Goldsmiths
UNIVERSITY OF LONDON

Sentima

Louis Kampman

Supervisor: Tim Blackwell

Submitted in partial fulfilment of the requirements
for the degree of BSc Computer Science
of the University of London.

Department of Computing
Goldsmiths, University of London

5th May 2022

I certify that this dissertation, and the research to which it refers, are the result of my own work.

Abstract

Sentima is a Python application which predicts the sentiment of tweets related to an entered hashtag and visualises the 'popularity' of the posts about it. The application implements model training for the Positive/Negative, Bag of Words and TF-IDF machine learning algorithms. The application includes a Tweepy Twitter Scraper, and its front-end component is made using React. The application uses a variety of techniques including Data Mining, Machine Learning, Neural Networks and Data Visualisation.

Contents

Chapter 1

Introduction

1.1 Focus

The focus of this project is to build a web application which can aid financial analysts in making informed decisions on investments by analyzing Twitter's public perception of searched hashtags.

The analysis of data from Twitter is an example of sentiment analysis which has proven to help businesses understand the feelings of their consumers without conducting formal research and is becoming more and more common today. Financial tools using data collected from Twitter revolve strongly on the use of sentiment prediction, however these approaches to market research are still not widely used as shown by my survey results. With an application like this, financial analysts could derive informed input into the decision making process then taken by portfolio managers and traders.

1.2 Motivation

Often, financial research involves following articles and data provided by large institutions such as Bloomberg (1), and the Financial Times (2). Other tools are also available such as Refinitiv Reuters (3), MSCI (4), Acuris (5), Markit (6), Redd (7), SmartKarma (8) and Yahoo Finance (9), and the price for maintaining subscriptions to these sites are very pricy. With the use of social media growing and the huge amounts of data available, I believe that if financial analysts could use this untapped data, they would be able to understand and interpret the meaning behind thousands of social media posts and understand the public's view of a certain asset or topic, which could help supplement the large amount of time allocated to other research.

The problem my project aims to solve is the time taken for financial analysts to conduct research into their investments. Research can be a time-consuming process involving reading countless articles over a long-time frame. The hope I have for my project is to widen the bottleneck around research in the world of finance and provide quick and easy visualisations which can update an analyst of the public perception of a product. The application does not seek to replace current research methods and trends, it seeks to aid as a complementary tool to an analysts arsenal.

1.3 Aims

My project's novel contribution is its simplicity and accessibility to users. Other resources like this are not available for free and are restrictive to a user's goals. My project aims to simplify what people have done in the past by having an application with minimal interface clutter and which provides a reliable and efficient resource for analysing the public's opinion on a hashtag. Examples I am referring to include products such as Hashtagify (10), a product very similar to mine, and FinTwit (11), a product which utilises twitter data to forecast financial market prices. Most research conducted in finance is done through more established sources, for example the Financial Times and Bloomberg. My project aims to complement this research and allow those looking to invest in the financial markets to make more educated decisions and successfully navigate through price fluctuations in the market. An issue that I have found is that other services like the previously mentioned tools, have too much functionality which overlaps with what is commonly seen in other tools such as the current industry leader, Bloomberg. FinTwit for example, provides a detailed analysis for American stock and its prices, whereas my application can be used for any company or asset which has a hashtag and does not seek to over complicate an analysts' workflow, but instead give them an indication early on in their research whether a financial asset should be avoided, short sold or bought. In my eyes, these applications have sought to help aid the decision making process, whereas my project aims to aid the research process.

The use of Machine Learning to uncover meanings behind qualitative data interests me and I believe that developing applications that do this could provide some huge benefits to the world of Finance, Marketing and more.

I plan for my project to be tested using user surveys once the implementation phase has been complete, this way I can quantify how successful my application's creation has been. My wish is for my application to allow financial analysts to spend less time researching, add insight into the decision making process at work and the visualisations and analysis provided by application to allow for adding to a user's understanding and opinion of a specific research topic. The application should also provide utility for the financial community to forecast an outcome based off the opinions shown by the scraped tweets in my application and outline potential shifts in markets for my users. This section is expanded further in the specification chapter of this report and more.

1.4 Objectives

The objectives I have for my application revolve around these key points: responsiveness, readability, simplicity and reliability. I would like the application to firstly be responsive, so whenever a user changes their screen's size, navigates to another page or makes a search, the application must be fast in responding to these changes and changing its state.

I also seek to implement my application whereby the readability is very good. The text size should always be generous and text should be simple and easy to understand. Having a simple application in my belief is also beneficial as it will help to center our user's attention on our visualisations and even allow for easy navigation.

The reliability of my application is also important to me as the foundation of the application, my machine learning model must preform at a high prediction accuracy so that the returned information is reliable and trust worthy to my users.

Chapter 2

Background

2.1 Sentiment Analysis

Sentiment analysis is the use of natural language processing and text analysis to identify and extract information in qualitative data. The field helps businesses understand the social sentiment of their brand, product, or service and is widely used in combination with reviews, survey responses, online and social media and healthcare materials. Applications created using sentiment analysis range from focuses on marketing and customer service to clinical medicine and finance.

2.2 Natural Language Processing

Natural language processing is the field of artificial intelligence which strives to give computers the intelligence to understand text and spoken words and derive meaning from them. Natural language processing seeks to do this through the use of mathematical modelling for language using statistics. By modelling this data, machine learning models can be trained to give computers the ability to process human language.

2.3 Twitter Scraping

Text mining is a part of the process involved for Natural Language Processing and involves mining data from sites such as Twitter. Twitter is recognised as a gold mine for data, almost all tweets are public and available to analysts to collect and feed into machine learning models. There are many ways to extract data from Twitter and in my project, I decided to use the Tweepy python package. Tweepy is an open-source package provided by python that helps developers access the python twitter API.

2.4 Financial Analysis

Financial analysis is the assessment of the viability, stability, sustainability and profitability of a business, sub-business, or project. It involves using financial data from statements and reports for businesses to derive educated decisions. Financial analysis is typically done by someone with a

Chartered Financial Analyst certification. For someone to effectively evaluate the financial performance of a business for example requires sourcing information from three sources: a balance sheet, an income statement, and a cash flow statement. As stated in Gooding and Briscoe's paper (12), it is increasingly becoming more and more difficult for financial analysis to track and read all relevant texts of research. New ways of helping financial analysts using Machine Learning are constantly being developed as shown in this paper. Popular projects involve predicting the stock price movements of financial assets using data from sources such as Bloomberg and Reuters along with techniques such as Natural Language Processing. As highlighted in Gooding and Briscoe's paper, most financial performances obtained from past systems provide moderate performances and this is argued because quantitative data is not utilised in these models. Because of this, it has been highlighted those models which (instead of forecasting market performances) classify the category of financial articles have become more useful in the data collection stage of analysts and an example of this is TextMiner, the text mining component of the portfolio management system, Warren (13). Reading through the findings detailed in these papers further motivated me to continue developing my project as I was confident that it continued with the conclusion I found: the current use of Natural language processing in machine learning models would be most helpful to financial analysts not through the use of predicting the future price of assets outright, but by improving the efficiency of collecting data and classifying it as well as reducing the time required to do so. My project does not seek to predict future prices of assets to help financial analysts, however, offers analysts a tool to visualise and understand the Twitter community's perception of a hashtag related to an asset, which can then help traders and portfolio managers to then make trading decisions. Highlighted further on in my report (Results section reflecting on the survey I conducted), social media is an untapped source of data which has the potential to be very helpful and the amount of data available from social media is constantly growing. The importance of social media data is further reinforced by the Forbes article I read during the development of my project (14).

2.5 Current Tools

TALK ABOUT HASHTAGIFY AND FINTWIT!

Chapter 3

Specification

3.1 System requirements

I intend to create a financial tool which will allow my users to search for specific hashtags on Twitter and gain an insight into the public's perception of the topic. I want my application to work in combination with Twitter's API, allowing me to can collect recently published tweets from their site and later conduct sentiment analysis on the scraped data. The results obtained from assigning a sentiment to the tweets will then be visualised to make the data more readable and easy to understand. The application should be easy to use and to achieve this, I aim to minimise the clutter and amount of text visible on the user interface as well as to reduce any scrolling where necessary.

For a more in depth summary of the requirements I have made for this project, please navigate to the following sub-sections containing tables of my application's functional and non-functional requirements.

3.1.1 Functional requirements

The functional requirements of an application refer to the ones that define what an application is supposed to do. In the case of my application, the main functionality revolves around the navigation bar (which contains the search bar and navigation links to other pages) as well as the home screen which serves as the canvas in which the users will view the returned analysis. Furthermore, the functional requirements are centered upon the use of the application's different pages for the different data displayed throughout, as well as the functionality revolving around the machine learning model and Twitter scraper.

Title	Description
View an empty 'Home' page before any search has been made. The navigation bar should be visible to allow for searching hashtags and navigating to other pages	• Displays home page with a navigation bar containing the application's name and logo, a hashtag search bar (which will take user entries to populate the home page's components) and a drop-down navigation menu. • Access the 'Home', 'About' and 'Contact-Us' pages
View the 'Home' page	• Displays three components upon searching for a hashtag with the visualisations: Table of scraped Tweets, Pie chart of calculated sentiments and Word cloud of popular words found in the scraped Tweets. • Displays the navigation bar, which will allow the users to navigate to the application's other pages
View the 'About' page	• Displays two components which provide an introduction to the application as well as instructions on how to use and navigate it. • Displays the navigation bar, which will again allow the users to navigate to my application's other pages.
View the 'Contact-Us' page	• Displays the contact information of the application developer. • Displays the navigation bar, which will again allow the users to navigate to my application's other pages.
Search for a hashtag of interest and see visualisations on the obtained tweet text and calculated sentiments page	• Users should be able to enter a hashtag of their choosing and have results returned on the 'Home' screen with small waiting times.

3.1.2 Non-functional requirements

Non-functional requirements are those which judge the operation of the application rather than the specific behaviours or its functionality. In the case of my application, having minimal loading

times and a clean, well sized design would be examples of requirements I strive to follow in my implementation. These improve the user experience of the application, professionalism and many other aspects.

Title	Description
Performance	• The application must be able to run smoothly and without any noticeable run-time issues.
Capacity	• The application must be able to process a large number of consecutive searches and the number of tweets returned from every search should be tweaked well so that the application does not go over the limit set by Twitter's Developer platform (based on my application's 'Essential' access level.
Reliability	• The application's machine learning algorithm which predicts the Tweet text's sentiment should be reliable and predict the sentiment of tweets with a good accuracy. • Furthermore, the processing made on Tweets when they are collected from the scraper is important because for example, the stemming done helps to simplify the words inputted into the application's sentiment prediction model to their root meaning.
Accessibility	• The application must be accessible and easy to use for as many people as possible. • There shouldn't be any scrolling and the data visualisations should be clear and easy to interpret. The design of the application as a whole should be well designed, without clutter.

3.1.3 Testable Requirements

The testable requirements of my application will be used to quantify the success of implementation for my application. The requirements listed in the table below will be tested through the use of my second user testing survey where participants working in finance will answer questions about my project after watching a screen-recorded demonstration uploaded on YouTube and nested in my Google Forms document.

The results from this user testing can be found in the results chapter of this report, and within the discussion section, I will be evaluating the degree in which these requirements have been met. The way in which I will be validating whether these requirements have been met is with the use of the second user testing survey I conducted after completing the implementation phase of my application.

For each of the requirements, I would like their respective question(s) from the user survey to reflect a positive result. In order for the results to be positive, the respective question from the survey should show a minimum of 50 percent of the participants responding with likelihood values

of 3 or higher. The way range of values for these likelihoods will be measured within the range of 1 to 5 (1 representing 'not at all' and 5 representing 'extremely'). I would also like the average result for each of these questions to be a minimum likelihood value of no less than 2.5.

Title	Description
Application should help reduce the amount of time analysis spend researching.	For the first testable requirement of my application, I will be testing to see whether my user testing participants working in finance believe my application could provide utility in reducing the amount of time they need to spend researching into potential trades.
Application should be helpful in the decision making process.	My application's user testing should show results displaying that there is an interest in my application and its utility.
Analysis from application should add to a user's understanding and opinion of a topic.	A goal I have for my application is for it to help my users deepen their understanding of a topic and hence critique their current opinions of their entered hashtag's field.
Application should have utility in forecasting an outcome by opinion.	Another goal for my application is for my testing to reflect a utility for forecasting an asset's outcome via price fluctuations, by analysing the opinions shown through my application's visualisations.
Application should outline shifts in financial markets.	The application's visualisations for a hashtag's sentiment should provide analysts with a solid introduction into their research by outlining the public's opinion. This analysis should help, alongside their other research with the decision making process.

Chapter 4

Design

When designing my application, the core goals I wanted to achieve were to implement a front end which is simplistic, looks professional, is responsive, reactive and finally, easy to use and comprehend. I believe my designs helped a lot in preparing me for what to expect later in the implementation phase and more.

4.1 Navigation Bar

For the design of the navigation bar of my site, I took inspiration from sites such as OpenSea and YouTube, two sites with search bars in their navigation bar. The way these sites are designed was helpful in creating a similar layout and allowed me to have a final product with human utility.

Initially, you will find from the following images of my high fidelity application designs, that I did not have a drop down menu for my navigation links to the other pages in my application. This was designed towards the end of my project's implementation as I saw that there was an area of improvement I could take advantage of.

4.2 Home Page

For the home page of my application, I created several options for final designs. I designed the home page and how it should look before any searches are made (Figure 11.1), as well as the default home page after a user searches (Figure 11.2). This home page design only contains 3 components for visualising the data collected; a table, pie chart and word cloud component. The reason I made this design was in case the other designs I had in mind did not work when trying to implement them and their additional visualisations.

The next designs, as I touched upon, contain additional visualisations. An example of this is in the following design (Figure 11.3), our home page with the additional ternary graph. I found out one could use a ternary graph when visualising three separate fields of data and thought it would be a great option for visualising the overall number or Retweets, Replies and Likes tweets with a certain hashtag got. Sadly however, I ran into issues when adding this functionality to my project and stuck with my earlier design.

The following image (Figure 11.4) shows our home page with a total of 5 components for visualising the data fetched from our back end. On top of the 3 default components (table, pie chart and word cloud), I added the ternary component (mentioned earlier) as well as a world map which

would be used to visualise the different scale of sentiments relative to their location of publishing. I believe this would have added a cool addition to my project and add another angle for analysts to go further in depth with the information provided to them.

4.3 About Page

The design for the last two pages (about and contact us) is quite simple as the data within is static. For the about page (Figure 11.5), I decided to keep it simple and have two containers of text for the users to read. One with an introduction of the project, with inspiration from the text in this report, and another was designed to give users instructions on how to use the application.

4.4 Contact Us Page

For the contact us page (Figure 11.6), I once again kept the design very simple for the static data I would be displaying. I decided to only have one small component on this page large enough to contain my contact information such as my email and GitHub account.

Chapter 5

Methods

This chapter will provide a detailed overview of the methodology followed in developing my application. There are three sections in this chapter containing information about the implementation of my machine learning model, Twitter scraper and React front end.

Each of these sections form the foundation to my application. Once I had implemented the background functionality of my project (the sentiment analysis model and Twitter scraper), I then worked on making the user interface to create a full stack application. Please refer to Figure 11.7 for a visualisation depicting the application's file structure.

Throughout the implementation of my application, I have been following the a timeline I set for myself from the start. This diagram can be found in the appendix section, Figure 11.8

5.1 Model

The implementation of my machine learning model was done in five main sections: load and analyse, data visualisation, text normalisation, text representation and sentiment model. The model was developed using the interface provided by Jupyter Notebook and the programming language Python. After the model's creation, it was saved as a SAV file using the Joblib Python library and utilised in the app.py file in our full-stack application, Sentima's back-end.

Our sentiment prediction model was trained using a large data set collected from the website, Kaggle. The name of the data set is called "Sentiment140 data set with 1.6 million tweets". As mentioned in the title, this data set contains 1,600,000 tweets containing information such as their sentiments, ids, their publish date, user and text.

The data visualisation I did uncovered a balanced data set, and the text normalisation and representation allowed us to abstract a numerical meaning behind our data and allow our models to learn patterns in the data with a higher efficiency.

5.1.1 Imported Libraries

The libraries I was required to import into my model's notebook included: Pandas (15), Numpy (16), Random (17), Matplotlib (18), WordCloud (19), Regex (20), Emoji (21), Contractions (22), NLTK (23), Scikit-learn (24), Seaborn (25) and Joblib (26). Furthermore, the first stage of my project was programmed using the Python programming language (27).

The libraries and their corresponding functionalities are listed below:

Library	Functionality Used	Description
Pandas	read csv	Pandas is a fast, powerful, flexible and easy to use data analysis and manipulation tool, built for the Python programming language.
Numpy	zeros, log	Numpy is a Python library which adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate these arrays.
Random	randint	Implements pseudo-random number generators for various distributions.
Matplotlib	subplots, figure, pie	Python plotting library which extends Numpy.
WordCloud		Library which allows us to make visual representations of text data with Word Clouds.
Regex	sub	Python library which provides regular expression matching operations.
Emoji	demojize	Python library which allows you to emojify and de-emojify texts and emojis.
Contractions	contractions dict, fix	A Python library for expanding and creating common English contractions from text. This library was essential for the text normalisation section of my model's implementation.
NLTK	wordtokenize, PorterStemmer, LancasterStemmer, SnowballStemmer, WordNetLemmatizer, WordNet	A leading platform for building Python programs to work with human language data. Used in our project for the Tokenisation, Stopword Removal and Stemming in the text normalisation stage.
Scikit-Learn	CountVectorizer, TfidfVectorizer, traintestsplit, Logistic Regression, accuracyscore, confusionmatrix	Machine Learning library that features various classification, regression and clustering algorithms.
Seaborn	heatmap	Python data visualisation library based on matplotlib. Used in the last stage of implementation for my machine learning model (Sentiment Model) where I plotted the confusion matrices for each of my three models (Positive/Negative Frequencies, Count Vector/ Bag of Words and TF-IDF).

Joblib	dump	Tool providing lightweight pipe-lining in Python. In the case of my project, once I had finished creating my final sentiment prediction model, I used the Joblib library to save my python model object into a file later linked into my application's python back end.

5.1.2 Load and Analyse

The beginning of my model's implementation required the loading and analysis of the data contained in the Kaggle data set I used (28). The data set containing the 1.6 million tweets I would be training and testing my model was pre-processed in this section in the following steps:

- Figure 11.9 - Created a Python data frame storing the imported CSV data containing our data set. Set the header labels to None and outputted a sample of 10 samples from the data frame.
- Figure 11.10 - Added headers to the data frame.
- Figure 11.11 - Removed unnecessary columns such as Flag.
- Figure 11.12 - As the Kaggle data set was originally designed with the sentiments 0 and 4 (negative and positive), I replaced any instances where the values were 4, to 1.

5.1.3 Data Visualisation

To improve my data's readability for our model training, I decided to make some visualisations in order to better understand the data set and its features. I displayed the ratio of positive and negative tweets in the Kaggle data (using pie charts) as well as for the types of words used in positive and negative tweets (using word clouds). The observations collected from these visualisations were that we had a balanced data set with a 1:1 ratio between positive and negative tweets. Furthermore, the Word Clouds I created highlighted a correlation between the types of words used and the assigned sentiment of our tweets. These were encouraging to see as this showed that our data set would prove to be useful in our project. The steps I followed in this section included:

- Figure 11.13 - Created a Pie Chart to display the number of instances for positive and negative tweets in our entire data frame.
- Figure 11.14 - Outputted the value counts for each of the sentiments in our data frame and found that we had 800,000 tweets for both negative and positively rated tweets.
- Figure 11.15 - Created a Word Cloud displaying the most commonly used words for tweets with positive sentiments.
- Figure 11.16 - Created another Word Cloud, this time displaying common words for tweets with negative sentiments.

5.1.4 Text Normalisation

In this section of my model's implementation, I chose to tackle text normalisation. This involved cleaning and simplifying the messy formatting, spelling mistakes and poor grammar found in tweets. I also tokenised the Tweet text in my data set and stemmed each word for to ensure for better model performance in the next steps. The steps I just touched upon are detailed below:

- Figure 11.17 - Created a method to remove retweet tags from Tweet text.
- Figure 11.18 - Created a method to remove user tags from Tweet text.

- Figure 11.19 - Created a method to replace emojis with their text counterpart.
- Figure 11.20 - Created a method to remove URLS from Tweet text.
- Figure 11.21 - Created a method to remove hashtag characters from Tweet text.
- Figure 11.22 - Created a method to replace uppercase letters in Tweet text to a lowercase format.
- Figure 11.23 - Created a method to remove character repetition from Tweet text.
- Figure 11.24 - Created a method to remove punctuation repetition from Tweet text.
- Figure 11.25 - Created a method to remove word contractions from Tweet text.
- Figure 11.26 - Created a tokenization function to convert strings of Tweet text to arrays of strings/words (tokens).
- Figure 11.27 - Created a method for stemming tokens of Tweet text, allowing you to select between the Porter, Lancaster and Snowball stemmers provided by the NLTK Python library.
- Figure 11.28 - Created a method combining the functionalities from each of the previously created methods, called processtweet, and then tested this function out with fake Tweets.

5.1.5 Text Representation

To represent my text as numerical data to my model, I utilised functions made earlier for text normalisation to transform all the tweets in my data into tokenized and stemmed text. I then created three separate vectorizer methods using Positive/Negative Frequencies, Bag of Words and Term Frequency – Inverse Document Frequency (TF-IDF) to be tested later upon our model creation. It was discovered later in my project's development that the best performing model used the TF-IDF vectorizer. The steps I followed for this section touched upon above are highlighted below:

- Figure 11.29 - Created a new column in my data frame where I stored the tokenized version of each of our data point's Tweet text after running our processtweet function on the entire data frame's Tweet text column.
- Figure 11.30 - Converted the token and target (sentiment) columns into Python lists named X and y.
- Figure 11.31 - Created a test corpus and the methods buildfreqs and tweettofreq for building a frequency dictionary for every word in our corpus and a the sentiment of the tweet it came from. Tweettofreq enabled me to convert tweets to their 2D vector version using the frequency dictionary created prior. This created vector, representing the number of times the words in an inputted tweet had been used in tweets of positive sentiment and negative sentiment relative to our corpus. These functions would then be used later on in the project.
- Figure 11.32 - I then created a visualisation displaying the differences between the generated vectors for words like sad and happi (stemmed version of happy).
- Figure 11.33 - I created the fitcv function for the count vector/ bag of words vectorization method for this part of our implementation. NEEDS MORE WRITING
- Figure 11.34 - I created the fittfidfnew method (as opposed to the old version, so that I could save the method with joblib without any errors). NEEDS MORE WRITING

5.1.6 Sentiment Model

Finalising my model's development, I then trained models created with the three separate vectorisation methods (using a 80:20 training and test ratio). I created three models using the logistic regression python library and after plotting the confusion matrixes fitted on the outputted test

predictions, found that the TF-IDF model I had created was the best performing with 78.27 percent accuracy. This model was then saved as a PKL file using the Joblib Python library which allowed me to then use the model later on in the back end of the full stack application I would later start developing. The aforementioned steps I followed are listed below:

- Figure 11.35 - Created a helper function for plotting Seaborn confusion matrices later on.
- Figure 11.36 - Split my data into training and test sets.
- Figure 11.37 - Created a method for fitting a logistic regression model with X and y training data. This function would then be used later to input our three seperate vectorized training data versions.
- Figure 11.38 - Fitted a model on my vectorized (Positive/Negative Frequencies vectorization method used) train data using the above function created earlier.
- Figure 11.39 - Fitted another model on our vectorized training data (this time, with the Count Vector/ Bag of Words method) using the function fitlr.
- Figure 11.40 - Fitted our last experimentation model (using the TF-IDF method) using the fitlr method.
- I then compared the performance of each of our three trained models vs the unseen test set, revealing our best preforming logistic regression model (with a maxiter value of 1000 and the vectorization method of TF-IDF), our final model. The plotted confusion matrices can be found in
- Figure 11.41 - Tested the newly created prediction model with a fake tweet.
- Figure 11.42 - Saved my model and vectorization method using the Python Joblib library for my back end's access.

5.2 Twitter Scraper

For my twitter scraper, I decided to use the Tweepy python package, allowing me to connect directly to Twitter's API. This required the creation of an account on Twitter's developer platform (29) to generate personalised keys (API key and API security key for example) which would allow me to connect to their platform. The functionality of my scraper relies on two main functions: getClient (Figure 11.43) and searchTweets (Figure 11.44). Testing was also done to confirm that the scraper worked as intended (Figure 11.45) and I was pleased to see that everything worked as intended.

getClient's purpose involved connecting and returning the client object from tweepy. As for searchTweets, the function would call getClient, use the Twitter developer function, searchrecent-tweets and populate an array called results with 5 elements (id, text, retweet count, reply count, like count) per index. The searchrecenttweets function provided by Twitter is available through "Essential" (default) access level, which uses the second released version of Twitter's API. I used the parameters query, tweet fields, place fields and max results which allowed me to tweak the output type of my searches.

- The query parameter would be passed by the user in the front end.
- The tweet fields represented the additional fields of data I wanted to scrape for each of the collected tweets form our search. In this case, I wanted public metrics (which would return data for each tweet's count of retweets, replies and likes).
- The place fields parameter is redundant as this was inputted whilst I was trying to collect the place object for our tweets which would allow me to then make a visualisation representing the popularity of hashtags across different countries.

- The max results parameter allows developers to tweak the number of results they would like to collect from their Twitter search. In my case, the value for max results was set to 10 permanently to prevent the application from going over the ”Essential” access level search limit (450 tweets per 15 minutes).

5.3 React Front End

When working on the front end of my application, I chose to use ReactJS (30) as my development library. ReactJS is an open source JavaScript library designed by Facebook for creating rich and engaging web applications quickly and efficiently with minimal code. The core objective of ReactJS is providing the best possible rendering performance. Its strength comes from the focus on individual components. Instead of working on the entire web app, ReactJS allows developers to simplify and break down the code for their user interface into smaller components.

The development of the front end of my application came in four distinct phases, development on the navigation bar, home page, other pages (about and contact-us) and the application's CSS and are detailed in the subsections below.

I have also included screen-shot images of my React code as a reference for this methodology section. These can be found in the appendix section at Figures 11.46 to 11.54.

5.3.1 Navigation Bar

I decided to implement the navigation bar first as the application's core functionality would depend on this component and I wanted to make sure that the back and front end had some form of connection (with the use of a fetch command) through the use of a POST method at least.

I used an online logo generator, FreeLogoDesign (23) to create my application, Sentima's logo, created my search bar and button and added navigation links on the right hand of the navigation bar for users to navigate to the other pages, about and contact-us.

The way my navigation bar was structured was, the root file (index.js) contained each of the components which comprised of the final navigation bar and the additional file (NavSearch.js), comprised of the code which created our search input bar and button as well as the function which POSTs the entered hashtag into the back end, fetching the analysed tweet data back, once processed.

Later on in the development of my project, once I had already finished implementing the other functionality such as the entire home page and the CSS for the application, I decided to implement a drop down menu for the navigation bar in order to make the component look more clean and professional. I felt that having the large text which could be used for users to navigate to other pages was unnecessary and as a result, I utilised use states to check whether a button was clicked. Depending on the click, either the search bar, or navigation links would be displayed at one time.

5.3.2 Home Page

Once I had implemented the functionality for the Navigation bar, I then moved on to visualising the scraped and classified Tweet data. The order in which I implemented the three different components seen in the home screen was as follows: first I worked on the table displaying all the fields of data collected from our back end, I then worked on the pie chart and finally on the word cloud.

