
# 6CCS3PRJ Final Year
# Automated Analysis of Comments in Online Video Platforms

Final Project Report

Author: Daniel Van Cuylenburg

Supervisor: Dr Tommy Thompson

Student ID: k19012373

April 29, 2022

**Abstract**

YouTube had over 2 billion active users in 2021. With each video posted on the platform, users are able to post comments about their opinions, suggestions, critiques, and so on that the creators of these videos are able to read, highlight and moderate. Unfortunately, these creators do not have an effective way of analysing their audiences response to their videos without having to scroll through hundreds if not thousands of comments manually.

This project aims to use a range of computational techniques to analyse the sentiments, viewpoints, and most discussed topics of any YouTube video's audience by processing that video's dataset of comments through a variety of natural language processing and machine learning methods. The project then aims to display this processed data and statistics to the user through a GUI.

**Originality Avowal**

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Daniel Van Cuylenburg

April 29, 2022

**Acknowledgements**

I would like to take this opportunity to thank my supervisor, Dr Tommy Thompson, for his invaluable feedback and support across the project.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation, Aims, & Objectives

In recent history, the widespread use of the internet has led to the creation of a variety of social media platforms that allow users to post content in form of text, pictures, videos, and all of them combined. The largest social media platforms often allow users to post comments under each other's posts, and even under each other's comments.

In 2021, YouTube, one of the biggest social media platforms, has more than 2 billion active users [4, 5], meaning that more than a quarter of the world's population uses YouTube every month. With each video posted on the platform, users are able to post comments and sub-comments about their opinions, suggestions, critiques, and so on that the creators of these videos are able to read, highlight and moderate.

For some of the larger content creators on YouTube, their videos may harbour thousands of posted comments in even just the first 48 hours of posting the video, making it highly impractical for these content creators to read each and every comment. Content creators who may want to feel the general response of their audiences to their latest videos may find it excruciatingly boring to have to sit and scroll through repetitive comments for hours. Additionally, creators who want statistics specifically about what their audiences have been commenting on, or who may want to find reoccurring themes within their fanbase's comments, are also unable to do so. These creators lack an application that is able to automatically process comments from their videos through a range of techniques that analyse natural language.

Natural Language Processing (NLP) refers to a range of computational techniques used for the analysis of text written in a natural language [1]. Sentiment Analysis (SA) refers to the

identification and classification of a viewpoint or opinion expressed in a piece of text using NLP methods [6].

One of the major aims of this study is to perform NLP and SA on a range of comments of a given YouTube video, input by the user, and to return to the user a report of the processed comments along with helpful statistics and charts about the comments. This is done to give content creators a greater insight into their userbase, and more specifically, insight into what a channel's fans might think of a specific video. This would also make it easier for creators to see what the best and worst parts of the video were, as well as common suggestions, making it easier for creators to improve their work.

More specifically, this study aims to present to the user each comment's sentiment, meaning if the comment is positive, negative, or neutral; each comment's valence, which is the sentiment of the comment expressed as a numerical value; the percentage of comments that are positive or negative overall; frequency and importance of specific words across all the comments; and a range of other similar statistics.

The YouTube suggestions algorithm is a very powerful machine learning algorithm and can sometimes suggest unexpected, seemingly random recommendations to its consumers. A video that has been uploaded to YouTube for some number of months, or even years, could suddenly be taken up by the YouTube algorithm and start being suggested to certain users, sometimes even drastically increasing the view count of said video. Another possible reason for an influx in the views of a video could be that the video was referenced in some form of media, like a tweet or a discord server message, resulting in a drastic increase in viewership.

Another major aim of this study is to give content creators a way to see what commenters from the last 24 hours, 7 days and 30 days think about their video, as well as the differences between the sentiments and views of commenters across the whole lifespan of the video in contrast with the last 24 hours, 7 days and 30 days. This is done to help content creators who have had their videos recently drastically increase in viewership for some seemingly unknown reason figure out why that has happened based on new comments posted on their videos. This would also help the creators understand if the commenters bought in by the influx in viewership, likely brought in by a recommendation somewhere, enjoyed their video more or less than the early commenters that have posted on the video, who are likely to be fans of the channel.

## 1.2 Report Structure

This chapter has given an introduction to the project and outlined the aims and objectives. Chapter 2 discusses the techniques available for completing these aims and objectives, as well as a description of how these techniques have been previously applied in existing work. Chapter 4 discusses the design and implementation of the project. Chapter 5 discusses the videos used to test the application, and if all of the requirements have been met. Chapter 6 discusses the legal, social, ethical and professional issues faced during the implementation of this project. Chapter 7 discusses the project overall and talks about further improvements that could be made to the project.

# Chapter 2

# Background & Literature Review

This chapter will describe the techniques available to complete my aims and objectives. This includes NLP, the levels of NLP, SA, the levels of SA, and the types of SA. A literature review is then conducted, describing where and how these techniques have been previously applied in relation to analysing social media comments.

## 2.1 Natural Language Processing

NLP boils down to two distinct focuses: language processing and language generation [1, 7, 8]. Language processing refers to the analysis of a language so as to produce a meaningful representation of the language and can be seen as the skill of reading. Language generation refers to the production of language from a representation and can be seen as the skill of writing. This project focuses on the task of language processing.

Generally, NLP refers to a range of computational techniques used for the analysis of text written in a natural language at one or more levels of linguistic analysis for the purpose of achieving human-like language processing to tackle a range of tasks [1]. In other words, these computational techniques are used to extract grammatical structure and meaning from some text in order to perform some useful task(s) [8]. A natural language is a language that is used for everyday communication by humans, for example, English, Russian, and so on. NLP is considered to be a discipline within the field of artificial intelligence and machine learning.

It has been suggested that the goal of NLP is to allow computers to understand natural language, but that this has not been fully accomplished [1, 9]. A fully accomplished system would be able to:

1. Paraphrase an input text
2. Translate the text into another language
3. Answer questions about the contents of the text
4. Draw inferences from the text

While over the last several decades, NLP has made some serious progress on the first three points, NLP systems still cannot accurately draw inferences from text.

Due to the increasing availability of computers with large volumes of memory and high-speed processing capabilities, and the large amounts of text available via the internet over recent years, NLP researchers have been able to develop systems that can deal reasonably well with general and specialised texts. These systems are able to account for a much larger portion of the ambiguity and variability of a language compared to previous years. Some of these systems have been described in the existing research section 2.6.

NLP techniques can be useful in a variety of fields, such as tutoring and automatic marking systems, duplication and spam detection/filtration, search engines/results, language translation, data analysis, and so on.

## 2.2 Levels of Natural Language Processing

One way of presenting what actually happens during NLP is to use a "levels of language" approach [1, 7, 8, 10]. The following description of the various levels of NLP will be presented sequentially, however, it is important to note that actual NLP is much more dynamic, meaning that the levels can interact in a variety of orders. Humans typically use the information we gain from higher levels of processing to assist in lower levels of processing. For example, someone may use the fact that a certain document they are reading is about mathematics, and as a result, interpret a word with multiple possible meanings as its mathematical meaning.

### 2.2.1 Phonology

Phonology, in relation to linguistics, is the systematic arrangement of sound. Speech sounds within words, variations of pronunciation when words are spoken together, and fluctuations in intonation and stress are all dealt with at the phonological level. Phonological NLP systems deal with speech do so by encoding the sound waves into digitized signals, whereafter these

signals are interpreted through the use of various rules, or otherwise by comparing to a pre-built language model.

### 2.2.2 Morphology

At the morphological level, words are broken down into morphemes, which are units of a language that cannot be broken down further that carry some meaning. For example, the word "paralegals" can be broken down into the morphemes "para-", "legal", and "-s". The meaning of each morpheme stays the same across all words, and as a result, NLP systems can perceive and represent the meaning of each morpheme. An instance of this is the use of the "-ed" morpheme, which usually conveys that the action of a verb occurred in the past tense [1].

### 2.2.3 Lexical

At the lexical level, NLP systems work with and are able to interpret the meaning of individual words. In this phase, data can be processed in a variety of ways, including tokenization, stemming, and lemmatization.

Tokenization can be performed on a piece of text to separate the text into tokens, each of which represents an instance of a term. The sentence "the driver is racing in his father's car" can be broken down into the list of tokens ["the", "driver", "is", "racing", "in", "his", "father's", "car"].

Stemming can be used to remove the suffix of a word to get its root form, also known as the 'stem' of a word. For example, if I decided to apply this technique to the sentence "the driver is racing in his father's car", one possible result could be "the driv is rac in hi farther's car".

Lemmatization is used to reduce a word to its dictionary form. The lemma of a word contains all the metadata of a word as a dictionary entry; this includes the definition of the word and the stems and affixes of the word.

Parts-of-speech (POS) tagging is the process of assigning labels to each token in a piece of text that indicate which category of words the token belongs to using the token's definition and context in the sentence. Traditionally, the English language contains eight parts-of-speech that are most commonly used [11]:

1. Noun
2. Verb
3. Adjective

4. Preposition

5. Pronoun

6. Adverb

7. Conjunction

8. Interjunction

It is important to note that modern NLP systems categorise tokens into a much larger variety of categories. For example, some systems may want to capture the tense of a verb in the POS tag, and as a result, have multiple verb POS tags for different tenses.

POS tagging is a fundamental building block of many NLP systems [12]. POS tagging can be used for a variety of NLP methods, such as word sense disambiguation (described in section 2.2.5), SA (described in section 2.3), and so on. Adjectives are important indicators of subjectivities and opinions, so identifying them can be an important first step in performing SA [13].

Additionally, words that can only have one possible meaning can be replaced by semantic representations of that meaning. The exact representation format is determined by the NLP system in use; one example of a representation can be constructed using logical predicates. Using the tokenized word "launch" as an example, figure 2.1 is a representation of the word decomposed into its more basic properties.

```
launch (a large boat used for carrying people on rivers, lakes harbors, etc.)
((CLASS BOAT) (PROPERTIES (LARGE)
(PURPOSE (PREDICATION (CLASS CARRY) (OBJECT PEOPLE))))
```

Figure 2.1: Representation of the meaning of the word "launch" [1]

Simplified lexical representations like the one in figure 2.1 allow NLP systems to associate meaning with words.

A lexicon is a large collection of lexical representations. The lexical representations of these words can either be complex, like the one in figure 2.1, or much simpler, for example, each word in a lexicon could be simply tagged as either positive or negative. Some NLP systems may choose to use lexicons to determine the characteristics of words, for parsing sentences, and to recognise words and phrases. Various lexicons have been described in section 2.5.4.

### 2.2.4 Syntactic

The syntactic level focuses on analysing the positions of words in a sentence to uncover the grammatical structure of the sentence. This can be described as analysing the tokens in a body of text conforming to the rules of formal grammar. Grammatical rules are applied to categories and groups of words, rather than individual words. The output of this level is a representation of the text that reveals the structural dependency relationships between the words. For example, the two sentences “Sophie slapped Jack” and “Jack slapped Sophie” are only syntactically different, but each holds a different meaning – clearly, the order and dependency of the words contributes to the meaning of both sentences.

At this level, many different types of relationships can be identified and characterized within sentences, such as an ”agent”, a ”possessor of”, or an ”is a” relationship [10].

### 2.2.5 Semantic

At the semantic level, the possible meanings of a sentence are determined based on the interactions among individual word-level meanings in the sentence, as well as the structure of the sentence. During the semantic analysis, semantic disambiguation is performed on words with multiple potential meanings whose meanings could not be determined at the lexical level and only allows for one and only one meaning of a word to be represented semantically.

An example of semantic disambiguation, otherwise known as word sense disambiguation, can be seen in the following two sentences [14]:

1. This device is used to jam the signal.

2. I am stuck in a traffic jam.

Performing semantic disambiguation on both of these sentences with respect to the word ‘jam’ would return the following:

1. The device is used to (deliberate radiation or reflection of electromagnetic energy for the purpose of disrupting enemy use of electronic devices or systems) the signal.

2. I am stuck in a traffic (stuck and immobilized).

### 2.2.6 Discourse

The discourse level works with text that is longer than a sentence. In other words, text that is comprised of multiple sentences is not just interpreted as a concatenation of those sentences, but

rather meaning is conveyed through connections between component sentences that describe the properties of the text as a whole.

Different techniques are employed at this level, one example being anaphora resolution. This technique can be described as the process of resolving what a pronoun or noun phrase refers to, usually done by replacing said pronoun or noun with the appropriate entity to which they refer. An example of this would be changing the sentence "Felix arrived, but nobody saw him" to "Felix arrived, but nobody saw Felix".

Another example of a technique used at this level is discourse structure recognition, where the function of a sentence in a text is determined in order to add to the meaningful representation of a text. An example of discourse structure recognition would be the representation of an article as multiple parts such as lead, main story, previous events, evaluation, bibliography, and so on [1].

### 2.2.7 Pragmatic

At the pragmatic level, extra meaning that has been read into texts without actually being encoded in them is interpreted. An NLP system utilizes context over the contents of the text. At this level, an NLP system requires a vast amount of knowledge about the world; this could include knowledge of plans, goals and intentions. An example of this could be the sentence "Do you know what time it is?", which could be someone asking the time, or could be an angry mother yelling at her son to get out of bed. Context is needed to discern the meaning of the sentence.

### 2.2.8 Summary

An NLP system does not have to use all of the levels of language, but since humans have been shown to use all of these levels to gain understanding of natural language, the more levels of language that an NLP system utilizes, the more proficient that NLP systems is. Typically, NLP systems tend to use the lower levels of language, as these levels deal with smaller units of analysis, such as sentences, words and morphemes, which are simpler to deal with compared to whole pieces of text, and world knowledge. These lower levels have also been more thoroughly researched and implemented compared to the higher levels.

Social media comments, and more specifically YouTube comments, tend to harbour some unique features compared to normal bodies of texts. Due to the informal nature of social media's commenting systems, users are usually encouraged to post short, straight-to-the-point

comments that don't usually even span longer than a single sentence. BuzzSumo's analysis of more than 800 million Facebook posts [15] found that posts with less than 50 characters received more user engagement compared to longer posts. As a result, social media posters are conditioned to post shorter comments that are more likely to gain attention.

Due to this, the **discourse level** is not very relevant to analysing most YouTube comments, as this level deals with text comprising of multiple sentences. As well as this, techniques usually applied at the **pragmatic level** may be hard to meaningfully implement due to the lack of a source of objective knowledge about the contents of a YouTube video, and as a result little to no knowledge about the context surrounding a comment. Additionally, the **phonological level** is not relevant to the project, as YouTube comments are obviously not spoken.

On the other hand, the **morphological level** and the **lexical level** are extremely useful when analysing comments, as these levels deal with individual words and the parts of words, and don't rely on a lengthy text. Levels that focus on the positions and interactions between words within sentences, such as the **syntactic** and **semantic levels**, can also be useful when analysing comments, but may also fall flat some of the time when a comment comprises of only a single or couple words or is just some combination of emojis.

## 2.3 Sentiment Analysis

Sentiment analysis (SA), also known as opinion mining, refers to the identification and classification of a viewpoint or opinion expressed in a piece of text using information retrieval and computational linguistics [6]. SA is used to identify polarity, meaning positive, neutral, and negative opinions; feeling and emotions, meaning anger, sadness, happiness; urgency; and intentions, such as interest vs disinterest. SA is performed with the help of various techniques, such as NLP, text mining, and other methods [16]. SA is usually performed on data that has some sort of sentiment expressed within it, such as blogs, product/customer reviews, news articles, and so on.

Data must be preprocessed before any NLP or SA can be performed on it. This is known as the process of reducing the noise in the dataset [2]. Pokharel et al. [17] proposed the following four factors that sum up the noise found in social media comments:

1. **Non-standard Language**: Comments are not always written in the English language. They can also contain slang or shortened/improper forms of words. It is difficult to extract meaning from such data.

2. **Spelling errors**: Due to the informal settings in which social media comments are usually posted, many comments contain spelling errors. It is important to either correct or remove such words, as these words would add unnecessary features to the classification model, decreasing the final accuracy of the classifier. For example, ”u” and ”you” convey the same meaning to the informal writer and reader, but if the spelling of the words is ignored, they would be treated as two different words by any classifier.

3. **Unformatted texts**: Some comments contain computer code and other symbols, also decreasing the accuracy of the classifier if not removed.

4. **Trivial comments**: Some comments posted on videos are not at all relevant to the video or channel of the video. A sizable amount of comments are just commenters trying to market their products, or simply show their presence. These comments are not useful to content creators and only add unnecessary overhead when training classifiers.

Usually, the data preprocessing is performed through pipelines, passing the dataset from one processing step to the next. The preprocessing steps incorporated in each study depend on how the data will later be processed. For example, some NLP methods, such as the use of the VADER lexicon, can make use of emojis found in some comments, while other NLP methods cannot. One example of a preprocessing pipeline has been proposed by Sun et al. [2] in figure 2.2.

1. **Removal of hyperlinks and numbers**: Occasionally YouTube comments contain URL links that the commenters have added to provide additional information or context to their comments. An example of this is the following comment:

   *That’s what the word police means! https://dictionary.cambridge.org/dictionary/english/police*

   These links do not themselves provide any additional meaning. Commenters also sometimes use numbers to communicate some statistic or other; these numbers as features do not contain any semantic meaning. Due to this, Sun et al. has decided to remove both URL links and numbers from their datasets.

2. **Abbreviation extending**: otherwise known as replacing contractions with their extended versions, makes it easier to perform further text analysis. An example of this would be to replace the word ”aren’t” with the words ”are not” in a comment.

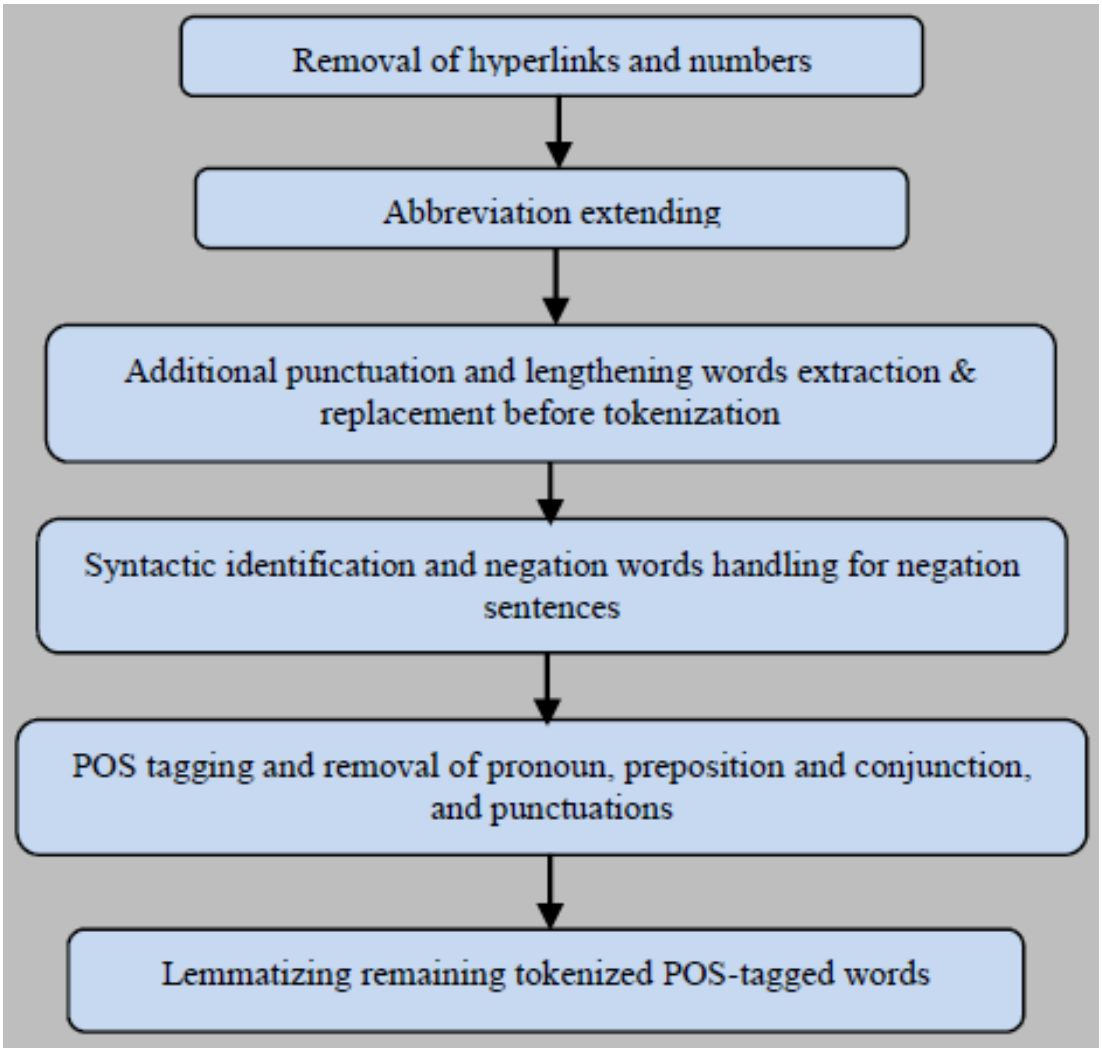



Figure 2.2: An example of a preprocessing pipeline [2]

3. **Additional punctuation and lengthening words extraction & replacement before tokenization**: The paper suggests that online unstructured texts contain additional punctuations that represent some sort of strong sentiment, or otherwise change the sentiment of the comment. It is suggested that rather than completely removing this punctuation, as seems to be common in many NLP systems [8, 17], the punctuation is replaced by suitable corresponding adverbs with a period. An example of this would be replacing the punctuation "!!!!!!" with "extremely.".

4. **Negation identification, handling, POS tagging, removal of pronouns, prepositions and conjunctions, and punctuations**: At these stages, POS tagging, described in section 2.2.3, is performed to assign parts-of-speech tags to each word, identifying how different words interact within a sentence and the role of every word in a sentence.

5. **Lemmatization**: The process of lemmatization has been described in section 2.2.3. The purpose of lemmatization is to reduce the feature dimensions, ensuring that words with different forms, but the same meaning, can be interpreted as the same word.

Another preprocessing technique that has not yet been mentioned is the removal of stop words from the dataset. Stop words are the words that occur the most commonly in a language, and typically carry very little useful information. Some examples of stop words are ”a”, ”of”, and ”for”. Removal of stop words is usually conducted by storing a list of stop words and removing any words in a dataset that appear on the list. Song et al. have concluded that the impact of removing stop words is small, but that this step should still be taken to improve the efficiency of the NLP system/classifier by reducing the size of the feature space [18]. Care has to also be taken when performing this step, as it has been suggested by both Yu et al. and Saif et al. that removing stop words actually reduced the sentiment classification accuracy of their systems [19, 20]. This is because some stop words can be highly discriminative features for some specific classification tasks.

A further example of a preprocessing technique is simply making sure that the dataset comprises all lower case letters. This step is also taken to reduce the feature space, as we wouldn't want the words ”Apple” and ”apple” to be interpreted as different words in our analysis. However, care has to be taken, as the use of upper case letters can change the sentiment of a piece of text; the writer may use upper case letters to express some opinion in a more extreme way than using lower case letters would have permitted.

## 2.4 Levels of Sentiment Analysis

SA can occur at different levels [21]. This could range from broadly analysing the entirety of a piece of text in a document, to more concretely analysing areas such as looking at each sentence individually within the document.

### 2.4.1 Document Level

When performing document level SA, a single piece of text or document written on a single topic is analysed as a whole. During the data preprocessing stages, irrelevant sentences must be identified so that they do not skew the sentiment of the whole document.

This level of SA works best when the document or piece of text is written by a single person, and where an opinion is expressed on a single entity [21]. This may not be suitable for

some types of text, such as product reviews, where the review may attempt to compare similar products with each other.

### 2.4.2 Sentence Level

Sentence level SA is concerned with performing SA on individual sentences. At this level, subjectivity classification is performed on each sentence to determine whether each sentence is opinionated or not opinionated, or otherwise objective or subjective. An objective sentence presents some factual information, while a subjective sentence expresses views, emotions, beliefs and/or personal feelings, and tends to contain opinion words, which help to determine the sentiment of the sentence. Following this, the polarity of each subjective sentence is calculated, meaning that each subjective sentence is classified as positive, neutral, or negative.

Both document level and sentence level SA are generally quite useful but fail to identify what people like and dislike. This is because although these techniques are able to predict the sentiment of whole sentences and documents, they do not analyse the entities within a sentence, so are unable to discern specifically what people like/dislike. Sentence level SA is not very useful for me because each comment usually comprises a sentence or less, meaning that most comments would only have a single sentence analysed.

### 2.4.3 Entity/Aspect/Feature Level

At this level, a finer-grained SA is performed. The goal here is to identify and extract object features/entities that have been commented on with an opinion, and then find out the sentiment of those features/entities. It is at this level that comparative statements are processed.

### 2.4.4 Summary

As stated previously, social media users are conditioned to post relatively short, straight-to-the-point comments usually spanning the length of a sentence or less. **Entity/aspect/feature level** SA can be performed on a set of YouTube comments to identify topics that have been repeatedly mentioned (likely because they were covered in the video) and find the sentiment surrounding these topics based on all of the video's comments as a whole. **Document level** SA can be performed to find the sentiment of each of the individual comments, where the average sentiment across all of the comments can be calculated. This is even more suitable for YouTube comments because virtually all YouTube comments are written by individuals, a lot of the time

expressing an opinion on a single entity. **Sentence level** SA, on the other hand, requires texts comprising of multiple sentences, which is not very common for YouTube comments.

## 2.5 Natural Language Processing & Sentiment Analysis Methods

In this section, various techniques and algorithms that can be used to perform NLP and SA have been discussed. These include count and TF-IDF vectorization, machine learning approaches, and lexicon-based methods.

### 2.5.1 Count vectorization

Count vectorization involves counting the number of occurrences that an n-gram appears in a document and creating a document term matrix, where each column is dedicated to an n-gram in the document, while each row represents a document in the corpus. Corpora are described in section 2.5.3. An n-gram is a sequence of n items, in this case, words, taken from a given piece of text. This matrix can be transposed into a single table, where each row represents an n-gram in the document, with the single column representing how many times the n-gram has occurred in the document.

Count vectorization can be performed on any number of n n-grams. An n-gram with only a single word in it (n = 1) is known as a unigram; a pair of words (n = 2) is known as a bigram; three words (n = 3) are known as a trigram. We can specify if we want to perform count vectorization using unigrams, bigrams, trigrams or any other value of n. If we perform count vectorization using bigrams, the occurrences of each two-consecutive-word pair found in the document will be counted, using the exact same method as described above.

### 2.5.2 Term Frequency–Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that shows the relevance of some sets of keywords across a document [22]. TF-IDF is one of the most recognized algorithms in the field of data mining [23]. The algorithm is not set in stone, and different studies have chosen different ways of calculating this value. Multiple variations of calculating the TF-IDF value are described below. When describing TF-IDF, performing ”TF-IDF vectorization” will mean applying the TF-IDF method to the data.

Term frequency is defined as how many times a certain n-gram is present in a document. One way of calculating this is by taking the total number of occurrences of this n-gram in the document and dividing this by the total number of terms present in the document. For example, if the word "police" appeared 10 times in a document, and the document contained a total of 5000 words, the term frequency would be 10/5000 = 0.002. Other ways of calculating this value exist, including:

1. Just using the total number of occurrences of an n-gram in the document without any division.

2. Using boolean frequencies, meaning using a value of "1" if the word occurs in the document, and "0" otherwise [24].

3. Similarly to above, taking the total number of occurrences of this n-gram in the document and dividing this by the total number of terms present in the document, except not counting duplicate terms, meaning that even if the same word occurs in the document multiple times, it is only counted once in the denominator.

4. Using the log (1 + total number of occurrences of an n-gram) [25]

5. Using 1 + log(total number of occurrences of an n-gram) [24].

Inverse document frequency is defined as how common an n-gram is amongst a group of documents. The inverse document frequency assigns greater weights to infrequent words, and lower weights to frequent words. The closer the value is to 0, the more common that word is. This can be calculated by taking the total amount of documents, dividing this number by the number of documents that contain the n-gram, and calculating the logarithm of this. For example, if we have 10 documents in total, and 5 of those documents contain the word "police", then the inverse document frequency for the word "police" can be calculated as loge(10/5) = 0.3010.

A higher frequency of an n-gram in a set of documents will give a higher term frequency, while the fewer occurrences of an n-gram in a set of documents will yield a higher inverse document frequency. Putting this all together, the TF-IDF is the multiplication of both of these terms. In our example, the TF-IDF value would be 0.002 * 0.3010 = 0.000602.

Applying TF-IDF vectorization to a set of data also yields a document-term matrix, where each column is dedicated to an n-gram in the document, while each row represents a document in the corpus. This can also be transposed into a single table, where each row represents an

n-gram in the document, with the single column representing the importance of the n-gram in the document.

In a similar fashion to count vectorization, TF-IDF vectorization can also be performed on unigrams, bigrams, trigrams and so on. If using bigrams, the importance of each two-consecutive-word pair will be calculated.

### 2.5.3 Machine Learning Approaches

Machine learning can be defined as the scientific field that gives machines the ability to learn without being strictly programmed [26]. The general goal of machine learning is to recognize patterns in data, which inform the way unseen problems are treated [27]. Machine learning techniques are the main tools used for text classification and were first established as useful techniques for performing SA by Pang et al. [28]. Machine learning techniques are very useful for performing SA [16]. This is because SA is usually considered as a binary classification, meaning that a machine learning classifier would be trained to classify data into either the positive or negative categories. Machine learning methods excel in this field.

However, these machine learning techniques use somewhat shallow statistics-based methods that usually perform SA and classify at the document level, not analyzing in detail the object of each expressed sentiment [2].

An example of a problem that can be tackled using machine learning is the detection of credit card fraud; the task being to label each credit card transaction as either “fraud” or “not fraud” [29]. Training the machine learning classifier would involve inputting a collection of historical credit card transactions, each labelled with “fraud” or “not fraud”. One goal here may be to improve the accuracy of the classifier when assigning labels.

For any of these approaches, training datasets are used to learn documents, while testing datasets are used to validate the performance of the method.

Machine learning techniques can be split into two categories: supervised learning, where labelled datasets are designed to ‘supervise’ algorithms into classifying data, and unsupervised learning, where unlabelled datasets are analysed and clustered.

#### Naïve Bayes

Naïve Bayes is a simple but effective supervised learning classification algorithm and is mostly used for document-level classification. Naïve Bayes classifiers assume that there are no dependencies amongst an entity’s features, which is also known as conditional independence [30].

This is obviously not the case for real-life data. Some advantages of this technique include its simpleness, its speed, and its lack of a need for large amounts of data.

### k-Nearest Neighbour

The k-Nearest Neighbour (k-NN) algorithm is a supervised machine learning algorithm that is used for classification and regression. k-NN classifiers rely on category labels attached to the training and testing data. When performing k-NN classification, an object is classified by a plurality vote from its neighbours – the object is assigned to the class that the majority of its neighbours have been assigned to, based on the k value being used. If k = 1, the object is just assigned to the class of its closest neighbour. When performing k-NN regression, an object is assigned a property value, which is the average of the values of the k nearest neighbours. Furthermore, the k-NN algorithm is known as a type of lazy algorithm, where the value of an object is only approximated locally, where all computation is deferred until classification. Usually, the Euclidean or Manhattan distance is used to calculate the distance between two points, but the Chebyshev norm or the Mahalanobis distance can also be used.

### Decision Tree

Decision tree algorithms are supervised learning algorithms that build classification models consisting of simple decision rules. A decision tree model is a flowchart-like tree structure, where each node in the tree represents a test on an attribute, with each edge representing an outcome of the tests [31]. Each leaf node represents a classification label from the dataset. When the classifier needs to classify a new data point, starting from the root node, the data points attributes are tested against each of the nodes, where a path is traced to a leaf node. The label at the leaf node is then assigned to the data point.

### Support Vector Machines

Support Vector Machines (SVM) are very common statistical supervised learning algorithms, considered by some people to be the best text classification method when performing SA [13, 32, 33]. SVMs learn by example to assign labels to objects [34]. These algorithms are also used for classification and regression. The objective of an SVM algorithm would be to find a hyperplane in an N-dimensional space that distinctly classifies the data points, with N being the number of features [3]. In other words, an SVM is an algorithm for maximizing a particular mathematical function with respect to a given collection of data [34].

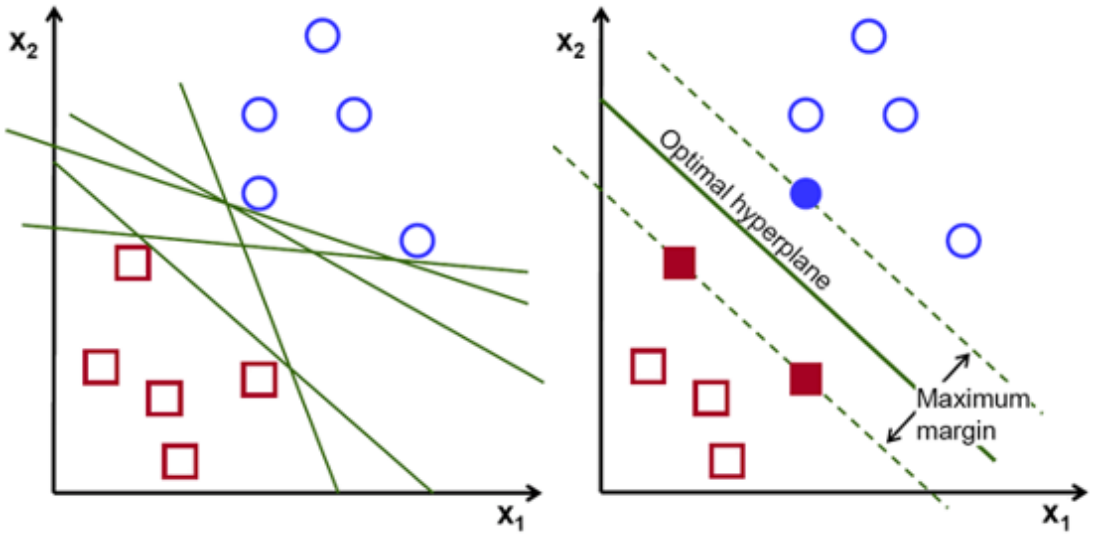



Figure 2.3: Possible hyperplanes [3]

As shown in the left-most graph in figure 2.3, there are many hyperplanes that can be chosen to separate the two classes of data points. Data points that fall on different sides of a hyperplane would be classified into different classes. The goal of an SVM would be to find the hyperplane with the maximum margin, or otherwise the largest distance between both classes of data points. By maximizing the margin distance, future data points are able to be classified with a higher degree of confidence.

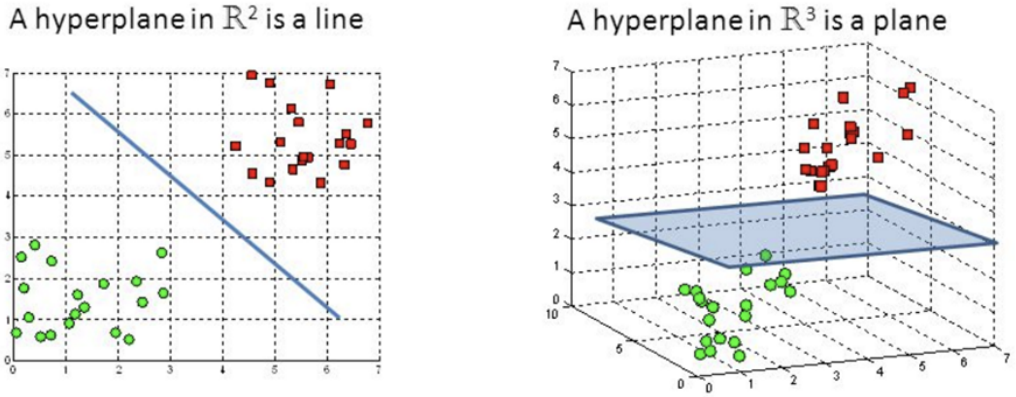



Figure 2.4: Hyperplanes in two-dimensional and three-dimensional feature spaces [3]

If there are two input features, the hyperplane would appear as a straight line, as seen in the left-most graph in figure 2.4, while if there are three input features, the hyperplane would appear as a two-dimensional plane, as seen in the right-most graph.

If no single hyperplane can separate the data into two classes, a kernel function is used to

add an additional dimension to the data. Essentially, the kernel function is a mathematical procedure that allows an SVM to perform a higher-dimensional classification on a set of data that is of a lower dimension [34]. If an appropriate kernel function is used, previously inseparable data will become separable in the resulting higher-dimensional space.

Support vector machines are able to build models where new examples can be assigned to each category. SVM algorithms are known to be effective in high dimensional spaces.

#### Corpora

A corpus is a large set of texts that are used to do statistical analysis. A labelled corpus is a corpus in which each piece of text has been assigned some label/category. Labelled corpora can be used to train machine learning classifiers, such as the naïve Bayes and SVM classifiers described above. In this way, a labelled corpus is fed into a classifier as training data, where the classifier builds a model of the training data. This model can then be used to assign labels to new unlabelled data.

An example of a labelled corpus is the corpus used in Pokharel et al. [17], where YouTube comments have been classified into six categories - positive, negative, interrogative, imperative, corrective, and miscellaneous. Pokaharel et al. build their training dataset by manually going through and labelling a set of YouTube comments that are then fed into various machine learning classifiers.

Two corpora are used in this study, both of which are described in the specification in section 4.5.

### 2.5.4 Lexicon-Based Methods

As described in section 2.2.3, lexicons are collections of words that contain semantic and/or grammatical information about these words. A lexicon-based approach would involve using a sentiment lexicon to score a piece of text by aggregating the sentiment scores of all words or phrases in the document [35].

#### VADER

The Valence Aware Dictionary for Sentiment Reasoning (VADER) is a gold-standard sentiment lexicon that is specifically attuned to microblog-like contexts [36, 37]. The lexicon uses five generalizable rules that embody grammatical and syntactical conventions that are used when expressing sentiment. In microblog-like contexts, such as Twitter - a social media site where

comments are limited to 280 characters, VADER even outperforms human raters when analysing classification accuracy, and is statistically just as good as human raters at matching ground truths. VADER is able to classify a text as positive, neutral, and negative.

#### Textblob

Textblob is a simple Python library for processing textual data and contains a lexicon-based sentiment analyser. It contains some predefined rules in the form of a weight dictionary, where each word in the dictionary has some sentiment score. This dictionary can help to calculate a text's polarity and sentiment. This sentiment analyser is also able to classify a text as positive, neutral or negative.

Textblob is also able to return a numerical value for the subjectivity of a piece of text. The subjectivity value quantifies the amount of factual information, as opposed to personal opinion, expressed within the text. The higher the subjectivity score, the more subjective the text is.

#### SentiWordNet

SentiWordNet is a lexical resource created to support SA and opinion mining [38].

WordNet is a large lexical database of English nouns, verbs, adjectives, and adverbs, which are grouped into sets of cognitive synonyms called synsets [39]. Each synset expresses a distinct concept, and these synsets are interlinked via lexical and semantic relations. The WordNet search [1] can be used to view any word's semantic and lexical relations, as well as its multiple meanings. WordNet can be thought of as a thesaurus, as words are grouped together based on their meanings, but with a couple of key differences. Firstly, WordNet links together words based on their senses, meaning that words that are close to one another in the network are semantically disambiguated, as described in section 2.2.5. Secondly, the semantic relations between words are labelled, unlike in a thesaurus.

SentiWordNet is the result of the annotation of all of the synsets found in WordNet. Each synset is given a positive, negative, and objective score.

## 2.6 Existing Work

A variety of research studies have been conducted into performing NLP and SA on social media comments, and even specifically YouTube comments, as well as in other contexts, like Twitter. Most existing research seems to be focused on the development of algorithms and computational

[1] http://wordnetweb.princeton.edu/perl/webwn

techniques for SA and feature extraction. Some of these studies have already been discussed in this chapter. This section further discusses relevant existing work.

No existing commercial applications could be found that specifically perform NLP and SA on YouTube comments.

In their thesis, Poche et al. [40] have used SVM and naïve Bayes classifiers to classify comments into two categories - ”content concerns” and ”miscellaneous”. The results showed that SVMs could categorise comments into either category with an average accuracy of 77%.

As already described in section 2.5.4, Hutto et al. [36] built a simple rule-based model for general sentiment analysis and compared its effectiveness with other machine learning-based methods, including naïve Bayes and SVM algorithms. The VADER lexicon is specifically attuned to micro-blog like contexts, which are similar to YouTube comments. Other studies have built off the VADER lexicon. Chaithra et al. [37] use the VADER lexicon to label a dataset of YouTube mobile unboxing video comments as either positive, neutral or negative, and then trains a naïve Bayes classifier with the labelled data. The classifier is then applied to testing data and is found to have an accuracy of 79.78%.

Rinaldi et al. [41] use an SVM algorithm to classify Indonesian YouTube comments about smartphones into seven classes: spam/off-topic, product-negative, product-neutral, product-positive, video-negative, video-neutral, and video-positive. They focus on finding the best kernel function in terms of accuracy for the SVM. Out of the four kernel functions that are tested, the linear kernel function comes out on top with an accuracy of 62.76%.

Clearly, a lot of existing research makes use of machine learning classification models to classify social media comments into various categories - this study has also made use of some of these machine learning algorithms, such as SVMs. On the other hand, many existing studies tend to use comments from Twitter and Reddit, as opposed to YouTube, unlike this study which works with YouTube comments. Of all of the existing work on YouTube comments, a lot of studies focus on other languages, such as Indonesian and Indian, unlike this study which focuses on the English language.

# Chapter 3

# Specification Requirements

This chapter highlights the functional and non-functional requirements of the project, as well as the specification of the project. Some of these requirements have already been outlines in section 1.1. These requirements were generated at the start of the project, and then adapted as the project progressed based on feedback from the project supervisor, and the time left until the deadline of the project.

## 3.1 Functional Requirements

1. The software must retrieve all comments of a YouTube video, taken as user input, and store them in a CSV file.
2. The software must filter and clean the data using the following methods:
   (a) Remove empty comments.
   (b) Remove non-English comments.
   (c) Remove URL links.
   (d) Remove special symbols.
   (e) Convert any uppercase letters into lowercase letters.
   (f) Remove accented characters.
   (g) Expand contractions.
   (h) Replace some punctuation with relevant words.
   (i) Remove all punctuation.

(j) Correct any spelling errors.

3. The software must process the data into a state where NLP and SA can be performed on it. This must include the following procedures:

   (a) Tokenise all of the comments.

   (b) Perform lemmatization on the tokens.

   (c) Remove all stop words from the tokens.

4. The software must analyse the data using the following methods:

   (a) Using a lexicon to assign each comment a polarity (sentiment) score.

   (b) Performing count vectorization on the data.

   (c) Performing TF-IDF vectorization on the data.

   (d) Train a machine learning-based classifier using categorically labeled corpora; classify each comment into a category.

5. The software must output the following statistics about the data, giving the user an option to filter any of the statistics to only include data from the last 30 days, the last 7 days and the last 48 hours, as well as all time:

   (a) The total amount of positive, neutral and negative comments over the whole dataset.

   (b) The average valence of a comment.

   (c) A list of all of the comments, with an option to filter the comments by most positive and by most negative.

   (d) A list of the most commonly used words throughout the dataset, sorted by the most common words, displayed as raw data as well as through a bar chart and a word cloud.

   (e) A list of the most important words used throughout the dataset, sorted by the most important words, displayed as raw data as well as through a bar chart and a word cloud.

   (f) A breakdown of what categories have been assigned to each of the comments, and the frequencies of these categories, displayed as raw data as well as through a bar chart and a word cloud.

6. For each unique YouTube video, the software must save the following data using CSV files:

   (a) The comments and timestamps of each comment.

   (b) The comments, with each stage of the cleaning process displayed in its own column.

   (c) The comments, along with their polarity scores, sentiments (positive, neutral, or negative), and assigned categories.

   (d) A report of the total amount of positive, neutral and negative comments over the whole dataset, as well as this data for the last 30 days, the last 7 days and the last 48 hours.

