**Department of Computer Science**

Individual Project - CS3IP16

# An interface for performing Sentiment Analysis on Twitter

Student name: Yiannis Hadjicharalambous

Student number: 25015646

Supervisor: Dr Frederic Stahl

Date of submission: 29th April 2019

## Abstract

With the eternally growing number of social media users, arises a perpetually large amount of chaotic, unorganized data. To make sense of this disarray, a Graphical User Interface, that performs sentiment analysis on Twitter statuses, has been implemented. It makes a prediction on the polarity of a given subject area and provides some analytics on the word frequency of this query. This report will investigate the research undertaken, how the problem’s solution has been conceptualized, implemented and tested, the steps and thought process behind the implementation and reflections on the finished solution.

## Acknowledgments

I would like to thank my supervisor Dr Frederic Stahl for his consistent support and guidance throughout this project. With his invaluable help, this project became extremely more manageable.

## Table of Contents

## Glossary of Terms and Abbreviations

API – Application Programming Interface

CLI – Command Line Interface

GUI – Graphical User Interface

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

NLP – Natural Language Processing

NLTK – Natural Language Tool Kit

PID – Project Initiation Document

Regex – Regular expression

# 1 Introduction

Approximately 80% of the world's data is unorganized, much of that in textual form such as emails, support tickets, chats, social media, surveys, articles, and documents. Manually sorting through it all would be difficult, expensive, and unbearably time-consuming [1]. Using sentiment analysis allows us to make sense of this chaos through automation, providing interesting insights that would otherwise be unattainable. Sentiment analysis is a process of identifying and categorizing opinions in text.

Data Science includes methods, technologies and algorithms used to make sense of the large amount of digital data that is being continuously generated. The aim of this project was to comprehend and appreciate different aspects of Data Science and Data Analytics, in specific. By developing an application that could be used in a real-world scenario, it would give me first-hand experience of what it is like to develop a robust and complex system that is expected to have an interface for end-users.

With big social data analytics, we can understand much more about social phenomena and their impact that until now has been very difficult or impossible to measure. Twitter is a microblogging platform that has been recently reported to have over 300 million monthly active users all over the globe. In the UK it is estimated that there are 14 million monthly active users, which is about 24% of the UK population, over the age of 13 [2].

The personal motivation to undertake this particular project is that I have had an interest in data science and analytics for quite some years now. By going through with this project, it would satisfy this lust of mine to enter this, then, unknown territory of data science and decide if this field is something I would want to continue working on and possibly pursue a career in.

This report will discuss what the problem is in addition to thoroughly scrutinize how it has been solved, implemented (e.g. which practices, techniques and algorithms were used) and tested, as well as possible real-life applications, reflections and potential future improvements on the finished product.

# 2 Problem Articulation & Technical Specification

## 2.1 Problem Articulation

The problem that this project approaches is analysing as well as visualizing big data streams. An article from 2018 states that 2.5 quintillion bytes of data were created each day and that "over the last two years alone 90% of the data in the world was generated" [3]. Every day 500 million Tweets are tweeted and almost 5 billion YouTube videos are watched. On a daily basis, these numbers are growing bigger as the number of Internet users keeps increasing. Clearly, big data streams are commonly found on social media platforms where data is constantly generated by the millions of active users.

Automation is something every computer scientist strives for. Being able to automate human sentiment is a process that has many use cases in commercial as well as academic environments.

Companies can use sentiment analysis to help determine the emotional tone behind a string of words expressed on an online platform and as such better advertise or analyse feedback on their product or even forecast market movement based on public opinion.

Academically, as well as individually, understanding human emotion through text has been a hard task to accomplish. Specific aspects such as multilingual sentiment analysis, emotion detection, context detection, sarcasm detection, Internet slang detection are challenging topics to this day.

## 2.2 Stakeholders

Every project has some stakeholders that have or are, in their own different means, invested in it. The key stakeholders of this project are:

- Yiannis Hadjicharalambous, me. As the developer of this solution I have full responsibility of providing a solution to this problem, by meeting the requirements and producing the outputs that have been mentioned in the PID.
- Dr Frederic Stahl. As this project's supervisor, he has reliably monitored this project's progress from start to finish and has provided ideas using his expertise on Big Data Analytics.

## 2.3 Technical Specification

The requirements that the solution should be developed against (the technical specification) are:

- The solution must provide an intuitive interface for the end-user.
- The solution should be as robust and bug-free as possible.
- The solution must be able to successfully authenticate and access the Twitter API.
- The developer needs to develop or implement an existing sentiment analysis model, based on a classification algorithm.
- The solution must be able to, in real time, grab real time Tweets based on a user input keyword and simultaneously visualize their sentiment score.
- The solution must be able to grab existing Tweets again based on a user input keyword and visualize their sentiment score.
- The solution must be able to grab existing Tweets based on two user input keywords and visualize the comparison in their sentiment score.
- The user must be given several recency options when searching existing Tweets.

- The solution must be able to grab existing Tweets from a Twitter user, requesting the user to input the account's Twitter handle (e.g. realDonaldTrump) and visualize their sentiment score.
- A case study of a reasonably large number of collected Tweets must be conducted and be available to the user for reference.
- The visualized output should also include a moving average to smooth out short-term fluctuations.

To summarize, this project implements a system that provides an interface for, among other features, ingesting real time Tweets as well as previously collected Tweets, predicting a sentiment score and visualizing the accumulated results. A sentiment analysis approach has to be used that scores and classifies the Tweets. A possible model that could be used is an analyser that would give back a sentiment score in the interval [-1,1] with -1 for very negative, 0 for neutral, and for +1 very positive, which is the probability distribution of the polarity of the sentiment returned by the model. After the back end has been developed, a GUI needs to be in place for to ease the end-user into the system.

## 2.4 Project Constraints

However, there is one major constraint. The most popular platforms' (e.g. Facebook, Instagram, Twitter, YouTube) APIs usually fall into two categories:

- Free, which are intended for the community and have very limited accessibility and requests.
- Paid, which are intended for enterprises and have almost unrestricted accessibility and requests.

For this project the free Twitter API is used which, as mentioned above is very constraining. For example, there is a rate limit of 18,000 requests for searching Tweets, in addition to being limited to 7-day old Tweets at most, every 15 minutes [4]. This limits the number of existing Tweets the system can extract and perform analysis upon.

# 3 Literature Review

By now, it is clear that there are three main objectives or key features for this system. It should analyse sentiment, it should do it on a social media network where a lot of real-time data is constantly created and lastly it should provide a, preferably graphical, user interface for the end-user to view the visualized results.

Exhaustive research of existing pertinent literature was carried out before development of the program such that techniques that this project could potentially benefit from can be implemented and bad practices be avoided.

## 3.1 Sentiment Analysis

There is a great number of machine learning techniques that can perform sentiment analysis. The paper from the Indian Institute of Technology, written by Subhabrata Mukherjee [5], thoroughly examines sentiment analysis and compares a number of approaches to it. It states the many different challenges for sentiment analysis, some of which were mentioned above. Sarcasm, context, world knowledge (as mentioned in the paper, consider the statement: “He is a Frankenstein.”), entity identification (for example “Windows is better than Linux” is positive for Windows but negative for Linux) and handling negation are some of the problems that need to be dealt with when trying to approach sentiment analysis.

Before evaluating the different algorithms and approaches, it is important to use the right metric. Accuracy alone doesn’t always tell the full story. Metrics such as Cohen’s kappa (κ), precision and recall and F1 score could prove to be more trustworthy statistic measures since they take into account a broad set of parameters.

The paper focuses on three approaches: Naïve Bayes classifier, Logistic Regression, and Support-Vector Machines. They all perform quite well, with each being better in specific scenarios. For example, Naïve Bayes performs the best without boosting. However, a big drawback is that all of these alone are bag-of-words models. As a result, they do not capture context and do not precisely analyse the discourse which is crucial for sentiment analysis.

Another paper that influenced how the solution must be approached, from the Institute for Research on Population and Social Policies (IRPPS) [6], briefly describes the different steps that are involved when analysing sentiment data, as well as a thorough classification of the different sentiment analysis approaches with respect to their features/techniques and advantages/limitations.

The paper introduces the three main sentiment analysis approaches: machine learning, lexicon based and hybrid, as such:

- Machine learning, as investigated above, uses classification algorithms to classify text; it consists of two sets of documents: a training and a test set. The training set is used for learning the differentiating characteristics of a document, while the test set is used for checking how well the classifier performs.
- The lexicon-based approach uses sentiment dictionary with opinion words and matches them with the data for determining polarity. The paper introduces three techniques to construct a sentiment lexicon: manual construction, corpus-based methods and dictionary-based methods. The manual construction is a difficult and time-consuming task. Corpus-based methods can produce opinion words with relatively high accuracy. Finally, in the dictionary-based techniques,

the idea is to first collect a small set of opinion words manually with known orientations, and then to grow this set by searching in the WordNet dictionary (a lexical database) for their synonyms and antonyms.

- Last but not least, the hybrid approach. It is a combination of both the machine learning and the lexicon-based approaches which has the potential to improve the sentiment classification performance.

Figure 1, below, condenses this information in a succinct diagram:

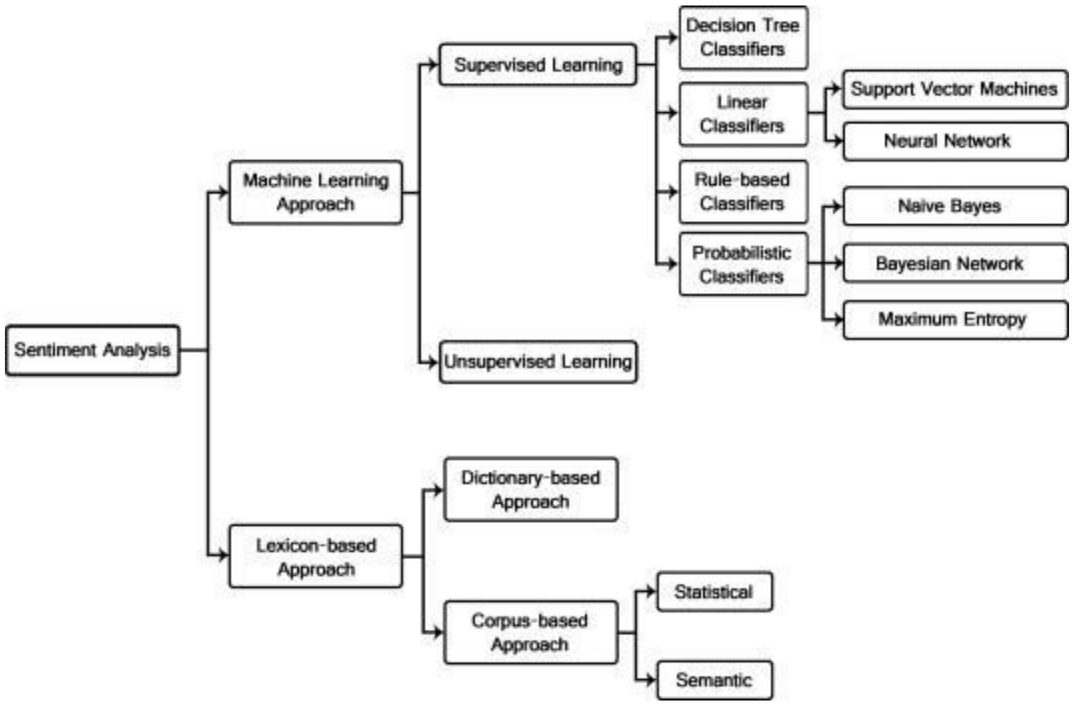


*Figure 1. Sentiment analysis approaches*

The paper then stresses the advantages and limitations of each approach. The main advantage of machine learning approaches is the ability to adapt and create trained models for specific purposes and contexts, while the limitation is that it is difficult integrating into a classifier, general knowledge which may not be acquired from training data. Furthermore, learnt models often have poor adaptability between domains or different text genres because they often rely on domain specific features from their training data.

Lexicon-based approaches have the advantage that general knowledge sentiment lexicons have wider term coverage, however these approaches have two main limitations. Firstly, the number of words in the lexicons is finite, which may constitute a problem when extracting sentiment from very dynamic environments. Secondly, sentiment lexicons tend to assign a fixed sentiment orientation and score to words, irrespective of how these words are used in a text.

The main advantages of hybrid approaches are: the lexicon/machine-learning cooperation, the detection and measurement of sentiment at the concept level, and the lesser sensitivity to changes in topic domain. While the main limitation is that reviews are with a lot of noise (irrelevant words for the subject of the review) are often assigned a neutral score because the method fails to detect any sentiment.

The paper also briefly touches on the steps required to analyse the data, such as data collection, text preparation, sentiment detection, sentiment classification and presentation of output, all of which should be considered when developing the solution.

The main takeaway from this paper, is that the hybrid approach looks very promising and has the potential to improve the sentiment classification performance, which is a possible solution to the problem that has been considered. The concept of using stop words was also introduced. Stop words are words which are ignored in NLP, usually the most common words such as "the, "to", "a", "is".

## 3.2 Sentiment Analysis on Twitter

Now, sentiment analysis on an online platform used by hundreds of millions requires a more complex approach and as the paper from Prerna Mishra, Ranjana Rajnish and Pankaj Kumar explains [7], analysis of opinions & its classification on the basis of polarity (positive, negative, neutral) is a challenging task. A lot of work has been done on sentiment analysis of Twitter data and a lot more needs to be done. Pre-processing is an essential step when dealing with big data, especially data that comes from a social platform. Tasks like stemming or lemmatization, stop words and tokenization are necessary in order to prepare the data for processing. After pre-processing has taken place, feature extraction happens. Part-of-speech tagging, term frequency or term co-occurrence can be performed to then be able to apply sentiment analysis. Finally, the model is trained on a predefined dictionary of positive and negative words and then classifying the extracted features into. The paper uses NLTK which has built-in libraries for the aforementioned tasks making it very easy to manipulate the data as well as build a model.

This paper has implemented a really strong framework, all things considered, and it has influenced this project quite a lot. For instance, pre-processing and feature extraction are definitely going to benefit this project and the platform, NLTK, is very easy to use with a great range of functionality. However, this paper could have implemented a better model, making the current solution and results look very ad hoc-like.

Another paper on sentiment analysis on Twitter that has helped me to better understand the whole process, comes from Paramita Ray and Amlan Chakrabarti [8]. More specifically it has been very informative on topics such as pre-processing and the lexical approach to sentiment analysis. It has, very clearly, brought forward and explained aspects of Tweet pre-processing that had not been examined. For example, identify when a Twitter user is being mentioned when there's a '@' symbol before a username, or adding an emoticon to a synset, that is a set of synonyms, and in a way giving it an emotional tag.

For the sentiment analysis, a lexical approach has been used, more specifically a dictionary-based approach. The approach depends on finding words from Tweets, and then matches the word with the dictionary. If there is a positive match, the positive score is incremented, or the word is tagged as positive. If it is a negative word, then the negative score is incremented, or else the word is tagged as negative. Otherwise, the word is tagged as neutral. If a negation comes across then the opposite is done, i.e. increment the negative score for positive words and the positive score for negative words. After some more steps, such as expanding acronyms and replacing emoticons with the corresponding synset, the overall score of the Tweet is calculated. This is calculated by subtracting the negative scores from the positive scores.

Lexical analysis is an interesting approach to sentiment analysis and could be worth considering if the dictionary is extended with slang and acronyms. However, the time taken to develop such a dictionary must be considered as this can be a very lengthy task.

## 3.3 Interface

A possibility of using a web framework to host the interface was briefly considered. Frameworks like Dash, Flask and Django were explored. However, simpler alternatives that do as good a job as the aforesaid have been revealed.

Other than web frameworks, there are two widely used libraries for creating interfaces: TkInter [9] and PyQt [10]. Both are Python bindings to their respective GUI toolkits. Pictures and videos of programs with interfaces that have been developed with these two libraries were examined.

TkInter, from Tk Interface, is Python's de facto standard GUI and is included in installs of Python. Tkinter is free software released under a Python license.

PyQt, from the GUI toolkit Qt, can be implemented as a Python plug-in. Qt includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer.

PyQt can make an appealing interface with not too much trouble using Qt Designer, however the code might sometimes be hard to follow, while TkInter is simpler, however not as flashy, and very light to run. Both libraries provide very similar functionalities and were both considered when developing the interface part of the solution.

# 4 The Solution Approach

This section will identify and evaluate the possible solution options against the problem statement and technical specification. A detailed justification of the chosen solution approach will be given.

## 4.1 Twitter API

First off, access to the Twitter API is required so that data collection can begin, which is the first step of sentiment analysis. Therefore, in order to get access to the API, access to a developer account needs to be applied for. After being reviewed and accepted, keys and tokens can be generated, which will allow use of the Twitter API. As already mentioned, Twitter API allows for both streaming real-time Tweets and searching existing Tweets, which are the two main areas of interest for this project.

## 4.2 Programming Language

Python is one of the most popular programming languages for data science at the moment, mainly due to being intuitive and providing incredible code readability. R could also be considered for this project. It provides a lot of libraries for data visualization and analysis. The language implementation used must also be considered. For example, Python has many implementations, CPython (the reference implementation), PyPy (written in Python itself) and Cython (designed to give C-like performance with code that is written mostly in Python with optional additional C-inspired syntax) are some of the most popular ones. Python has a plethora of libraries for all kinds of uses, especially in the data science areas that this project is concerned with: data manipulation, visualization, analytics and machine learning. Furthermore, it has a number of wrappers for the Twitter API and remarkable tool kits for NLP.

## 4.3 Sentiment Analysis Approach

After data collection, some pre-processing has to happen. It consists of cleaning the extracted data before analysis. Non-textual contents and contents that are irrelevant for the analysis are identified and eliminated. After the text is prepared, sentiment classification can begin.

A number of algorithms and approaches were reviewed. Machine learning, rule-based and hybrid are the three most commonly known and used approaches to sentiment classification.

Machine learning approaches deal sentiment analysis as a classification problem where a classifier is given a text and returns the corresponding class, for instance, positive, negative, or neutral. The classifier needs to be trained and in order to do that it needs a large labelled training set. The set is fed into the machine learning algorithm to generate a model. Subsequently, we predict unseen data by feeding it into the model, which generates predicted tags [11].

A number of classification algorithms have been investigated and the best performing ones are:

- Naïve Bayes classifier: a family of probabilistic algorithms, applying Bayes' theorem.
- Support-Vector Machines: a non-probabilistic linear classifier.
- Maximum Entropy classifier: performs multinomial logistic regression.

Figure 2 illustrates a high-level view of the steps in the machine learning sentiment analysis approach:

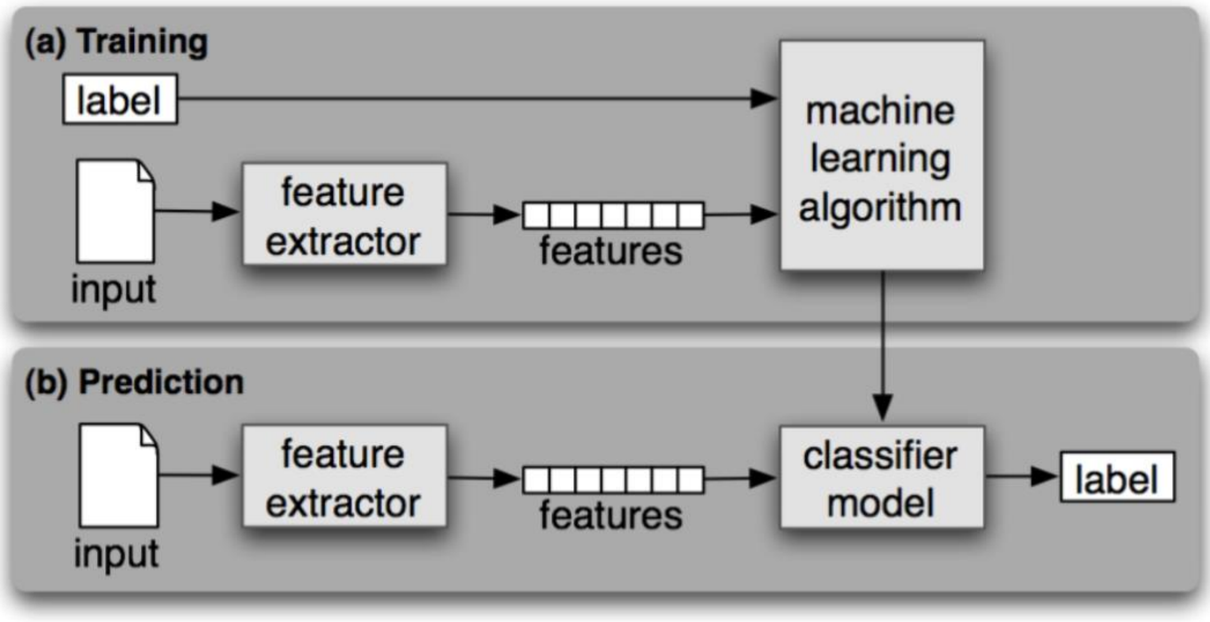


*Figure 2. A high-level view of how a machine learning approach works*

Rule-based approaches define a set of rules that identify subjectivity, polarity, or the subject of an opinion. The rules may use a variety of inputs, such as:

- Classic NLP techniques like stemming, tokenization, part of speech tagging and parsing.
- Other resources, such as lexicons.

This system is quite naïve as it doesn't take into consideration how words are combined in a sequence. A more advanced processing can be made, but these systems get very complex quickly. It can be very hard to maintain since new rules may be needed to add support for new expressions and terminology. Besides, adding new rules may have undesired outcomes as a result of the interaction with previous rules. As a result, these systems require important investments in manually tuning and maintaining the rules.

Lastly, hybrid approaches. By combining both aforementioned approaches, the methods can improve cancel out most of the limitations in each of the former approaches. Hybrid approaches exploit both machine learning and elements from rule-based representation, through the analysis of concepts that do not explicitly convey relevant information, but which are indirectly linked to other concepts that do so.

## 4.4 Presentation of Output

Python has a wide range of libraries that provide data visualization, matplotlib and seaborn are two popular and reliable examples, with a variety of different diagram types such as, graphs, histograms, scatter plots, time series, pie charts, bar charts, heatmaps, etc. A library called wordcloud does what its name infers, creates word clouds given a string of text.

It is expected that a GUI shall be provided for the end-user. TkInter and PyQt are the two libraries that fit for use in this project due to their ease of use.

## 4.5 IDEs & Version Control

There's a number of IDEs that support Python. Visual Studio, Visual Studio Code, Vi/Vim, Eclipse are just some. IDLE is an IDE for Python, which is bundled with the default implementation of the language. Finally, another IDE that stands out is PyCharm. It is developed by JetBrains specifically for the Python language.

Version control is an integral part of every project. GitHub and GitLab are web-based hosting services for version control using Git. Git is a distributed version-control system for tracking changes in source code during software development. The university has its own Git repository server at csgitlab.reading.ac.uk.

## 4.6 Definition of Solution Approach

The final solution approach will be detailed and justified here. Python has been chosen as the programming language to use to implement this project. As mentioned above, it is ideal for this project as it provides an overwhelming number of libraries for data science, analytics and visualization. The free developer API plan will be used, and data will be collected using a Python wrapper for the Twitter API, called tweepy.

After data collection, each Tweet will be cleaned and prepared for analysis using a regular expression. It will then be stored in a pandas DataFrame, rather than a list, to make data processing an easier task.

