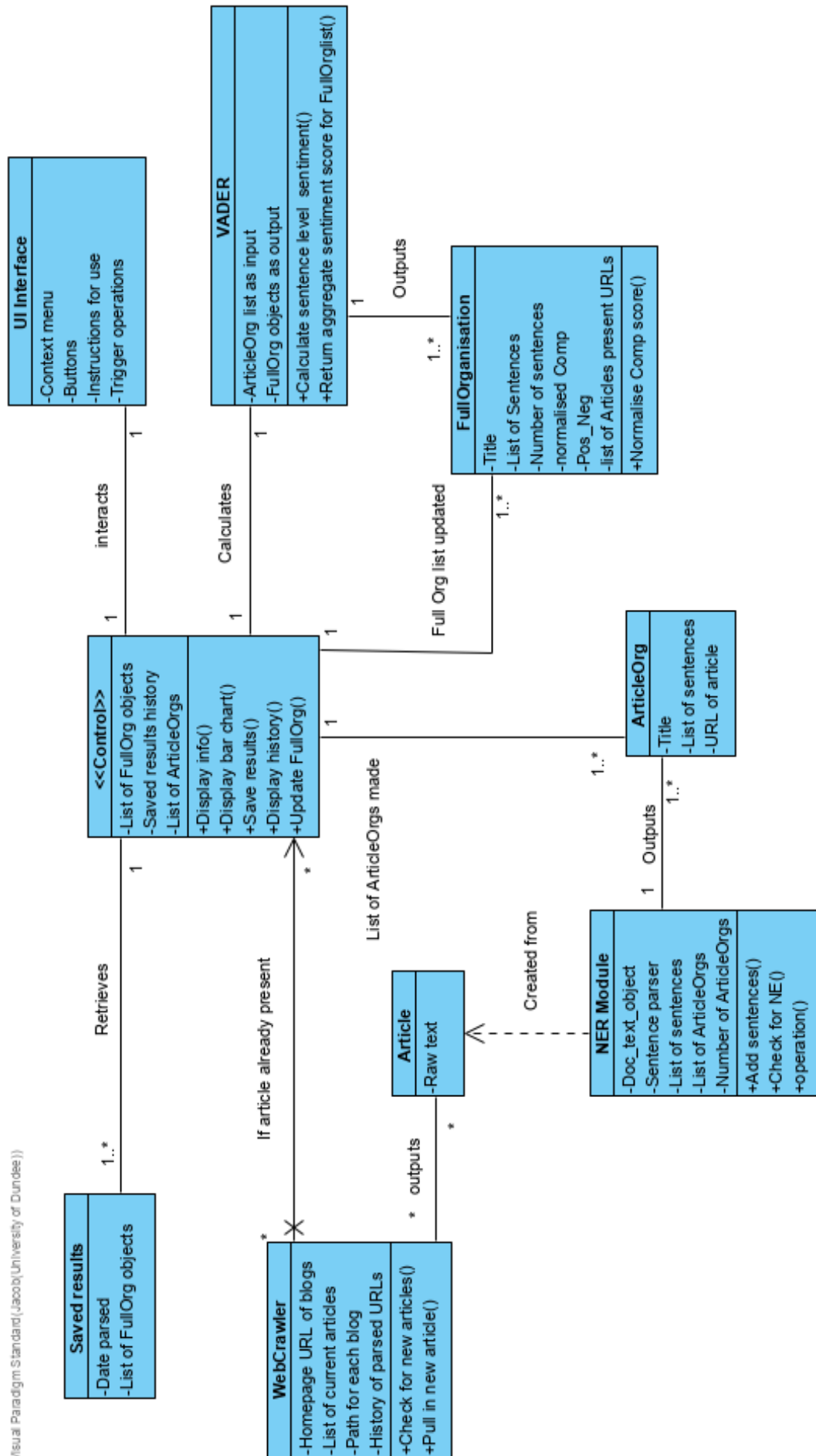
Wyner, A, Schneider, J, Atkinson, K & Bench-Capon, T. (2012). *Semi-automated argumentative analysis of online product reviews*. in Computational Models of Argument - Proceedings of COMMA 2012. 1 edn, Frontiers in Artificial Intelligence and Applications, no. 1, vol. 245, IOS Press, pp. 43-50. <https://doi.org/10.3233/978-1-61499-111-3-43> [Accessed 11/08/2020]

Xing, F.Z., Cambria, E. & Welsch, R.E. (2018). *Natural language based financial forecasting: a survey*. Artif Intell Rev 50, 49–73. Available at: <https://doi.org/10.1007/s10462-017-9588-9> [Accessed 11/08/2020]