I faced several challenges in this stage of my project's implementation. I faced some confusion when making the first (table) visualisation on the home page and spend a lot of time figuring out

how to traverse the JSON file I had collected form the back end. Once I had figured out how to map this data, I knew that I wouldn't have to worry about any issues regarding my data and I was right, from here on out, the main difficulties encountered were in regards to the libraries I needed to import for the two other visualisations.

The libraries I imported to make these visualisations were: Pie (from "react-chartjs-2" and "chart.js/auto") and Wordcloud (from "react-wordcloud"). For these visualisations, I made components (PieChart and MyCloud) with their own files in the "Others" and "components" folders.

For the development of my word cloud, I decided to add some code within the MyCloud object I had created which would manually iterate through each of the scraped tweets and split their text into arrays of individual word strings. Unfortunately, I perhaps should have implemented additional functionality in my backed which could have allowed this text to have some normalisation such as the removal of unnecessary words such as stop words. I did not add this functionality to the front end object of MyCloud, to avoid increasing the processing time when searching for a hashtag. Overall however, I am happy with how this visualisation turned out and believe that it has a positive affect on my application.

I had also tried creating a ternary graph visualisation to display the scraped tweets' overall number of retweets, replies and likes, however due to the issues I faced when trying to import several different libraries for this visualisation, I decided not to continue spending time on this task, and instead work on my project's report.

5.3.3 Other Pages

The implementation of the other pages in my application took place once I had completed creating the home screen containing three components of visualisations which would be populated upon a user searching for a hashtag. This phase was very straight-forward and the only time consuming thing that took place was refining the CSS for these pages under different screen sizes, etc.

5.3.4 CSS

Once I had developed the core functionality and design of my application, I moved onto refining the CSS of my project for the last time. Overall, I ended up with 6 different screen size conditions for my application's CSS as I noticed that when resizing the application, the navigation bar would face issues of overlapping as well as the components in the home page. To counteract this, I reduced the text size of some components upon screen size reduction; I also made a condition for my home screen table, where the additional columns displaying the data for our scraped tweets' retweet, reply and like counts. For the CSS for my application's mobile version, I changed the design of the home page whereby the visualisation components would be ordered in the shape 1 x 3, so a user would need to scroll down to see each of these.

Chapter 6

Results

6.1 Machine Learning Model

As mentioned in the previous chapter, my final machine learning model was selected out of the three I tested from the different vectorization methods I used (Positive/Negative Frequencies, Count Vector/Bag of Words and TF-IDF). In order, the sentiment prediction accuracies collected when predicting the test set was as follows: 69.44 percent (Figure 11.55), 78.11 percent (Figure 11.56) and 78.27 percent (Figure 11.57). Out of these, the best model and the vectorization used for this, was the TF-IDF with 78.27 percent prediction accuracy against unseen data.

6.2 Twitter Scraper

For the development of my Twitter Scraper, I successfully implemented its functionality. One issue I encountered was related to the permissions granted to me by Twitter's Developer Platform. The Tweepy function "Search Recent Tweets" used in the searchTweets function I created only offers developers the ability to scrape 450 tweets per 15 minutes. This has caused some limitations for the scale of my project, however I believe that in the future, a good thing first step in improving the utility of my project and its concept, would be to apply again for a more advanced access level. Unfortunately, I was unable to implement functionality in my project for visualising the sentiments of tweets in different geographical regions as I found that many scraped tweets I was collecting in my implementation were posted by users with their location tracking disabled, which left this data empty. With the ability to scrape more tweets at a time, perhaps I would have been able to collect more data for geographical sentiments, however this Tweepy functionality would still have the underlying flaw. I am happy that I was able to implement the additional functionality in my scraper for collecting the Retweet, Reply and Like count for tweets and this added a lot to my project. In the future, I would like to continue working on the Ternary plot I mentioned earlier so that these data points can be better visualised.

6.3 Front End

Overall, I am very happy with the results that I have been able to achieve from the design and implementation phases of my project in regards to its front end. I was able to familiarise with the

React framework after starting my project with very little experience and successfully implemented a well designed, reactive web application. Each of the components and the functionality nested within them are great and serve their function perfectly. The navigation bar is bold, clean and the search bar and navigation bars within are very accessible and easy to use. The home page's components used for displaying our scraped data and visualisations work very well and only appears once a hashtag has been searched and collected. Furthermore, the visualisations are clear, not too bold on the screen (through colour and size) and possess cool responsiveness and animation. The word cloud turned out well in my eyes and I am very happy with the additional dimension it adds to my application.

In the design phase of my project, I drafted three alternative designs for my home page and the components within. Unfortunately, due to complications I encountered in the development of the additional components (ternary graph and world map), I was unable to add these to my application. In the future I would like to add these to my project as the second round of user testing (shown in the following sections) showed that my survey participants believed they would add good additions to my project.

Overall however, I am happy with what I have achieved as the second user survey results reflect that my application has good promise and that its concept could flourish into a very useful tool if one had the resources to develop it further.

6.4 Testing

For the testing of my application, I decided to use White box testing and user testing as I believe that the combination of these two was most suitable for my project and would be very useful in identifying successes and failures in my project.

6.4.1 White Box Testing

White box testing is a method of software testing in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code based testing and Glass box testing.

The testing I conducted for this section can be found in the appendix section, in Figure 11.58. Overall, I am thrilled with the results from this round of testing as it uncovered no major flaws in my application, and using the technique of white box testing, I was able to outline issues with my application and amend any issues.

Examples of this include the issues I faced when collecting data from the Twitter API for a searched Tweet's location and when trying to import and implement the different visualisation objects (Ternary Graph for example). The expected results and detailed description of the issues and how I fixed these can be found in the image I referenced.

6.4.2 User Testing

The user testing of my application was done through the use of Google Forms surveys. During my project's development, I conducted my first out of two surveys (Figure 11.59) which asked my test group (all of whom worked in finance) what they thought of the prospect for having an application

like mine. The below table contains each of the questions I asked as well as the information I was able to gather from the results.

Question	Results	Figure
How much time do you roughly spend researching your profession each week?	Results showed that many participants (27 out of 44) spend between 1 to 10 hours per week researching, whereas 12 (27 percent) spend more than that. These results highlight my previous writing about the magnitude of time that goes into financial research.	Figure 11.60
Do you use Twitter as a source of research?	The results from this question highlighted that many of my participants did not use Twitter as a tool in their work at the time the survey was conducted (25 out of 44). This magnified the idea in my head which was that current financial services do not take advantage of the vast amounts of data at their disposal from social media sites and the rest of the web, encouraging me to continue my development.	Figure 11.61
Do you take investment decisions based on Twitter opinions?	The results from this survey question reflected the one prior as many of our participants didn't use Twitter much at all at the time. The results showed that 77 percent of our participants never took investment decisions based off Twitter opinions.	Figure 11.62
How likely would you be influenced by research collected from Twitter on the public's perception of a topic?	Many of my survey participants showed a disinterest in to the prospect of analysing the public's perception of a topic on Twitter (18 out of 44), however more showed some interest (26 participants). I was happy to see these results as it showed that there was some interest and demand for my application.	Figure 11.63
Do you think Twitter could add an additional dimension to your understanding of a topic?	The results for this question were great as it showed that 86 percent of the participants were open to the idea of using Twitter to expand their understanding of a topic.	Figure 11.64
Do you think Twitter could add an additional dimension to forecasting an outcome by opinion?	I again received positive feedback, showing that 80 percent of my participants understood the potential for using Twitter as a tool for understanding the public's opinion and h ow they could affect the markets	Figure 11.65
Do you think opinions voiced on Twitter can impact the financial market and its directions?	These results were great as they showed 91 percent of my survey participants agreed with the idea that sites such as Twitter and the communities within have the power to shift market prices.	Figure 11.66

Do you every change your opinion based on Tweets you read?	The results from this survey question showed that 52 percent of the participants were unlikely to change their opinions based of Tweets they read. I can understand this conclusion as many Tweets contain little information which can be of use when changing one's mind, however my application would negate this with the use of many Tweets.	Figure 11.67

Overall, the first of my two surveys showed success and proved that the concept for my project interested my participants, who justified its conception. After conducting my survey, I was excited to continue planning and implementing my application so that I could then see the results from my second round of testing, where the individuals whom were interested in the first round would be impressed and those that were not could change their minds and reflect the project's quality through their survey responses.

The second round of user testing I conducted was after completing the implementation of my application and involved asking the same questions as seen before in the first survey and additional questions once my users watched a recorded demonstration of my project which I posted to YouTube (33). The below table contains each of the questions I asked as well as the information I was able to gather from the results.

Question	Results	Figure
Do you work in finance?	Many of the users I was able to participate in my second survey worked in the field of finance (30 out of 51, or 59 percent). I am happy that I included this additional question in my survey as it ended up adding an additional dimension to my user testing whereby I could understand the perception of my project from two different angles.	Figure 11.68
How many hours do you spend per week researching for your profession?	Due to an improvement in the answer requirements I set for this question, I was able to reduce the number of responses which were of no use such as those found in the first survey under 'other'. The percentage of the responses from participants working in finance that research within the hour range of 1 - 20 did not change very much in the second survey (75 to 70 percent) and the maximum recorded number of hours spent increased by at least 10; I found that 5 (four of whom worked in finance) of my participants spend more than 50 hours a week researching.	Figure 11.70
Do you use Twitter as a source of research?	The answer 'never' decreased by 14 percent and 'sometimes' increased by 18 percent since my last survey. Overall, the results found in this question changed considerably as the number of that answered with 'never' went down, perhaps reflecting that the community might have already started embracing the medium more.	Figure 11.71
Do you take investment decisions based on Twitter opinions?	Again, the number of people that answered with 'never' decreased, this time by a lot (27 percent). Furthermore, in the first survey, no participants answered with 'often' and 'always'. Perhaps in recent times such as Elon Musk's acquisition of the company has made the medium more attractive for financial news and analysis.	Figure 11.72
How likely would you be influenced by research collected from Twitter on the public's perception of a topic?	The responses from this question were again quite different to what I saw in my first survey. The number of participants which answered with 1 (for not at all) decreased by 24 percent and the number of people whom answered with a 4 or 5 increased by 15 percent.	Figure 11.73

Do you think Twitter could add an additional dimension to your understanding of a topic?	Once more, I found no differences between this set of results and those found in the first round of user surveys.	Figure 11.74
Do you think Twitter could add an additional dimension to forecasting an outcome by opinion?	There were no major changes found in the results of our first survey's question and this one's, highlighting a continued interest in the prospect of analysing Tweets.	Figure 11.75
Do you think opinions voiced on Twitter can impact the financial market and its directions?	Similar to the previous question, there were no major changes in the results for this question.	Figure 11.76
Do you change your opinion based on Tweets you read?	Once more, no major differences in results were observed	Figure 11.77
Do you think a tool like this could help reduce the amount of time people in finance spend on research?	The results for this and the following questions are very good to see. 37 percent of my participants responded with a likelihood of 4 or 5, 47 percent for 3 and 17 percent for the following options, 2 and 1. These are good results as they show that roughly 84 percent of my participants in finance found my application could serve to reduce the amount of time analysts spend researching.	Figure 11.78
Do you think a tool like this could be helpful to analysts when deriving financial decisions?	43 percent of my participants answered with likelihoods of 4 and 5, 43 percent for 3 and 13 percent for 2 and 1. Again, these results showed massive promise and I was very happy to see that my participants found interest in my application and even messaged me to congratulate me on what I was able to produce after completing the survey.	Figure 11.79
Do you think that the analysis made when using this application could add an additional dimension to your understanding of a topic?	I was very pleased with the responses for this question as I received no cases of likelihoods being 1 (for 'not at all'). Only one participant responded with 2, 12 (40 percent) selected 3, 40 percent also selected 4 and the rest (5 or 17 percent) selected 5 as their answer.	Figure 11.80

Do you think this tool could add an additional dimension to forecasting an outcome by opinion?	Again, the results I obtained were very pleasing. 10 percent responded with a likelihood of 1 and 2, 37 percent with 3 and 53 percent with 4 and 5.	Figure 11.81
Do you think this tool could help outline shifts in financial markets?	Once more, I obtained good results. 13 percent of my participants answered with likelihoods of 1 and 2 and 47 percent responded with 4 and 5.	Figure 11.82
Do you think using a tool like this could help change your opinion on a topic?	The results obtained form this question were once again overwhelmingly positive. 50 percent of the responses had likelihood entries of 4 to 5, 27 percent entered 3 and the rest (23 percent) entered 1 to 2.	Figure 11.83
In your mind, how much more useful could my project be with the use of the ternary visualisation showed in the image above?	The results from this survey question showed to me that adding the additional functionality of the ternary graph I designed would make a positive addition to my application.	Figure 11.84
In your mind, how much more useful could my project be with the use of both the ternary graph and world map visualisations as shown in the image above?	Once more, my survey participants enjoyed the idea of adding additional visualisations to my application. The conclusion I made from these two questions on the design of my project is that I will work on these two additions immediately if I ever decide to work on improving my project.	Figure 11.85

As seen in the above table, the second round of user testing gave me solid feedback, concluding that I had successfully implemented my application whereby my survey participants showed an interest in its functionality and voted positively to the different questions I asked in regards to its potential utility in the field of finance. My users showed interest and my application showed promise in analysing short term market trends using Twitter Sentiment Analysis.

Analysing Testable Requirements

The results collected from the second user testing survey I conducted as mentioned above, returned some really positive results for my project's concept and current functionality. In the specification chapter of this report, I outlined the system requirements at the start of my project's implementation. Within this section, I detailed the testable requirements for my project, the goals which could be tested from the results obtained above to quantify the success of my project's programming.

My analysis can be seen in the below table:

Title	Results
Application should help reduce the amount of time analysis spend researching.	The results shown above outperformed this goal significantly. My wish of only 50 percent of the finance community to see the utility of my project in saving time in the research stage of decision making was beaten by 33 percent! The average likelihood for the responses was calculated to be 3.3, higher than the minimum of 2.5 (by 0.8).
Application should be helpful in the decision making process.	The results again outperformed this goal significantly, this time by a little more. The minimum of 50 percent of participants responding with a likelihood of 3 and higher was exceeded by 36 percent! The average likelihood for the responses was calculated to be 3.43, higher than the minimum of 2.5 (by 0.93).
Analysis from application should add to a user's understanding and opinion of a topic.	The results for this requirement were collected from two of my survey's questions. All in all, the results were amazing for this section. 57 percent of the survey participants said their understanding of topics could be improved with my app (likelihood of 4 to 'extremely' likely, 5). 50 percent of the survey participants said their opinion of topics could be changed with my app's analysis (likelihood of 4 to 'extremely' likely, 5). For both of these question's results, the average entered likelihood was entered as 3.7 and 3.4.
Application should have utility in forecasting an outcome by opinion.	The results in this section showed that 90 percent of the survey participants believed my application could help forecast an outcome in the financial markets from analysing Twitter opinions (entered likelihoods of 3 to 5, 'extremely'). This was very positive and outperformed my minimum requirement for 50 percent for this likelihood range. Furthermore, the average entered likelihood was calculated as 3.56.
Application should outline shifts in financial markets.	For my final testable non-functional requirement, the results from my second user survey showed that I had outperformed once again, the minimum requirements I had set for my application. 87 percent of my finance participants answered with likelihoods ranging between 3 and 5, an improvement of 37 percent! The average entered likelihood was also calculated for the final time and was found to be 3.46.

6.5 Project Appearance and Back End Code

As mentioned before, I was able to implement the back end functionality for my application successfully. This was done through the use of the app.py file stored in my project's back end folder. The code within this file can be seen in the Figures: 11.86, 11.87, 11.88 and 11.89. With this functionality, along with the application's final user interface, my application was able to satisfy the goals I had set out to achieve. Figures demonstrating the final look of my application can be found in Figures 11.90 to 11.100. These images show the application's display both in desktop and mobile formats with different searched hashtag inputs to demonstrate how our visualisation components

operate.

Chapter 7

Discussion

7.1 Machine Learning Model

I am pleased with the result I was able to obtain for my machine learning algorithm. With a 78.27 percent accuracy on unseen data, my algorithm is able to efficiently predict the sentiment of Tweet text with the use of the training data I used from the Kaggle data set.

7.2 Scraper

The Twitter scraper turned out to be successful and it was implemented using the Tweepy Python library to scrape 10 tweets every search, collecting data such as the Tweet ID, Text, Retweet, Reply and Like counts. Unfortunately, I was unable to gain any utility from collecting data such as the place object of each tweet, however this is not a major issue as this functionality can be added in the future once I have been granted better access rights to Twitter's API. With the ability to scrape more tweets per search, I believe the limited results one can obtain for the place data of Tweets could be negated and prove useful for the project with the visualisation of the globe and the trends of sentiments in different regions.

7.3 Front End

In the scope of the front end of my project, I am very pleased with the results obtained for my project. I successfully managed to implement a clean, responsive, reactive and professional looking application, which in my eyes looks better than the other projects out there (such as FinTwit and Hashtagify).

7.4 Summary

I believe that the results I was able to attain for this project show that I was able to successfully able to implement the functionality for the application idea I came up with in the conceptualisation stage of my project. I have been able to make a financial tool that, reflected by the user testing survey results I collected showed that financial analysts and others working in the field of finance, found utility from.

I believe that my project contributes to the field of Twitter Sentiment Analysis by providing a tool which is simple to use, understand and navigate (which in my mind, is a big improvement from the options available from FinTwit and Hashtagify). Furthermore, the tool is currently free to use, without any annoying barriers to entry such as account registration, etc. Overall, I am pleased as I believe I have created the most user friendly application out of those I have listed previously.

I believe that the results I have collected for this project fit in closely with similar projects such as the two I mentioned previously. They are all very similar in terms of utility, and as I said before, what makes my project better is its usability and simplicity.

The success of my project is again reflected by the success I had fulfilling the testable requirements I set for my application. These results can be referred to in the results chapter of this report. The minimum requirement for each of my testable requirements (which inspired the questions in the user reports) of 50 percent of participants needing to agree with questions within the likelihood range of 3 to 5 was larger by an average of 41.8 percent (relative to our participant group size of 30 participants working in finance and 51 overall).

Chapter 8

Limitations and Future Works

8.1 Machine Learning Model

Although I am quite satisfied by what I have been able to achieve in the scope of our machine learning model, I do believe that some areas could have been improved.

There are two limitations which I can easily outline now: time and data constraints.

I believe that if I had a few more weeks of implementation my project could have improved significantly, and in regards to the machine learning model in my application's back end, I believe that I would have been able to improve the scope of the model tweaking and testing I did. I believe that tweaking the hyper parameters of the logistic regression model used for making my machine learning algorithm would have allowed me to find higher prediction accuracy's.

I believe that projects which involve the use of machine learning models can always be improved with a larger data set, and one limitation which comes to mind immediately when thinking about my machine learning model is its data set. Although I believe that the quality of my data set was great, and the scale was also quite large, I am sure my project would have seen positive externalities to me increasing the size of this.

If I were to develop my project further, I would spend more time improving the testing and hyper parameter tweaking of my machine learning models, in the hope of achieving a greater sentiment prediction accuracy.

I believe that other routes of investigation could be opened up by this. An example might be that I could decide to create several machine learning algorithms for languages other than English, so that when I run the searchtweets function provided by the Twitter API, we can not only predict the sentiment of English text, but other languages which would be collected in tweets from our scraper (as the scraper, at our access level on twitter's development site doesn't provide functionality for specifying the language of tweets you would like to collect).

8.2 Twitter Scraper

In the development of my application, the twitter scraper and front end phases of my project's implementation proved to be those with the most limitations encountered.

For this section, the main limitations I encountered were in regards to the access rights provided to me by Twitter's API, and the frequency of tweets which had data regarding their location on the globe.

Firstly, the ”essential” API access level I had meant that I would only be able to use the Twitter API v2 methods, therefore the function I was required to use for scraping Tweets was called searchrecenttweets. At my access level, the number of tweets I could scope every 15 minutes was set to 450, leaving me with quite a lot of restriction when looking at the scope and size of results available to users of my application. Because of this limitation, I decided to restrict the number of tweets scraped from each hashtag search to 10 to remove any worries of the application running into errors once too many searches had been made in this time window.

Secondly, in regards to collecting the location data for our scraped Tweets, I found that during the implementation of this functionality, Tweets I was able to scrape showed that very few of the Tweets had any location data and this is because the users which published these Tweets likely disabled any location tracking for their Twitter app. In response to this issue, I decided to leave this piece of functionality out of my project for now as I would need to improve my access rank on the Twitter API platform before considering adding this to my project.

If I were to develop this project further, I would spend time leasing with the Twitter team in order to get either the ”Elevated” or ”Academic Research” access level and work on the previously mentioned areas for improvement.

I believe that combating these issues could add a very cool additional dimension to my project as it would allow me to add the World Map visualisation showed in the design of my project. This visualisation would allow users to see trends in popularity for hashtags across different global regions which could be very useful in implication to specific trades a user may be looking into. For example, if our user were looking to invest in healthcare in Asia, and found that a hashtag related to this field had many negative connotations in this regions tweets, perhaps they would use this information to reevaluate their decision or plans.

I also believe that in the future, I could improve my project further (once getting better access privileges from Twitter’s API) by adding the functionality for users to create their own accounts and be able to login, logout, etc. In their account page, I could then implement a personal ’library’ for our users where they ’favorite’ certain hashtags in order to keep track of sentiment changes over time.

Chapter 9

Conclusion

As highlighted by the text in the previous sections and the user surveys I have conducted, I have successfully implemented my application that serves as a tool for financial analysts to conduct early research into a trade they are interested in. With the combined analysis from my application and other tools for a more in depth analysis, I believe that my application can help steer portfolio managers and traders into making educated trades. The application's ease of use and understanding as well as simplicity stand out compared to other tools and its scope is also broad. Users have complete freedom with their searches and can get an analysis for any working hashtag representing stocks or underrepresented markets such as foreign exchanges, commodities, stocks and bonds. Furthermore, my application is not restrictive in regards to analysing only assets from specific markets such as those listed in America, and can be used globally, for assets listed in any market. The overall aim and utility for my application was a success. Each of my testable requirements were met and exceeded. The average proportion of my survey participants whom responded with high likelihoods (4 to 5, 'extremely') of my project supporting it's requirements was 51 percent.

As mentioned in the previous chapter, limitations and future works, there are many avenues of improvement which can be taken. My early research (detailed in the background chapter) on Ted Briscoe's work was very inspirational to my project's aims. Models which aid analysts in their research as opposed to those which try to do their work in forecasting market performances tend to offer a more positive utility. Knowing this steered me away from making a project which would predict the price fluctuations of assets using Twitter scraping, but instead pursue another route. This knowledge allowed for me to keep my project simple and the user surveys and overall results shown in this report reflect the positive affect it has had. I believe that my project and its concept has shown great promise and could be potentially worked on further to add the additional functionality I have mentioned previously. Additionally, I think that my project's target audience could also be expanded quite easily, such as people in the marketing industry. Furthermore, other social media sites could also be extended onto the application's scope whereby analysis could be done across chosen platforms.

Chapter 10

Bibliography

1. Bloomberg, https://www.bloomberg.com/professional/solution/bloomberg-terminal/
2. Financial Times, https://www.ft.com/
3. Refinitiv, https://www.refinitiv.com/
4. MSCI, https://www.msci.com/
5. Acuris, https://www.acuris.com/
6. Markit, https://ihsmarkit.com/index.html
7. Redd, https://www.reddintelligence.com/
8. SmartKarma, https://www.smartkarma.com/home/
9. Yahoo Finance, https://uk.finance.yahoo.com/
10. Hashtagify, https://hashtagify.me/hashtag/newyear
11. FinTwit, https://fintwit.ai/
12. Sian Gooding, and Ted Briscoe: "Active Learning for Financial Investment Reports", 2019, https://aclanthology.org/W19-6404.pdf
13. Young-Woo Seo, Joseph Giampapa and Katia Sycara: "Financial News Analysis for Intelligent Portfolio Management", 2004, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.3254&rep=rep1&type=pdf
14. Peter Suciu: "Business Owners Increasingly Rely on Social Media To Gather Customer Data", 2021, https://www.forbes.com/sites/petersuciu/2021/12/21/business-owners-increasingly-rely-on-social-media-to-gather-customer-data/?sh=536875bd1d31
15. Pandas, https://pandas.pydata.org/
16. Numpy, https://numpy.org/
17. Random, https://docs.python.org/3/library/random.html
18. Matplotlib, https://matplotlib.org/
19. WordCloud, https://amueller.github.io/word_cloud/
20. Regex, https://docs.python.org/3/library/re.html
21. Emoji, https://pypi.org/project/emoji/
22. Contractions, https://pypi.org/project/contractions/
23. NLTK, https://www.nltk.org/
24. Scikitlearn, https://scikit-learn.org/stable/#
25. Seaborn, https://seaborn.pydata.org/
26. Joblib, https://joblib.readthedocs.io/en/latest/
27. Python, https://www.python.org/

28. Sentiment140 dataset with 1.6 million tweets https://www.kaggle.com/kazanova/sentiment140
29. Twitter Developer Site, https://developer.twitter.com/en
30. React.js, https://reactjs.org/
31. Opensea, https://opensea.io/
32. FreeLogoDesign, https://editor.freelogodesign.org/
33. YouTube Project Demonstration, https://www.youtube.com/watch?v=rAEGGX3t76I&t=16s