As for sentiment analysis, the package TextBlob is, as dubbed, the prince of NLP (with NLTK being the king) and will be used in this project. It combines NLTK and Pattern, a package with tools for NLP, among others. TextBlob has two built-in sentiment analysers: PatternAnalyzer (mainly lexicon and rule based) and NaiveBayesAnalyzer (Naïve Bayes classifier based on a NLTK corpus of labelled movie reviews). If an approach that includes machine learning is to be chosen, using TextBlob, the analyser has to be trained every time the program is run. This means that, not only will it take some time to give accurate results, but the performance hit will be massive. The chosen analyser is PatternAnalyzer, based on the WordNet lexical database. It is faster than the NaiveBayes Analyzer and, based on testing with both the results are very similar. The analyser could be trained beforehand but that could still mean that the results might not be as accurate. Additional detail about the two sentiment analysers will be given in the implementation section.

Matplotlib will be mainly used to visualize the analysed results as well as wordcloud to generate a word cloud. TkInter has been chosen over PyQt, mainly because of personal preference on simplicity and being lighter to run, which is of paramount importance on projects that deal with data analytics. PyCharm has been chosen as the IDE to develop the solution in. It provides code analysis, a graphical debugger, an integrated unit tester. PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactoring and rich navigation capabilities, making it ideal for this project. Lastly, to address version control, code has regularly been committed to the university's GitLab repository server.

# 5 Implementation

This section will thoroughly examine the steps that were undertaken in order to initially create a theoretical premise of the system's design, and finally implement the solution as a complete system.

## 5.1 Design

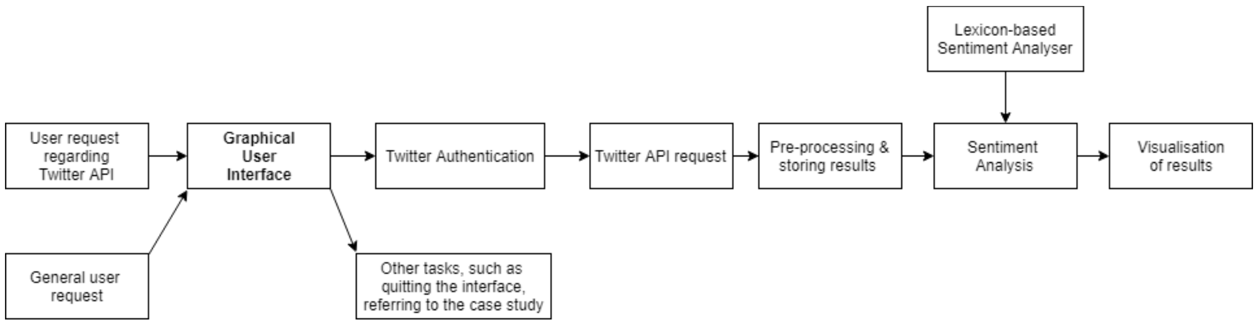

Lexicon-based Sentiment Analyser
User request regarding Twitter API
Graphical User Interface
Twitter Authentication
Twitter API request
Pre-processing & storing results
Sentiment Analysis
Visualisation of results
General user request
Other tasks, such as quitting the interface, referring to the case study

*Figure 3. A high-level diagram showing the structural outline of the processes in the system*

As shown in Figure 3, a high-level view of how the system should work, has been produced.

The first thing the user will see is the GUI, built on TkInter, with all available choices to them. If the user wishes to perform sentiment analysis on some Twitter data, authentication has to take place. Twitter requires third-party applications to use OAuth (Open Authorization), a standard for token-based authentication and authorization. OAuth does this by allowing access tokens to be supplied to third-party clients without sharing user credentials. After authentication has been successful, a request is made to the free Twitter API using the Python API wrapper. After the Tweets have been collected, they are cleaned using a regex. The cleaned tweets are fed to the sentiment analysis model which, as mentioned above, is a lexicon-based analyser. Finally, the results are visually represented to the user. Any requests that do not require data collection, or manipulation, do not need to go through the aforesaid steps.

The interface layout is mainly conceptualized on the idea that all available functions should be shown to the user on the main window. In some projects this would be overwhelming to the user but, given that a variety of functionalities was considered and only the most suitable and informative for the user were selected for implementation, this solution implements the most refined elements and functions justifying the need to show them all when the program starts.

From a first look, it might seem unwise to use a rule/lexicon-based sentiment analyser on data from such a dynamic environment such as Twitter, where the data could be anything. Typos, slang and abbreviations are important challenges when analysing data from a social media platform and this type of an analyser would probably not perform that well. However, additional words and emoticons were added from the Twitter Part-of-Speech Annotated Data [12]. In that way, the analyser will be able to cope with the aforesaid challenges since this data will be added to the lexicon and considered when defining the rules. A high-level diagram of how the analyser works can be found below in Figure 4:

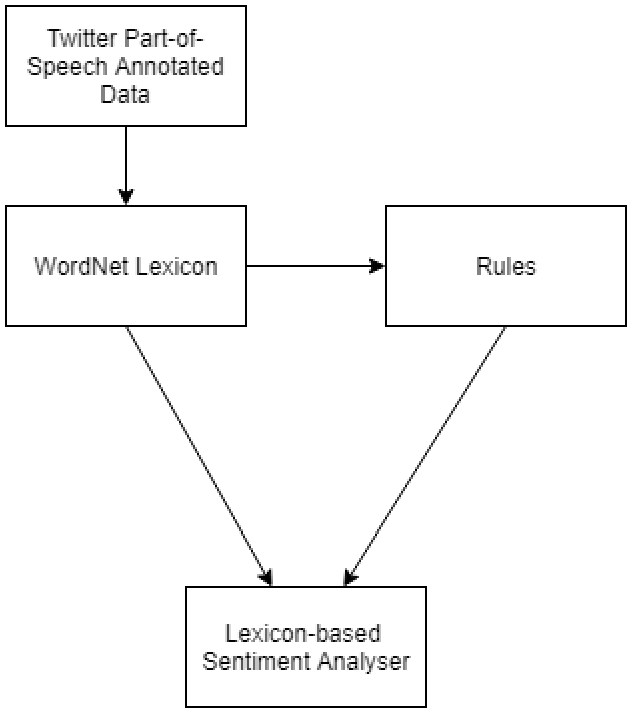

Twitter Part-of-Speech Annotated Data
WordNet Lexicon
Rules
Lexicon-based Sentiment Analyser

*Figure 4. A high-level view of the sentiment analyser*

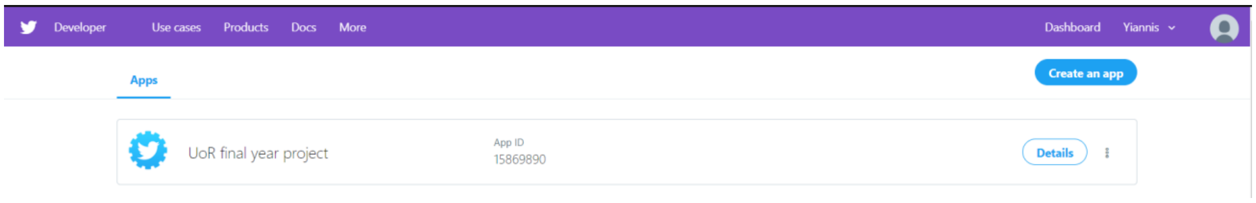
To visualize the results, it was decided to show up to 3 graphs so as not to overwhelm the user. A variety of different types of visualization representations had been considered. From histograms, scatter plots, time series, pie charts, bar charts to heatmaps and word clouds. Time series was chosen to plot the sentiment scores, pie charts and word clouds were chosen to visualize the word frequency in Tweets and to illustrate the difference using stop words makes.

## 5.2 Implementation

In this part, I will thoroughly detail every step of implementing this project and the thought process behind it, from the very start of development to the final version of the solution.

### 5.2.1 Setting up access to the API

First off, an app needs to be created on the Twitter developer platform, as shown below in Figure 5:

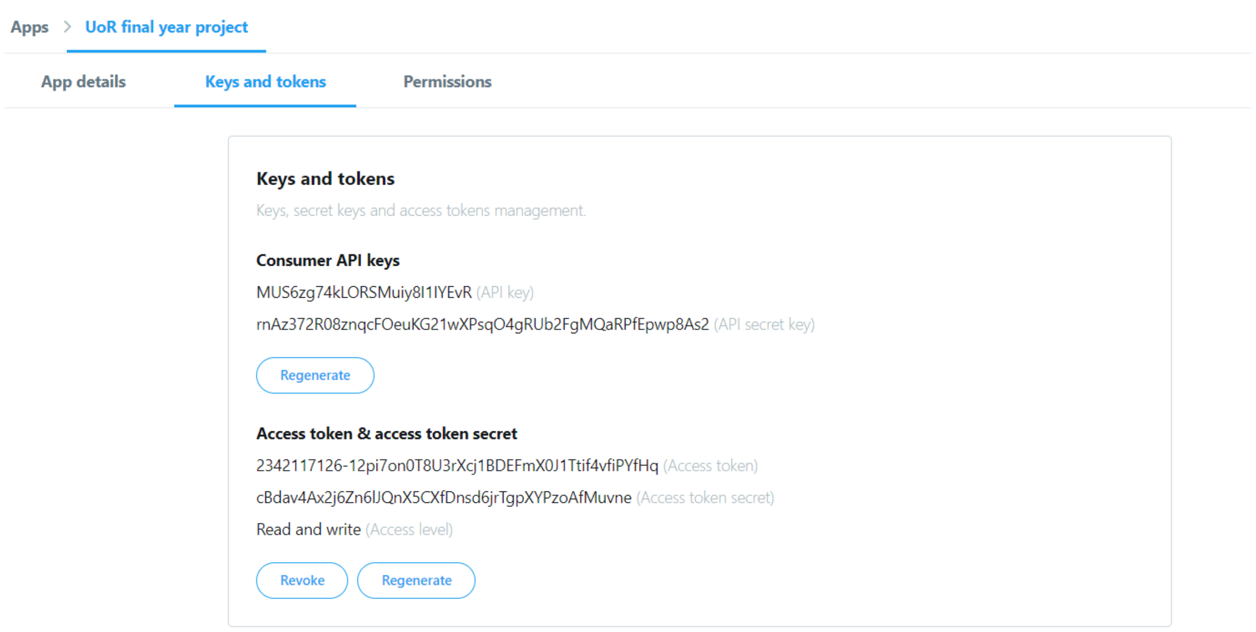


UoR final year project

App ID
15869890

*Figure 5. Twitter Developer page*

This will generate keys and tokens that are required for authentication and authorization, similar to the ones shown below in Figure 6 (pictured keys and tokens are not valid):



**Keys and tokens**

Keys, secret keys and access tokens management.

**Consumer API keys**

MUS6zg74kLORSMuiy8I1IYEvR (API key)

rnAz372R08znqcFOeuKG21wXPsqO4gRUb2FgMQaRPfEpwp8As2 (API secret key)

**Access token & access token secret**

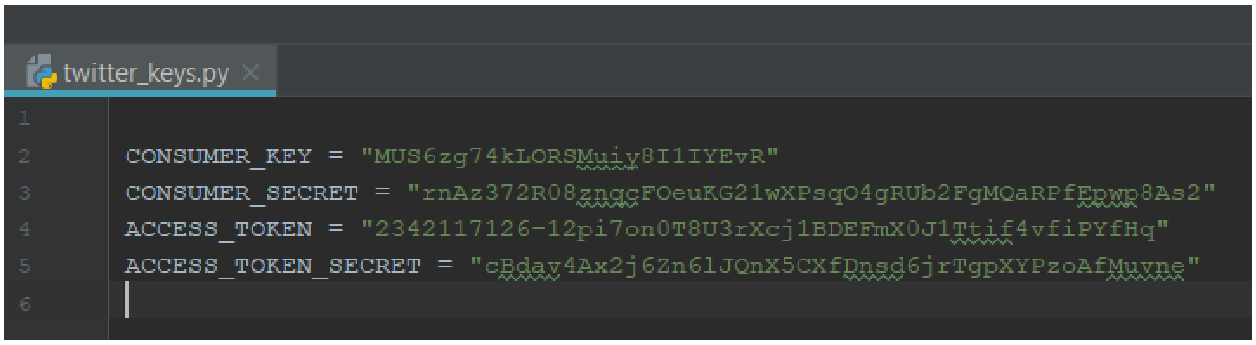
2342117126-12pi7on0T8U3rXcj1BDEFmX0J1Ttif4vfiPYfHq (Access token)

cBdav4Ax2j6Zn6lJQnX5CXfDnsd6jrTgpXYPzoAfMuvne (Access token secret)

Read and write (Access level)

*Figure 6. Twitter API keys and access tokens generated from the developer website*

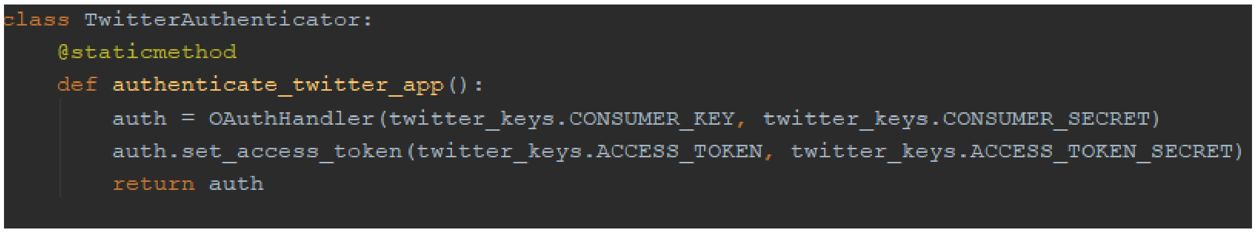
These keys are then taken and will be placed as strings in a separate file, as presented in Figure 7, for the sake of clarity, as such:



```python

CONSUMER_KEY = "MUS6zg74kLORSMuiy8I1IYEvR"
CONSUMER_SECRET = "rnAz372R08znqcFOeuKG21wXPsqO4gRUb2FgMQaRPfEpwp8As2"
ACCESS_TOKEN = "2342117126-12pi7on0T8U3rXcj1BDEFmX0J1Ttif4vfiPYfHq"
ACCESS_TOKEN_SECRET = "cBdav4Ax2j6Zn6lJQnX5CXfDnsd6jrTgpXYPzoAfMuvne"

```

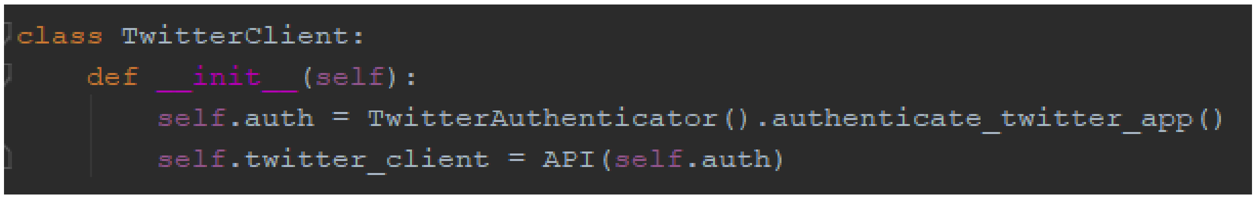
*Figure 7. The keys and tokens are entered in PyCharm*

The Python wrapper for the Twitter API, tweepy, then comes into use. Figure 8 shows the function calls required to authenticate with Twitter:



```python
class TwitterAuthenticator:
    @staticmethod
    def authenticate_twitter_app():
        auth = OAuthHandler(twitter_keys.CONSUMER_KEY, twitter_keys.CONSUMER_SECRET)
        auth.set_access_token(twitter_keys.ACCESS_TOKEN, twitter_keys.ACCESS_TOKEN_SECRET)
        return auth
```

*Figure 8. Authentication process*



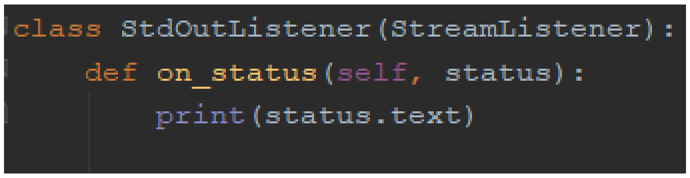
```
class TwitterClient:
    def __init__(self):
        self.auth = TwitterAuthenticator().authenticate_twitter_app()
        self.twitter_client = API(self.auth)
```

*Figure 9. Adding the authenticator object in the init method of another class*

In Figure 8, the OAuthHandler module of the package deals with authentication when it is given the consumer API keys. Based on that authentication, the access tokens are then used to set up a session. The method is then called in the **TwitterClient** class, as shown in Figure 9, where an **auth** object is created so that authentication is established.  Lastly, a **twitter_client** object is created using the wrapper to pass the authentication handler that is going to be used to access the API. It is important that this is placed in the __init__ method of the class, because this is where all the API requests will be made, so authentication has to first take place before making a request to the Twitter API.

There is, however, one exception. The API module only deals with requests to existing Tweets. It does not deal with streaming real-time Tweets. The streaming API will be needed for that. The streaming API works differently since it is not used to pull data from Twitter, but it pushes messages to a persistent session. The steps are the following:

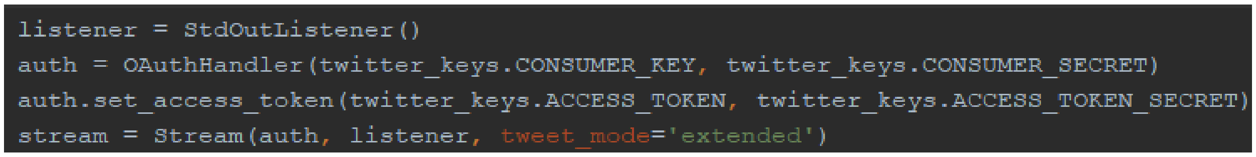
- Create a class that inherits from StreamListener, as seen below in Figure 10:



```
class StdOutListener(StreamListener):
    def on_status(self, status):
        print(status.text)
```

*Figure 10. Override the on_status method to print the status of all streamed Tweets*

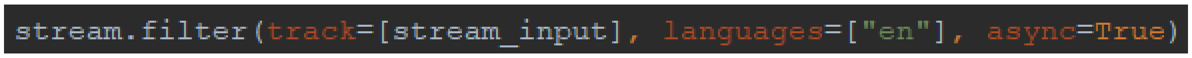
- Using that class create a Stream object and connect to the Twitter API, exemplified below in Figure 11:



```
listener = StdOutListener()
auth = OAuthHandler(twitter_keys.CONSUMER_KEY, twitter_keys.CONSUMER_SECRET)
auth.set_access_token(twitter_keys.ACCESS_TOKEN, twitter_keys.ACCESS_TOKEN_SECRET)
stream = Stream(auth, listener, tweet_mode='extended')
```

*Figure 11. Authenticate and set up a streaming sesion*

- Finally using the Stream object to filter results, as shown in Figure 12:



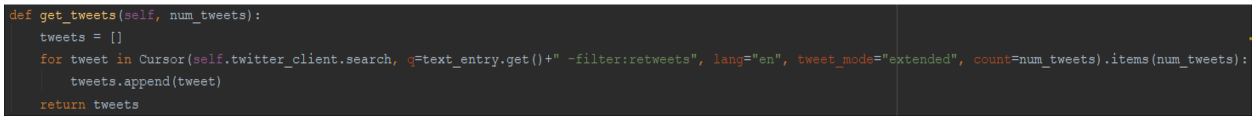
```
stream.filter(track=[stream_input], languages=["en"], async=True)
```

*Figure 12. Filter the Tweets that come in*

In Figure 10, class StdOutListener is created, inheriting from StreamListener and overriding on_status to print a Tweet's text. In Figure 11, an instance of Stream establishes a streaming session and routes messages to the StreamListener instance. The tweet_mode argument allows compatibility with Twitter's extended mode of Tweets (280 characters instead of 140) [13] [14]. Finally, in Figure 12, the Tweets are mainly filtered by the argument track, which is a list of keywords that at least one of those will have to be included in the Tweets. In this case, it is a variable that will be input be the user. The last two arguments ask that only Tweets written in English are extracted and that the stream will run on a new thread so that it will run concurrently with other parts of the program.

### 5.2.2 Data Collection

Now that access to the API has been successful, Tweets can be retrieved upon request. However, the constraints on the free plan have to be taken into account. There's a limit of searching up to 18,000 Tweets, which cannot be more than 7 days old, for every 15 minutes. Therefore, it has been decided that every request to the API asks for 3,000 Tweets.  To search for tweets that matches a search query the API.search method is called. Tweepy provides the Cursor interface to iterate through different types of Twitter objects. After a request, the Twitter API normally returns a JSON with all the attributes but tweepy translates it to a dictionary, making it very easy to access a response's attributes. An example of how it all comes together is shown below in Figure 13:

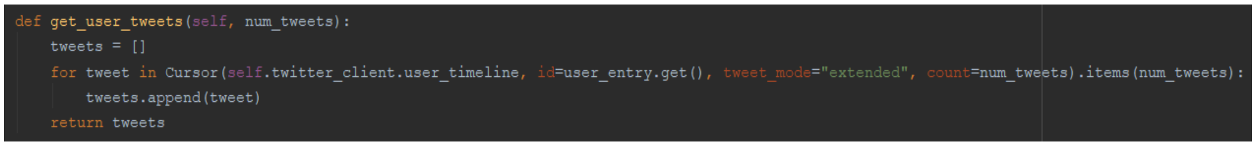


```
def get_tweets(self, num_tweets):
    tweets = []
    for tweet in Cursor(self.twitter_client.search, q=text_entry.get()+" -filter:retweets", lang="en", tweet_mode="extended", count=num_tweets).items(num_tweets):
        tweets.append(tweet)
    return tweets
```

*Figure 13. Method to search existing Tweets given a query*

The twitter_client object, from __init__, is used to search Twitter for a query q that will be entered by the user. "-filter:retweets" excludes Retweets from the results, which would otherwise clutter the list with Retweets. Every time the loop is run, a Tweet is appended in the list (that contains a dictionary of attributes). We need to include the number of Tweets twice, once for the Cursor to know when to stop and once for the API to know how many Tweets we are requesting, or else it will keep going until the rate limit has been reached.

Tweets can also be collected from a particular user, instead of a query in a Tweet, by replacing the API search method with the user_timeline method and the keyword input with the user's Twitter handle, like demonstrated in Figure 14:



```
def get_user_tweets(self, num_tweets):
    tweets = []
    for tweet in Cursor(self.twitter_client.user_timeline, id=user_entry.get(), tweet_mode="extended", count=num_tweets).items(num_tweets):
        tweets.append(tweet)
    return tweets
```

*Figure 14. Method to search existing Tweets from a specific user*

Another way of collecting Tweets is using the streaming part of the API, as briefly described above. Since streaming tweets has no limits on the number of Tweets, it allows for more freedom, which is where there should be more focus from a developer and user standpoint.

One way to use real-time Tweets and the user's participation will be to ask the user to enter a keyword that they want real-time Tweets to include. The approach to achieve this is shown in Figures 11 & 12. A live graph will then show the sentiment score as more Tweets are being collected as well as other visualization options. The way of doing this will be explained later on in the appropriate subsections.

After discussion with the supervisor of this project, Dr Frederic Stahl, it was recommended that a case study is recorded for an event which would most likely cause change in public opinion. In this way, this would both confirm that the sentiment analysis part of the program is working well and provide results on a topic based on a big number of Tweets. In anticipation of EU leaders voting on UK's Brexit deal on the 25th November 2018, live Tweets were collected between the 23rd – 27th November 2018 each day for a couple hours, that contained either "brexit" or "#brexit". Below in Figures 15 & 16, there are some visualizations of the results, that, as mentioned above, will be explained at later subsections of the Implementation section.