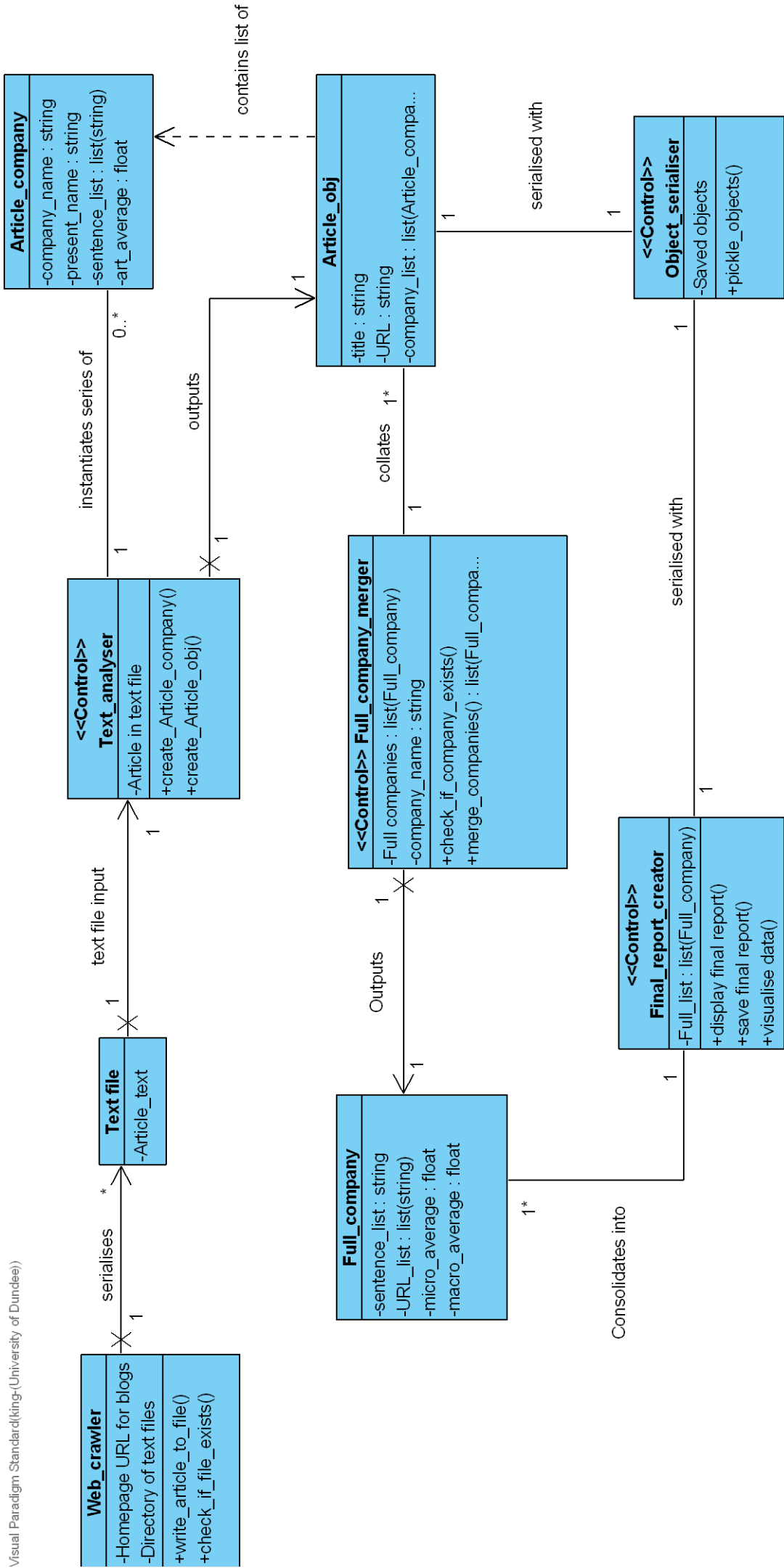
# Appendices

## Appendix 1:

First iteration E-R diagram

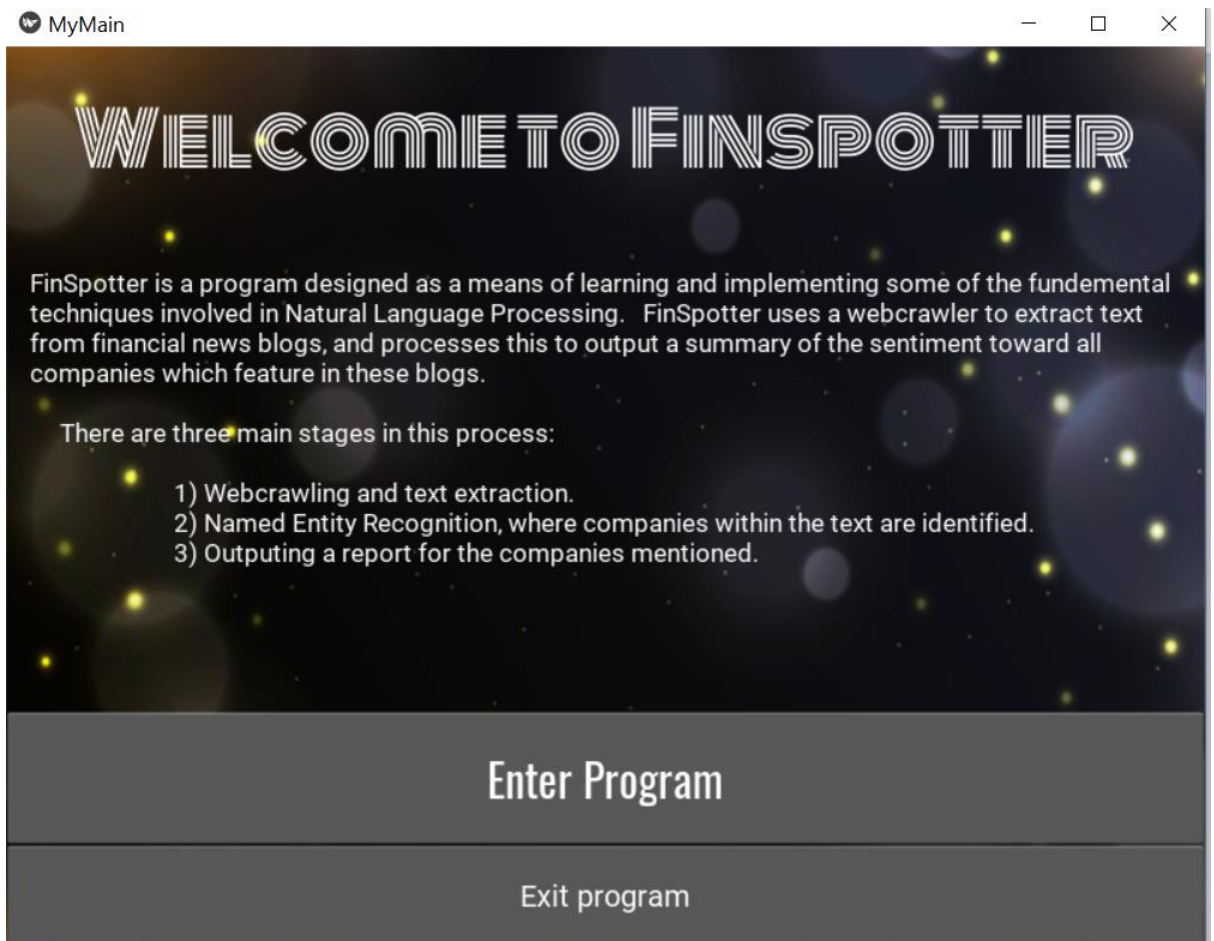