Chapter 11

Appendix

Designs, File Structure and Project Timeline

Figure 11.1: Home Page Design - No Content

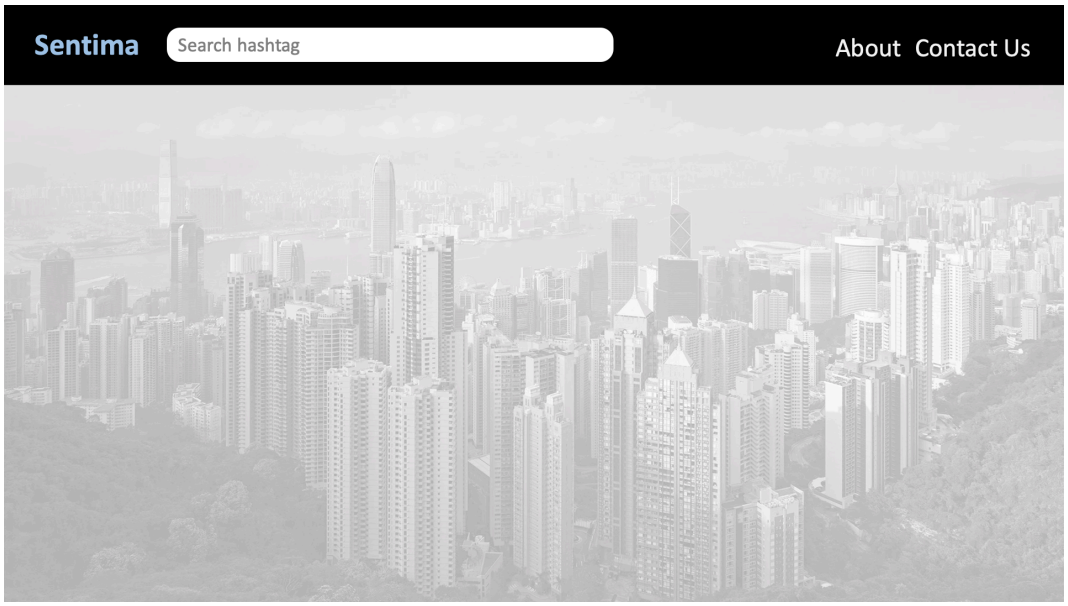


Figure 11.2: Home Page Design

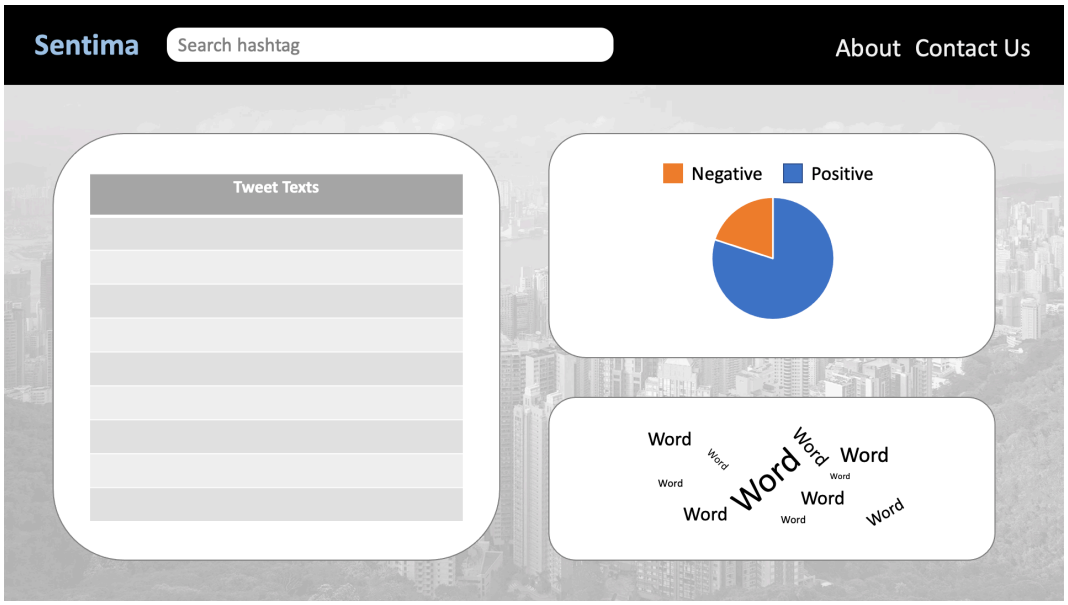


Figure 11.3: Home Page Design - With Ternary Graph

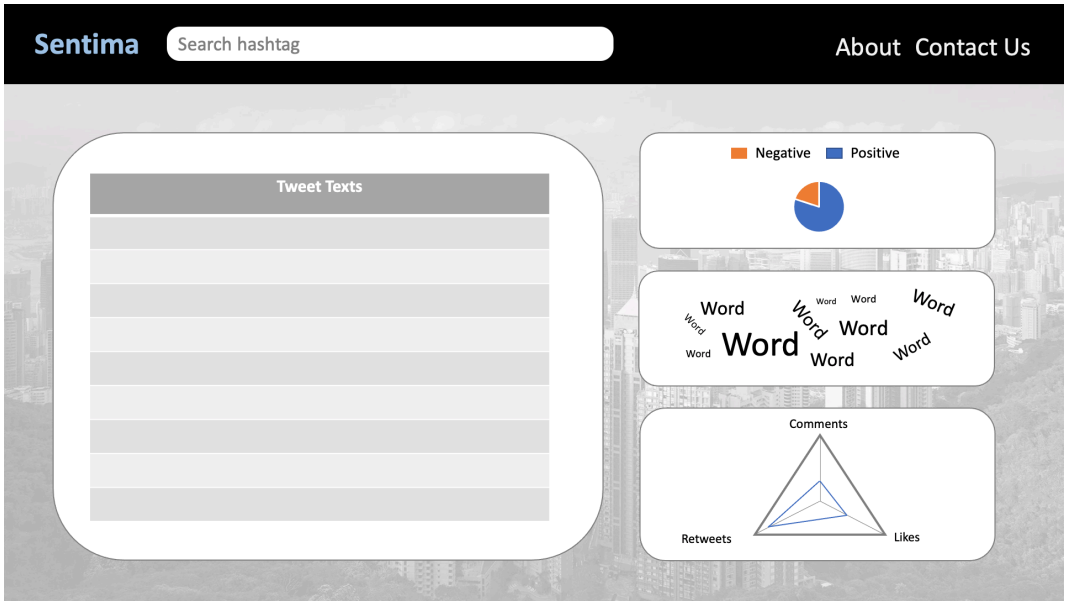


Figure 11.4: Home Page Design - With Ternary Graph and World Map Visualisation

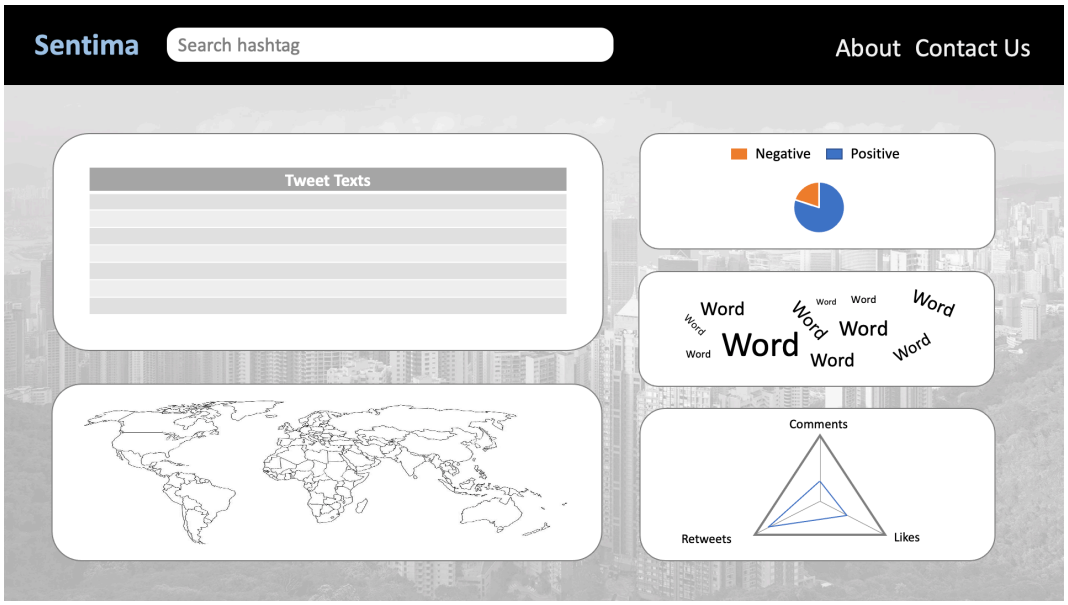


Figure 11.5: About Page Design

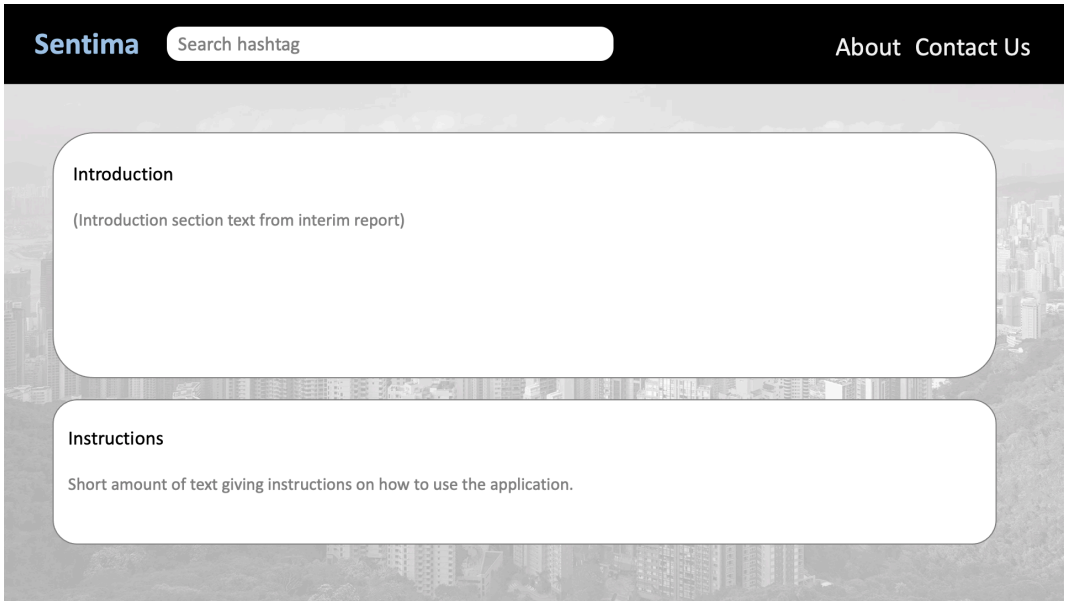


Figure 11.6: Contact Us Page Design

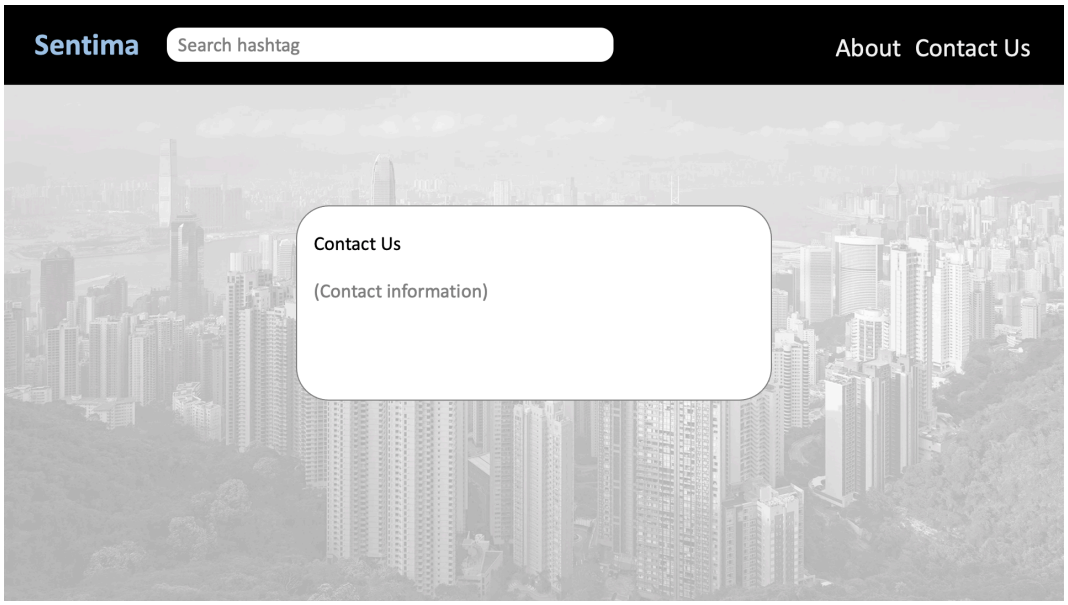


Figure 11.7: Application File Structure

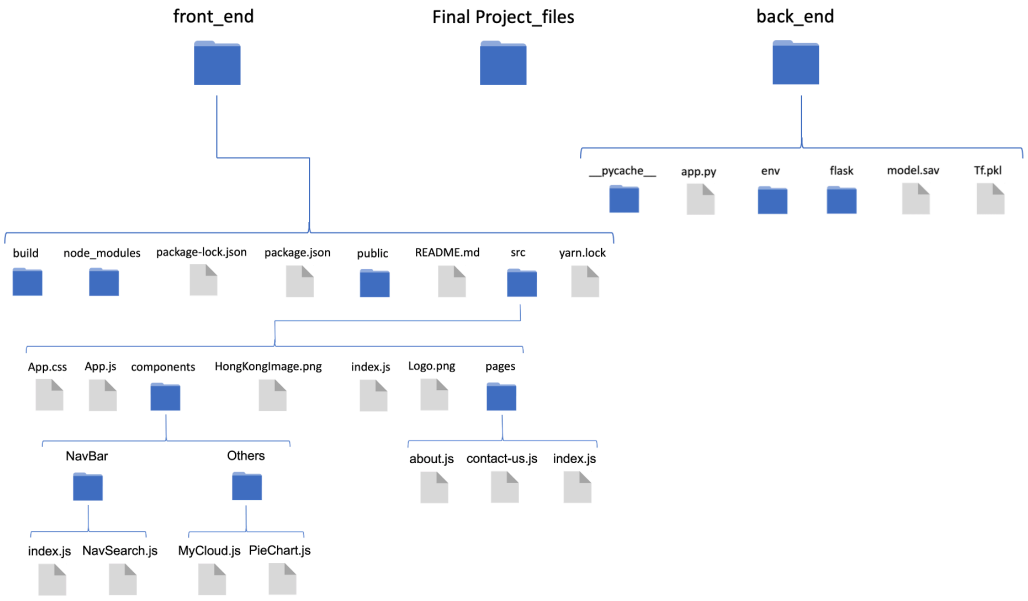


Figure 11.8: Project Timeline

Weeks Left	Date	Aim	Deadlines
29	22/10/2021	Planning - Complete Model Specification Document	Project Specification
		Stage 1 Sentiment Prediction Model	
28	29/10/2021	Data Collection – download dataset of tweets and visualise the data collected (check ratio of positive and negative tweet sentiments, etc)	
27	5/11/2021	Text Normalisation – functionality removing unnecessary text (user tags, repeated characters, and punctuation) to clean up tweet data. Simplify text further by deciding between the use of stemming and lemmatisation.	
26	12/11/2021	Text Vectorisation – conduct dataset pre-processing. Create three different methods using the different formulas, Positive/Negative Word Frequencies, Bag-of-Words and TF-IDF.	
25	19/12/2021	Sentiment Analysis – Separate dataset into test and training sets and train our model.	Prototype/proof of concept
24	26/12/2021	Tweak model to ensure highest accuracy possible.	
23	3/1/2022	Measure Performance and create pipeline – Create a final method for tweet sentiment prediction.	
		Stage 1 Twitter Scraper	
22	10/1/2022	Final preparation and research on Tweepy package	
21	17/1/2022	Implementation of scraper	
20	24/1/2022	"	
19	31/1/2022	Final tweaks for scraper	
		Stage 1 Combine tweet prediction model with scraper	
18	7/2/2022	Combine functionality	
17	14/2/2022	Testing	
		Stage 2 Web Page User Interface	
16	21/2/2022	Design	Interim Report
15	28/2/2022	Begin implementation - create react.js containers for navigation bar, list to display scraped tweets before and after text normalisation, containers for graphical representation and footer.	
14	4/3/2022	"	
13	11/3/2022	Base functionality - Create search bar which allows users to enter hashtag they would like to analyse from Twitter.	
12	18/3/2022	Complete implementation of search bar	
11	25/3/2022	Work on how to visualise output of sentiment prediction model	
10	4/4/2022	" Presentation – begin work on	
9	11/4/2022	"	
8	18/4/2022	Refine front end design to reach visual goal	Presentation
7	25/4/2022	Implement last few items	
6	1/5/2022	"	
5	8/5/2022	Final tweaks of web page	
4	15/5/2022	Testing	
		Final Fixes	
3	22/5/2022		
2	29/5/2022		
1	6/6/2022		Final hand-in

Machine Learning Model

Figure 11.9: Data Frame Creation

```
In [3]: df = pd.read_csv("../models_data/training.1600000.processed.noemoticon.csv", header=None)
        df.sample(10)
```

Out[3]:

	0	1	2	3	4	5
1309505	4	2013084147	Tue Jun 02 21:53:08 PDT 2009	NO_QUERY	Dianneav	@RevzNexus Thanks for the comment
1176563	4	1981279296	Sun May 31 08:49:09 PDT 2009	NO_QUERY	EstJesusNoWhere	@jakuba16 I have new ideas for my new friends ...
795563	0	2327418527	Thu Jun 25 08:19:20 PDT 2009	NO_QUERY	missnani	my booger's birthday is this weekend and im go...
723723	0	2261870687	Sat Jun 20 21:29:22 PDT 2009	NO_QUERY	_LiLLiAN	getting ready to go out to eat w/ immi&pet...
178268	0	1965928962	Fri May 29 16:48:15 PDT 2009	NO_QUERY	TomCayman	Ecaytrade people who said you'd come by stuff ...
874998	4	1680241098	Sat May 02 11:13:37 PDT 2009	NO_QUERY	dancedani27	@WalterFuckinLee my mom said to get your ass o...
1339617	4	2018689372	Wed Jun 03 10:21:26 PDT 2009	NO_QUERY	PrettyOddAshley	Just explaned to Juanita the wonders of 'Littl...
893127	4	1691744409	Sun May 03 18:31:09 PDT 2009	NO_QUERY	rashaancruze	@GeishaLee oooh thank you! i have heard of it ...
636922	0	2234051478	Thu Jun 18 22:20:38 PDT 2009	NO_QUERY	kimfalloon	@TeganAshton I think I'm incapable of NOT gett...
863733	4	1677153784	Sat May 02 00:41:39 PDT 2009	NO_QUERY	hinshelm	Well, I'm here Everone can relax now ;)

Figure 11.10: Data Frame Headers

```
In [4]: headerList = ["Target", "ID", "Date", "Flag", "User", "Text"]
        df.to_csv("data_table.csv", header=headerList, index=False)
        df = pd.read_csv("data_table.csv")
        df.sample(10)
```

Out[4]:

	Target	ID	Date	Flag	User	Text
247846	0	1982567016	Sun May 31 11:28:01 PDT 2009	NO_QUERY	QueenVirus	@MATTHARDYBRAND I love that palce but they don...
237102	0	1980134058	Sun May 31 05:48:15 PDT 2009	NO_QUERY	ZomBieKween	I seriously need help im so sorry I'aurance i ...
1526598	4	2177052021	Mon Jun 15 05:20:49 PDT 2009	NO_QUERY	josephdexter	@CandiCaine At work, getting caught up on pape...
954327	4	1824823470	Sun May 17 03:05:34 PDT 2009	NO_QUERY	imbea	@KayTeePerry thank you for following meee! i l...
170789	0	1963015311	Fri May 29 12:04:26 PDT 2009	NO_QUERY	sabraswell	So glad its friday! Only 1 week of schol left....
1090740	4	1969827093	Sat May 30 01:22:32 PDT 2009	NO_QUERY	maxamphoto	@caterham7 haha no... but it really does sound...
92790	0	1760205350	Sun May 10 20:55:34 PDT 2009	NO_QUERY	amy_tran	my neighborhood is starting to get really ghet...
1432984	4	2060365999	Sat Jun 06 18:46:15 PDT 2009	NO_QUERY	718pm	@NigroDicaprio bout to take a nap. when I awak...
698075	0	2254010923	Sat Jun 20 09:05:26 PDT 2009	NO_QUERY	saffyre9	@cyandle I totally am... cold & rainy here...
1219788	4	1989982198	Mon Jun 01 03:29:09 PDT 2009	NO_QUERY	illbecavalier	@myelle911 did you trim it? haha. biggest surp...

Figure 11.11: Column Removal

```
In [5]: df = df.drop(columns="Flag")
        df.sample(10)
```

Out[5]:

	Target	ID	Date	User	Text
206182	0	1973063393	Sat May 30 10:24:43 PDT 2009	suckerpunch405	Band practice is being delayed due to church y...
130435	0	1835311441	Mon May 18 05:41:28 PDT 2009	Amy113456	I got sent home from School. Nearly Fainted in...
1563796	4	2187157664	Mon Jun 15 19:52:28 PDT 2009	luscioustonya	@CJWRIGHTXXX I see ur site is redesigned nice ...
1209120	4	1988896004	Sun May 31 23:41:55 PDT 2009	unicyclistjoe	is home after a long weekend. Got to ride in t...
205868	0	1972984989	Sat May 30 10:15:26 PDT 2009	meagansue	Going to watch the little sis ice skate. Ma...
153000	0	1932865063	Tue May 26 22:24:09 PDT 2009	sarahklee	now it's, "i want to do it by myself, no ...
574704	0	2210543365	Wed Jun 17 11:42:55 PDT 2009	Ginnyinthepants	twitter hates me. it's really upsetting i thi...
769271	0	2301302730	Tue Jun 23 15:15:46 PDT 2009	RedBessBonney	hmmm.. missed some spots with the sunscreen th...
1346509	4	2044313822	Fri Jun 05 09:10:37 PDT 2009	Owl311	@eveningvicar hope that helps my friend.... an...
610088	0	2223991717	Thu Jun 18 08:57:34 PDT 2009	atr_hugo	Ford asked for an embargo on Taurus SHO review...

Figure 11.12: Binary Sentiments

```
In [6]: df["Target"].replace({4:1}, inplace=True)
        df.sample(10)
```

Out[6]:

	Target	ID	Date	User	Text
1440768	1	2061714443	Sat Jun 06 21:27:16 PDT 2009	StephanieMcFly	@tommcfly aww good night starboy love you!
766238	0	2299796685	Tue Jun 23 13:21:26 PDT 2009	Draven	@drsuzy that's sad.
1062070	1	1964018339	Fri May 29 13:36:00 PDT 2009	trudietess	I get to spend a week with @casualtybrand ! Ho...
1204602	1	1986331724	Sun May 31 18:41:02 PDT 2009	celineyap	@jeffro88 I would wish it was "Monday ret...
727924	0	2262942259	Sat Jun 20 23:34:54 PDT 2009	ashleysusername	@Nicole274 aww ima kick his ass... Ash
541300	0	2200042740	Tue Jun 16 18:21:31 PDT 2009	xhozt	@GadgetVirtuoso ah not on win mob yet though ...
940557	1	1793984801	Thu May 14 04:37:04 PDT 2009	SilkCharm	@gctim well, none of that amazing dialogue was...
735197	0	2264728052	Sun Jun 21 04:46:11 PDT 2009	mrated	@lizawhitten It's fathers day? Not here. .....
210725	0	1974310858	Sat May 30 12:52:09 PDT 2009	mCamz	Jordy told me to tweet about him, so I am. He ...
1009687	1	1880924235	Fri May 22 02:59:20 PDT 2009	grkboi	@ikathryn have a good trip. Hope the traffic i...

Figure 11.13: Pie Chart of Data Frame Sentiments

```
In [11]: sentiment_count = df["Target"].value_counts()
         plt.pie(sentiment_count, labels=sentiment_count.index, autopct='%1.1f%%', shadow=True, startangle=90)
         plt.show()
```

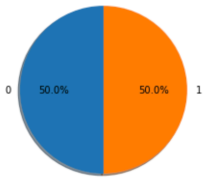


Figure 11.14: Outputted Sentiment Value Counts

```
In [12]: # printing the count of positve and negative tweets in our dataframe
         print("Number of + tweets: {}".format(df[df["Target"]==1].count()[0]))
         print("Number of - tweets: {}".format(df[df["Target"]==0].count()[0]))
```

```
Number of + tweets: 800000
Number of - tweets: 800000
```

Figure 11.15: Word Cloud - Positive Sentiments in Data Frame

```
In [14]: pos_tweets = df[df["Target"]==1]
         txt = " ".join(tweet.lower() for tweet in pos_tweets["Text"])
         wordcloud = WordCloud().generate(txt)
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis("off")
         plt.show()
```

Figure 11.16: Word Cloud - Negative Sentiments in Data Frame

```
In [15]: neg_tweets = df[df["Target"]==0]
         txt = " ".join(tweet.lower() for tweet in neg_tweets["Text"])
         wordcloud = WordCloud().generate(txt)
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis("off")
         plt.show()
```

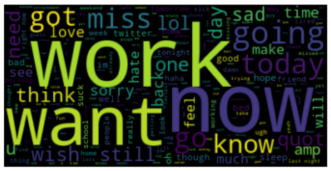


Figure 11.17: Remove Retweet Tags Method

Example tweet we will be normalising

We will be using the below tweet to apply our new functionality of text normalisation on as our testing.

```
In [17]: tweet = "RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #Science"
```

Retweet Tag

From what I have seen online it seems that the text as shown in the above tweet example can contain text such as "RT". For now we can keep this functionality, however if we find in the future that when we do not get an instances of this such as when we use the scraper, we may consider removing this sub-section.

The cell below creates a method which replaces the retweet tag from tweets with a default replacement string. Currently, we have set the replacement to be an empty string, so the text just gets removed.

```
In [18]: def replace_retweet_tag(tweet, default_replace=""):
             tweet = re.sub('RT\s+', default_replace, tweet)
             return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

```
In [19]: print("Unprocessed tweet: {}".format(tweet))
         print("Processed tweet: {}".format(replace_retweet_tag(tweet)))
```

```
Unprocessed tweet: RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #
Science
Processed tweet: @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #Scien
ce
```

Figure 11.18: Remove User Tags Method

User Tag

The cell below creates a method which allows us to replace user tag text with a default replacement string. At the moment, the function makes it so that the text is replaced with an empty string (so it is essentially removed).

```
In [20]: def replace_user_tag(tweet, default_replace=""):
             tweet = re.sub('\B@\w+', default_replace, tweet)
             return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

```
In [21]: print("Unprocessed tweet: {}".format(tweet))
         print("Processed tweet: {}".format(replace_user_tag(tweet)))
```

```
Unprocessed tweet: RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #
Science
Processed tweet: RT  How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #Science
```

Figure 11.19: Replace Emojis Method

Emojis

The cell below creates a method which allows us to replace emojis with their corresponding text.

For this functionality, we will need to import the emoji library (shown in the cell below).

In [22]:
```
import emoji
```

The below cell contains our method which we will use to replace emojis with their correspondnig text (with the help of the demojize function provided to us by the emoji package we imported).

In [23]:
```
def replace_emoji(tweet):
    tweet = emoji.demojize(tweet)
    return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

In [24]:
```
print("Unprocessed tweet: {}".format(tweet))
print("Processed tweet: {}".format(replace_emoji(tweet)))
```

```
Unprocessed tweet: RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #
Science
Processed tweet: RT @user How are you? :smiling_face_with_halo: I found a website that reminded me of you https://ww
w.space.com/ #Space #Science
```

Figure 11.20: Remove URLs Method

URLs

The cell below creates a method which allows us to replace url links with a default replacement parameter. At the moment, I have set the default_replace parameter to be an empty string so that the url is removed and isn't replaced with any other text.

In [25]:
```
def replace_url_link(tweet, default_replace=""):
    tweet = re.sub('(http|https):\/\S+', default_replace, tweet)
    return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

In [26]:
```
print("Unprocessed tweet: {}".format(tweet))
print("Processed tweet: {}".format(replace_url_link(tweet)))
```

```
Unprocessed tweet: RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #
Science
Processed tweet: RT @user How are you? 😇 I found a website that reminded me of you  #Space #Science
```

Figure 11.21: Remove Hashtag Characters Method

Hashtags

The cell below creates a function which replaces hashtags with a default replacement value. At the moment this parameter is set to be an empty string (like the previous methods). In the future, however I plan on working on this function as hashtags will be an important part of my project.

In [27]:
```
def replace_hashtag(tweet, default_replace=""):
    tweet = re.sub('#+', default_replace, tweet)
    return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

In [28]:
```
print("Unprocessed tweet: {}".format(tweet))
print("Processed tweet: {}".format(replace_hashtag(tweet)))
```

```
Unprocessed tweet: RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ #Space #
Science
Processed tweet: RT @user How are you? 😇 I found a website that reminded me of you https://www.space.com/ Space Scie
nce
```

Figure 11.22: Replace Uppercase Characters Method

New tweet example

We will be using this example tweet to normalise the linguistic features shown below.

```
In [29]: tweet = "GOOOOOOOOOOGLE!!! Why aren't you working?"
```

Capital Letters

The cell below contains code which creates a function that converts uppercase characters to lowercase. This will simplify the work the machine learning algorithm will need to do when predicting sentiments as words will normally be case sensative.

```
In [30]: def to_lowercase(tweet):
             tweet = tweet.lower();
             return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

```
In [31]: print("Unprocessed tweet: {}".format(tweet))
         print("Processed tweet: {}".format(to_lowercase(tweet)))
```

```
Unprocessed tweet: GOOOOOOOOOOGLE!!! Why aren't you working?
Processed tweet: gooooooooooogle!!! why aren't you working?
```

Figure 11.23: Remove Character Repetition Method

Character repetition

The cell below creates a function which we will use to replace repeating characters. This means that whenever a specific character is in a concatenation longer than 2 characters long, the extra characters will be removed.