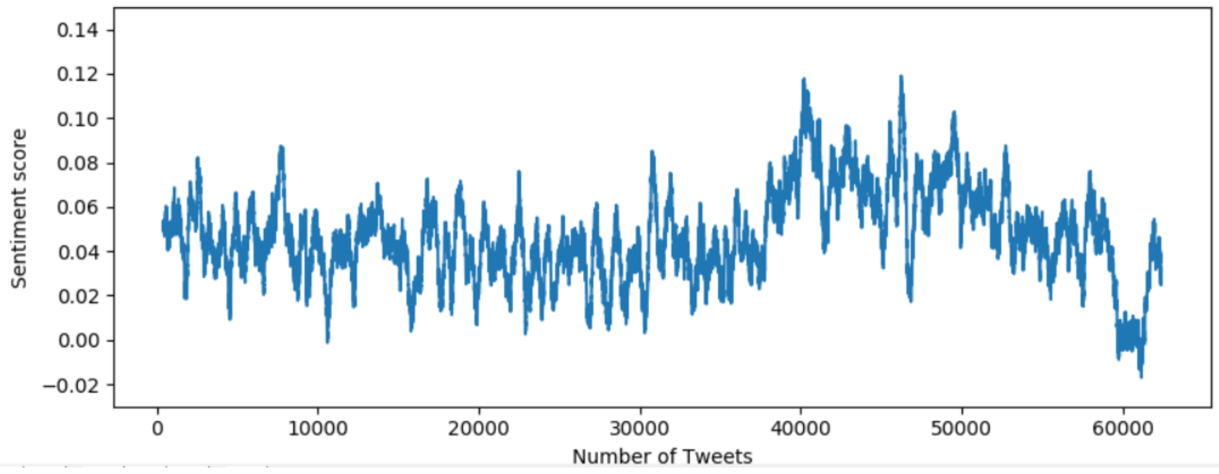
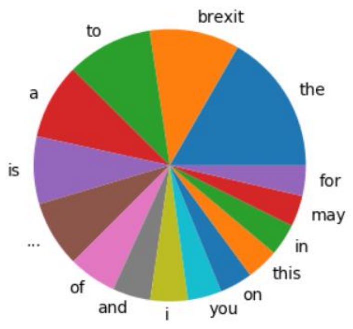


*Figure 15. Time series of the sentiment scores*



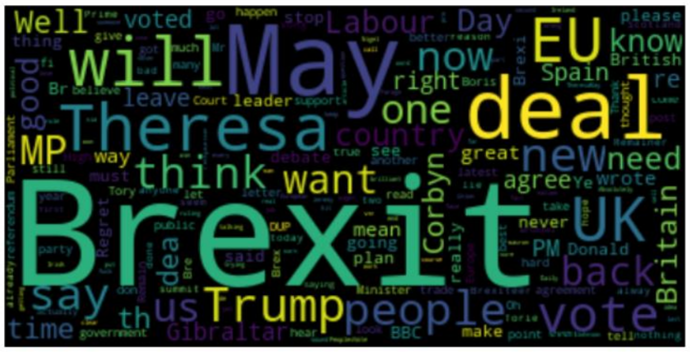
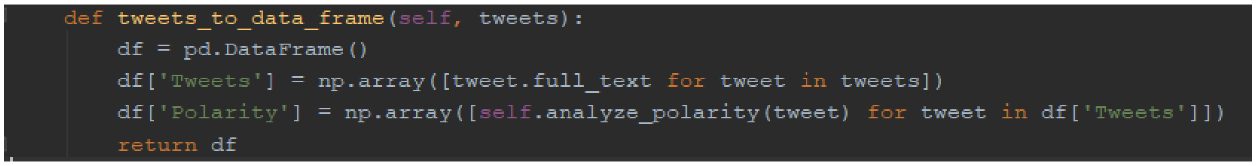


*Figure 16. Word frequency visualization*

### 5.2.3 Pre-processing & Data Manipulation

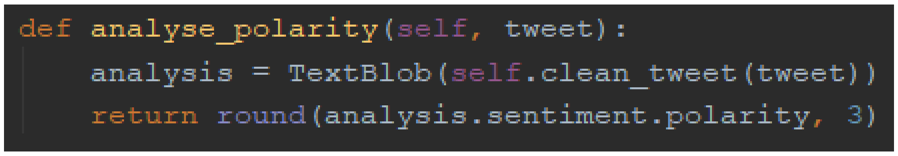
Data manipulation and pre-processing take place just after the Tweets are collected. The only attribute that is of importance in this project, from the dictionary returned by the API, is the status text attribute.



```python
    def tweets_to_data_frame(self, tweets):
        df = pd.DataFrame()
        df['Tweets'] = np.array([tweet.full_text for tweet in tweets])
        df['Polarity'] = np.array([self.analyze_polarity(tweet) for tweet in df['Tweets']])
        return df
```

*Figure 17. Process the data in a data frame*

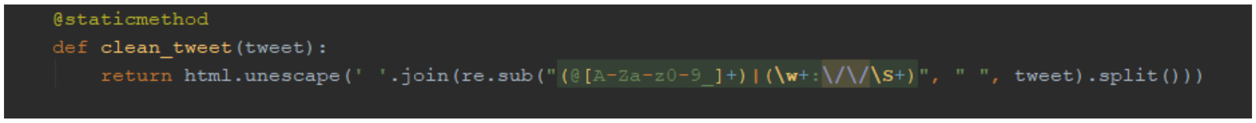
In Figure 17, two Series, or columns, are created: Tweets and Polarity. Tweets column is created to only take the text, or "full_text" for the extended Tweet mode, attribute and using NumPy store it in a data frame. The Polarity series is created to calculate the sentiment polarity by using the "Tweets" and passing it to the analyse_polarity method.



```python
def analyse_polarity(self, tweet):
    analysis = TextBlob(self.clean_tweet(tweet))
    return round(analysis.sentiment.polarity, 3)
```

*Figure 18. Analyse a cleaned Tweet's sentiment polarity*

In Figure 18, before passing the Tweet to the TextBlob analyser, it passes it to the clean_tweet method which will strip the Tweet of any problematic or redundant characters. The sentiment analysis part will be discussed in the following subsection below.



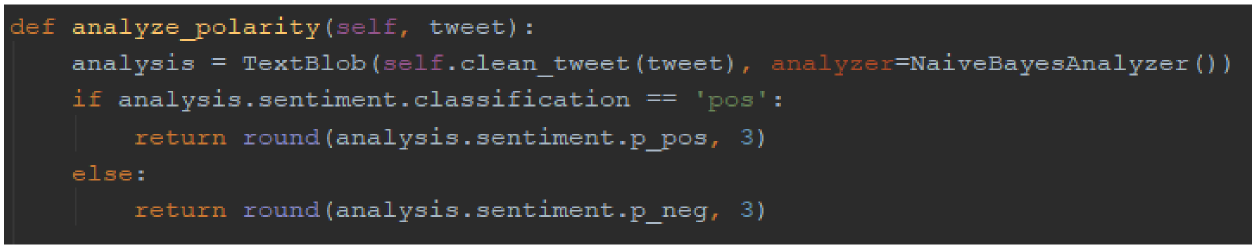
```python
    @staticmethod
    def clean_tweet(tweet):
        return html.unescape(' '.join(re.sub("(@[A-Za-z0-9_]+)|(\w+:\/\/\S+)", " ", tweet).split()))
```

*Figure 19. Clean a Tweet from redundant characters and unescape escaped characters*

Now, as stated above, the next step is to clean the tweet of unnecessary content using regex. The code snipper in Figure 19 shows how this is done. The regex is asked to substitute any hyperlinks (2nd part of the regex) and any Twitter mentions with whitespace. Join and split are used to be able to check every word separately. Since we are basically making HTTP requests when retrieving Tweets, some special characters are encoded. For example, "&" is encoded to "&amp;" and a double quote is encoded to "&quot;". The html module helps with manipulating HTML. Unescape(str) is used to convert these references to Unicode characters. The @staticmethod decorator was used to make the method static. The text is now ready for sentiment analysis.

### 5.2.4 TextBlob

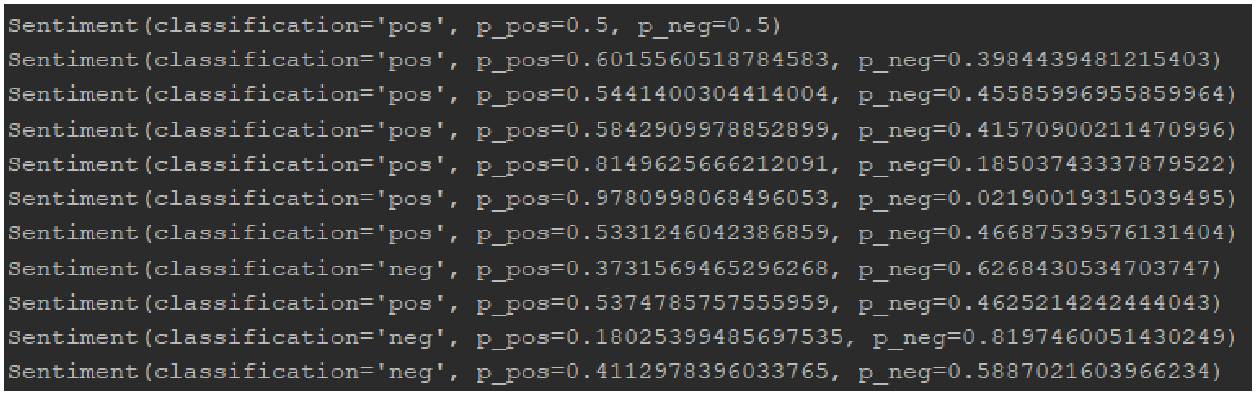
In this subsection the code in Figure 18 will be explained in further detail, as well as the specifics of how TextBlob works. After the Tweet is cleaned, it is passed to the TextBlob object and assigned to the analysis variable. Now a whole range of features are available, such as spelling correction, translation, parsing, word count, tokenization, but most importantly sentiment polarity (and subjectivity). As mentioned before, a lexicon-based approach, using the WordNet dictionary, has been followed. The Tweet is then rounded to 3 decimal places and returned to the data frame into the "Polarity" series.



```
def analyze_polarity(self, tweet):
    analysis = TextBlob(self.clean_tweet(tweet), analyzer=NaiveBayesAnalyzer())
    if analysis.sentiment.classification == 'pos':
        return round(analysis.sentiment.p_pos, 3)
    else:
        return round(analysis.sentiment.p_neg, 3)
```

*Figure 20. Naïve Bayes analyser*

Above, in Figure 20, is the implementation of the Naïve Bayes classification, which has been rejected. The changes in the code are minimal however the difference in performance is massive which was one of the major reasons to decide not to choose this approach. An example of the first few results is shown below in Figure 21. The keyword "love" was given to search in Tweets. After manually going through the Tweets that are classified in the Figure below, all of them were positive, but some were mistakenly classified. The algorithm's accuracy does improve; however, the analyser's speed stays slow.



```
Sentiment(classification='pos', p_pos=0.5, p_neg=0.5)
Sentiment(classification='pos', p_pos=0.6015560518784583, p_neg=0.3984439481215403)
Sentiment(classification='pos', p_pos=0.5441400304414004, p_neg=0.4558599695585964)
Sentiment(classification='pos', p_pos=0.5842909978852899, p_neg=0.41570900211470996)
Sentiment(classification='pos', p_pos=0.8149625666212091, p_neg=0.18503743337879522)
Sentiment(classification='pos', p_pos=0.9780998068496053, p_neg=0.021900193150394495)
Sentiment(classification='pos', p_pos=0.5331246042386859, p_neg=0.46687539576131404)
Sentiment(classification='neg', p_pos=0.3731569465296268, p_neg=0.6268430534703747)
Sentiment(classification='pos', p_pos=0.5374785757555959, p_neg=0.4625214242444043)
Sentiment(classification='neg', p_pos=0.18025399485697535, p_neg=0.8197460051430249)
Sentiment(classification='neg', p_pos=0.4112978396033765, p_neg=0.5887021603966234)
```

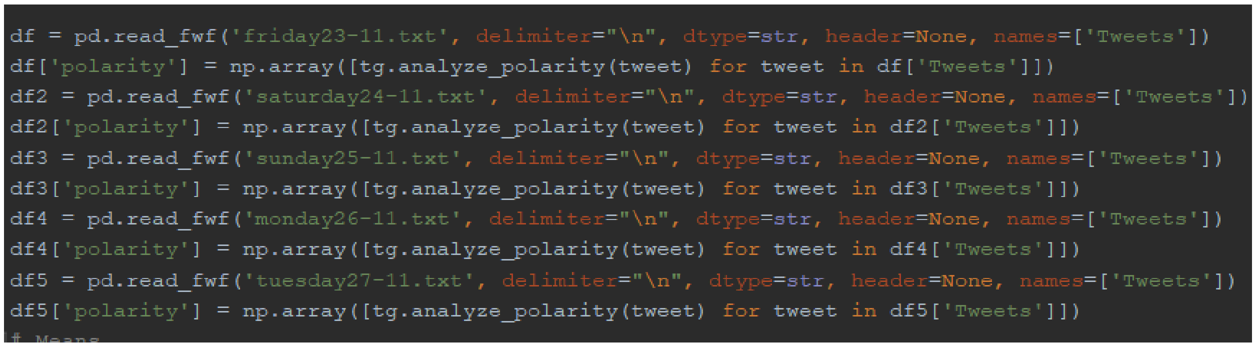
*Figure 21. Output of the analysis of each Tweet passed to the analyser*

### 5.2.5 Further Data Manipulation & Data visualization

A number of different visualization techniques were considered and tested. Matplotlib provides very useful resources for creating visualizations.

Coming back to the case study discussed earlier, in Figure 15 a time series of the moving averages of the sentiment scores is shown, using the rolling method from pandas. In Figure 16, a pie chart and a word cloud are drawn, based on the occurrence of each word in the Tweets. However, there are some differences between the two. The main difference between the bar chart and word cloud is that the word cloud module uses stop words. Therefore, connecting words, which are the most common words in sentences, are ignored. Another difference is that for the pie chart, only the 15 most occurring words were chosen to be kept, to focus on the most important words and remove noise. Other visualization graphs were considered, such as a horizontal bar plot and a histogram but they were rejected over the pie chart and word cloud, with the main reason being the clarity that the latter offer.

To analyse the case study data, each day's Tweet files are loaded in data frames. The polarity of each Tweet is then calculated by passing each Tweet through the analyse_polarity method, that first cleans the Tweet, from earlier. A code snippet of this is shown in Figure 22 below:



```
df = pd.read_fwf('friday23-11.txt', delimiter="\n", dtype=str, header=None, names=['Tweets'])
df['polarity'] = np.array([tg.analyze_polarity(tweet) for tweet in df['Tweets']])
df2 = pd.read_fwf('saturday24-11.txt', delimiter="\n", dtype=str, header=None, names=['Tweets'])
df2['polarity'] = np.array([tg.analyze_polarity(tweet) for tweet in df2['Tweets']])
df3 = pd.read_fwf('sunday25-11.txt', delimiter="\n", dtype=str, header=None, names=['Tweets'])
df3['polarity'] = np.array([tg.analyze_polarity(tweet) for tweet in df3['Tweets']])
df4 = pd.read_fwf('monday26-11.txt', delimiter="\n", dtype=str, header=None, names=['Tweets'])
df4['polarity'] = np.array([tg.analyze_polarity(tweet) for tweet in df4['Tweets']])
df5 = pd.read_fwf('tuesday27-11.txt', delimiter="\n", dtype=str, header=None, names=['Tweets'])
df5['polarity'] = np.array([tg.analyze_polarity(tweet) for tweet in df5['Tweets']])
```

*Figure 22. Reading in the collected Tweets and creating two Series for each file*

The polarity columns are then concatenated using pandas. A moving average can be calculated using the rolling method from pandas again, allowing to specify the size of the moving window. The resulting column is then used by matplotlib to plot a time series, as shown in Figure 15. The figure shows that there's a small increase in the average sentiment score halfway through the graph and that was probably due to the fact that the deal was accepted by the EU leaders, but soon after the scores went back to normal.

The visualizations in Figure 16, do not take the sentiment score into account, they only take the occurrence of each word in all the Tweets.

To plot the pie chart, two lists are needed: one for the occurrences of each word and another for the word labels. Although the pie chart can also be drawn without labels, it wouldn't make sense to have it drawn without labels. To draw the chart, the 'Tweets' Series are concatenated. After that, the to_string() method, from pandas, is used to convert the concatenated Series into one long string. That string is then cleaned using the clean_tweet method. The string's characters need to be converted to lower-case so that the case will not be a factor when counting the unique occurrences of each word. This is achieved by using the lower() method. The split() method needs to be used to split the string into words. By default, the separator value is a whitespace. Every word is now a list item and next up is counting how many times each item is found in the list. The Counter data type can do this relatively fast. The most_common(n) method returns a list of the n most common elements and their pairs, sorting it from the most common to the least. By storing it as a dictionary, instead of a list, it will be easier to extract the occurrence of each word since it will be stored in a "key: value" format. It helps that dictionaries are insertion ordered in Python 3.6. Now the next step is to store the keys and the values in two separate lists. To do that, two empty lists are declared that elements of the dictionary will be appended to on every iteration of the for loop. This could also be done using a list comprehension technique. Finally, the pie chart can be drawn.

It will be easier to draw the word cloud picture, since some data manipulation has already taken place. In addition to that, the word cloud module is smart and will also manipulate the data. It just needs to be passed one long string of words, which has been created above using to_string(), to generate the word cloud.

Here's how everything comes together and the code snipper to follow it in Figures 23 & 24:

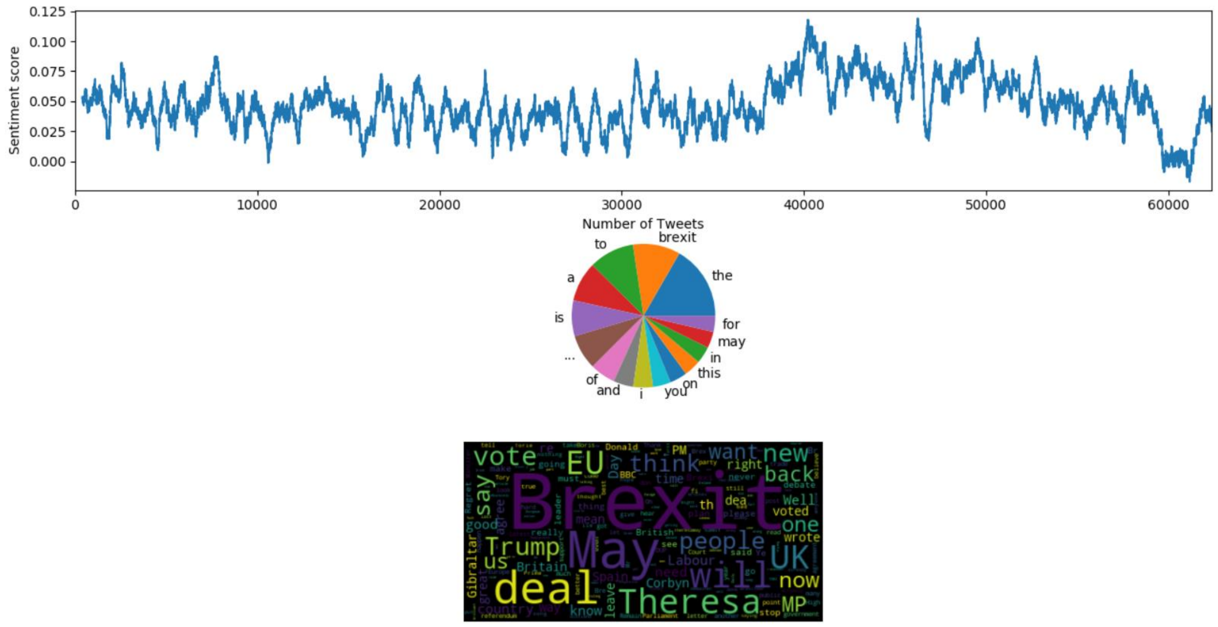
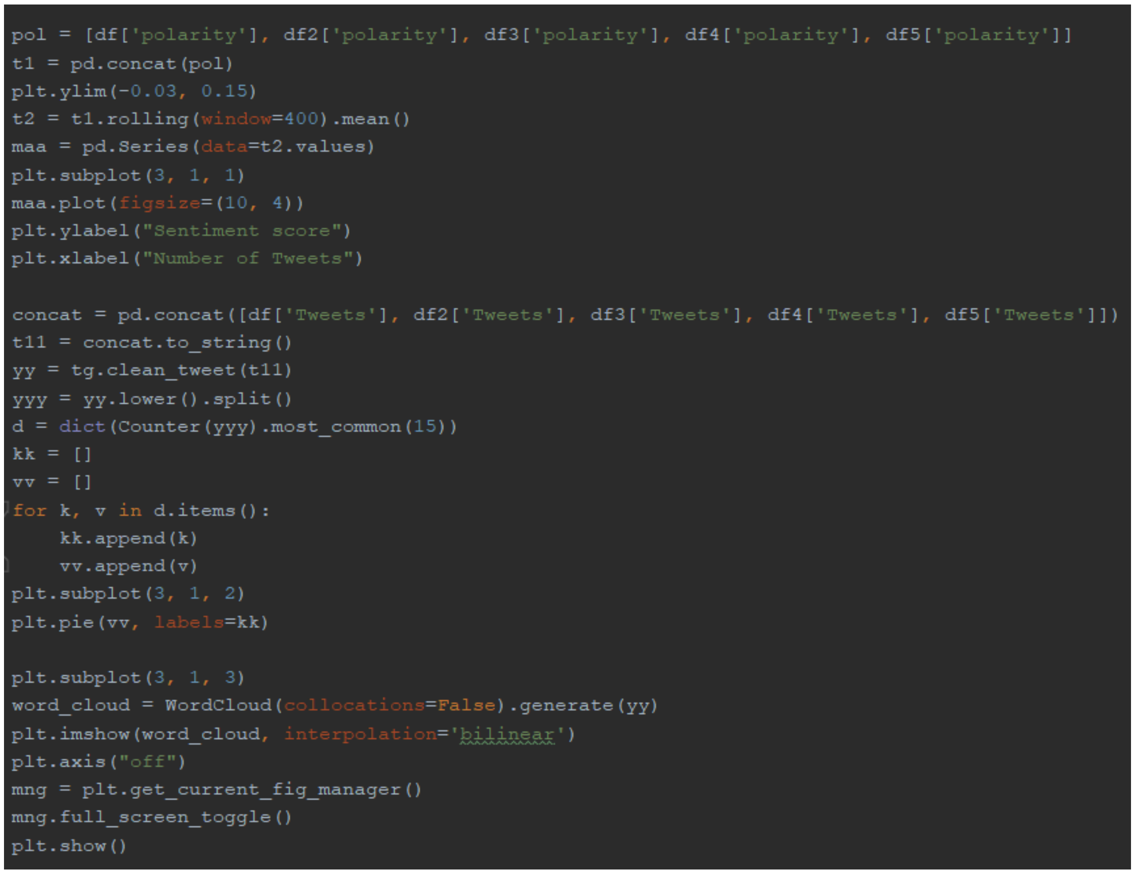