Appendix 2:  
Third iteration of  
E-R diagram



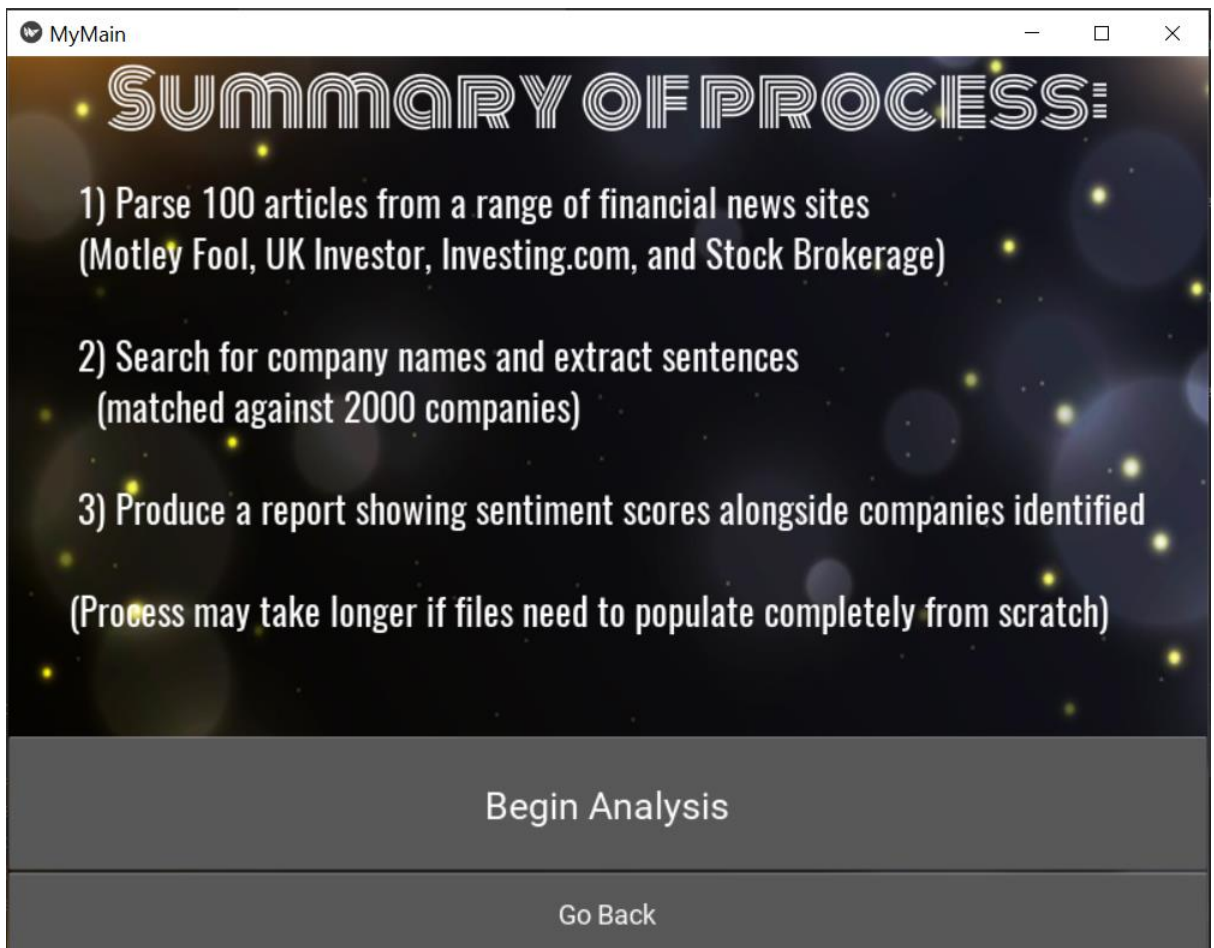
Appendix 3:

The welcome screen for FinSpotter



Appendix 4:

The analysis screen for FinSpotter



#### Appendix 5:

The output window for FinSpotter

MyMain				
Diageo	Sentences	Articles	Micro: 0.893	Macro: 0.893
Unilever	Sentences	Articles	Micro: 0.893	Macro: 0.893
Barratt Developments	Sentences	Articles	Micro: 0.818	Macro: 0.818
Pfizer	Sentences	Articles	Micro: 0.647	Macro: 0.647
Yara International	Sentences	Articles	Micro: 0.519	Macro: 0.519
Persimmon	Sentences	Articles	Micro: 0.388	Macro: 0.513
Bank of America	Sentences	Articles	Micro: 0.459	Macro: 0.459
Pandora	Sentences	Articles	Micro: 0.447	Macro: 0.447
Tesla	Sentences	Articles	Micro: 0.383	Macro: 0.383
Netflix	Sentences	Articles	Micro: 0.35	Macro: 0.35
Previous screen		Help		Visualise top ten

#### Appendix 6:

The sentences popup window

MyMain

Diageo	Sentences	Articles	Micro: 0.472	Macro: 0.337
Alphabet	Sentences	Articles	Micro: 0.334	Macro: 0.334

Diageo

Stocks like Unilever and Diageo, world leaders in their respective fields, maybe the best long-term for a Stocks and Shares ISA. 0.8934