For example, if we have the word "Hellllllllo". There are 8 ls in concatenation here, so we will remove the uneccessary 6 so that we are left with "Hello".

```
In [32]: def char_repetition(tweet):
             tweet = re.sub(r'(.)\1+', r'\1\1', tweet)
             return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

```
In [33]: print("Unprocessed tweet: {}".format(tweet))
         print("Processed tweet: {}".format(char_repetition(tweet)))
```

```
Unprocessed tweet: GOOOOOOOOOOGLE!!! Why aren't you working?
Processed tweet: GOOGLE!! Why aren't you working?
```

Figure 11.24: Remove Punctuation Repetition Method

Punctuation repetition

The cell below contains a function which removes repeating punctuation symbols. This works very much like the char_repetition function, however we will make it so that we only end up with one of the symbols, and not two.

```
In [34]: def punct_repetition(tweet, default_replace=""):
             tweet = re.sub('[\?\.\!]+(?=[\?\.\!])', default_replace, tweet)
             return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

```
In [35]: print("Unprocessed tweet: {}".format(tweet))
         print("Processed tweet: {}".format(punct_repetition(tweet)))
```

```
Unprocessed tweet: GOOOOOOOOOOGLE!!! Why aren't you working?
Processed tweet: GOOOOOOOOOOGLE! Why aren't you working?
```

Figure 11.25: Remove Word Contractions Method

The cell below creates a method which converts contractions into the two words that they are made up of.

```
In [39]: def convert_contractions(tweet):
           tweet = contractions.fix(tweet)
           return tweet
```

The below cell is used to display the changes of the test tweet when we run the function on it.

```
In [40]: print("Unprocessed tweet: {}".format(tweet))
         print("Processed tweet: {}".format(convert_contractions(tweet)))
```

```
Unprocessed tweet: GOOOOOOOOOOGLE!!! Why aren't you working?
Processed tweet: GOOOOOOOOOOGLE!!! Why are not you working?
```

Figure 11.26: Tokenization Method

In [48]:
```
def tokenize_function(tweet,
                      keep_punct = False,
                      keep_alnum = False,
                      keep_stop = False):
    token_list = word_tokenize(tweet)

    if not keep_punct:
        token_list = [token for token in token_list
                      if token not in string.punctuation]
    if not keep_alnum:
        token_list = [token for token in token_list if token.isalpha()]
    if not keep_stop:
        stop_words = set(stopwords.words('english'))
        stop_words.discard('not')
        token_list = [token for token in token_list if not token in stop_words]
    return token_list
```

Example tweet

New example tweet that we will be testing our custom tokenization function on:

In [49]:
```
tweet = "hello there, how are you?"
```

The below cell shows the different outputs we can get from the different forms of tokenization we do when altering the parameters of our tokenize_function.

In [50]:
```
print("Tweet tokens: {}".format(tokenize_function(tweet,
                                                  keep_punct=True,
                                                  keep_alnum=True,
                                                  keep_stop=True)))
print("Tweet tokens: {}".format(tokenize_function(tweet, keep_stop=True)))
print("Tweet tokens: {}".format(tokenize_function(tweet, keep_alnum=True)))
```

```
Tweet tokens: ['hello', 'there', ',', 'how', 'are', 'you', '?']
Tweet tokens: ['hello', 'there', 'how', 'are', 'you']
Tweet tokens: ['hello']
```

Figure 11.27: Stemming Method

Token examples

List of examples we will test stemming on:

In [52]:
```
tokens = ["director", "directed", "direct", "directing"]
```

The below cell creates stemming objects we will be using in the future.

In [53]:
```
porter_stemmer = PorterStemmer()
lancaster_stemmer = LancasterStemmer()
snoball_stemmer = SnowballStemmer('english')
```

The below cell creates a method which we will be using to stem tokens. The "tokens" parameter is where we will be inputing the tokens to be processed and the "stemmer" is where we will be inputing the specific stemmer to be used (from the above cell).

In [54]:
```
def stem_tokens(tokens, stemmer):
    token_list = []
    for token in tokens:
        token_list.append(stemmer.stem(token))
    return token_list
```

The below cell is used to display the different outputs we get from running the different stemming options on our token examples.

In [55]:
```
print("Porter stems: {}".format(stem_tokens(tokens, porter_stemmer)))
print("Lancaster stems: {}".format(stem_tokens(tokens, lancaster_stemmer)))
print("Snowball stems: {}".format(stem_tokens(tokens, snoball_stemmer)))
```

```
Porter stems: ['director', 'direct', 'direct', 'direct']
Lancaster stems: ['direct', 'direct', 'direct', 'direct']
Snowball stems: ['director', 'direct', 'direct', 'direct']
```

Figure 11.28: Final Text Normalisation Method

Test tweet

Test Tweet (of a longer length) to be processed:

In [66]:
```
tweet = r"""RT @elonmusk : you are so cooooooool!
I Can't believe you are REal!!! 😎  ...
I have been saving up to buy one of your Teslas!
Everyone, check out his work at https://www.tesla.com/en_gb and LET ME KNOW WHAT YOU THINK OF HIM!!!
#ELONMUSK #Elon #Legend #Revolutionary"""
```

The cell below contains a custom process_tweet function that can be used to process tweets end-to-end and normalise the test within them.

In [67]:
```
def process_tweet(tweet, verbose=False):
    if verbose: print("Original tweet: \n{}".format(tweet))

    ## Twitter Features
    tweet = replace_retweet_tag(tweet) # replace retweet
    tweet = replace_user_tag(tweet, "") # replace user tag
    tweet = replace_url_link(tweet) # replace url
    tweet = replace_hashtag(tweet) # replace hashtag
    if verbose: print("Post Twitter processing tweet:\n{}".format(tweet))

    ## Word Features
    tweet = to_lowercase(tweet) # lower case
    tweet = convert_contractions(tweet) # replace contractions
    tweet = punct_repetition(tweet) # replace punctuation repetition
    tweet = char_repetition(tweet) # replace word repetition
    tweet = replace_emoji(tweet) # replace emojis
    if verbose: print("Post Word processing tweet:\n{}".format(tweet))

    ## Tokenization & Stemming
    tokens = tokenize_function(tweet, keep_alnum=False, keep_stop=False) # tokenize
    stemmer = SnowballStemmer("english") # define stemmer
    stem = stem_tokens(tokens, stemmer) # stem tokens
    return stem
```

The cell below tests the process_tweet function on our test tweet string. And outputs the results - there will be a big difference!

In [68]:
```
print("Oringinal tweet: \n{}".format(tweet))
print("\nProcessed tweet: \n{}".format(process_tweet(tweet, verbose=False)))
```

```
Oringinal tweet:
RT @elonmusk : you are so cooooooool!
I Can't believe you are REal!!! 😎  ...
I have been saving up to buy one of your Teslas!
Everyone, check out his work at https://www.tesla.com/en_gb and LET ME KNOW WHAT YOU THINK OF HIM!!! #ELONMUSK #Elon
#Legend #Revolutionary

Processed tweet:
['cool', 'not', 'believ', 'real', 'save', 'buy', 'one', 'tesla', 'everyon', 'check', 'work', 'let', 'know', 'think',
'elonmusk', 'elon', 'legend', 'revolutionari']
```

Figure 11.29: New Data Frame Column

In [71]:
```
df["Tokens"] = df["Text"].apply(process_tweet)
# add a new column to the data frame called 'tokens'
df.head(10)
```

Out[71]:

	Target	ID	Date	User	Text	Tokens
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...	[aww, bummer, shoulda, got, david, carr, third...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	scotthamilton	is upset that he can't update his Facebook by ...	[upset, not, updat, facebook, text, might, cri...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	mattycus	@Kenichan I dived many times for the ball. Man...	[dive, mani, time, ball, manag, save, rest, go...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	ElleCTF	my whole body feels itchy and like its on fire	[whole, bodi, feel, itchi, like, fire]
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	Karoli	@nationwideclass no, it's not behaving at all....	[not, behav, i, mad, not, see]
5	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	joy_wolf	@Kwesidei not the whole crew	[not, whole, crew]
6	0	1467811592	Mon Apr 06 22:20:03 PDT 2009	mybirch	Need a hug	[need, hug]
7	0	1467811594	Mon Apr 06 22:20:03 PDT 2009	coZZ	@LOLTrish hey long time no see! Yes.. Rains a...	[hey, long, time, see, yes, rain, bit, bit, lo...
8	0	1467811795	Mon Apr 06 22:20:05 PDT 2009	2Hood4Hollywood	@Tatiana_K nope they didn't have it	[nope, not]
9	0	1467812025	Mon Apr 06 22:20:09 PDT 2009	mimismo	@twittera que me muera ?	[que, muera]

Figure 11.30: X and y Lists

The cell below converts dataframe into two lists: one for the tweet tokens (X) and one for the tweet sentiment/target (y).

In [72]:
```
X = df["Tokens"].tolist()
y = df["Target"].tolist()
```

The below cell prints the first 10 elements in the new X and y lists.

In [73]:
```
print("Array of token objects: \n{}".format(X[0:10]))
print("\nArray of sentiments corresponding to the above tokens: \n{}".format(y[0:10]))
```

```
Array of token objects:
[['aww', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day'], ['upset', 'not', 'updat', 'facebook', 'text',
'might', 'cri', 'result', 'school', 'today', 'also', 'blah'], ['dive', 'mani', 'time', 'ball', 'manag', 'save', 'res
t', 'go', 'bound'], ['whole', 'bodi', 'feel', 'itchi', 'like', 'fire'], ['not', 'behav', 'i', 'mad', 'not', 'see'],
['not', 'whole', 'crew'], ['need', 'hug'], ['hey', 'long', 'time', 'see', 'yes', 'rain', 'bit', 'bit', 'lol', 'i', 'f
ine', 'thank'], ['nope', 'not'], ['que', 'muera']]

Array of sentiments corresponding to the above tokens:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 11.31: Frequency Dictionary Method

Positive/Negative Frequency

In this section, we will be using the data type, corpus. A corpus is a language resource consistin of a large structured set of texts. Corpuses are used for statistical analysis and hypothesis testing.

Below, we will be using the test corpus to continue our working.

In [74]:
```
corpus = [["i", "love", "listening", "to", "music"],
          ["i", "miss", "the", "old", "days"],
          ["i", "love", "you"],
          ["she", "is", "so", "sad"],
          ["you", "laughed", "a", "lot", "last", "night"],
          ["my", "watch", "was", "stolen"]]

sentiment = [1, 0, 1, 0, 1, 0]
```

Creating a build_freqs function which will be used to build a dictionary with the word sentiment as index and count of occurence as value

In [75]:
```
def build_freqs(tweet_list, sentiment_list):
    freqs = {}
    for tweet, sentiment in zip(tweet_list, sentiment_list):
        for word in tweet:
            pair = (word, sentiment)
            if pair in freqs:
                freqs[pair] += 1
            else:
                freqs[pair] = 1
    return freqs
```

Building the frequency dictionary on the test corpus by passing it into the above function

In [76]:
```
freqs = build_freqs(corpus, sentiment)
```

In [77]:
```
print(freqs)
```

```
{('i', 1): 2, ('love', 1): 2, ('listening', 1): 1, ('to', 1): 1, ('music', 1): 1, ('i', 0): 1, ('miss', 0): 1, ('th
e', 0): 1, ('old', 0): 1, ('days', 0): 1, ('you', 1): 2, ('she', 0): 1, ('is', 0): 1, ('so', 0): 1, ('sad', 0): 1,
('laughed', 1): 1, ('a', 1): 1, ('lot', 1): 1, ('last', 1): 1, ('night', 1): 1, ('my', 0): 1, ('watch', 0): 1, ('wa
s', 0): 1, ('stolen', 0): 1}
```

Build the frequency dictionary on the entire data frame using the function we created

In [78]:
```
freqs_all = build_freqs(X, y)
```

To see if this has worked, lets return the frequency of some words

In [79]:
```
print("Frequency of 'love' in +ve tweets: {}".format(freqs_all[("love", 1)]))
print("Frequency of 'love' in -ve tweets: {}".format(freqs_all[("love", 0)]))
print("\nFrequency of 'hate' in +ve tweets: {}".format(freqs_all[("hate", 1)]))
print("Frequency of 'hate' in -ve tweets: {}".format(freqs_all[("hate", 0)]))
```

```
Frequency of 'love' in +ve tweets: 62237
Frequency of 'love' in -ve tweets: 22197

Frequency of 'hate' in +ve tweets: 3037
Frequency of 'hate' in -ve tweets: 19808
```

Create a tweet_to_freqs function used to convert tweets to a 2-d array using the frequency dictionary

In [80]:
```
def tweet_to_freq(tweet, freqs):
    x = np.zeros((2,))
    for word in tweet:
        if (word, 1) in freqs:
            x[0] += freqs[(word, 1)]
        if (word, 0) in freqs:
            x[1] += freqs[(word, 0)]
    return x
```

Print the 2-d vector by using the tweet_to_freqs function and the corpus dictionary

In [81]:
```
print(tweet_to_freq(["i", "love", "listening", "to", "music"], freqs))
```

```
[7. 1.]
```

The above output shows us the number of times the words in this tweet have been used in tweets of positive sentiment and negative sentiment.

- 'i' - +ve : 2, -ve : 1
- 'love' - +ve : 2, -ve : 0
- 'listening' - +ve : 1, -ve : 0
- 'to' - +ve : 1, -ve : 0
- 'music' - +ve : 1, -ve : 0

So, the number corresponding to the positive tally is 2 + 2 + 1 + 1 + 1 = 7 and the number corresponding to the negative tally is 1 + 0 + 0 + 0 + 0 = 1. Therefore, we get the output vector of [7. 1.].

Now, we will print the 2-d vector by using the tweet_to_freq function and the dataset dictionary

In [82]:
```
print(tweet_to_freq(["i", "love", "listening", "to", "music"], freqs_all))
```

```
[172858. 156012.]
```

Figure 11.32: Visualisation of Words and Their Vectors

In [83]:
```
fig, ax = plt.subplots(figsize = (8, 8))

word1 = "happi"
word2 = "sad"

def word_features(word, freqs):
    x = np.zeros((2,))
    if (word, 1) in freqs:
        x[0] = np.log(freqs[(word, 1)] + 1)
#         print(x[0])
    if (word, 0) in freqs:
        x[1] = np.log(freqs[(word, 0)] + 1)
#         print(x[1])
    return x

x_axis = [word_features(word, freqs_all)[0] for word in [word1, word2]]
y_axis = [word_features(word, freqs_all)[1] for word in [word1, word2]]

ax.scatter(x_axis, y_axis)

# label axes
plt.xlabel("Log Positive count")
plt.ylabel("Log Negative count")

ax.plot([0, 9], [0, 9], color = 'red') # plots the red line
plt.text(x_axis[0], y_axis[0], word1) # labels word1 scatter point
plt.text(x_axis[1], y_axis[1], word2) # labels word2 scatter point
plt.show()
```

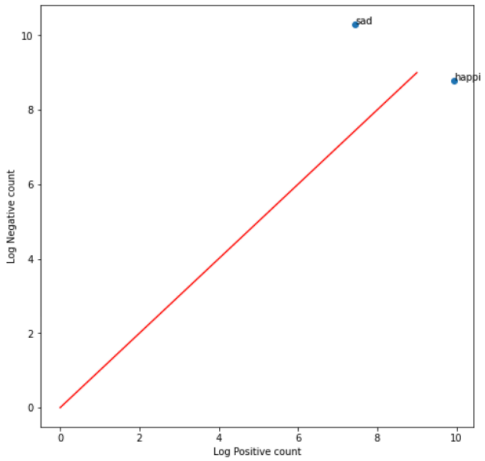


Figure 11.33: fit cv Method

Bag of Words

For this section, we will continue using the test corpus we made earlier.

Import CountVectorizer from Scikit-learn library

In [84]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

Create a fit_cv function used to build the Bag-of-Words vectorizer with the test corpus

In [85]:
```
def fit_cv(tweet_corpus):
    cv_vect = CountVectorizer(tokenizer=lambda x: x,
                              preprocessor=lambda x: x)
    # define vectorizer variable as a count vectorizer from scikitlearn
    # there are 2 parameters to add though. The count vectorizer is usually used to transfer over raw tweets,
    # thus it incorporates its own tokeniser and preprocessing utilities.
    # we already created a specific tokenizer, so we can bypass this by using a lambda on the tokenizer and
    # preporcessor parameters.
    # this way, CountVectoriser will be fooled and use the preprocessed sentence we feed into it.
    cv_vect.fit(tweet_corpus)
    # our vectoriser is almost ready. We need to fitted to some data.
    # fitted means: what we did earlier... look at two tweets and featuring all the unique words as a matrix.
    return cv_vect
```

Use the fit_cv function to fit the vectorizer on to the test corpus

In [86]:
```
cv_vect = fit_cv(corpus)
# vector now created with corpus input
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_extraction/text.py:516: UserWarning: The parameter 'token_
pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

Get the vectorizer features

In [87]:
```
ft = cv_vect.get_feature_names_out()
```

In [88]:
```
print("There are {} features in the test corpus".format(len(ft)))
print(ft)
```

```
There are 23 features in the test corpus
['a' 'days' 'i' 'is' 'last' 'laughed' 'listening' 'lot' 'love' 'miss'
 'music' 'my' 'night' 'old' 'sad' 'she' 'so' 'stolen' 'the' 'to' 'was'
 'watch' 'you']
```

Convert the test corpus into a matrix by using the vectorize

In [89]:
```
cv_mtx = cv_vect.transform(corpus)
# creating a matrix for the test corpus
```

Print the matrix shape

In [90]:
```
print("Matrix shape is: {}".format(cv_mtx.shape))
```

```
Matrix shape is: (6, 23)
```

Convert the matrix into an array

In [91]:
```
cv_mtx.toarray()
# shows the frequency of each word in a line of the corpus - 1st line has 'i', 'love', 'listening',
# 'to', 'music' once.
```

Out[91]:
```
array([[0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0],
       [0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
        0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
        0],
       [1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
        0]])
```

Transform a new tweet by using the vectorizer

In [92]:
```
new_tweet = [["i", "miss", "my", "stolen", "watch", "from", "last", "night"]]
cv_vect.transform(new_tweet).toarray()
# output frequency of how many times the features were used in the new tweet
```

Out[92]:
```
array([[0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
        0]])
```

Now to test an example tweet which has unknown word features

In [93]:
```
unknown_tweet = [["John", "Cena", "invisible"]]
cv_vect.transform(unknown_tweet).toarray()
```

Out[93]:
```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0]])
```

Figure 11.34: fit tfidf new Method

Term Frequency - Inverse Document Frequency (TF-IDF)

Again, we will use the test corpus we made earlier in our testing

Import TfidfVectorizer from Scikit-learn Library

In [94]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Two cells below are new and are required for saving the tfidf vectorizers

In [95]:
```
def pre_processor_funct(x):
    return x
```

In [96]:
```
def fit_tfidf_new(tweet_corpus):
    tf_vect = TfidfVectorizer(preprocessor=pre_processor_funct,
                              tokenizer=pre_processor_funct)
    tf_vect.fit(tweet_corpus)
    return tf_vect
```

Create a fit_tfidf function used to build the TF-IDF vectorizer with the corpus

In [97]:
```
def fit_tfidf(tweet_corpus):
    tf_vect = TfidfVectorizer(preprocessor=lambda x: x,
                              tokenizer=lambda x: x)
    tf_vect.fit(tweet_corpus)
    return tf_vect
```

Use the fit_cv function to fit the vectorizer on the corpus, and transform the corpus

In [98]:
```
tf_vect = fit_tfidf(corpus)
tf_mtx = tf_vect.transform(corpus)
```

Get the vectorizer features (matrix columns)

In [99]:
```
ft = tf_vect.get_feature_names()
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

In [100]:
```
print("There are {} features in the this test corpus".format(len(ft)))
print(ft)
```

```
There are 23 features in the this test corpus
['a', 'days', 'i', 'is', 'last', 'laughed', 'listening', 'lot', 'love', 'miss', 'music', 'my', 'night', 'old', 'sad',
'she', 'so', 'stolen', 'the', 'to', 'was', 'watch', 'you']
```

Print the matrix shape

In [101]:
```
print(tf_mtx.shape)
```

```
(6, 23)
```

Convert the matrix to an array

In [102]:
```
tf_mtx.toarray()

# we are getting decimal numbers, not integers (like what Bag-of-Words was returning)
# the rarer a word is in the corpus, the higher the number present in the matrix.
```

Out[102]:
```
array([[0.        , 0.        , 0.3397724 , 0.        , 0.        ,
        0.        , 0.490779  , 0.        , 0.4024458 , 0.        ,
        0.490779  , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.490779  ,
        0.        , 0.        , 0.        ],
       [0.        , 0.47249269, 0.32711256, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.47249269,
        0.        , 0.        , 0.        , 0.47249269, 0.        ,
        0.        , 0.        , 0.        , 0.47249269, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.51259296, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.60714432, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.60714432],
       [0.        , 0.        , 0.        , 0.5       , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.5       ,
        0.5       , 0.5       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.4198708 , 0.        , 0.        , 0.        , 0.4198708 ,
        0.4198708 , 0.        , 0.4198708 , 0.        , 0.        ,
        0.        , 0.        , 0.4198708 , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.34430007],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.5       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.5       , 0.        , 0.        ,
        0.5       , 0.5       , 0.        ]])
```

Transform a new tweet (same as one tested earlier) by using the vectorizer

In [103]:
```
tf_vect.transform(new_tweet).toarray()
```

Out[103]:
```
array([[0.        , 0.        , 0.27198077, 0.        , 0.39285843,
        0.        , 0.        , 0.        , 0.        , 0.39285843,
        0.        , 0.39285843, 0.39285843, 0.        , 0.        ,
        0.        , 0.        , 0.39285843, 0.        , 0.        ,
        0.        , 0.39285843, 0.        ]])
```

Figure 11.35: Seaborn Helper Function

In [104]:
```
import seaborn as sn

def plot_confusion(cm):
    plt.figure(figsize = (5,5))
    sn.heatmap(cm, annot=True, cmap="Blues", fmt='.0f')
    plt.xlabel("Prediction")
    plt.ylabel("True value")
    plt.title("Confusion Matrix")
    return sn
```

Figure 11.36: Splitting Data

Train/Test Split

Remind ourselves what X and y looks like

In [105]:
```
print(X[0:10])
print(y[0:10])
```

```
[['aww', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day'], ['upset', 'not', 'updat', 'facebook', 'text',
'might', 'cri', 'result', 'school', 'today', 'also', 'blah'], ['dive', 'mani', 'time', 'ball', 'manag', 'save', 'res
t', 'go', 'bound'], ['whole', 'bodi', 'feel', 'itchi', 'like', 'fire'], ['not', 'behav', 'i', 'mad', 'not', 'see'],
['not', 'whole', 'crew'], ['need', 'hug'], ['hey', 'long', 'time', 'see', 'yes', 'rain', 'bit', 'bit', 'lol', 'i', 'f
ine', 'thank'], ['nope', 'not'], ['que', 'muera']]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Import the train_test_split function from the Scikit-Learn package

In [106]:
```
from sklearn.model_selection import train_test_split
```

Use the train_test_split function to split arrays of X and y into training and testing variables

In [107]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=0,
                                                    train_size=0.80)
# 80 percent of the data will be allocated to training the model and the other 20 percent will be for testing our
# accuracy
```

In [108]:
```
print("Size of X_train: {}".format(len(X_train)))
print("Size of y_train: {}".format(len(y_train)))
print("\n")
print("Size of X_test: {}".format(len(X_test)))
print("Size of y_test: {}".format(len(y_test)))
print("\n")
print("Train proportion: {:.0%}".format(len(X_train)/
                                        (len(X_train)+len(X_test))))
```

```
Size of X_train: 1280000
Size of y_train: 1280000


Size of X_test: 320000
Size of y_test: 320000


Train proportion: 80%
```

Figure 11.37: fit lr Method

Logistic Regression

Import the LogisticRegression model from Scikit-learn

In [110]:
```
# a logistic regression model is suitable for binary prediction like sentiment analysis
from sklearn.linear_model import LogisticRegression
```

Create a fit_lr function model used to fit a Logistic Regression model on X and y training data

In [111]:
```
def fit_lr(X_train, y_train):
    model = LogisticRegression(max_iter = 1000) # needed to increase the maximum number of iterations due to the
    # scale of data I am processing
    model.fit(X_train, y_train)
    return model
```

Figure 11.38: Positive Negative Frequencies Model

Positive/Negative Frequency

Use the build_freqs function on training data to create a frequency dictionary

Use the frequency dictionary together with tweet_to_freq function to convert X_train and X_test data to 2-d vectors

In [112]:
```
# pos/neg needs a word dictionary to look up words based on sentiment.
freqs = build_freqs(X_train, y_train)
X_train_pn = [tweet_to_freq(tweet, freqs) for tweet in X_train]
X_test_pn = [tweet_to_freq(tweet, freqs) for tweet in X_test]
```

Fit the Logistic Regression model on training data by using the fit_lr function

Print the model coefficients (betas and intercerpt)

In [113]:
```
model_lr_pn = fit_lr(X_train_pn, y_train)
print(model_lr_pn.coef_, model_lr_pn.intercept_)
```

```
[[ 2.49057592e-05 -1.86346815e-05]] [2.90903156e-10]
```

Figure 11.39: Count Vector/ Bag of Words Model

Count Vector

Use the fit_cv function on training data to build the Bag-of-Words vectorizer

Transform X_train and X_test data by using the vectorizer

In [114]:
```
cv = fit_cv(X_train)
X_train_cv = cv.transform(X_train)
X_test_cv = cv.transform(X_test)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_extraction/text.py:516: UserWarning: The parameter 'token_
pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

Fit the Logistic Regression model on training data by using the fit_lr function

In [115]:
```
model_lr_cv = fit_lr(X_train_cv, y_train)
```

Figure 11.40: TF-IDF Model

TF-IDF

Use the fit_cv function on training data to build the Bag-of-Words vectorizer

Transform X_train and X_test data by using the vectorizer

In [116]:
```
tf = fit_tfidf(X_train)
X_train_tf = tf.transform(X_train)
X_test_tf = tf.transform(X_test)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_extraction/text.py:516: UserWarning: The parameter 'token_
pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

Fit the Logistic Regression model on training data by using the fit_lr function

In [118]:
```
model_lr_tf = fit_lr(X_train_tf, y_train)
```

Figure 11.41: Model Testing

Mini-Pipeline

Final test tweet used to check if the model works as expected

In [129]:
```
tweet = r"""RT @elonmusk : you are so cooooooool!
I Can't believe you are REal!!! 😎  ...
I have been saving up to buy one of your Teslas!
Everyone, check out his work at https://www.tesla.com/en_gb and LET ME KNOW WHAT YOU THINK OF HIM!!!
#ELONMUSK #Elon #Legend #Revolutionary"""
```

Create a predict_tweet function used to pre-process, transform and predict tweet sentiment

In [130]:
```
def predict_tweet(tweet):
    processed_tweet = process_tweet(tweet)
    transformed_tweet = tf.transform([processed_tweet])
    prediction = model_lr_tf.predict(transformed_tweet)

    if prediction == 1:
        return "Prediction is positive sentiment"
    else:
        return "Prediction is negative sentiment"
```

Predicting our tweet's sentiment by using the predict_tweet function

In [131]:
```
predict_tweet(tweet)
```

Out[131]:
```
'Prediction is positive sentiment'
```

Figure 11.42: Saving Model and Vectorization Method

Saving Model

Below cell imports the joblib library which allows us to save the tf-idf model from sklearn.

In [132]:
```
import joblib
```

In [133]:
```
filename = 'model.sav'
joblib.dump(model_lr_tf, filename)
```

Out[133]:
```
['model.sav']
```

Save the fit_tfidf method as tf which is used for predicting the sentiment of a tweet

In [134]:
```
tf_new = fit_tfidf_new(X_train)
joblib.dump(tf_new, 'tf.pkl')
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_extraction/text.py:516: UserWarning: The parameter 'token_
pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