## 3.2 Non-Functional Requirements

1. The software must output accurate and reliable results.
2. The software must output data in a reliable time frame.
3. The software must be easy to deploy, configure and maintain in the future.
4. All data collected must remain anonymous.
5. The software produced should meet the highest professional and ethical standards set out by the BCS.

## 3.3 Specification

### 3.3.1 Python

The Python programming language [1] has been chosen for this project because of its suitability in performing NLP tasks, as well as its good string-handling functionality. Furthermore, I am quite familiar with the programming language.

Throughout the project, the Google Python Style Guide [2] has been followed.

### 3.3.2 Comma-Separated Values Files

Comma-separated values files (CSV files) have been used to store data across the whole project. The CSV format is generally faster and less complicated when compared to the normal excel

[1] https://www.python.org
[2] https://google.github.io/styleguide/pyguide.html

format. On top of this, Python and the other libraries used in the project are more suited to work with CSV files, and already have commands built in to write to CSV files. For example, the pandas library, which is discussed in section 3.3.5, has the command "*to_csv*", which is able to write to a CSV file.

### 3.3.3 Google API - YouTube Data API

Using the Google API, more specifically the YouTube Data API [3], seems to be the fastest, most reliable way of downloading a YouTube video's comments. The YouTube Data API allows programmers to incorporate functions that would normally be executed on YouTube's website. Google provides this service completely free of charge. The exact process of configuring the Google Cloud Platform developers console has been described in section 4.1.

### 3.3.4 Natural Language Toolkit

The Natural Language Toolkit (NLTK) [4] is a suite of open-source Python libraries and programs used for NLP. The NLTK provides convenient functions and wrappers that can be used for common NLP tasks, as well as raw and preprocessed versions of corpora, large, structured sets of machine-readable texts.

One of the NLTK's useful features is the tokenizer package, which is able to turn a string into a list of tokens. Tokenization must be applied to a dataset before various machine learning methods can be applied to said data. A further useful feature is the WordNet lemmatizer that is provided, and can be used to reduce words to their dictionary forms. Another feature of the NLTK is the POS tagger, which is able to assign POS tags to tokens.

Another useful resource is the list of stop words that the NLTK provides. Removing stop words is done to remove any excess noise from the data. NLTK's list of stop words can be compared against a dataset to remove any of the unwanted stop words from said dataset.

Tokenization, lemmatization and POS tagging have all been described in section 2.2.3, while the removal of stop words has been discussed in section 2.3.

### 3.3.5 Pandas

Pandas [5] is another open-source library that is most widely used for data science/analysis, as well as various machine learning tasks.

[3] https://developers.google.com/youtube/v3/docs
[4] https://www.nltk.org
[5] https://pandas.pydata.org

Pandas provides dataframes, which are data structures that contain labelled rows and columns. This can be seen as a dictionary-like container for series objects. These are immensely useful when dealing with large amount of data, such as comments or reviews…

### 3.3.6 NumPy

NumPy [6], which stands for numerical python, is a python library that is used for various mathematical computations, mainly working with arrays, but also working with matrices… Will be used for pre-processing of text…

### 3.3.7 Matplotlib

Matplotlib [7] is a plotting library in Python used for creating static, animated and interactive visualizations in Python. Some of my requirements involve plotting bar charts - this library will be used to do this.

### 3.3.8 Scikit-learn

Scikit-learn [8] is one of the most widely used open-source python libraries in the field of data analysis and machine learning. It provides simple and efficient tools for predictive data analysis, and features various classification, regression and clustering algorithms.

The scikit-learn count vectorizer, TF-IDF vectorizer, label encoder and SVM classifier have been used in this project.

### 3.3.9 vaderSentiment

The VADER lexicon has been described in section 2.5.4. This library [9] provides the following: the VADER lexicon as a text file, various corpuses, also as text files, and the python code for a rule-based sentiment analysis engine.

### 3.3.10 TextBlob

The TextBlob library [10] has been briefly touched upon in section 2.5.4. This library uses resources from the NLTK library to achieve its tasks. TextBlob's sentiment analyser is used in this project as a 'backup' lexicon-based method, in cases where the VADER lexicon is unable

[6] https://numpy.org
[7] https://matplotlib.org
[8] https://scikit-learn.org
[9] https://github.com/cjhutto/vaderSentiment
[10] https://textblob.readthedocs.io/en/dev

to assign a polarity score to a comment (or cases where VADER assigns a polarity score of 0). Textblob's subjectivity classification feature is also used in the project.

The TextBlob library also comes with a spelling correction feature, which is able to correct basic spelling mistakes in comments. This is also applied to the dataset.

### 3.3.11 WordCloud

WordCloud [11] is a word cloud generator for Python. Word clouds are collections of words depicted in different sizes. The bigger a word appears in the cloud, the more times it was mentioned in a text (or some other metric aside from frequency). Word clouds are helpful data visualization techniques that allow people to view data in a simplified way, compared to just in a table. Word clouds have been generated during the project to display the frequency and importance of words in a dataset.

### 3.3.12 PyQt5 & Qt Designer

PyQt5 [12] is a free cross-platform graphics user interface (GUI) toolkit, implemented as a Python plug-in. PyQt5 includes a substantial set of GUI widgets that can be used to construct a GUI.

Qt Designer [13] is a piece of software for designing and building GUIs using the PyQt5 widgets described above. Qt Designer allows users to create and customize windows, dialogues and forms in a what-you-see-is-what-you-get manner, which is output as Qt UI files. These files can then be translated into python files with the use of the dev tool "pyuic5" [14] in a command prompt terminal.

### 3.3.13 Corpora

This section describes the corpora used in this study.

#### GoEmotions

GoEmotions [15] is a corpus developed by the Google Research team that consists of 58,000 Reddit comments that have been annotated manually with 28 categories representing emotions. Reddit [16] is an American social media website where users can post content such as links, text

[11] https://github.com/amueller/word$_c$*loud*
[12] https://doc.qt.io/qtforpython
[13] https://doc.qt.io/qt-5/qtdesigner-manual.html
[14] https://pypi.org/project/pyuic5-tool
[15] https://github.com/google-research/google-research/tree/master/goemotions
[16] https://www.reddit.com/

posts, images and videos. Users can also post comments on each others posts. Both posts and comments can either be upvoted or downvoted. The list of 28 emotion categories is available on the GitHub page[17]. The corpus was developed by Demszky et al. [42], where a classification algorithm was also built that uses the corpus. In this study, only the GoEmotions corpus is used.

**OffensEval (SemEval-2019)**

OffensEval is a corpus developed by Zampieri et al. [43], and is available for download [18]. OffensEval contains 11,916 Twitter comments that have been labeled as either offensive or not offensive.

[17] https://github.com/google-research/google-research/blob/master/goemotions/data/emotions.txt
[18] https://github.com/cardiffnlp/tweeteval/tree/main/datasets/offensive

# Chapter 4

# Design & Implementation

This chapter discusses the design and implementation of the project. This includes the data collection and cleaning processes, performing SA on the data, performing vectorization on the data, the classification of the data into categories, and the implementation of the user interface.

As stated previously, the Google Python Style Guide [1] has been followed in each of the python files created. The code was run through the YAPF styler [2], which restructured the code so that it adhered to the Google Style Guide. After this restructuring, some minor manual style changes were made where the styler did not do such as great job.

The system begins by initializing the "**Main**" class inside of the "*main.py*" file, described in more detail in section 4.6. This class is the main class of the system, and is responsible for making calls to all of the other classes.

Figure 4.1 is a flowchart of the whole system. As a note, the words "function" and "method" are used interchangeably across the chapter.

## 4.1 Data Collection

Once the program is run, the menu window shown in figure 4.6 is displayed. The user is given two options - either enter a YouTube video's URL into the text box and press "Import from YouTube URL", or select the name of a video from previously downloaded data and press "Import from CSV file".

If the user chooses to import comments from a YouTube video's URL, the program instantiates the "**CommentFetcher**" class, passing in the URL entered by the user.

[1] https://google.github.io/styleguide/pyguide.html
[2] https://github.com/google/yapf

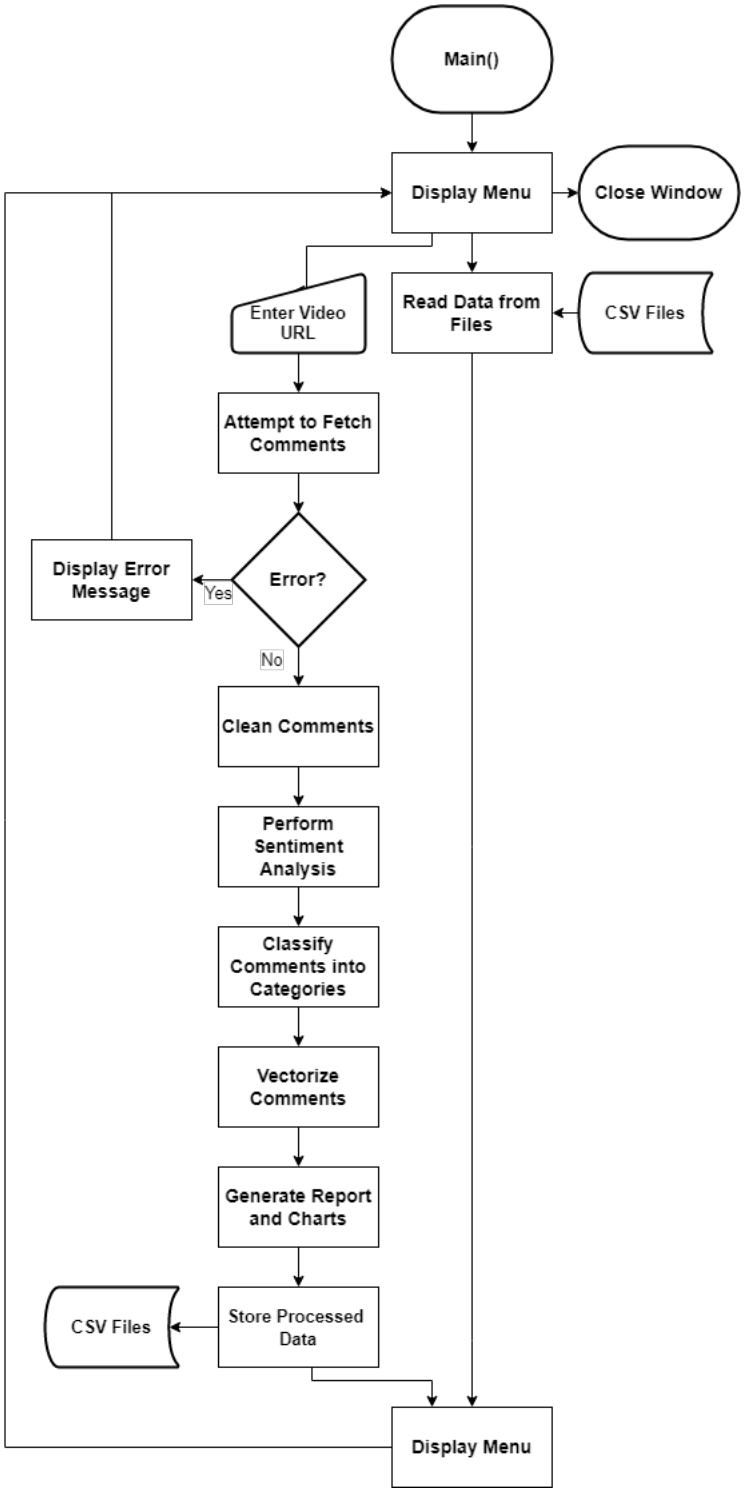



Figure 4.1: Flowchart of the system flow

Here, the YouTube Data API, described in section 3.3.3, is used to fetch the comments of the entered URL. Some testing was done using the ”youtube-comment-scraper-python” library, but this library was extremely unreliable and took a longer amount of time to download the

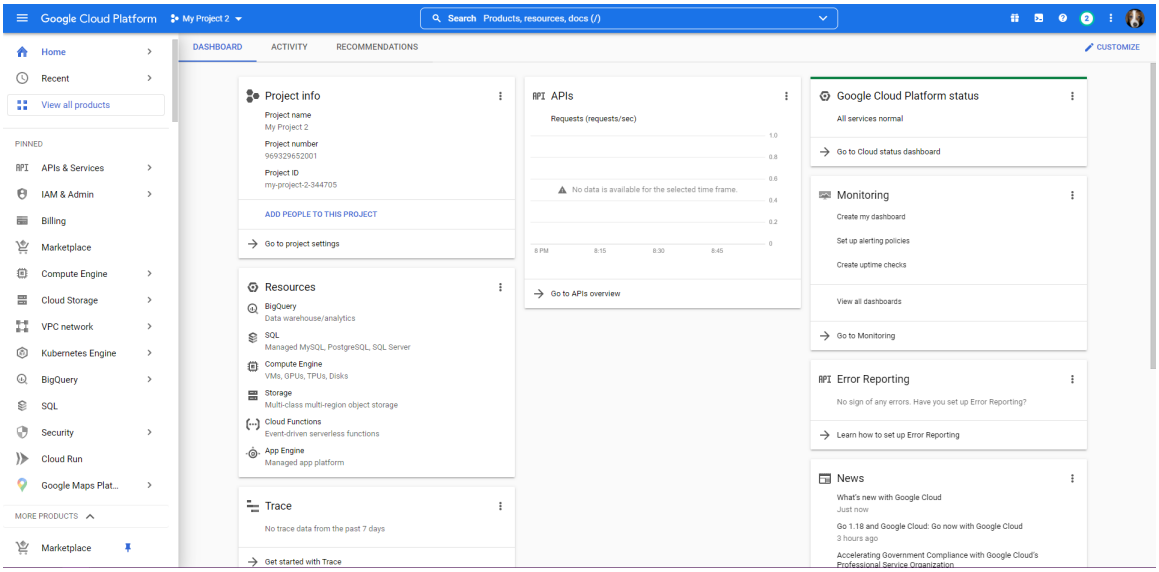



Figure 4.2: Google Cloud Platform home page

same amount of data. A decision was made to instead use the YouTube Data API.

Before being able to use the YouTube Data API, the Google Cloud Platform had to be configured [3]. Initially, the "APIs & Services" section had to be accessed through the sidebar on the left side of the web page, as displayed in figure 4.2.

A new project was then created by pressing the drop-down list of projects, and pressing "New Project", as seen in figure 4.3.

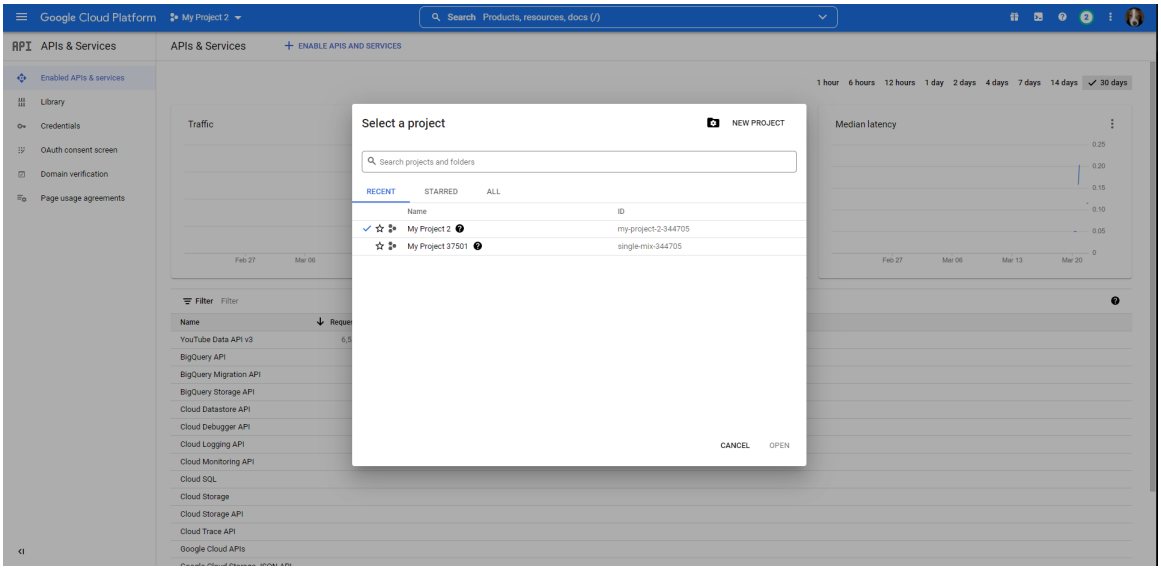



Figure 4.3: Google Cloud Platform APIs & Services page

Following this, the term "YouTube Data API v3" was entered into the search bar, where

[3] https://console.cloud.google.com

the relevant result was found and pressed. Once on this web page, the ”Enable” button was pressed, as seen in figure 4.4.

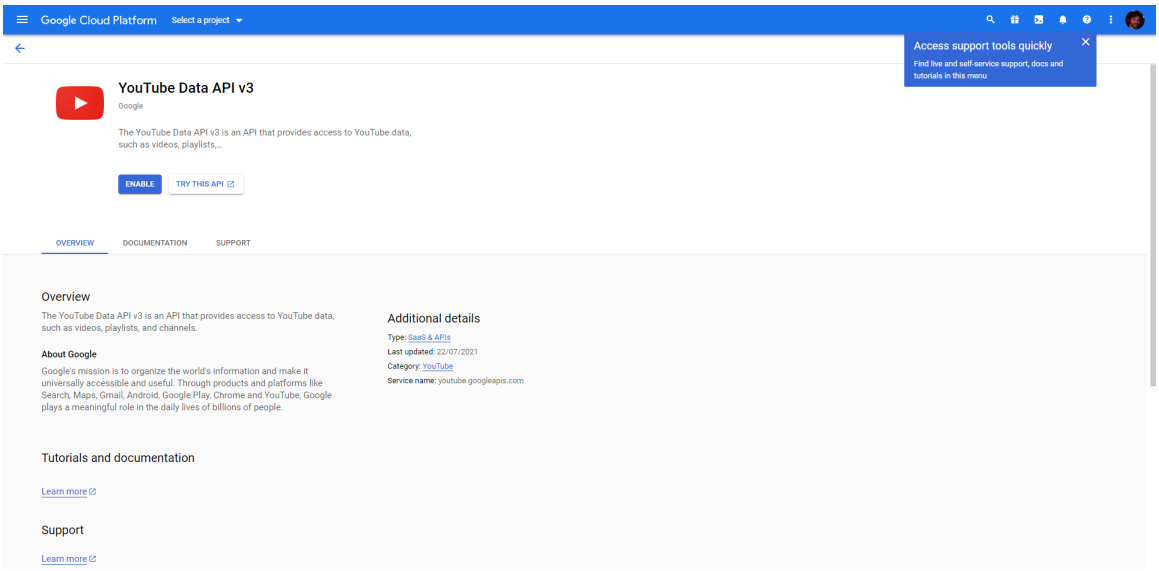



Figure 4.4: Google Cloud Platform YouTube Data API v3 page

Finally, the ”Credentials” section was accessed, where ”Create Credentials” was pressed, creating a new API key. This API key was copied and pasted into a text file names ”API Key”, as this API key would be required when fetching comments. Everything on Google's end was now set up.

The program then creates a YouTube resource object and retrieves all of the YouTube comments attached to the input URL, saving each comment along with its timestamp in a pandas DataFrame. The timestamp format is ”YYYY-MM-DD’T’HH:MM:SS”. As an example, the 21st of March 2022, at 17:30, would be represented as ”2022-03-21T17:30:00”.

A decision was made to only use the main comments posted to a video, ignoring any sub-comments, those being comments that are posted as a reply to other comments. This was done because the majority of sub-comments are not directly relevant to the video, but rather hinge on the original comment. If these were left in the dataset, a lot of irrelevant data would have been catalogued within the analysed data.

The system then fetches the title of the YouTube video and creates a folder with this name, with punctuation being replaced by whitespaces. The comments are saved in a CSV file, with the two columns described above, them being named ”*Comment*” and ”*Time*”. This CSV file is simply named ”*data.csv*”. The title is also saved as a pickle file. Pickle files are just files used to save and load python data.

The comments are then read into Python from the CSV file, where they are then passed

into the "**CommentCleaner**" class for cleaning. If the user had instead selected the "Import from CSV file" option, all of the above would have been skipped, and instead, the system would have read into Python the relevant CSV file's data, selected by the user.

## 4.2 Data Cleaning

In order to perform various lexicon-based methods and machine learning methods on the data, the data must be cleaned. The importance of data cleaning is outlined in section 2.3. This section describes the cleaning process. The "**CommentCleaner**" class is used to clean the data. A flowchart of the whole data cleaning process can be seen in figure 4.5.

The data is stored as a pandas DataFrame, and for each section of the cleaning process, a new column in the DataFrame is created. Table 4.1 shows each of the columns created during the data cleaning process, along with each function that performs this cleaning step, and an example of a comment at each stage of the cleaning. Once the data has been preprocessed, the cleaned data is stored in a file named "*cleaned_data.csv*".

| **Column Name** | **Section** | **Function Name** | **Example** |
|---|---|---|---|
| Comment | 4.1 | retrieve | Does his lips* mvoing mean he's LYING? 😂 https://www.dictionary.com/browse/lying |
| no_symbols | 4.2.2 | remove_symbols | Does his lips mvoing mean hes LYING? https-swww.dictionary.combrowselying |
| no_punctuation_and_symbols | 4.2.2 | remove_punctuation_and_symbols | Does his lips mvoing mean hes LYING https-swwwdictionarycombrowselying |
| no_urls | 4.2.3 | remove_urls | Does his lips mvoing mean hes LYING? |
| no_urls2 | 4.2.3 | remove_urls | Does his lips mvoing mean hes LYING |
| lower_case | 4.2.4 | lower_case | does his lips mvoing mean hes lying |
| expanded_contractions | 4.2.5 | expand_contractions | does his lips mvoing mean he is lying |
| expanded_contractions2 | 4.2.5 | expand_contractions | Does his lips mvoing mean he is LYING? |
| spelling | 4.2.6 | correct_spelling | does his lips moving mean he is lying |
| spelling2 | 4.2.6 | correct_spelling | Does his lips moving mean he is LYING? |
| tokens | 4.2.7 | tokenise | ['does', 'his', 'lips', 'moving', 'mean', 'he', 'is', 'lying'] |
| lemmas | 4.2.8 | lemmatize | ['doe', 'his', 'lip', 'moving', 'mean', 'he', 'is', 'lying'] |
| lemmas_no_stopwords | 4.2.9 | remove_stopwords | ['doe', 'lip', 'moving', 'mean', 'lying'] |

Table 4.1: Each of the pandas DataFrame columns, along with their respective functions and an example comment at that stage

For nearly every stage of the data cleaning, the pandas "*apply*" method is run on the

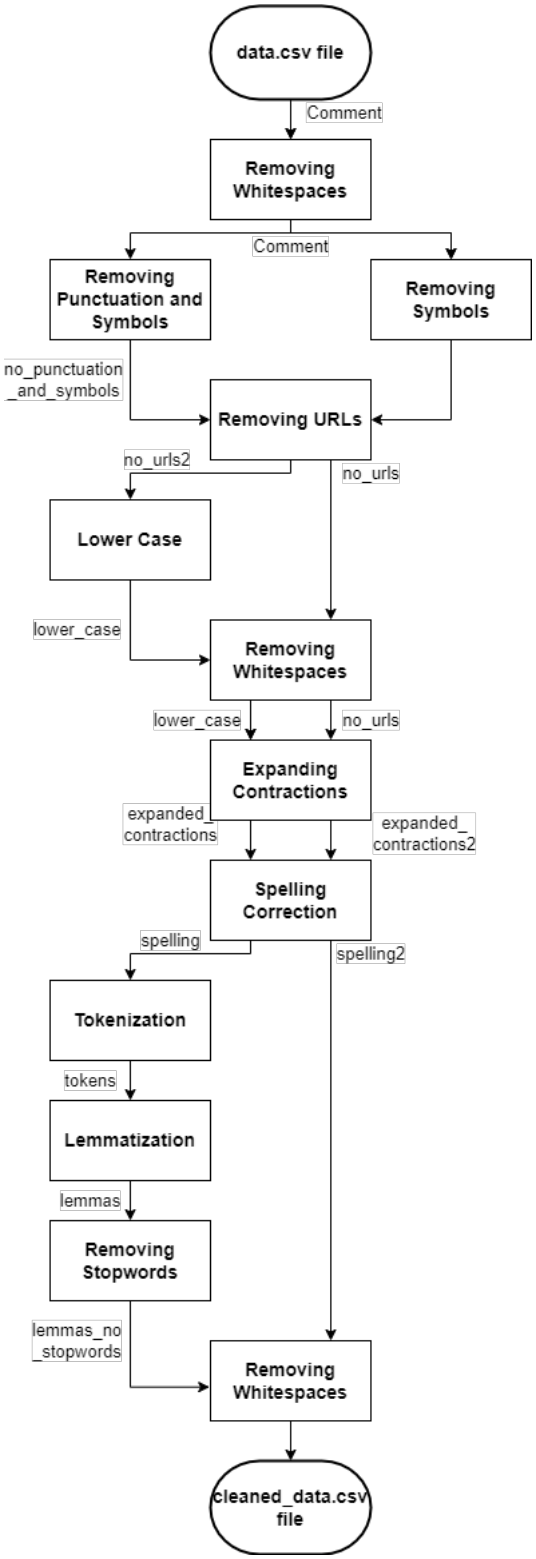



Figure 4.5: Flowchart of the data cleaning process

comments column of the dataset. This method iterates through each row in the DataFrame (in this case, only using the comments column), and applies a method to each comment. These methods are described in greater detail in their respective sections.

### 4.2.1 Removing Whitespace

Firstly, each row in the DataFrame is checked to ensure that any of the comment's columns are not just whitespace. Occasionally, when downloading comments from a video, an empty comment that does not contain any characters appears in the dataset. Due to this, a method was added to iterate through each row in the DataFrame, removing any rows where any column's value is just whitespace ie does not contain any characters. The reason for empty comments appearing in the dataset is unknown, but likely due to some sort of importing error, as YouTube does not allow users to post empty, whitespace-filled or characterless comments.

This function is run thrice during the data cleaning process. The second time is after any symbols and punctuation has been removed, in case some of the comments just contained symbols and punctuation that were now removed. The third time is the last step in the cleaning process and is done in case some of the comments contained only stop words, which have now been removed.

### 4.2.2 Removing Symbols and Punctuation

When performing various methods on the data, for example, count vectorization, symbols and punctuation must be removed from the dataset. On the other hand, Lexicons like VADER can actually make use of punctuation, as discussed in section 2.5.4. Because of this, the dataset splits down two forks from this point; one, column name "*no_punctuation_and_symbols*", where both symbols and punctuation are removed from the dataset, and another, column name "*no_symbols*", where only symbols are removed. The "*no_symbols*" column contains punctuation such as periods, commas, exclamation marks, and question marks, and will later be fed into the VADER lexicon, while the "*no_punctuation_and_symbols*" column will be used to train the machine learning classifiers, as well as during the count vectorization and TF-IDF vectorization.

Symbols can comprise other ASCII characters that are not letters, and other letters that are not of the English language. The reason for removing them is that no meaning can be drawn from these symbols and none of the methods used in this study take into account random ASCII characters or characters from other languages.

This is done using a Python list comprehension for loop. The method that is applied to each comment checks each character of the comment to see if it is a letter or whitespace (or punctuation, as discussed above). If the character meets the criteria, it is kept, otherwise, it is replaced by the empty space "".

### 4.2.3 Removing URL's

The machine learning methods and lexicons used in the project cannot draw any meaning from URLs. As a result, any URLs in the dataset are seen as noisy, useless data.

When planning the project, some research was done into the possibility of opening any hyperlinks present in the dataset and replacing these URLs with any text found on the linked web pages. For example, if a comment had contained a URL that linked to a web page of the dictionary definition of the word "Police" (for example this URL "https://dictionary.cambridge.org/dictionary/english/police"), the system would replace the URL with the text "the official organization that is responsible for protecting people and property, making people obey the law, finding out about and solving crime, and catching people who have committed a crime". However, after some thought about this, it was decided that this was not a priority feature, as the largest majority of comments on the website do not contain URLs, and that pulling text from a web page may not even match up with what the commenter was trying to express.

Consequently, YouTube comments that contained URLs had them removed using Python's regular expression module. The method applied to each comment checks if the comment contains the term "http" or "https", followed by or preceded by any non-whitespace character zero or more times. If the comment contained this phrase, the phrase was replaced by the empty space "". This string is then returned. Two new columns are created during this procedure - The "*no_urls*" column, created from the "*no_symbols*" column, and the "*no_urls2*" column, created from the "*no_punctuation_and_symbols*" column.

### 4.2.4 Lower Case

As discussed in section 2.5.4, lexicons like VADER can make use of both upper case and lower case characters. It was also discussed in section 2.3 that it is extremely important for the dataset's comments to all be in lower case before applying various methods, such as count vectorization, to ensure that the results of the analysis are consistent.

Due to this, only the "*no_urls2*" column, that being the data that is not intended to be used by the VADER lexicon, is processed during this function, where a new column named

"*lower_case*" is created. The method that is applied to each comment iterates through each character in the comment, checking if this character is a letter using the "*isalpha*" method. If the character is a letter, the lower case version of the character is appended to a new list, otherwise, the character is just appended to this new list. The list is then returned.

### 4.2.5 Expanding Contractions

To keep the data as consistent as possible, any contractions found within the dataset are expanded. If this step was not taken, words such as "don't" would be processed separately compared to words such as "do" and "not", even though the user means the same thing.

Each word in the "*lower_case*" and "*no_urls*" columns is checked against a dictionary of contractions, with the keys being the contracted words, for example, "don't", and the values being the expanded contractions, that being "do not" in this case. This dictionary is stored in the "*contractions.py*" file. If any matches are found, the keys are replaced by the values. Two new columns, "*expanded_contractions*" and "*expanded_contractions2*", are created respectively. This method does not always replace the contraction with its correctly expanded word, for example, the contraction "how's" could expand to "how has", "how is, or "how does", where the application substitutes the most commonly used expansion. A better method to complete this step of the cleaning would be to choose the correct contraction's expansion depending on the context of the word a sentence. However, the largest majority of comments do not contain contractions that are incorrectly expanded, so this has very little impact on the accuracy of the processed data.

### 4.2.6 Spelling Correction

To get rid of excess noise in the data, the user is given the option of enabling the spelling correction feature, where each comment can be been run through TextBlob's spelling corrector. This takes in a piece of text and outputs that text, attempting to fix any spelling errors within the text. Once again, two new columns are created, named "*spelling*" and "*spelling2*"; both of these columns have data from the "*expanded_contractions*" and "*expanded_contractions2*" fed into TextBlob's spelling corrector, respectively.

The "*spelling2*" column is now ready to be fed into the lexicon-based methods, while the "*spelling*" still needs to be processed further.

### 4.2.7 Tokenization

Tokenization was discussed in section 2.2.3, and is performed on each comment in the "*spelling*" column. Each comment was split into a list of tokens, with each token being split at whitespace. This was done using the NLTK word tokenizer. The result is a list of words (tokens) stored in the "*tokens*" column.

### 4.2.8 Lemmatization

Lemmatization was also discussed in section 2.2.3, and is also performed on each of the tokenized comments. This converts each token into its basic root. This allows for all words with the same suffix(es) to be grouped together. This was done with the help of the WordNetLemmatizer and its "*lemmatize*" method. This results in the "*lemmas*" column being created with this data.

### 4.2.9 Removing Stop Words

The removal of stop words has been discussed in section 2.3. Initially, stop words were not removed from the dataset, resulting in a lot of stop words being displayed to the user in the n-gram frequency and importance tables that simply did not provide any useful insight into the user's audience and their views on the video. A decision was made to remove stop words from the dataset where count vectorization and TF-IDF vectorization are to be performed, reducing excess noise in the dataset and hopefully showing the user the more relevant n-grams in their comments.

Using the NLTK stop words corpus, any words in the comments that matched words in the corpus were removed. The method that is applied to each comment appends the word to a new list if the word is not a stop word, and returns this list. This data is stored in a column named "*lemmas_no_stopwords*".

### 4.2.10 Date-Time Formatting

Each of the date-time stamps, representing when each of the comments has been posted, is formatted here to make it more readable to the user. The additional "T" and "Z" letters that were added by the YouTube Data API are removed from the date-time stamps. A "
n" newline character is also added in between the date and the time so that when these date-time stamps are displayed to the user in the comments screen in figure 4.10, they take up less vertical space and are more readable.

### 4.2.11 Removing Duplicates

Initially, duplicates were automatically removed from the dataset. This decision was made with the hopes of eliminating excess useless noise in the dataset, as well as eliminating any duplicate comments posted by bots, which would likely skew the count vectorization and TF-IDF vectorization data towards a group of n-grams. However, later, when the application was being tested, it was discovered that a lot of short duplicate comments posted by users did contain meaningful responses, not simply just spam. A good example of duplicate comments that do provide some meaningful insight are "good job" or "great job", as well as some negative examples of this, are described in the testing section 5.1.2. This is because a lot of commenters like to just post the single phrases "good job" or "great job", praising their content creators for making a good video. If duplicated were removed here, a lot of these features from the data would be removed from the results for no good reason. As a result, duplicates were not removed from the datasets.

## 4.3 Sentiment Analysis

The "**CommentSentiment**" class is used to perform SA on the comments. This is done with the help of various lexicons, described in section 2.5.4. The VADER, TextBlob, and (Senti)WordNet lexicons have been used here. The VADER lexicon is used to give each comment a polarity score, stored in the "*Polarity*" column in the pandas DataFrame. Each comment is given a polarity score between -1 and 1.

If the assigned polarity score is 0, the comment is assigned another polarity score, except this time using the TextBlob lexicon. If the polarity score is still 0, the (Senti)WordNet lexicon is used to assign a polarity score. For each word (token) in the comment, the NLTK POS tagger is used to assign a tag to each token. Following this, each word is matched with a WordNet synset based on the assigned POS tag. Each synset is assigned a polarity score, and these polarity scores are summed for the whole sentence. This value is then used as the polarity of the comment. This is done to help mitigate results where the VADER and TextBlob lexicons are unable to assign a polarity score due to either not enough information in the comment or too many noisy features in the comment.

Additionally, a further column named "*Sentiment*" is created, where each comment is labelled as either positive if its polarity score is greater than or equal to 0.05, negative if the polarity score is less than or equal to -0.05, or otherwise neutral. This makes it easier for the

user to identify a comment as positive or negative, rather than having to look through only numerical data about a comment's sentiment.

The techniques used here to assign a sentiment score to each comment are examples of techniques at the lexical level of NLP that deal with individual words, as the lexicon-based methods described above use lexicons that store sentiment scores for words in the English language, and use these scores to assign an average score to each comment. The SA performed here is an example of document level SA, as each individual comment is assigned a sentiment. When assigning this sentiment, entity level SA can be performed if the SentiWordNet lexicon is used, as the sentiment of individual words is interpreted and concatenated into one sentiment score. However, the actual entities that are analysed do not have their sentiments individually stored in this project.

Furthermore, TextBlob is also used to assign each comment a subjectivity score, ranging between 0 and 1. The higher the subjectivity score of a comment, the more likely it is to contain personal opinion as opposed to factual, objective information. This is done by calculating the intensity of each word and summing this across the whole comment. The intensity of a word is determined based on how much the word "modifies" the next word [44]. This information is stored in the "*Subjectivity*" column.

## 4.4 Vectorization

The "**CommentVectorizer**" class performs count vectorization and TF-IDF vectorization on the data. This is done using sklearn's "**CountVectorizer**" and "**TfidfVectorizer**" classes.

Both count vectorization and TF-IDF vectorization are performed in a similar fashion. Initially, a vectorizer is created using sklearn's respective vectorizer class. The system performs both of these vectorizations on unigrams, bigrams and trigrams. The "*fit_transform*" method is run with the "*lemmas_no_stopwords*" column, which makes the vectorizer learn the vocabulary dictionary, as described in sections 2.5.1 and 2.5.2. A document-term matrix is produced, with each column dedicated to an n-gram in the document, and with each row representing a comment in the dataset. This matrix is converted into a pandas DataFrame and is then transposed into a DataFrame with each row representing an n-gram, and with the single column representing either the occurrences or the importance of each n-gram, depending on which technique/vectorizer has been used.

Initially, no maximum feature constraints were placed on the vectorizers. However, once the testing of the application began, videos with a large number of comments, for example the

testing video titled "Future - Mask Off (Official Music Video)" documented in section 5.1.3, which has more than 150,000 comments, caused the application to crash due to a memory error caused by the large number of unique n-grams that the vectorizers attempted to quantify. As a result, the "max_features" argument was used with both of the vectorizers above to limit the number of features, or n-grams, to 2000. This number was decided upon after some trial and error. All of the testing was done using a machine with 32GB of RAM. The maximum feature amount can be toggled using the global constant "MAX_FEATURES" at the top of the "*main.py*" file if using a machine with a smaller amount of memory, or a video with an extreme amount of comments with a large enough variety of unique n-grams.

This DataFrame is then displayed to the user on its appropriate screen in the GUI, depending on the n-gram that the user selects. This data is also converted into bar charts and word clouds using the "*generate_charts*" function, as described in section 4.6.

The vectorization performed in this study is an example of lexical level NLP, as individual words (or n-grams) are processed and visualised.

## 4.5 Classification

The "**CommentClassifier**" class is used to classify comments into categories. This is done with the help of the GoEmotions corpus used by Demszky et al. [42] and the OffensEval corpus used by Zampieri et al. [43], described in section .

The GoEmotions corpus file was originally downloaded from its GitHub page [4], but needed to be formatted and cleaned first before being used. The code used to clean the corpus resides in the "*clean_corpora*" function. The corpus consists of three CSV files which are initially loaded into one DataFrame. This data is then cleaned using the same "**CommentCleaner**" class described earlier. Each comment in the dataset is then assigned an emotion category, stored in the "*Category*" column. This dataset is then output and saved as a single CSV file named "*cleaned_emotions_corpus.csv*". The dataset is only cleaned a single time, whereafter on the next execution of the program the system just loads the preprocessed data from the generated CSV file. By using this method, there is no need to go through the lengthy cleaning process every execution of the program. All of these files reside in the "*Python/Corpora/Emotions*" folder of the project.

The OffensEval corpus is cleaned and stored in the same way. The raw data is just a single CSV file instead of three. The data for this corpus is stored in the "*Python/Corpora/OffensE-*

[4] https://github.com/google-research/google-research/tree/master/goemotions

*val*" folder.

The categories are then encoded into numbers using sklearn's "**LabelEncoder**" class. Following this, the "*lemmas_no_stopwords*" column of each corpus is fit and transformed into a count vectorizer, which returns a document-term matrix, where each column is dedicated to a word in the document, while each row represents a comment in the corpus.

An sklearn decision tree classifier is then trained using the GoEmotions matrix, as this algorithm seemed to be the most suitable for this multi-class problem consisting of 28 unique categories. When implementing the classifier, multiple sklearn classifiers were tested to see which classifier would be able to classify the most amount of comments into emotion categories, not including "neutral". These included the following classifiers: decision trees, k-nearest neighbours, naïve Bayes, random forest, gradient boosting, and SVMs. Most of these classifiers are suited to work with multi-class problems (as opposed to binary problems) [45], and as a result, were tested here. Decision trees produce the highest degree of accuracy while still being able to accurately classify the data, and as a result, this classifier was the one used in the final version when working with the GoEmotions corpus.

An sklearn SVM classifier was used for the OffensEval corpus, as this corpus involved a binary classification - either the comment is offensive, or it is not. SVM classifiers tend to perform better with binary classifications compared to multi-class classifications [45].

These trained classifiers are saved as pickle files, so that the classifiers do not have to be trained every time. This also saves substantial time.

The "*lemmas_no_stopwords*" column of the YouTube video's comments is also then used to produce a document-term matrix from the count vectorizer. Using this, the classifier predicts an emotion category and offensive value (yes or no) for each of the comments. Using the label encoders "*inverse_transform*" method, each emotion category is returned from its matching number. This information is stored in the "*Emotion*" and "*Offensive?*" columns of the comments DataFrame.

Later in the execution of the program, the emotional analysis is summarised, where the frequency of each assigned emotion in the dataset is calculated and displayed to the user on the "Emotional Analysis" screen.

## 4.6 Main

The "**Main**" class is responsible for the execution of the system and is the first class to be instantiated when the program is run. Various methods of this class are run that are responsible

for each part of the system's execution.

Once the "**Main**" class has been instantiated, the "*generate_directories*" function generates all of the folders that will be needed during the execution of the program, if they do not already exist. This includes the main "*Python*" folder, where all of the other folders that are used during the execution of the program are stored. This also includes the "*Videos*" folder, inside of the "*Python*" folder.

The class then calls the "*display_window*" function for the first time to display the menu. If the user enters a URL and presses "Import from YouTube URL", the system attempts to fetch the video's comments, as described in section 4.1. If the URL is valid, meaning if the YouTube Data API is able to find a video relating to the entered URL and successfully download its comments, the system proceeds with the data cleaning and processing. If the URL is invalid, or the video that the URL relates to does not have any comments, the "*display_error*" function is called to display the appropriate error message to the user. The menu is then displayed again.

If the system is able to download a video's comments successfully, the "*process_comments*" function is called, which itself instantiates the "**CommentCleaner**", "**CommentSentiment**" and "**CommentClassifier**" classes and makes calls to their appropriate functions, cleaning, performing SA and classifying the data, as described across this whole section.

Following this, the class makes a call to the "*generate_empty_dataframes*" and "*split_data_by_time_ranges*" functions, which generate three new pandas DataFrames: one for comments from the last 30 days, one for the last 7 days, and one for the last 48 hours. This is done by comparing the timestamp saved with each comment to the current date, and then adding the comment to the appropriate DataFrame(s) if they are within the correct date range. Once these DataFrames have been generated, each of them is passed into the "**CommentVectorizer**" class, which performs count vectorization and TF-IDF vectorization on the data.

The "*generate_charts*" function makes calls to the "*generate_chart*" function, which as the names would suggest, generates the bar charts and word clouds relating the count vectorized and TF-IDF vectorized data, and saves them in the respective video's files in a file named "*Charts*". This is done using the "matplotlib" library, discussed in section 3.3.7, as well as the "wordcloud" library, discussed in section 3.3.11. For both the count vectorized and TF-IDF vectorized data, four bar charts and four word clouds are created for each time range of data (all time, last 30 days, last 7 days, last 48 hours). After the creation of each bar chart, the chart is rotated 90 degrees clockwise to make it more readable. The x-axis labels have also been rotated to match this alignment.

The "*generate_report*" function is the next one to be called, which generates the report that is displayed on the first screen of the main window GUI, shown in figure 4.8. This comprises the total number of positive, neutral and negative comments in each of the time ranges, as well as the total number of comments, and the average valence and average subjectivity of the comments in each of the time ranges. On top of this, the amount of offensive classified comments is also appended. When calculating the average valence of the comments, comments with a valence of 0 are not counted. This decision was made because most of the comments that are assigned a valence of 0 most likely do have some sort of positive or negative connotation, but either harbour too many noisy features or do not contain enough features for the lexicon-based methods to assign a score. Leaving the 0 scores in the average calculation would skew the average valence towards 0, making the data's average valence score more neutral than it otherwise would be.

The last function to be called during the execution of the "*process_comments*" function is "*generate_files*", which outputs the cleaned and processed data into CSV files. This is done so that if the user wants to see the results of the system's analysis on this same video, the system does not have to clean and process the data from scratch but can instead upload this data from these generated files, saving time and processing power. This has also been done to allow flexibility for the user; the user can now download and move a video's cleaned and processed data to any other machine in a common CSV file format, and view and possibly further process this data if need be. All of the count vectorized and TF-IDF vectorized data is also saved as a pickle file. As mentioned previously, pickle files are just files used to save and load python data. All of the files generated for each video processed can be seen in table 4.2.

| **Name** | **Type** | **Description** |
|---|---|---|
| title | pickle | Title of the video |
| data | CSV | Raw comments with timestamps |
| cleaned_data | CSV | Cleaned comments |
| emotional_analysis | pickle | Emotional analysis |
| report | CSV | Report |
| processed_all | CSV | Processed comments |
| processed_30_days | CSV | Processed comments from last 30 days |
| processed_7_days | CSV | Processed comments from last 7 days |
| processed_48_hours | CSV | Processed comments from last 48 hours |
| vectorizer_data | pickle | Vectorizer data |
| Charts | Folder with PNGs | Folder with bar charts and word clouds |

Table 4.2: All files generated for each video processed

Going back to the menu window, if the user had instead chosen to import data from a CSV file, where the user had previously already entered a URL and had its comments processed,

the class would skip all of the cleaning and processing described above, and instead run the "*read_from_files*" function. Here, the previously generated processed data and report are loaded into their appropriate variables, ready to be passed into the GUI and displayed to the user.