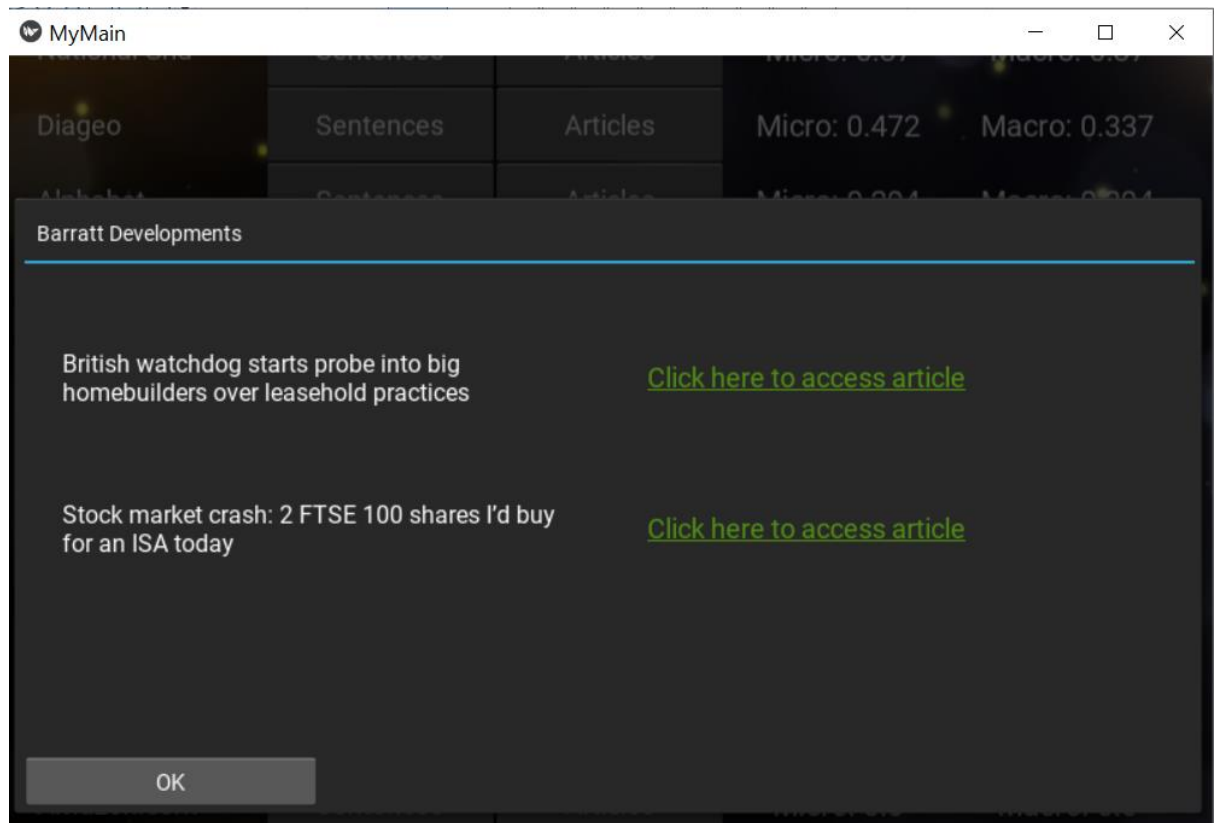
First up, alcoholic beverages giant Diageo (LSE: DGE) which owns a number of top brands including Johnnie Walker and Tanqueray. 0.2732