Out[134]:
```
['tf.pkl']
```

Twitter Scraper

Figure 11.43: getClient, a method for connecting to Twitter's API

```
23  # keys needed to communicate with Twittter's API:
24  api_key = 'K9JI3yGwiGUacSdQTbUG6EjKt'
25  api_secret_key = 'FuxMofcCLkRm5gIZkLOnvKUliXTUAYcm8zZM8fIJzFDV6sGaRw'
26  bearer_token = 'AAAAAAAAAAAAAAAAAAAAAMqPWAEAAAAAkfR2tuTUyfEjQVg8rJnMXJE0tew%3D4G4asHdTXh9hcTvv0oqnq6kLUU6W4V6bbe6bGxJEzRGcxnoWQH'
27  access_token = '1470048487-TXkyFfTABq3wdywcvd59ETePhZyXMZDGOzZUxtV'
28  access_token_secret = 'z9f5F41mDGTostzEBQFO0RCXyvJX1HvTA2h3jD9jbGFxA'
29  vps_directory = '/Users/louiskampman/Desktop/Goldsmiths 3rd Year/Final Project/Project'
30  num_of_tweets = 10
31
32  # method allowing us to create a client object
33  def getClient():
34      client = tweepy.Client(bearer_token = bearer_token,
35                             consumer_key = api_key,
36                             consumer_secret = api_secret_key,
37                             access_token = access_token,
38                             access_token_secret = access_token_secret)
39      return client
```

Figure 11.44: searchTweets, a method for scraping tweets from Twitter

```python
# search function to scrape tweets from twitter
def searchTweets(query, max_results):
    client = getClient()
    tweets = client.search_recent_tweets(query=query,
                                         tweet_fields=['public_metrics'],
                                         place_fields=['country'],
                                         max_results=max_results)
    tweet_data = tweets
    # tweet_data = tweets
    results = []

    if not tweet_data is None:
        for tweet in tweet_data.data:
            print("\n\n\n")
            obj = [None, None, None, None, None]
            obj[0] = tweet.id
            obj[1] = tweet.text
            obj[2] = tweet.public_metrics['retweet_count']
            obj[3] = tweet.public_metrics['reply_count']
            obj[4] = tweet.public_metrics['like_count']
            results.append(obj)
    else:
        return ''
    print("\n\n\n")
    for tweet in results:
        print(tweet)
        print("\n")
    return results
```

Figure 11.45: Example output of searchTweets

```
[1519691849010753538, '#LNG-Terminal in Wilhelmshaven: Baustart bereits nächste Woche. #Gas #Gas-Terminal #Deutschland.\nRobert #Habeck kommt jetzt mit #Tesla-Tempo.\nhttps://t.co/AWj21hcFkQ', 0, 0, 0]

[1519691841943261185, 'L'AT de @DCohendeLara : #TESLA en D. Rupture du nuage proche du twist et de MM200.Cours au contact du support à 823.43$ +Kijun de la LS. Si rupture en clôture, objectifs 762.18$ puis 730$
 = support + objectif rupture du twist de nuages. Inv.&gt;924$.\nhttps://t.co/3RwDxpVsvc https://t.co/xkXkHNs15T', 1, 0, 0]

[1519691823874121729, 'RT @digi_karma: MARGIN CALL: #ElonMusk would get margin called on his #Twitter loan if #Tesla shares ever dropped to $520 per share.\n\nIF th…', 1, 0, 0]

[1519691814877417476, '@elonmusk can you buy @ManUtd it's about 3-4billion n going to make a fortune with you at the helm!! #1958 #GlazersOut #manutd #Tesla #WelcomeErik', 0, 0, 0]

[1519691681376874497, 'RT @tle360: "Always Do What You Are Afraid To Do" — Ralph Waldo Emerson.  \nHappy Cryptoing!\n#Tesla #ElonMusk #Twitter #BTC #ETH #invest #Tr…', 82, 0, 0]

[1519691670744403968, 'RT @UW_CSTV: The US National Highway #Traffic Safety Administration may-finally-be taking more regulatory interest in #Tesla\'s "Autopilot"…', 1, 0, 0]

[1519691622749028354, '@VoltInuOfficial @elonmusk We want #Tesla #VOLTARMY love it https://t.co/womwmzMXYk', 0, 0, 1]

[1519691613550882816, '@cbouzy I will buy #tesla stocks once it is down 90% and Bezos has taken it over with hostility  .. Over to you @JeffBezos', 0, 0, 0]

[1519691588053745665, 'RT @Teslaconomics: People forget, Elon is a bear trapping genius. \n\nHe is in the process of trapping all the bears.\n\n🔥🧸🔥\n\n$TSLA #Tesla', 8, 0, 0]

[1519691487373537280, 'RT @vivek_mandhare: #MarriageStrike \n#GenderBiasedLaws \n#498A \n#FakeCases \n#Judiciary \n\n#ElonMusk \n#SUGA \n#Tesla \n#Madrid \n#mensrights \n#Me…', 42, 0, 0]
```

Front End Code

Figure 11.46: App.js Code

App.js

src > App.js > ...

```
import './App.css';
import NavBar from './components/Navbar';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './pages';
import About from './pages/about';
import ContactUs from './pages/contact-us';
import { useState } from 'react';



function App() {

  const [analysis, setAnalysis] = useState([])
  const [hash, setHash] = useState([])

  function getAnalysis(data){
    let res = []
    for(let d of data){
      let test = JSON.stringify(d)
      res.push(JSON.parse(test))
    }
    setAnalysis(res)
  }

  function getHashtag(hashtag) {
    setHash(hashtag);
  }

  return (
    <Router>
      <NavBar getAnalysis = {getAnalysis} getHashtag= {getHashtag}/>
      <Routes>
        <Route exact path="/" element={<Home data={analysis} hashtag={hash} />} />
        <Route exact path="/about" element={<About/>} />
        <Route exact path="/contact-us" element={<ContactUs/>} />
      </Routes>
    </Router>
  )
}

export default App;
```

Figure 11.47: Home Page Code

index.js M

src > pages > index.js > ...

```
import React from 'react';
import '../App.css';
import Chart from '../components/Others/PieChart'
import MyCloud from '../components/Others/MyCloud'

const Home = ({data, hashtag}) => {
        console.log(hashtag)

        return (
            hashtag.length !== 0 ? (
                <div className='background_image'>
                    <div className='table_component'>
                        <h2>Response from { hashtag }:</h2>
                        <div className='table'>
                            <table>
                                <thead>
                                    <tr className='tableHeaders'>
                                        <th>Sentiment</th>
                                        <th>Text</th>
                                        <th className='column'>Retweets</th>
                                        <th className='column'>Replies</th>
                                        <th className='column'>Likes</th>
                                    </tr>
                                </thead>
                                <tbody>
                                    {data.map(d => (
                                        <tr>
                                            <th>{d.Sentiment[1]}</th>
                                            <th>{d.Text}</th>
                                            <th className='column'>{d.Retweets}</th>
                                            <th className='column'>{d.Replies}</th>
                                            <th className='column'>{d.Likes}</th>
                                        </tr>
                                    ))}
                                </tbody>
                            </table>
                        </div>
                    </div>
                    <div className="visArea">
                        <div className="pieContainer">
                            <h2>Pie Chart:</h2>
                            <div className="cont">
                                <Chart className='chart' data={data}/>
                            </div>
                        </div>
                        <div className="cloudContainer">
                            <h2>Word Cloud:</h2>
                            <MyCloud className='cloud' data={data}/>
                        </div>
                    </div>
                </div>
            ) :
            (
                <div className='background_image'/>
            )
        )
}

export default Home
```

Figure 11.48: Navbar index.js Code

index.js

src > components > Navbar > index.js > ...

```
import React, { useState } from 'react'
import { NavLink as Link } from 'react-router-dom'
import NavSearch from './NavSearch'
import '../../App.css'
import LogoImage from "../../LogoDark.png"

const Navbar = ({getAnalysis, getHashtag}) => {

// functionality for opening our navigation dropdown menu
const [click, setClick] = useState(false)

const handleClick = () => setClick(!click)


    return (
        <>
            <div className='nav'>
                <Link className="logo" to="/" >
                    <img src= {LogoImage} alt=" "></img>
                </Link>

                {!click &&
                    <NavSearch getAnalysisCallback={getAnalysis} getHashtagCallback={getHashtag}></NavSearch>
                }

                {click &&
                    <div className='dropdown-menu'>
                        <div className='dropdown-inner'>
                            <div className='link' >
                                <Link to="/" style={{'text-decoration':'none', 'color':'white'}}>
                                    Home
                                </Link>
                            </div>
                            <div className='link' >
                                <Link to="/about" style={{'text-decoration':'none', 'color':'white'}}>
                                    About
                                </Link>
                            </div>
                            <div className='link'>
                                <Link to="/contact-us" style={{'text-decoration':'none', 'color':'white'}}>
                                    Contact Us
                                </Link>
                            </div>
                        </div>
                    </div>
                }

                <div className='menuIcon' onClick={handleClick}>
                    <i className={click ? 'fas fa-times' : 'fas fa-bars'}/>
                </div>
            </div>
        </>
    )
}

export default Navbar
```

Figure 11.49: NavSearch Code

NavSearch.js

src > components > Navbar > NavSearch.js > ...

```
import React,{useState} from "react";
import '../../App.css';
import { FaSearch } from "react-icons/fa";

function NavSearch ({getAnalysisCallback, getHashtagCallback}){
    const [hashtag, setHashtag] = useState("");

    async function get_sentiment_analysis(hashtag) {
        const requestOptions = {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({ hashtag: hashtag })
        }
        await fetch('http://localhost:5000/get_sentiment_analysis', requestOptions)
            .then((response) => response.json())
            .then((data) => getAnalysisCallback(data))
            .catch(err => console.log(err))
    }

    function get_hashtag() {
        console.log(hashtag)
        getHashtagCallback(hashtag)
    }

    return (
        // added another div container for the others inside
    <div className="search">
        <div className="searchInputs">
            <input
                type="text"
                placeholder="Search hashtag"
                name="s"
                onChange={event =>setHashtag(event.target.value)}
                onKeyPress={(e) => e.key === 'Enter' && [get_sentiment_analysis(hashtag), get_hashtag()]}
            />
            <button type="submit" className="searchButton" onClick={()=>[get_sentiment_analysis(hashtag), get_hashtag()]}>
                <div className="searchIcon">
                    <FaSearch/>
                </div>
            </button>
        </div>
    </div>
    )
}

export default NavSearch;
```

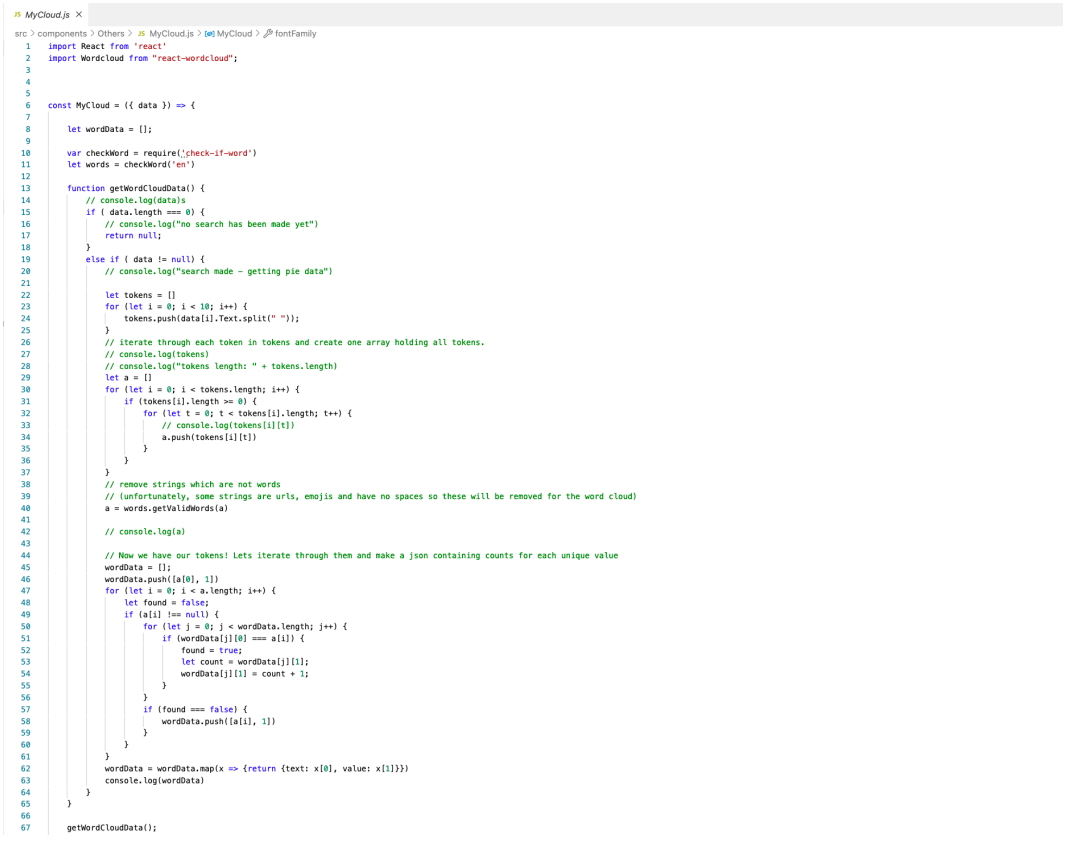
Figure 11.50: PieChart Code

PieChart.js

src > components > Others > PieChart.js > ...

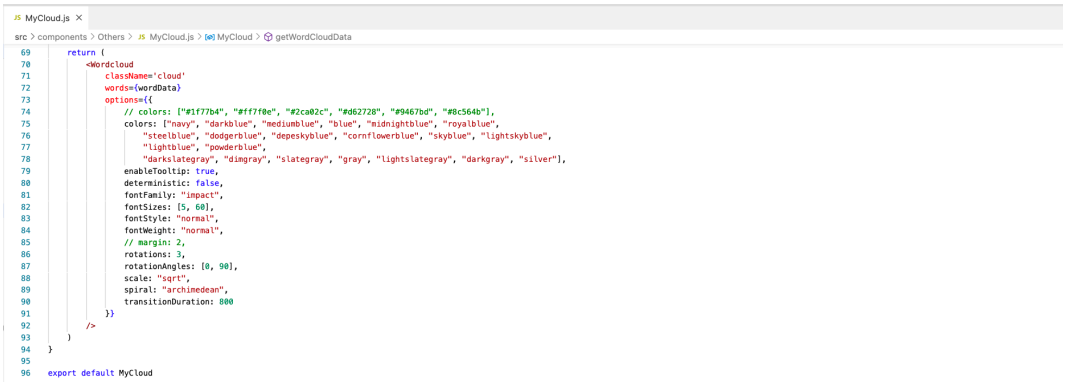
```
import React from 'react'
import { Pie } from "react-chartjs-2";
import "chart.js/auto"

const Chart = ({ data }) => {

    function getPieData() {
        // console.log(data)
        let a = [0, 0]
        if (data.length === 0) {
            // console.log("no search has been made yet")
            return [0, 0];
        }
        else if ( data != null ) {
            // console.log("search made - getting pie data")
            for (let i = 0; i < 10; i++) {
                // console.log("~~~~Element " + i + "~~~~\nSentiment: " + data[i].Sentiment[1] + ", Text: " + data[i].Text)
                if (data[i].Sentiment[1] === '0') {
                    a[0] += 1;
                } else if (data[i].Sentiment[1] === '1') {
                    a[1] += 1;
                }
            }
        }
        return a
    }

    const pie_data = getPieData();

    return (
        <Pie
            className='pieChart'
            data = {{
                labels: ["Negative", "Positive"],
                datasets: [
                    {
                        label: 'Count',
                        data: pie_data,
                        backgroundColor: ["Crimson", "Navy"],
                    },
                ],
            }}
            // height= {300}
            // width= {300}
            options={{
                tooltips: {enabled: false},
                hover: {mode: null},
                responsive: true,
                maintainAspectRatio: false,
            }}
        />
    )
}

export default Chart
```

Figure 11.51: MyCloud Code - Part 1



```
MyCloud.js
src > components > Others > MyCloud.js > MyCloud > fontFamily
import React from 'react'
import Wordcloud from "react-wordcloud";



const MyCloud = ({ data }) => {

    let wordData = [];

    var checkWord = require('check-if-word')
    let words = checkWord('en')

    function getWordCloudData() {
        // console.log(data)s
        if ( data.length === 0) {
            // console.log("no search has been made yet")
            return null;
        }
        else if ( data != null) {
            // console.log("search made - getting pie data")

            let tokens = []
            for (let i = 0; i < 10; i++) {
                tokens.push(data[i].Text.split(" "));
            }
            // iterate through each token in tokens and create one array holding all tokens.
            // console.log(tokens)
            // console.log("tokens length: " + tokens.length)
            let a = []
            for (let i = 0; i < tokens.length; i++) {
                if (tokens[i].length >= 0) {
                    for (let t = 0; t < tokens[i].length; t++) {
                        // console.log(tokens[i][t])
                        a.push(tokens[i][t])
                    }
                }
            }
            // remove strings which are not words
            // (unfortunately, some strings are urls, emojis and have no spaces so these will be removed for the word cloud)
            a = words.getValidWords(a)

            // console.log(a)

            // Now we have our tokens! Lets iterate through them and make a json containing counts for each unique value
            wordData = [];
            wordData.push([a[0], 1])
            for (let i = 0; i < a.length; i++) {
                let found = false;
                if (a[i] !== null) {
                    for (let j = 0; j < wordData.length; j++) {
                        if (wordData[j][0] === a[i]) {
                            found = true;
                            let count = wordData[j][1];
                            wordData[j][1] = count + 1;
                        }
                    }
                    if (found === false) {
                        wordData.push([a[i], 1])
                    }
                }
            }
            wordData = wordData.map(x => {return {text: x[0], value: x[1]}})
            console.log(wordData)
        }
    }

    getWordCloudData();
```

Figure 11.52: MyCloud Code - Part 2



```
MyCloud.js
src > components > Others > MyCloud.js > MyCloud > getWordCloudData
    return (
        <Wordcloud
            className='cloud'
            words={wordData}
            options={{
                // colors: ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b"],
                colors: ["navy", "darkblue", "mediumblue", "blue", "midnightblue", "royalblue",
                    "steelblue", "dodgerblue", "depeskyblue", "cornflowerblue", "skyblue", "lightskyblue",
                    "lightblue", "powderblue",
                    "darkslategray", "dimgray", "slategray", "gray", "lightslategray", "darkgray", "silver"],
                enableTooltip: true,
                deterministic: false,
                fontFamily: "impact",
                fontSizes: [5, 60],
                fontStyle: "normal",
                fontWeight: "normal",
                // margin: 2,
                rotations: 3,
                rotationAngles: [0, 90],
                scale: "sqrt",
                spiral: "archimedean",
                transitionDuration: 800
            }}
        />
    )
}

export default MyCloud
```

Figure 11.53: About Page Code

```
about.js
src > pages > about.js > ...
import React from 'react'

const About = () => {
    return (
        <div className='background_image'>
            <div className='about_page'>
                <div className='supplementary_page_contents'>
                    <h2>Introduction</h2>
                    <br/>
                    <h3>Focus</h3>
                    <br/>
                    <p>The focus of this project is to build a web application which can visualise the current popularity of entered Twitter hashtags which can then be used as a tool for financial analysts to track investments.</p>
                    <br/>
                    <p>The analysis of data from Twitter is an example of sentiment analysis which has proven to help businesses understand the feelings of their consumers without conducting formal research and is becoming more and more common today. Financial tools using data coll
                    <br/>

                    <h3>Motivation</h3>
                    <br/>
                    <p>Often, financial research involves following articles and data provided by large institutions such as Bloomberg, Reuters, and the Financial Times. With the use of social media growing and the huge amounts of data available, I believe if financial analysts cou
                    <br/>
                    <p>The problem my project aims to solve is the time taken for financial analysts to conduct research into their investments. Research can be a time-consuming process involving reading countless articles over a long-time frame. The hope I have for my project is t
                    <br/>

                    <h3>Aims and Objectives</h3>
                    <br/>
                    <p>My project's novel contribution is its simplicity and accessibility to users. Other resources like this are not available for free and are restrictive to a user's goals. My project aims to simplify what people have done in the past by having an application wi
                    <br/>
                    <p>Furthermore, most research conducted in finance is done through more established sources, for example the Financial Times and Bloomberg. My project aims to complement this research and allow those looking to invest into financial markets to make more educated
                    <br/>
                    <p>The use of Machine Learning to uncover meanings behind qualitative data interests me and I believe that developing applications that do this could provide some huge benefits to the world of Finance, Marketing and more.</p>
                </div>
                <div className='instructions_content'>
                    <h2>Instructions</h2>
                    <br/>
                    <p>In order to use the application, users must enter a hashtag into the search bar at the top of the screen (within the navigation bar) and wait for a response to be sent back from the application.</p>
                    <br/>
                    <p>Due to a restriction in the Tweepy library which allows for connecting to Twitter's deleopment program and scraping Tweets, each search returns 10 Tweets each. This data is displayed through the use of Tables, Pie Charts and WordClouds to give users a rough i
                    <br/>
                    <p>Users can also navigate to other supplementary pages such as the about and contact-us pages through the use of the dropdown button on the right hand side of the navigation bar.</p>
                    <br/>
                    <p>For users that are new to the application, please feel free to test out the search bar by entering hashtags such as "#happy" and "#sad".</p>
                </div>
            </div>
        </div>
    )
}

export default About
```

Figure 11.54: Contact-Us Page Code

```
contact-us.js
src > pages > contact-us.js > ...
import React from 'react'

const ContactUs = () => {
    return (
        <div className='background_image'>
            <div className='contact_us_page'>
                <div className='contact_page_contents'>
                    <h2>Louis Kampman</h2>
                    <br/>
                    <p>Email - Louis.Kampman@gmail.com</p>
                    <br/>
                    <p>GitHub - LouVic12</p>
                </div>
            </div>
        </div>
    )
}

export default ContactUs
```

Machine Learning Model Accuracies

Figure 11.55: Confusion Matrix - Positive Negative Frequencies Model

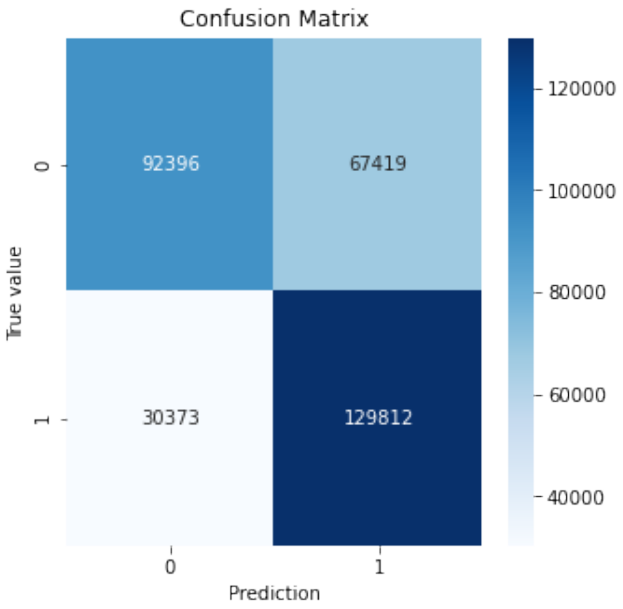


Figure 11.56: Confusion Matrix - Count Vector/ Bag of Words Model

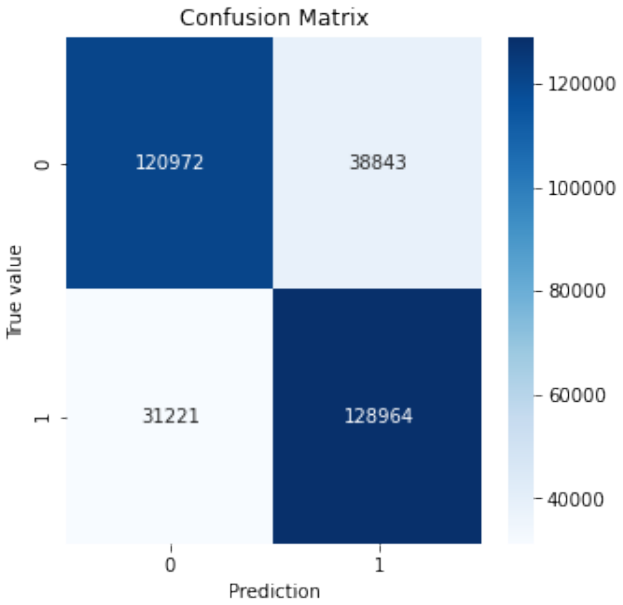
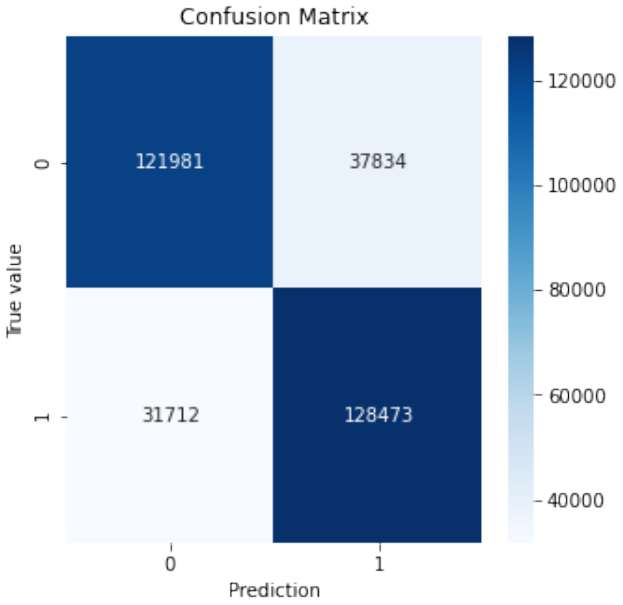


Figure 11.57: Confusion Matrix - Final (TF-IDF) Model



White Box Testing

Figure 11.58: White Box Testing Table

Test Number	Description	Test Data	Expected Results	Actual Results	Passed?	Action Taken
			Application Initialisation			
1	Upon loading the app, you are displayed with an empty home screen	N/A	The home screen (with no main body components or visualisations) should be displayed	Result was successfully met as expected	Yes	N/A
			Navigation Bar			
2	The search bar responds to button and 'enter' button clicks	N/A	The search bar's functionality should allow users to submit their entered hashtag either by clicking the 'enter' button on their keyboard or by clicking the button on the right of the search bar.	Result was successfully met as expected	Yes	N/A
3	The dropdown menu opens and closes properly	N/A	The dropdown menu works properly, whereby the search bar is hidden when opened, and replaced by the navigation links to the home, about and contact-us pages and these changes can be reversed when closing the menu.	Result was successfully met as expected	Yes	N/A
4	Navigation links to other pages work	N/A	The navigation links available from the site's logo and dropdown menu work properly and bring users to the correct page.	Result was successfully met as expected.	Yes	N/A
5	Navigation bar is responsive to screen size changes	N/A	The navigation bar adjusts its design relative to the screen size of a device. There shouldn't be any cases where components overlap or ruin the page's design	Result was successfully met as expected.	Yes	N/A
			Home Page			
6	Components are only displayed after first search.	N/A	The home screen components for the table of results, pie chart and word cloud visualisations should only be displayed after the first hashtag search has been made upon the application opening. Once a search is made, the components should be displayed immediately, already having been loaded.	Result was NOT met	No	Currently, my application satisfies the first point (components displayed only after first search), however there are typically two seconds of loading necessary after these are displayed before the visualisations are then populated with their data (I found that the word cloud was slowing this process down).
7	Table of results component implemented successfully	N/A	The table of results component should satisfy its design requirements and display data correctly and clearly, without error.	Result was successfully met	Yes	N/A
8	Pie Chart component implemented successfully	N/A	The pie chart component should satisfy the design requirements and display our data clearly, without error.	Result was successfully met	Yes	N/A
9	Ternary Chart component implemented successfully	N/A	The ternary chart component should satisfy the design requirements and display our data clearly, without error.	Result was NOT met	No	I made this feature unavailable due to time constraints. If I had more time, I would certainly add this to my application in the future. I decided to instead display this data using the table component (with three additional columns when the screen size is large enough).
10	Geographical visualisation component implemented successfully	N/A	The ternary chart component should satisfy the design requirements and display our data clearly, without error.	Result was NOT met	No	I made this feature unavailable due to time constraints. If I had more time, I would certainly add this to my application in the future.
11	Home page components are responsive to screen size changes	N/A	The home page adjusts its design relative to the screen size of a device. There shouldn't be any cases where components overlap or ruin the page's design	Result was successfully met	Yes	N/A
			About Page			
12	Text components display as intended	N/A	The text components found in the about page should display as intended, without error.	Result was successfully met	Yes	N/A
13	About page components are responsive to screen size changes	N/A	The about page adjusts its design relative to the screen size of a device. There shouldn't be any cases where components overlap or ruin the page's design	Result was successfully met	Yes	N/A
			Contact Us Page			
14	Contact us page component displays as expected	N/A	The text component found in the contact-us page should display as intended, without error.	Result was successfully met	Yes	N/A
15	Contact us page is responsive to screen size changes	N/A	The contact us page adjusts its design relative to the screen size of a device. There shouldn't be any cases where components overlap or ruin the page's design	Result was successfully met	Yes	N/A
			Machine Learning Model			
16	Machine Learning model is functional in the backend and predicts tweets accurately	N/A	The accuracy of our machine learning model on unseen data should have a high prediction accuracy and work without any disruptive errors to our application. Tweets with clear sentiments visible to users should have their predicted sentiments reflected properly ("#Happy" search should return largely positive sentiments, etc)	Result was successfully met	Yes	N/A
			Twitter Scraper			
17	Twitter scraper can connect to Twitter's API without issue	N/A	The twitter scraper should be able to connect to Twitter's API server, using our personal keys without any issue	Result was successfully met	Yes	N/A
18	Scrapes tweets without running into errors which break the disrupt or break the application	N/A	Our twitter scraper should be reliable and not run into errors which break our application	Result was successfully met	Yes	N/A
19	Scraper can collect the Tweet ID, text, retweet count, reply count, like count and location from each fetch	N/A	The scraper should be able to collect multiple metrics for each of our tweets, with limited empty data values.	Result was NOT met	No	Although I was able to implement a large portion of the data collection, I wanted for searching recent tweets, I was arguably unsuccessful when it came to collecting the place objects of tweets. I found that this was the case because a large portion of the tweets I collected in my testing had empty data values for their location field. I decided to remove this functionality from my project as the amount of place data collected would remain very limited.