After the data has been either been imported and processed or loaded from files, a final call is made to the "*display_window*" function, which displays the main data viewing window. All of the data is imported into the "**MainWindow**" GUI class and displayed to the user. When the user closes the main window, the menu is displayed again. If the user quits the menu, the application closes.

## 4.7 The Graphics User Interface

As discussed in section 3.3.12, the GUI was built using PyQt5 and Qt Designer. Qt Designer has been used in this project to design the basic interface of the software. However, the actual connections between the buttons, text boxes, drop boxes and the menu bar used in the GUI, as well as the GUI controller methods and variables, were manually written. After designing each of the windows in Qt designer, a command prompt terminal was opened and the directory where each of the Qt UI files was saved was navigated to. Following this, the following two commands were run:

1. *pyuic5 -o Menu.py Menu.ui*
2. *pyuic5 -o MainWindow.py MainWindow.ui*

Each of the two commands produced a python file containing python code for each of the windows respectively. In each of these files, a single class representing the designed window had been created. This class contained two methods - "*setup_ui*", which creates each of the GUI widgets, including the menu bar, buttons, text boxes, drop boxes, tables, charts (images), and text; and "*retranslate_ui*", which simply sets the text on each of the menu bar actions, buttons, and the windows name.

Both of these classes have been copy-pasted into one file named "*gui.py*". Following this, each class was manually run through, where all variable and function names have been changed to match the Google Python Style Guide. The only function names that have not been changed are the inherited class functions, where changing the function names would throw an error.

Additionally, three classes have been added to the "*gui.py*" file; "**QLineEdit**", "**ListTableModel**", "**DataframeTableModel**".

The "**QLineEdit**" PyQt5 class has been overridden, where the "*mousePressEvent*" method has been changed. The only reason that this has been done is so that when the user clicks the text box to enter their URL, the "Enter YouTube URL" text prompt disappears, rather than the user having to manually delete the text prompt.

The "**ListTableModel**" class, which inherits the PyQt5 "**QAbstractTableModel**" class, and has been written to display a list of lists as a table to the user. This model is used when displaying the vectorized data to the user.

The "**DataframeTableModel**" class, which also inherits the PyQt5 "**QAbstractTableModel**" class, has been written to display a pandas DataFrame as a table to the user. This model is used when displaying the report and a list of processed comments to the user.

### 4.7.1 Menu

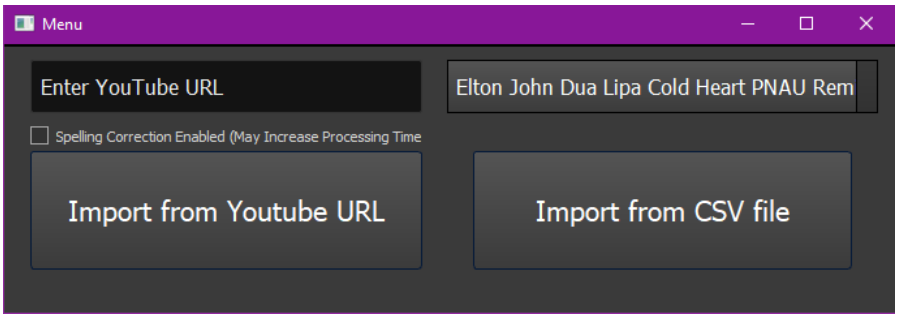



Figure 4.6: Menu window

The menu, displayed in figure 4.6, is the first window displayed to the user once the program is run. As stated previously, the menu gives the user two options: either type in a URL and import the comments from the respective video or select a CSV file and import the comments from there instead. The menu comprises two buttons, a text box for entering a URL, a drop-down list for selecting the name of the CSV file/video, and a check box that allows the user to turn the spell correction feature on or off.

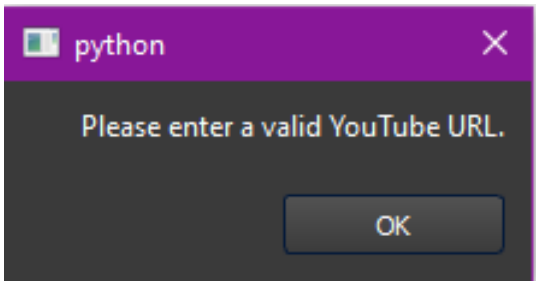



Figure 4.7: Menu error message

If the user presses the "Import from YouTube URL" button, the system saves the current text inside of the text box, and closes the menu. This text is retrieved by the **Main** class, which then attempts to download the comments from the specified URL. If the URL is invalid, an error message like the one in figure 4.7 is displayed, followed by the menu again. If the URL is valid, the system retrieves and processes the comments, opening the main window to display the results to the user.

The drop-down list comprises all of the file names found in the "*Videos*" folder. When the user selects an item from the list, the corresponding file name is saved as a variable. If the user then presses the "Import from CSV file" button, the menu is closed, and the **Main** class uses the file name specified in the drop box to retrieve the processed comments from their CSV files, opening the main window to display the results to the user.

### 4.7.2 Main Window

Once the user has made a selection using the menu, and once the data has been processed or imported, the "**MainWindow**" classes "*import_data*" method is called to import this processed data into the class. All of the processed data and statistics are displayed to the user through the main window. The user can select which scree of data/statistics they want to view using the drop-down list found in the menu bar. There are a total number of nine screens, each of which is discussed in the following subsections in greater detail. The first screen to be displayed to the user is the report screen. The variables displayed in table 4.3 are used to track the current elements to be displayed to the user.

| **Variable** | **Type** | **Purpose (assuming the appropriate screen is selected)** |
|---|---|---|
| *comments* | DataFrame | Current subset of comments being displayed to the user |
| *screen* | String | Current screen being displayed to the user |
| *chart* | String | Current chart being displayed to the user |
| *filter* | String | Current time range filter |
| *table* | List of lists | Current vectorizer data being displayed to the user |
| *sort_by* | String | Current sort metric (by sentiment, subjectivity or post date-time) |

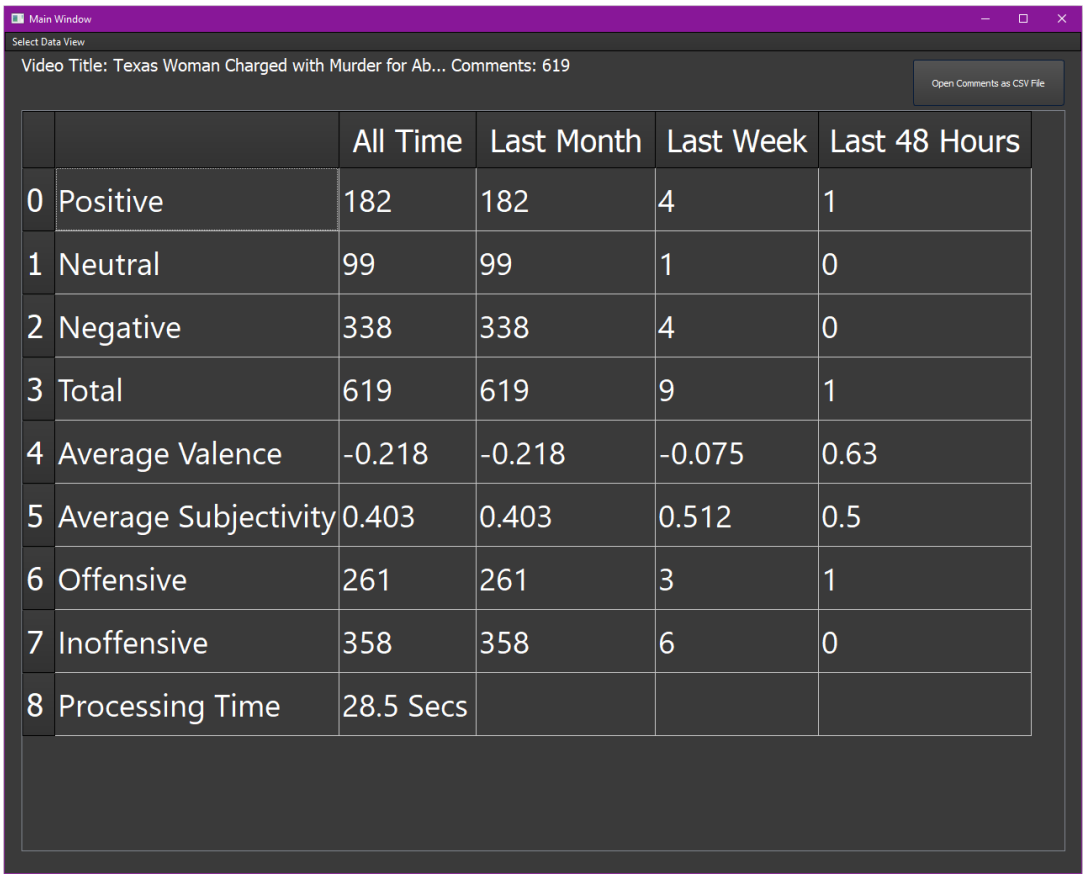
Table 4.3: **MainWindow** classes controller variables

Each radio button and menu bar button has a function defined that is run when the button is pressed, changing one of the variables in table 4.3, and then calling the "*refresh_ui*" method. Firstly, the "*refresh_ui*" method runs the "*hide_all*" method, which simply hides all of the UI

elements. Secondly, various radio sort and filter buttons are displayed depending on the current screen. Thirdly, the current screens corresponding tables and charts are displayed. The data in the tables is sorted depending on the current radio buttons selected.

Displayed on all of the screens in the top right corner is a button with the text "Open Comments as CSV File", which simply opens the CSV file of all of the processed comments, previously saved as "*processed_all.csv*", while in the top left corner there is the title of the video, along with how many comments were imported.

### Report



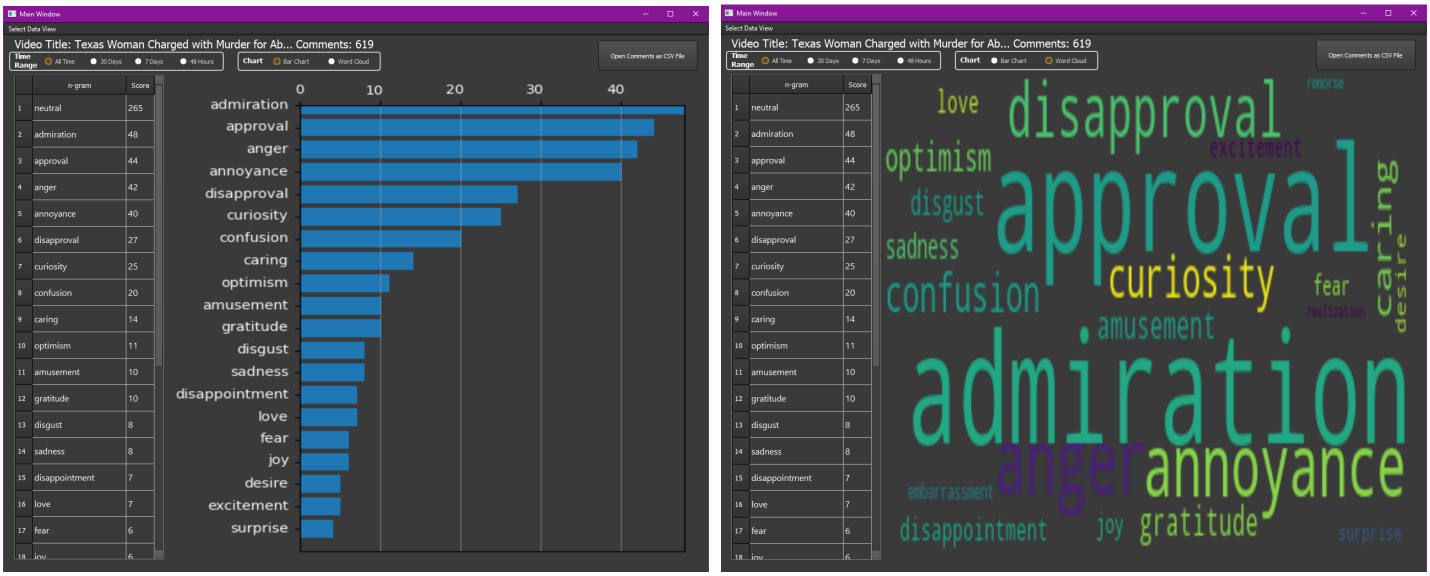

| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 182 | 182 | 4 | 1 |
| 1 | Neutral | 99 | 99 | 1 | 0 |
| 2 | Negative | 338 | 338 | 4 | 0 |
| 3 | Total | 619 | 619 | 9 | 1 |
| 4 | Average Valence | -0.218 | -0.218 | -0.075 | 0.63 |
| 5 | Average Subjectivity | 0.403 | 0.403 | 0.512 | 0.5 |
| 6 | Offensive | 261 | 261 | 3 | 1 |
| 7 | Inoffensive | 358 | 358 | 6 | 0 |
| 8 | Processing Time | 28.5 Secs | | | |

Figure 4.8: Report screen

The report screen, displayed in figure 4.8, gives the user a general overview of the sentiment and subjectivity of the selected video's comments. The screen displays the report generated by the "*generate_report*" function in the "**Main**" class, discussed in section 4.6. As stated previously, this comprises the total number of positive, neutral and negative comments in each of the time ranges (all time, last 30 days, last 7 days, last 48 hours), as well as the total number of comments, and the average valence and average subjectivity of the comments in each of the

time ranges. The report also shows how many comments were classified as offensive compared to inoffensive. The report screen does not display any radio buttons, as there is no data to sort or filter.

### Emotional Analysis



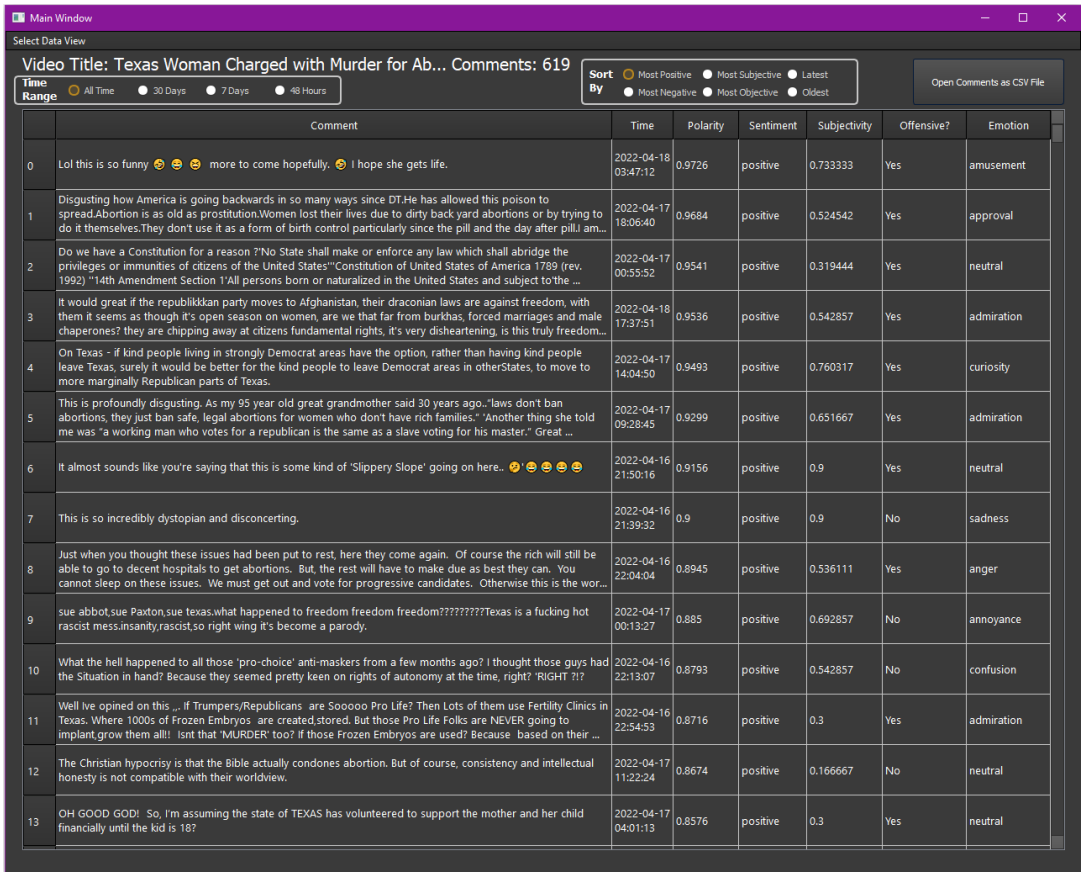

(a) Emotional analysis bar chart screen (b) Emotional analysis word cloud screen

Figure 4.9: Unigram word frequency and importance screens

The emotional analysis screen, displayed in figures 4.9, shows the user a breakdown of the emotional classification performed on the dataset with the help of the GoEmotions corpus. 28 emotion categories [5] have been used, where each comment has been assigned to a single category. The frequency of each category across the dataset has been calculated and displayed to the user on this screen. This data is also represented as a bar chart and word cloud - note that these charts omit the comments classified as "neutral", as this is always the most assigned category, and displaying this in the charts would not provide a great degree of insight to the user. The user can also filter this data for specific time ranges.

[5] https://github.com/google-research/google-research/blob/master/goemotions/data/emotions.txt

### List of Comments

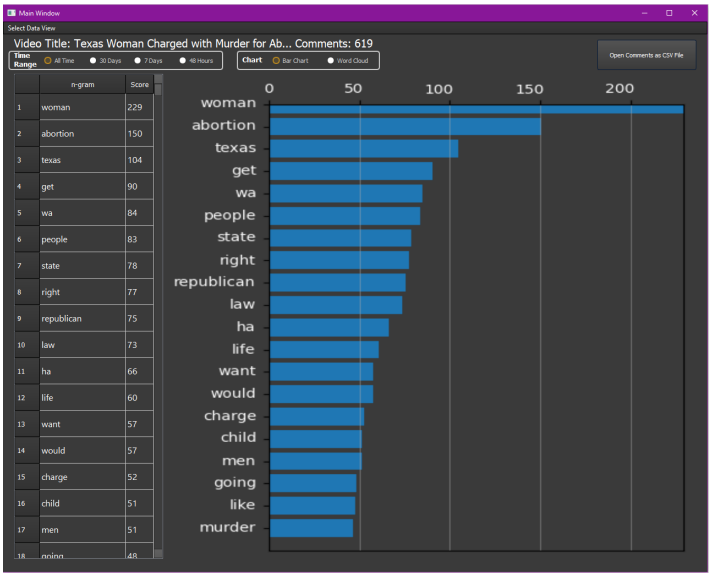



| | Comment | Time | Polarity | Sentiment | Subjectivity | Offensive? | Emotion |
|---|---|---|---|---|---|---|---|
| 0 | Lol this is so funny 😂 😂 😭 more to come hopefully. 😂 I hope she gets life. | 2022-04-18 03:47:12 | 0.9726 | positive | 0.733333 | Yes | amusement |
| 1 | Disgusting how America is going backwards in so many ways since DT.He has allowed this poison to spread.Abortion is as old as prostitution.Women lost their lives due to dirty back yard abortions or by trying to do it themselves.They don't use it as a form of birth control particularly since the pill and the day after pill.I am... | 2022-04-17 18:06:40 | 0.9684 | positive | 0.524542 | Yes | approval |
| 2 | Do we have a Constitution for a reason ?'No State shall make or enforce any law which shall abridge the privileges or immunities of citizens of the United States'"Constitution of United States of America 1789 (rev. 1992) "14th Amendment Section 1'All persons born or naturalized in the United States and subject to'the ... | 2022-04-17 00:55:52 | 0.9541 | positive | 0.319444 | Yes | neutral |
| 3 | It would great if the republikkkan party moves to Afghanistan, their draconian laws are against freedom, with them it seems as though it's open season on women, are we that far from burkhas, forced marriages and male chaperones? they are chipping away at citizens fundamental rights, it's very disheartening, is this truly freedom... | 2022-04-18 17:37:51 | 0.9536 | positive | 0.542857 | Yes | admiration |
| 4 | On Texas - if kind people living in strongly Democrat areas have the option, rather than having kind people leave Texas, surely it would be better for the kind people to leave Democrat areas in otherStates, to move to more marginally Republican parts of Texas. | 2022-04-17 14:04:50 | 0.9493 | positive | 0.760317 | Yes | curiosity |
| 5 | This is profoundly disgusting. As my 95 year old great grandmother said 30 years ago.."laws don't ban abortions, they just ban safe, legal abortions for women who don't have rich families." 'Another thing she told me was "a working man who votes for a republican is the same as a slave voting for his master." Great ... | 2022-04-17 09:28:45 | 0.9299 | positive | 0.651667 | Yes | admiration |
| 6 | It almost sounds like you're saying that this is some kind of 'Slippery Slope' going on here.. 🤣😂😂😂😂 | 2022-04-16 21:50:16 | 0.9156 | positive | 0.9 | Yes | neutral |
| 7 | This is so incredibly dystopian and disconcerting. | 2022-04-16 21:39:32 | 0.9 | positive | 0.9 | No | sadness |
| 8 | Just when you thought these issues had been put to rest, here they come again. Of course the rich will still be able to go to decent hospitals to get abortions. But, the rest will have to make due as best they can. You cannot sleep on these issues. We must get out and vote for progressive candidates. Otherwise this is the wor... | 2022-04-16 22:04:04 | 0.8945 | positive | 0.536111 | Yes | anger |
| 9 | sue abbot,sue Paxton,sue texas.what happened to freedom freedom freedom?????????Texas is a fucking hot rascist mess.insanity,rascist,so right wing it's become a parody. | 2022-04-17 00:13:27 | 0.885 | positive | 0.692857 | No | annoyance |
| 10 | What the hell happened to all those 'pro-choice' anti-maskers from a few months ago? I thought those guys had the Situation in hand? Because they seemed pretty keen on rights of autonomy at the time, right? 'RIGHT ?!? | 2022-04-16 22:13:07 | 0.8793 | positive | 0.542857 | No | confusion |
| 11 | Well Ive opined on this ... If Trumpers/Republicans are Soooo Pro Life? Then Lots of them use Fertility Clinics in Texas. Where 1000s of Frozen Embryos are created,stored. But those Pro Life Folks are NEVER going to implant,grow them all!! Isnt that 'MURDER' too? If those Frozen Embryos are used? Because based on their ... | 2022-04-16 22:54:53 | 0.8716 | positive | 0.3 | Yes | admiration |
| 12 | The Christian hypocrisy is that the Bible actually condones abortion. But of course, consistency and intellectual honesty is not compatible with their worldview. | 2022-04-17 11:22:24 | 0.8674 | positive | 0.166667 | No | neutral |
| 13 | OH GOOD GOD! So, I'm assuming the state of TEXAS has volunteered to support the mother and her child financially until the kid is 18? | 2022-04-17 04:01:13 | 0.8576 | positive | 0.3 | Yes | neutral |

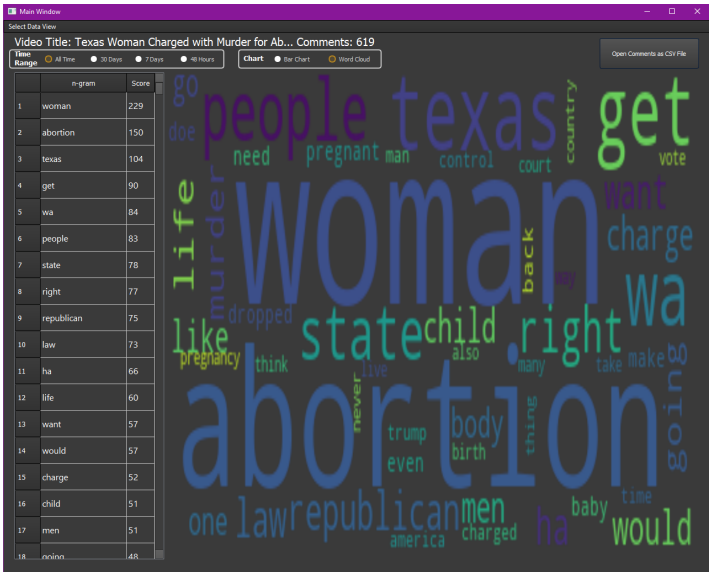
Figure 4.10: Comments screen

The list of comments screen in figure 4.10 allows the user to view the comments of the selected video all in one scrollable table. The user is able to sort the table by the most positive comments, the most negative comments, the most subjective comments, the least subjective comments, the most recently posted comments, and the oldest comments. The user can also filter the comments for specific time ranges.

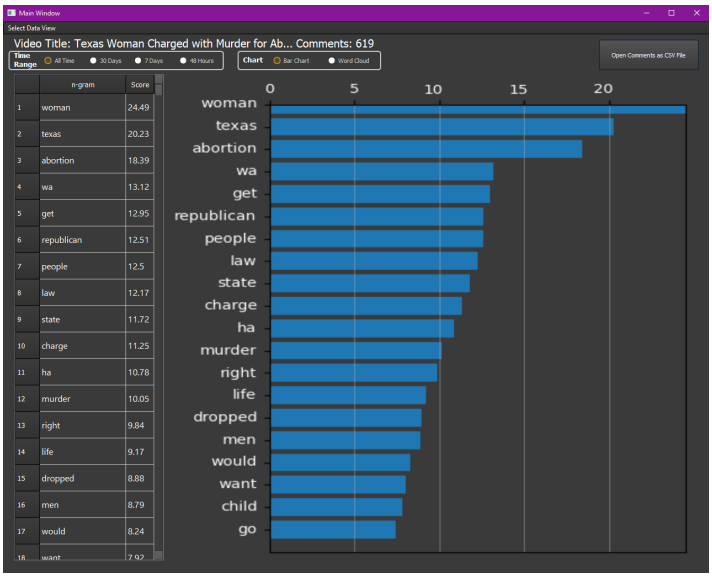
### Unigram Word Frequency & Importance



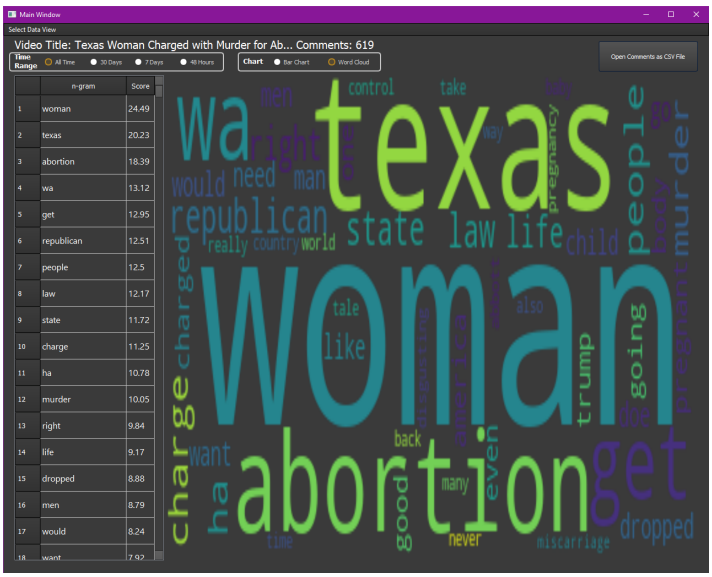

(a) Unigram frequency screen with bar chart




(b) Unigram frequency screen with word cloud




(c) Unigram importance screen with bar chart



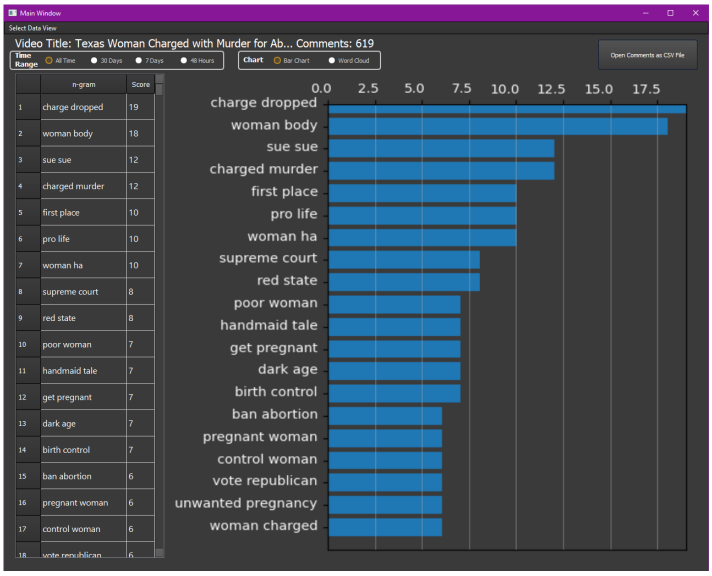

(d) Unigram importance screen with word cloud

Figure 4.11: Unigram word frequency and importance screens

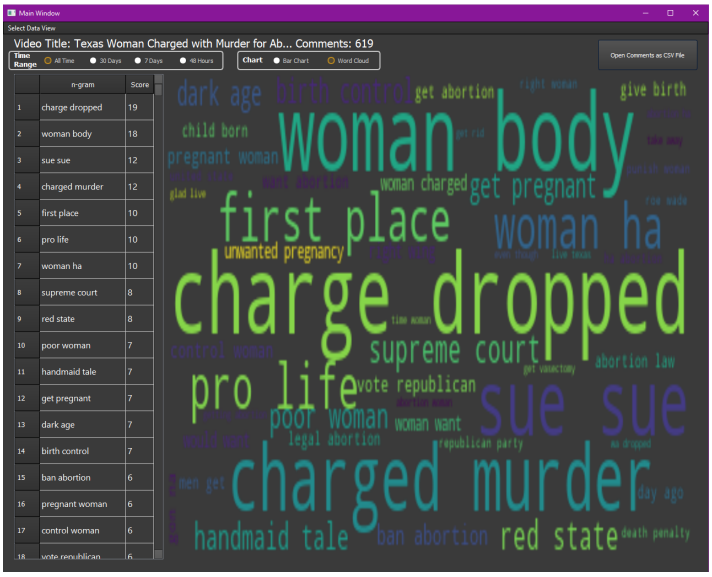
Figures 4.11 shows both the unigram count vectorized and TF-IDF vectorized data screens, meaning that the frequency and importance of each unigram is displayed. This is displayed as a table down the left side of the screen. The bar charts and word clouds representing the data in the table, created in the "**Main**" classes "*generate_charts*" function, are also displayed to the user. On each screen, the user is given an option to view either the bar chart, as seen in

figures 4.11a and 4.11c, or the word cloud, as seen in figures 4.11b and 4.11d. The user is able to filter the data for the specified time ranges.

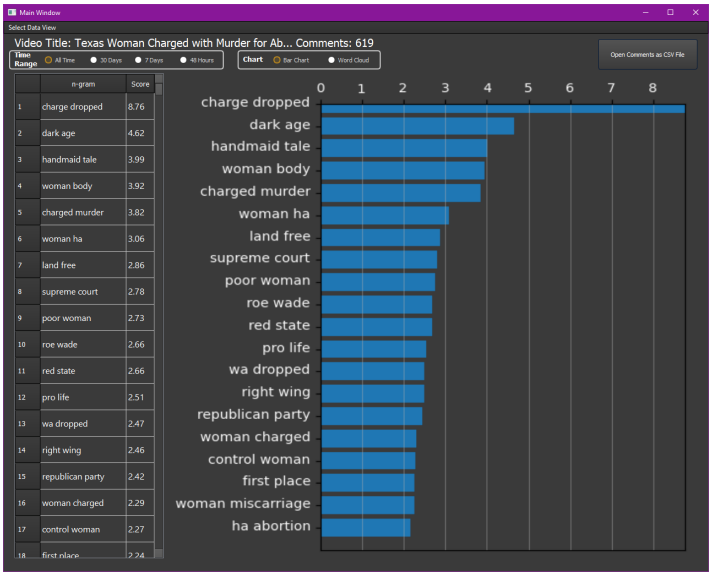
### Bigram Word Frequency & Importance



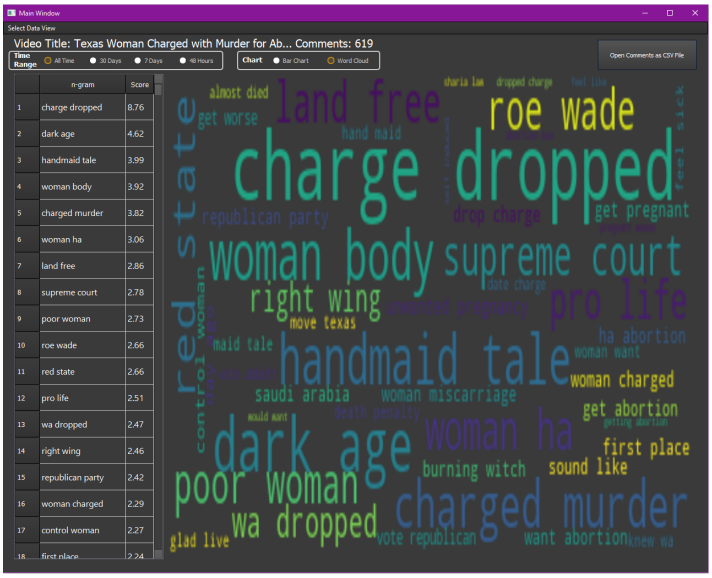

(a) Bigram frequency screen with bar chart




(b) Bigram frequency screen with word cloud




(c) Bigram importance screen with bar chart



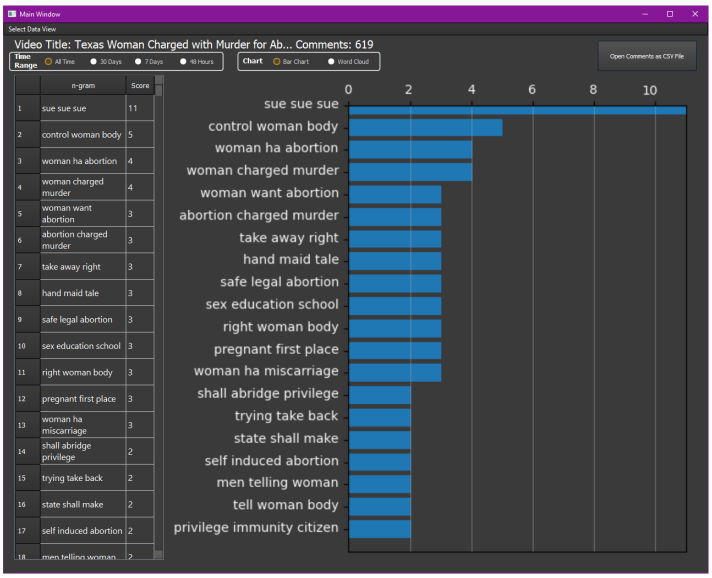

(d) Bigram importance screen with word cloud

Figure 4.12: Bigram word frequency and importance screens

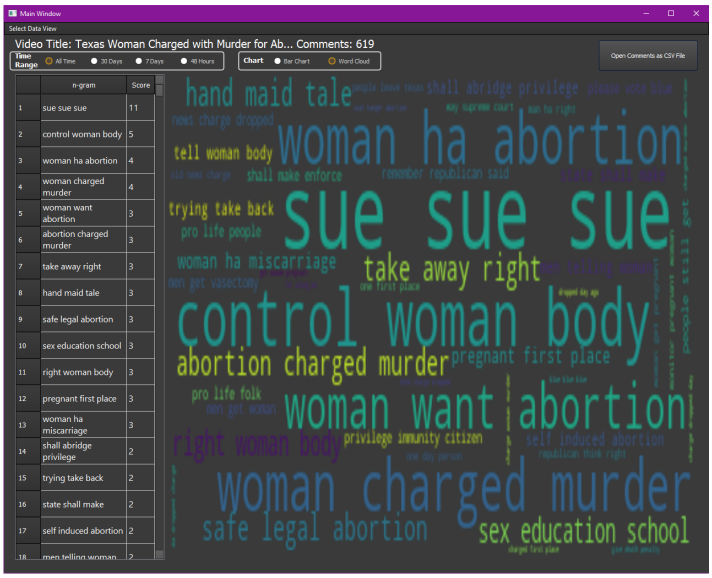
The bigram frequency and importance screens, displayed in figures 4.12, are very similar to the unigram screens, except this time bigrams have been used instead of unigrams during the

vectorization, meaning that all two-consecutive-word pairs have been counted and had their importance calculated. All other functionality of the screen stays the same. The time range filters and bar chart/word cloud buttons operate in the exact same way.

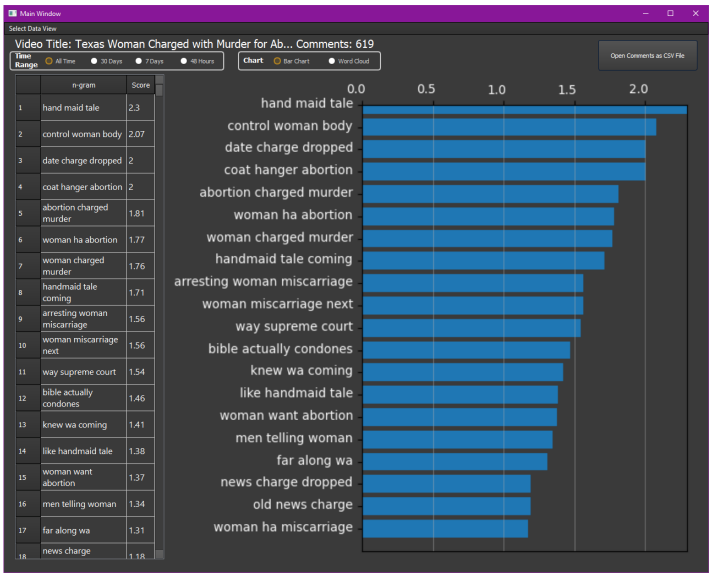
### Trigram Word Frequency & Importance



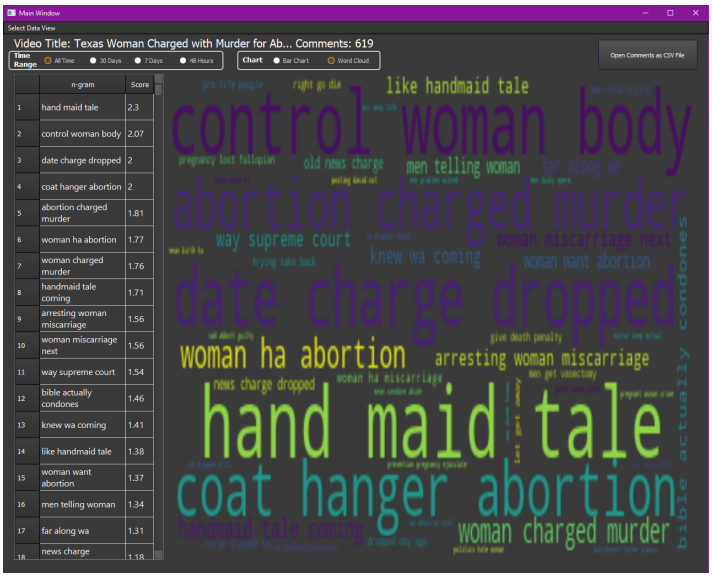

(a) Trigram frequency screen with bar chart




(b) Trigram frequency screen with word cloud




(c) Trigram importance screen with bar chart




(d) Trigram importance screen with word cloud

Figure 4.13: Trigram word frequency and importance screens

The trigram frequency and importance screens, displayed in figures 4.13, are very similar to the unigram and bigram screens, except this time trigrams have been used during the vectorization, meaning that all three-consecutive-word pairs have been counted and had their importance calculated. All other functionality of the screen stays the same, meaning that the time range filters and bar chart/word cloud buttons operate in the exact same way.

## 4.8 Building the Executable

After finishing the application, PyInstaller [6] was used to convert the Python code into an executable file.

A script file (.bat) was written, titled "YouTube Comments Analyser", that creates a resources folder, duplicates "*main.exe*, then runs the command "
resources
main.exe". This file could now be opened to start the application.

The time taken to launch the application was around 30 seconds on my machine, which is not optimal. Multiple Python-to-executable libraries were tested; PyInstaller was the library that produced the fastest opening executable file.

[6] https://pyinstaller.org/

# Chapter 5

# Results & Evaluation

This chapter evaluates the application by testing the application with five hand-picked videos to show some examples of the application's normal use cases and limit test the application. The requirements are then listed, where each requirement is discussed to see if it has been achieved.

## 5.1 Results

Each of the subsections in this section documents a category of videos. The first section analyses a video that is not very effectively processed by the system. The second section analyses a video that is effectively analysed by the system. The third section analysis three music videos used to test the limits of the system, as well as show the similarities between the audience's commenting behaviours. Additionally, the first two sections analyse political videos, while the third section analyses music videos. Different categories of videos were selected to fully test the system's performance on a variety of videos.

All of the videos used in this section have been loaded into and stored within the application. Any of them can be opened using the "Import from CSV files" option within the application - this is encouraged when reading this section for a greater degree of insight/data. Additionally, instead of using my application, the relevant CSV files can also be opened in excel - these are located in the "/Python/Videos" folder. Most of the frequency and importance screens have been displayed in their bar chart form, with most of the word clouds being omitted, as they would have taken too much space just displaying the exact same data in a different format.

All dates, view counts, comment counts, and subscriber counts are accurate at the time of publishing of the report but may have changed since. As a note, when "time ranges" are men-

tioned, this is referring to the four time ranges used throughout the project when cataloguing comments, those being comments posted from the last 30 days, 7 days and 48 hours, as well as across the whole lifespan of the video. When talking about the audience with respect to a video in this chapter, this refers to the people who have watched and posted a comment on said video.

Google gives users a quota of 10,000 units per day per Google Cloud project. A google cloud project was created for this application - this is described in section 4.1. Downloading 20 comments from one video uses 1 quota unit. Therefore, a limit of 200,000 comments can be downloaded per day per project. This would not be a problem for the majority of content creators, who most likely do not want to analyse more than 200,000 comments per day. However, situations, where analysts may want to process such an amount of comments, are conceivable. If this limit is reached, the method described in section 4.1 as well as in the user guide can be used to create a new API key and reset the limit.