*Figure 23*

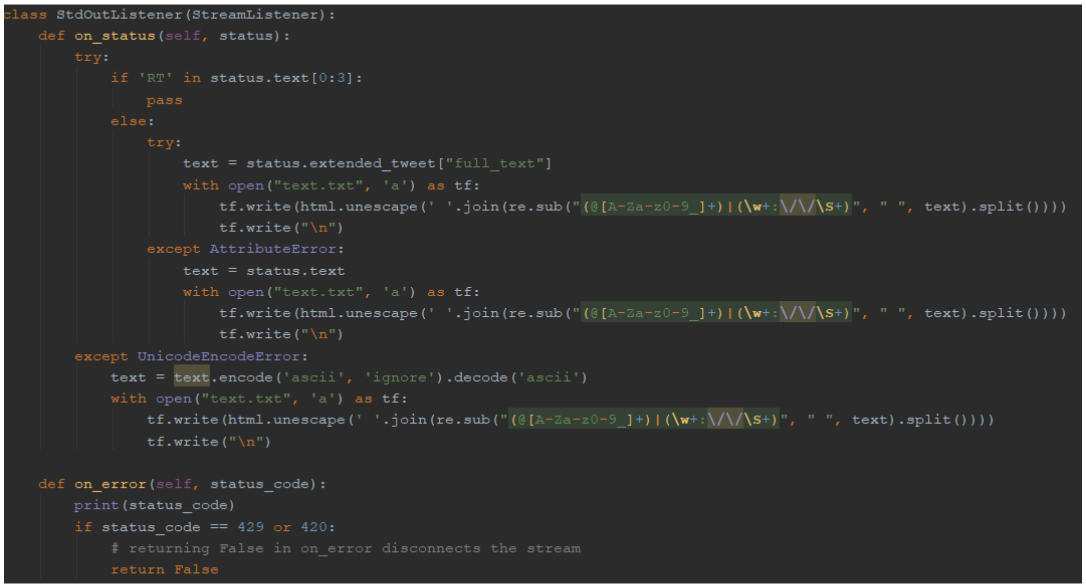


```
pol = [df['polarity'], df2['polarity'], df3['polarity'], df4['polarity'], df5['polarity']]
t1 = pd.concat(pol)
plt.ylim(-0.03, 0.15)
t2 = t1.rolling(window=400).mean()
maa = pd.Series(data=t2.values)
plt.subplot(3, 1, 1)
maa.plot(figsize=(10, 4))
plt.ylabel("Sentiment score")
plt.xlabel("Number of Tweets")

concat = pd.concat([df['Tweets'], df2['Tweets'], df3['Tweets'], df4['Tweets'], df5['Tweets']])
t11 = concat.to_string()
yy = tg.clean_tweet(t11)
yyy = yy.lower().split()
d = dict(Counter(yyy).most_common(15))
kk = []
vv = []
for k, v in d.items():
    kk.append(k)
    vv.append(v)
plt.subplot(3, 1, 2)
plt.pie(vv, labels=kk)

plt.subplot(3, 1, 3)
word_cloud = WordCloud(collocations=False).generate(yy)
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
mng = plt.get_current_fig_manager()
mng.full_screen_toggle()
plt.show()
```

*Figure 24. Visual analysis of the collected Tweets and a code snippet*

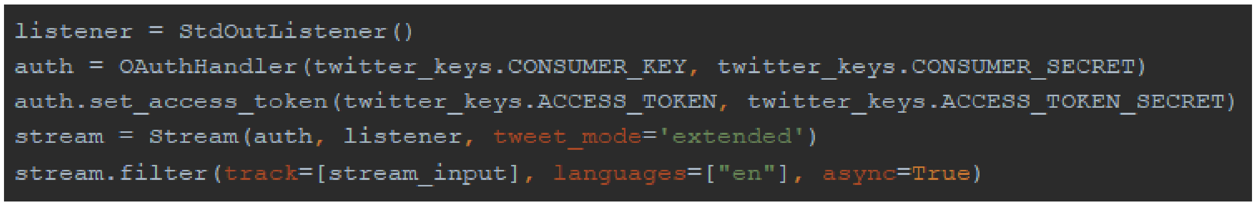
So far only static graphs have been explained. To be able to visually display real-time Tweet analytics, some dynamic visualization is needed. Matplotlib provides an animation function which updates a graph as data is changed. At the same time that Tweets are written in a file, the function will plot their sentiment scores. The TkAgg backend will also be used (Anti-grain geometry rendering to a Tk canvas) since the interface is built using TkInter, which will be thoroughly explained in the next subsection.

First off, it is important that we break connection upon being rate limited, even though that rarely happens when streaming Tweets unless multiple sessions are running, to avoid the exponential back-off cooldown. When not rate limited, real-time Tweets will be saved in a text file after the Tweet has been cleaned and if it's not a Retweet. Two exceptions must be checked: when the Tweets are written in extended or normal mode and when emojis are in the text which would create problems with storing them in a text file. The class is implemented and then called as shown below in Figures 25 & 26:



```
class StdOutListener(StreamListener):
    def on_status(self, status):
        try:
            if 'RT' in status.text[0:3]:
                pass
            else:
                try:
                    text = status.extended_tweet["full_text"]
                    with open("text.txt", 'a') as tf:
                        tf.write(html.unescape(' '.join(re.sub("(@[A-Za-z0-9_]+)|(\w+:\/\/\S+)", " ", text).split())))
                        tf.write("\n")
                except AttributeError:
                    text = status.text
                    with open("text.txt", 'a') as tf:
                        tf.write(html.unescape(' '.join(re.sub("(@[A-Za-z0-9_]+)|(\w+:\/\/\S+)", " ", text).split())))
                        tf.write("\n")
        except UnicodeEncodeError:
            text = text.encode('ascii', 'ignore').decode('ascii')
            with open("text.txt", 'a') as tf:
                tf.write(html.unescape(' '.join(re.sub("(@[A-Za-z0-9_]+)|(\w+:\/\/\S+)", " ", text).split())))
                tf.write("\n")

    def on_error(self, status_code):
        print(status_code)
        if status_code == 429 or 420:
            # returning False in on_error disconnects the stream
            return False
```

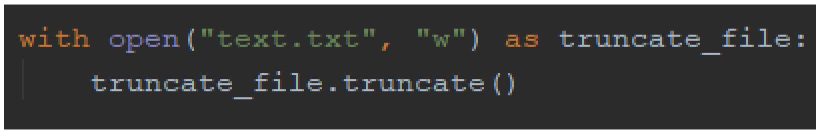
Figure 25. Class for capturing real-time Tweets



```
listener = StdOutListener()
auth = OAuthHandler(twitter_keys.CONSUMER_KEY, twitter_keys.CONSUMER_SECRET)
auth.set_access_token(twitter_keys.ACCESS_TOKEN, twitter_keys.ACCESS_TOKEN_SECRET)
stream = Stream(auth, listener, tweet_mode='extended')
stream.filter(track=[stream_input], languages=["en"], async=True)
```

Figure 26. Setting up a stream session

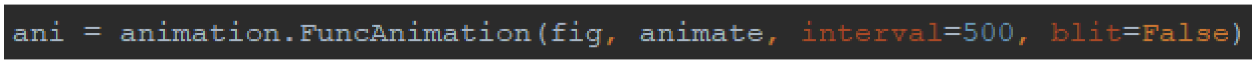
Before the animation part the text file needs to be empty, since every time the function is called there will be different user input and new Tweets generated. The truncate() method truncates the file's size. Figure 27 below shows how this is done:



```
with open("text.txt", "w") as truncate_file:
    truncate_file.truncate()
```

Figure 27. Truncating the file

Now the animation can begin, using the line of code shown below in Figure 28:



```
ani = animation.FuncAnimation(fig, animate, interval=500, blit=False)
```

*Figure 28. Matplotlib animation call*

The first argument refers to the object that is used for the animation, the second argument refers to the function to call each frame. Next, are optional arguments, such as the delay between each frame, set to 500 ms, and whether blitting (optimizes drawing) will be used.

Now, the animate function needs to be defined. The text.txt file is opened in r+ mode, which is for reading and writing but the stream is positioned at the beginning of the file. The file is read and is split by the newline character as new Tweets are stored in new lines. Another file, score.txt needs to be created. It will hold the value of the sentiment scores, that will be plotted in the y axis, and the number of Tweets, that will be plotted in the x axis. It iteratively writes the number of Tweets and their sentiment on a new line, for example:

1,0.55

2,-0.25

3,0.333

After the loop has ended, each line of the file is now read. Two lists are created: one for the x values and another for the y values. The strings are casted to floating-point numbers, split with the separator being comma, and added to the corresponding list. They are then plotted. The y values are later stored in a data frame so that the method responsible for creating a moving average is used. Appropriate titles, labels are added. The final result and code snippet are shown below in Figures 29, 20 and 31:

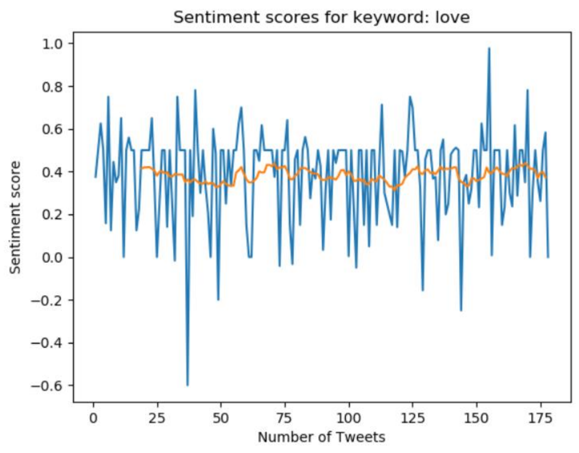


*Figure 29. Live graph*

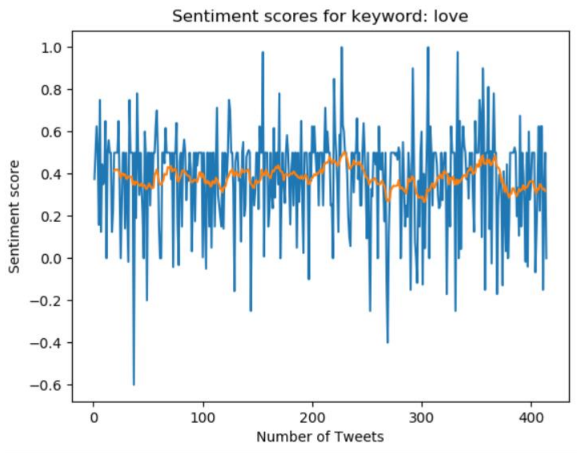
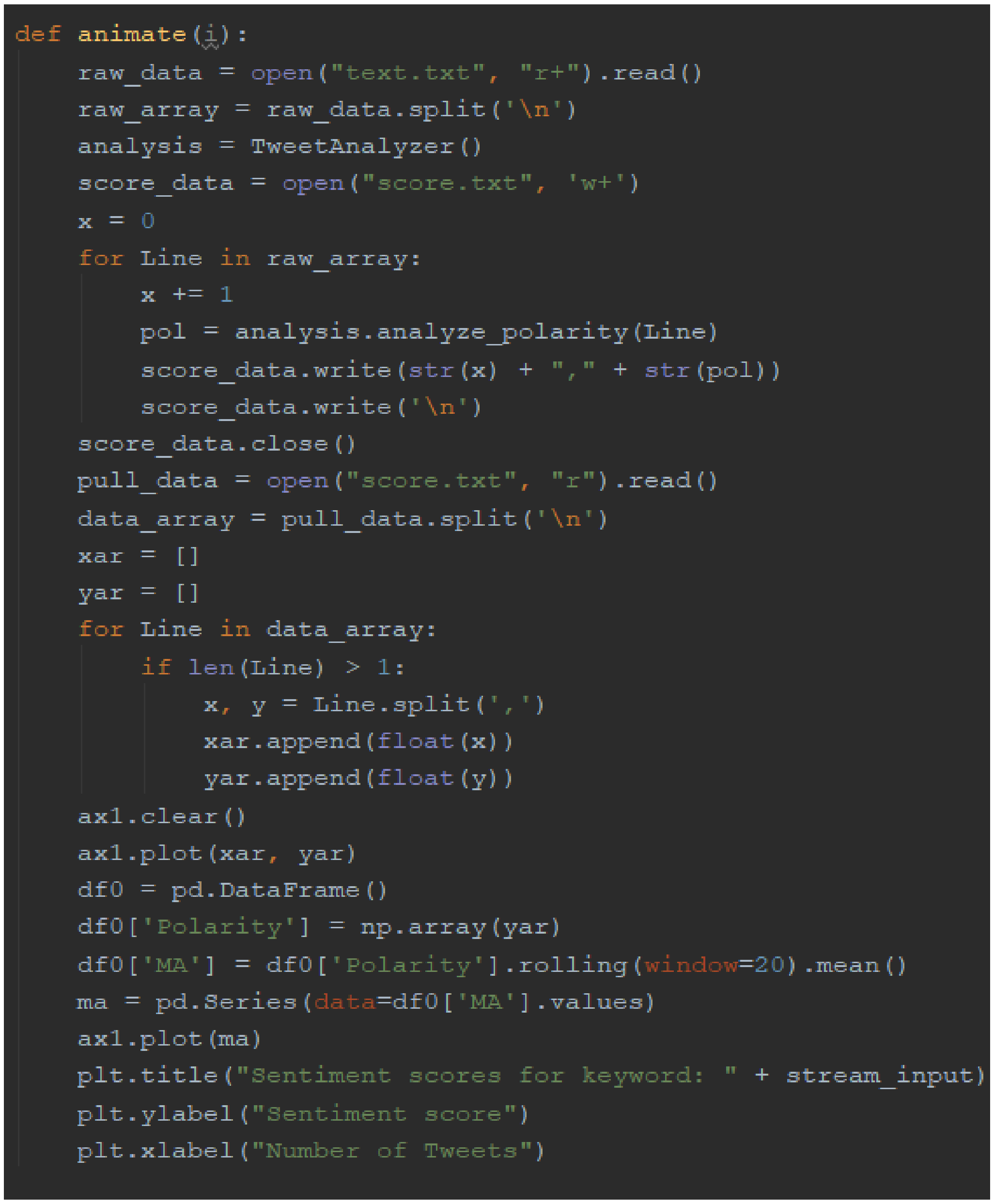


*Figure 30. Same live graph with more Tweets*



```python
def animate(i):
    raw_data = open("text.txt", "r+").read()
    raw_array = raw_data.split('\n')
    analysis = TweetAnalyzer()
    score_data = open("score.txt", 'w+')
    x = 0
    for Line in raw_array:
        x += 1
        pol = analysis.analyze_polarity(Line)
        score_data.write(str(x) + "," + str(pol))
        score_data.write('\n')
    score_data.close()
    pull_data = open("score.txt", "r").read()
    data_array = pull_data.split('\n')
    xar = []
    yar = []
    for Line in data_array:
        if len(Line) > 1:
            x, y = Line.split(',')
            xar.append(float(x))
            yar.append(float(y))
    ax1.clear()
    ax1.plot(xar, yar)
    df0 = pd.DataFrame()
    df0['Polarity'] = np.array(yar)
    df0['MA'] = df0['Polarity'].rolling(window=20).mean()
    ma = pd.Series(data=df0['MA'].values)
    ax1.plot(ma)
    plt.title("Sentiment scores for keyword: " + stream_input)
    plt.ylabel("Sentiment score")
    plt.xlabel("Number of Tweets")
```

*Figure 31. Code snippet for the live graph*

### 5.2.6 Interface

The GUI must be the first thing the user sees when the program is run. It must be as simple and as intuitive as possible so that the user has the best possible experience. Now that all of the backend code is finished, the user interface can be developed.

As already mentioned, the GUI is written in TkInter which is a Python binding to the Tk GUI toolkit. At a very basic level, there needs to be at least a main window, that instantiates the class and creates a top-level widget of Tk which usually is the main window, and a main event loop that will take action against each event. Now more widgets can be added in the main window. A Frame widget is added to be used as a container widget to organize other widgets.

The three main types of widgets that were used are:

- Buttons, that display a button which triggers a command, usually a function.
- Labels, that display a single-line caption for other widgets. It can also contain images.
- Entries, that display a single-line text field for user input.

Another important type of widget is the PhotoImage class. Its purpose is to display coloured images. The Twitter logo will be the picture that will be read in. It will be placed inside a Label widget and then every widget created henceforth will be referencing that Label for the parent window instead of the Frame. This is done so that the widget with the Twitter logo becomes the background picture and the widgets will be placed inside its dimensions. It is important that the user is not able to resize the window, which could introduce issues the GUI, so the resizable method has been used and set to (False, False) making sure the window always remains at the pre-set dimensions. The grid() method has been used to manage the geometry of the widgets, which organizes widgets in a table-like structure. For instance, here’s a code snippet, for a Button widget:

```
ttk.Button(background, text="Latest", command=gui.analysis).grid(row=3)
```

After the background label has been added, the widgets that are necessary to the user are next. The user will be able to analyse existing Tweets in 3 ways:

- An entered string, that could be a keyword or hashtag, that retrieved Tweets must include
- Two entered strings, that could be keywords or/and hashtags, that retrieved Tweets must include
- An entered string corresponding to a Twitter user profile, that their latest Tweets will be retrieved from

The options for the first two ways are:

- Retrieve the latest Tweets
- Retrieve Tweets from one day ago
- Retrieve Tweets from three days ago

By default, the Twitter API returns the latest tweets. However, this can be changed when using the datetime built-in package and specifying it in the “until” parameter of the API wrapper method.

The user will be able to analyse real-time Tweets only by entering one keyword or hashtag. Another TkInter window will then open for the dynamic graph and more options that will be discussed below.

Other options that do not require access to the Twitter API are:

- Access the case study results, that shows a combined picture of Figures 15 & 16
- Exiting the program

Themed Tk (TTK) has been used, that gives the widgets an improved look and feel, for buttons, labels and entries.

Figures 32 & 33 show how the main window of the GUI looks, as well as a part of the code:

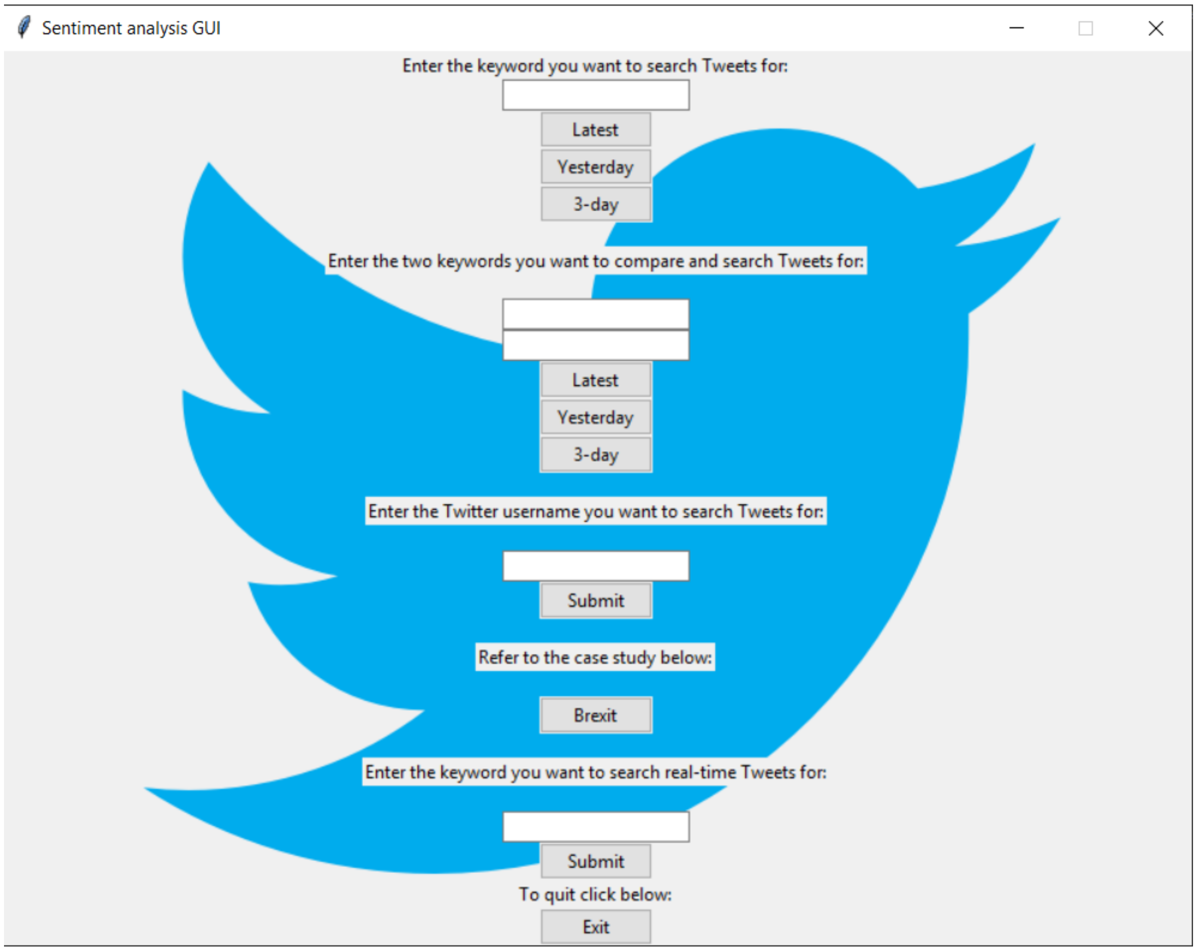
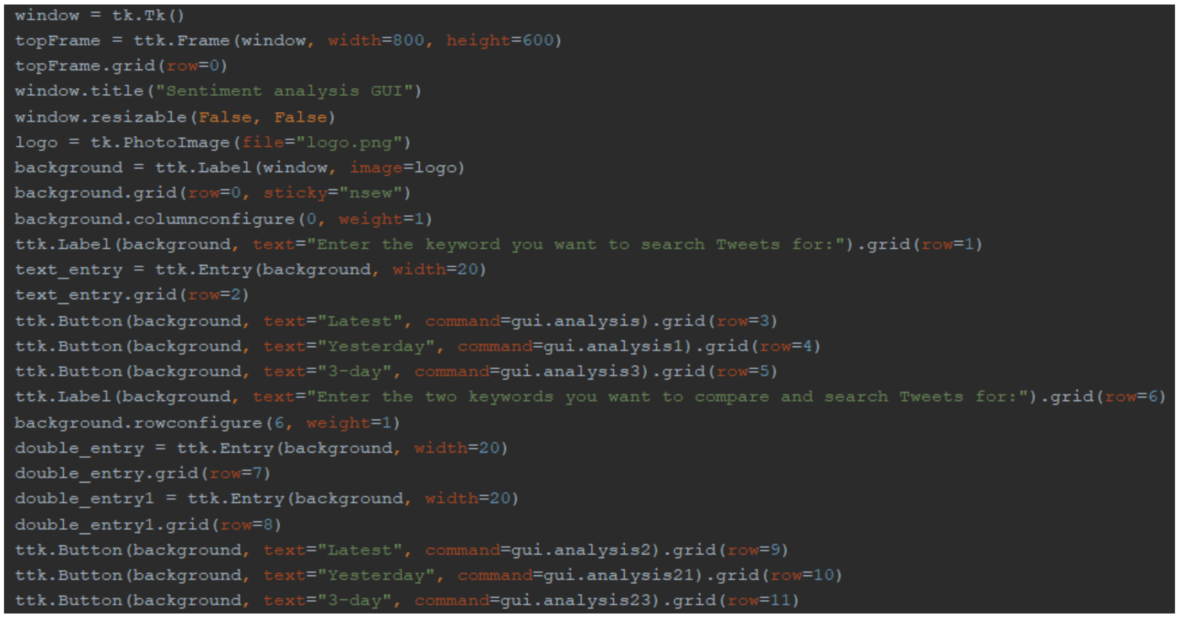