User Testing

First Survey

Figure 11.59: My First Survey

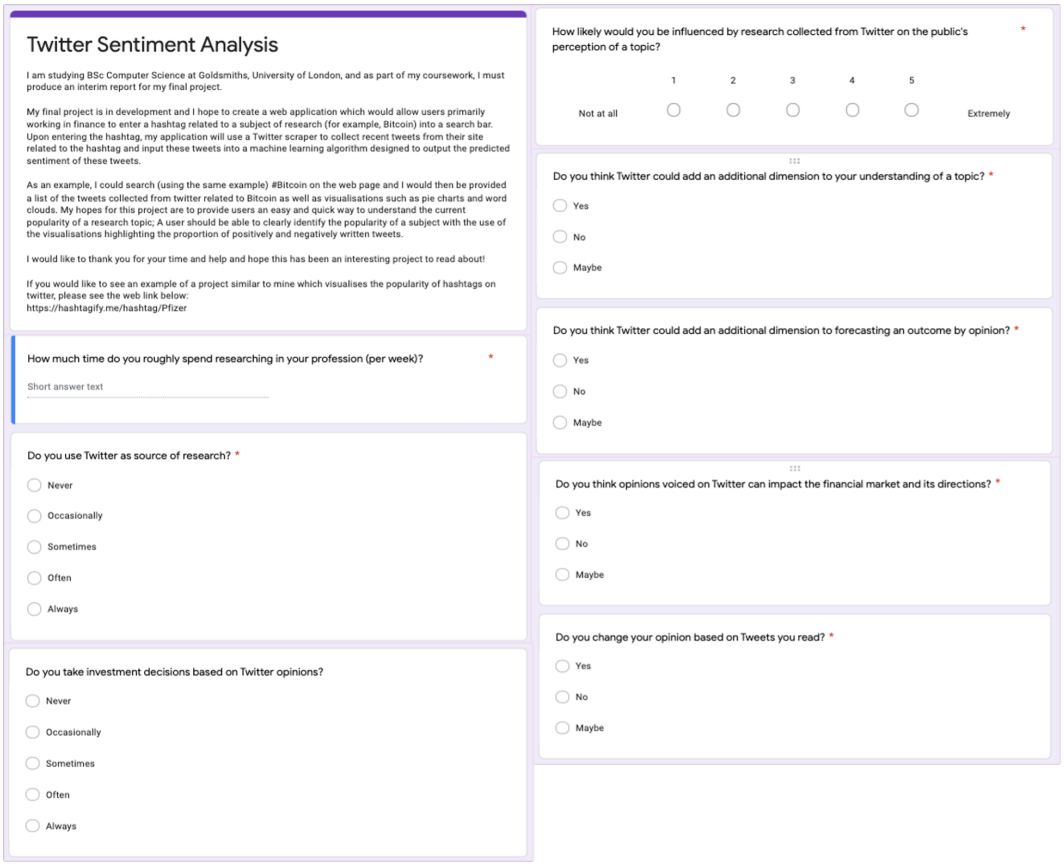


Figure 11.60: Question 1 Results

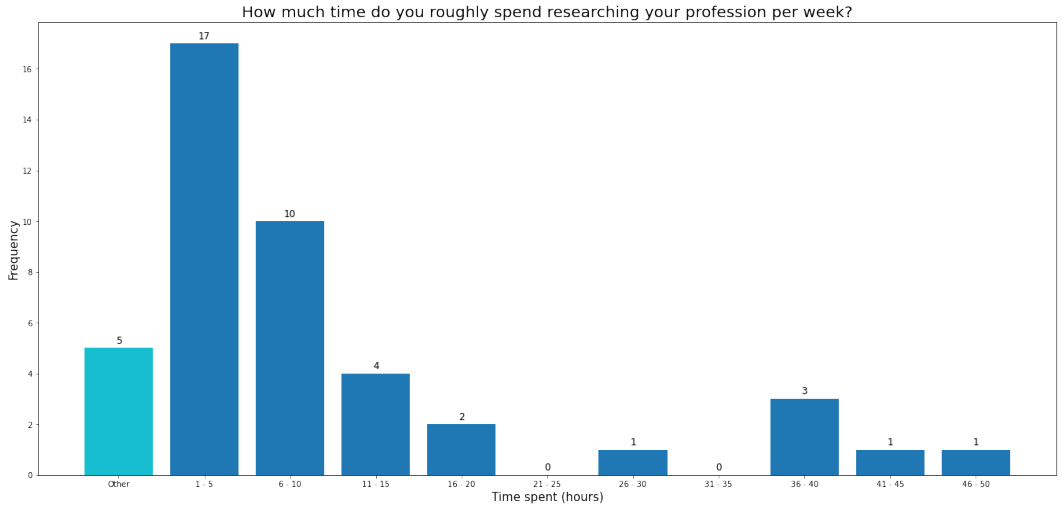


Figure 11.61: Question 2 Results

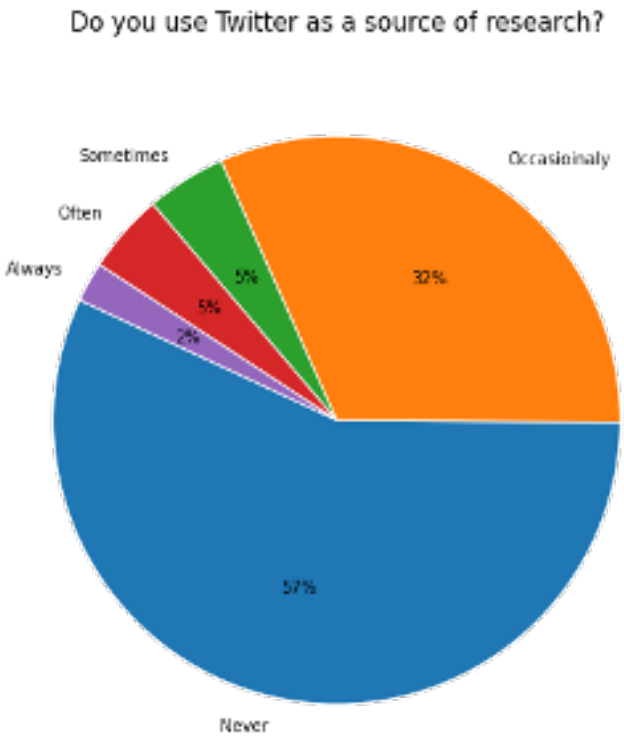


Figure 11.62: Question 3 Results

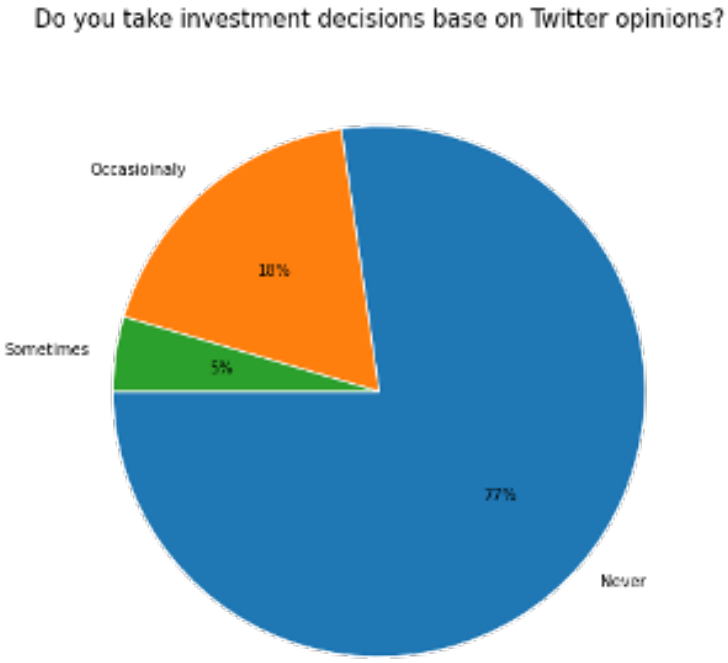


Figure 11.63: Question 4 Results

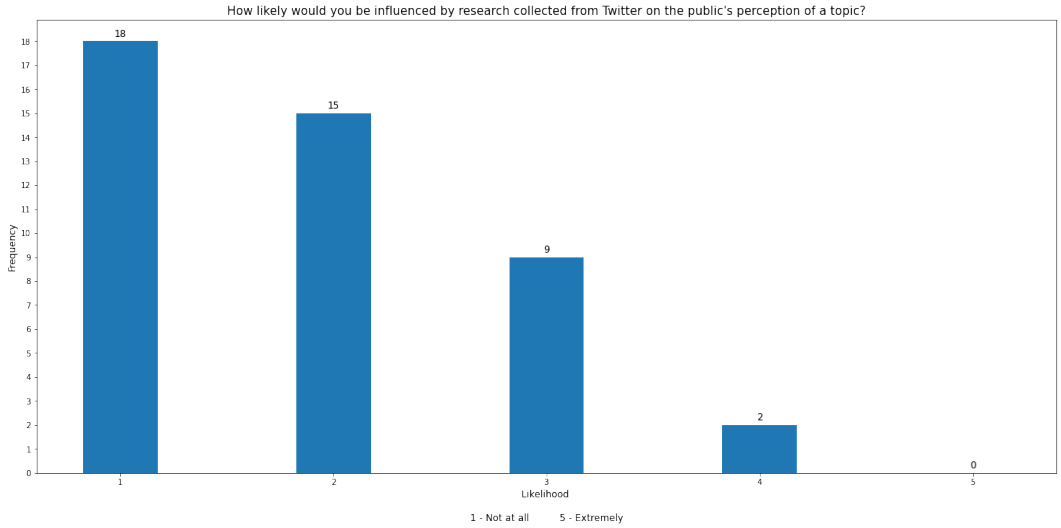


Figure 11.64: Question 5 Results

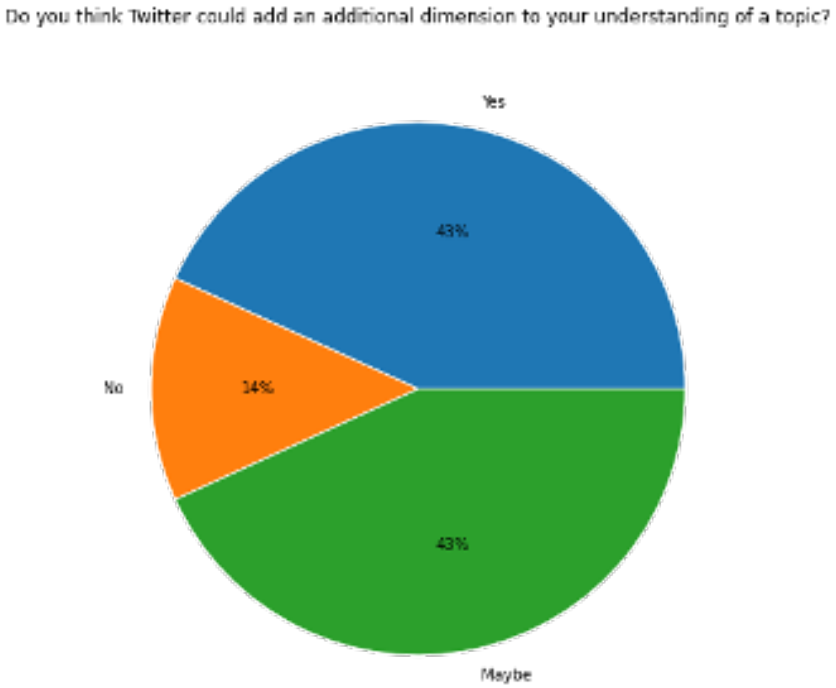


Figure 11.65: Question 6 Results

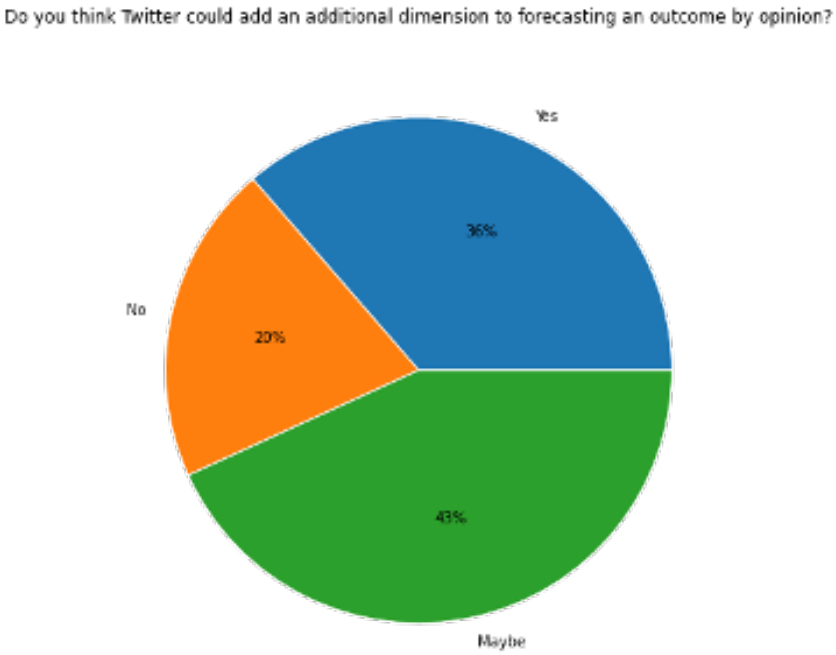


Figure 11.66: Question 7 Results

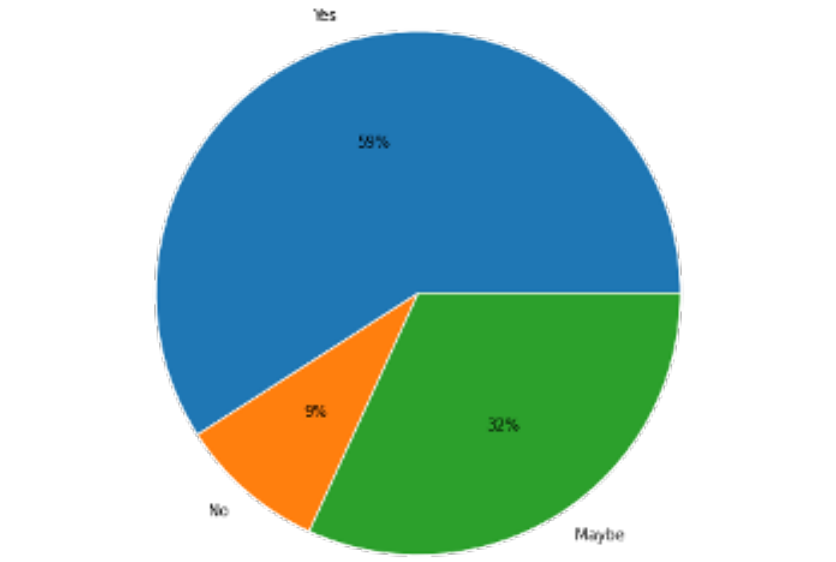
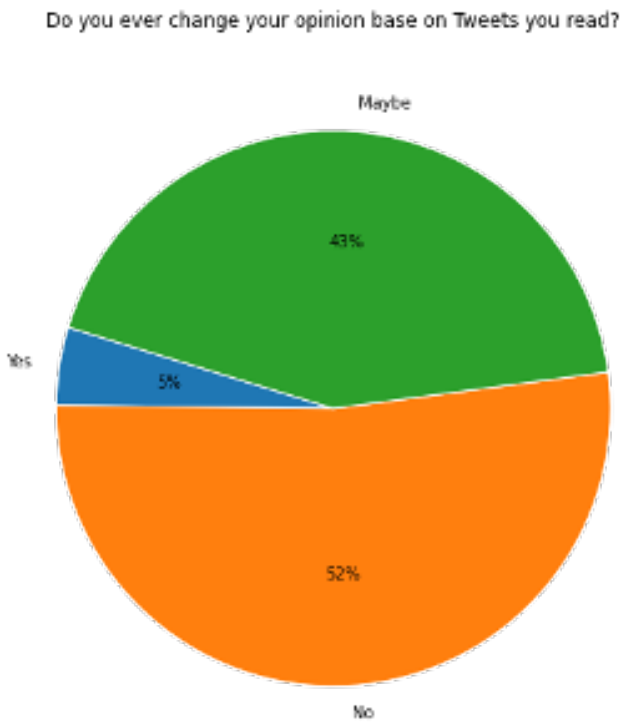