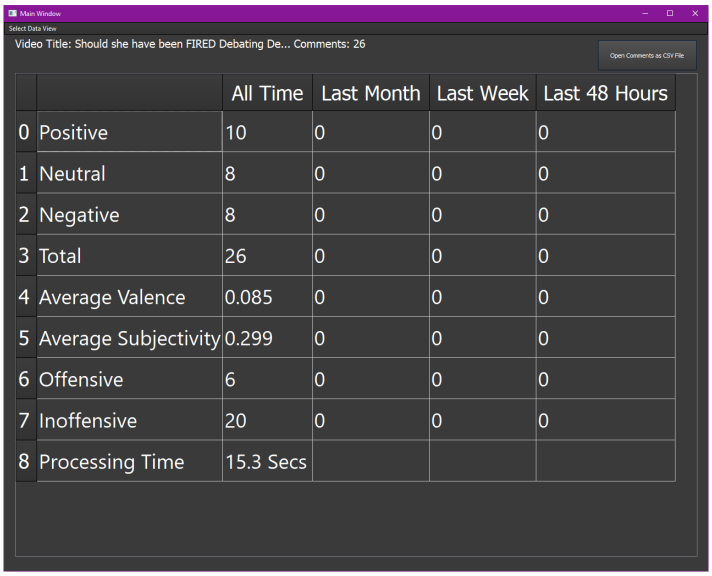
When discussing the amount of time that the application takes to process a video's comments, the spell check is always disabled. Additionally, the testing is being done with the final executable of the application on a machine with 32GB of RAM and an AMD Ryzen 5 1500X 3.5 GHz quad-core processor. As a result, processing the test videos used in this section on a different machine with different specifications may cause significant changes to processing times.

| **No.** | **Title** | **Creator** | **Views** | **Main Comments** | **URL** | **Sentiment Assigned** | **Emotion Assigned** | **Offence Assigned** |
|---|---|---|---|---|---|---|---|---|
| 1 | Should she have been FIRED? – Debating Destiny | Bastiat | 4000 | 26 | `https://www.youtube.com/watch?v=KlbDJcsNTNo` | 69% | 31% | 23% |
| 2 | Progressive Appears on Daily Wire, It Doesn't Go Well | David Pakman | 340,000 | 4138 | `https://www.youtube.com/watch?v=dwOSlwtyA_M` | 87% | 65% | 43% |
| 3 | Mask Off | Future | 535,000,000 | 153,359 | `https://www.youtube.com/watch?v=xvZqHgFz51I` | 56% | 33% | 13% |
| 4 | Hurt | Johnny Cash | 153,000,000 | 38,351 | `https://www.youtube.com/watch?v=8AHCfZTRGiI` | 74% | 56% | 29% |
| 5 | Cold Heart (PNAU Remix) | Elton John, Dua Lipa | 241,000,000 | 26,828 | `https://www.youtube.com/watch?v=qod03PVTLqk` | 68% | 49% | 14% |

Table 5.1: Music videos used to test the application

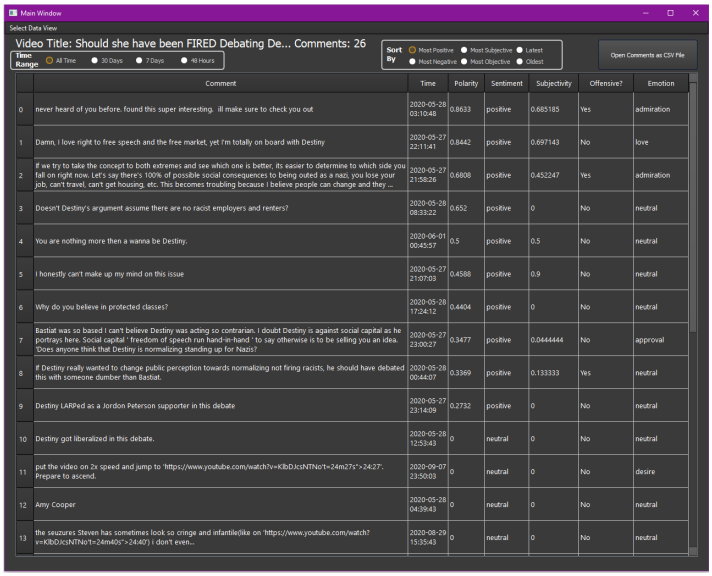
Table 5.1 shows all five of the videos used to test the application. The "Sentiment Assigned", "Emotion Assigned" and "Offence Assigned" columns documents the percentage of comments that the system was able to assign a non-neutral sentiment, a non-neutral emotion category and an offensive label to respectively. Each of these percentages were manually calculated by observing the statistics on each of the music videos report pages.

### 5.1.1 First Video



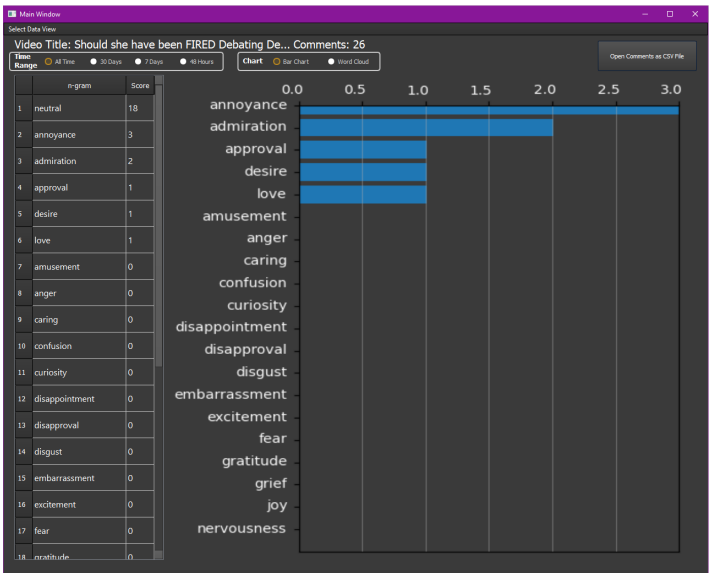

| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 10 | 0 | 0 | 0 |
| 1 | Neutral | 8 | 0 | 0 | 0 |
| 2 | Negative | 8 | 0 | 0 | 0 |
| 3 | Total | 26 | 0 | 0 | 0 |
| 4 | Average Valence | 0.085 | 0 | 0 | 0 |
| 5 | Average Subjectivity | 0.299 | 0 | 0 | 0 |
| 6 | Offensive | 6 | 0 | 0 | 0 |
| 7 | Inoffensive | 20 | 0 | 0 | 0 |
| 8 | Processing Time | 15.3 Secs | | | |

(a) Report screen



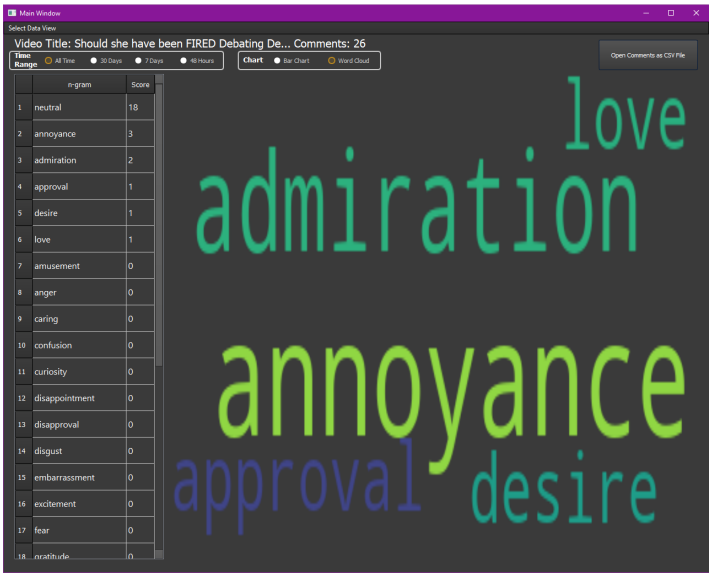

| | Comment | Time | Polarity | Sentiment | Subjectivity | Offensive? | Emotion |
|---|---|---|---|---|---|---|---|
| 0 | never heard of you before. found this super interesting. ill make sure to check you out | 2020-05-28 03:10:48 | 0.8633 | positive | 0.685185 | Yes | admiration |
| 1 | Damn, I love right to free speech and the free market, yet I'm totally on board with Destiny | 2020-05-27 22:11:41 | 0.8442 | positive | 0.697143 | No | love |
| 2 | If we try to take the concept to both extremes and see which one is better, its easier to determine to which side you fall on right now. Let's say there's 100% of possible social consequences to being outed as a nazi, you lose your job, can't travel, can't get housing, etc. This becomes troubling because I believe people can change and they ... | 2020-05-27 21:58:26 | 0.6808 | positive | 0.452247 | Yes | admiration |
| 3 | Doesn't Destiny's argument assume there are no racist employers and renters? | 2020-05-28 08:33:22 | 0.652 | positive | 0 | No | neutral |
| 4 | You are nothing more then a wanna be Destiny. | 2020-06-01 00:43:57 | 0.5 | positive | 0.5 | No | neutral |
| 5 | I honestly can't make up my mind on this issue | 2020-05-27 21:07:03 | 0.4588 | positive | 0.9 | No | neutral |
| 6 | Why do you believe in protected classes? | 2020-05-28 17:24:12 | 0.4404 | positive | 0 | No | neutral |
| 7 | Bastiat was so based I can't believe Destiny was acting so contrarian. I doubt Destiny is against social capital as he portrays here. Social capital ' freedom of speech run hand-in-hand ' to say otherwise is to be selling you an idea. 'Does anyone think that Destiny is normalizing standing up for Nazis? | 2020-05-27 23:00:27 | 0.3477 | positive | 0.0444444 | No | approval |
| 8 | If Destiny really wanted to change public perception towards normalizing not firing racists, he should have debated this with someone dumber than Bastiat. | 2020-05-28 00:44:07 | 0.3369 | positive | 0.133333 | Yes | neutral |
| 9 | Destiny LARPed as a Jordon Peterson supporter in this debate | 2020-05-27 23:14:09 | 0.2732 | positive | 0 | No | neutral |
| 10 | Destiny got liberalized in this debate. | 2020-05-28 12:53:43 | 0 | neutral | 0 | No | neutral |
| 11 | put the video on 2x speed and jump to 'https://www.youtube.com/watch?v=KlbDJcsNTNo?t=24m27s'>24:27'. Prepare to ascend. | 2020-09-07 23:50:03 | 0 | neutral | 0 | No | desire |
| 12 | Amy Cooper | 2020-05-28 04:39:43 | 0 | neutral | 0 | No | neutral |
| 13 | the seuzures Steven has sometimes look so cringe and infantile(like on 'https://www.youtube.com/watch?v=KlbDJcsNTNo?t=24m40s'>24:40') i don't even... | 2020-08-29 15:35:43 | 0 | neutral | 0 | No | neutral |

(b) Comments screen




| | n-gram | Score |
|---|---|---|
| 1 | neutral | 18 |
| 2 | annoyance | 3 |
| 3 | admiration | 2 |
| 4 | approval | 1 |
| 5 | desire | 1 |
| 6 | love | 1 |
| 7 | amusement | 0 |
| 8 | anger | 0 |
| 9 | caring | 0 |
| 10 | confusion | 0 |
| 11 | curiosity | 0 |
| 12 | disappointment | 0 |
| 13 | disapproval | 0 |
| 14 | disgust | 0 |
| 15 | embarrassment | 0 |
| 16 | excitement | 0 |
| 17 | fear | 0 |
| 18 | gratitude | 0 |

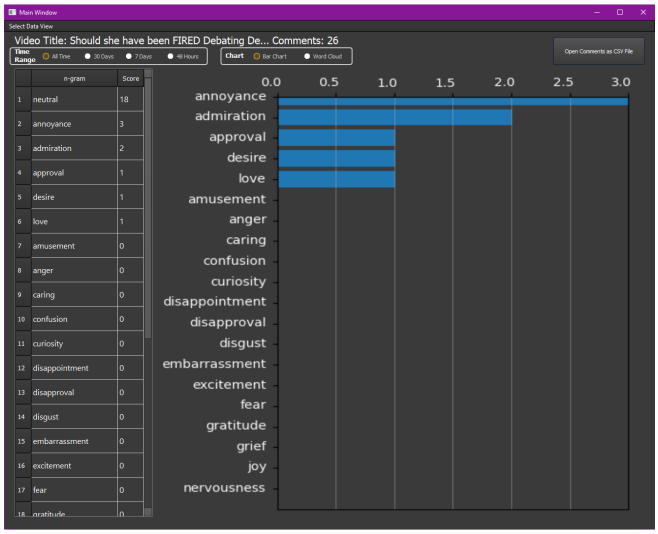
(c) Emotional analysis bar chart screen



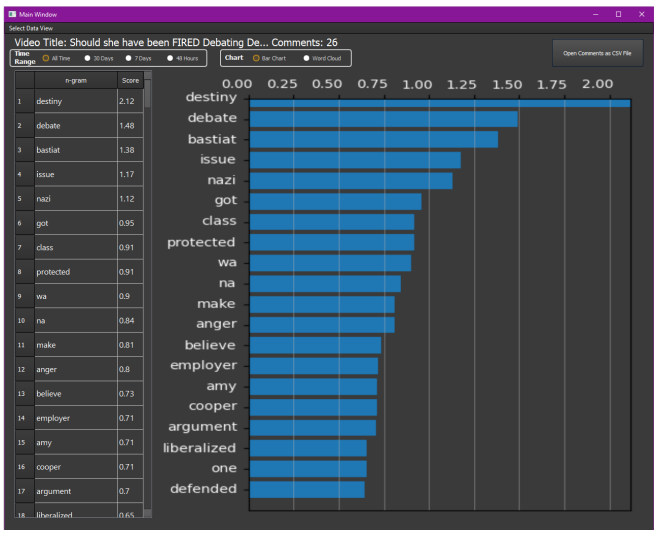

| | n-gram | Score |
|---|---|---|
| 1 | neutral | 18 |
| 2 | annoyance | 3 |
| 3 | admiration | 2 |
| 4 | approval | 1 |
| 5 | desire | 1 |
| 6 | love | 1 |
| 7 | amusement | 0 |
| 8 | anger | 0 |
| 9 | caring | 0 |
| 10 | confusion | 0 |
| 11 | curiosity | 0 |
| 12 | disappointment | 0 |
| 13 | disapproval | 0 |
| 14 | disgust | 0 |
| 15 | embarrassment | 0 |
| 16 | excitement | 0 |
| 17 | fear | 0 |
| 18 | gratitude | 0 |

(d) Emotional analysis word cloud screen

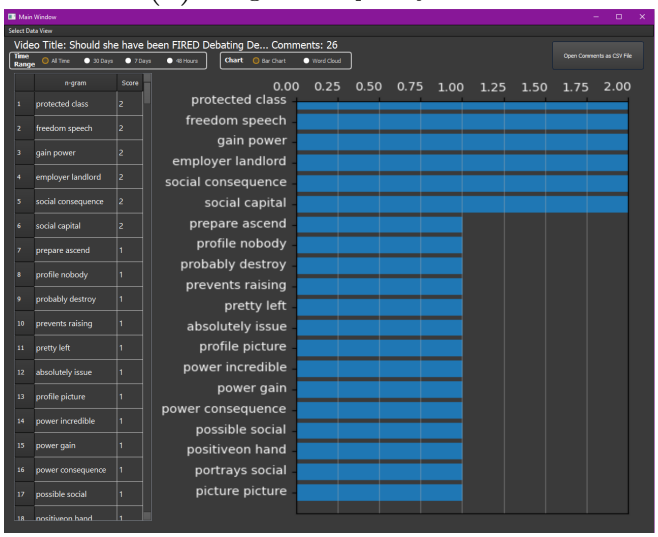
Figure 5.1: Test 1 GUI screens



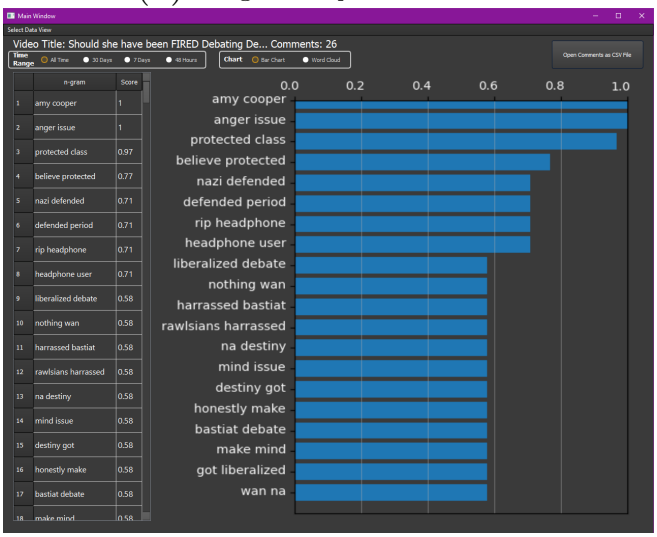

(a) Unigram frequency screen



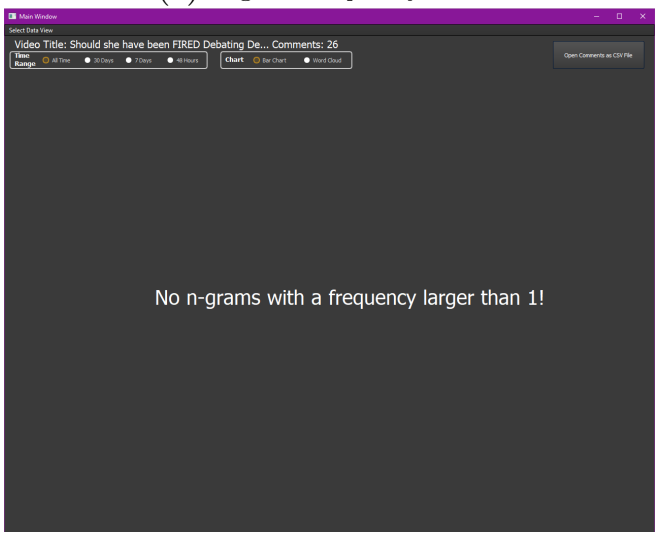

(b) Unigram importance screen



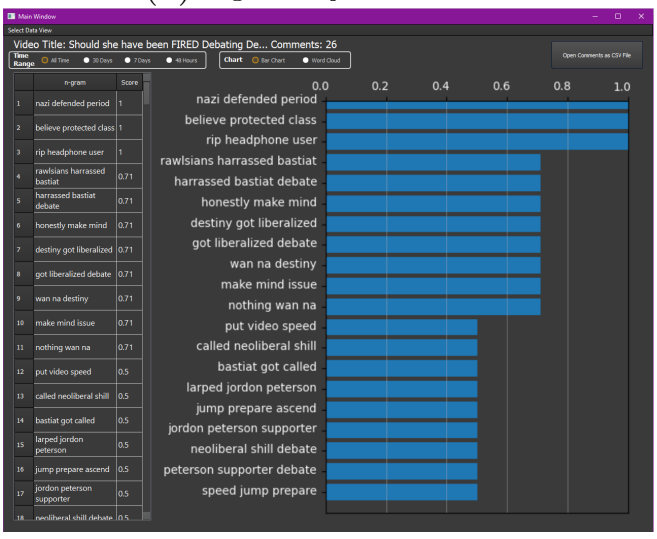

(c) Bigram frequency screen




(d) Bigram importance screen




(e) Trigram frequency screen




(f) Trigram importance screen

Figure 5.2: Test 1 GUI screens

The first test video is named ”Should she have been FIRED? – Debating Destiny”. This is a political video consisting of a discussion between two content creators surrounding the issues of managers and companies being able to fire their staff due to their political views.

This video is an example of a video that is not amazingly suited for my application for a number of reasons. The biggest issue is that the video is not very popular, summing just over 4000 views and 26 comments (not including reply-comments). This means that the system only has 26 comments to analyse, which is not a very rich dataset.

Another issue with this video is that it was posted on 27/05/2020, which is almost two years ago. As displayed in figure 5.1a, the report shows that the system was not able to find any comments that were posted in the three specific time ranges analysed across the project, meaning that all 26 comments were more than 30 days old. The latest comment was posted on 11/05/2021, almost a year ago. Old videos like this one that have not been commented on recently due to their post date do not provide any insight in terms of the time ranges, as all of the comments are older than 30 days. The application's time range filters when filtering the data are not needed for videos like this one.

On the other hand, even though the video only has 26 comments, the unigram frequency and importance screens, displayed in figures 5.2a and 5.2b, still do show some meaningful insight. The names of the two content creators, these being ”Destiny” and ”Bastiat”, are high up in the table of unigram frequencies, with Destiny being mentioned 14 times, while with Bastiat being mentioned 6 times, even though the video was posted on Bastiat's channel. This is probably because Destiny's channel has 419,000 subscribers [1], while Bastiat's only has 9000 subscribers [2], meaning that Destiny is a much more popular creator with a larger amount of fans. This insight shows content creators who their audience was more interested in discussing.

Looking at the bigram screens displayed in figures 5.2c and 5.2d, little meaningful insight can be gained here due to the limited dataset of 26 comments. 6 bigrams have been mentioned twice, meaning that the content creator would be able to get the general idea of the repeated topics from the frequency screen.

The trigram screens displayed in figures 5.2e and 5.2f give little insight, as none of the trigrams are used more than once in the dataset, resulting in no data being displayed in figure 5.2e. The trigram importance screen gives the user some insight into the unique trigrams mentioned in the dataset, meaning that the words used in the trigrams that are most important, ie the trigrams with a score of 1 in figure 5.2f, have not been used in any other comments. This

[1] https://www.youtube.com/user/destiny

[2] https://www.youtube.com/channel/UCHTnzqCT2rSs1$_2 HmZiVuRA$

is a common result in datasets with small amounts of comments.

The total processing time for this video, display in the report in figure 5.1a, is 15.3 seconds. This comprises the time taken to download, clean, and process the dataset. This relatively short processing time makes sense for a video with only 26 comments, and would likely not bother the user in any way, unlike the other test videos. This equates to roughly 1.7 comments processed per second.

### 5.1.2 Second Video

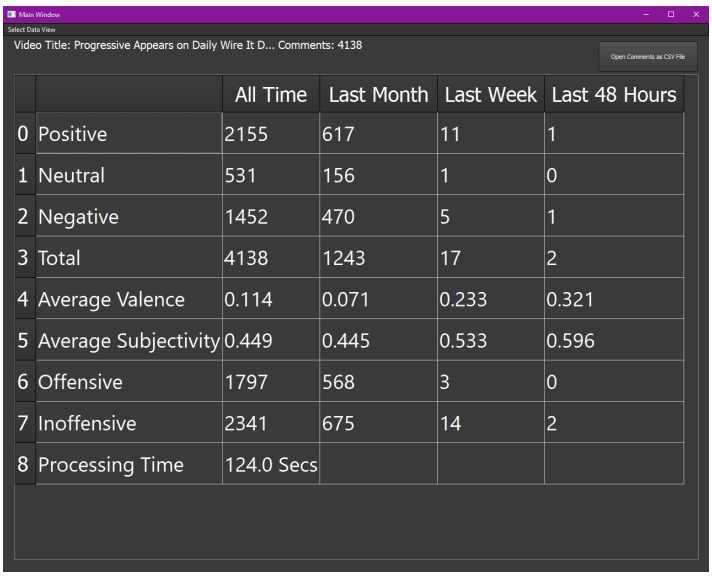
The second test video is named "Progressive Appears on Daily Wire, It Doesn't Go Well" [3]. This also happens to be a political video, and is an interview on a show named The Daily Wire between two political commentators: Michael Knowles and David Pakman. The conversation centers around the topic of a judge, Ketanji Brown Jackson, being nominated to the supreme court. The video has been posted on David's news channel.

This video, in comparison to the first one, is much more suitable for my application. This video has over 330,000 views and over 4000 comments, meaning that it is much more popular compared to the first test video used. Popular videos like this one are examples of videos where the time range filters are quite useful, as this video has comments posted from all of the time ranges used in the application. The user is able to view different comments and their n-gram data posted within these time ranges, allowing the user to distinguish between data from when the video was initially posted, likely from users who are subscribed to the channel and have their notification turned on, compared to data from users who are more likely to have seen the video as a result of a recommendation by YouTube's system or otherwise.

As shown in the report in figure 5.3a, the average valence of a comment within this dataset is 0.11. The system usually seems to assign an average valence between -0.2 and 0.2 on videos with a large enough dataset of comments. Videos with smaller datasets tend to have an average that skews one way or another, likely just due to chance because of the lack of data. Even though the system is able to pick out both extremely negative and positive comments, the average always seems to even out for the most part - this is true for test videos 2-5, those being the videos that have a rich dataset.

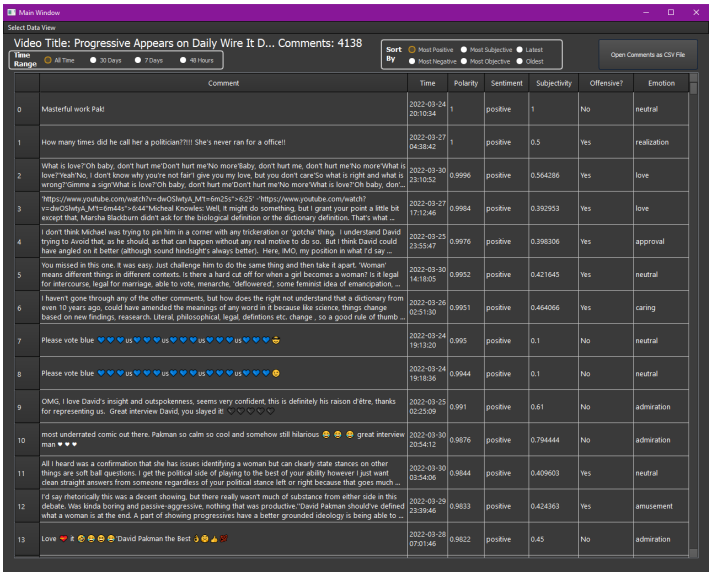
Out of the 4138 comments that were collected, the system was able to assign a sentiment to 3607 (87%) of them. The types of comments that were ranked the most negative and most positive tended to follow the same archetypes. Either the comment is a very short comment that

[3] https://www.youtube.com/watch?v=dwOSlwtyA$_M$

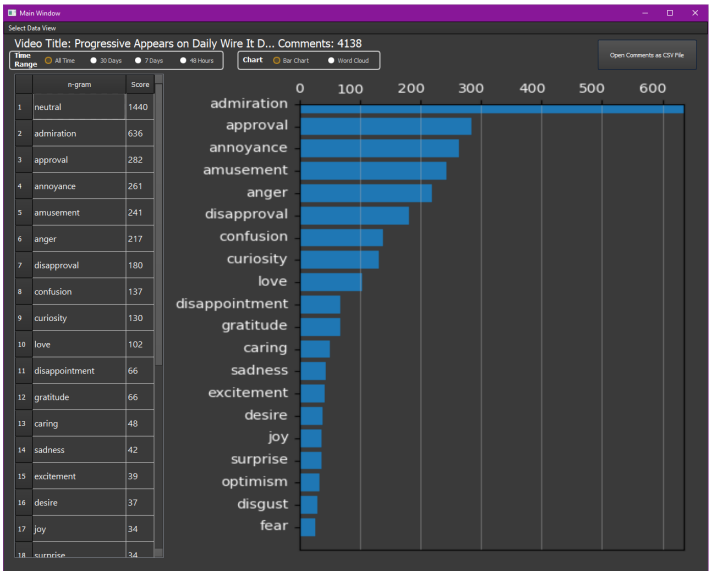



| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 2155 | 617 | 11 | 1 |
| 1 | Neutral | 531 | 156 | 1 | 0 |
| 2 | Negative | 1452 | 470 | 5 | 1 |
| 3 | Total | 4138 | 1243 | 17 | 2 |
| 4 | Average Valence | 0.114 | 0.071 | 0.233 | 0.321 |
| 5 | Average Subjectivity | 0.449 | 0.445 | 0.533 | 0.596 |
| 6 | Offensive | 1797 | 568 | 3 | 0 |
| 7 | Inoffensive | 2341 | 675 | 14 | 2 |
| 8 | Processing Time | 124.0 Secs | | | |

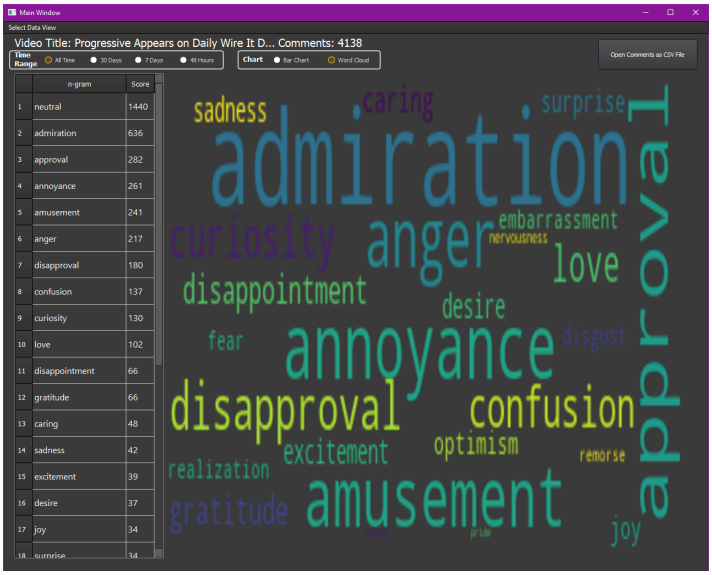
(a) Report screen




(b) Comments screen




(c) Emotional analysis bar chart screen

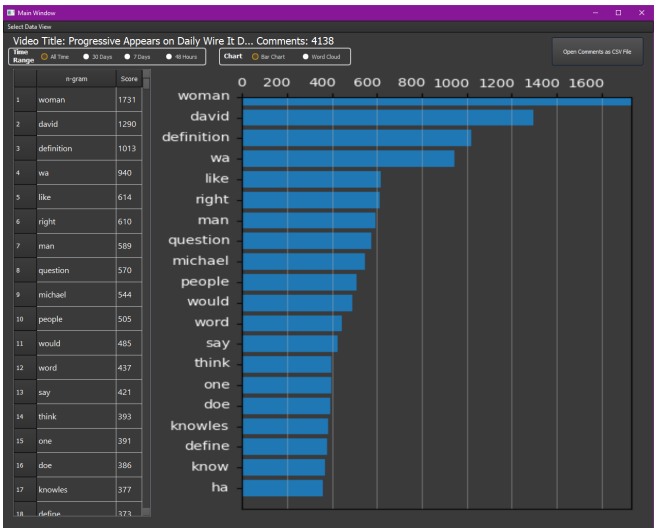



(d) Emotional analysis word cloud screen

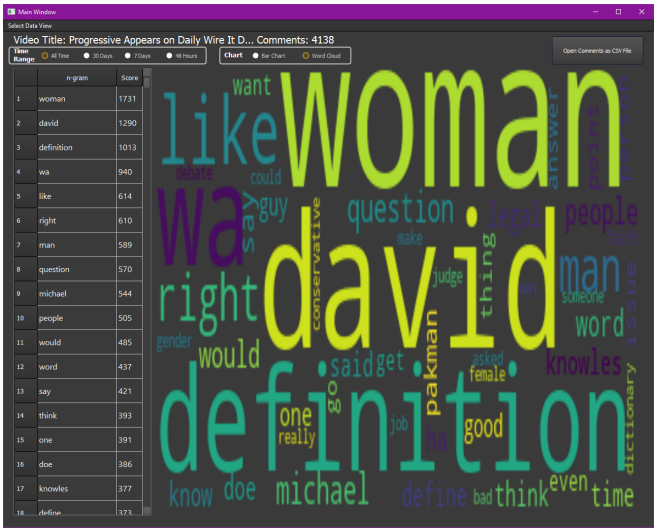
Figure 5.3: Test 2 GUI screens

only contained a few or even a single, as according to the system, extremely positive/negative word(s), or the comment is a lengthy paragraph containing many positive/negative words. These comments sometimes contained a large number of emojis. A few examples of the most positive and negative comments in this dataset have been displayed in table 5.2. Looking at comments like these, it seems that the system is able to accurately rank them, as these comments seem to match their polarity scores.

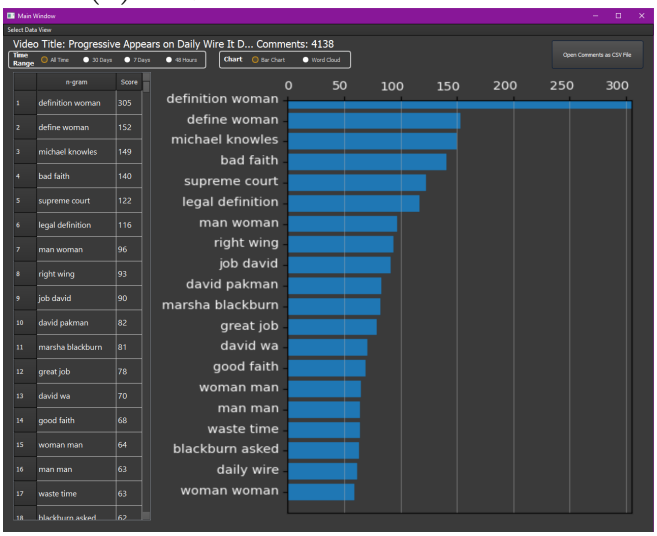
Comments that were not assigned a sentiment (assigned neutral), or otherwise comments



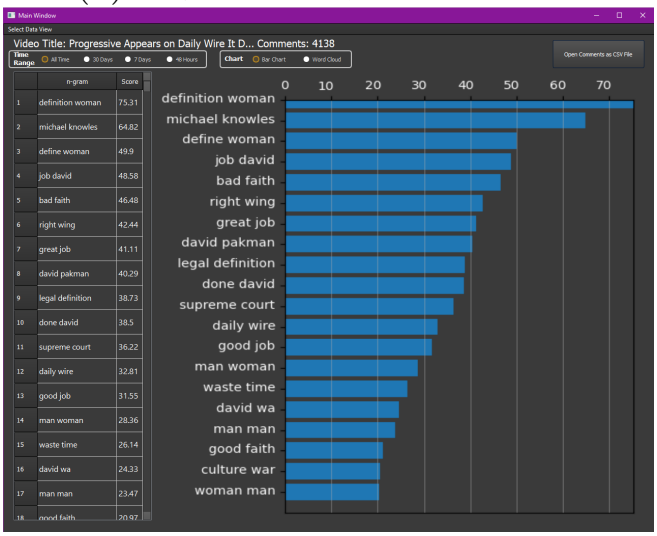

(a) Unigram frequency bar chart screen



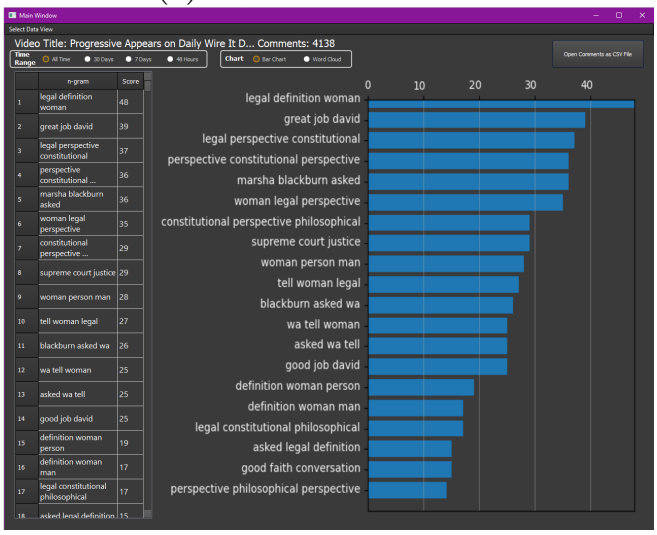

(b) Unigram frequency word cloud screen



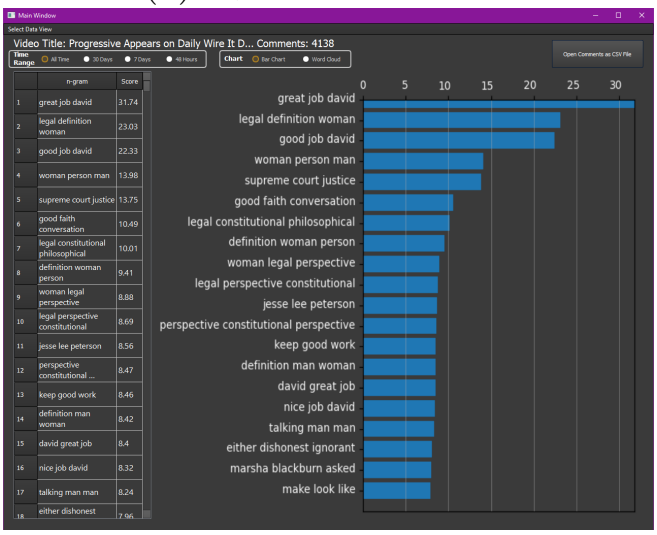

(c) Bigram frequency screen




(d) Bigram importance screen




(e) Trigram frequency screen




(f) Trigram importance screen

Figure 5.4: Test 2 GUI screens

| Comment | Polarity |
|---|---|
| Masterful work Pak! | 1 |
| Love 💝it 🤣😂🤣😂 Pakman the Best 😀👍💯 | 0.9822 |
| You missed in this one. It was easy. Just challenge him to do the same thing and then take it apart. 'Woman' means different things in different contexts. Is there a hard cut off for when a girl becomes a woman? Is it legal for intercourse, legal for marriage, able to vote, menarche, 'deflowered', some feminist idea of emancipation, having a Bat Mitzvah?" And 'the' dictionary definition? The fact that there are more than one dictionary, each containing more than one entry gives the lie to that. The debate of prescriptive/descriptive only exists in the public mind. Prescriptive went out with the Victorians and Confucius. Language does not have a spec (see the use of 'preferred' in the Chicago Manual)."And there are more than one type of dictionary– popular, medical, biological, anthropological. If you mean the colloquial definition, you'd better go look up the squishiness of the word Colloquial. As well as Dialect, Ethnicity, and I don't know, maybe Wittgenstein.' (*Comment goes on for another 542 words...*) | 0.9952 |
| Her 'base' ?!? She's not up for election | -1 |
| Progressives hate women nothing more. This is the ultimate form of misogyny. Allowing men to cosplay as women. Making mockery of women. Any woman that can just go along with this foolery that anything can be a woman dispite the biological difference you are part of the problem. Progressive policies will eventually undermine the same feminist that so called fought for these rights. Facts don't care about feelings sorry not sorry. A man can't be a woman and a woman can't be a man. You have gender disphoria you need help. Mutilating your bodies trying to indoctrinate children in your weird fantasies to get approval and validation of your existence. It's disgusting and down right evil. The same women who want to protect women's sports women's spaces are the same ones allowing it to get destroyed. You refuse to acknowledge facts over emotions. Hard data over how you feel one day or the next. | -0.9786 |
| Just for a change I'd like to hear one of these conservative Talking Heads talk about something other than culture issues. You know, policies. Financial policies, foreign policies, just any policies that they stand for other than these stupid culture War issues. But that's all they've got. They have no policies, and frankly I'm sick of hearing about the culture War. I just don't care about it. It's all really quite meaningless. | -0.9734 |

Table 5.2: Examples of some of the most positive and negative comments from the second test video

with a polarity score of 0, tended to also have a subjectivity score of 0. This is because the lexicon-based methods in both cases could not find enough meaningful features in these comments to assign a score. The average subjectivity of a comment within this dataset is 0.542, also around the middle point. Both the most subjective and least subjective comments tended to be the shorter, single-sentence comments.

The emotional analysis screens in figures 5.3c and 5.3d show us that the overwhelming emotions displayed by the audience are admiration, approval, annoyance, amusement, and anger. The system was able to assign an emotion to 2698 (65%) of the comments.

Various topics have been discussed in the 11-minute video. These included the supreme court, transgender issues, the definition of a woman, bad faith actors, legal definitions, the right wing (politically), and Marsha Blackburn (member of the US senate) and Kentaji Brown Jackson (the judge being nominated), the two people involved in the particular topic being

discussed. The definition of a woman is the most discussed topic of the video and is brought up multiple times. These topics can be observed in the n-gram screens, where the user is able to see what subjects their audience has been discussing the most.

The unigram frequency screens, displayed in figures 5.4a and 5.4b, show that the unigram "woman" was the most mentioned unigram in the dataset, which matches up with the content of the video where the definition of a woman was brought up multiple times. The second most frequent unigram "david" references the name of the content creator, which adds up with expectations. The other unigrams on this screen can be used to gain a general understanding of what was discussed by the audience but fail to give specific topics. Additionally, some of the most mentioned unigrams (sometimes in their lemmatized forms), such as "wa", "like", and "question", do not really give the user a great insight into their audience's discussions. This seems to be a common occurrence in videos that discuss topics in greater detail, where the comments posted are quite lengthy.

The bigram screens, displayed in figures 5.4c and 5.4d, are more useful for analysing which topics the audience was talking about the most. Once again, the first "definition woman" bigram, as well as the second "define woman" bigram, ranked the highest by quite a large margin, further backing up the fact that the definition of a woman was the most interesting and discourse-provoking topic in the video according to the audience. The third bigram "michael knowles" actually ranked a lot higher than the bigram "david pakman", suggesting the audience was eager to discuss this commentator. According to figure 5.4c, the user is able to see that after the definition of a woman topic, the next most discussed topics were the "bad faith" topic, the "supreme court", "legal definition(s)", transgender issues (based on the bigram "man woman"), the "right wing", "Marsha Blackburn", and so on. This is a great indicator of which topics the audience is most interested in, and can certainly be used by this content creator to help plan out future video topics. As well as this, figure 5.4d shows us that the bigram "job david" was actually the most important bigram in the dataset, even though it ranked eighth in the frequency data, suggesting that it has been mentioned in a large number of unique comments in the dataset. This example emphasizes the differences in and importance of performing both count and TF-IDF vectorization. The bigram "job david" likely stems from the trigrams "good job david", "great job david", and similar trigrams, which can be confirmed by looking at the trigram importance screen in figure 5.4f, and is discussed in the next paragraph.

The trigram screens, displayed in figures 5.4e and 5.4f, also give a great insight into the topics discussed by the audience. The third trigram "legal definition woman" also backs up the

fact that the definition of a woman is the most interesting topic in the video according to the audience. Figure 5.4e also gives the user some insight into the specific topics most frequently discussed, such as the "legal perspective (constitutional)" on the matter, what "Marsha Blackburn (was) asked", the philosophical and legal perspectives on this issue, the "supreme court justice", and so on. This would allow the content creator to choose what specifically they would want to talk about in their future videos based on the interests of their audiences. As touched upon earlier, figure 5.4f shows that the most unique trigram per each individual comment were the "great job david" and "good job david", likely posted by people who wanted to congratulate the content creator for completing the interview well. This also matches the emotional analysis, where the most exhibited emotions in the dataset were admiration and approval, as discussed earlier.

Removal of duplicate comments has been discussed in section 4.2.11, where a decision was made to keep duplicate comments. This video is both an example of why keeping duplicates is important, but also why they can add unnecessary noise to the data. The trigrams "good job david" and "great job david" would not appear as high in the importance rankings in figure 5.4f if duplicates were removed, which would make the results less accurate, as based on the trigram frequency screen in figure 5.4f, the trigram "great job david" has been used 39 times, while the trigram "good job david" has been used 25 times.

Additionally, the use of a political video hopefully increases the variety in the sentiment/polarity of the comments. This is backed up by the video's high percentage of offensive comments as displayed in table 5.1 - 43%, compared to the other music videos. Additionally, the system was able to assign the highest percentage of comments into sentiment categories from this video compared to the music videos.

The total processing time for this video, as displayed in figure 5.3a, is 124 seconds, which is much more substantial compared to the first test video. This is obviously because of the 4138 comments that need to be processed.

### 5.1.3 Music Videos (videos 3, 4, 5)

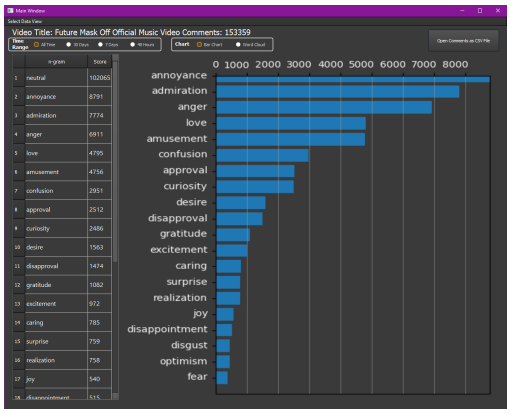
The next three videos used to test the application are music videos.

The first video is a very popular music video by Future, totalling more than 500 million views. On the video's web page, the number of comments posted sums up to more than 200,000. However, after downloading the dataset, it appears that only 153,359 of those comments are main comments (as opposed to reply comments). Although the largest majority of content

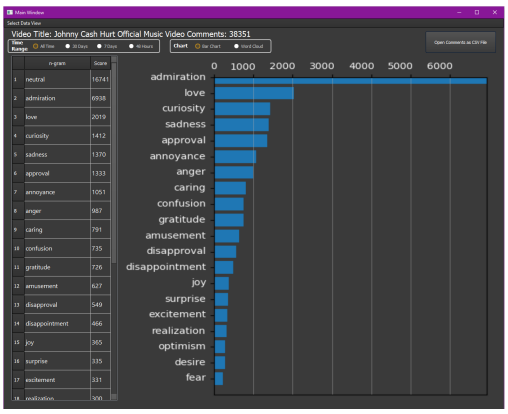
creators or even YouTube users would not need to analyse the comments of such a popular video, this video has been used as a limit test on the system and harbours the largest amount of comments out of all of the test videos. It was mentioned previously that the maximum number of comments that could be downloaded in one day is 200,000 (a Google Cloud services policy); even though the total amount of comments on this video exceeds this limit, our system only downloads main comments, so this video is still able to be processed.