Sentiment analysis GUI
Enter the keyword you want to search Tweets for:
Latest
Yesterday
3-day
Enter the two keywords you want to compare and search Tweets for:
Latest
Yesterday
3-day
Enter the Twitter username you want to search Tweets for:
Submit
Refer to the case study below:
Brexit
Enter the keyword you want to search real-time Tweets for:
Submit
To quit click below:
Exit

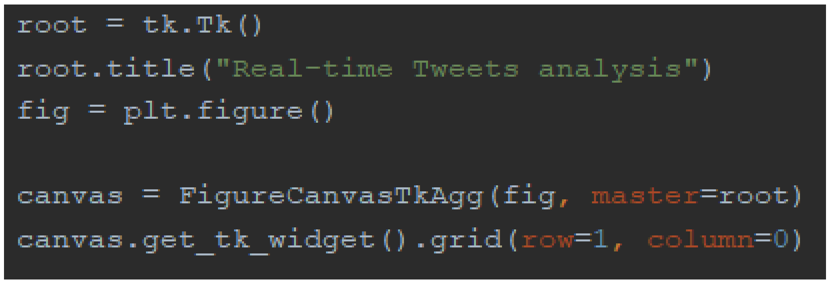
*Figure 32. The GUI*



```
window = tk.Tk()
topFrame = ttk.Frame(window, width=800, height=600)
topFrame.grid(row=0)
window.title("Sentiment analysis GUI")
window.resizable(False, False)
logo = tk.PhotoImage(file="logo.png")
background = ttk.Label(window, image=logo)
background.grid(row=0, sticky="nsew")
background.columnconfigure(0, weight=1)
ttk.Label(background, text="Enter the keyword you want to search Tweets for:").grid(row=1)
text_entry = ttk.Entry(background, width=20)
text_entry.grid(row=2)
ttk.Button(background, text="Latest", command=gui.analysis).grid(row=3)
ttk.Button(background, text="Yesterday", command=gui.analysis1).grid(row=4)
ttk.Button(background, text="3-day", command=gui.analysis3).grid(row=5)
ttk.Label(background, text="Enter the two keywords you want to compare and search Tweets for:").grid(row=6)
background.rowconfigure(6, weight=1)
double_entry = ttk.Entry(background, width=20)
double_entry.grid(row=7)
double_entry1 = ttk.Entry(background, width=20)
double_entry1.grid(row=8)
ttk.Button(background, text="Latest", command=gui.analysis2).grid(row=9)
ttk.Button(background, text="Yesterday", command=gui.analysis21).grid(row=10)
ttk.Button(background, text="3-day", command=gui.analysis23).grid(row=11)
```

*Figure 33. Code snippet to show how the widgets are put together*

When the user opts to use the real-time function, a new Tk window will be used. A canvas will be drawn using the FigureCanvasTkAgg module that, as mentioned, provides a backend for rendering on a Tk Canvas. The canvas is bound to the figure that is also used for the live graph and the master window is the new TkInter window that was just created. A code snippet that demonstrates the aforementioned steps can be found in Figure 34 below:



```
root = tk.Tk()
root.title("Real-time Tweets analysis")
fig = plt.figure()

canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().grid(row=1, column=0)
```

*Figure 34. Creating a canvas backend*

Figures 26, 27, 28 and 31 are actually in the function that creates this canvas and the new TkInter window. Therefore, the live graph will appear as soon as the button is pressed to call the function, since the line of code in Figure 28 is executed. A number of buttons has been a under the graph that provide different functionalities. A "Clear" button to clear all Tweets saved thus clearing the graph is the first button. This is done by truncating the file size of the text.txt file, as shown above in Figure 27. A "Stop" button, that stops the streaming of real-time Tweets immediately has also been added. Next are buttons for a word cloud and a pie chart and lastly an "Exit" button that is a combination of the "Clear" and "Stop" buttons as well as destroy the TkInter window and safely close the figure and canvas. An example of how everything comes together is shown below in Figure 35:

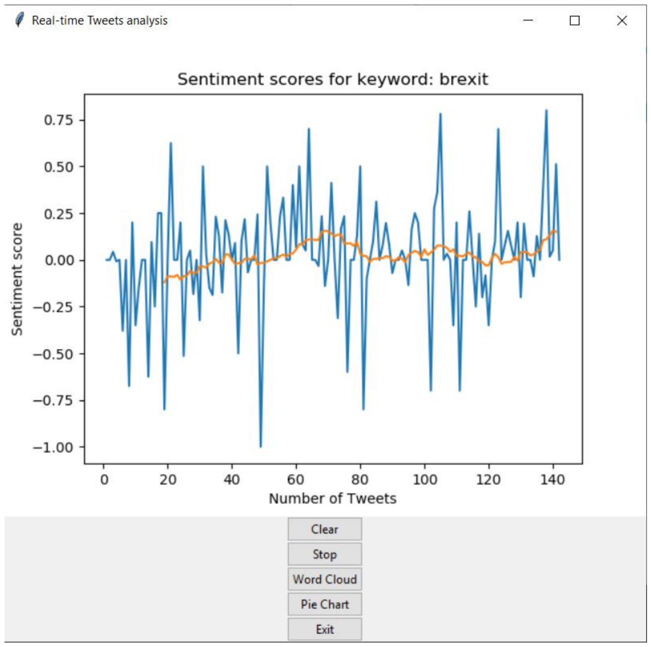


*Figure 35. The GUI for the live graph*

## 5.3 Summary & Remarks

The design and implementation phase will be briefly summarized along with some remarks.

A high-level view of how the system works has been illustrated with two diagrams: one for the system as a whole and another for the sentiment analysis. The specifics of each step of implementation have been comprehensively explained, from getting access to the Twitter API and collecting data to visualizing the analysed results. Code snippets, and visual output where applicable, has been provided to help illustrate how each step has been implemented.

A second case study was conducted between the 13th – 17th January 2019 in anticipation of the "meaningful vote" in the House of Commons on the government's Brexit deal. Tweets were collected every day for two hours, giving a sum of about 65,000 Tweets. The deal was defeated by a big margin, which also lead to a no-confidence vote against Theresa May on the 16th January 2019. After analysing the Tweets, no noticeable trends could be found. A personal interpretation is that people's opinions were split over this event. The visualized results are illustrated below in Figure 36:

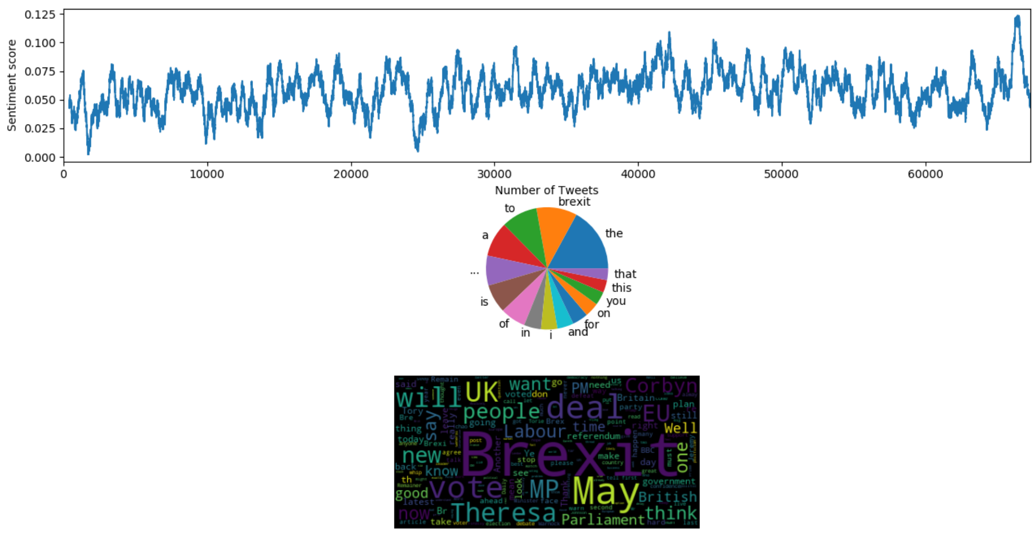


Figure 36. Visual analysis of the 65,000 collected Tweets

# 6 Testing: Verification and Validation

Testing is a vital part of any sizeable project. The aim of testing the solution is to verify that its completion has satisfied the requirements as enumerated in the technical specification. This section will present the different testing practices that have been performed for this project and present the results from these tests. However, it is worth keeping in mind what Edsger Dijkstra has famously said: “Testing shows the presence, not the absence of bugs”. In other words, testing reduces the possibility of undiscovered bugs remaining in the software but even if no bugs are found, it is not a proof of correctness. An agile development methodology was followed for this project. An agile methodology supports early delivery, continual improvement (through testing), and rapid response to change. For example, before developing the GUI, the built-in command-line interface (CLI) was used to test the program and gauge its performance, while after the GUI was developed, the GUI was used instead. Both dynamic and static tests have been used throughout the testing phase. The next section (Section 7) will provide an assessment of the results in the type of a discussion to decide whether the tests performed in this section have satisfied the criteria set in the PID.

## 6.1 Verification

The process of evaluating a program during a given development phase is usually how verification is defined. Verification involves checking that the program meets the requirements compared to its specification. It is often wise to ask the question “Are we building the product right?” during the verification stage. Reviews of the requirement and design specifications, code inspections and building test cases are different ways of verifying a program [15]. A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

## 6.2 Validation

Validation is defined as the process of evaluating a program during or at the end of the development phase to determine whether it satisfies the requirements and expectations of the customer. The corresponding question to ask in the validation phase is: “Are we building the right product?”. Testing the finished program against the user’s needs is the most common way of performing validation. In this case, the requirements that were emphasised in the PID will be the user needs that will be compared against the finished product.

## 6.3 Limitations

As mentioned above, testing reduces the possibility of undiscovered bugs, but that does not mean that the program is bug-free. Testing can only investigate the program requirements, it might not detect issues with other functionalities. We cannot test every valid input as we cannot test every invalid input. As the program cannot be completely tested, a compromise has to be made between the thoroughness and time of the testing phase.

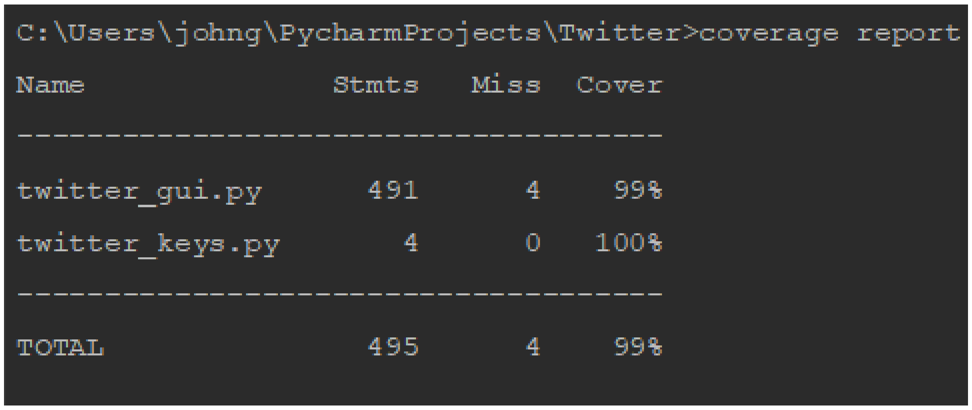
More specifically for this project, Python 3.6.0 and Tweepy 3.6.0 are used. Earlier or later versions of either might cause issues with the program. For example, in Python 3.7 async and await are now reserved keywords which would mess with the tweepy package that implements async based on the 3.6 version of Python. The machine used for testing is running Windows 10 Home. Users with different operating systems might have compatibility issues with the program.

## 6.4 Tests

### 6.4.1 Code Coverage

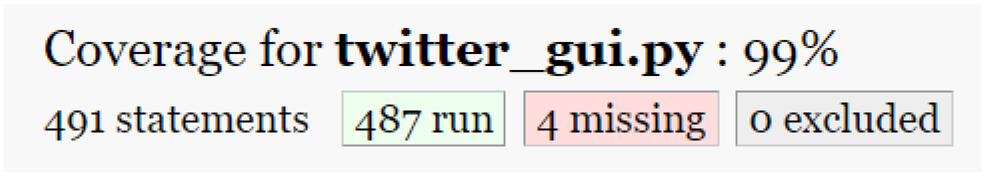
A code coverage module, named “coverage”, was installed to test the coverage of the source code. A program with high test coverage, that has had more of its source code executed during testing, suggests that the program has a lower chance of containing undetected bugs. However, it should be kept in mind that having all of the code covered while testing doesn't mean everything is tested completely, because, as mentioned, it would be impossible to test under every possible situation.

The main Python file for the GUI was tested for coverage which also links to the file for the API keys and tokens. About 500 statements were tested (excluding comments and blank lines) with only 4 statements not executed, as shown below in Figures 37, 38, 39 and 40:



```
C:\Users\johng\PycharmProjects\Twitter>coverage report
Name                  Stmts   Miss  Cover
-----------------------------------------
twitter_gui.py          491      4    99%
twitter_keys.py           4      0   100%
-----------------------------------------
TOTAL                   495      4    99%
```

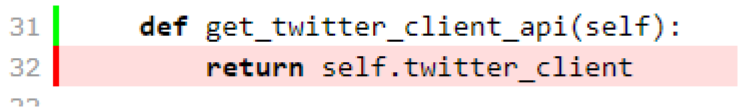
*Figure 37. Code coverage report in the console*



Coverage for **twitter_gui.py** : 99%

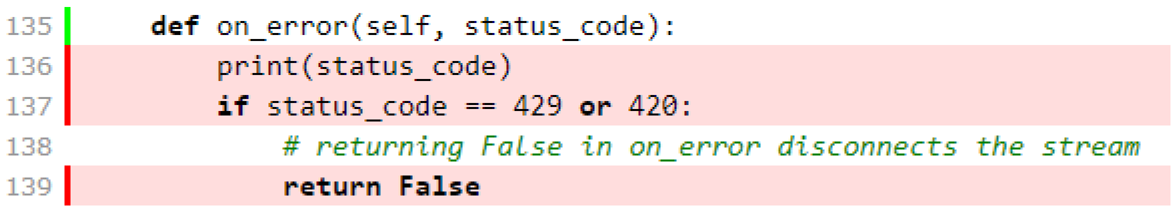
491 statements 487 run 4 missing 0 excluded

*Figure 38. An HTML-style coverage report*



```
    def get_twitter_client_api(self):
        return self.twitter_client
```

*Figure 39. Parts which were not executed*



```
    def on_error(self, status_code):
        print(status_code)
        if status_code == 429 or 420:
            # returning False in on_error disconnects the stream
            return False
```

*Figure 40. Parts which were not executed*

A getter function for the Twitter Client class had been created but not used anywhere in the code. It was redundant since the authentication and access to the API was set up in the init method of the class.

In Figure 40, the on_error method is overridden such that the error code is printed to console and if it's a rate limiting code the stream is disconnected to prevent further time punishments. These statements were never reached since an error hasn't occurred. Conditions that typically tend to go one way have to get appropriate attention otherwise they could prove to be troublesome. Now that there's almost no unused or dead code, more intense testing can begin.

### 6.4.2 Code Review & Walkthrough

Code review, also called peer review, is a static software quality assurance activity in which one or several people check a program mainly by viewing and reading parts of its source code, and they usually do so after implementation or as an intermission of implementation. At least one of the people must not be the code's author [16]. A fellow computer science undergraduate student has reviewed the source code of the program a couple times during different stages of development. For instance, once the data collection part was finished and once again when the GUI was implemented, he reviewed the code to give his opinion in terms of improving code quality and presentability and helped me in trying to find better solutions that could optimize the program.

A walkthrough is a form of software peer review in which a programmer leads interested parties go through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems [17]. Frequent walkthroughs were conducted with the supervisor of this project, Dr Stahl, during our regular meetings where new parts of the program where shown during the development phase. He has provided very important feedback towards enhancing the program by suggesting ideas and concepts that hadn't been previously considered.

### 6.4.3 Unit Testing

A high-level abstraction (Table 1) of how the different parts of the program are divided to classes and how these classes divide to methods or units has been created to facilitate the process of unit and integration testing.

*Table 1*

| Part of the program | Class | Unit |
|---|---|---|
| GUI & visualization | Gui | analysis (latest tweets)<br>analysis1 (day old tweets)<br>analysis3 (3-day old tweets)<br>analysis2 (2 keywords & latest)<br>analysis21<br>analysis23<br>user_analysis (user tweets)<br>image (case study)<br>stream (live tweets) |
| Connection to Twitter | TwitterAuthenticator | authenticate_twitter_app |
| Data collection | TwitterClient | get_tweets (latest tweets)<br>get_tweets1 (day old tweets)<br>get_tweets3 (3-day old tweets)<br>get_tweets21 (1st keyword)<br>get_tweets22 (2nd keyword)<br>get_tweets211<br>get_tweets212<br>get_tweets231<br>get_tweets232<br>get_user_tweets |
| | StdOutListener | on_status<br>on_error |
| Analysis | TweetAnalyzer | clean_tweet<br>analyse_polarity<br>tweets_to_data_frame |

Edge cases should also be considered when dynamically testing the program. An edge case typically involves input values that require special handling in an algorithm behind a computer program. As a measure for validating the behaviour of computer programs in such cases, unit tests are usually created; they are testing boundary conditions of an algorithm, function or method [17].

Measuring the performance, in terms of speed, of parts of a program in Python was performed with the built-in library called time. By measuring and comparing the time taken to execute a part of the program after and before changes were made, it made it easier to optimize the speed of the program.

A table of the test cases that specified the different tests that were performed on separate units/methods of the system can be found below in Table 2:

Table 2

| Test case ID | Test description | Unit name | Expected outcome | Observed outcome |
|---|---|---|---|---|
| 1 | Twitter Authentication using wrong keys | authenticate_twitter_app | Authorization error | As expected, HTTP 401 (Unauthorized) error |
| 2 | Twitter Authentication using correct keys | authenticate_twitter_app | No authorization errors | As expected, no errors occurred, and access was granted |
| 3 | Test how the Tweet cleaning method responds to edge cases, for example Unicode characters, multiple usernames mentioned, and escaped strings | clean_tweet | Only text that is valuable for sentiment analysis is kept | String is stripped of redundant and possibly problematic characters |
| 4 | Given an object with similar attributes as a Tweet test whether it is stored as a data frame | tweets_to_data_frame | Selected object's full_text attribute is stored in a data frame for later use | The full_text attribute from the Tweet is stored in a data frame. |
| 5 | Analyse various strings of different types of sentimental polarity or complexity | analyse_polarity | Correct sentiment score is returned back | For the most part, the score agrees with input strings, with slight inaccuracies on sarcasm |
| 6 | Analyse the differences between the two types of analysers, NaiveBayesAnalyzer & PatternAnalyzer | analyse_polarity | Best performing sentiment analyser is selected, based on speed and accuracy | As explained in the Implementation section, PatternAnalyzer outperforms NaiveBayesanalyzer, and is therefore selected |
| 7 | Test the different GUI widgets' placement appearance and underlying code | Main function | An overall consistent and presentable GUI | A simple but concise and robust GUI |

Python has a built-in unit testing module called unittest. It provides a rich set of tools for constructing and running tests. Unittest requires that:

- The tests are put into classes as methods
- A series of special assertion methods in the unittest.TestCase class is used instead of the built-in assert statement

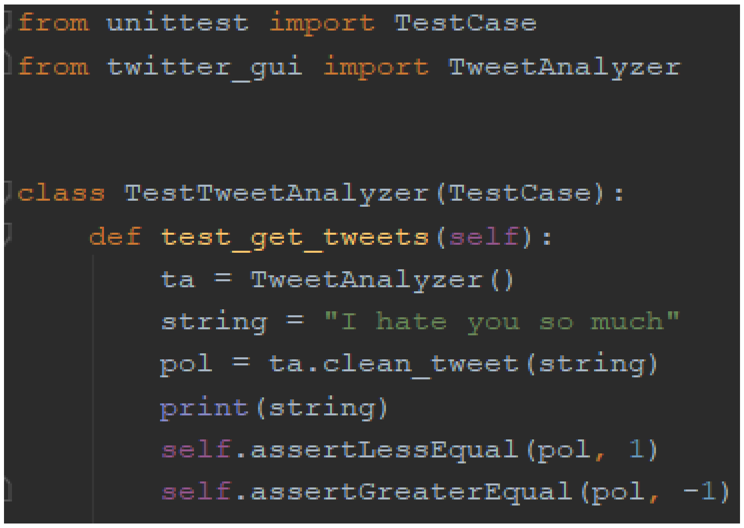
Pytest is another module that has some other great features:

- Support for the built-in assert statement instead of using special self.assert*()methods
- Support for filtering for test cases

Due to its simplicity and clarity, unittest has been selected instead of pytest. The unittest module is not limited to unite testing. It can also be used for integration testing, since single methods often depend on other methods or parts of the program.

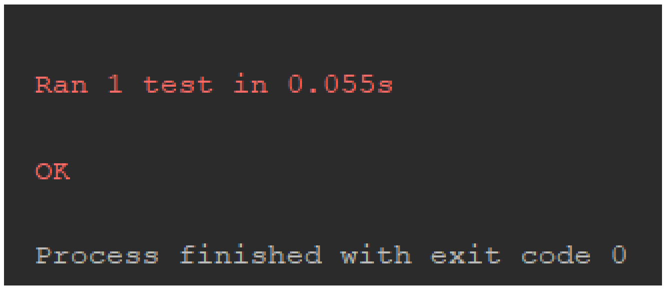
Other than using testing packages, testing included debugging the program using the PyCharm debugger's features, such as breakpoints and stepping, as well as using the print function to confirm that a part of the code has been executed, especially when testing edge cases.

An example of a unit test on the analyse_polarity method is that it can only take values between -1 and 1. Using the unittest module and the TweetAnalyzer class a unit test can be done like shown in Figures 41 & 42:



```
from unittest import TestCase
from twitter_gui import TweetAnalyzer


class TestTweetAnalyzer(TestCase):
    def test_get_tweets(self):
        ta = TweetAnalyzer()
        string = "I hate you so much"
        pol = ta.clean_tweet(string)
        print(string)
        self.assertLessEqual(pol, 1)
        self.assertGreaterEqual(pol, -1)
```

*Figure 41. A unit test using the TestCase class*



```
Ran 1 test in 0.055s

OK

Process finished with exit code 0
```

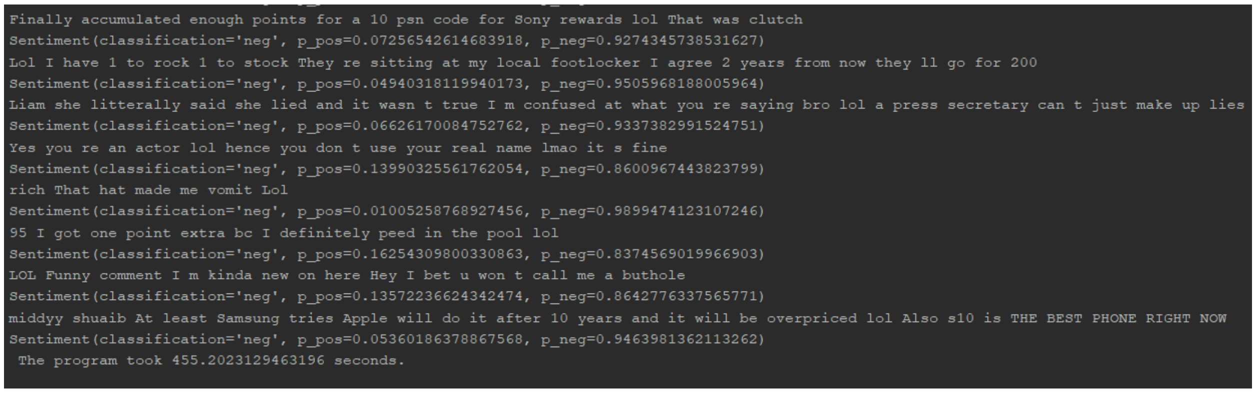
*Figure 42. The results from the unit test*

In Figure 41 the TestCase class is imported from the unittest package as well as the TweetAnalyzer class, which contains the method that analyses tweets, analyse_polarity.