Figure 11.67: Question 8 Results



Second Survey

Figure 11.68: My Second Survey

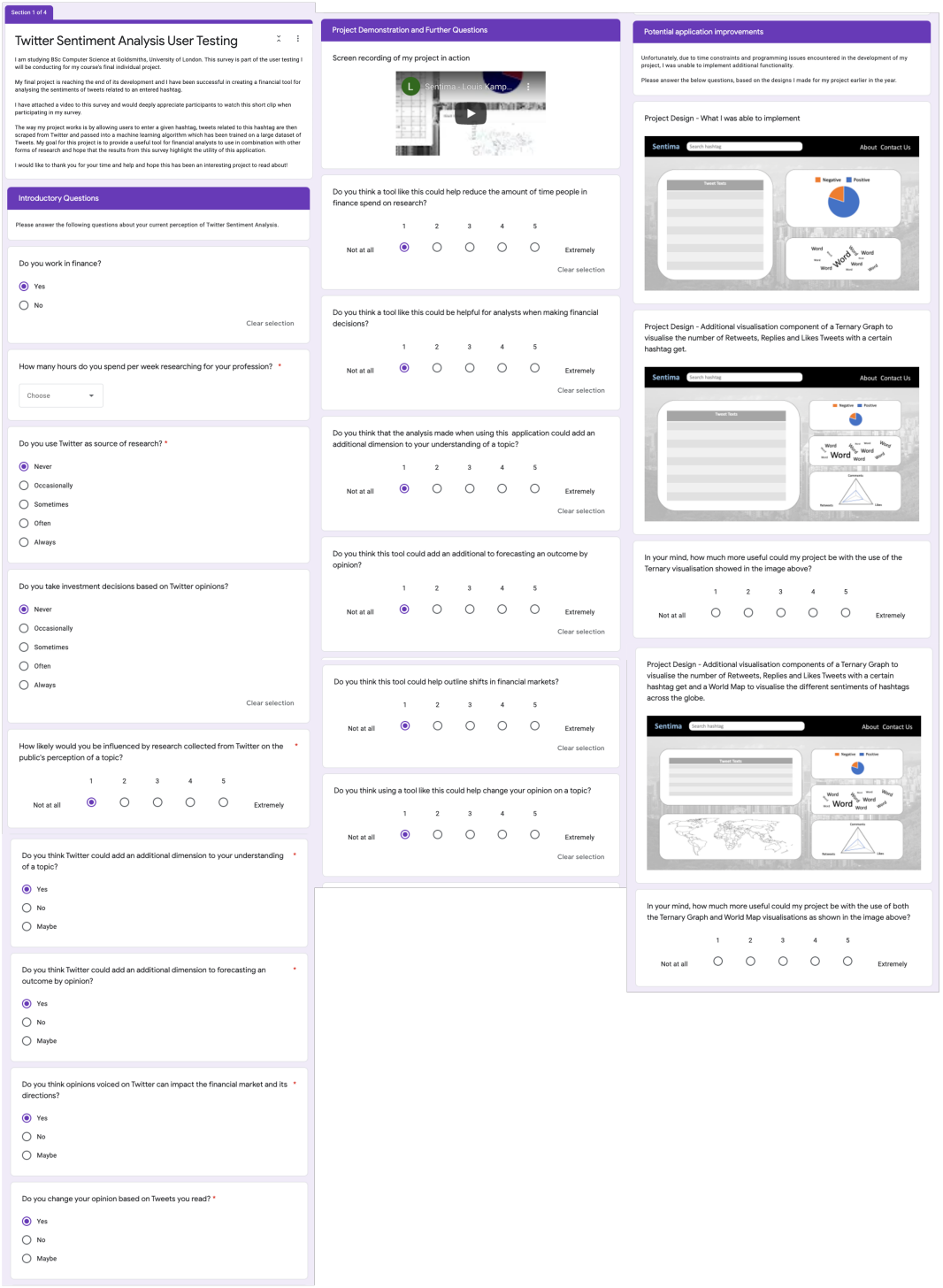


Figure 11.69: Question 1 Results

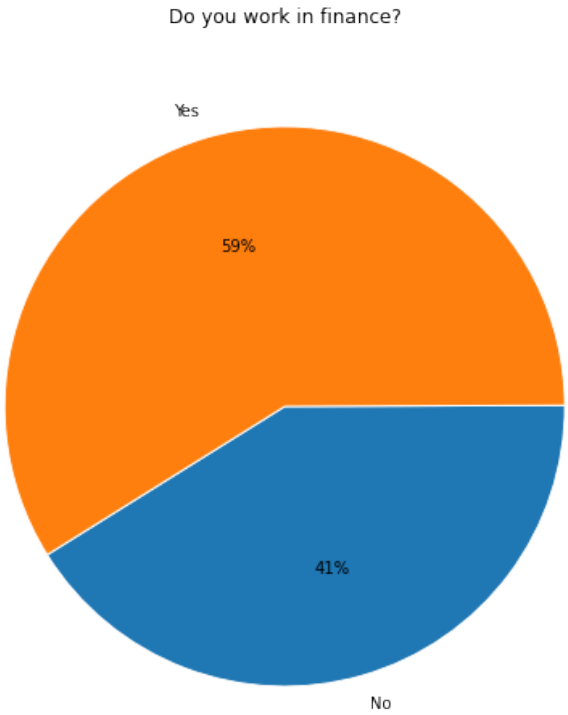


Figure 11.70: Question 2 Results

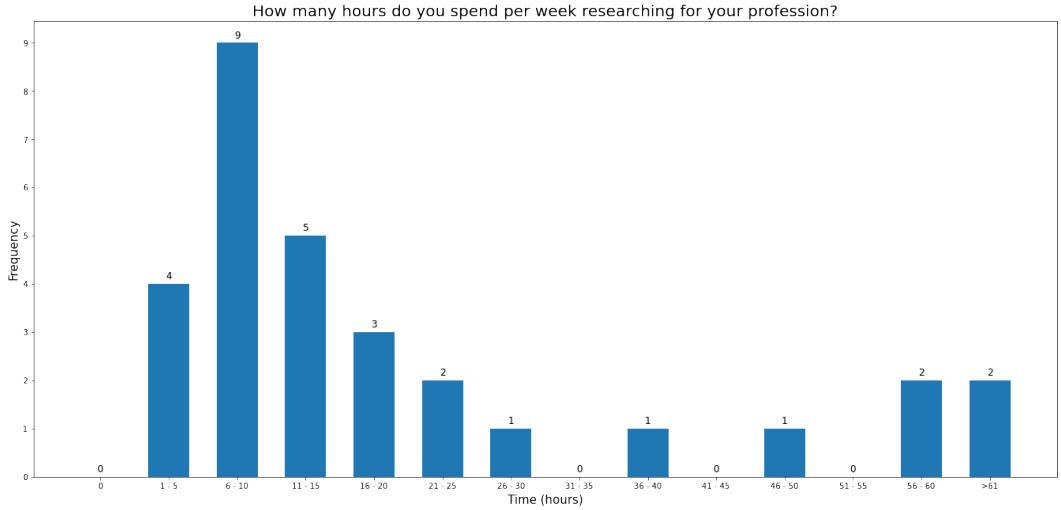


Figure 11.71: Question 3 Results

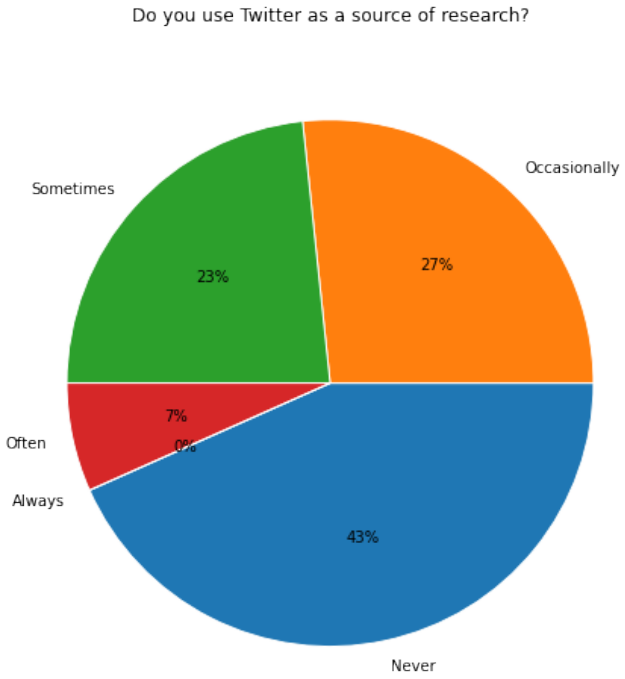


Figure 11.72: Question 4 Results

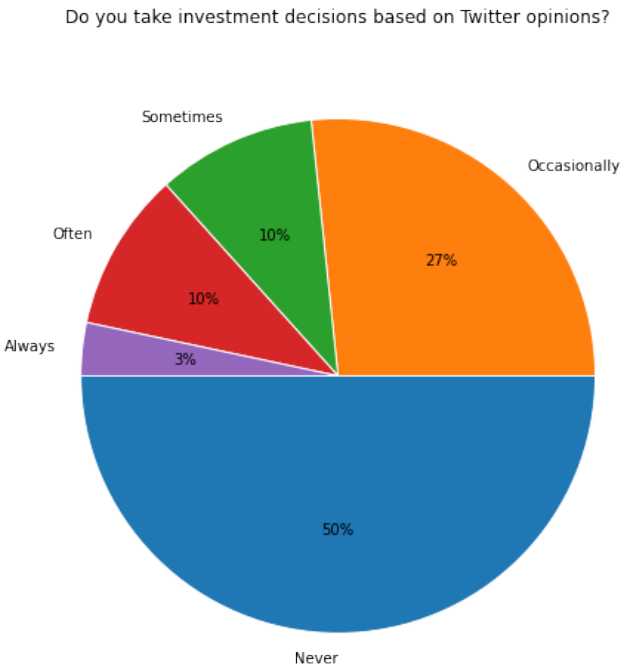


Figure 11.73: Question 5 Results

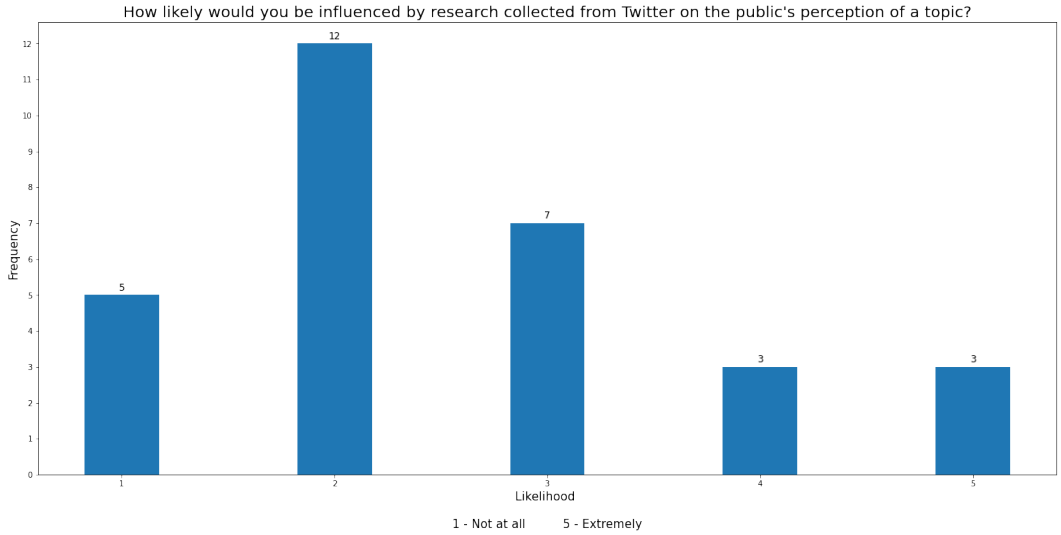


Figure 11.74: Question 6 Results

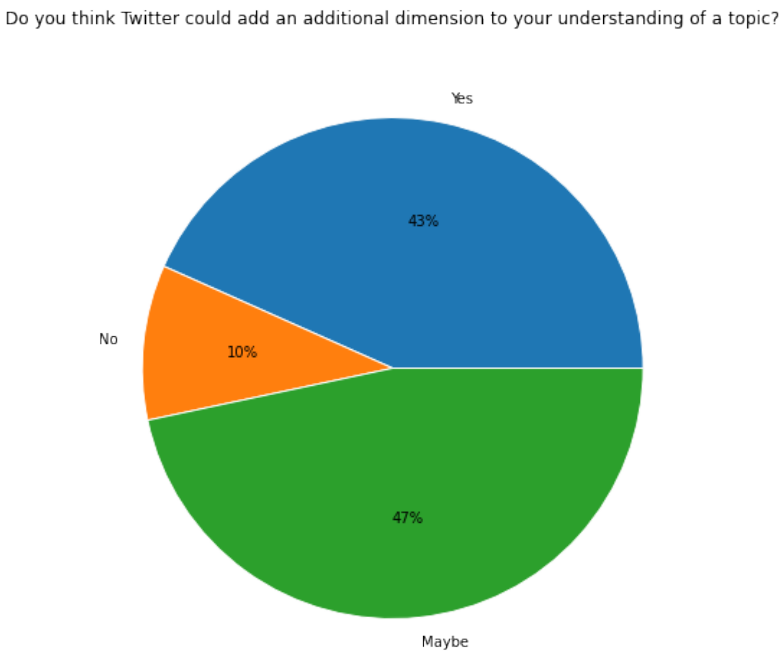


Figure 11.75: Question 7 Results

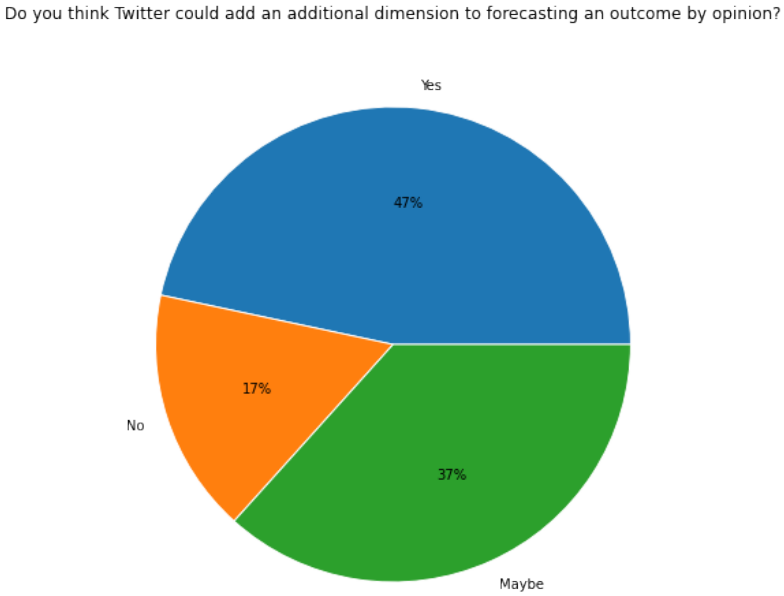


Figure 11.76: Question 8 Results

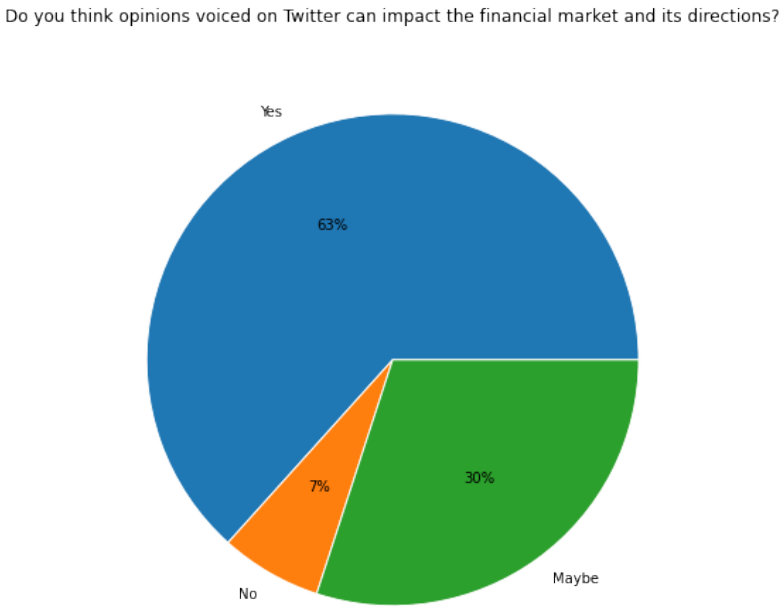


Figure 11.77: Question 9 Results

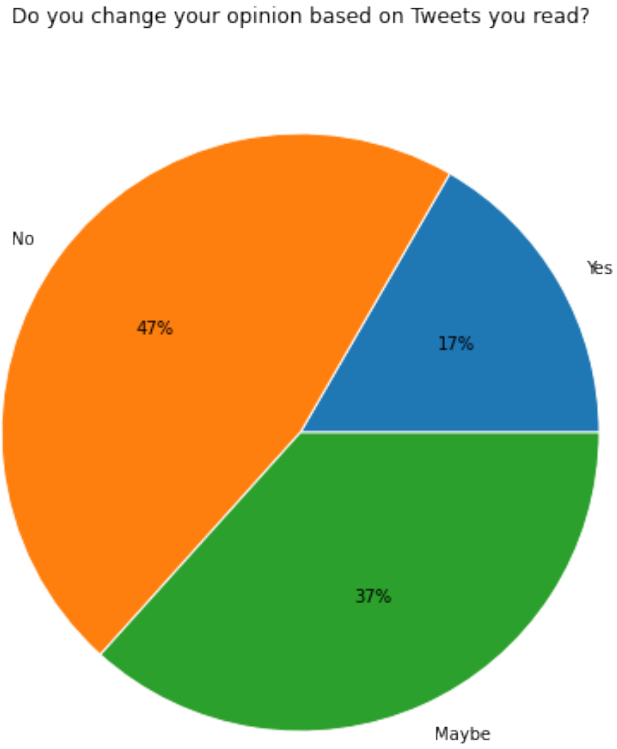


Figure 11.78: Question 10 Results

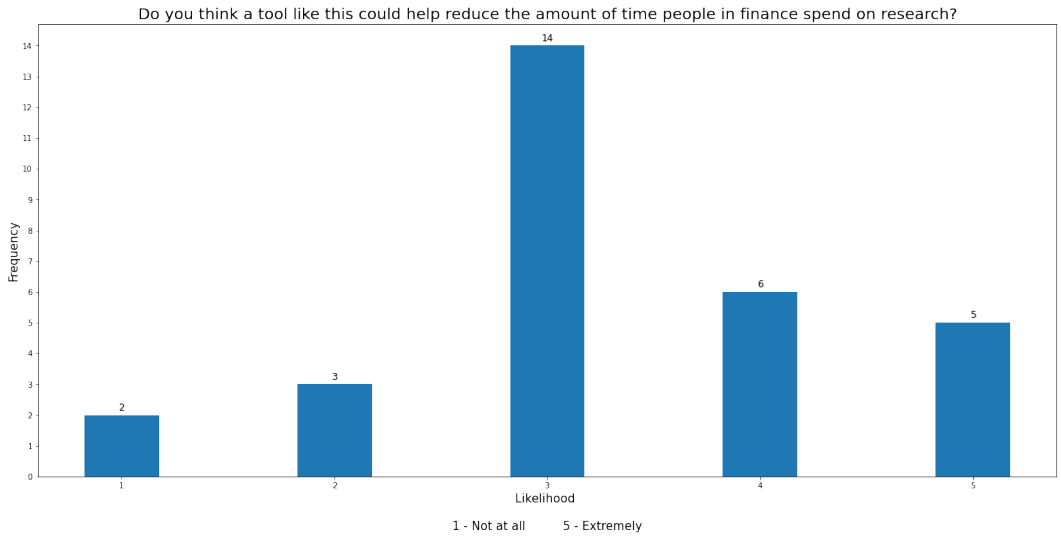


Figure 11.79: Question 11 Results

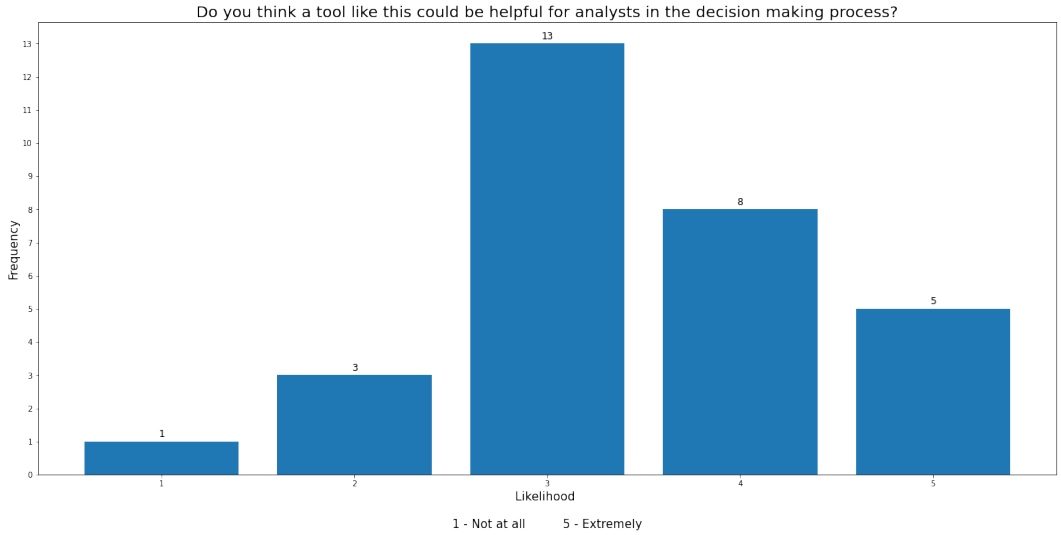


Figure 11.80: Question 12 Results

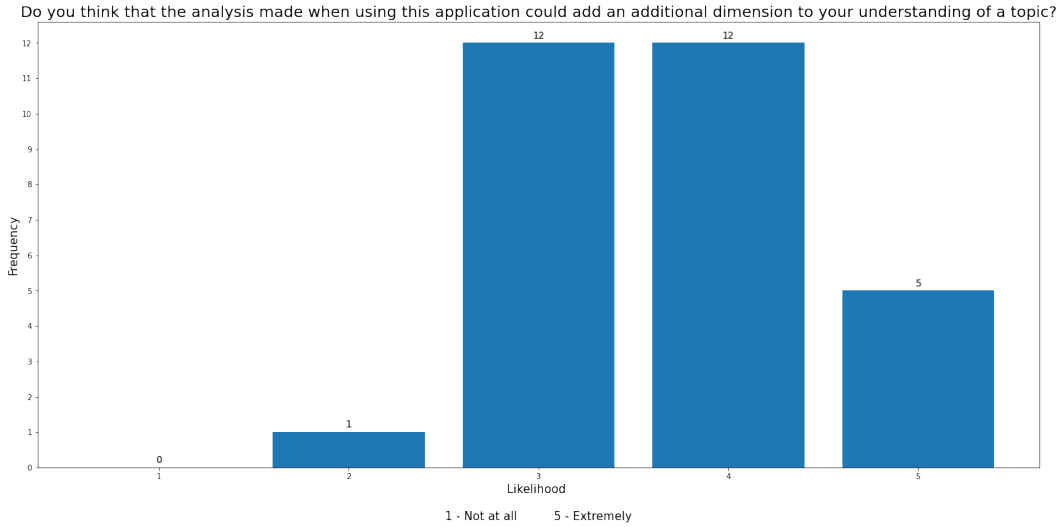


Figure 11.81: Question 13 Results

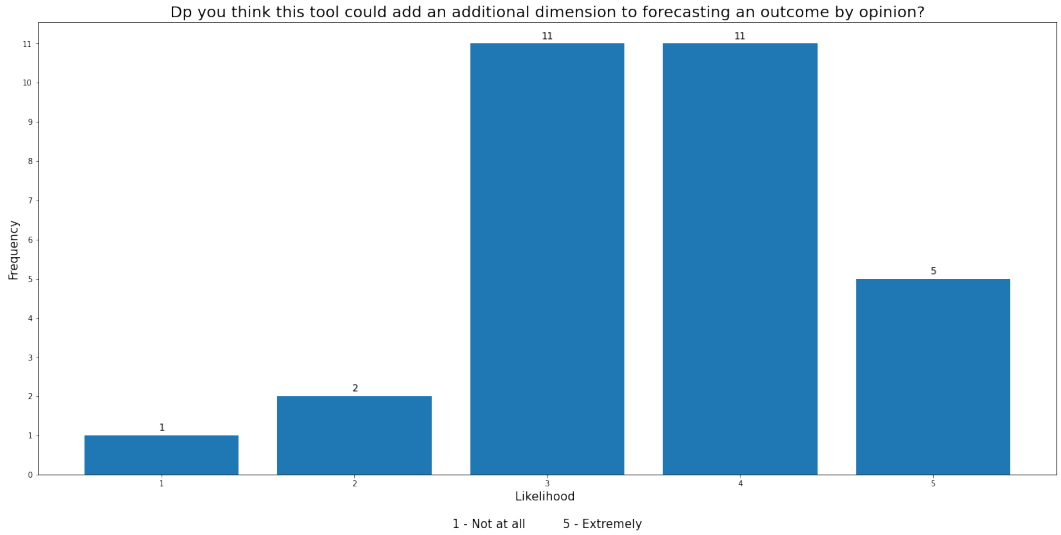


Figure 11.82: Question 14 Results

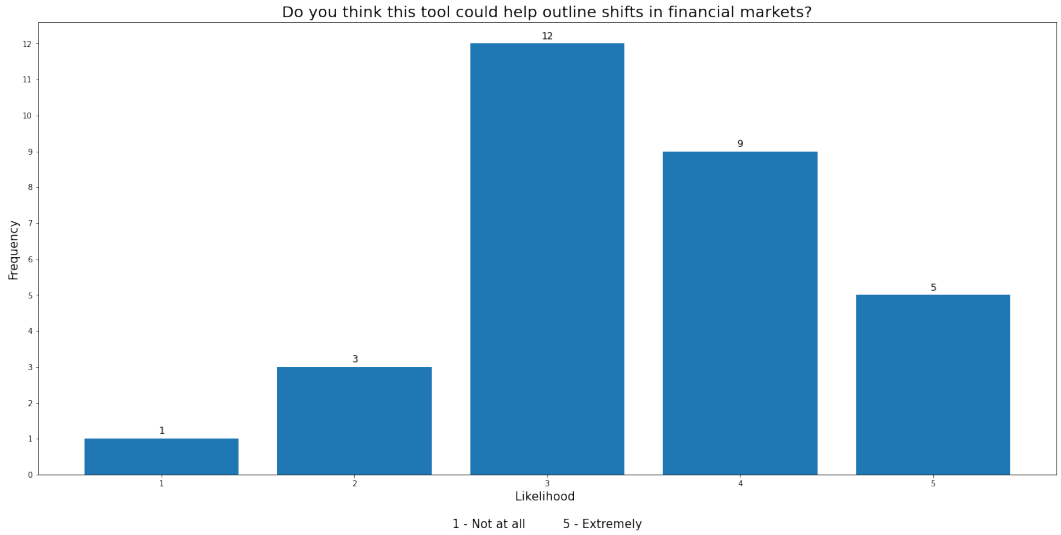


Figure 11.83: Question 15 Results

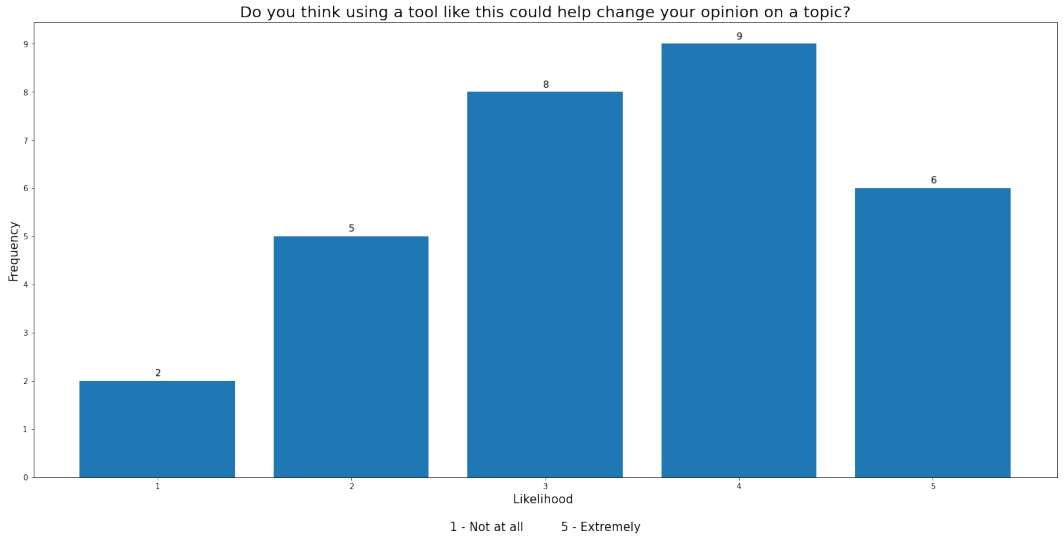


Figure 11.84: Question 16 Results

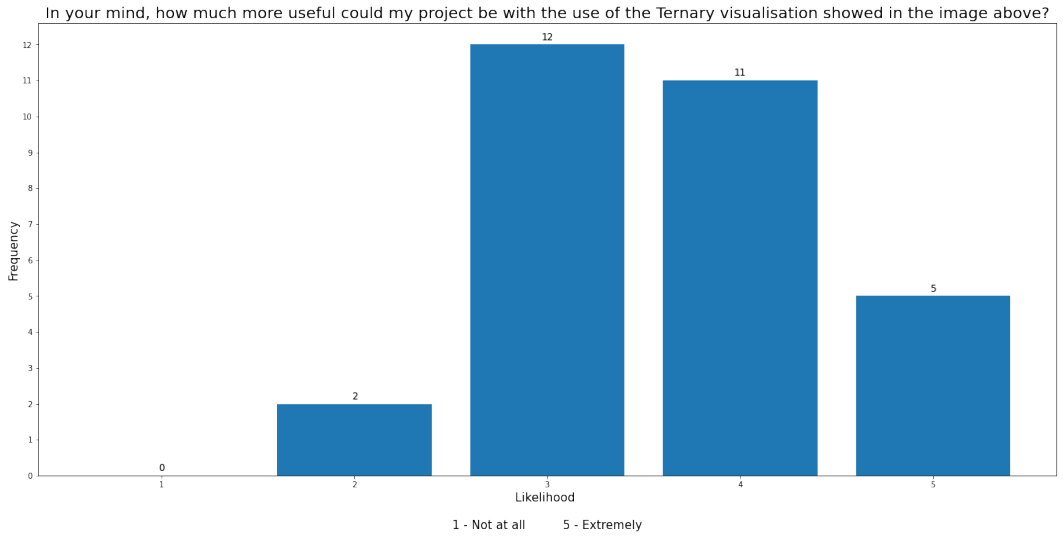
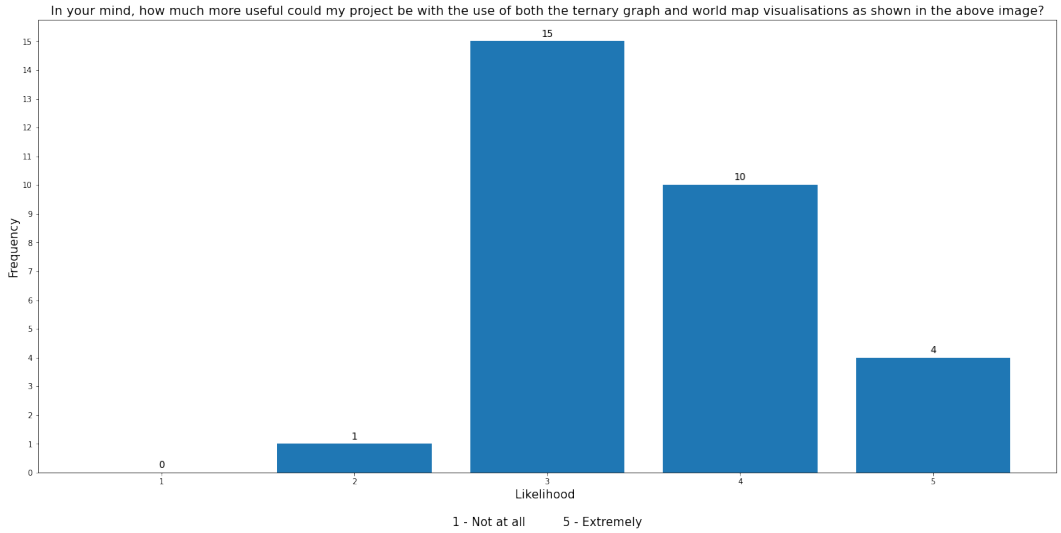


Figure 11.85: Question 17 Results



Back End app.py file

Figure 11.86: app.py file - Part 1

app.py 3, M

app.py > ...

```python
from flask import Flask,request,request,jsonify
import ast
import tweepy
import joblib
import re
import contractions
import emoji
import nltk
from nltk.tokenize import word_tokenize
import string
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
import dill as pickle
from flask_cors import CORS, cross_origin
from wordcloud import WordCloud

nltk.download('stopwords')

app = Flask(__name__)
CORS(app)

# keys needed to communicate with Twittter's API:
api_key = 'K9JI3yGwiGUacSdQTbUG6EjKt'
api_secret_key = 'FuxMofcCLkRm5gIZkLOnvKUliXTUAYcm8zZM8f1JzFDV6sGaRw'
bearer_token = 'AAAAAAAAAAAAAAAAAAAAAMqPWAEAAAAAkfR2tuTUyfEjQVg8rJnMXJE0tew%3D4G4asHdTXh9hcTvv0oqnq6kLUU6W4V6bbe6bGxJEzRGcxnoWQH'
access_token = '1470048487-TXkyFfTABq3wdywcvd59ETePhZyXMZDGOzZUxtV'
access_token_secret = 'z9f5F41mDGTostzEBQFO0RCXyvJX1HvTA2h3jD9jbGFxA'
vps_directory = '/Users/louiskampman/Desktop/Goldsmiths 3rd Year/Final Project/Project'
num_of_tweets = 10

# method allowing us to create a client object
def getClient():
    client = tweepy.Client(bearer_token = bearer_token,
                           consumer_key = api_key,
                           consumer_secret = api_secret_key,
                           access_token = access_token,
                           access_token_secret = access_token_secret)
    return client

# search function to scrape tweets from twitter
def searchTweets(query, max_results):
    client = getClient()
    tweets = client.search_recent_tweets(query=query,
                                         tweet_fields=['public_metrics'],
                                         place_fields=['country'],
                                         max_results=max_results)
    tweet_data = tweets
    # tweet_data = tweets
    results = []

    if not tweet_data is None:
        for tweet in tweet_data.data:
            print("\n\n\n")
            obj = [None, None, None, None, None]
            obj[0] = tweet.id
            obj[1] = tweet.text
            obj[2] = tweet.public_metrics['retweet_count']
            obj[3] = tweet.public_metrics['reply_count']
            obj[4] = tweet.public_metrics['like_count']
            results.append(obj)
    else:
        return ''
    print("\n\n\n")
    for tweet in results:
        print(tweet)
        print("\n")
    return results