Large videos like these ones are the reason that a limit on the number of features, or n-grams, was enforced during the count vectorization and TF-IDF vectorization, as discussed in section 4.4. Initially, when this video was input into the application, the machine running the application ran out of memory, causing the application to crash. A feature limit was decided upon and set, whereafter no memory issues were encountered.

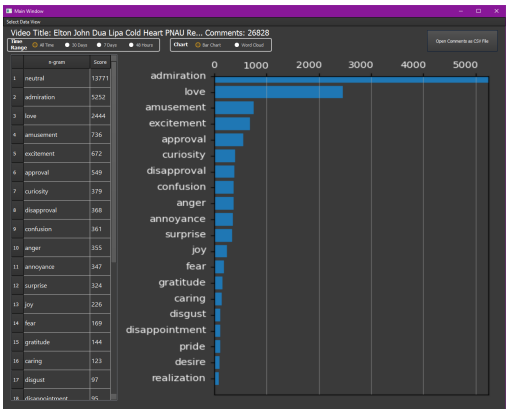
The total processing time for this first video is 2785 seconds or 46 minutes; this comes out to roughly 55 comments per second. This is an extreme amount compared to any of the other test videos and obviously results from the large number of comments. This video shows that even videos with over 150,000 comments can be successfully processed by my application, all be it over a large amount of time. When attempting to navigate to the list of comments screen, after pressing the respective menu button, the application freezes and takes around 10 seconds to load the comments into the table before being able to display them to the user. After this short amount of time, the application is successfully able to load and display the comments, where the user is able to scroll through the list of comments without any issues. After pressing any of the sort by options, the application takes a further couple of seconds before the comments are sorted and displayed. This is not optimal, but acceptable.




(a) Test 3 emotional analysis bar chart screen



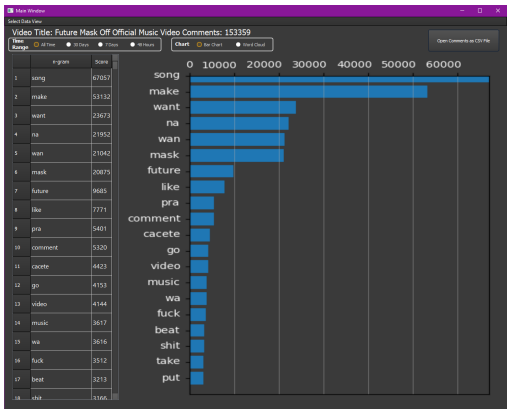

(b) Test 4 emotional analysis bar chart screen



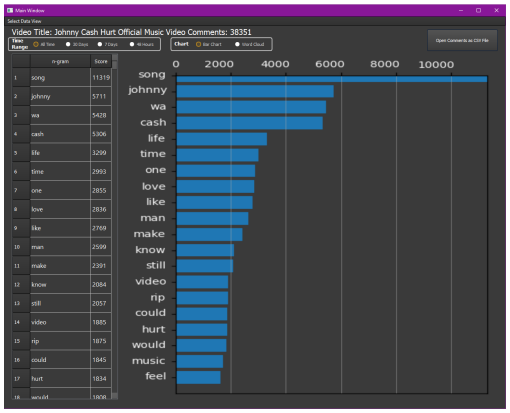

(c) Test 5 emotional analysis bar chart screen

Figure 5.5: Test 3, 4, 5 emotional analysis screens

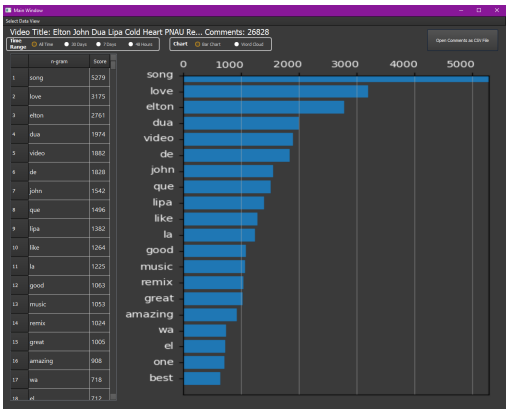
The emotional analysis screens of each of the music videos are displayed in figures 5.5. Videos 4 and 5 harbour similar emotions, with ”admiration” and ”love” being the top two most detected emotions by quite a large margin. This matches with the themes of these two songs, where the predominant comments are usually comments that praise the artists. Additionally, the artist Johnny Cash passed away in 2003, which matches the frequency of comments labelled with the ”admiration” category. Johnny Cash and Elton John are greatly respected people. On the other hand, the hip-hop music video ”Mask Off” gets a lot more mixed emotion categories.




(a) Test 3 unigram frequency bar chart screen




(b) Test 4 unigram frequency bar chart screen




(c) Test 5 unigram frequency bar chart screen

Figure 5.6: Test 3, 4, 5 unigram frequency screens

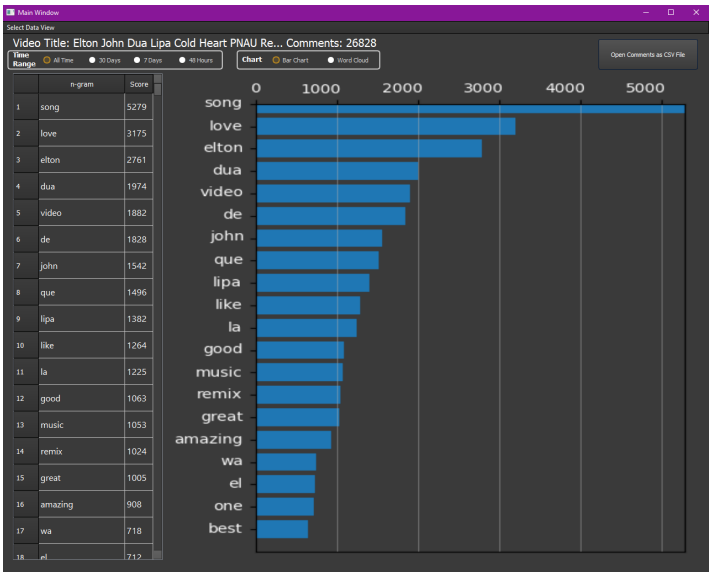
The frequency and importance screens for these music videos seem to contain repeating factors. For example, the majority of the time the word ”song” is the most mentioned and most important unigram in these datasets, as seen in figures 5.6, which is used in comments like the following examples taken from the dataset:

- *This song is and will be legendary for ever!*
- *i dont think that old school song fits this new school vid*

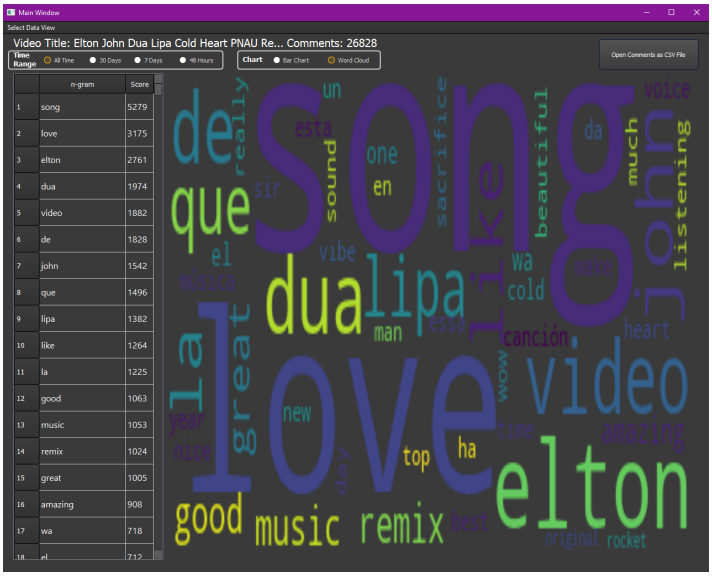
This makes sense, as all of these videos are music videos, but is not very useful to the content creators.

Another reoccurring theme is that some of the most mentioned n-grams tend to be the artist’s names, which would help analysts understand who the audience was most excited to talk about. Video 5’s unigram frequency screen in figure 5.7 shows us that the audience was more interested in mentioning Elton compared to Dua. This fact is also backed up by the bigram frequency screen displayed in figures 5.7.

A further reoccurring theme in music videos is the fact that the audience loves mentioning

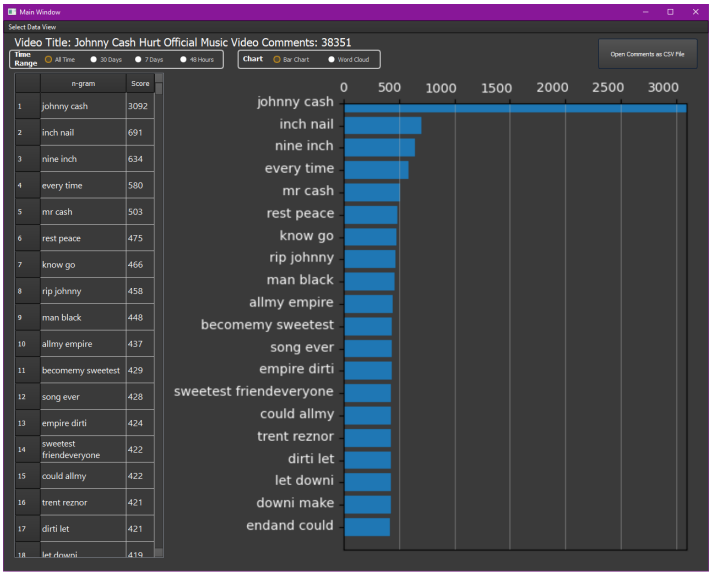



(a) Test 5 unigram frequency bar chart screen

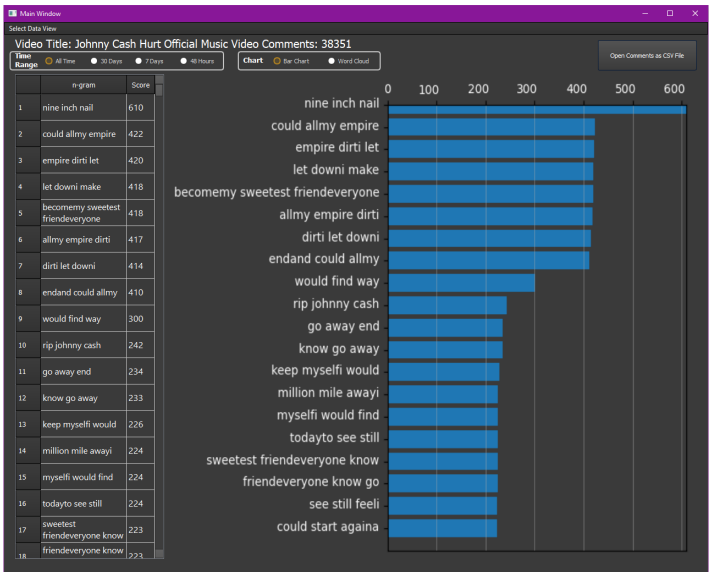



(b) Unigram frequency word cloud screen

Figure 5.7: Test 5 unigram word frequency screens




(a) Test 4 bigram frequency screen




(b) Test 4 trigram frequency screen

Figure 5.8: Test 4 bigram and trigram word frequency screens

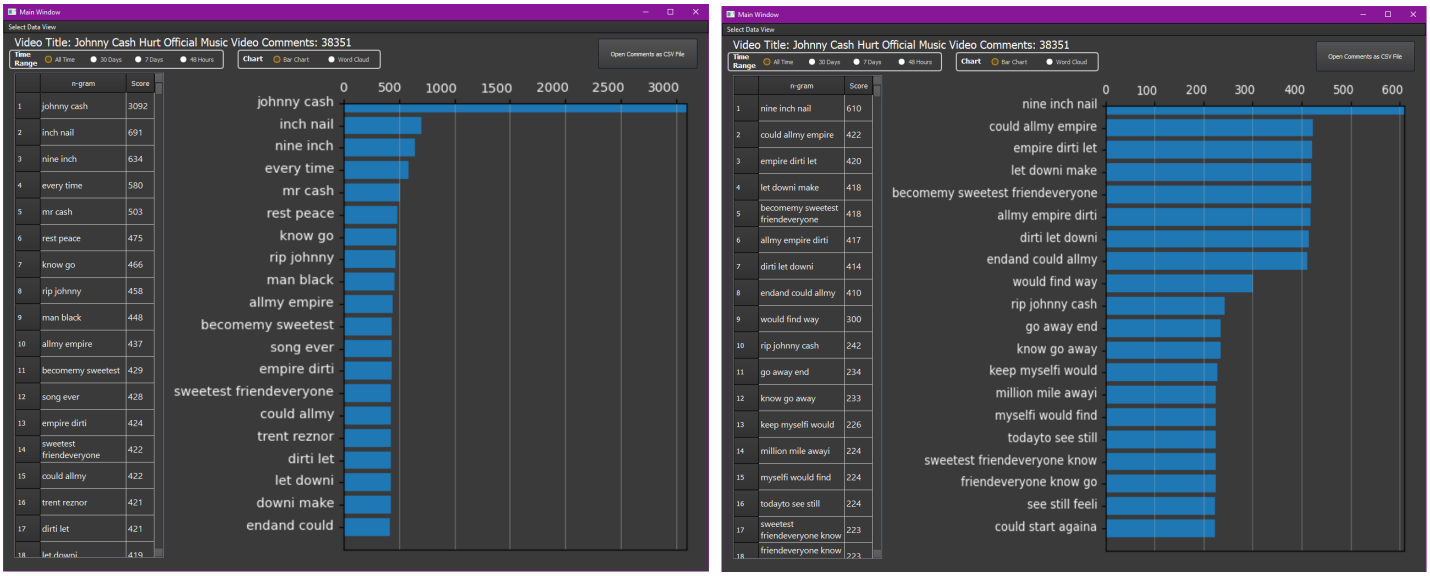
the song's lyrics in their comments. This can also be helpful to analysts, as the most popular/exciting verses in a song can be discovered through the n-gram frequency screens. Topics that these verses touch upon can then be incorporated into other songs. The bigram and trigram screens seem to be the most helpful screens when investigating what lyrics of a song are the most liked. For example, video 4's trigram frequency screen, as shown in figure 5.8b, shows that the audience seems to like the following phrases from the songs lyrics (with the missing

lyrics, taken out as stop words, added back in in brackets) [4]:

1. "...(And you) could (have it) all, (my) empire (of) dirt..."

2. "...become, (my) sweetest friend, everyone..."

3. "...dirt, (I will) let (you) down..."




(a) Test 4 bigram importance screen (b) Test 4 trigram importance screen

Figure 5.9: Test 4 bigram and trigram word importance screens

On the other hand, the bigram and trigram importance screens tell us more about the thoughts of the audience, rather than just their favourite, most mentioned lyrics. Figure 5.9b shows us that the second most important trigram is "rip johnny cash", referencing the death of the artist. The first "nine inch nail" trigram in the list is a reference to the band who originally performed the song. Some more trigrams from the top of the list, such as "love johnny cash", "cry every time", "song make cry", and "best cover ever" can be quite useful in investigating the thoughts of the audience.

### 5.1.4 Discussion

As mentioned previously, tables 5.3 "Sentiment Assigned", "Emotion Assigned" and "Offence Assigned" columns documents the percentage of comments that the system was able to assign a non-neutral sentiment, a non-neutral emotion category and an offensive label to respectively. The table has been repeated here.

[4] https://genius.com/Johnny-cash-hurt-lyrics

| No. | Title | Creator | Views | Main Comments | URL | Sentiment Assigned | Emotion Assigned | Offence Assigned |
|---|---|---|---|---|---|---|---|---|
| 1 | Should she have been FIRED? – Debating Destiny | Bastiat | 4000 | 26 | https://www.youtube.com/watch?v=KlbDJcsNTNo | 69% | 31% | 23% |
| 2 | Progressive Appears on Daily Wire, It Doesn't Go Well | David Pakman | 340,000 | 4138 | https://www.youtube.com/watch?v=dwOSlwtyA_M | 87% | 65% | 43% |
| 3 | Mask Off | Future | 535,000,000 | 153,359 | https://www.youtube.com/watch?v=xvZqHgFz51I | 56% | 33% | 13% |
| 4 | Hurt | Johnny Cash | 153,000,000 | 38,351 | https://www.youtube.com/watch?v=8AHCfZTRGiI | 74% | 56% | 29% |
| 5 | Cold Heart (PNAU Remix) | Elton John, Dua Lipa | 241,000,000 | 26,828 | https://www.youtube.com/watch?v=qod03PVTLqk | 68% | 49% | 14% |

Table 5.3: Music videos used to test the application

The system was able to assign sentiments to 87& of the second test video, and 56%, 74%, and 68% to the music videos respectively. The system was also able to assign an emotion category to 65% of the second test video, compared to 33%, 56%, and 49% to each of the music videos respectively. The second political video has 43% of its comments labelled as offensive, in comparison to 13%, 29%, and 14% respectively.

This is probably because the engagement in the comments section of a political video is vastly different from the types of comments posted to music videos. Indeed when observing the most positive and negative comments for each of these videos, the comments from the political one tended to be more long-form paragraphs. These findings suggest that videos that involve discussions about topics are better suited for my system than just short music videos.

The difference in offensive comments is quite substantial and is likely attributed to the nature of political videos, especially videos like the second test video that discuss contentious issues like transgender issues. Additionally, as stated previously, the artists of the fourth and fifth music videos are immensely respected, decreasing the likelihood of people posting offensive comments to their music videos, especially since the artist for the fourth video has passed away.

Generally, the more comments a video has, the longer the application takes to process the video. This is clearly shown in the data. Additionally, the more comments a video has, the more efficiently the system is able to process the comments. As the number of comments per video increases, the comments processed per second also increase.

## 5.2 Evaluation

In chapter 3, a list of requirements was proposed. This section outlines each of these requirements and discusses if they have been achieved.

### 5.2.1 Functional Requirements

1. **The software must retrieve all comments of a YouTube video, taken as user input, and store them in a CSV file.**

   When the system is launched, the first window that is displayed to the user gives the user an option to enter a YouTube video's URL and import comments from the URL. The "**CommentFetcher**" class is used to do this, and stores the comments in their relevant folders as CSV files. The user also has the option of opening a previously downloaded set of comments.

2. **The software must filter and clean the data using the following methods**:

   (a) **Remove empty comments.**

   (b) **Remove non-English comments.**

   (c) **Remove URL links.**

   (d) **Remove special symbols.**

   (e) **Convert any uppercase letters into lowercase letters.**

   (f) **Remove accented characters.**

   (g) **Expand contractions.**

   (h) **Replace some punctuation with relevant words.**

   (i) **Remove all punctuation.**

   (j) **Correct any spelling errors.**

   The "**CommentCleaner**" class is used to clean the data using most of the methods described above. For each of the items on the list, a function has been created inside the class that performs this cleaning. The only method not applied to the data on this list are (h), as there was no simple way to go about completing point (h). Given more time, point (h) would have been completed.

3. **The software must process the data into a state where NLP and SA can be performed on it. This must include the following procedures**:

(a) **Tokenise all of the comments.**

(b) **Perform lemmatization on the tokens.**

(c) **Remove all stop words from the tokens.**

All three of these points have been covered, also using the "**CommentCleaner**" class. Once again, each of the points on the list is completed using its own functions inside the class.

4. **The software must analyse the data using the following methods**:

   (a) **Using a lexicon to assign each comment a polarity (sentiment) score.**

   (b) **Performing count vectorization on the data.**

   (c) **Performing TF-IDF vectorization on the data.**

   (d) **Train a machine learning-based classifier using categorically labelled corpora; classify each comment into a category.**

   Point (a) is completed in the "**CommentSentiment**" class. This is done with the help of the VADER, TextBlob and SentiWordNet lexicons. Points (b) and (c) are all completed in the "**CommentVectorizer**" class. Point (d) is completed in the "**CommentClassifier**" class. An emotion category is assigned to each comment by initially training a decision tree classifier using the GoEmotions corpus [42], and then using this classifier to predict categories from the imported comments. The OffensEval corpus [43] is also used to first train an SVM classifier, and then label each comment as either an offensive comment or not.

5. **The software must output the following statistics about the data, giving the user an option to filter any of the statistics to only include data from the last 30 days, the last 7 days and the last 48 hours, as well as all time**:

   (a) **The total amount of positive, neutral and negative comments over the whole dataset.**

   (b) **The average valence of a comment.**

   (c) **A list of all of the comments, with an option to filter the comments by most positive and by most negative.**

   (d) **A list of the most commonly used words throughout the dataset, sorted by the most common words, displayed as raw data as well as through a bar chart and a word cloud.**

(e) **A list of the most important words used throughout the dataset, sorted by the most important words, displayed as raw data as well as through a bar chart and a word cloud.**

(f) **A breakdown of what categories have been assigned to each of the comments, and the frequencies of these categories, displayed as raw data as well as through a bar chart and a word cloud.**

A GUI has been designed and implemented with the help of the PyQt5 library and the Qt Designer application. Points (a) and (b) are displayed to the user in the main window, on the report screen - this first screen that is displayed once the data has been processed. Point (c) is displayed to the user on the list of comments screen. Points (d) and (e) are displayed to the user on the subsequent n-gram frequency and importance screens. Point (f) has been implemented using the emotional analysis screen, where the frequencies of each of the 28 emotion categories that have been assigned over the dataset are displayed. The frequency, importance, and emotional analysis screens all have the option of displaying the data of that respective screen to the user as either a bar chart or a word cloud.

6. **For each unique YouTube video, the software must save the following data using CSV files**:

   (a) **The comments and timestamps of each comment.**

   (b) **The comments, with each stage of the cleaning process displayed in its own column.**

   (c) **The comments, along with their polarity scores, sentiments (positive, neutral, or negative), and assigned categories.**

   (d) **A report of the total amount of positive, neutral and negative comments over the whole dataset, as well as this data for the last 30 days, the last 7 days and the last 48 hours.**

   This point is complete in the "**CommentsFetcher**" and "**Main**" classes. For every video, a new folder is created that stores all of this data. For any given video, point (a) is saved in a file named "data.csv", point (b) in "cleaned_data.csv", point (c) in "processed_data.csv", and point (d) in "report.csv". As described previously, table 4.2 shows a breakdown of each of the files that are saved with each video.

### 5.2.2 Non-Functional Requirements

1. **The software must output accurate and reliable results.**

   The accuracy of the results is hard to quantify, as there is no baseline for each YouTube video's comment's sentiments to compare the results to. The results described in section 5.1 indicate that the application does successfully summarise the contents of the comments. In the second test video, many topics that were mentioned in the video were listed, whereafter a lot of these topics were frequently mentioned in the n-gram screens. In the three music videos used to test the application, each of these video's n-gram pages frequently mentioned the song's lyrics and the artist's names, which a lot of the comments were comprised of.

   It is hoped from these data points a content creator would be able to gain an accurate reading of the views and sentiments of their audience about a particular video. This certainly seems to be the case based on the test videos. At the very least, some meaningful insight can be gained from the application when using an English video.

   The application is able to reliably process any YouTube video without crashing, assuming the video has less than 200,000 main comments. The application is also able to assign a sentiment to the majority of comments from any English YouTube video, as it was able to assign a sentiment to 71% of comments on average from all of the test videos.

2. **The software must output data in a reliable time frame.**

   The application is able to process videos in a reliable time frame. The first and second test videos, which represent the average YouTube video, were processed in INPUT and INPUT seconds respectively. Although this isn't extremely fast, most users using the application would not have to wait substantial amounts of time. The largest dataset used to test the application was the first music test video, containing over 150,000 main comments. The application was able to process this dataset in INPUT seconds.

   The majority of YouTube videos do not rise in popularity and get many comments posted, so the application should be suitable for the vast majority of content creators. Analysts who want to use the application with huge datasets like popular music videos may experience longer processing times.

3. **The software must be easy to deploy, configure and maintain in the future.**

   The software has been written to the highest standard. Each requirement is written in

its own class, and most of the sub-requirements have been written in their own functions. Each stage of the data cleaning and processing can be decoupled and reordered to another user's desire. The executable file can be used on any other machine that runs Windows 10.

4. **All data collected must remain anonymous.**

   No information about the author of any of the comments has been collected. The only information stored about each comment is the text of said comment and the date-time timestamp at which the comment was posted.

5. **The software produced should meet the highest professional and ethical standards set out by the BCS.**

   The application has been implemented adhering to the professional and ethical standards set out by the BCS. This is discussed in chapter 6.

# Chapter 6

# Legal, Social, Ethical and Professional Issues

This chapter discusses the legal, social, ethical and professional issues faced during the implementation of this project.

The British Computer Societies Code of Conduct [1] and Code of Good Practice [2] have been taken into consideration across the whole project.

The Code of Conduct has been followed to ensure that the project has due regard for public health, privacy, security and wellbeing of others and the environment; to ensure that the project complies with existing legislation; to uphold the reputation and good standing of the British Computer Society.

The Code of Good Practice has been followed to ensure that the project adheres to existing regulations, following the standards relevant to King's College London; contributes to current developments in the relevant specialist area; uses appropriate methods and tools; contributes to the education of the public whenever there is an opportunity to do so; demonstrates a high level of professional competence with respect to existing legislation, regulations, and security recommendations; and takes all reasonable care to ensure that work and the consequences of said work cause no unacceptable risk to safety.

Any YouTube video's comments can be downloaded as a CSV file through the use of this project. Additionally, various corpora have been used as training data for machine learning classifiers, with the two corpora used in the study totalling more than 58,000 Reddit comments

[1] https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/
[2] http://www.inf.ed.ac.uk/teaching/courses/pi/2013$_2$014/*notes*/1/*cop.pdf*

and 11,000 Tweets. No personal information and no identifiers are saved with any of these comments/tweets, meaning that all of the data that has been collected is completely anonymous, and cannot be linked back to the poster/commenter.

Further, when classifying each of these comments into sentiment, emotion and offensive? categories, no rights have been violated in this process.

Additionally, various open-source libraries have been used over the course of this project, discussed in section 3.3. Additionally, the stylesheets used within the project have their licenses stated at the top of their files. This project is not being used commercially.

All sources for any code taken and modified have been clearly referenced within the code. All citations in the text, sources of images and website references have been clearly referenced within the report. The project report and the code consists of my own work, and any external sources have been cited.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

One of the aims of this project was to perform NLP and SA on a range of comments of a given YouTube video, input by the user, and to return to the user a report of the processed comments along with helpful statistics and charts about the comments. The literature review discussed NLP and SA in great detail, outlining various methods that would help achieve this goal. Some of these methods were then implemented. Various lexicon-based methods were used to perform SA and assign sentiment to each of the comments in a dataset. A decision tree classifier was trained using a corpus found online and then used to categorise a dataset's comments into various emotion categories. A SVM was also used to classify each comment in the dataset as either offensive or inoffensive. Additionally, the unigrams, bigrams, and trigrams found within the dataset were both counted and assigned an importance score. The result was an application able to download and process comments from a YouTube video and display these comments, as well as a range of statistics about these comments, with the help of bar charts and word clouds, to the user.

Another aim of this study was for the user to be able to sort all of the processed data by time range, so that the user would be able to see the sentiments, emotions, and viewpoints of users who recently watched and posted a comment on their videos, as opposed to their normal audiences. The result of this is that all of the data screens in the application are able to be filtered for four specific time ranges, helping the user do this.

This study has mainly aimed to create an application for content creators. However, other groups of people, such as market researchers, can also use the application to gain a better

understanding of a certain audience/market.

Choosing to use three separate lexicon-based methods to assign sentiments to as many comments as possible proved to be a good idea, as the average amount of comments assigned some non-zero polarity is 71% (the average value calculated based on all of the testing videos), meaning that the system is able to assign a sentiment to the majority of the comments from any English YouTube video. Perhaps an even better approach would have been to build an independent lexicon-based evaluation method specifically tuned to analysing YouTube comments so that each comment would have been evaluated using the exact same method, rather than one of the three lexicon-based methods, each of which evaluates comments differently. This would have taken a substantial amount of time.

The choice to use a decision tree classifier, as well as a SVM classifier as opposed to some of the other machine learning methods described in chapter,2.5.3 was made based on the existing literature available. A better approach may have been to split the corpus data into training and testing datasets, training a large variety of classifiers using the training datasets, and then evaluating their performance using the testing datasets, choosing the best classification method for each corpus. This would have made sure that the most effective (highest accuracy) classifier was used for each of the category assignments. This would have also taken a substantial amount of time but would have increased the accuracy of the data.

## 7.2 Future Work

There is a lot of room for expansion in this project.

This project mainly focuses on working at the lexical NLP level, mostly working with individual words (or bigrams or trigrams) rather than focusing on the grammatical structure of each comment, or the meanings of each of a comment's sentences. As most YouTube comments do not usually span more than a sentence, this is fine for most comments. However, this means that a lot of information that lengthier comments provide is not examined. Methods that work at other levels of NLP and SA, described in sections 2.2 and 2.4, can be implemented.

Methods applied at the syntactic and semantic NLP levels that would be able to discern the meanings of groups of words in sentences as well as whole sentences could be implemented to display to the user what specifically the commentors are trying to get across, and which of these meanings are the most popular amongst the audience. For example, one could conceive a system where the meaning of each of the comments is calculated and displayed next to the comment, where comments with similar meanings are counted, and the most used meanings

are displayed to the user.

Methods applied at the pragmatic level can also be used, where first the application would have to somehow gather information about the video, and then apply that knowledge when discerning the meaning of comments. Information about the video can be gathered either from the description of the video or by interpreting the words mentioned in the video as text. This text can then be fed into some sort of automated system that searches the internet for any topics mentioned, building a knowledge base from this information, and then using this knowledge base to better understand each of the comments.

The paper focused on working at the document level and entity level SA, where the sentiment of individual words in the comments, as well as the sentiment of each comment, is calculated and displayed. Sentence level SA can be incorporated into this paper, where each sentence of a comment can be assigned a sentiment score. The sentiment scores of each sentence in a comment could be displayed to the user for a more in-depth understanding of the comments. Additionally, comments containing sentences with both positive and negative sentiments could also be displayed as "interesting" comments. Furthermore, entity level SA can be performed where the entities that come up the most in the dataset can be calculated, where each of these entities have sentiment scores assigned to them before being displayed to the user. The user would then be able to see what entities across the dataset have been described in a positive or negative light, or what entities were most loved or hated by the audience.

More n-gram levels can be added, where groups of four or more consecutive words could be processed.

Various features to improve the understanding of the system when performing NLP could also be implemented when cleaning the data. One of these features could be a better contraction expansion method, where instead of simply replacing a contraction with it's most popular expanded phrase, the method would analyse the context of the contraction (other words around the contraction) and make a decision based on these words, improving the accuracy of the dataset. Another possible feature could be to replace any emotional punctuation, for example, "!!!!!!", with a phrase, in this case, " extremely", as done in Sun et al. [2].

A feature could be added that gives an option to the user, asking them if they want duplicate comments to not be included in the dataset. This could be beneficial to popular content creators that get a lot of bots posting repeated comments on their videos so that this data does not skew the results in any way.

A couple of other basic quality of life features could be introduced. Another method of

generating an executable file could be found where the executable file is able to be opened faster; a search feature that allows the user to filter the list of comments or the frequency and importance pages for specific words; a more accurate way of expanding contractions; a loading screen during the processing of the data.

# References


[1] E. D. Liddy, *Natural Language Processing for Information Retrieval*.

[2] F. Sun, A. Belatreche, S. Coleman, T. M. McGinnity, and Y. Li, "Pre-processing online financial text for sentiment classification: A natural language processing approach," in *2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*. IEEE, 2014, pp. 122–129.

[3] R. Gandhi, "Support vector machine — introduction to machine learning algorithms," Jun 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[4] B. Dean, "How many people use youtube in 2022? [new data]." [Online]. Available: https://backlinko.com/youtube-users

[5] "Youtube user statistics 2022: Global media insight." [Online]. Available: https://www.globalmediainsight.com/blog/youtube-users-statistics/

[6] A. Nishajebaseeli and K. Ezra, "A survey on sentiment analysis of (product) reviews," *International Journal of Computer Applications*, vol. 47, pp. 36–39, 06 2012.

[7] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," 08 2017.

[8] A. Reshamwala, D. Mishra, and P. Pawar, "Review on natural language processing," *IRACST – Engineering Science and Technology: An International Journal (ESTIJ)*, vol. 3, pp. 113–116, 02 2013.

[9] G. Chowdhury, "Natural language processing," *ARIST*, vol. 37, pp. 51–89, 01 2005.

[10] S. Feldman, "Nlp meets the jabberwocky: Natural language processing in information retrieval," *ONLINE-WESTON THEN WILTON-*, vol. 23, pp. 62–73, 1999.

[11] E. Atwell, "Development of tag sets for part-of-speech tagging," 2008.

[12] S. Ramanujam, "Twitter nlp example: How to scale part-of-speech tagging with mpp (part 1)," Oct 2014. [Online]. Available: https://tanzu.vmware.com/content/blog/twitter-nlp-example-how-to-scale-part-of-speech-tagging-with-mpp-part-1

[13] B. Liu *et al.*, "Sentiment analysis and subjectivity." *Handbook of natural language processing*, vol. 2, no. 2010, pp. 627–666, 2010.

[14] P. Majumder, "Word sense disambiguation: Importance in natural language processing," Jun 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/word-sense-disambiguation-importance-in-natural-language-processing/

[15] S. Moeller, "The ultimate guide to facebook engagement (with data)." [Online]. Available: https://buzzsumo.com/blog/ultimate-guide-facebook-engagement-2017/

[16] S. B. Moralwar and S. N. Deshmukh, "Different approaches of sentiment analysis," 2015.

[17] R. Pokharel and D. Bhatta, "Classifying youtube comments based on sentiment and type of sentence," *arXiv preprint arXiv:2111.01908*, 2021.

[18] F. Song, S. Liu, and J. Yang, "A comparative study on text representation schemes in text categorization," *Pattern Anal. Appl.*, vol. 8, no. 1–2, p. 199–209, sep 2005.

[19] B. Yu, "An evaluation of text classification methods for literary study," *Literary and Linguistic Computing*, vol. 23, no. 3, pp. 327–343, 2008.

[20] H. Saif, Y. He, and H. Alani, "Semantic sentiment analysis of twitter," in *International semantic web conference*. Springer, 2012, pp. 508–524.

[21] A. Katrekar, "An introduction to sentiment analysis."

[22] S. Qaiser and R. Ali, "Text mining: use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, 2018.

[23] A. A. Hakim, A. Erwin, K. I. Eng, M. Galinium, and W. Muliady, "Automated document classification for news article in bahasa indonesia based on term frequency inverse document frequency (tf-idf) approach," in *2014 6th international conference on information technology and electrical engineering (ICITEE)*. IEEE, 2014, pp. 1–4.

[24] C. Manning, P. Raghavan, and H. Schutze, "Term weighting, and the vector space model," *Introduction to information retrieval*, pp. 109–133, 2008.

[25] "Term frequency - inverse document frequency statistics." [Online]. Available: https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html

[26] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[27] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, "Machine learning and the physical sciences," *Reviews of Modern Physics*, vol. 91, no. 4, p. 045002, 2019.

[28] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," *arXiv preprint cs/0205070*, 2002.

[29] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[30] A. Ashari, I. Paryudi, and A. M. Tjoa, "Performance comparison between naïve bayes, decision tree and k-nearest neighbor in searching alternative design in an energy simulation tool," *International Journal of Advanced Computer Science and Applications*, vol. 4, 12 2013.

[31] H. Sharma and S. Kumar, "A survey on decision tree algorithms of classification in data mining," *International Journal of Science and Research (IJSR)*, vol. 5, no. 4, pp. 2094–2097, 2016.

[32] S. R. Das and M. Y. Chen, "Yahoo! for amazon: Sentiment extraction from small talk on the web," *Management science*, vol. 53, no. 9, pp. 1375–1388, 2007.

[33] J. Smailović, M. Grčar, M. Žnidaršič, and N. Lavrač, "Sentiment analysis on tweets in a financial domain," in *4th Jožef Stefan International Postgraduate School Students Conference*, vol. 1. Citeseer, 2012, pp. 169–175.

[34] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[35] S. Kannan, S. Karuppusamy, A. Nedunchezhian, P. Venkateshan, P. Wang, N. Bojja, and A. Kejariwal, "Chapter 3 - big data analytics for social media," in *Big Data*, R. Buyya,

R. N. Calheiros, and A. V. Dastjerdi, Eds. Morgan Kaufmann, 2016, pp. 63–94. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128053942000039

[36] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," 01 2015.

[37] V. Chaithra, "Hybrid approach: naive bayes and sentiment vader for analyzing sentiment of mobile unboxing video comments," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 5, pp. 4452–4459, 2019.

[38] S. Baccianella, A. Esuli, and F. Sebastiani, "Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, 2010.

[39] "What is wordnet?" [Online]. Available: https://wordnet.princeton.edu/

[40] E. Poché, N. Jha, G. Williams, J. Staten, M. Vesper, and A. Mahmoud, "Analyzing user comments on youtube coding tutorial videos," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 196–206.

[41] E. Rinaldi and A. Musdholifah, "Fvec-svm for opinion mining on indonesian comments of youtube video," in *2017 International Conference on Data and Software Engineering (ICoDSE)*. IEEE, 2017, pp. 1–5.

[42] D. Demszky, D. Movshovitz-Attias, J. Ko, A. Cowen, G. Nemade, and S. Ravi, "Goemotions: A dataset of fine-grained emotions," 2020. [Online]. Available: https://arxiv.org/abs/2005.00547

[43] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar, "Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval)," *arXiv preprint arXiv:1903.08983*, 2019.

[44] P. Shah, "My absolute go-to for sentiment analysis - textblob." Nov 2020. [Online]. Available: https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524#:~:text=TextBlob%20calculates%20subjectivity%20by%20looking,%2C%20it%20is%20too%20boring%E2%80%9D.

[45] J. Brownlee, "4 types of classification tasks in machine learning," Aug 2020. [Online]. Available: https://machinelearningmastery.com/types-of-classification-in-machine-learning/

# Appendices

Daniel Van Cuylenburg

April 29, 2022

# Contents

# Appendix A

# Extra Information

The appendices contain information that is peripheral to the main body of the report. Information typically included in the Appendix are things like tables, proofs, graphs, test cases or any other material that would break up the theme of the text if it appeared in the body of the report. It is necessary to include your source code listings in an appendix that is separate from the body of your written report (see the information on Program Listings below).