Prior to Covid-19, Diageo shares were flying. 0.296

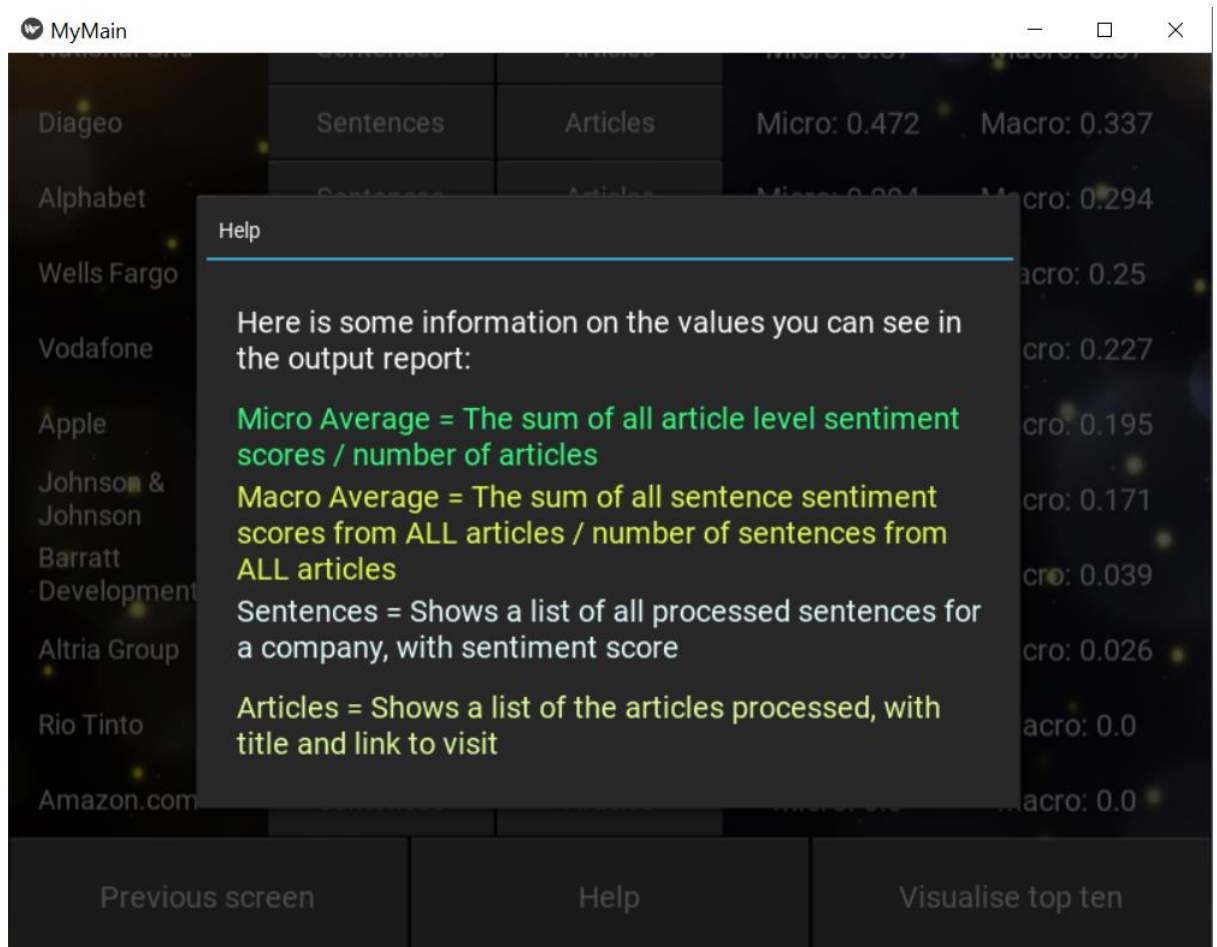
OK



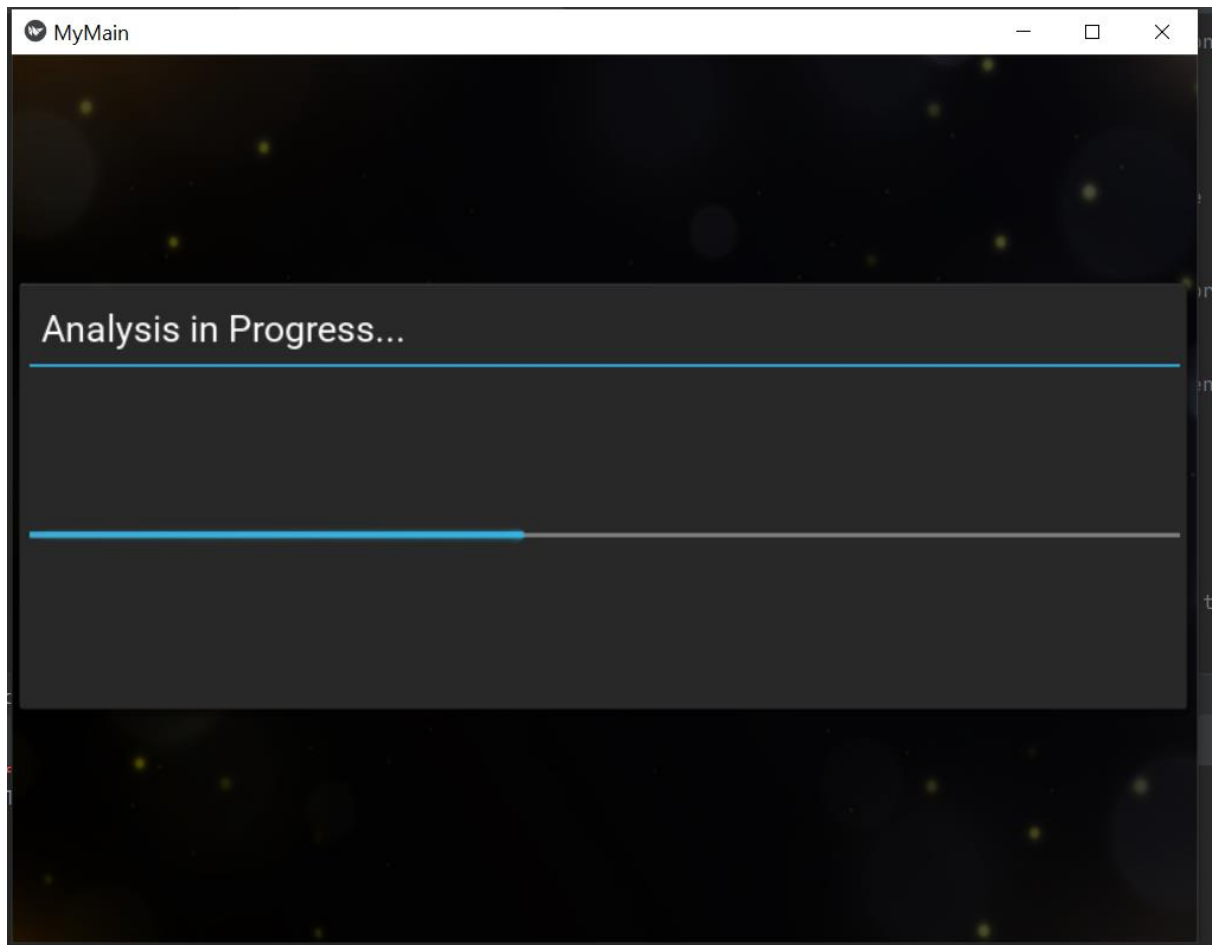
Appendix 7:  
The Articles  
popup window