```

Figure 11.87: app.py file - Part 2

app.py 3, M

app.py > ...

```
# replace retweet tag
def replace_retweet_tag(tweet, default_replace=""):
    tweet = re.sub('RT\s+', default_replace, tweet)
    return tweet

# replace user tag
def replace_user_tag(tweet, default_replace=""):
    tweet = re.sub('\B@\w+', default_replace, tweet)
    return tweet

# replace url tag
def replace_url_link(tweet, default_replace=""):
    tweet = re.sub('(http|https):\/\S+', default_replace, tweet)
    return tweet

# replace hashtag
def replace_hashtag(tweet, default_replace=""):
    tweet = re.sub('#+', default_replace, tweet)
    return tweet

# to lowewrcase method
def to_lowercase(tweet):
    tweet = tweet.lower()
    return tweet

# convert contraction method
def convert_contractions(tweet):
  tweet = contractions.fix(tweet)
  return tweet

# remove punctuation repetition method
def punct_repetition(tweet, default_replace=""):
    tweet = re.sub('[\?\.\!]+(?=[\?\.\!])', default_replace, tweet)
    return tweet

# remove character repetition method
def char_repetition(tweet):
    tweet = re.sub(r'(.)\1+', r'\1\1', tweet)
    return tweet

# replace emojis
def replace_emoji(tweet):
    tweet = emoji.demojize(tweet)
    return tweet

# stopwords
stop_words = set(stopwords.words('english'))
stop_words.discard('not')

# tokenize function
def tokenize_function(tweet,
                      keep_punct = False,
                      keep_alnum = False,
                      keep_stop = False):
    token_list = word_tokenize(tweet)

    if not keep_punct:
        token_list = [token for token in token_list
                      if token not in string.punctuation]
    if not keep_alnum:
        token_list = [token for token in token_list if token.isalpha()]
    if not keep_stop:
        stop_words = set(stopwords.words('english'))
        stop_words.discard('not')
        token_list = [token for token in token_list if not token in stop_words]
    return token_list

```

Figure 11.88: app.py file - Part 3

app.py 3, M

app.py > ...

```
# snowball stemmer
snowball_stemmer = SnowballStemmer('english')

# stem tokens method
def stem_tokens(tokens, stemmer):
    token_list = []
    for token in tokens:
        token_list.append(stemmer.stem(token))
    return token_list


# process tweet function to normalise the tweet text
def process_tweet(tweet, verbose=False):
    if verbose: print("Original tweet: \n{}".format(tweet))
    ## Twitter Features
    tweet = replace_retweet_tag(tweet) # replace retweet
    tweet = replace_user_tag(tweet, "") # replace user tag
    tweet = replace_url_link(tweet) # replace url
    tweet = replace_hashtag(tweet) # replace hashtag
    if verbose: print("Post Twitter processing tweet:\n{}".format(tweet))
    ## Word Features
    tweet = to_lowercase(tweet) # lower case
    tweet = convert_contractions(tweet) # replace contractions
    tweet = punct_repetition(tweet) # replace punctuation repetition
    tweet = char_repetition(tweet) # replace word repetition
    tweet = replace_emoji(tweet) # replace emojis
    if verbose: print("Post Word processing tweet:\n{}".format(tweet))
    ## Tokenization & Stemming
    tokens = tokenize_function(tweet, keep_alnum=False, keep_stop=False) # tokenize
    stemmer = SnowballStemmer("english") # define stemmer
    stem = stem_tokens(tokens, stemmer) # stem tokens
    return stem

# preprocessor lambda
# def pre_processor_funct(x):
#     return x

# make a pridict tweet function using the model and process tweet functions
def predict_tweet(tweet):
    # call joblib and collect saved sentiment perdictor
    filename1 = 'model.sav'
    loaded_model = joblib.load(filename1)
    # call joblib and collect saved text vectorizer
    filename2 = 'tf.pkl'
    # loaded_vectorizer = joblib.load(filename2)
    with open(filename2, 'rb') as f:
        def pre_processor_funct(x):
            return x
        loaded_vectorizer = pickle.load(f)
    # processors
    processed_tweet = process_tweet(tweet)
    print("\033[1;35mProcessed tweet:\033[0m\n{}".format(processed_tweet))
    transformed_tweet = loaded_vectorizer.transform([processed_tweet])
    print("\033[1;35m\nTransformed tweet:\033[0m\n{}".format(transformed_tweet))
    prediction = loaded_model.predict(transformed_tweet)
    print("\033[1;35m\nSentiment prediction:\033[0m\n{}\n".format(prediction))
    return prediction


# test input for hashtag: #happy
hashtag = "#happy"

```

Figure 11.89: app.py file - Part 4

app.py 3, M

app.py > ...

```python
@app.route("/get_sentiment_analysis", methods=["POST","GET"])
def getSentiment():
    # GET IN SERVER
    reponse = jsonify(message="Server is running")
    # if request.data is present (entry was given), set hashtag to this entered string and analyse
    if request.data:
        global hashtag
        hashtag = request.data
        print("hashtag = request.data")
        print(hashtag)
        hashtag = hashtag.decode("UTF-8")
        print("\nPrinting request.data")
        print(request.data)
        hashtag = ast.literal_eval(hashtag)["hashtag"]
    print("\n\033[1;30;43m--------------------------------------------------------------------------------------------------------------\033[0m")
    print("\n\033[1;32mGET SENTIMENT RUNNING...\033[0;32m\n   " + hashtag + "\033[0m\n")
    # Recreate the python code found in the jupyter notebook for my twitter scraper
    tweets = searchTweets(hashtag, num_of_tweets)
    # Printing the contents of tweets (in a nice format)
    print("\n\033[1;32mScraping Tweets...\033[0m\n")
    for t in tweets:
        print(t[1] + "\n")
    # Now to push collected tweets into sentiment prediction model by using the model.sav document
    # array of same length as 'tweets' that stores binary sentiment values of 'tweets'
    sentiments = [None] * num_of_tweets
    # array of dictionaries
    results = []
    print("\n\033[1;32mProcessing tweet text and predicting sentiment...\033[0m\n")
    # iterate thorugh tweets and input index 1 of each array element
    for i in range(len(tweets)):
        print("\033[1;31mTweet {} / 10 \033[0m".format(i + 1))
        print("\033[1;30;41m--------------------------------------------------------------------------------------------------------------\033[0m")
        print("\033[1;35mPredicting sentiment of tweet text below...\033[0m\n{}\n".format(tweets[i][1]))
        # We will not include tweet ids in our data. Text will be passed into results twice (calculated sentiment and string of text)
        results.append({'Sentiment':str(predict_tweet(tweets[i][1])), 'Text':str(tweets[i][1]), 'Retweets': str(tweets[i][2]), 'Replies': str(tweets[i][3]), 'Likes': str(tweets[i][4])})
        print(results[i])
    jSon = jsonify(results)

    # THE CODE BELOW PRINTS INTO OUR TERMINAL THE RESULTS
    resultsVis = []
    print("\033[1;30;44m\n--------------------------------------------------------------------------------------------------------------\033[0m")
    print("\033[1;34m\nAll sentiment results collected:\033[0m")
    for i in results:
        resultsVis.append(i['Sentiment'])
    # iterate through string array of results and remove square brackets
    resultsNew = []
    for i in resultsVis:
        if i == '[1]':
            resultsNew.append(1)
        else:
            resultsNew.append(0)
    print(resultsNew)
    # print ratio
    print("\033[0;34mPositive / Negative ratio of results\033[0m")
    zeroCount = 0
    oneCount = 0
    for i in resultsNew:
        if i == 1:
            oneCount = oneCount + 1
        if i == 0:
            zeroCount = zeroCount + 1
    print("{} / {} \n".format(oneCount, zeroCount))

    # Enable Access-Control-Allow-Origin and return results.
    reponse.headers.add("Access-Control-Allow-Origin", "*")
    print(jSon)
    return jSon
```

Application Demonstration

Figure 11.90: Home Page - Before Search

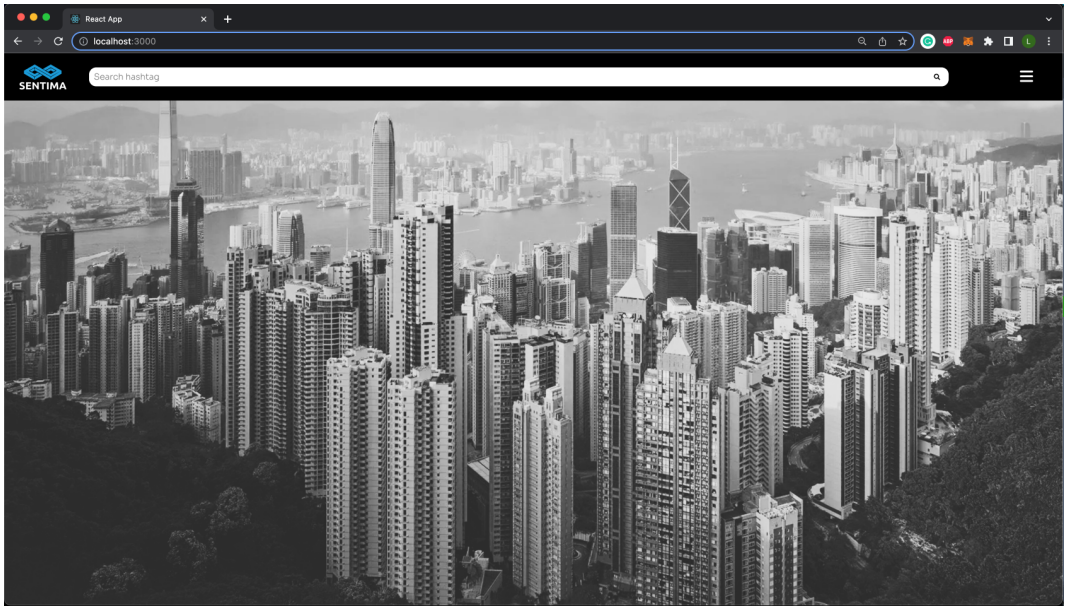


Figure 11.91: Home Page - Apple Search

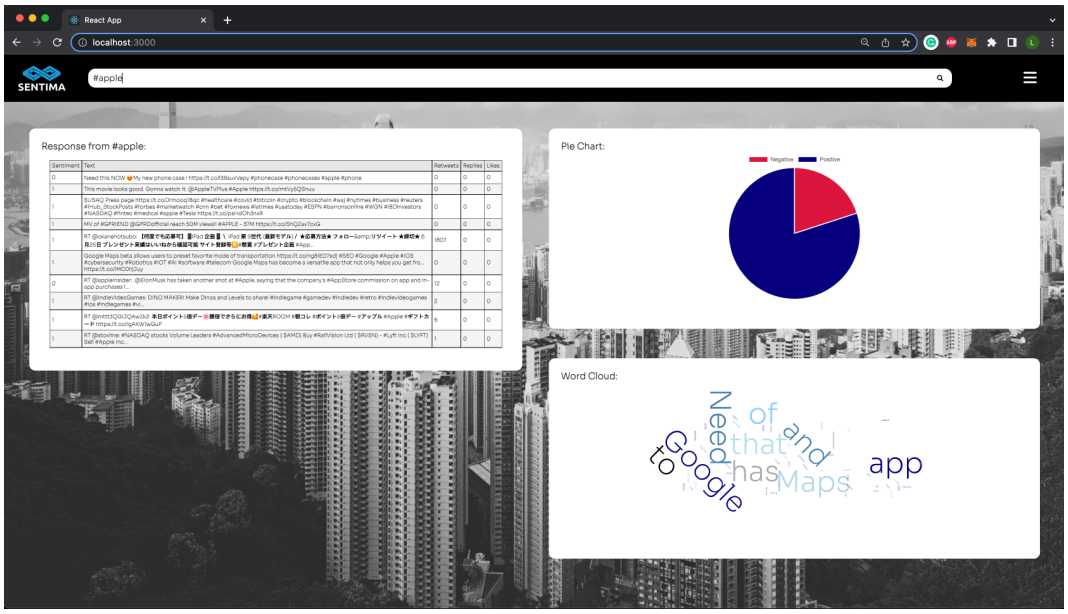


Figure 11.92: Home Page - War Search

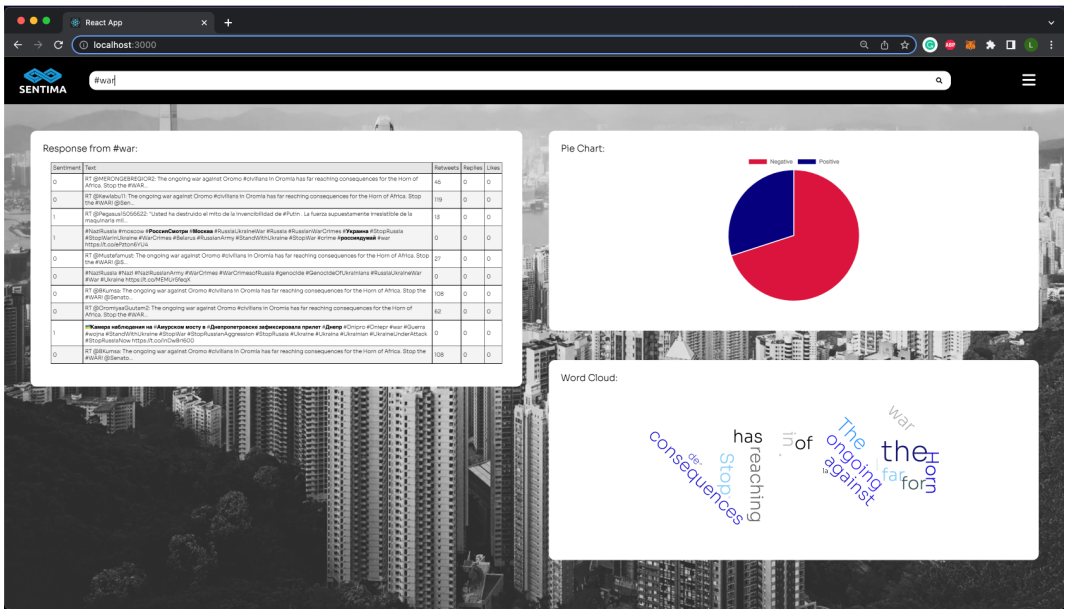


Figure 11.93: Home Page - Blockchain Search

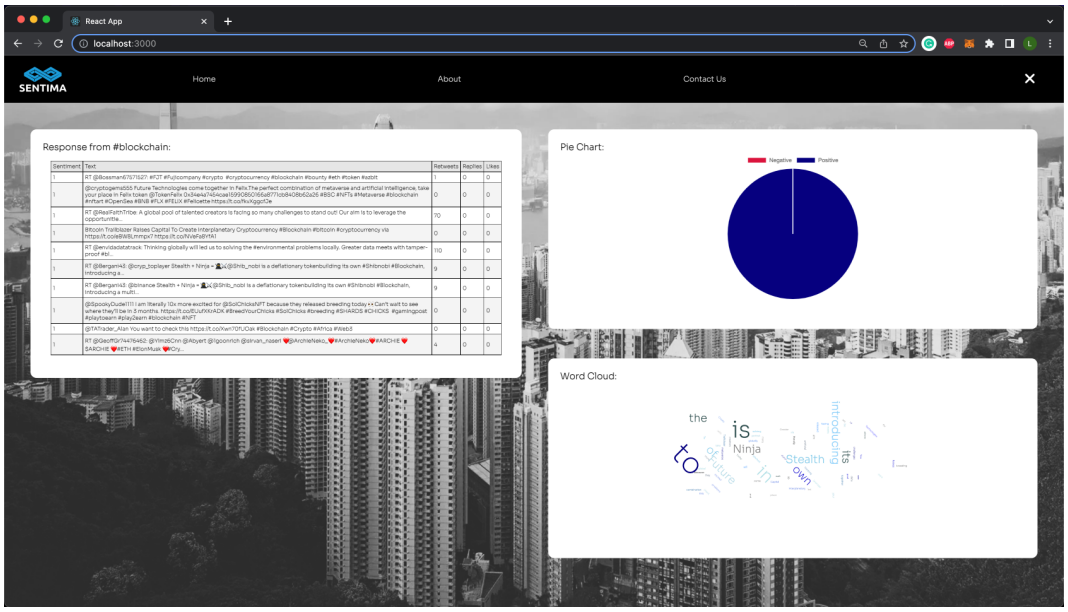


Figure 11.94: About Page

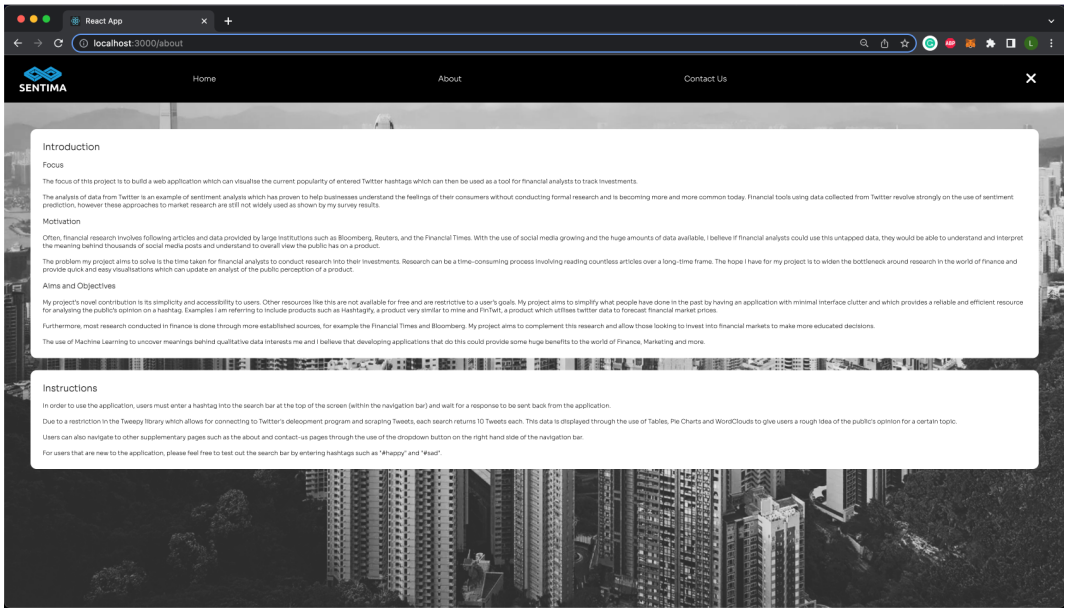


Figure 11.95: Contact Us Page

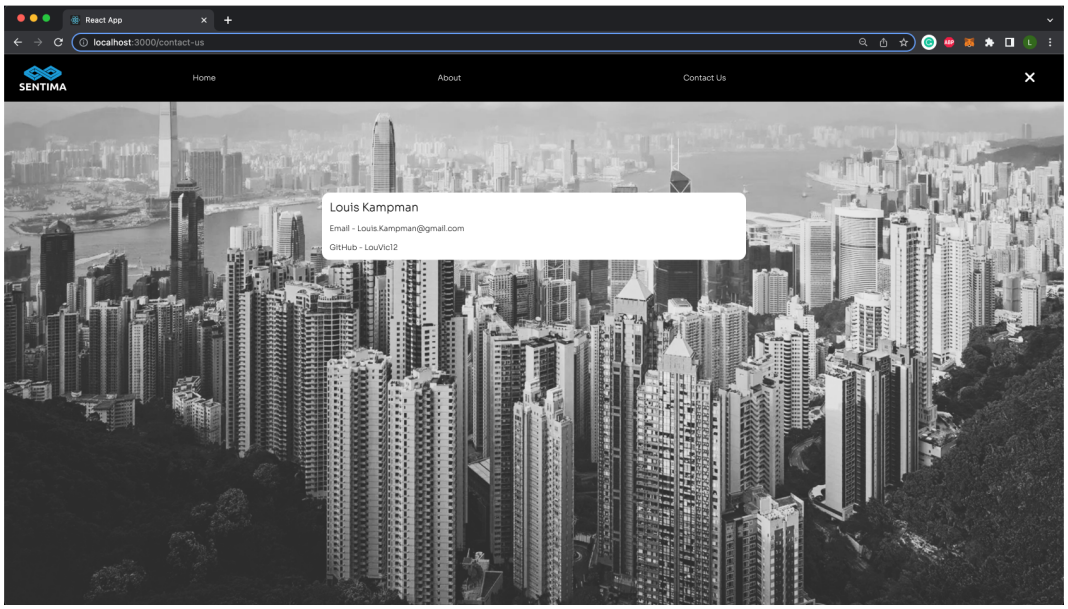


Figure 11.96: Mobile Hope Page - Part 1

Sentiment	Text
1	RT @Bossman67571527: #FJT #Fujicompany #crypto #cryptocurrency #blockchain #bounty #eth #token #azbit
1	@cryptogems555 Future Technologies come together in Felix.The perfect combination of metaverse and artificial intelligence, take your place in Felix token @TokenFelix 0x34e4a7454cae15990850166a8771cb8408b62a26 #BSC #NFTs #Metaverse #blockchain #nftart #OpenSea #BNB #FLX #FELIX #Felicette https://t.co/fkvXggcfJe
1	RT @RealFaithTribe: A global pool of talented creators is facing so many challenges to stand out! Our aim is to leverage the opportunitie...
1	Bitcoin Trailblazer Raises Capital To Create Interplanetary Cryptocurrency #Blockchain #bitcoin #cryptocurrency via https://t.co/eBW8Lmmpx7 https://t.co/NVeFa8YfA1
1	RT @envidadatatrack: Thinking globally will led us to solving the #environmental problems locally. Greater data meets with tamper-proof #bl...
1	RT @Bergani43: @cryp_toplayer Stealth + Ninja = 🥷⚔@Shib_nobi is a deflationary tokenbuilding its own #Shibnobi #Blockchain, introducing a...
1	RT @Bergani43: @binance Stealth + Ninja = 🥷⚔ @Shib_nobi is a deflationary tokenbuilding its own #Shibnobi #Blockchain, introducing a multi...
1	@SpookyDude1111 I am literally 10x more excited for @SolChicksNFT because they released breeding today 👀Can't wait to see where they'll be in 3 months. https://t.co/EUufXKrADK #BreedYourChicks #SolChicks #breeding #SHARDS #CHICKS #gamingpost #playtoearn #play2earn #blockchain #NFT
1	@TATrader_Alan You want to check this https://t.co/Xwn70fUOak #Blockchain #Crypto #Africa #Web3
1	RT @GeoffGr74476462: @Ylmz6Cnn @Abyert @1goonrich @sirvan_naseri ❤️@ArchieNeko_❤️ #ArchieNeko❤️#ARCHIE ❤️$ARCHIE ❤️#ETH #ElonMusk ❤️#Cry...

Figure 11.97: Mobile Home Page - Part 2

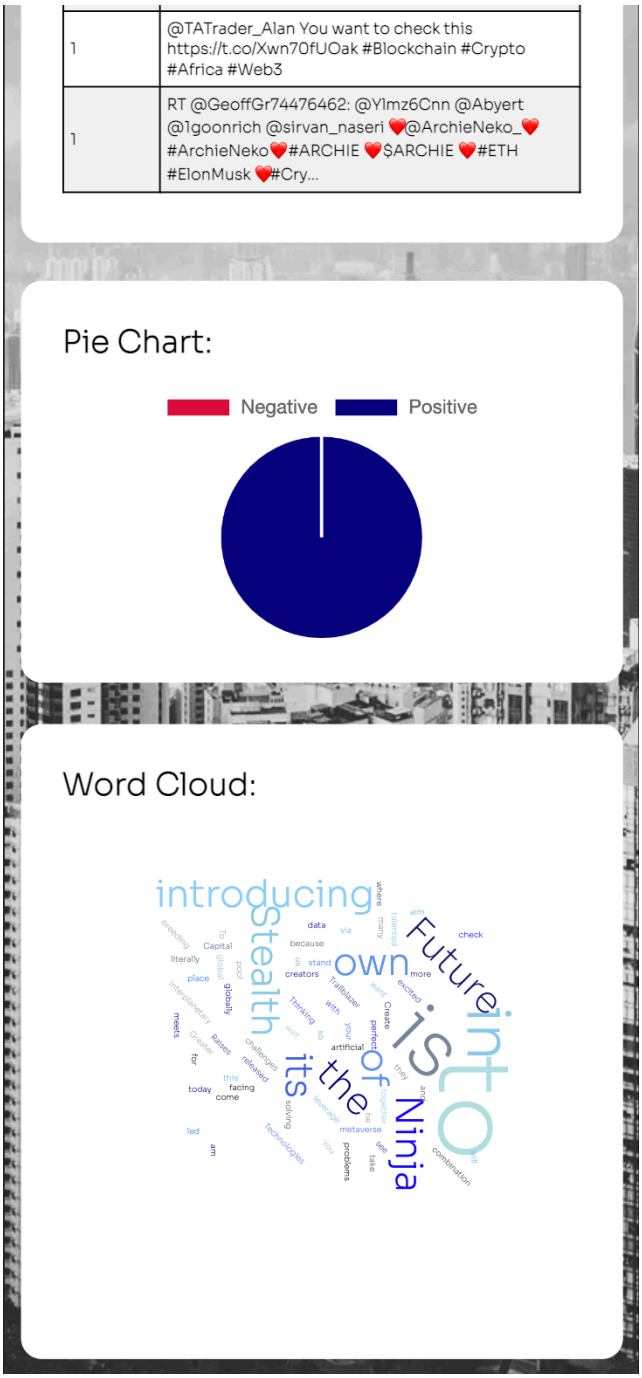


Figure 11.98: Mobile About Page - Part 1

Figure 11.99: Mobile About Page - Part 2

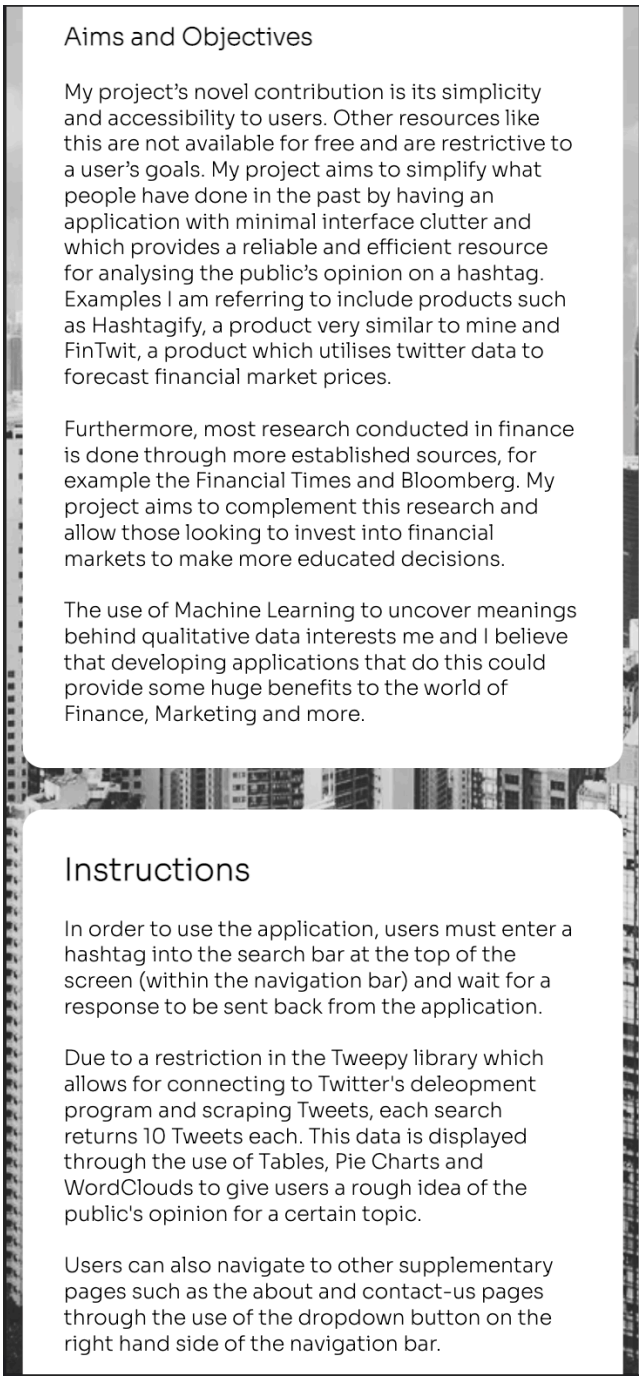


Figure 11.100: Mobile Contact Us Page