A class named TestTweetAnalyzer that inherits from the TestCase class is created. PyCharm can help with testing by creating stub files from the class to be tested. The last two statements require that the "pol" variable, which holds the sentiment polarity value of the given string, is respectively less or equal to 1 and greater or equal to -1.

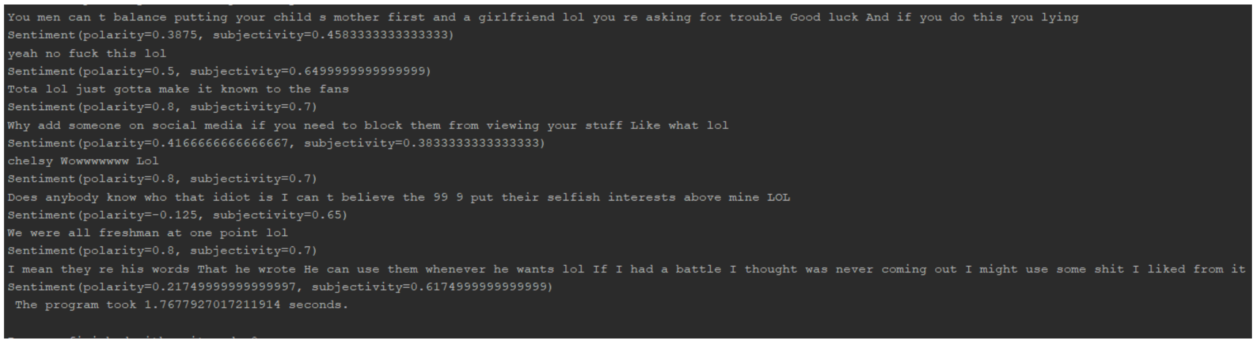
In Figure 42 the test is run and is successfully passed, noted with the red "OK" text.

The two sentiment analysers have been carefully tested and compared. Their performance has been timed using the time package. Figures 43 & 44 show the differences in the time taken for each analyser:



```
Finally accumulated enough points for a 10 psn code for Sony rewards lol That was clutch
Sentiment(classification='neg', p_pos=0.07256542614683918, p_neg=0.9274345738531627)
Lol I have 1 to rock 1 to stock They re sitting at my local footlocker I agree 2 years from now they ll go for 200
Sentiment(classification='neg', p_pos=0.04940318119940173, p_neg=0.9505968188005964)
Liam she litterally said she lied and it wasn t true I m confused at what you re saying bro lol a press secretary can t just make up lies
Sentiment(classification='neg', p_pos=0.06626170084752762, p_neg=0.9337382991524751)
Yes you re an actor lol hence you don t use your real name lmao it s fine
Sentiment(classification='neg', p_pos=0.13990325561762054, p_neg=0.8600967443823799)
rich That hat made me vomit Lol
Sentiment(classification='neg', p_pos=0.01005258768927456, p_neg=0.9899474123107246)
95 I got one point extra bc I definitely peed in the pool lol
Sentiment(classification='neg', p_pos=0.16254309800330863, p_neg=0.8374569019966903)
LOL Funny comment I m kinda new on here Hey I bet u won t call me a buthole
Sentiment(classification='neg', p_pos=0.13572236624342474, p_neg=0.8642776337565771)
middyy shuaib At least Samsung tries Apple will do it after 10 years and it will be overpriced lol Also s10 is THE BEST PHONE RIGHT NOW
Sentiment(classification='neg', p_pos=0.05360186378867568, p_neg=0.9463981362113262)
 The program took 455.2023129463196 seconds.
```

*Figure 43. Naïve Bayes analyser testing*

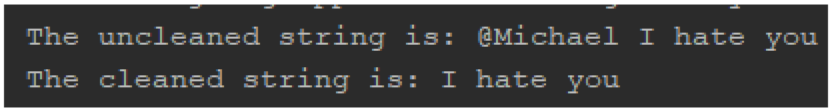


```
You men can t balance putting your child s mother first and a girlfriend lol you re asking for trouble Good luck And if you do this you lying
Sentiment(polarity=0.3875, subjectivity=0.4583333333333333)
yeah no fuck this lol
Sentiment(polarity=0.5, subjectivity=0.6499999999999999)
Tota lol just gotta make it known to the fans
Sentiment(polarity=0.8, subjectivity=0.7)
Why add someone on social media if you need to block them from viewing your stuff Like what lol
Sentiment(polarity=0.4166666666666667, subjectivity=0.3833333333333333)
chelsy Wowwwwwww Lol
Sentiment(polarity=0.8, subjectivity=0.7)
Does anybody know who that idiot is I can t believe the 99 9 put their selfish interests above mine LOL
Sentiment(polarity=-0.125, subjectivity=0.65)
We were all freshman at one point lol
Sentiment(polarity=0.8, subjectivity=0.7)
I mean they re his words That he wrote He can use them whenever he wants lol If I had a battle I thought was never coming out I might use some shit I liked from it
Sentiment(polarity=0.21749999999999997, subjectivity=0.6174999999999999)
 The program took 1.7677927017211914 seconds.
```

*Figure 44. Pattern analyser testing*

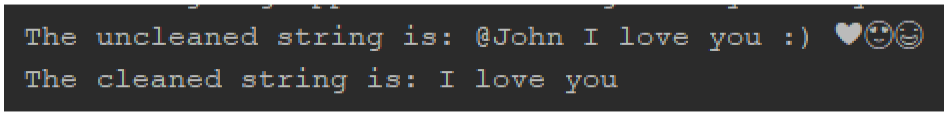
The differences between the two analysers is massive. For 100 Tweets it took the Naïve Bayes analyser 455 seconds while it took just short of 2 seconds for the Pattern Analyser to finish. In addition to the performance, the Pattern Analyser is more accurate than the Naïve Bayes Analyser. The Naïve Bayes analyser has mislabelled more than half of the 100 Tweets, while the Pattern Analyser has only mislabelled about 15 Tweets.

The clean_method was also thoroughly manually tested. In Figures 45 & 46 below are some examples of some strings before and after pre-processing:



```
The uncleaned string is: @Michael I hate you
The cleaned string is: I hate you
```

*Figure 45. Example of a cleaned Tweet with a Twitter mention*



```
The uncleaned string is: @John I love you :) ♥🙄😆
The cleaned string is: I love you
```

*Figure 46. Example of a cleaned Tweet with emojis*

For the GUI, most of the testing comprised of a lot of trial and error until the visual output was appealing.

### 6.4.4 Integration Testing

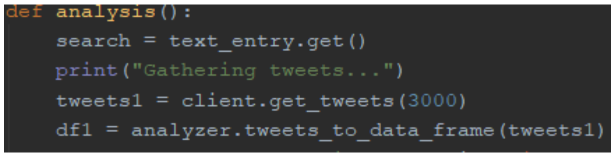
Integration testing is a level of software testing where individual units are combined and tested as a group. Integration testing has been the most attended to and important part of the verification phase since most parts of the program depend on other parts. A table of the test cases that specified the different tests that were performed on the difference modules, that are composed of smaller units, can be found in Table 3 below:

*Table 3*

| Test case ID | Test description | Module description | Units name | Expected outcome | Observed outcome |
|---|---|---|---|---|---|
| 1 | Data collection on live Tweets | Collecting live Tweets | authenticate_twitter_app, on_status | A session of live Tweets is commenced | A stream of live Tweets |
| 2 | Data collection on existing Tweets | Collecting existing Tweets | authenticate_twitter_app, get_tweets (all methods) | Tweets are retrieved based on the given parameters | Tweets based on the given query are returned |
| 3 | Twitter rate limits | Test the rate limits on the API methods and equally distribute the requests between them | authenticate_Twitter_app, get_tweets (all methods) | After 18,000 Tweets returned by the search function get an error code | HTTP error code 429 was raised, which means "Too Many Requests" |
| 4 | Storing real Tweets into a data frame | Collecting Tweets and storing them in a data frame for later data manipulation | authenticate_twitter_app, get_tweets (all methods), tweets_to_data_frame | Tweets are stored in a data frame | Tweets are displayed on the CLI as a data frame |
| 5 | Cleaning existing Tweets | Collecting existing Tweets and prepare the text for sentiment analysis | authenticate_twitter_app, get_tweets (all methods), tweets_to_data_frame, clean_tweet | Redundant and escaped characters are removed and unescaped | A clean Tweet with only the pure text is kept |
| 6 | Cleaning live Tweets | Collecting live Tweets and prepare the text for sentiment analysis | authenticate_twitter_app, on_status, on_error, clean_tweet | Redundant and escaped characters are removed and unescaped | A clean Tweet with only the pure text is kept |
| 7 | Sentiment analysis on live Tweets | Get the cleaned Tweet and perform sentiment analysis on it | authenticate_twitter_app, clean_tweet, on_status, on_error, analyse_polarity | A sentiment score is attached to each Tweet based on the TextBlob PatternAnalyzer | Every cleaned Tweet is printed just before its sentiment score |
| 8 | Sentiment analysis on existing Tweets | Get the cleaned Tweet and perform sentiment analysis on it | authenticate_twitter_app, clean_tweet, get_tweets (all methods), tweets_to_data_frame, analyse_polarity | A sentiment score is attached to each Tweet based on the TextBlob PatternAnalyzer | Each Tweet is shown with a sentiment column next to them |
| 9 | Visualization of collected Tweets | Clean the collected Tweets, analyse their polarity and illustrate their results | clean_tweet, analyse_polarity, pie, wordcloud, plot | A time series on the sentiment scores and a pie chart and a word cloud are presented | As expected. |

| | | | | | |
|---|---|---|---|---|---|
| 10 | Being able to use the GUI for the above | Link the GUI buttons' functions to the rest of the program | All methods from the Gui class, which implement the remaining methods from all other classes | Each button calls a function that plots the analysed data or shows the case study or exits | When a query is given, 3 visual representations are available. |

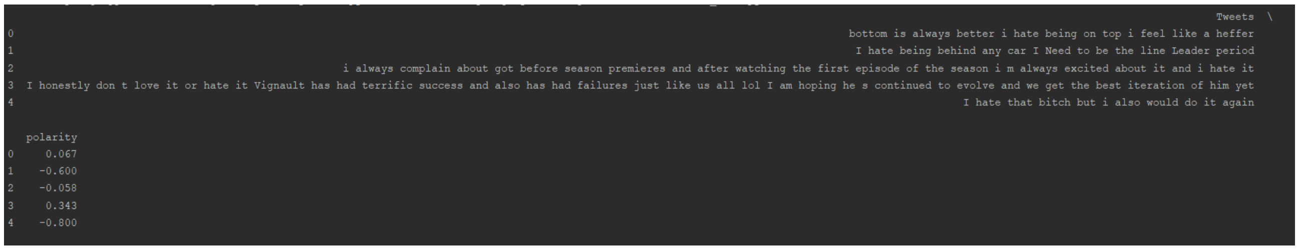
As mentioned above, the print function has been used quite a lot to confirm that a point in the code has been reached. For example, to verify that a button has been pressed in addition to confirming that the input field it is associated to is taken and stored in a variable and, in general that the function/command it references, has therefore been called, print statements have been used as Figure 47 shows below:



```
def analysis():
    search = text_entry.get()
    print("Gathering tweets...")
    tweets1 = client.get_tweets(3000)
    df1 = analyzer.tweets_to_data_frame(tweets1)
```

*Figure 47. Using a print statement*

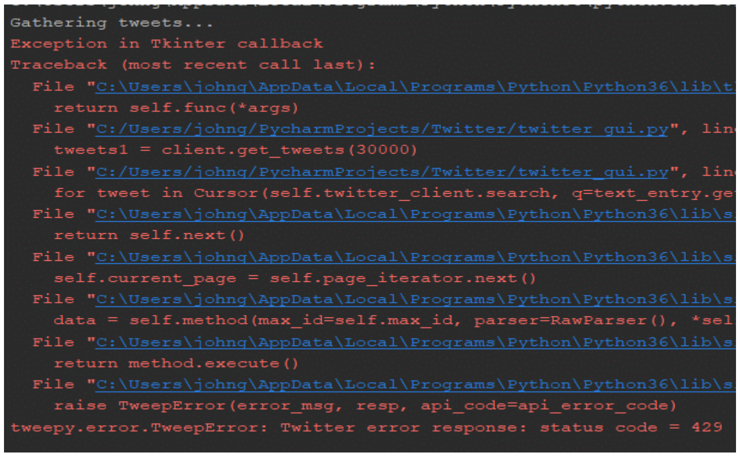
Here, cleaned real Tweets are saved into a data frame and displayed in the command-line interface. The authenticate_twitter_app is called as soon as a request to the API is made. Each Tweet is cleaned and then stored in the data frame. The result is illustrated below in Figure 48:



```
                                                                                                                                                       Tweets  \
0                                                                                                     bottom is always better i hate being on top i feel like a heffer
1                                                                                                      I hate being behind any car I Need to be the line Leader period
2                                                   i always complain about got before season premieres and after watching the first episode of the season i m always excited about it and i hate it
3  I honestly don t love it or hate it Vignault has had terrific success and also has had failures just like us all lol I am hoping he s continued to evolve and we get the best iteration of him yet
4                                                                                                                          I hate that bitch but i also would do it again

   polarity
0     0.067
1    -0.600
2    -0.058
3     0.343
4    -0.800
```

*Figure 48. Tweets stored in a data frame printed in console*

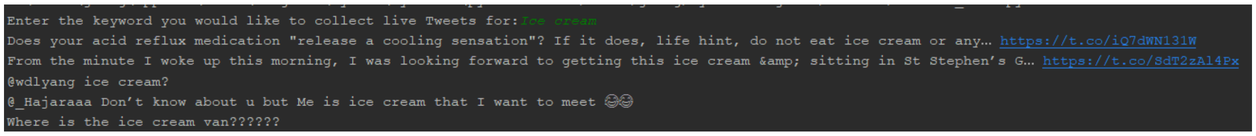
The rate limit imposed by Twitter raises an error when more requests than the limit are made. In Figure 49 below, 30,000 Tweets are requested, using the API.search method that is called in the get_tweets method:



```
Gathering tweets...
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\johng\AppData\Local\Programs\Python\Python36\lib\t
    return self.func(*args)
  File "C:/Users/johng/PycharmProjects/Twitter/twitter_gui.py", lin
    tweets1 = client.get_tweets(30000)
  File "C:/Users/johng/PycharmProjects/Twitter/twitter_gui.py", lin
    for tweet in Cursor(self.twitter_client.search, q=text_entry.ge
  File "C:\Users\johng\AppData\Local\Programs\Python\Python36\lib\s
    return self.next()
  File "C:\Users\johng\AppData\Local\Programs\Python\Python36\lib\s
    self.current_page = self.page_iterator.next()
  File "C:\Users\johng\AppData\Local\Programs\Python\Python36\lib\s
    data = self.method(max_id=self.max_id, parser=RawParser(), *sel
  File "C:\Users\johng\AppData\Local\Programs\Python\Python36\lib\s
    return method.execute()
  File "C:\Users\johng\AppData\Local\Programs\Python\Python36\lib\s
    raise TweepError(error_msg, resp, api_code=api_error_code)
tweepy.error.TweepError: Twitter error response: status code = 429
```

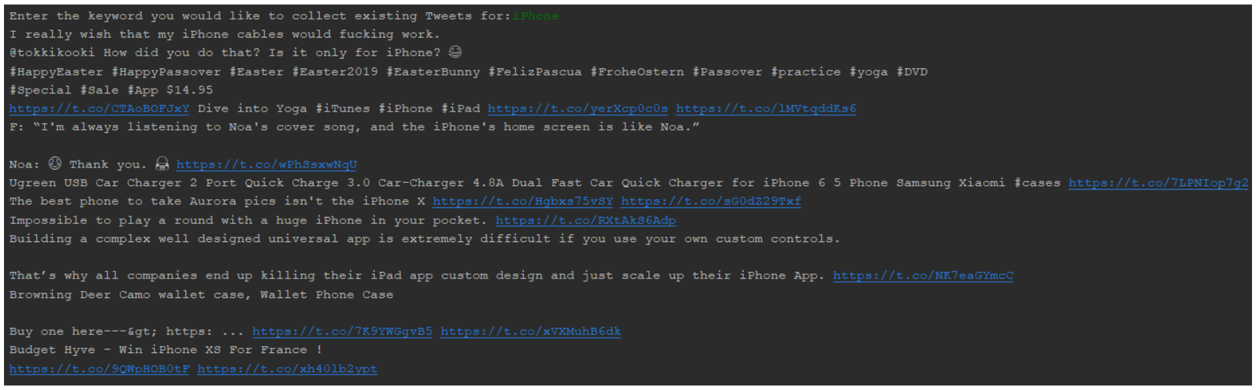
*Figure 49. HTTP 429 response code: Too many requests*

Testing on data collection mainly meant that the correct data was shown in the command-line interface. For example, in Figure 50 and Figure 51 the user is prompted for input. After the keyword is entered in, uncleaned Tweets are shown in the CLI.



```
Enter the keyword you would like to collect live Tweets for:Ice cream
Does your acid reflux medication "release a cooling sensation"? If it does, life hint, do not eat ice cream or any… https://t.co/iQ7dWN131W
From the minute I woke up this morning, I was looking forward to getting this ice cream &amp; sitting in St Stephen's G… https://t.co/SdT2zAl4Px
@wdlyang ice cream?
@_Hajaraaa Don't know about u but Me is ice cream that I want to meet 😂😂
Where is the ice cream van??????
```

Figure 50. Get live Tweets from an input keyword



```
Enter the keyword you would like to collect existing Tweets for:iPhone
I really wish that my iPhone cables would fucking work.
@tokkikooki How did you do that? Is it only for iPhone? 😂
#HappyEaster #HappyPassover #Easter #Easter2019 #EasterBunny #FelizPascua #FroheOstern #Passover #practice #yoga #DVD
#Special #Sale #App $14.95
https://t.co/CTAoBOFJxY Dive into Yoga #iTunes #iPhone #iPad https://t.co/yerXcp0c0s https://t.co/lMVtqddKs6
F: "I'm always listening to Noa's cover song, and the iPhone's home screen is like Noa."

Noa: 😢 Thank you. 🎧 https://t.co/wPhSsxwNqU
Ugreen USB Car Charger 2 Port Quick Charge 3.0 Car-Charger 4.8A Dual Fast Car Quick Charger for iPhone 6 5 Phone Samsung Xiaomi #cases https://t.co/7LPNIop7g2
The best phone to take Aurora pics isn't the iPhone X https://t.co/Hgbxs75vSY https://t.co/sG0dZ29Txf
Impossible to play a round with a huge iPhone in your pocket. https://t.co/RXtAkS6Adp
Building a complex well designed universal app is extremely difficult if you use your own custom controls.

That's why all companies end up killing their iPad app custom design and just scale up their iPhone App. https://t.co/NK7eaGYmcC
Browning Deer Camo wallet case, Wallet Phone Case

Buy one here---&gt; https: ... https://t.co/7K9YWGgvB5 https://t.co/xVXMuhB6dk
Budget Hyve - Win iPhone XS For France !
https://t.co/9QWpHOB0tF https://t.co/xh40lb2ypt
```

Figure 51. Get existing Tweets from an input keyword

Another area, in which multiple units were involved in, was testing how data is visualized. With the same input keyword as above, the sentiment scores and word frequency are visualized, represented as a time series, a pie chart and a word cloud. Figure 52 illustrates the visualized results of the keyword "iPhone":

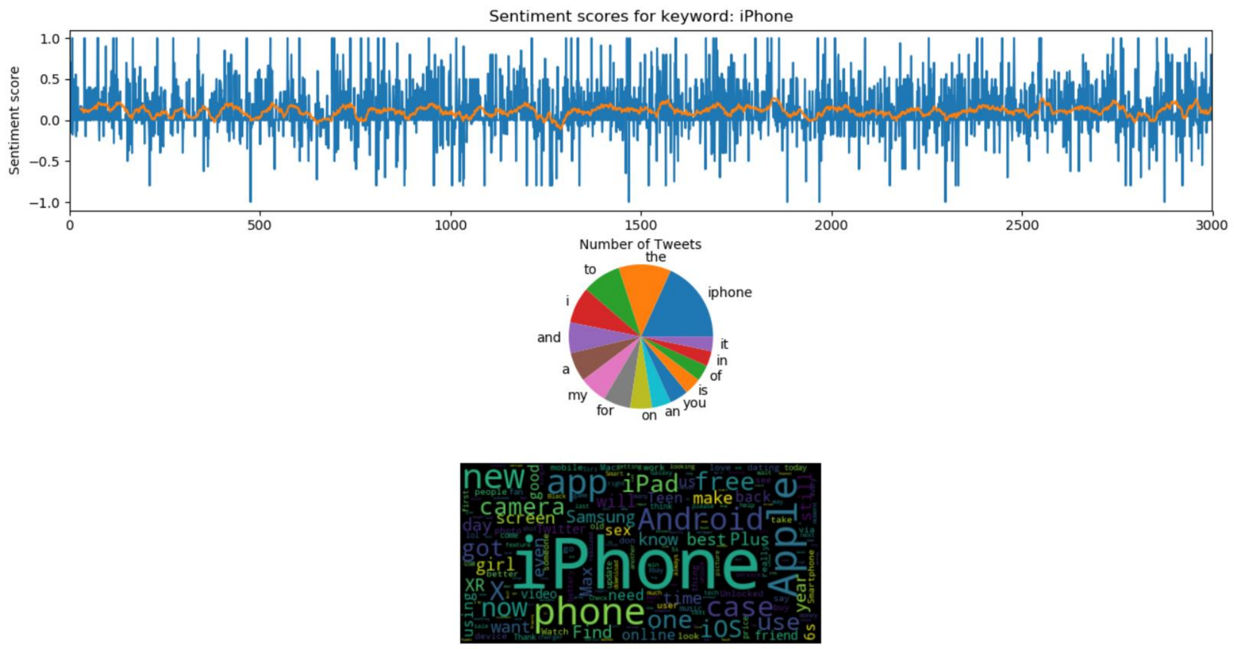


Figure 52. Visualizing query results

### 6.4.5 Acceptance Testing

After verifying that the system brings the functionality that is required in the previous subsections, formal testing with respect to user needs and requirements has to be conducted to validate, whether the system satisfies the acceptance criteria. The objective is to provide assurance that the finished product meets both the functional and non-functional requirements. In this case, the PID will be used to compare its requirements and specifications against the system. The solution must not deviate to a great extent from the PID and conform to the specified requirements.

As per the PID, the key features and functions of the project when first conceptualized are listed as:

- Authenticate access to the Twitter API.
- Grab arbitrary Tweets based on a particular subject/keyword.
- Analyse and manipulate those Tweets.
- Give a sentiment score to the Tweet.
- Make use of machine learning and a big dataset or user feedback to improve results.
- Produce visualized data in graphs or plots.