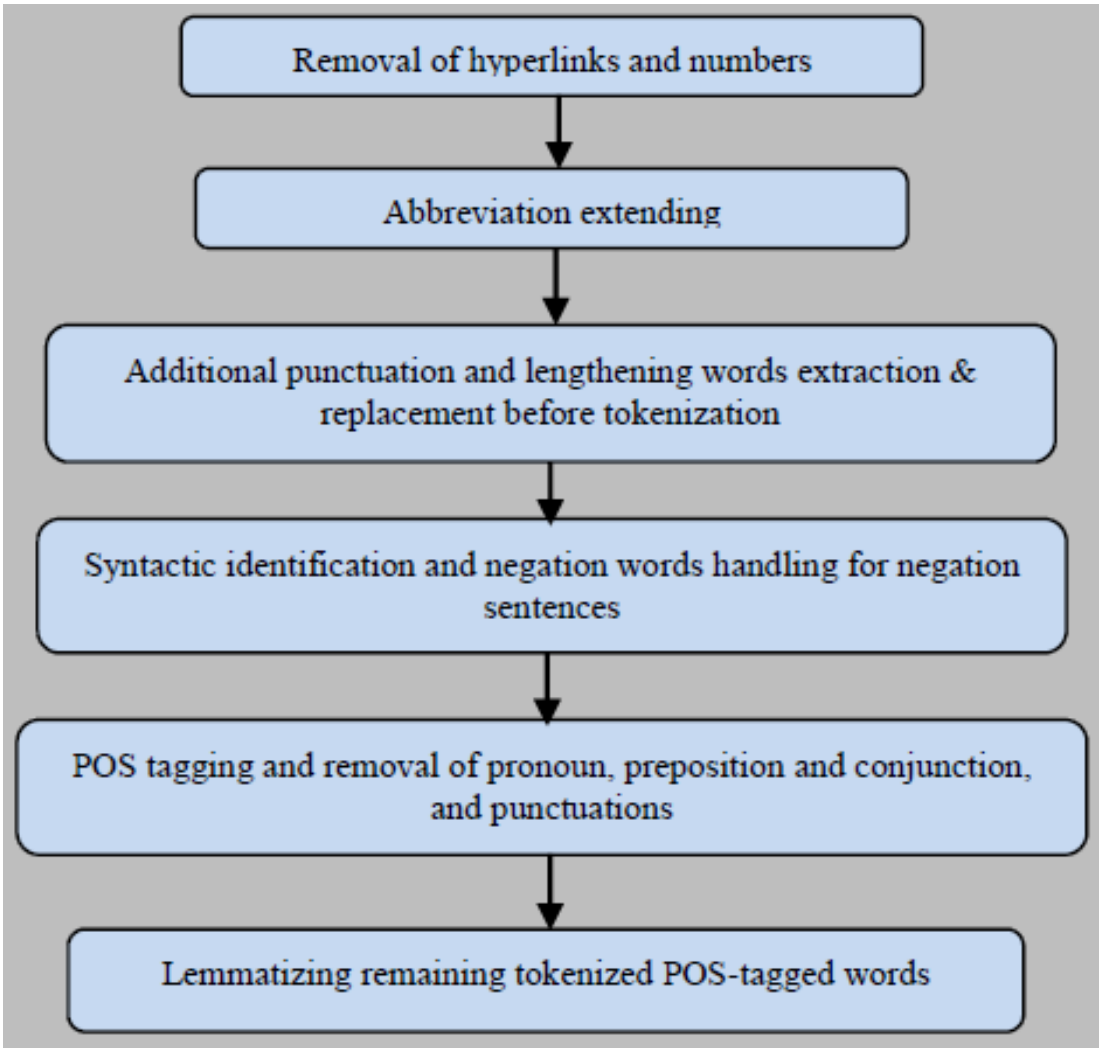
## A.1 Figures

### A.1.1 Figure 2.1 Representation of the meaning of the word ”launch” [1]

| launch (a large boat used for carrying people on rivers, lakes harbors, etc.)<br>((CLASS BOAT) (PROPERTIES (LARGE)<br>(PURPOSE (PREDICATION (CLASS CARRY) (OBJECT PEOPLE)))) |
|---|

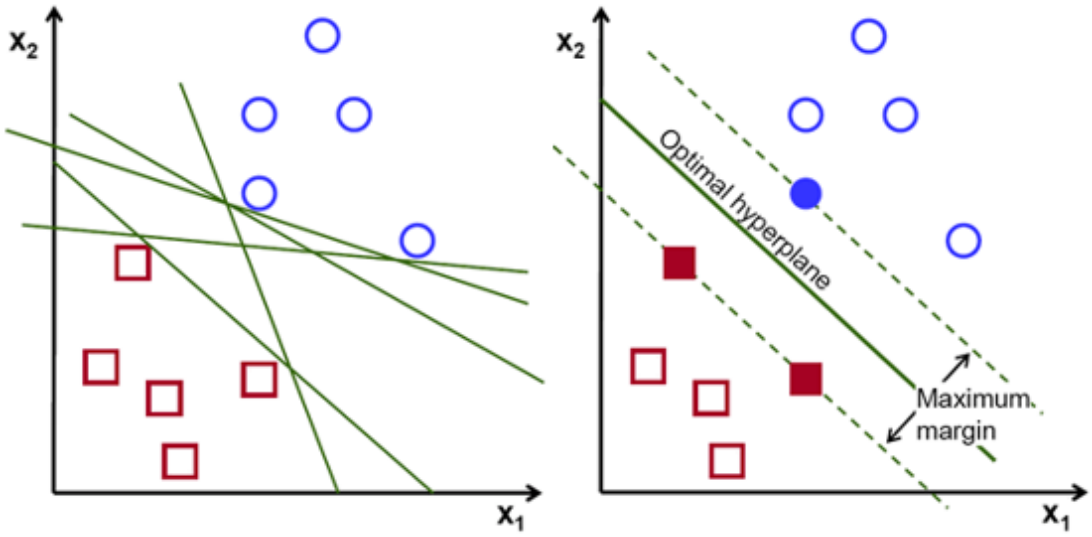
Representation of the meaning of the word ”launch” [1]

### A.1.2 Figure 2.2 An example of a preprocessing pipeline [2]



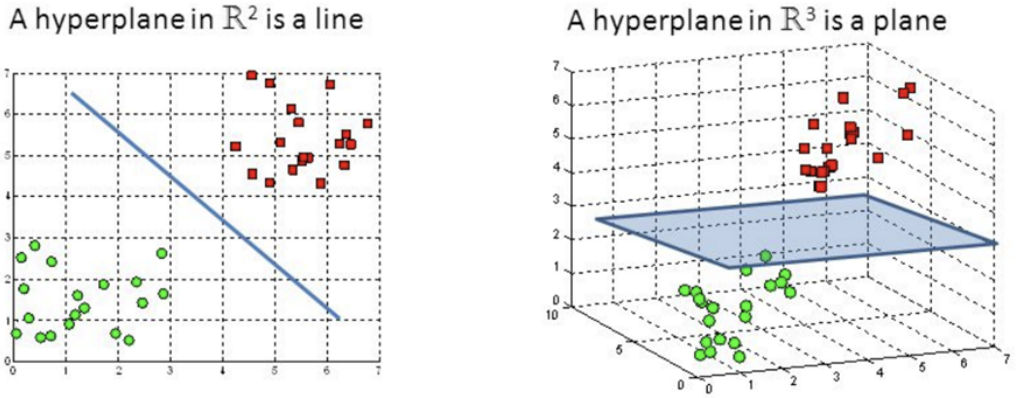

An example of a preprocessing pipeline [2]

### A.1.3 Figure 2.3 Possible hyperplanes



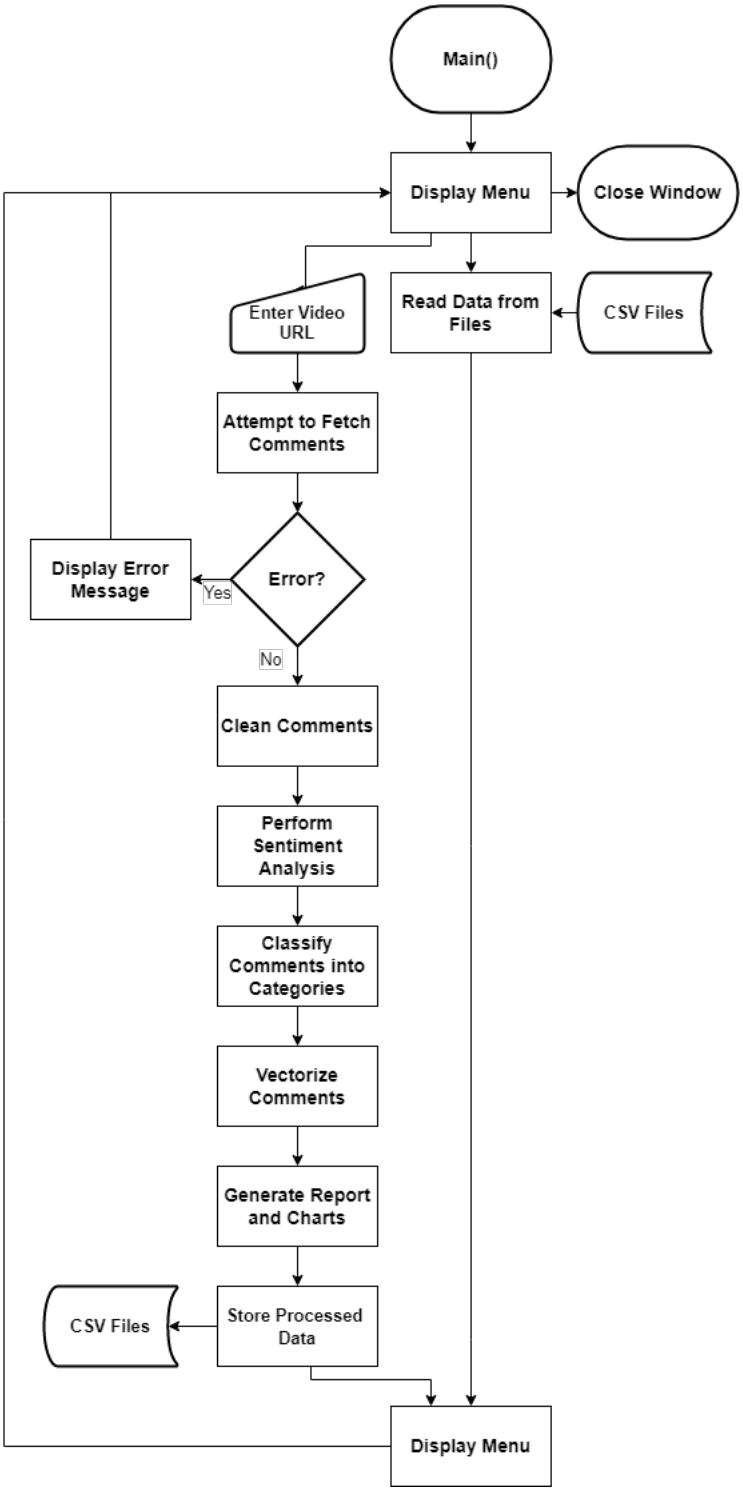

Possible hyperplanes [3]

### A.1.4 Figure 2.4 Hyperplanes in two-dimensional and three-dimensional feature spaces [3]



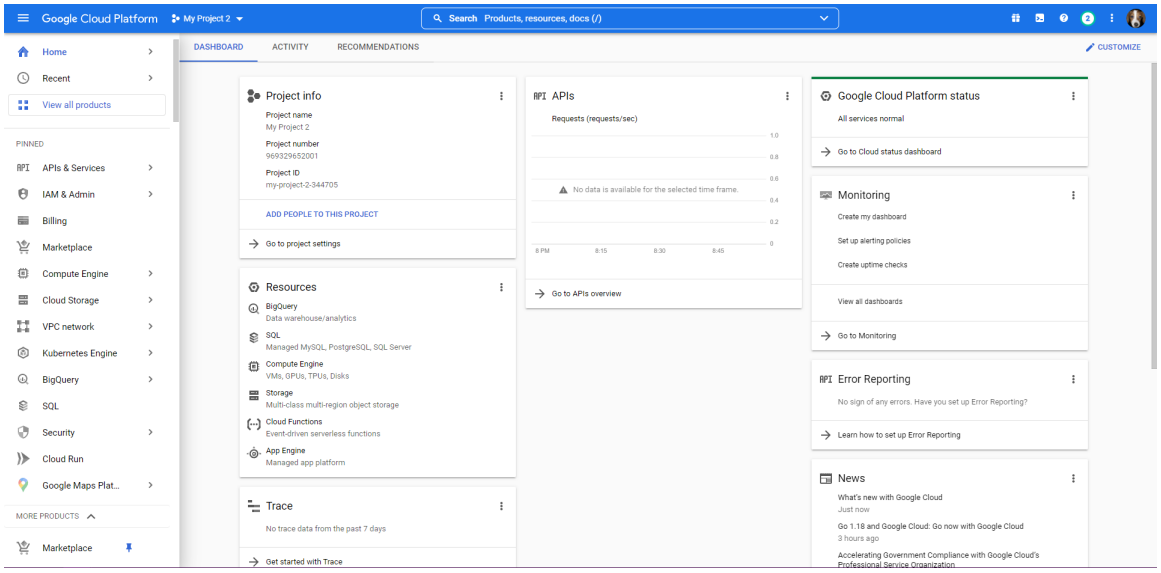

Hyperplanes in two-dimensional and three-dimensional feature spaces [3]

### A.1.5 Figure 4.1 Flowchart of the system flow



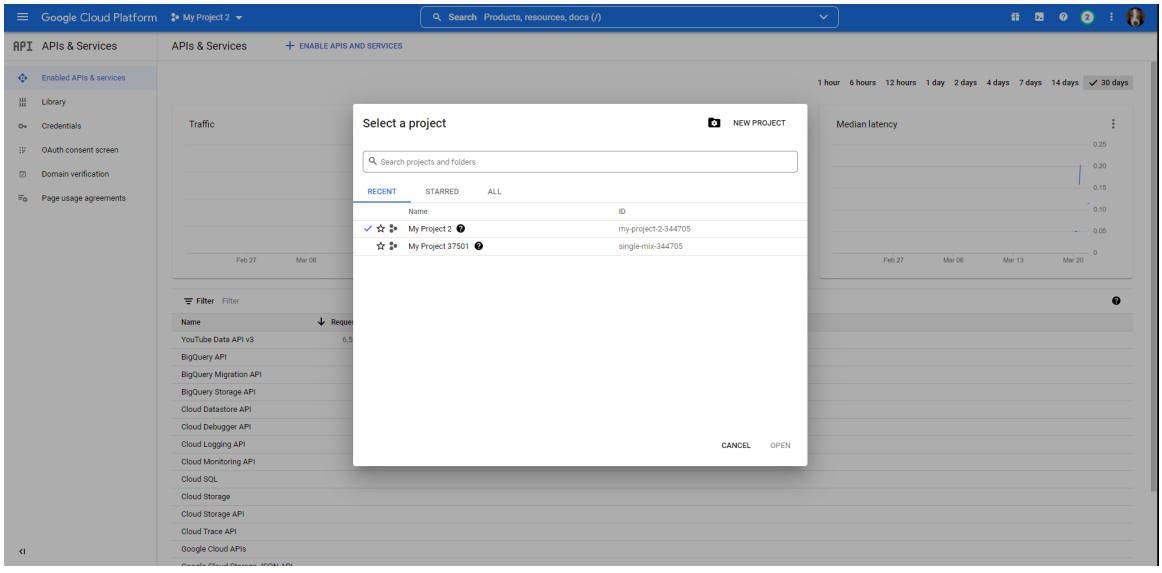

Flowchart of the system flow

### A.1.6 Figure 4.2 Google Cloud Platform home page



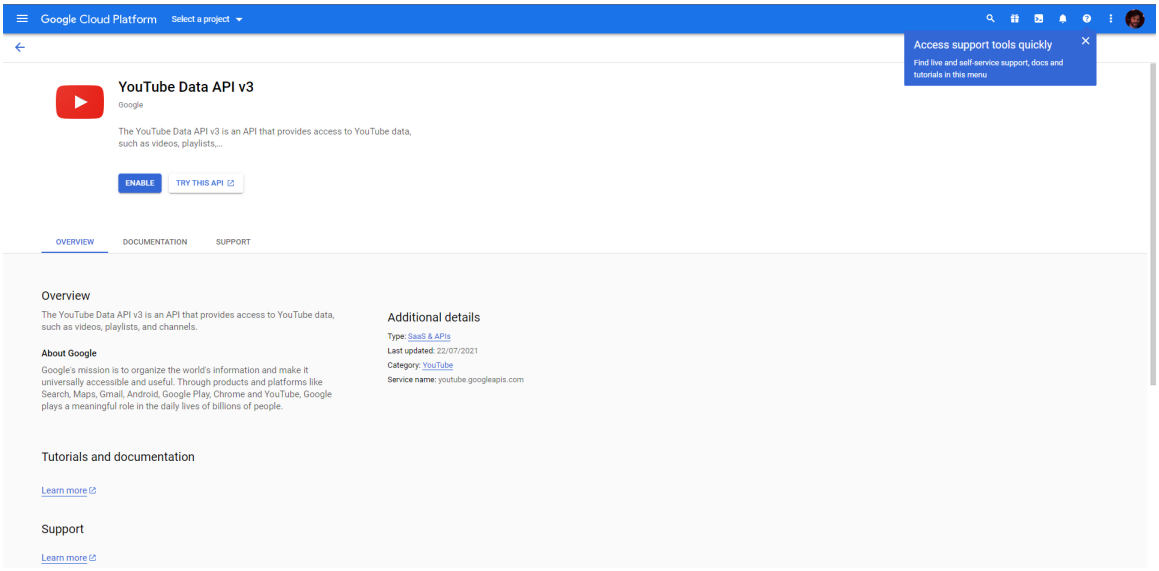

Google Cloud Platform home page

### A.1.7 Figure 4.3 Google Cloud Platform APIs & Services page



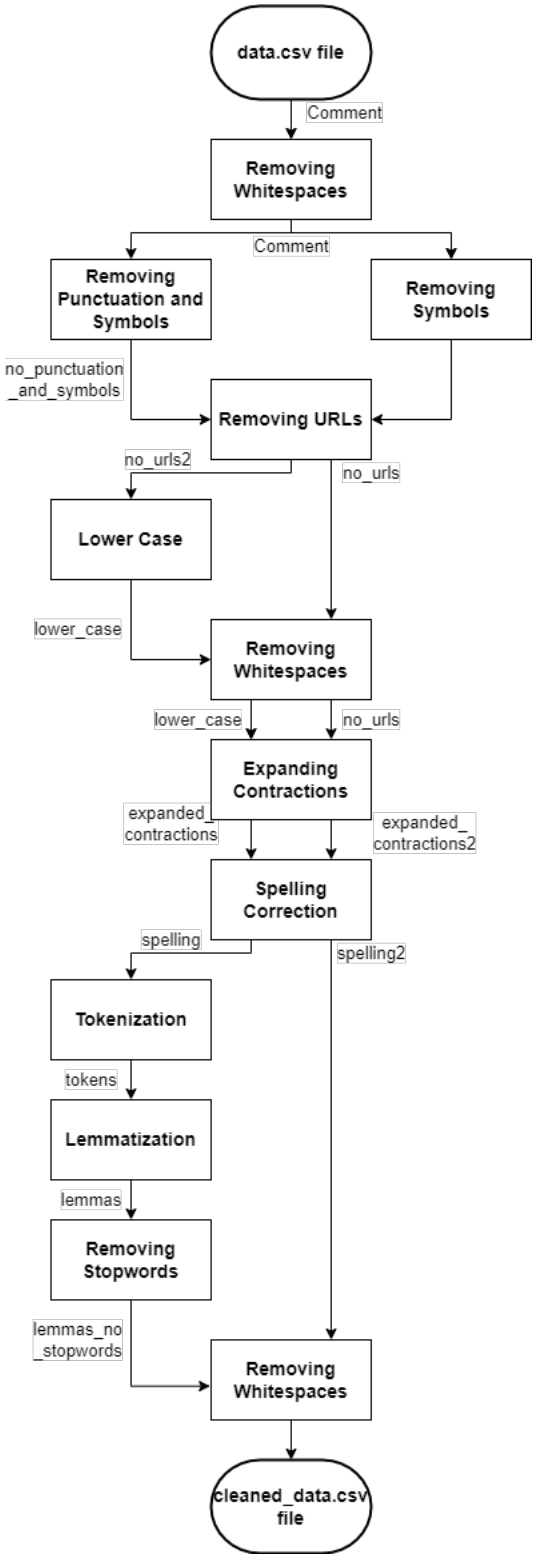

Google Cloud Platform APIs & Services page

### A.1.8 Figure 4.4 Google Cloud Platform YouTube Data API v3 page



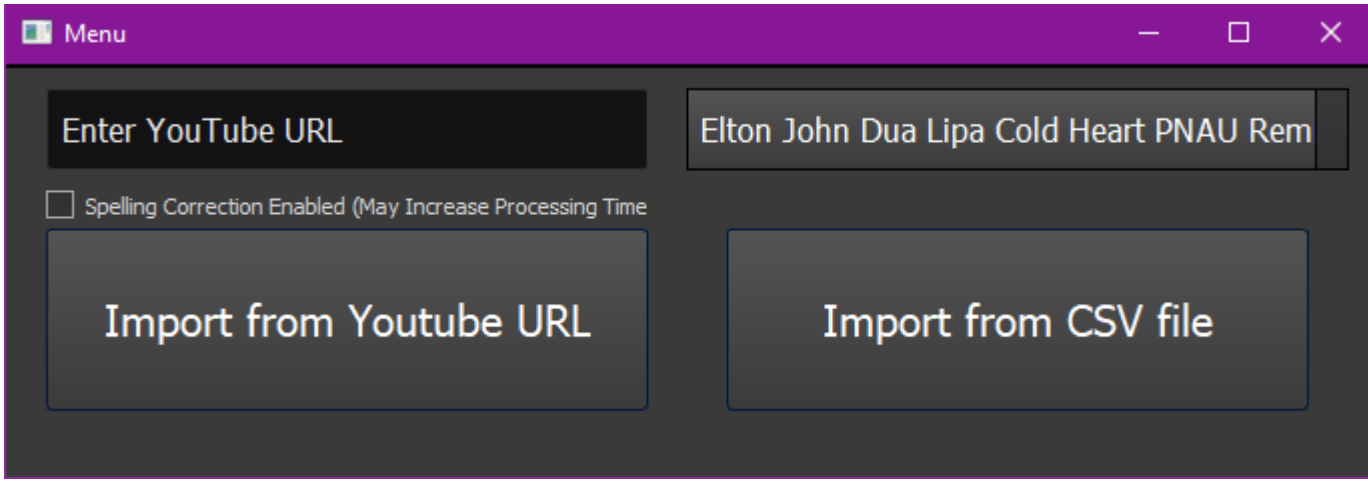

Google Cloud Platform YouTube Data API v3 page

### A.1.9 Figure 4.5 Flowchart of the data cleaning process



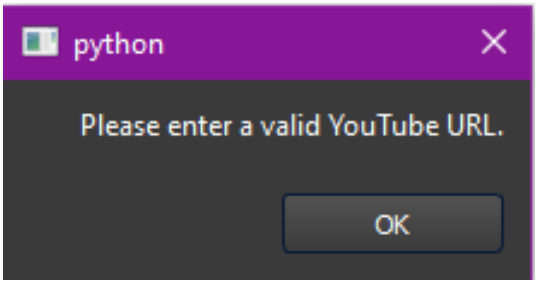

Flowchart of the data cleaning process

### A.1.10 Figure 4.6 Menu window



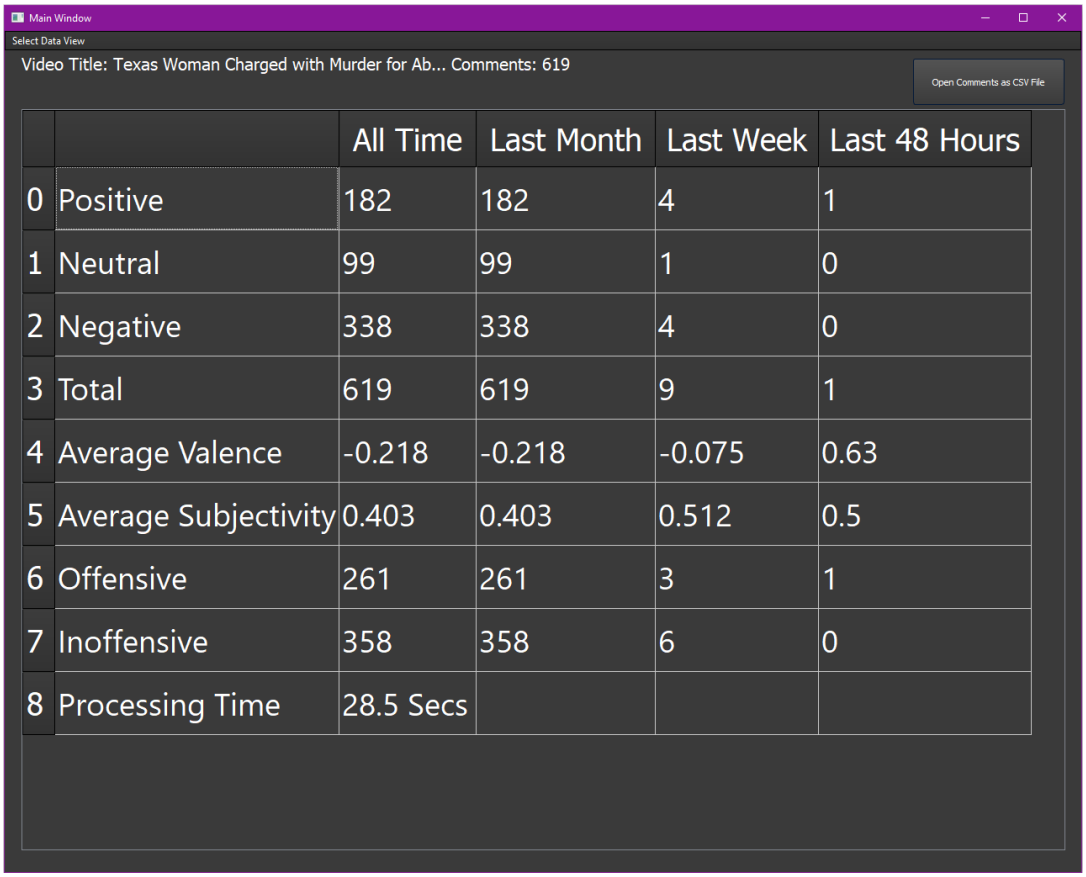

Menu window

### A.1.11 Figure 4.7 Menu error message




Menu error message

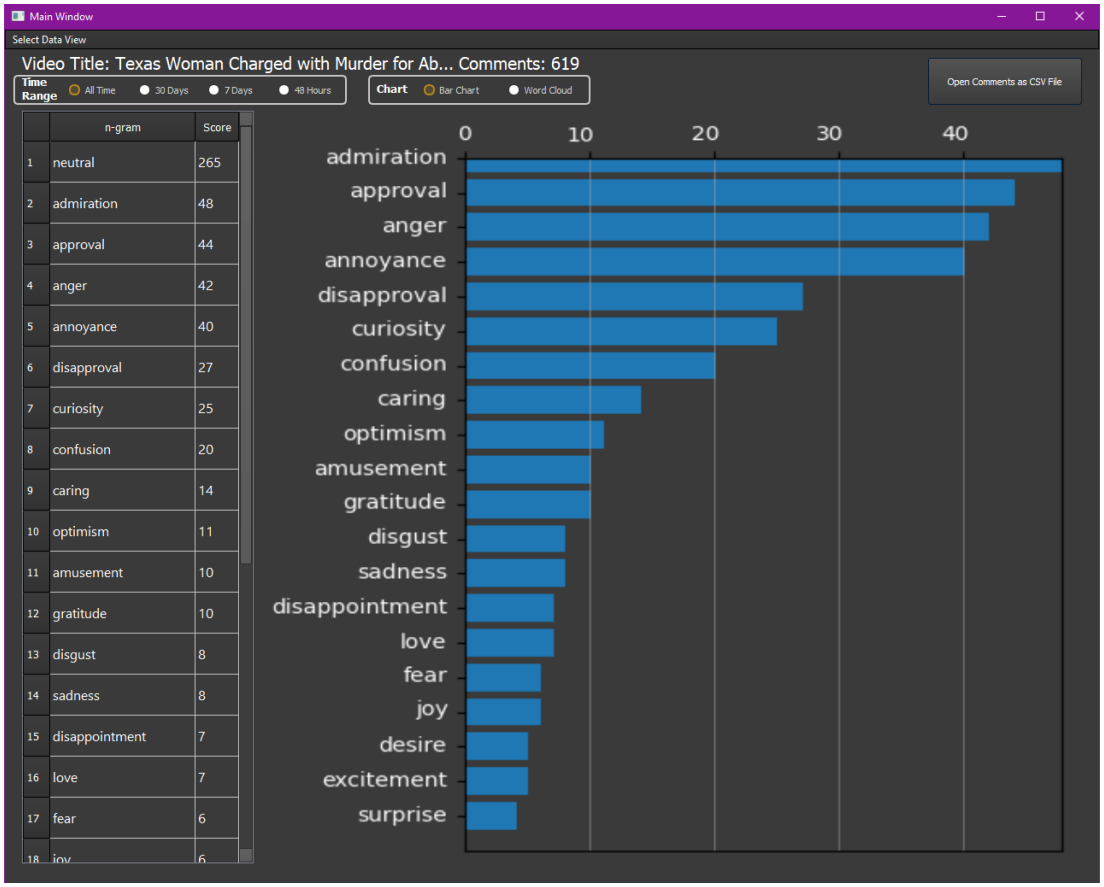
### A.1.12 Figure 4.8 Report screen



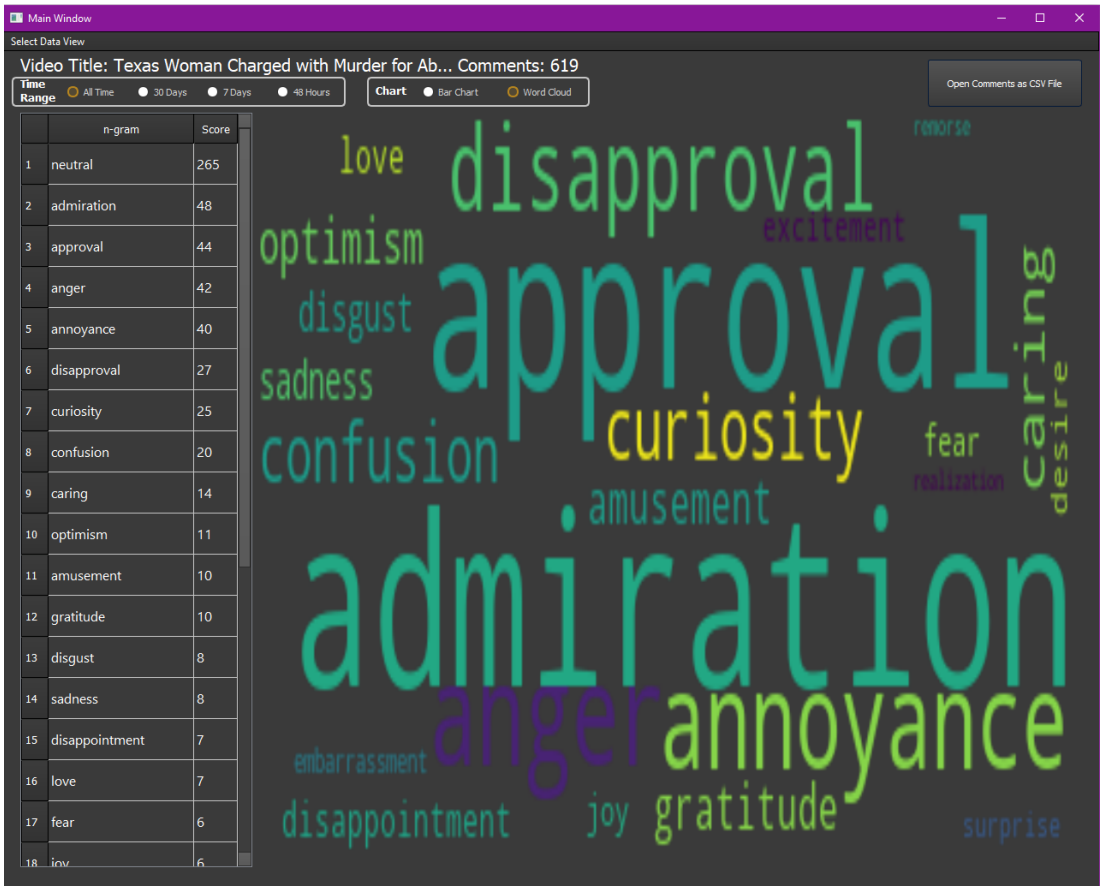

| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 182 | 182 | 4 | 1 |
| 1 | Neutral | 99 | 99 | 1 | 0 |
| 2 | Negative | 338 | 338 | 4 | 0 |
| 3 | Total | 619 | 619 | 9 | 1 |
| 4 | Average Valence | -0.218 | -0.218 | -0.075 | 0.63 |
| 5 | Average Subjectivity | 0.403 | 0.403 | 0.512 | 0.5 |
| 6 | Offensive | 261 | 261 | 3 | 1 |
| 7 | Inoffensive | 358 | 358 | 6 | 0 |
| 8 | Processing Time | 28.5 Secs | | | |

Report screen

### A.1.13 Figure 4.9 Unigram word frequency and importance screens

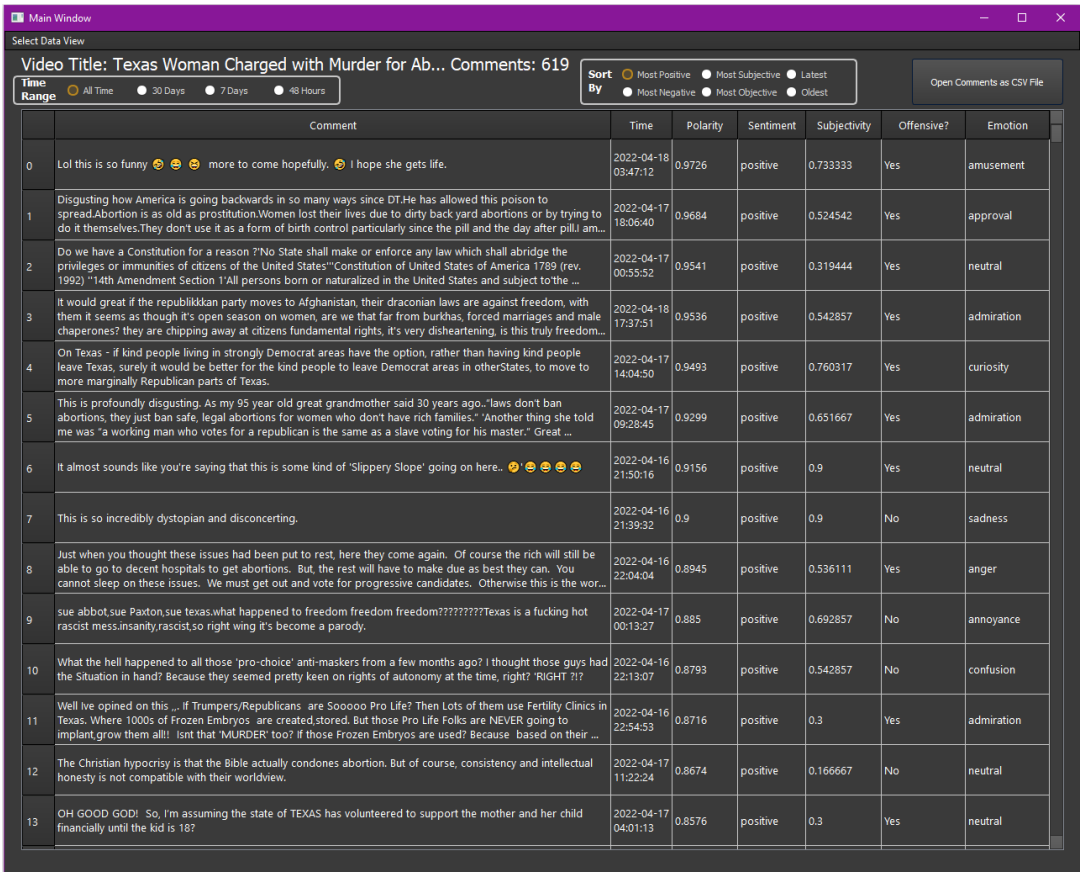



(a) Emotional analysis bar chart screen




(b) Emotional analysis word cloud screen

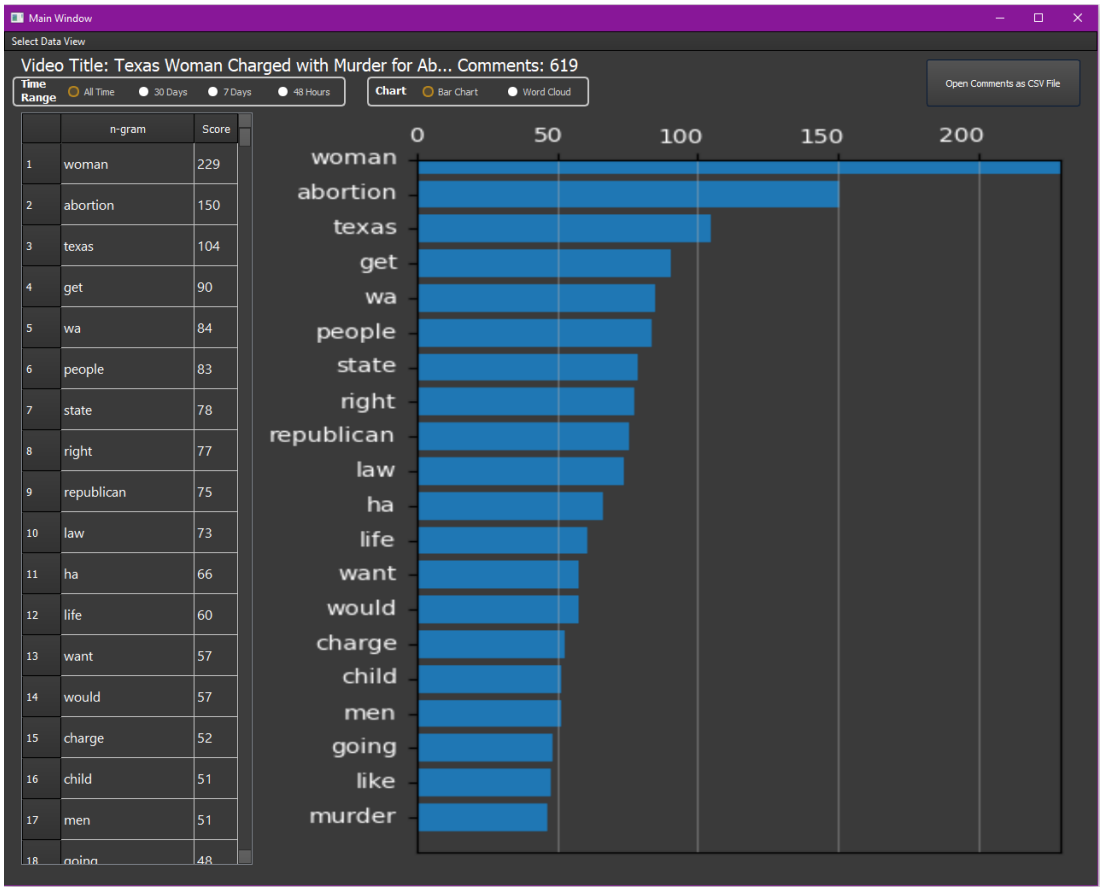
### A.1.14 Figure 4.10 Comments screen



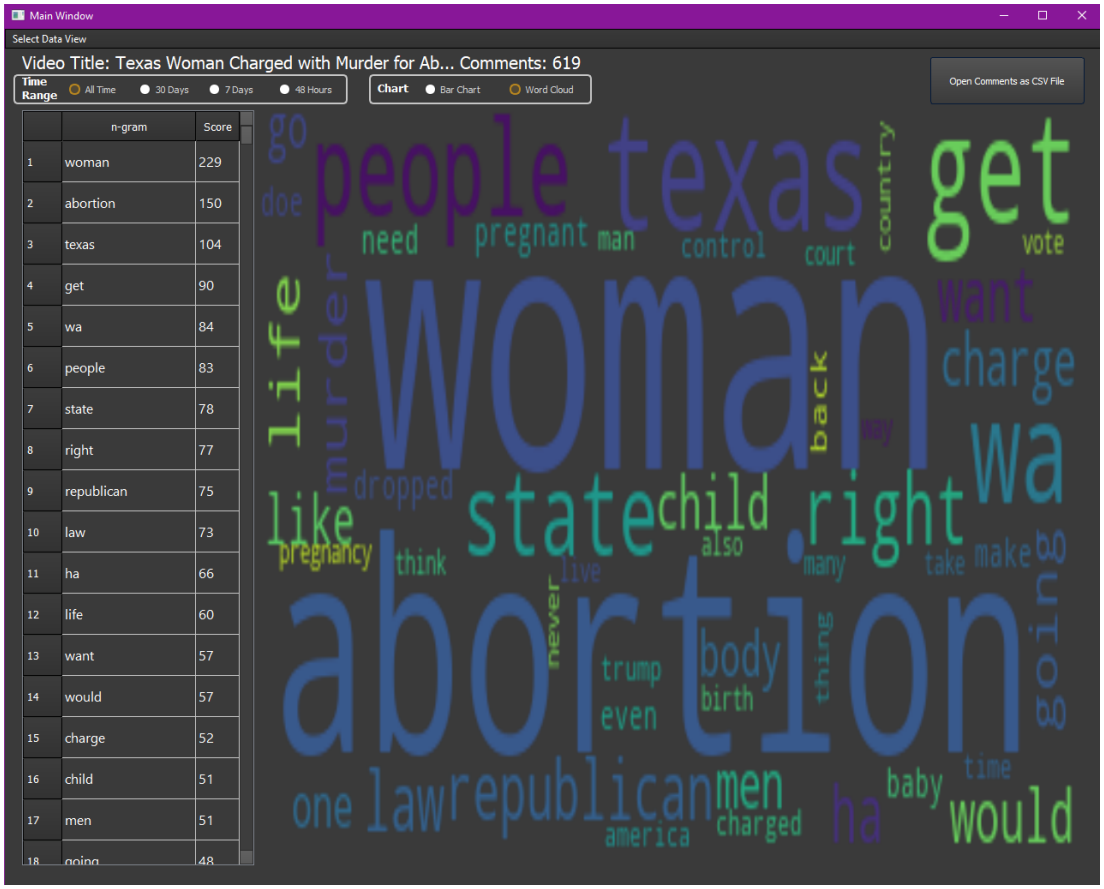

| | Comment | Time | Polarity | Sentiment | Subjectivity | Offensive? | Emotion |
|---|---|---|---|---|---|---|---|
| 0 | Lol this is so funny 😂 😀 😆 more to come hopefully. 😂 I hope she gets life. | 2022-04-18 03:47:12 | 0.9726 | positive | 0.733333 | Yes | amusement |
| 1 | Disgusting how America is going backwards in so many ways since DT.He has allowed this poison to spread.Abortion is as old as prostitution.Women lost their lives due to dirty back yard abortions or by trying to do it themselves.They don't use it as a form of birth control particularly since the pill and the day after pill.I am... | 2022-04-17 18:06:40 | 0.9684 | positive | 0.524542 | Yes | approval |
| 2 | Do we have a Constitution for a reason ?'No State shall make or enforce any law which shall abridge the privileges or immunities of citizens of the United States'"Constitution of United States of America 1789 (rev. 1992) "14th Amendment Section 1'All persons born or naturalized in the United States and subject to'the ... | 2022-04-17 00:55:52 | 0.9541 | positive | 0.319444 | Yes | neutral |
| 3 | It would great if the republikkkan party moves to Afghanistan, their draconian laws are against freedom, with them it seems as though it's open season on women, are we that far from burkhas, forced marriages and male chaperones? they are chipping away at citizens fundamental rights, it's very disheartening, is this truly freedom... | 2022-04-18 17:37:51 | 0.9536 | positive | 0.542857 | Yes | admiration |
| 4 | On Texas - if kind people living in strongly Democrat areas have the option, rather than having kind people leave Texas, surely it would be better for the kind people to leave Democrat areas in otherStates, to move to more marginally Republican parts of Texas. | 2022-04-17 14:04:50 | 0.9493 | positive | 0.760317 | Yes | curiosity |
| 5 | This is profoundly disgusting. As my 95 year old great grandmother said 30 years ago.."laws don't ban abortions, they just ban safe, legal abortions for women who don't have rich families." "Another thing she told me was "a working man who votes for a republican is the same as a slave voting for his master." Great ... | 2022-04-17 09:28:45 | 0.9299 | positive | 0.651667 | Yes | admiration |
| 6 | It almost sounds like you're saying that this is some kind of 'Slippery Slope' going on here.. 🤣😂😂😂😂 | 2022-04-16 21:50:16 | 0.9156 | positive | 0.9 | Yes | neutral |
| 7 | This is so incredibly dystopian and disconcerting. | 2022-04-16 21:39:32 | 0.9 | positive | 0.9 | No | sadness |
| 8 | Just when you thought these issues had been put to rest, here they come again. Of course the rich will still be able to go to decent hospitals to get abortions. But, the rest will have to make due as best they can. You cannot sleep on these issues. We must get out and vote for progressive candidates. Otherwise this is the wor... | 2022-04-16 22:04:04 | 0.8945 | positive | 0.536111 | Yes | anger |
| 9 | sue abbot,sue Paxton,sue texas.what happened to freedom freedom freedom????????Texas is a fucking hot rascist mess.insanity,rascist,so right wing it's become a parody. | 2022-04-17 00:13:27 | 0.885 | positive | 0.692857 | No | annoyance |
| 10 | What the hell happened to all those 'pro-choice' anti-maskers from a few months ago? I thought those guys had the Situation in hand? Because they seemed pretty keen on rights of autonomy at the time, right? 'RIGHT ?!? | 2022-04-16 22:13:07 | 0.8793 | positive | 0.542857 | No | confusion |
| 11 | Well Ive opined on this ,,. If Trumpers/Republicans are Sooooo Pro Life? Then Lots of them use Fertility Clinics in Texas. Where 1000s of Frozen Embryos are created,stored. But those Pro Life Folks are NEVER going to implant,grow them all!! Isnt that 'MURDER' too? If those Frozen Embryos are used? Because based on their ... | 2022-04-16 22:54:53 | 0.8716 | positive | 0.3 | Yes | admiration |
| 12 | The Christian hypocrisy is that the Bible actually condones abortion. But of course, consistency and intellectual honesty is not compatible with their worldview. | 2022-04-17 11:22:24 | 0.8674 | positive | 0.166667 | No | neutral |
| 13 | OH GOOD GOD! So, I'm assuming the state of TEXAS has volunteered to support the mother and her child financially until the kid is 18? | 2022-04-17 04:01:13 | 0.8576 | positive | 0.3 | Yes | neutral |

Comments screen

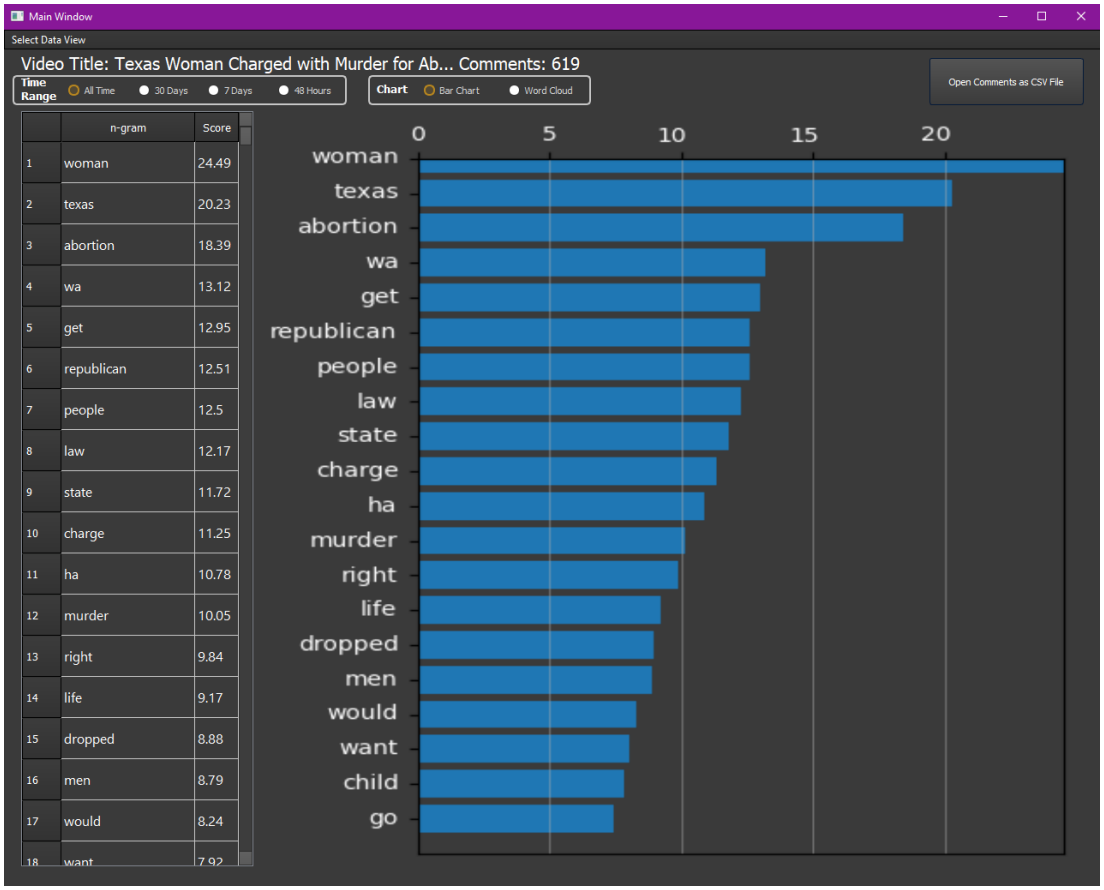
### A.1.15 Figure 4.11 Unigram word frequency and importance screens