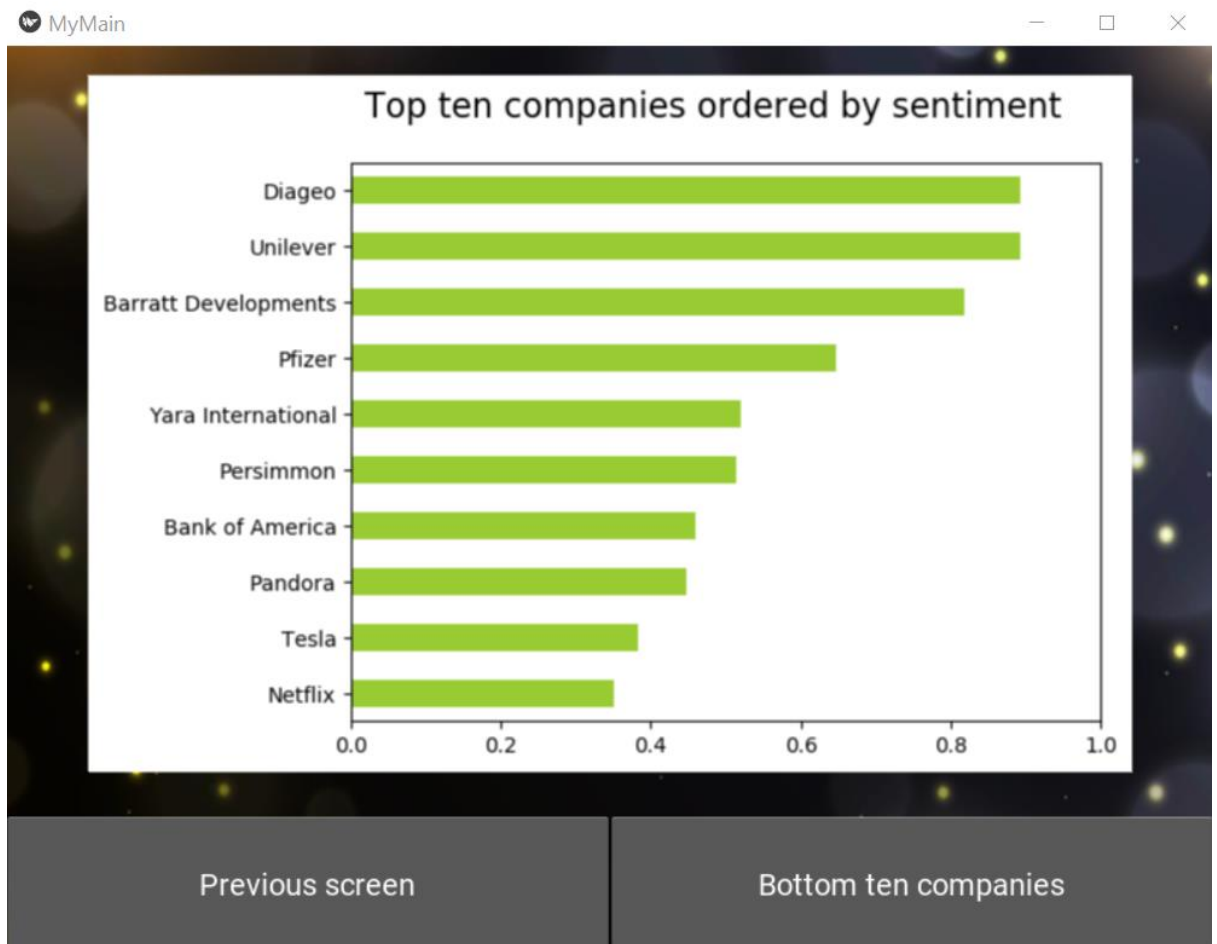
Appendix 8:  
The help  
popup window



Appendix 9:  
Loading screen  
for FinSpotter



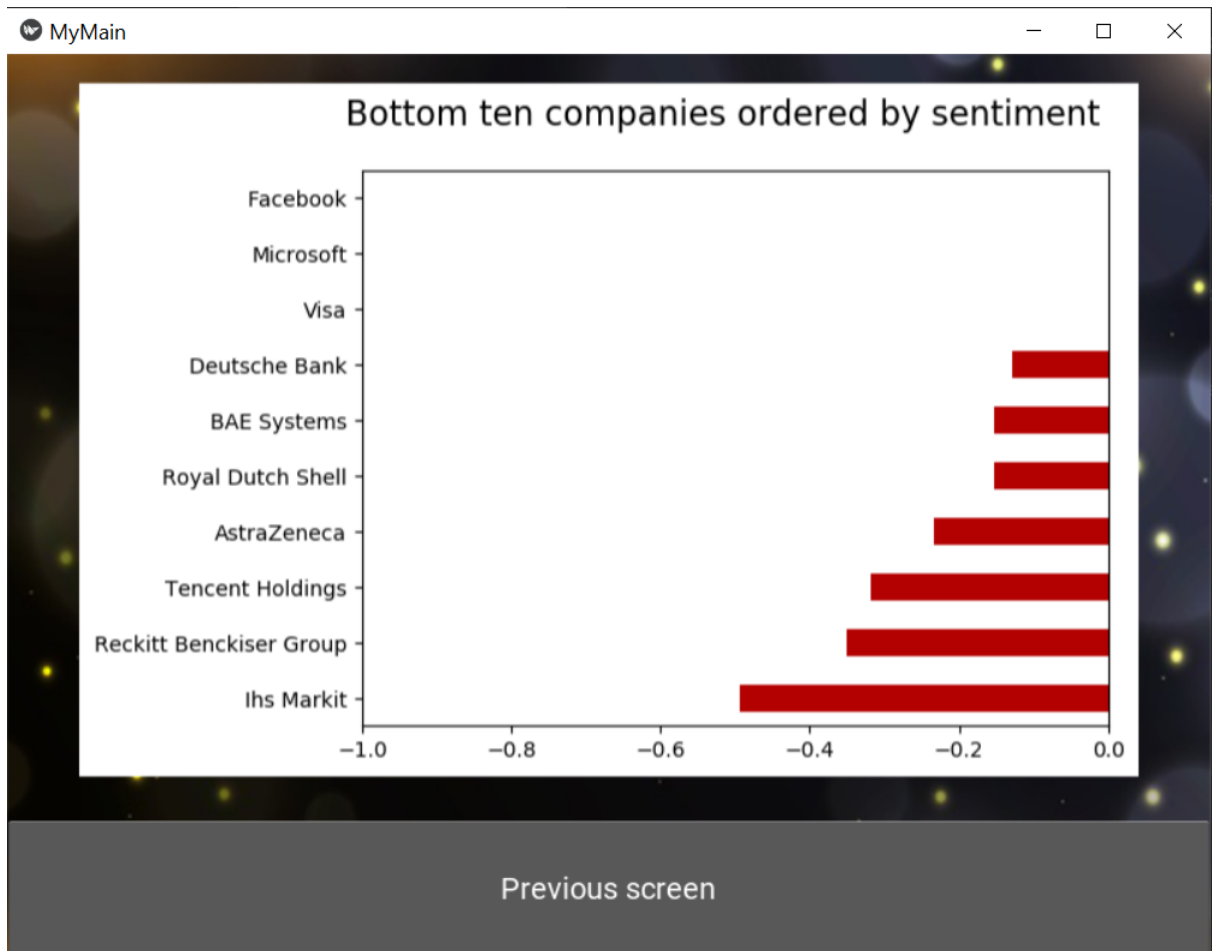
Appendix 10:  
The top 10  
companies bar  
chart





Appendix 11:

The bottom  
10 companies  
bar chart



*The first two questions in this survey relate to obtaining the consent of participants to participate. The Positive, Neutral, and Negative columns refer to the number of respondent answers. In total, there were 10 responses for this survey.*

Appendix 12:  
Comparing VADER  
scores with results  
from the survey

Question	VADER	<u>Positive</u>	<u>Neutral</u>	<u>Negative</u>
3	-0.7	0	1	9
4	0.25	8	2	0
5	0	9	1	0
6	0.27	7	2	1
7	0.735	6	4	0
8	-0.477	0	1	9
9	-0.32	0	1	9