Almost of all the requirements set in the PID seem to be met. Access to the Twitter API is successfully granted; Tweets are grabbed given an input keyword. They are then analysed and manipulated and given a sentiment score. Machine learning, and user feedback were not put to use since the pattern analyser does a pretty good job without the need of using a machine learning approach or user feedback. Finally, analysed data is visually represented using graphs and plots as specified.

The requirements that the solution was developed against, i.e. the technical specification, should also be considered in this testing phase. The following requirements were set:

- The solution must provide an intuitive interface for the end-user.
- The solution should be as robust and bug-free as possible.
- The solution must be able to successfully authenticate and access the Twitter API.
- The developer needs to develop or implement an existing sentiment analysis model, based on a classification algorithm.
- The solution must be able to, in real time, grab real time Tweets based on a user input keyword and simultaneously visualize their sentiment score.
- The solution must be able to grab existing Tweets again based on a user input keyword and visualize their sentiment score.
- The solution must be able to grab existing Tweets based on two user input keywords and visualize the comparison in their sentiment score.
- The user must be given several recency options when searching existing Tweets.
- The solution must be able to grab existing Tweets from a Twitter user, requesting the user to input the account's Twitter handle (e.g. realDonaldTrump) and visualize their sentiment score.
- A case study of a reasonably large number of collected Tweets must be conducted and be available to the user for reference.
- The visualized output should also include a moving average to smooth out short-term fluctuations.

Almost all of the requirements set in the technical specification above were met, with the exception of using a classification algorithm, as well as almost all of the requirements set in the PID. It is safe to conclude that the finished solution conforms to what is expected.

# 7 Discussion: Contribution and Reflection

The previous sections carefully and methodically exhibited the process of putting the design of the solution into effect, displaying the results and testing for possible errors. In this section the successes and failures or limitations of this project and its implementation will be emphasised, as well as discussing the test results in detail and compare relevant previous work. Finally, I will reflect on the learning experience throughout this project.

From the beginning, it was pretty clear what the final solution should do and how it should do it. Therefore, with that in mind, exhaustive background research on pertinent existing work has been carried out. Various possible solutions, for different subparts of the project, were considered out of which the most suitable ones were selected, implemented and tested. Prior to this, extensive knowledge of data science concepts as well as a programming language that can offer data science functionalities, such as efficiently analyse and visualize data, had to be acquired as this had not been covered in the curriculum at that point.

This project has mostly had positive results and only a couple let-downs. First and most importantly, time management is the most important part when dealing with a major project that requires substantial amount of work and planning. A time plan and a project plan were set in the early stages of the project, in the PID, in order to make sure that methodical and systematic amount of work is done throughout the duration of this project. This was achieved by splitting the project work into sections/milestones and adding tasks for each of these sections. By setting a time plan, it encouraged and ensured that each milestone was achieved in a timely manner. Having a high-level view of the design solution and the time plan from day one, gave a very clear idea and impression of how everything should be laid out which smoothed the entire process. This also allowed for different implementations to be explored. For example, when choosing which type of approach to follow for the sentiment analyser, both a machine learning approach as well as a lexicon-based approach were implemented and scrutinized, weighing the pros and cons of each approach to finally decide on the most beneficial one.

The implementation of the project had also been mostly successful. From setting up access to the Twitter API, collecting, pre-processing, manipulating, analysing and visualizing the data to the graphical user interface, it had been a smooth and pleasant experience. Any complications or difficulties that had arisen, were dealt with in the best possible way that would cause the least disruption on the system while still keeping the design of the system roughly the same.

The biggest limitation of this project was that there's a moderately strict limit on the number requests that can be made to the Twitter API, using the free plan, on existing Tweets. This has significantly limited the possible use cases of the solution significantly since the user cannot possibly get the full picture on a query just from a few thousand Tweets, that is too small of a dataset sample. However, since there is not a limit in accessing live Tweets, this part could be considered as the most valuable and integral part of the project. It can capture a bigger picture compared to accessing existing Tweets.

The two case studies make for a contrasting experience. On one hand, both case studies were conducted with attention to every detail and the data was collected, stored, processed and visualized in a professional manner. The first case study between the 23rd – 27th November 2018 has given some interesting results from which useful findings can be inferred. The trends in the results were then examined. On the other hand, the second case study, conducted between the 13th – 17th January 2019, had not given any actual information on the data. There was little to no change in the data when the Brexit deal was rejected in the House of Commons or during the no-confidence vote against Theresa May.

When creating the PID, some main objectives and outputs for this project were initially set. These objectives and outputs are shown below in Table 4:

*Table 4*

Objectives:

- Getting familiar with the libraries and the language
- Create an outline of the high-level side of the system
- Create a first draft of the system
- Build an analytics workflow
- Sentiment analysis
- Test: unit, integration, acceptance, system
- Finally adapt a model to run on real-time Twitter feeds including user feedback to facilitate adaptation to changes

Outputs:

- Ability to code in chosen language and initiate development of my system
- Have a general view of how the system will operate and from that be able to know how to develop it
- By having a first draft of the system, instead of a high-level view, I will be able to get feedback that will help me significantly improve my system
- Having an analytics workflow added to the system will massively improve the developing process
- Sentiment analysis is the last part of the system development
- By testing, I make sure that any test cases are covered, and no unexpected outcomes can occur
- Have a final robust version of the system that will allow me to begin writing up the documentation

Comparing the results of the, now finished, project to the objectives and outputs set in the PID, it would be reasonable to say that everything went as planned. Virtually all the outputs and objective were successfully carried through.

Although they were inconsequential, the project had some shortcomings. The technical specification in the PID states that some sort of machine learning model would be used to give a score to the data. However, after some more comprehensive research and observed analysis, it was decided that a lexicon-based approach would be more suitable, and thus this approach was rejected.

It was also mentioned that the user could provide feedback in some way to help improve the accuracy of the results. The idea does seem possible, but it would not make sense for the end-user to view the Tweets and give feedback on the prediction. In spite of that, if this idea was implemented the user would have to either give feedback on all of the results or a smaller subset. Firstly, it would be very time consuming to manually go through thousands of Tweets and decide on the sentiment they depict and secondly, it might prove to have little consequence on future results when feedback is given on a small subset. This task would be more suitable to be done by the developer when training the model. Besides that, this solution is developed with the aim to expect as little input from the end-user as possible.

Some additional ideas and concepts were considered but not put into effect. During implementation of the sentiment analyser, a third approach was also attempted: a hybrid combination of a machine learning approach on an annotated corpus. The following steps were established:

- The features to be used by the machine learning approach are identified and separated.
- The annotated corpus to be used for training and validation of the best classifier at different corpus sizes is built by the system.
- Sentiment lexicon of different sizes is built using the annotated corpus.
- These different approaches are combined and tested for better and optimized results.

However, the effort of implementing this approach could not be put in practice, and it was consequently rejected.

During the implementation of the GUI, there were also some ideas that fell through. For example, getting Tweets by time of the day instead of only days (in the current implementation, when a user is searching Tweets, the 3 chronological options are: latest Tweets, one-day old Tweets and three-day old Tweets). However, tweepy, and the Twitter API, only allows to specify dates when making a query. A workaround for this could be getting the Tweets for the particular day and, by using the created_at attribute of the Tweet that gives the time that a Tweet has been created at (for example: "created_at": "Sun Feb 25 19:31:07 +0000 2018"), only keep the ones with the selected time. For that to happen, a substantial number of requests has to be made so that the filtered number of Tweets is of a reasonable size; This means that the rate limit would be reached easier with a significantly smaller number of actually processed Tweets. In addition to that, it would most importantly take more processing and slow down the process.

There were also issues with TkInter. Currently the GUI looks like this:

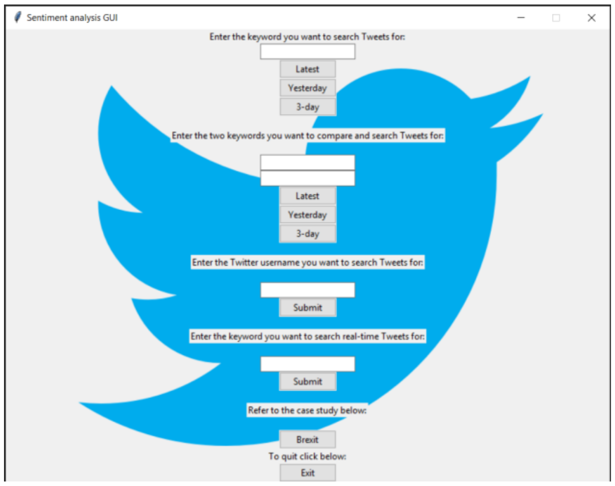

Sentiment analysis GUI
Enter the keyword you want to search Tweets for:
Latest
Yesterday
3-day
Enter the two keywords you want to compare and search Tweets for:
Latest
Yesterday
3-day
Enter the Twitter username you want to search Tweets for:
Submit
Enter the keyword you want to search real-time Tweets for:
Submit
Refer to the case study below:
Brexit
To quit click below:
Exit

*Figure 53. The system's GUI*

Something more compact would be preferred like the picture shown in Figure 54. However, it was very hard for me to create something like this in TkInter, as many problems occurred while trying to implement it. Ultimately, this idea has been rejected.

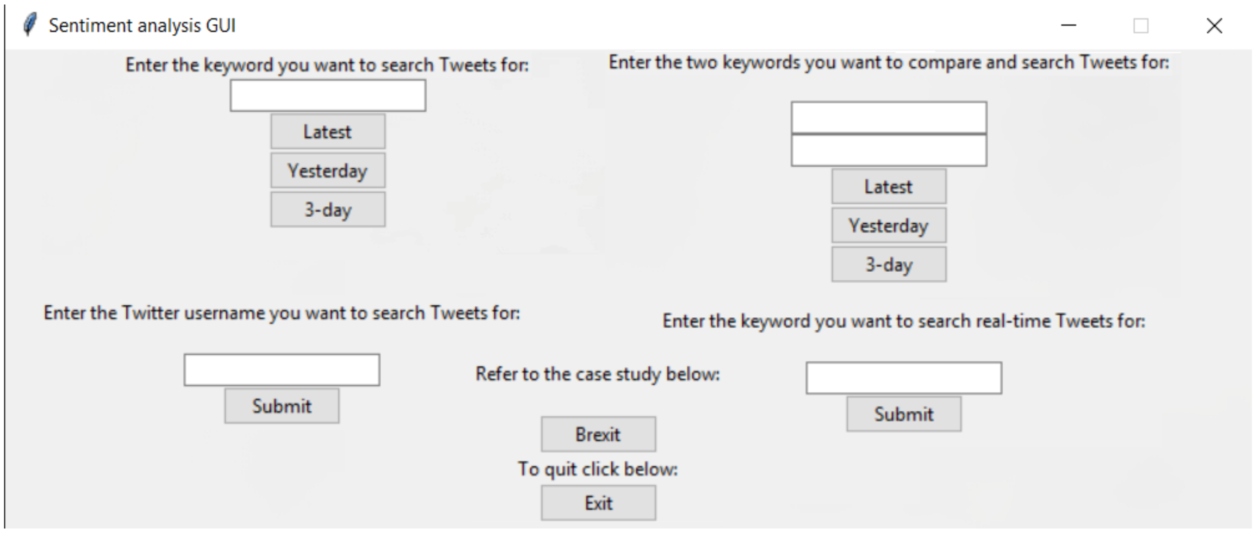
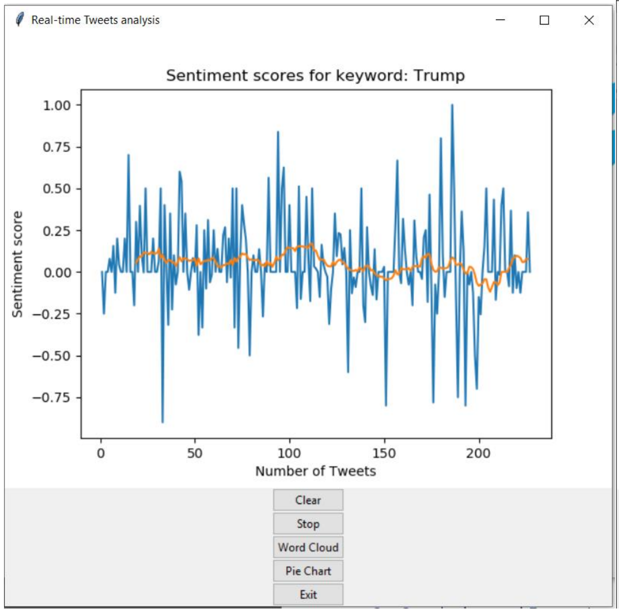

Sentiment analysis GUI
Enter the keyword you want to search Tweets for:
Latest
Yesterday
3-day
Enter the two keywords you want to compare and search Tweets for:
Latest
Yesterday
3-day
Enter the Twitter username you want to search Tweets for:
Submit
Enter the keyword you want to search real-time Tweets for:
Refer to the case study below:
Submit
Brexit
To quit click below:
Exit

*Figure 54. Intended GUI*

The co-operation of TkInter and matplotlib was also tricky to handle, especially for the real-time analysis part. As explained in the implementation section, the live graph works on a new Tk window. A canvas is drawn using the FigureCanvasTkAgg module that provides a backend for rendering on a Tk Canvas. After existing Tweets are searched and analysed, their visualizations are displayed on a matplotlib figure which is split into 3 subplots, one for each visualization. The same cannot be done for the pie chart and word cloud when analysing Tweets in real-time as the Tweets accumulated constantly change over time.

Therefore, only one subplot is used, for the live graph. When trying to add another subplot, it will mess up the figure because the shape of the figure has already been defined as having only one plot. The wordcloud library accommodates this issue by allowing use of another library called pillow, which is a wrapper for PIL - Python Imaging Library. Pillow allows the picture to be temporarily stored and viewed using Microsoft Photos. However, the pie chart method, which is in the matplotlib library, has to be drawn on a figure, which results in the graph to override the live graph, while still keeping the title and the axis labels. It should also be noted that since the pie chart overrides the live graph, access to that visualization is lost. In Figures 55 & 56 below, the change can be seen when pressing the Pie Chart button:



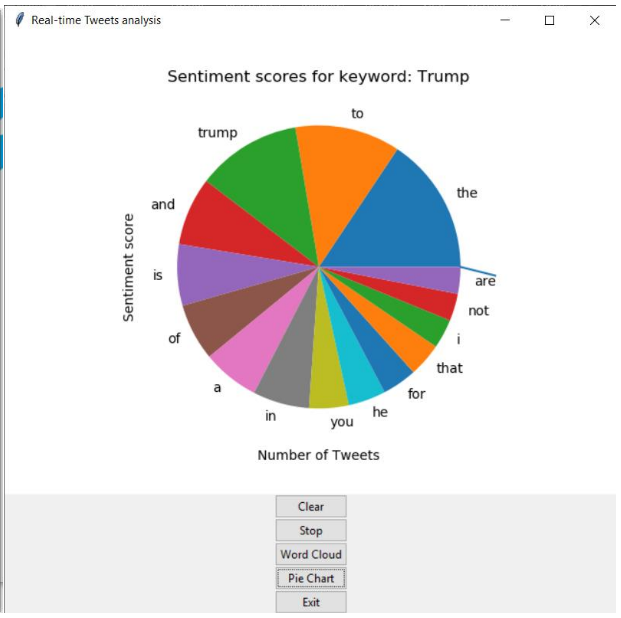
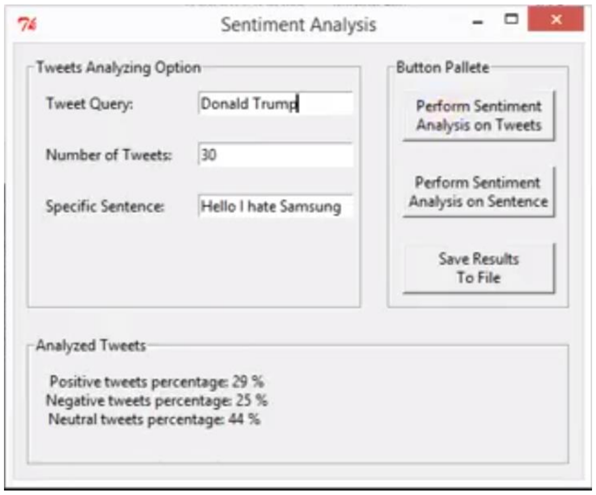


*Figure 55. Time series of live Tweets*

*Figure 56. Word frequency Pie chart on the same query*

Other than that, the testing results were generally very convincing in improving the robustness of the system. Most of the problems faced during development and testing were addressed and dealt with as soon as possible with the most optimal solution possible in mind. Unit testing and integration were the main techniques used for verifying the functionality of the system, while acceptance testing was used for validating that the requirements set in the PID were met. During different phases of the development, the program was thoroughly tested to ensure that the solution was as bug free as possible.

When comparing the finished version of this project to other similar existing projects, it makes for an interesting competition. For example, in Figure 57, a simpler implementation is shown. A Twitter search query and the number of Tweets to be searched can be specified. As well as that, a single specific input sentence can be input and analysed. The results are not visualised but instead classified as positive, neutral and negative. Nonetheless, there is a button to save the results to a file which can be very useful.



Tweet Query: Donald Trump
Number of Tweets: 30
Specific Sentence: Hello I hate Samsung

Positive tweets percentage: 29 %
Negative tweets percentage: 25 %
Neutral tweets percentage: 44 %

*Figure 57. A similar project*

A more complex application built using Dash, a framework for building analytical web applications, has also been used to compare the differences to this project. It predicts the sentiment score behind real-time Tweets regarding a currency, or crypto currency, and investigates the possible impact it could have on the currency's price, by looking at the trends the sentiment graph is showing. The developer has specifically chosen to concentrate on analysing sentiment on currencies, as shown in Figure 58, which has limited the project's use cases but can be very valuable for that area. The visualizations, albeit basic and unfashionable, clearly present the data without redundant information. There does not seem like there is a great deal of analysis in the backend. Finally, Retweets should be ignored in analysis but in this project they are not, as it can be seen on the bottom left window in Figure 59.

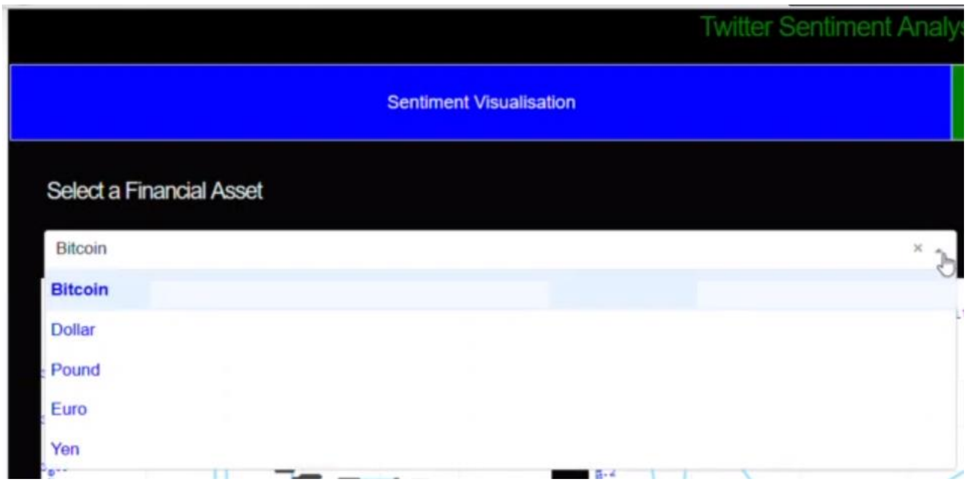
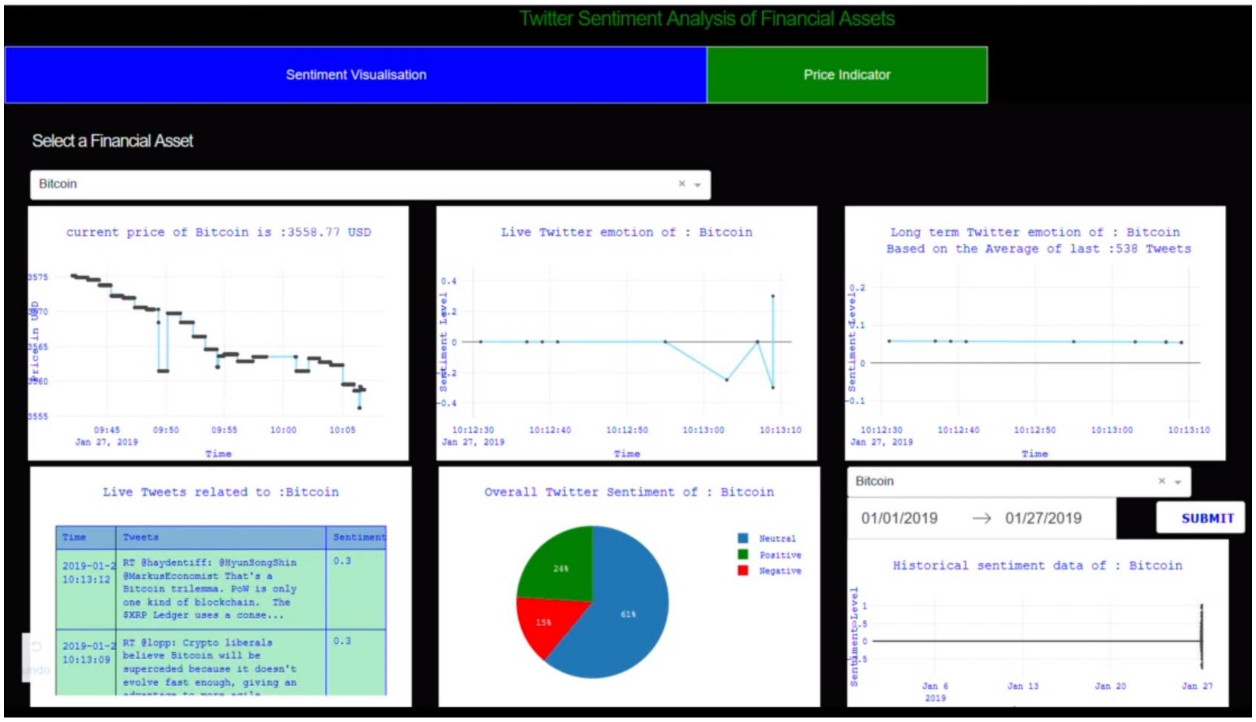


*Figure 58. The dropdown menu of another similar application*


Twitter Sentiment Analysis of Financial Assets
Sentiment Visualisation
Price Indicator
Select a Financial Asset
Bitcoin
current price of Bitcoin is :3558.77 USD
Live Twitter emotion of : Bitcoin
Long term Twitter emotion of : Bitcoin Based on the Average of last :538 Tweets
Live Tweets related to :Bitcoin
Overall Twitter Sentiment of : Bitcoin
Neutral
Positive
Negative
61%
24%
15%
Bitcoin
01/01/2019 → 01/27/2019
SUBMIT
Historical sentiment data of : Bitcoin
Time
Sentiment Level

| Time | Tweets | Sentiment |
|---|---|---|
| 2019-01-2 10:13:12 | RT @haydentiff: @HyunSongShin @MarkusEconomist That's a Bitcoin trilemma. PoW is only one kind of blockchain. The $XRP Ledger uses a conse... | 0.3 |
| 2019-01-2 10:13:09 | RT @lopp: Crypto liberals believe Bitcoin will be superceded because it doesn't evolve fast enough, giving an | 0.3 |

*Figure 59. The complete dashboard of the similar application*

While reflecting on my learning experience throughout this project, it has been very interesting from start to finish. I was exposed to fascinating areas of computer science, such as NLP and data science that I will definitely continue to explore and educate myself on and possibly pursue a career in. Since the field of data science and NLP was completely new to me, a lot of background knowledge had to be acquired. Python quickly became my favorite programming language I have ever learned. The readability and coding style are something very appealing about this language in addition to the many libraries that have encapsulated complex concepts in simple methods that even novice programmers can pick up with relative ease. Attending the Python and Data Science Applications module, during the autumn term, has helped me gain a significant understanding and profound inclination and interest for data science. The Data Mining module has also been interesting and facilitated my learning in machine learning and data mining in general.