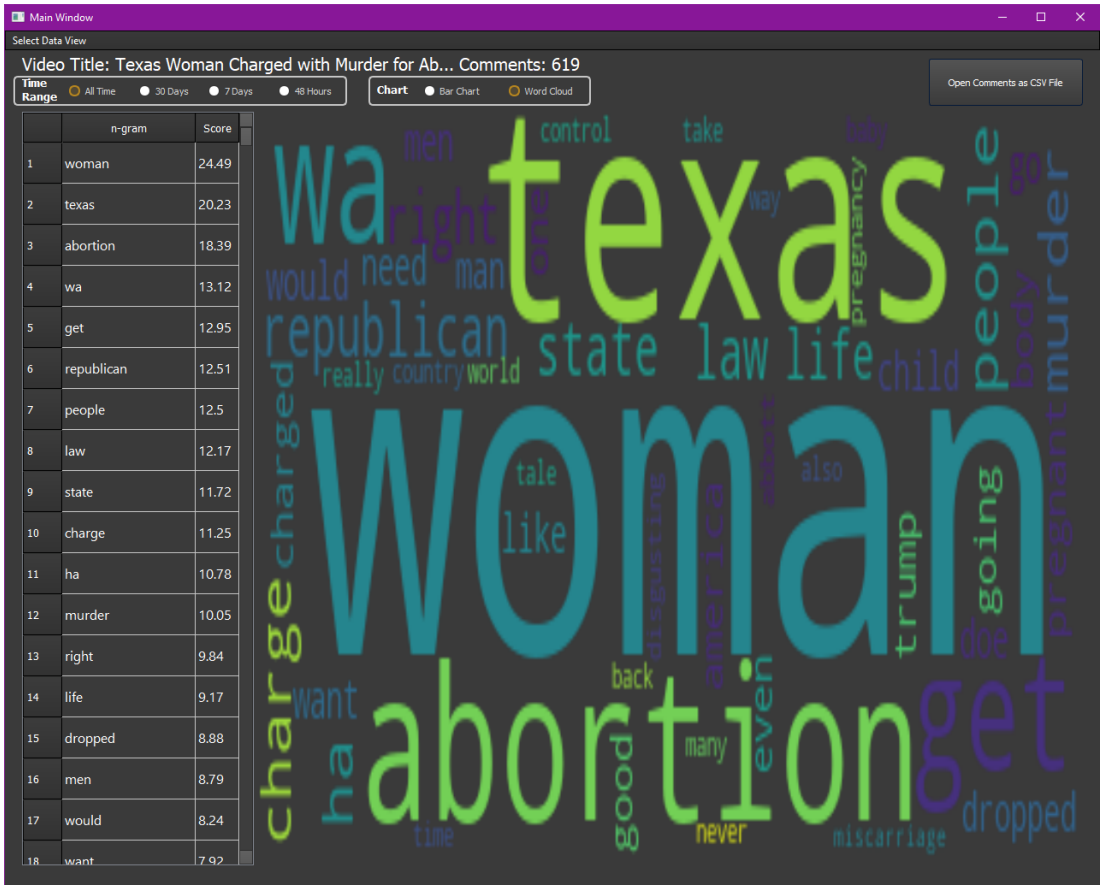

(a) Unigram frequency screen with bar chart




(b) Unigram frequency screen with word cloud

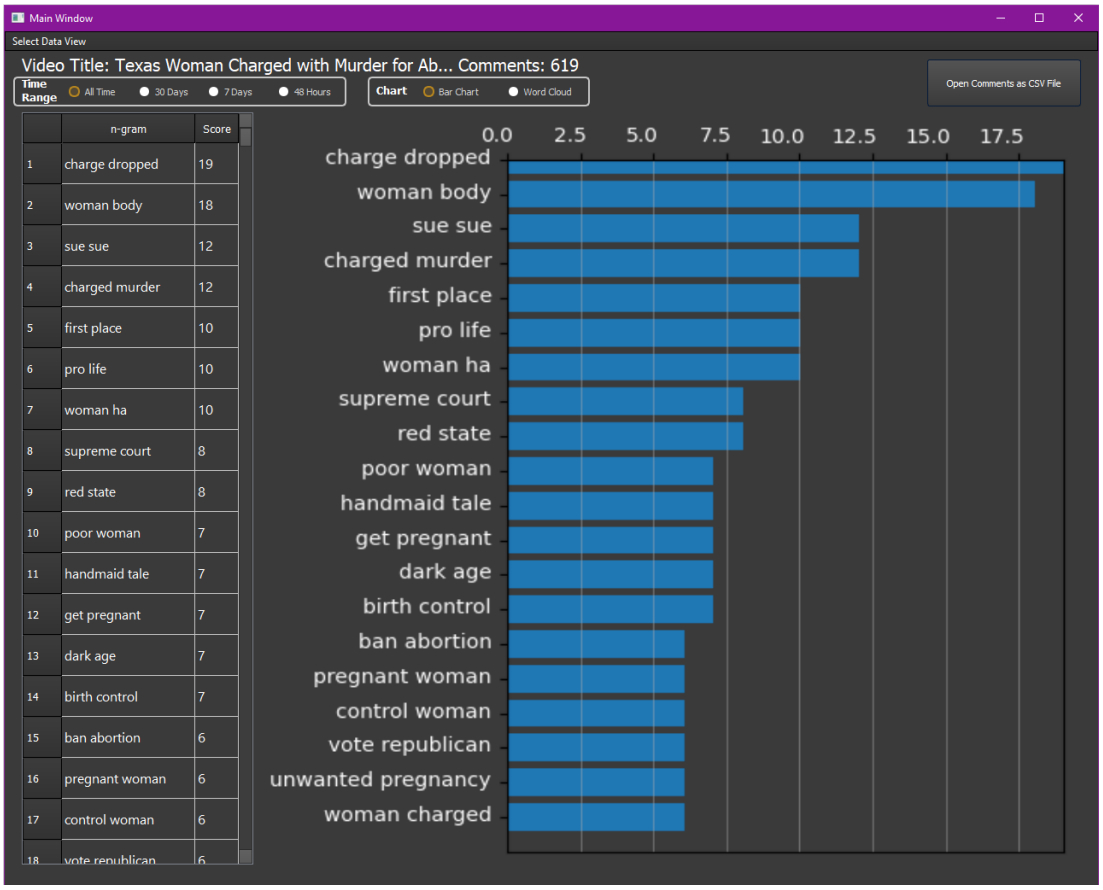



(c) Unigram importance screen with bar chart

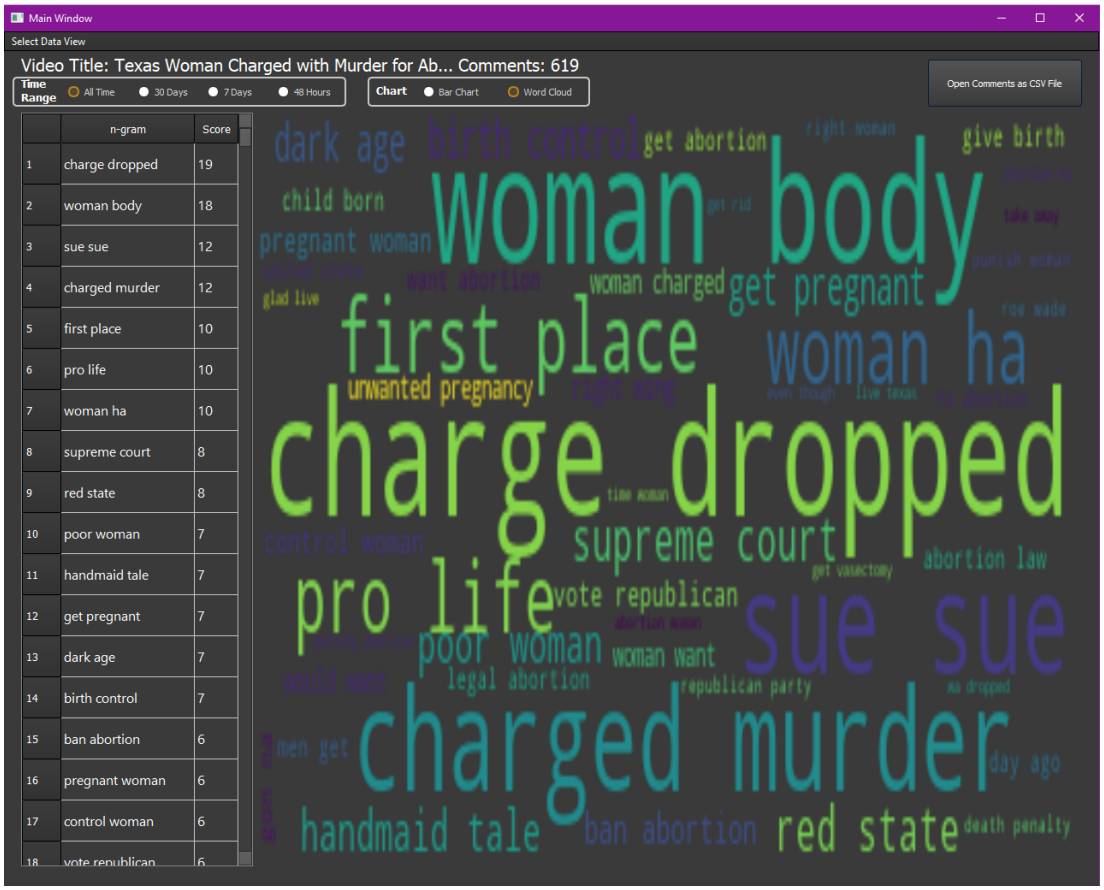



(d) Unigram importance screen with word cloud

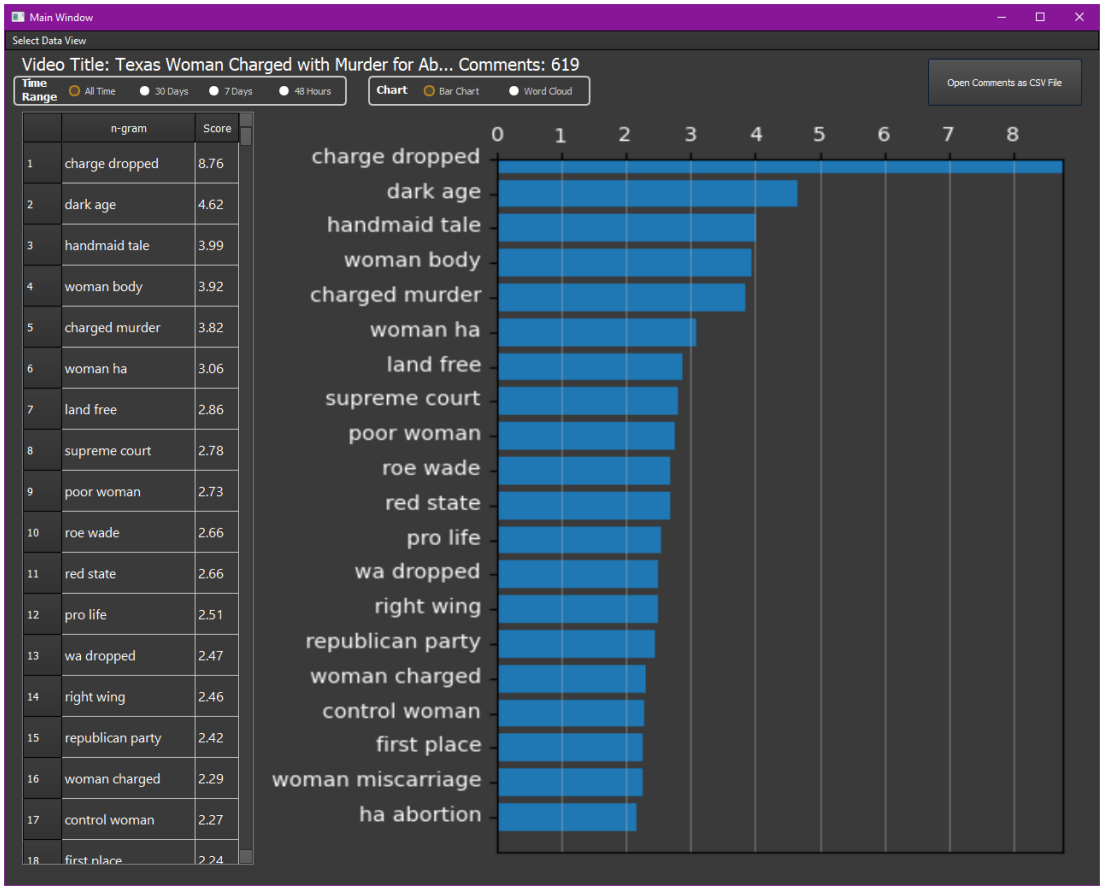
### A.1.16 Figure 4.12 Bigram word frequency and importance screens



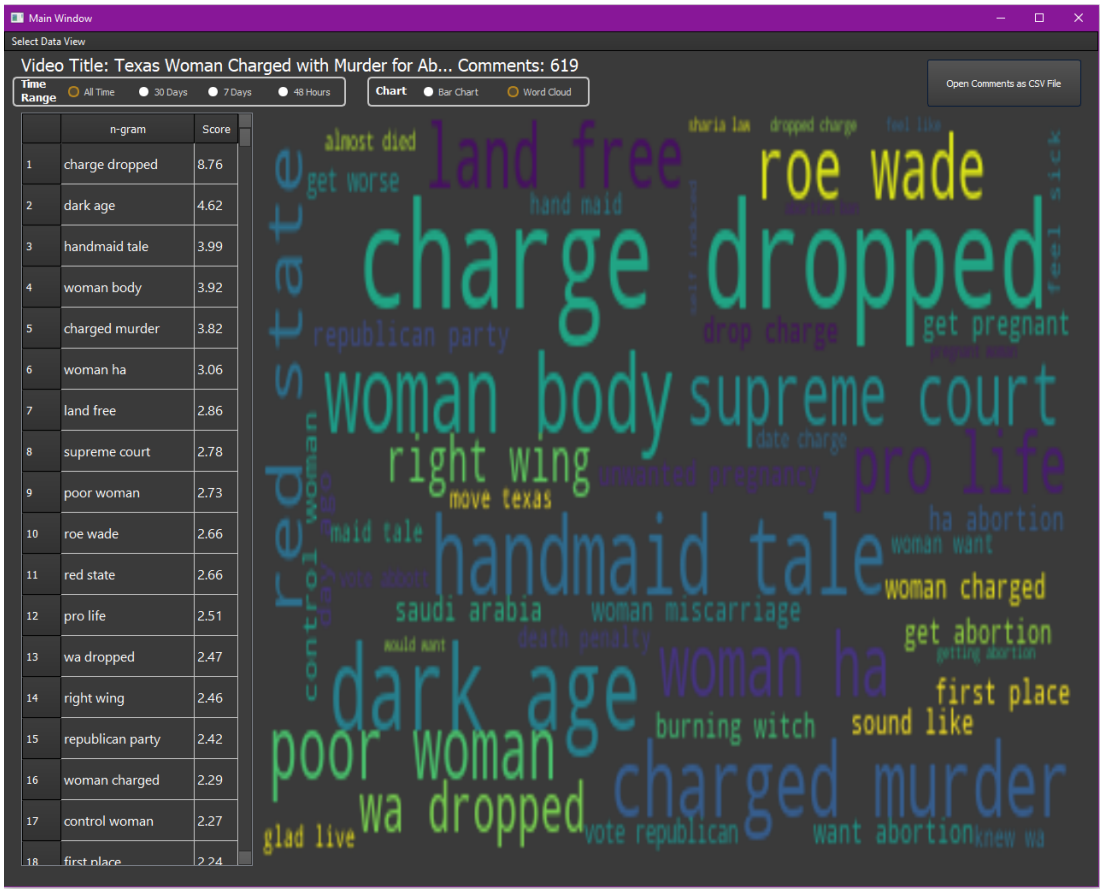

(a) Bigram frequency screen with bar chart




(b) Bigram frequency screen with word cloud

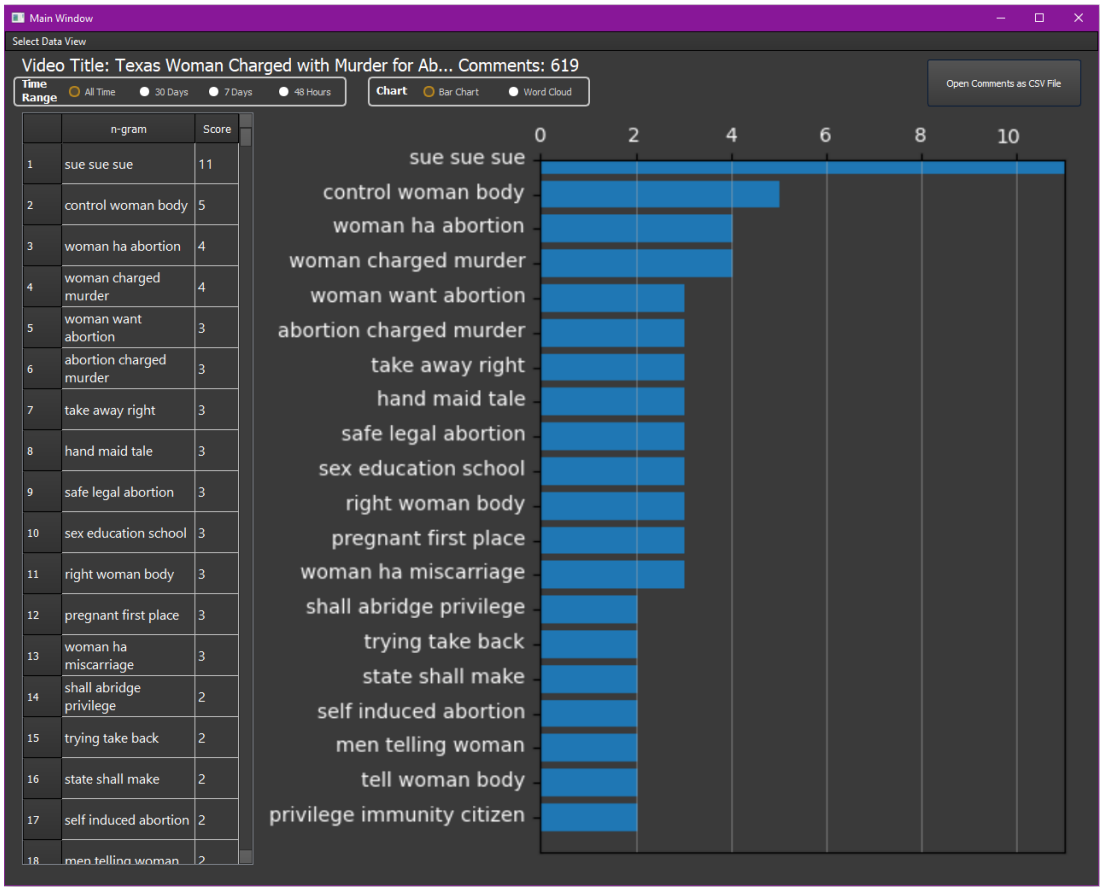



(c) Bigram importance screen with bar chart

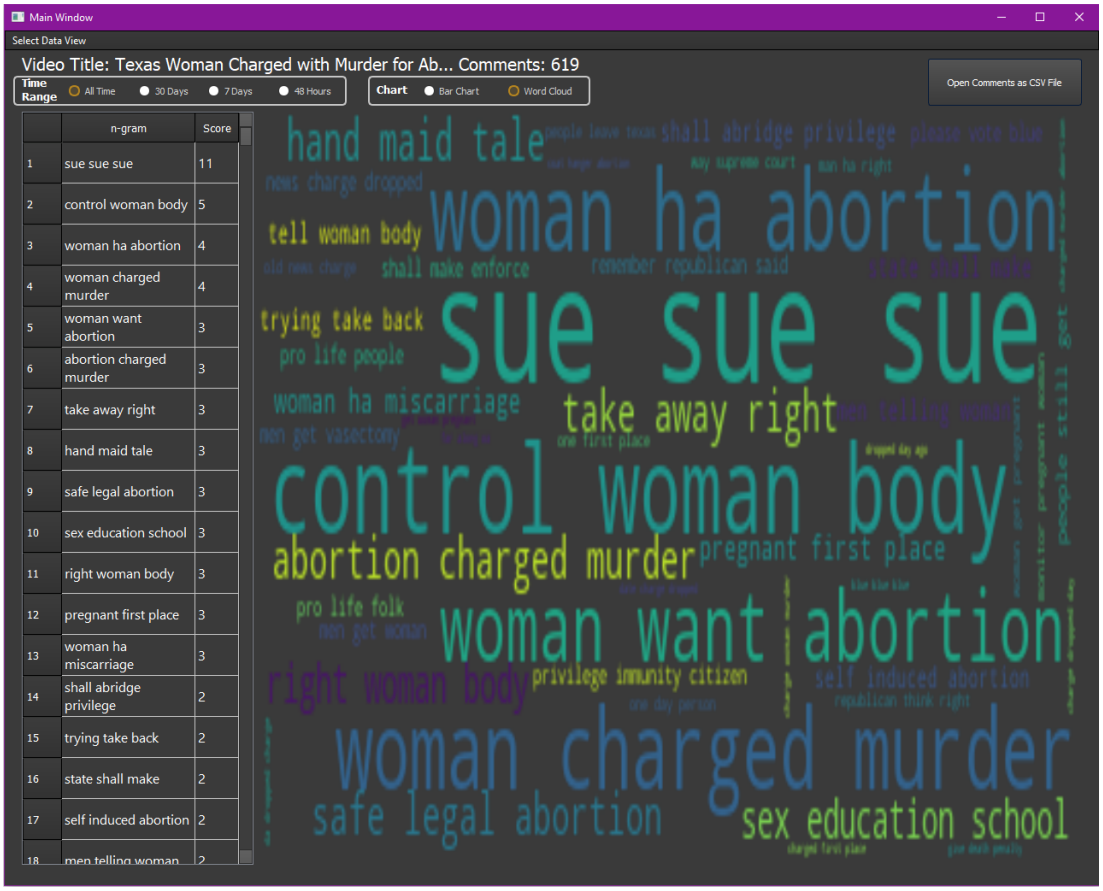



(d) Bigram importance screen with word cloud

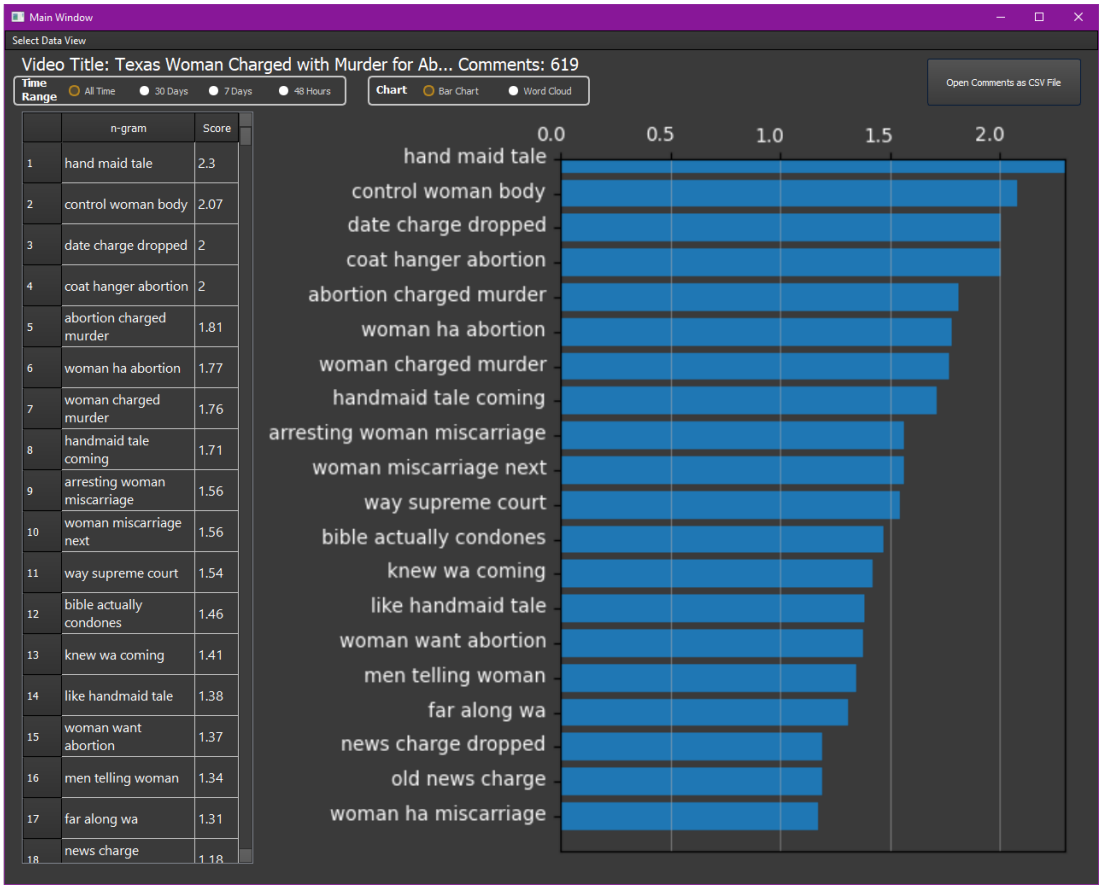
### A.1.17 Figure 4.13 Trigram word frequency and importance screens



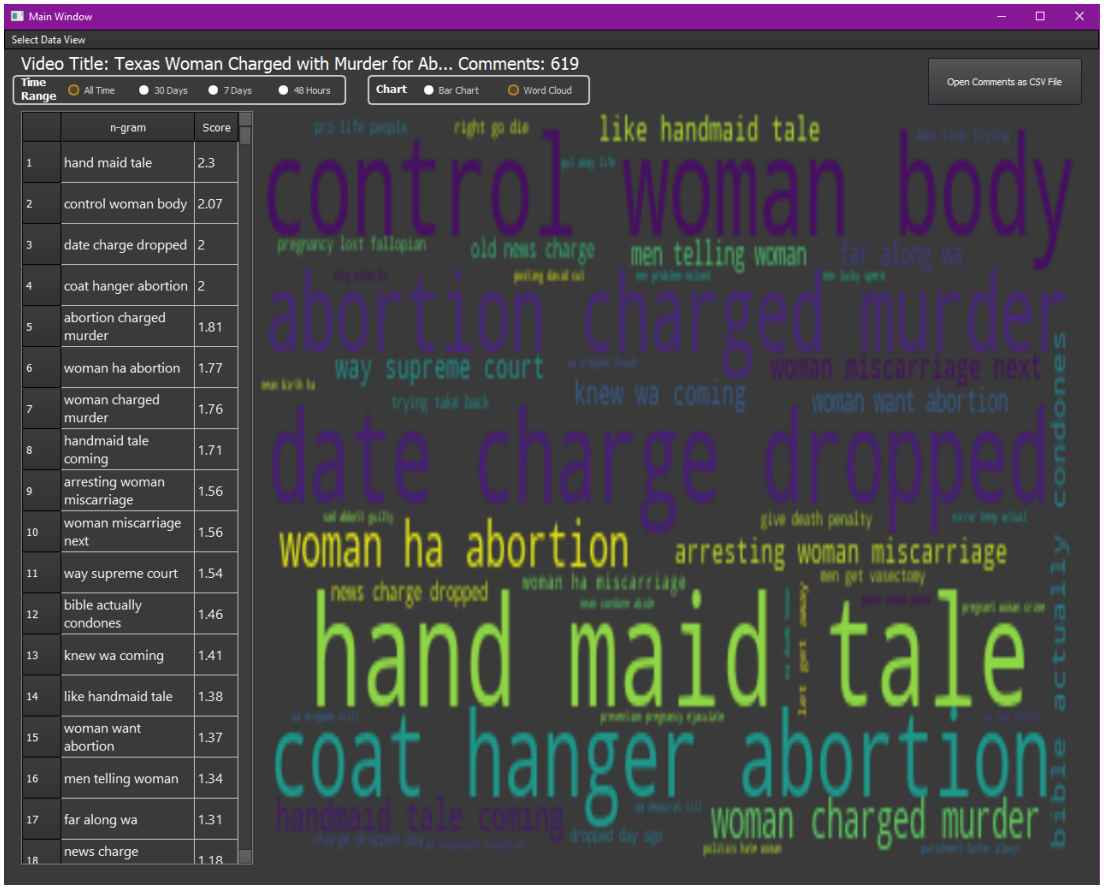

(a) Trigram frequency screen with bar chart




(b) Trigram frequency screen with word cloud

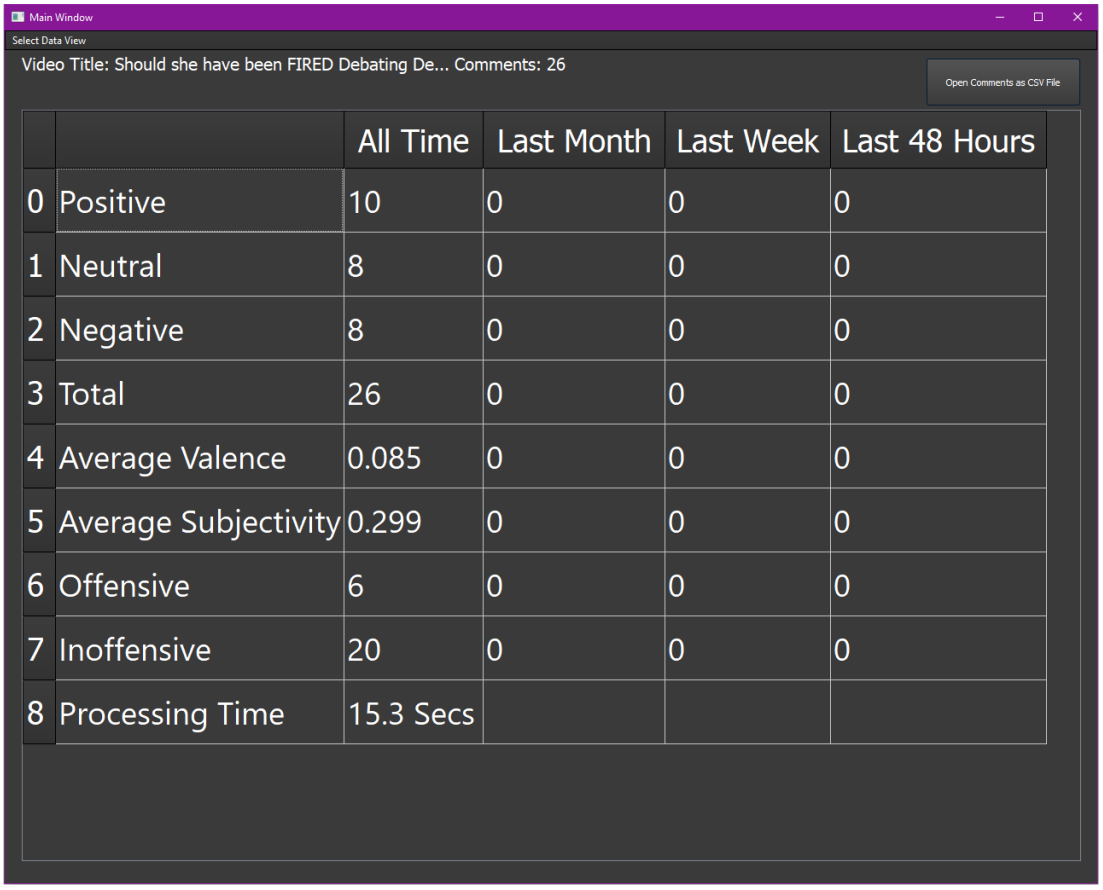



(c) Trigram importance screen with bar chart



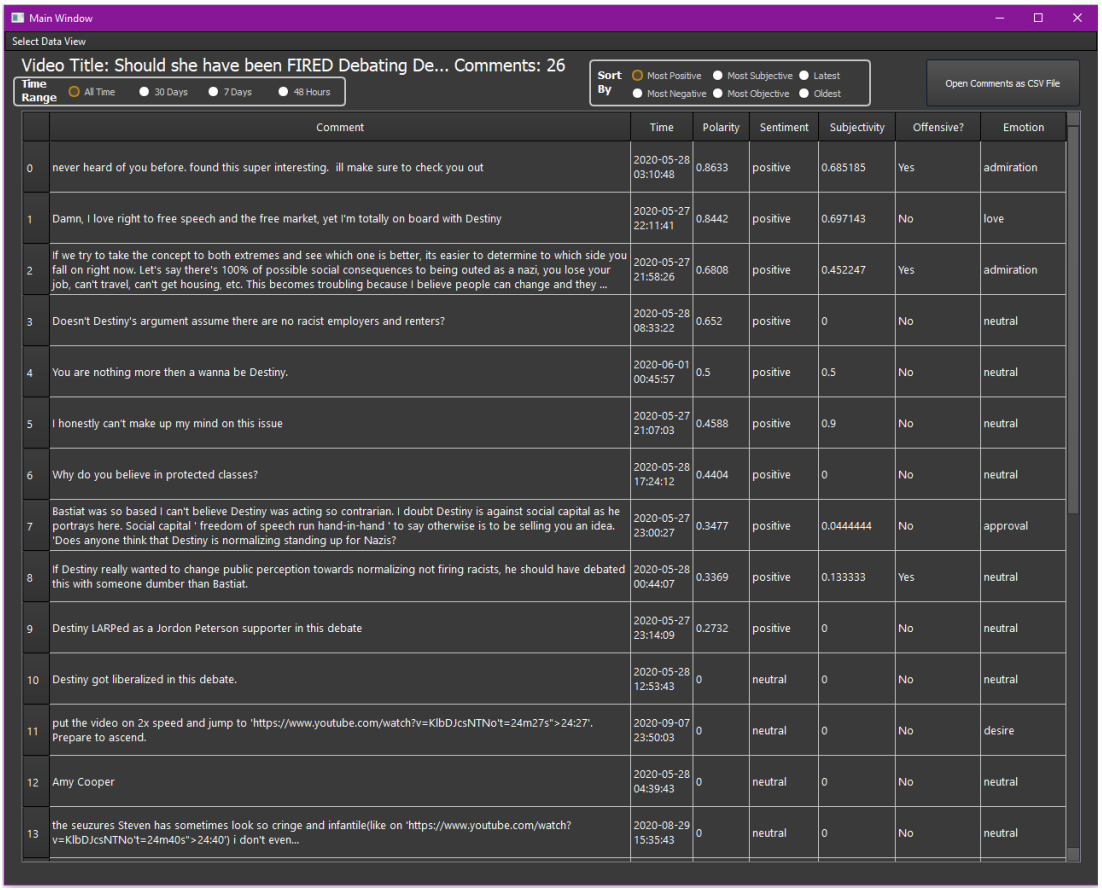

(d) Trigram importance screen with word cloud

### A.1.18 Figure 5.1 Test 1 GUI screens



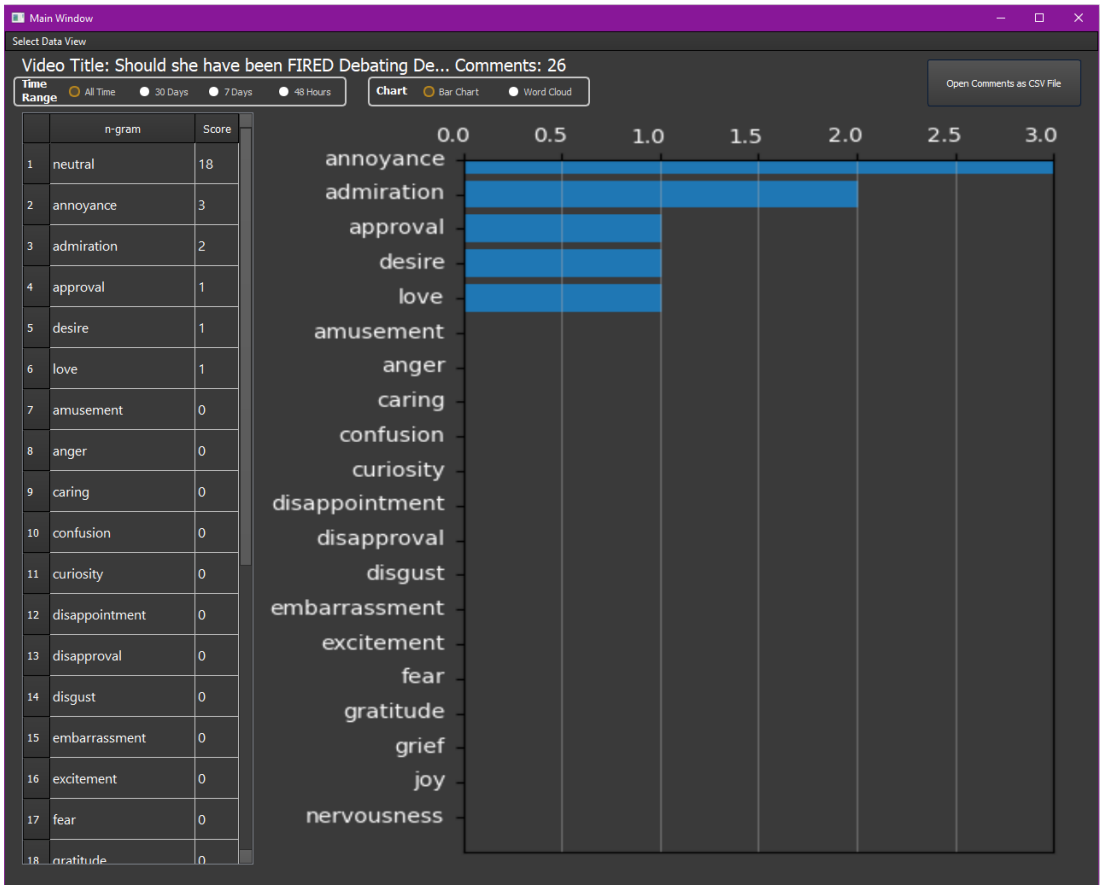

| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 10 | 0 | 0 | 0 |
| 1 | Neutral | 8 | 0 | 0 | 0 |
| 2 | Negative | 8 | 0 | 0 | 0 |
| 3 | Total | 26 | 0 | 0 | 0 |
| 4 | Average Valence | 0.085 | 0 | 0 | 0 |
| 5 | Average Subjectivity | 0.299 | 0 | 0 | 0 |
| 6 | Offensive | 6 | 0 | 0 | 0 |
| 7 | Inoffensive | 20 | 0 | 0 | 0 |
| 8 | Processing Time | 15.3 Secs | | | |

(a) Report screen

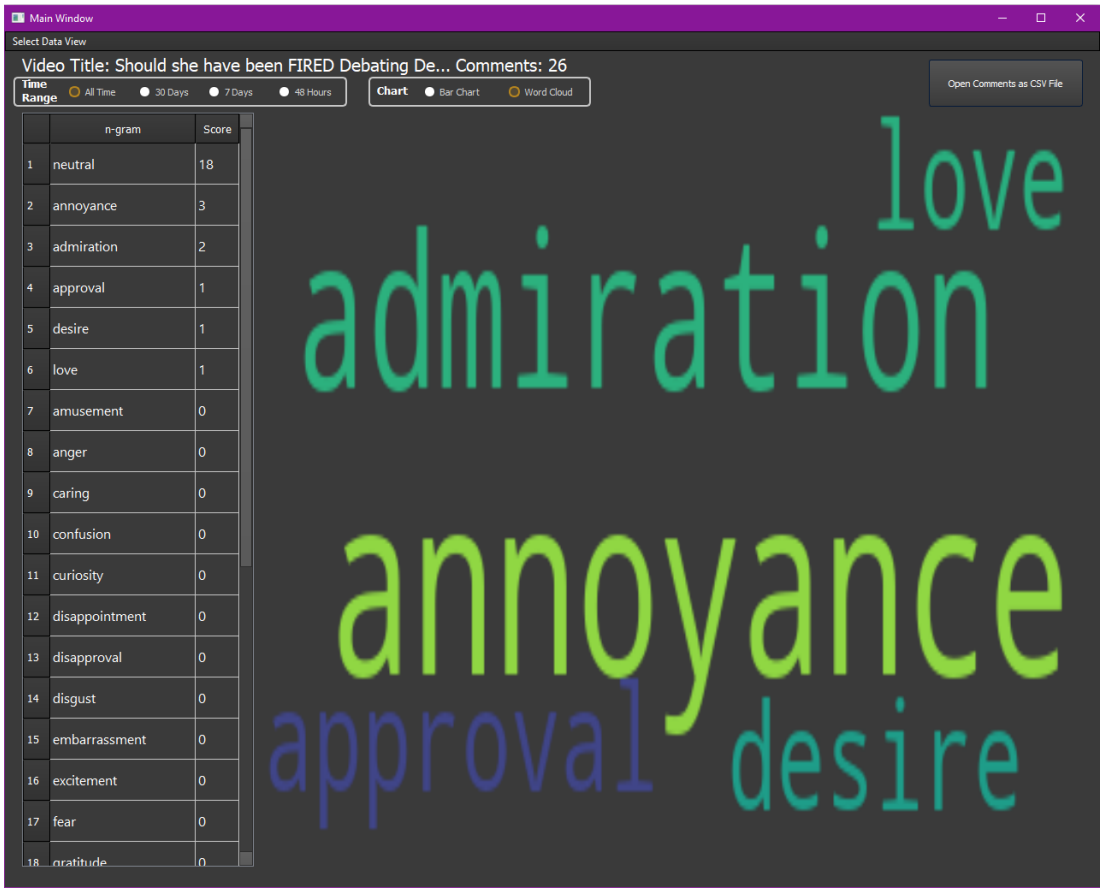



| | Comment | Time | Polarity | Sentiment | Subjectivity | Offensive? | Emotion |
|---|---|---|---|---|---|---|---|
| 0 | never heard of you before. found this super interesting. ill make sure to check you out | 2020-05-28 03:10:48 | 0.8633 | positive | 0.685185 | Yes | admiration |
| 1 | Damn, I love right to free speech and the free market, yet I'm totally on board with Destiny | 2020-05-27 22:11:41 | 0.8442 | positive | 0.697143 | No | love |
| 2 | If we try to take the concept to both extremes and see which one is better, its easier to determine to which side you fall on right now. Let's say there's 100% of possible social consequences to being outed as a nazi, you lose your job, can't travel, can't get housing, etc. This becomes troubling because I believe people can change and they ... | 2020-05-27 21:58:26 | 0.6808 | positive | 0.452247 | Yes | admiration |
| 3 | Doesn't Destiny's argument assume there are no racist employers and renters? | 2020-05-28 08:33:22 | 0.652 | positive | 0 | No | neutral |
| 4 | You are nothing more then a wanna be Destiny. | 2020-06-01 00:45:57 | 0.5 | positive | 0.5 | No | neutral |
| 5 | I honestly can't make up my mind on this issue | 2020-05-27 21:07:03 | 0.4588 | positive | 0.9 | No | neutral |
| 6 | Why do you believe in protected classes? | 2020-05-28 17:24:12 | 0.4404 | positive | 0 | No | neutral |
| 7 | Bastiat was so based I can't believe Destiny was acting so contrarian. I doubt Destiny is against social capital as he portrays here. Social capital ' freedom of speech run hand-in-hand ' to say otherwise is to be selling you an idea. 'Does anyone think that Destiny is normalizing standing up for Nazis? | 2020-05-27 23:00:27 | 0.3477 | positive | 0.0444444 | No | approval |
| 8 | If Destiny really wanted to change public perception towards normalizing not firing racists, he should have debated this with someone dumber than Bastiat. | 2020-05-28 00:44:07 | 0.3369 | positive | 0.133333 | Yes | neutral |
| 9 | Destiny LARPed as a Jordon Peterson supporter in this debate | 2020-05-27 23:14:09 | 0.2732 | positive | 0 | No | neutral |
| 10 | Destiny got liberalized in this debate. | 2020-05-28 12:53:43 | 0 | neutral | 0 | No | neutral |
| 11 | put the video on 2x speed and jump to 'https://www.youtube.com/watch?v=KlbDJcsNTNo't=24m27s">24:27'. Prepare to ascend. | 2020-09-07 23:50:03 | 0 | neutral | 0 | No | desire |
| 12 | Amy Cooper | 2020-05-28 04:39:43 | 0 | neutral | 0 | No | neutral |
| 13 | the seuzures Steven has sometimes look so cringe and infantile(like on 'https://www.youtube.com/watch?v=KlbDJcsNTNo't=24m40s">24:40') i don't even... | 2020-08-29 15:35:43 | 0 | neutral | 0 | No | neutral |