10	0	0	8	2
11	0.5	7	3	0
12	0	0	5	5
13	-0.2	7	1	2
14	0.36	9	1	0
15	0	2	8	0
16	-0.32	0	3	7
17	0.42	0	9	1
18	0.51	7	2	1
19	0.64	5	4	1
20	0	0	10	0
21	0.42	1	5	4
22	0.542	8	2	0

Appendix 12  
(Continued)

Question	VADER	Positive	Neutral	Negative
23	-0.32	0	0	10
24	-0.4	0	1	9
25	0.27	9	1	0
26	0.66	9	1	0
27	0.7	8	2	0
28	0.46	1	1	8
29	0.6	8	2	0
30	0.3	2	4	4
31	0.32	6	4	0
32	0.4	9	1	0
33	0	3	2	5
34	-0.51	0	2	8
35	0.273	7	2	1
36	-0.7	0	1	9
37	-0.38	0	7	3
38	-0.54	2	3	5
39	0.42	4	1	5
40	-0.527	2	1	7
41	0.66	1	2	7
42	0.2	8	2	0
43	0.8	3	7	0
44	-0.51	0	1	9
45	-0.296	0	4	6

Appendix 12  
(Continued)

Question	VADER	<u>Positive</u>	<u>Neutral</u>	<u>Negative</u>
46	0.75	6	2	2
47	0.34	3	7	0
48	0.51	7	2	1
49	0.34	7	3	0
50	0.572	7	2	1

[End of Report]