This has been the most important project I have been the sole developer of so far. It has provided an invaluable opportunity to comprehend the work that goes into much larger projects and the diverse set of skills, technical and not, that are needed for such tasks. It has practically helped me put the theory behind the software development life cycle, that I had not used until this project, into effect. Review of existing systems and literature, defining a technical specification and a solution approach before implementing the solution are practices that I will continue to use in the future. In terms of the Capability Maturity Model, which describes the different levels of formality and optimization of processes, I believe I have advanced from the initial level of being an ad-hoc engineer to establishing formally defined processes in place.

Overall, I am satisfied with the result of my project. If I were to do this project again, I would give the hybrid sentiment analysis approach more emphasis. It cancels out most of the drawbacks in each of the main two approaches to sentiment analysis. Hybrid approaches exploit machine learning and elements from lexicon-based approach, giving it the potential to improve the sentiment classification performance. Additionally, in hindsight, a different GUI toolkit could have been used. The current GUI looks quite bland to me. Had I used PyQt, it might have made the GUI look more attractive to the end-user.

## 8 Social, Legal, Health & Safety and Ethical Issues

The finished version of this project faces no Social, Legal, Health, Safety or Ethical issues. The solution attempts to make a prediction on user input keywords based on a predefined set of rules and words. The results should not be taken at face value. As also highlighted in the PID, access to Twitter is available to the public and Tweets are immediately viewable and searchable by anyone around the world. By agreeing to the Twitter User Agreement (Terms of Service, Privacy Policy, the Twitter Rules), the users also agree that their data can be viewed and used by anyone with or without access to the Twitter API. All material used have been appropriately referenced.

## 9 Conclusion and Future Improvements

The main aim of this module was to expose myself to different aspects of Data Science and develop an application that can be use in a real-world scenario. This project's main objective was to deliver a system that provides real-time analytics on Tweets. Visualizations of these analytics were provided. An interface was developed to ease the end-user into the system.

There have been a few small parts of the system that could have been approached differently and thus improve the overall aesthetic and functionality of the system, but I am, for the most part, very satisfied with the outcome.

In the future, I would like to work on the areas that I have had issues addressing, as discussed in section 7. I think that a hybrid approach to sentiment analysis can provide better results, since it neutralizes most of the machine learning and lexicon-based approaches' drawbacks. I would also like to try to make a more vibrant and interesting GUI using the PyQt toolkit. Lastly, I would develop the application using Premium Twitter API since that would alleviate most of the limitations that this project has had. By using the Premium API, it would also make the product something that can be used in a more commercial way.

All in all, I believe that this project has been a success and something rather useful has come out of it. For one thing, it has prepared me for solving real-world problems and developing real-world systems. In addition to that, this solution can be used in the real world by people with very real use cases for it.

## 10 References

[1] Why and How Companies Should Use Sentiment Analysis. (Online). Available at:

https://www.northeastern.edu/levelblog/2018/08/02/companies-use-sentiment-analysis/ (Accessed 27 Apr 2019)

[2] Countries with the most Twitter users 2019 | Statistic. (Online). Available at:

https://www.statista.com/statistics/242606/number-of-active-twitter-users-in-selected-countries/ (Accessed 27 Apr 2019)

[3] How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. (Online). Available at:

https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read (Accessed 27 Apr 2019)

[4] Standard Search API – Twitter developers. (Online). Available at:

https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html (Accessed 27 Apr 2019)

[5] (PDF) Sentiment Analysis : A Literature Survey. (Online). Available at:

https://www.researchgate.net/publication/236203597_Sentiment_Analysis_A_Literature_Survey (Accessed 27 Apr 2019)

[6] (PDF) Approaches, Tools and Applications for Sentiment Analysis Implementation. (Online). Available at:

https://www.researchgate.net/publication/283201292_Approaches_Tools_and_Applications_for_Sentiment_Analysis_Implementation (Accessed 27 Apr 2019)

[7] Sentiment analysis of Twitter data: Case study on digital India. (Online). Available at:

https://ieeexplore.ieee.org/document/7857607 (Accessed 27 Apr 2019)

[8] Twitter sentiment analysis for product review using lexicon method. (Online). Available at:

https://ieeexplore.ieee.org/document/8073512 (Accessed 27 Apr 2019)

[9] Graphical User Interfaces with Tk. (Online). Available at:

https://docs.python.org/3/library/tk.html (Accessed 27 Apr 2019)

[10] What is PyQt? (Online). Available at:

https://www.riverbankcomputing.com/software/pyqt/intro (Accessed 27 Apr 2019)

[11] Sentiment Analysis: nearly everything you need to know. (Online). Available at:

https://monkeylearn.com/sentiment-analysis/ (Accessed 27 Apr 2019)

[12] Tweet Natural Language Processing. (Online). Available at:

http://www.ark.cs.cmu.edu/TweetNLP (Accessed 27 Apr 2019)

[13] Giving you more characters to express yourself. (Online). Available at:

https://blog.twitter.com/official/en_us/topics/product/2017/Giving-you-more-characters-to-express-yourself.html (Accessed 27 Apr 2019)

[14] Tweet updates – Twitter Developers. (Online). Available at:

https://developer.twitter.com/en/docs/tweets/tweet-updates.html (Accessed 27 Apr 2019)

[15] Verification and Validation. (Online). Available at:

https://www.sqa.org.uk/e-learning/SDPL03CD/page_16.htm (Accessed 27 Apr 2019)

[16] IEEE Standard for Software Reviews. (Online). Available at:

https://ieeexplore.ieee.org/document/663254/definitions (Accessed 27 Apr 2019)

[17] Principles of Imperative Computation. (Online). Available at:

http://www.cs.cmu.edu/~rjsimmon/15122-s13/rec/07.pdf (Accessed 27 Apr 2019)

# 11 Appendices

## 11.1 Project Initiation Document (PID)

**Individual Project (CS3IP16)**

**Department of Computer Science**
**University of Reading**

**Project Initiation Document**

**PID Sign-Off**

| | |
|---|---|
| **Student No.** | **25015646** |
| **Student Name** | **Yiannis Hadjicharalambous** |
| **Email** | **y.hadjicharalambous@student.reading.ac.uk** |
| **Degree programme** (BSc CS/BSc IT) | **BSc CS** |
| | |
| **Supervisor Name** | **Frederic Stahl** |
| **Supervisor Signature** | Fred Stahl |
| **Date** | **05/10/2018** |

## SECTION 1 – General Information

### Project Identification

| | |
|---|---|
| **1.1** | **Project ID**<br>(as in handbook) |
| | 160 |
| **1.2** | **Project Title** |
| | Data Science, Analytics and Mining |
| **1.3** | **Briefly describe the main purpose of the project in no more than 25 words** |
| | A system that ingests microblogging data such as Twitter in real time and predicts a sentiment score for the individual tweets. |

### Student Identification

| | |
|---|---|
| **1.4** | **Student Name(s), Course, Email address(s)**<br>e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk |
| | Yiannis Hadjicharalambous, BSc CS, y.hadjicharalambous@student.reading.ac.uk |

### Supervisor Identification

| | |
|---|---|
| **1.5** | **Primary Supervisor Name, Email address**<br>e.g. Prof Anne Other, a.other@reading.ac.uk |
| | Frederic Stahl, f.t.stahl@reading.ac.uk |
| **1.6** | **Secondary Supervisor Name, Email address**<br>Only fill in this section if a secondary supervisor has been assigned to your project |
| | |

### Company Partner (only complete if there is a company involved)

| | |
|---|---|
| **1.7** | **Company Name** |
| | |
| **1.8** | **Company Address** |
| | |
| **1.9** | **Name, email and phone number of Company Supervisor or Primary Contact** |

| | |
|---|---|
| | |

## SECTION 2 – Project Description

| | |
|---|---|
| **2.1** | **Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.** |
| | Automation is something every computer scientist strives for. Being able to automate understanding human sentiment off a status can be very useful for many projects.<br><br>Developer access to the Twitter API keys and tokens as well as an in-depth knowledge of the API documentation.<br><br>Thorough knowledge of the Python language, its different styles, its techniques, the documentation and many of its libraries. The main one that will be used is tweepy, a Python library for accessing the Twitter API. Other libraries that will be useful are numpy (scientific computing), matplotlib (plotting), pandas (data manipulation and analysis), textblob (natural language processing such as sentiment analysis) and tflearn (deep learning).<br><br>A very good grasp of machine/deep learning that might be used to improve the system in order to make a better prediction and understand concepts such as sarcasm.<br><br>https://docs.python.org/3.6/<br>https://developer.twitter.com/en/docs<br>https://tweepy.readthedocs.io/en/v3.6.0/<br>https://docs.scipy.org/doc/numpy/<br>https://matplotlib.org/api/index.html<br>http://pandas.pydata.org/pandas-docs/stable/<br>https://textblob.readthedocs.io/en/dev/<br>http://tflearn.org/ |
| **2.2** | **Summarise the project objectives and outputs in about 400 words.** These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other. |

Objectives:

- Getting familiar with the libraries and the language
- Create an outline of the high-level side of the system
- Create a first draft of the system
- Build an analytics workflow
- Sentiment analysis
- Test: unit, integration, acceptance, system
- Finally adapt a model to run on real-time Twitter feeds including user feedback to facilitate adaptation to changes

Outputs:

- Ability to code in chosen language and initiate development of my system
- Have a general view of how the system will operate and from that be able to know how to develop it
- By having a first draft of the system, instead of a high-level view, I will be able to get feedback that will help me significantly improve my system
- Having an analytics workflow added to the system will massively improve the developing process
- Sentiment analysis is the last part of the system development
- By testing, I make sure that any test cases are covered and no unexpected outcomes can occur
- Have a final robust version of the system that will allow me to begin writing up the documentation

**2.3** **Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.

Key features include, but might not limited to, authenticate access to the Twitter API, grab arbitrary Tweets based on a particular subject/keyword, analyse and manipulate those Tweets, give a sentiment score to the Tweet, make use of machine learning and a big dataset or user feedback to improve results, produce visualized data in graphs or plots.

**2.4** **Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval? (If your project requires a questionnaire/interview for conducting research and/or collecting data, you will need to apply for an ethical approval)**

Twitter is public and Tweets are immediately viewable and searchable by anyone around the world. By agreeing to the terms of service of Twitter, the users also agree that their data can be viewed and used by anyone with or without access to the Twitter API.

| | |
|---|---|
| **2.5** | **Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost |
| | No items will be purchased. |
| **2.6** | **State whether you need access to specific resources within the department or the University e.g. special devices and workshop** |
| | I will not need access to specific resources. |

## SECTION 3 – Project Plan

| | |
|---|---|
| **3.1** | **Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. **Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report**. |

| **Task No.** | **Task description** | **Effort (weeks)** | **Outputs** |
|---|---|---|---|
| **1** | | | |
| 1.1 | **Background Research** | | … |
| 1.2 | Getting familiar with the libraries and the language | 3 | Ability to code in chosen language and initiate development of my system |
| | | | |
| **2** | **Analysis and design** | | |
| 2.1 | Create an outline of the technical side of the system | 2 | Have a general view of how the system will operate and from that be able to know how to develop it |
| | | | … |
| | | | |
| **3** | **Develop prototype** | | |
| 3.1 | Create a first draft of the system | 4 | By having a first draft of the system, instead of a high-level view, I will be able to get feedback that will help |

| | | | |
|---|---|---|---|
| | | | me significantly improve my system |
| 3.2 | Build an analytics workflow | 3 | Having an analytics workflow added to the system will massively improve the developing process |
| 3.3 | Sentiment analysis | 3 | Sentiment analysis is the last part of the system development |
| **4** | **Testing, evaluation/validation** | | |
| 4.1 | unit, integration, acceptance, system testing | 4 | By testing, I make sure that any test cases are covered and no unexpected outcomes can occur |
| 4.2 | Adapt a model to run on real-time Twitter feeds including user feedback to facilitate adaptation to changes | 2 | Have a final robust version of the system that will allow me to begin writing up the documentation |
| | | | … |
| **5** | **Assessments** | | |
| 5.1 | Write-up project report & log-book | 6 | Project Report |
| 5.2 | Produce poster | 1 | Poster |
| 5.3 | Work on project demonstration | 2 | |
| 5.4 | Feedback form | 1 | |
| | | | |
| | | | |
| | | | |
| **TOTAL** | **Sum of total effort in weeks** | **31** | |

**SECTION 4 - Time Plan for the proposed Project work**

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the cursor becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

| | **11.1.1.1.2 START DATE: 01/10/2018** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **11.1.1.1.3 Project Weeks** | | | | | | | | | | | | |
| **11.1.1.1.1 Project stage** | 0-3 | 3-5 | 5-9 | 9-12 | 12-15 | 15-19 | 19-21 | 21-24 | 24-27 | 27-30 | 30-33 | 33-36 | 36-39 |
| **1 Background Research** | ▨ | | | | | | | | | | | | |
| Getting familiar with the libraries and the language | | | | | | | | | | | | | |
| **2 Analysis/Design** | | | | | | | | | | | | | |
| Create an outline of the technical side of the system | | ▨ | | | | | | | | | | | |
| **3 Develop prototype.** | | | | | | | | | | | | | |
| First draft | | | ▨ | | | | | | | | | | |
| Analytics workflow | | | | ▨ | | | | | | | | | |
| Sentiment analysis | | | | | ▨ | | | | | | | | |
| **4 Testing, evaluation/validation** | | | | | | | | | | | | | |
| unit, integration, acceptance, system testing | | | | | | ▨ | | | | | | | |
| Adapt a final version of the model | | | | | | | ▨ | | | | | | |
| **5 Assessments** | | | | | | | | | | | | | |
| Project report and log book | | | | | | | | ▨ | ▨ | | | | |
| Poster & demonstration | | | | | | ▨ 1 week | | ▨ 2 weeks | | | | | |
| Feedback form | | | | ▨ 1 week | | | | | | | | | |

## RISK ASSESSMENT FORM

| | | | |
|---|---|---|---|
| **Assessment Reference No.** | | **Area or activity assessed:** | |
| **Assessment date** | | | |
| **Persons who may be affected by the activity (i.e. are at risk)** | | | |

**SECTION 1: Identify Hazards -** *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1.** | Fall of person (from work at height) | | **6.** | Lighting levels | | **11.** | Use of portable tools / equipment | | **16.** | Vehicles / driving at work | | **21.** | Hazardous fumes, chemicals, dust | | **26.** | Occupational stress | |
| **2.** | Fall of objects | | **7.** | Heating & ventilation | | **12.** | Fixed machinery or lifting equipment | | **17.** | Outdoor work / extreme weather | | **22.** | Hazardous biological agent | | **27.** | Violence to staff / verbal assault | |
| **3.** | Slips, Trips & Housekeeping | | **8.** | Layout , storage, space, obstructions | | **13.** | Pressure vessels | | **18.** | Fieldtrips / field work | | **23.** | Confined space / asphyxiation risk | | **28.** | Work with animals | |
| **4.** | Manual handling operations | | **9.** | Welfare facilities | | **14.** | Noise or Vibration | | **19.** | Radiation sources | | **24.** | Condition of Buildings & glazing | | **29.** | Lone working / work out of hours | |
| **5.** | Display screen equipment | | **10.** | Electrical Equipment | | **15.** | Fire hazards & flammable material | | **20.** | Work with lasers | | **25.** | Food preparation | | **30.** | Other(s) - specify | |

**SECTION 2: Risk Controls** - *For each hazard identified in Section 1, complete Section 2.*

| **Hazard No.** | Hazard Description | Existing controls to reduce risk | **Risk Level** (tick one) | | | Further action needed to reduce risks *(provide timescales and initials of person responsible)* |
|---|---|---|---|---|---|---|
| | | | High | Med | Low | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

| | | |
|---|---|---|
| **Name of Assessor(s)** | | **SIGNED** |
| **Review date** | | |

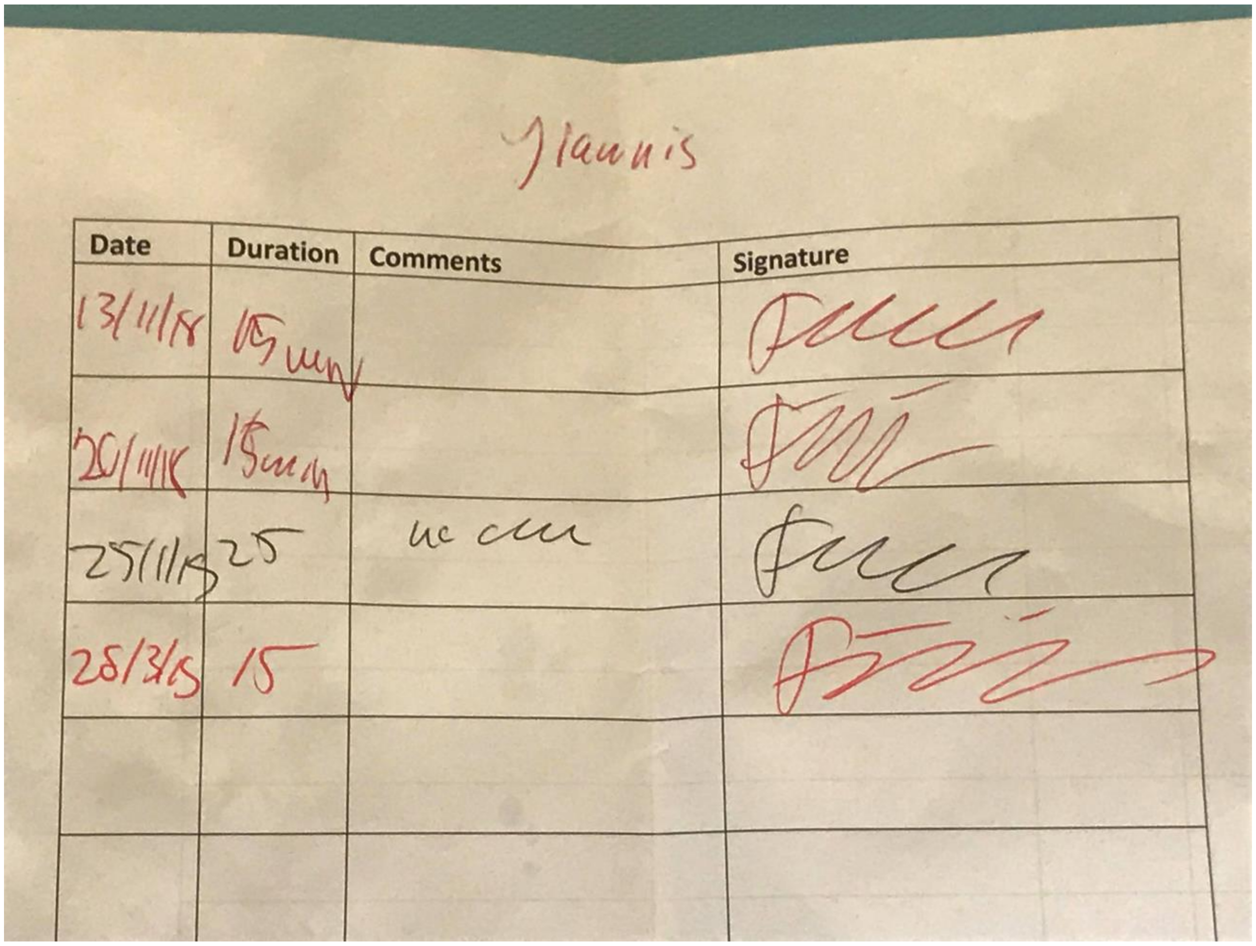
**Health and Safety Risk Assessments** – continuation sheet

| | |
|---|---|
| **Assessment Reference No** | |
| **Continuation sheet number:** | |

**SECTION 2 continued: Risk Controls**

| **Hazard No.** | Hazard Description | Existing controls to reduce risk | **Risk Level** (tick one) | | | Further action needed to reduce risks *(provide timescales and initials of person responsible for action)* |
|---|---|---|---|---|---|---|
| | | | High | Med | Low | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| **Name of Assessor(s)** | | | **SIGNED** | | | |
| **Review date** | | | | | | |

## 11.2 FYP meetings Sign-Off



Yiannis

| Date | Duration | Comments | Signature |
|---|---|---|---|
| 13/11/18 | 15 min | | [illegible] |
| 20/11/18 | 15 min | | [illegible] |
| 25/11/18 | 25 | [illegible] | [illegible] |
| 28/3/19 | 15 | | [illegible] |
| | | | |
| | | | |

## 11.3 Source Code

https://csgitlab.reading.ac.uk/bx015646/final-year-project/

## 11.4 Logbook

Logbook:

### Autumn Term

Week 1 (1st October 2018 – 7th October 2018):

- Learning the Python language
- Applying for developer access to Twitter
- Reading through the Twitter documentation

Week 2:

- Continue learning Python
- Find specific libraries that might be useful for the project
- Work on PID

Week 3:

- Continue learning Python
- Read through the documentation of libraries found, more specifically: tweepy, textblob and more widely used libraries like numpy, pandas and matplotlib.
- Do research on sentiment analysis and possibilities of using machine learning

Week 4:

- Still improving and working on my Python knowledge
- Develop a prototype of the program
- Present it to Professor Stahl to get feedback and work on that feedback

Week 5:

- Due to a lot of deadlines, very little work was done this week, mostly doing research on machine learning.

Week 6 (Reading week):

- Little work was done, mostly rereading through textblob's documentation for the classifiers.

Week 7:

- Decided to gather historical tweets, which isn't possible with Tweepy since it uses the Standard API – which can only search for Tweets up to 7 days old. Familiarised myself with Python libraries which use the Premium API, such as TwitterAPI and SearchTweets.

Week 8:

- After our weekly discussion with Prof. Stahl, it was decided to gather Tweets as they come for some days, i.e. stream and write them to a file, and then do sentiment analysis on that dataset.

- Collected Tweets through 4 days for several hours each day and worked on visually representing their sentiment.

Weeks 9-10:

- Focused on working on assignments that are due so little to no work was done.
- Looked at some examples of interfaces to get ideas for mine.

Week 11:

- Started looking at ways to make an end-user interface.

Christmas break:

- Collected more tweets to use for offline analytics.

## Spring Term

Week 1 (14th January 2019 – 20th January 2019):

- Decided to use TkInter for making a GUI and started familiarizing myself with the library.

Week 2:

- Created a first draft of the interface, to be improved with feedback from the professor.

Week 3:

- Improved the interface and added more features and buttons.

Week 4:

- Improved and optimized different parts of my code.

Week 5:

- Discussed with Prof. Stahl ways that my application can be extended and improved.

Week 6 (Reading week):

- No work was done this week due to assignments that were due.

Week 7:

- Added a live graph feature as well as a picture of the case study on the interface.

Week 8:

- Started doing research on the project report.

Week 9:

- Now have a general plan on how to lay my report and started with the write up.

Week 10:

- Continue writing up the report’s first sections and planning out future sections

Week 11:

- Met up with Prof. Stahl and talked about the project, presentation and the report.
- Continued with the write-up.

## Summer Term (1st April 2019 – 29th April 2019) :

- Wrote the report.
- Prepared for the presentation & demonstration.