(b) Comments screen

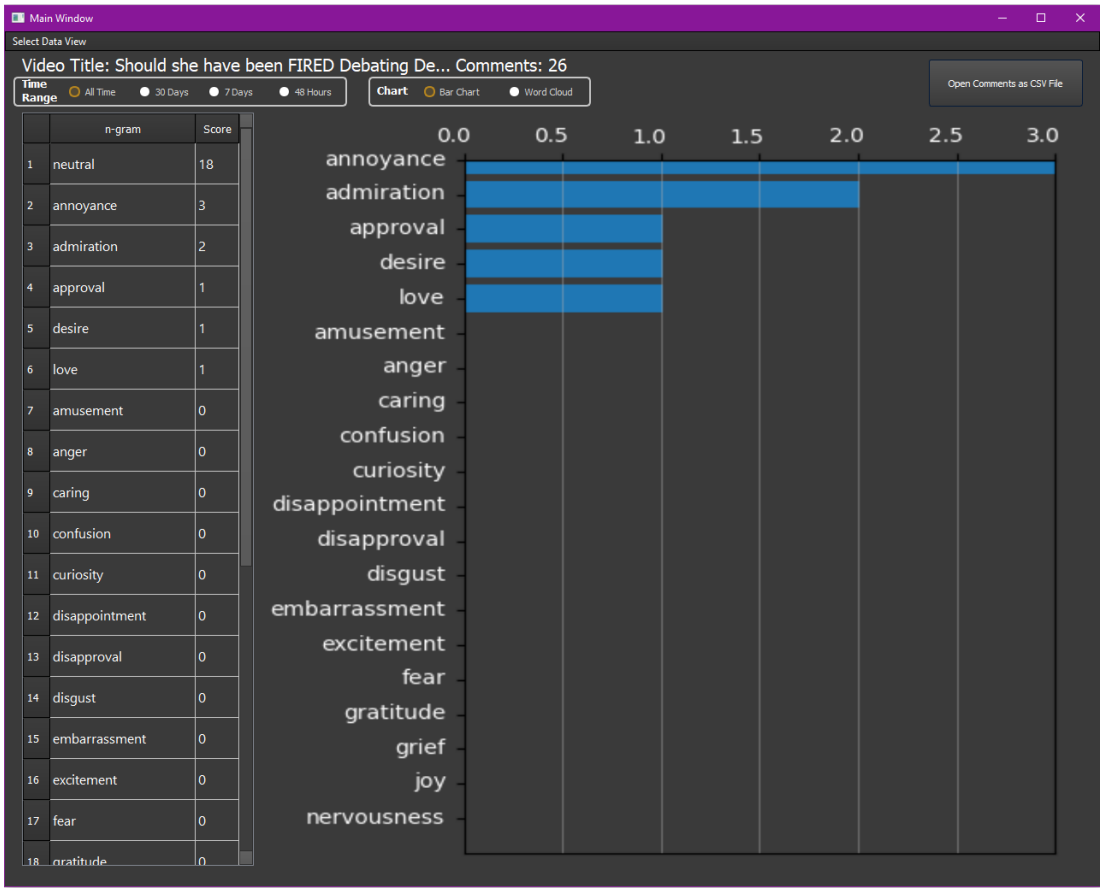



(c) Emotional analysis bar chart screen

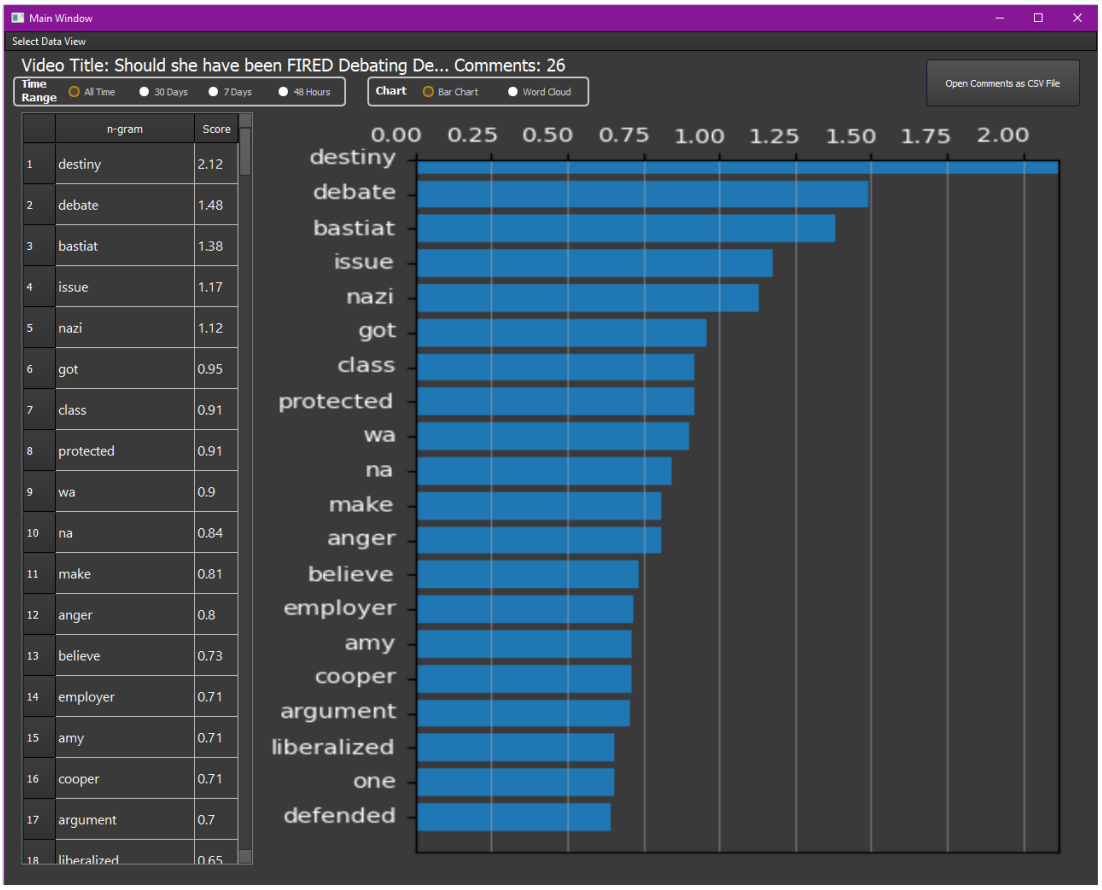



(d) Emotional analysis word cloud screen

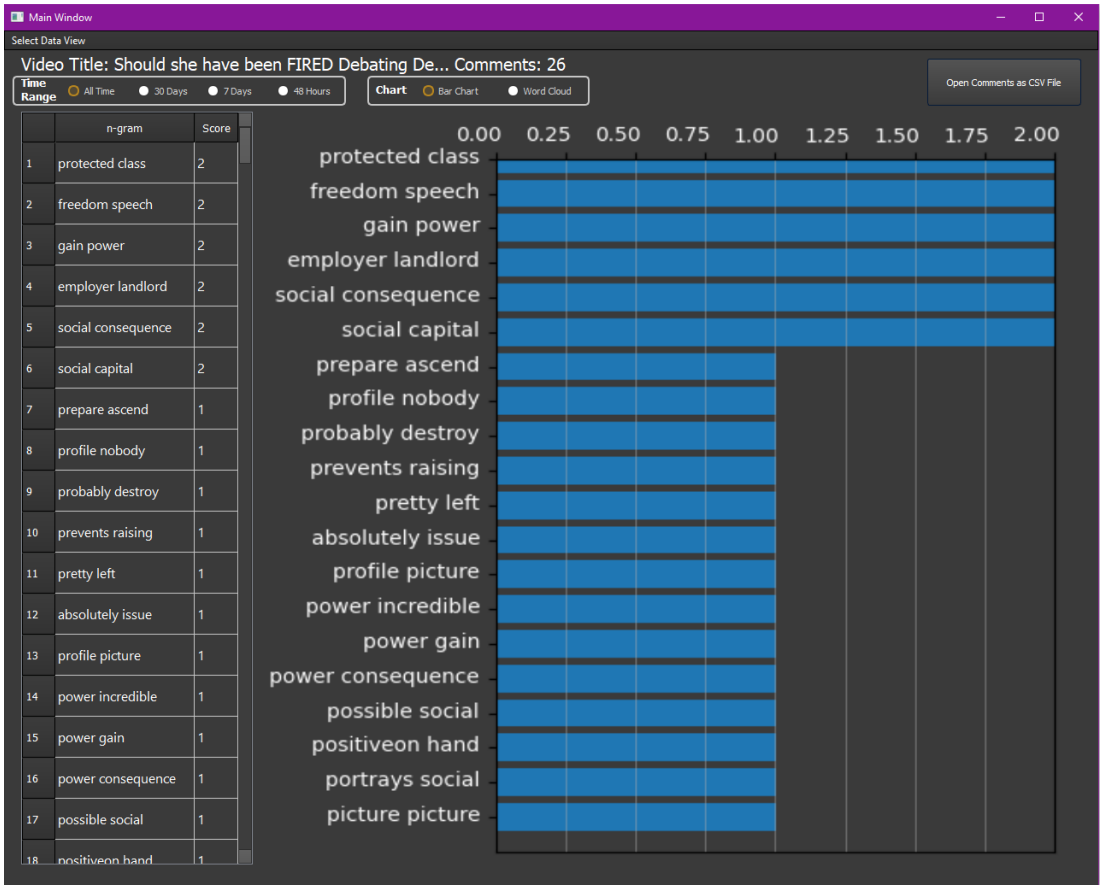
### A.1.19 Figure 5.2 Test 1 GUI screens



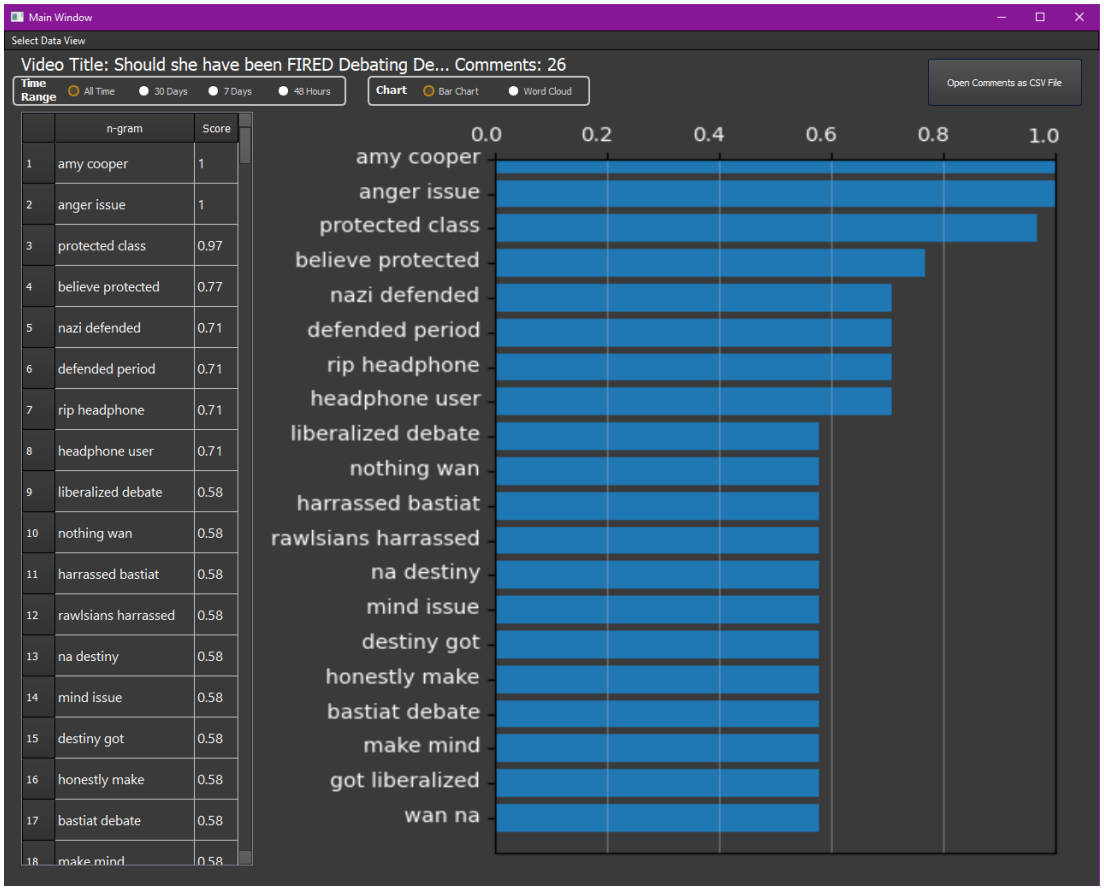

(a) Unigram frequency screen



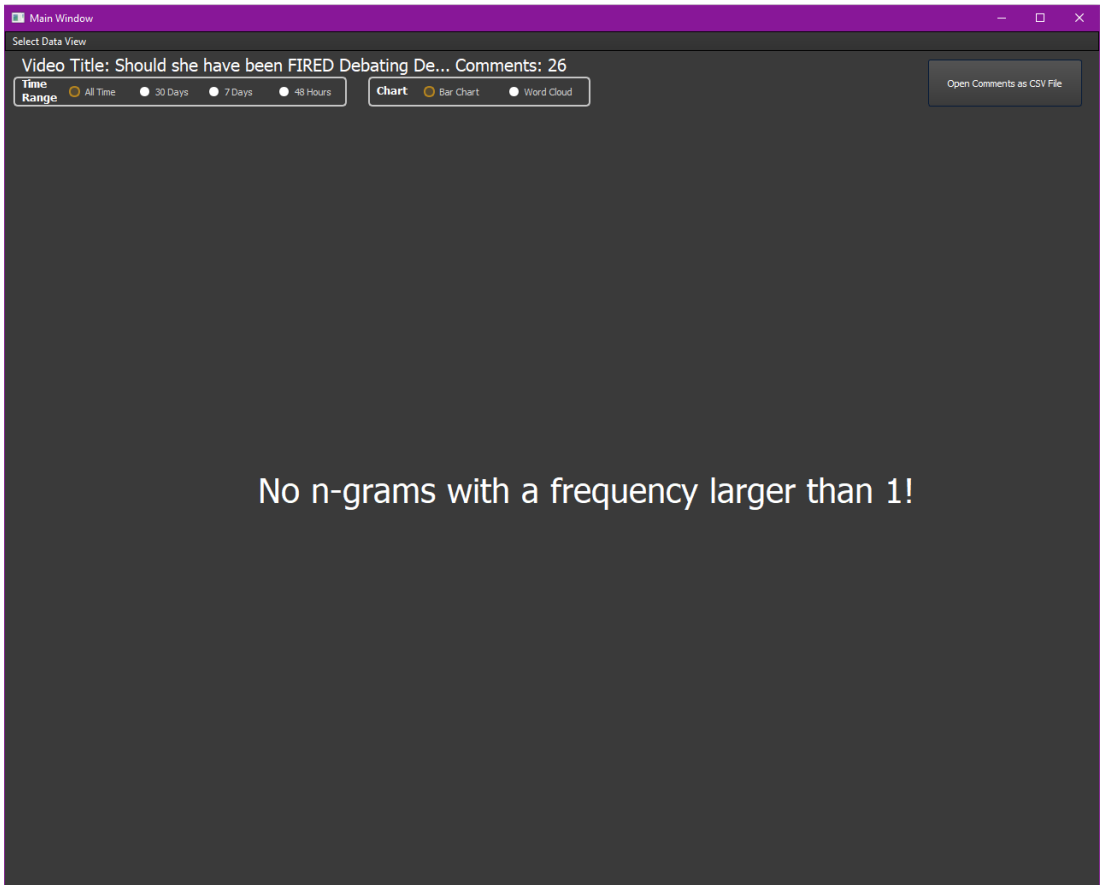

(b) Unigram importance screen



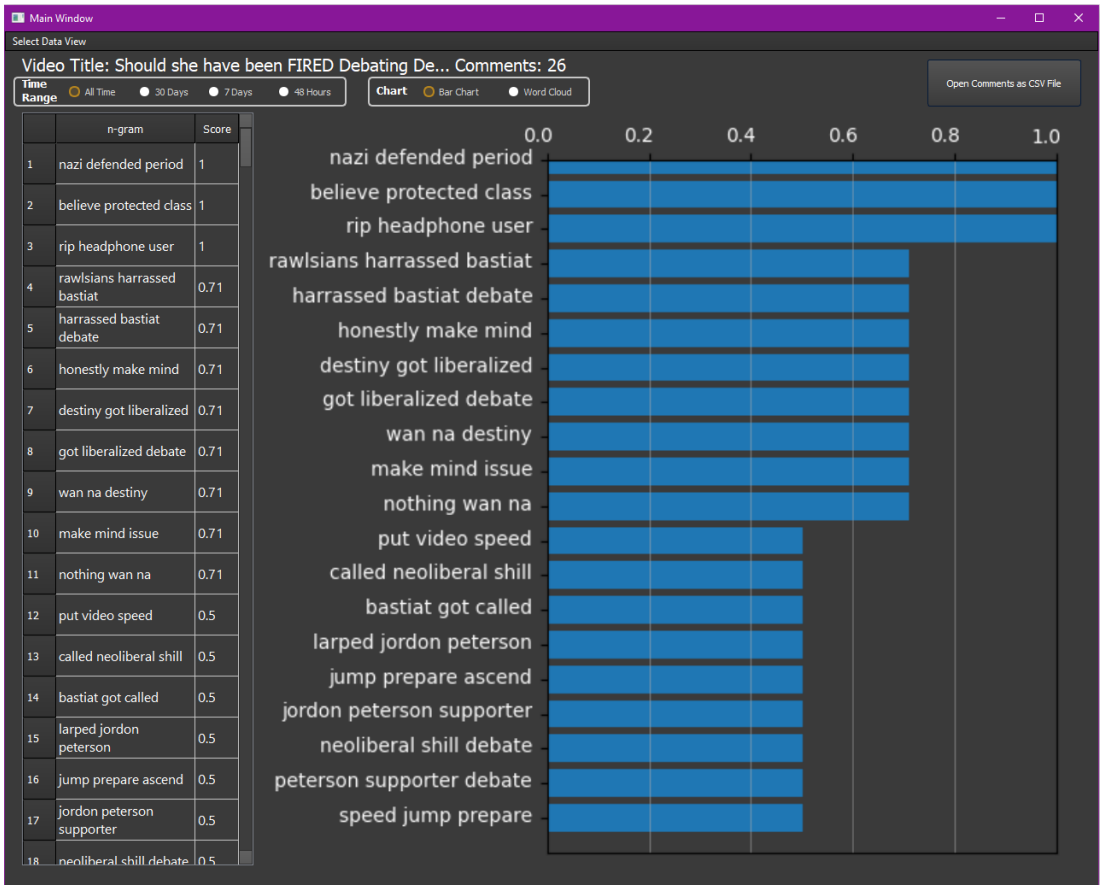

(c) Bigram frequency screen




(d) Bigram importance screen

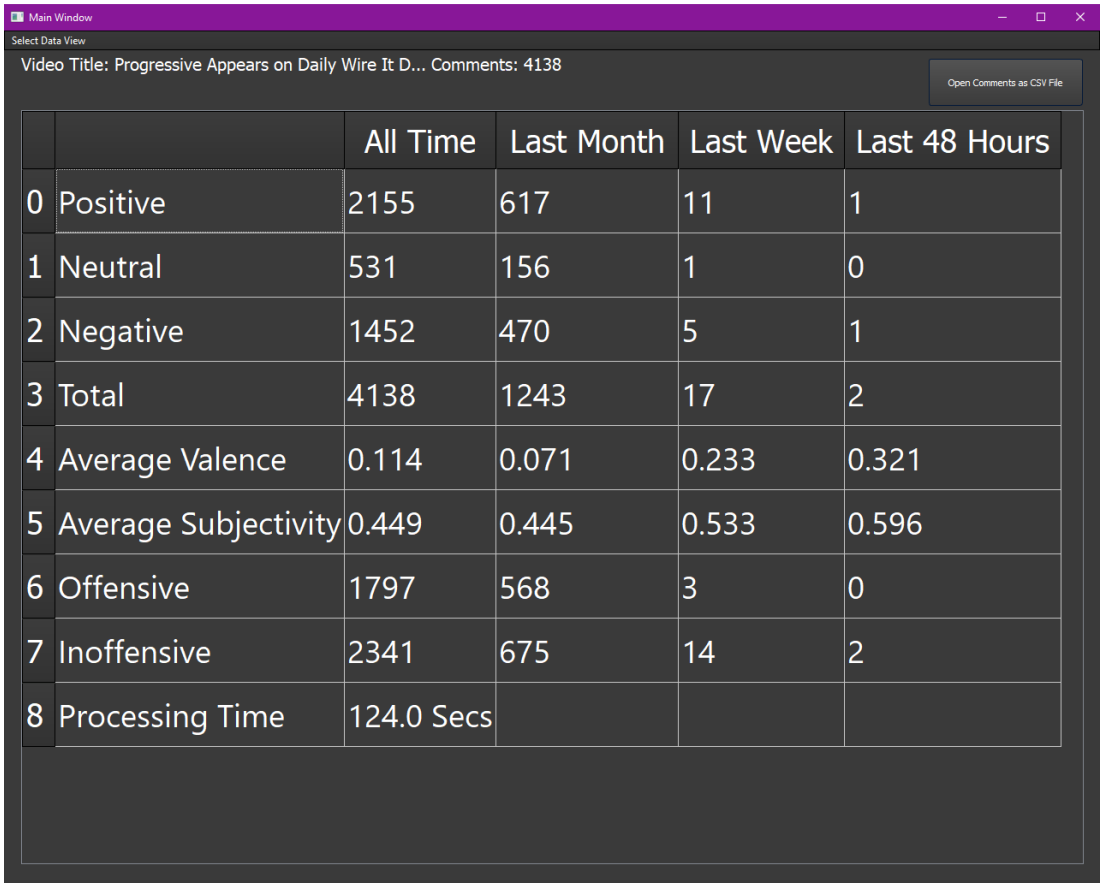



(e) Trigram frequency screen



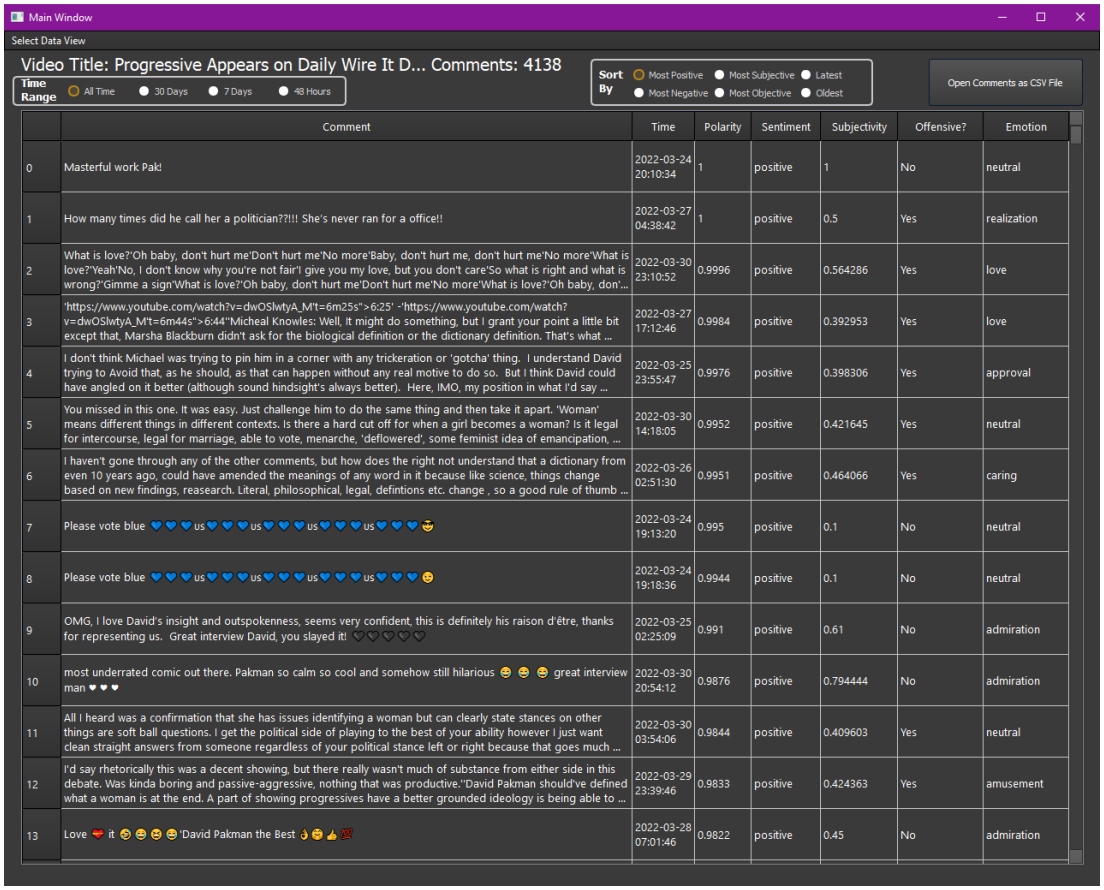

(f) Trigram importance screen

### A.1.20 Figure 5.3 Test 2 GUI screens



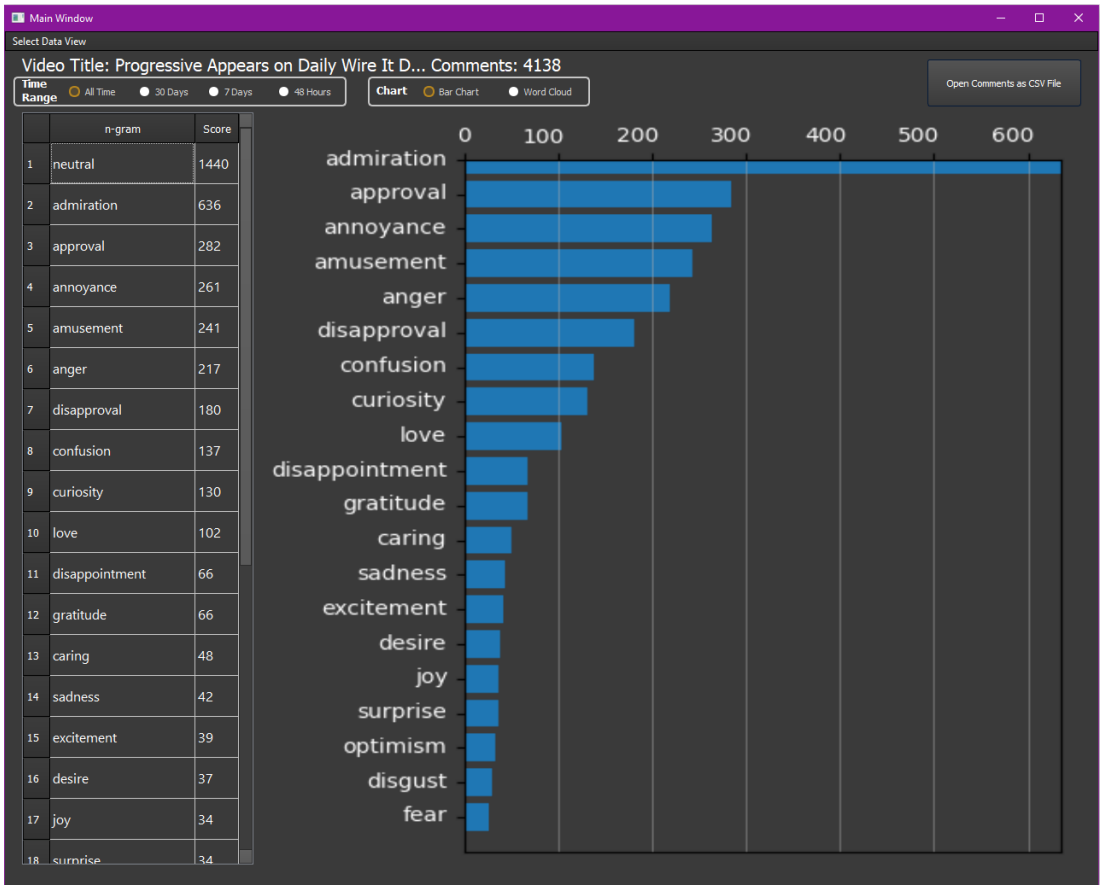

| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 2155 | 617 | 11 | 1 |
| 1 | Neutral | 531 | 156 | 1 | 0 |
| 2 | Negative | 1452 | 470 | 5 | 1 |
| 3 | Total | 4138 | 1243 | 17 | 2 |
| 4 | Average Valence | 0.114 | 0.071 | 0.233 | 0.321 |
| 5 | Average Subjectivity | 0.449 | 0.445 | 0.533 | 0.596 |
| 6 | Offensive | 1797 | 568 | 3 | 0 |
| 7 | Inoffensive | 2341 | 675 | 14 | 2 |
| 8 | Processing Time | 124.0 Secs | | | |

(a) Report screen

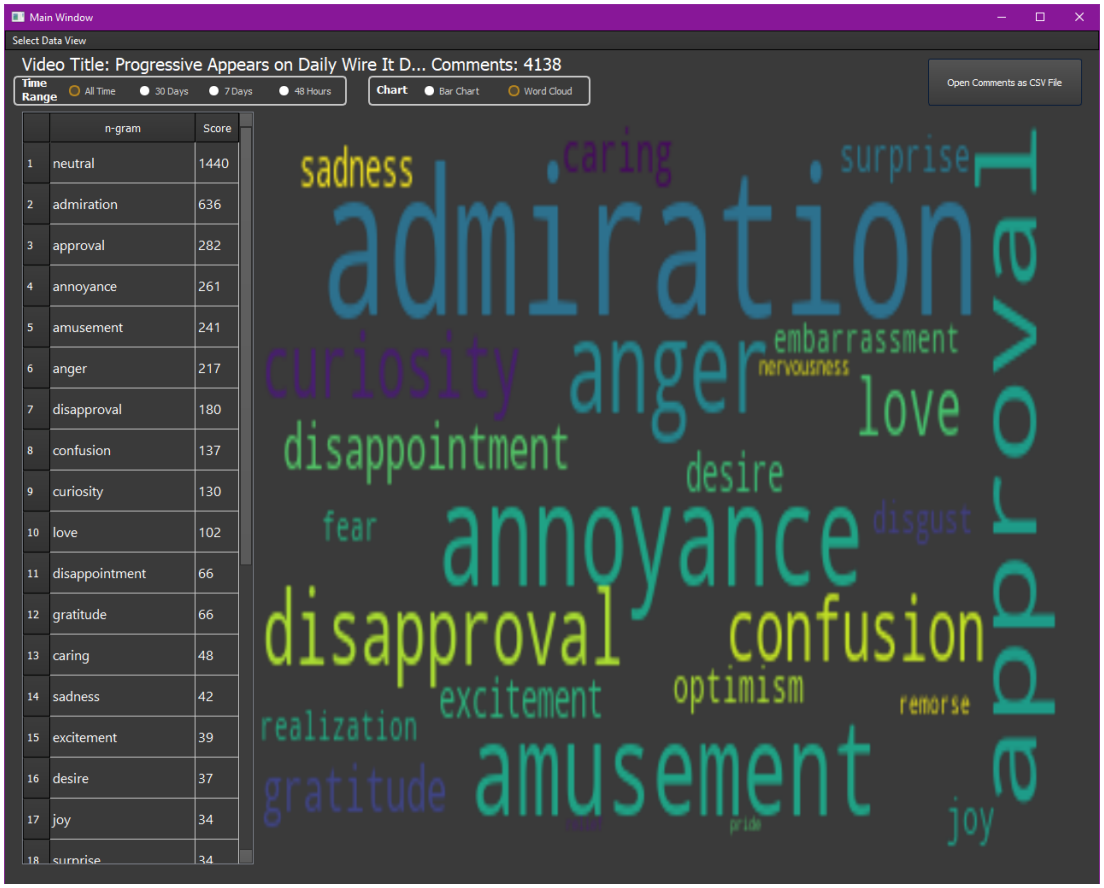



| | Comment | Time | Polarity | Sentiment | Subjectivity | Offensive? | Emotion |
|---|---|---|---|---|---|---|---|
| 0 | Masterful work Pak! | 2022-03-24 20:10:34 | 1 | positive | 1 | No | neutral |
| 1 | How many times did he call her a politician??!!! She's never ran for a office!! | 2022-03-27 04:38:42 | 1 | positive | 0.5 | Yes | realization |
| 2 | What is love?'Oh baby, don't hurt me'Don't hurt me'No more'Baby, don't hurt me, don't hurt me'No more'What is love?'Yeah'No, I don't know why you're not fair'I give you my love, but you don't care'So what is right and what is wrong?'Gimme a sign'What is love?'Oh baby, don't hurt me'Don't hurt me'No more'What is love?'Oh baby, don'... | 2022-03-30 23:10:52 | 0.9996 | positive | 0.564286 | Yes | love |
| 3 | 'https://www.youtube.com/watch?v=dwOSlwtyA_M't=6m25s">6:25' -'https://www.youtube.com/watch?v=dwOSlwtyA_M't=6m44s">6:44"Micheal Knowles: Well, it might do something, but I grant your point a little bit except that. Marsha Blackburn didn't ask for the biological definition or the dictionary definition. That's what ... | 2022-03-27 17:12:46 | 0.9984 | positive | 0.392953 | Yes | love |
| 4 | I don't think Michael was trying to pin him in a corner with any trickeration or 'gotcha' thing. I understand David trying to Avoid that, as he should, as that can happen without any real motive to do so. But I think David could have angled on it better (although sound hindsight's always better). Here, IMO, my position in what I'd say ... | 2022-03-25 23:55:47 | 0.9976 | positive | 0.398306 | Yes | approval |
| 5 | You missed in this one. It was easy. Just challenge him to do the same thing and then take it apart. 'Woman' means different things in different contexts. Is there a hard cut off for when a girl becomes a woman? Is it legal for intercourse, legal for marriage, able to vote, menarche, 'deflowered', some feminist idea of emancipation, ... | 2022-03-30 14:18:05 | 0.9952 | positive | 0.421645 | Yes | neutral |
| 6 | I haven't gone through any of the other comments, but how does the right not understand that a dictionary from even 10 years ago, could have amended the meanings of any word in it because like science, things change based on new findings, reasearch. Literal, philosophical, legal, defintions etc. change , so a good rule of thumb ... | 2022-03-26 02:51:30 | 0.9951 | positive | 0.464066 | Yes | caring |
| 7 | Please vote blue 💙💙💙us💙💙💙us💙💙💙us💙💙💙us💙💙💙😎 | 2022-03-24 19:13:20 | 0.995 | positive | 0.1 | No | neutral |
| 8 | Please vote blue 💙💙💙us💙💙💙us💙💙💙us💙💙💙us💙💙💙😉 | 2022-03-24 19:18:36 | 0.9944 | positive | 0.1 | No | neutral |
| 9 | OMG, I love David's insight and outspokenness, seems very confident, this is definitely his raison d'être, thanks for representing us. Great interview David, you slayed it! 🖤🖤🖤🖤🖤 | 2022-03-25 02:25:09 | 0.991 | positive | 0.61 | No | admiration |
| 10 | most underrated comic out there. Pakman so calm so cool and somehow still hilarious 😂 😂 😂 great interview man ♥ ♥ ♥ | 2022-03-30 20:54:12 | 0.9876 | positive | 0.794444 | No | admiration |
| 11 | All I heard was a confirmation that she has issues identifying a woman but can clearly state stances on other things are soft ball questions. I get the political side of playing to the best of your ability however I just want clean straight answers from someone regardless of your political stance left or right because that goes much ... | 2022-03-30 03:54:06 | 0.9844 | positive | 0.409603 | Yes | neutral |
| 12 | I'd say rhetorically this was a decent showing, but there really wasn't much of substance from either side in this debate. Was kinda boring and passive-aggressive, nothing that was productive."David Pakman should've defined what a woman is at the end. A part of showing progressives have a better grounded ideology is being able to ... | 2022-03-29 23:39:46 | 0.9833 | positive | 0.424363 | Yes | amusement |
| 13 | Love 💋 it 😉😂😂😂'David Pakman the Best 👌😁👍💯 | 2022-03-28 07:01:46 | 0.9822 | positive | 0.45 | No | admiration |

(b) Comments screen

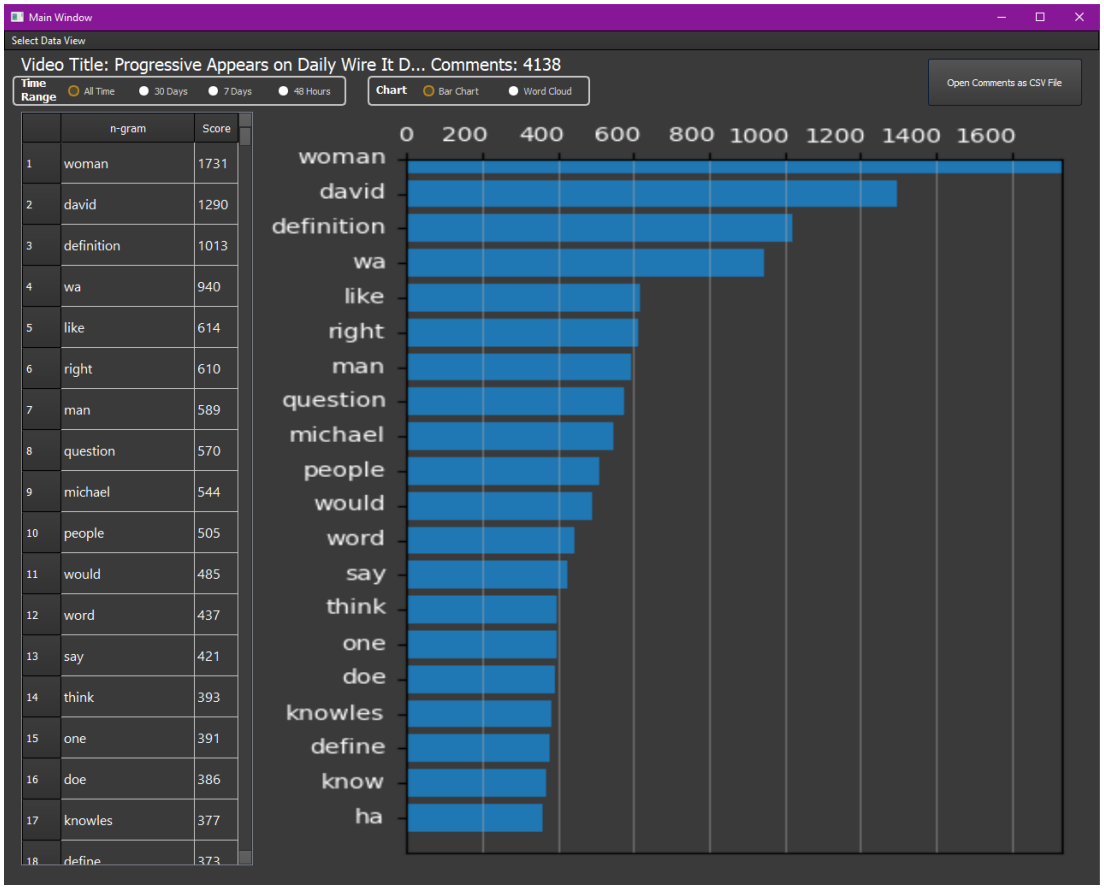



(c) Emotional analysis bar chart screen

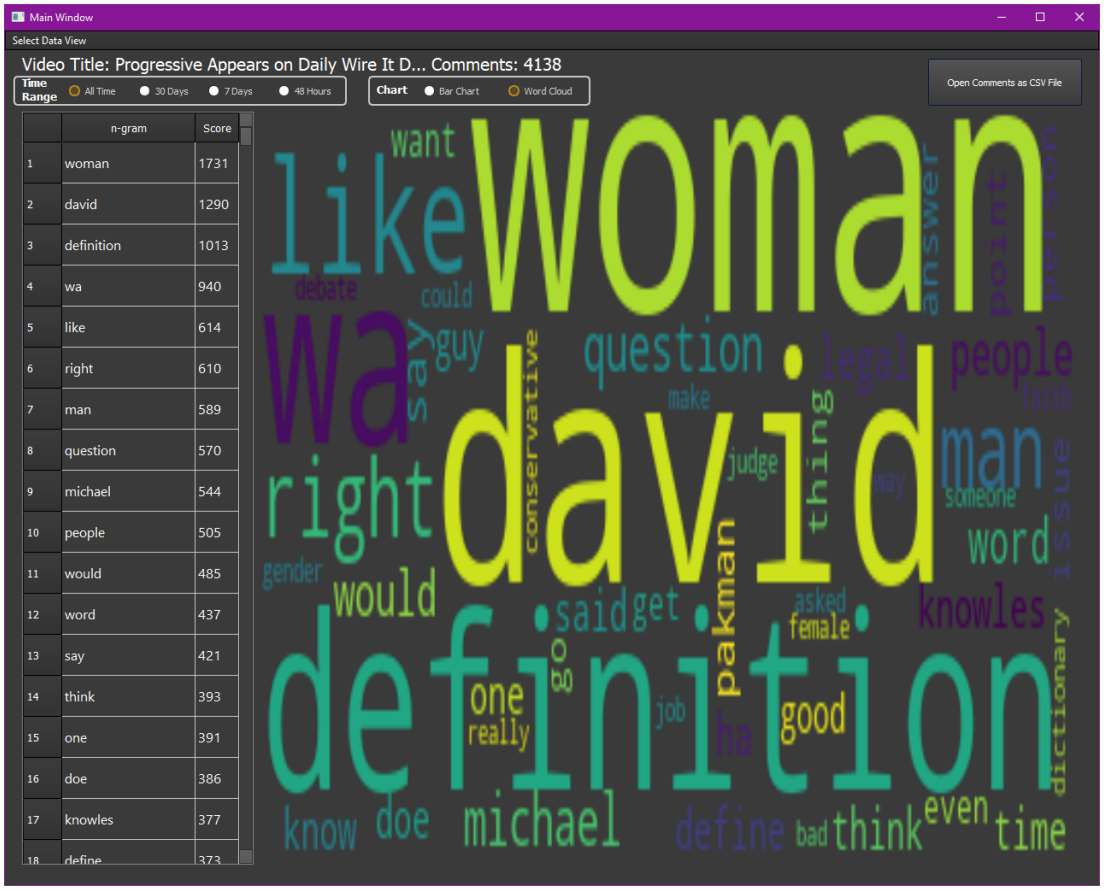



(d) Emotional analysis word cloud screen

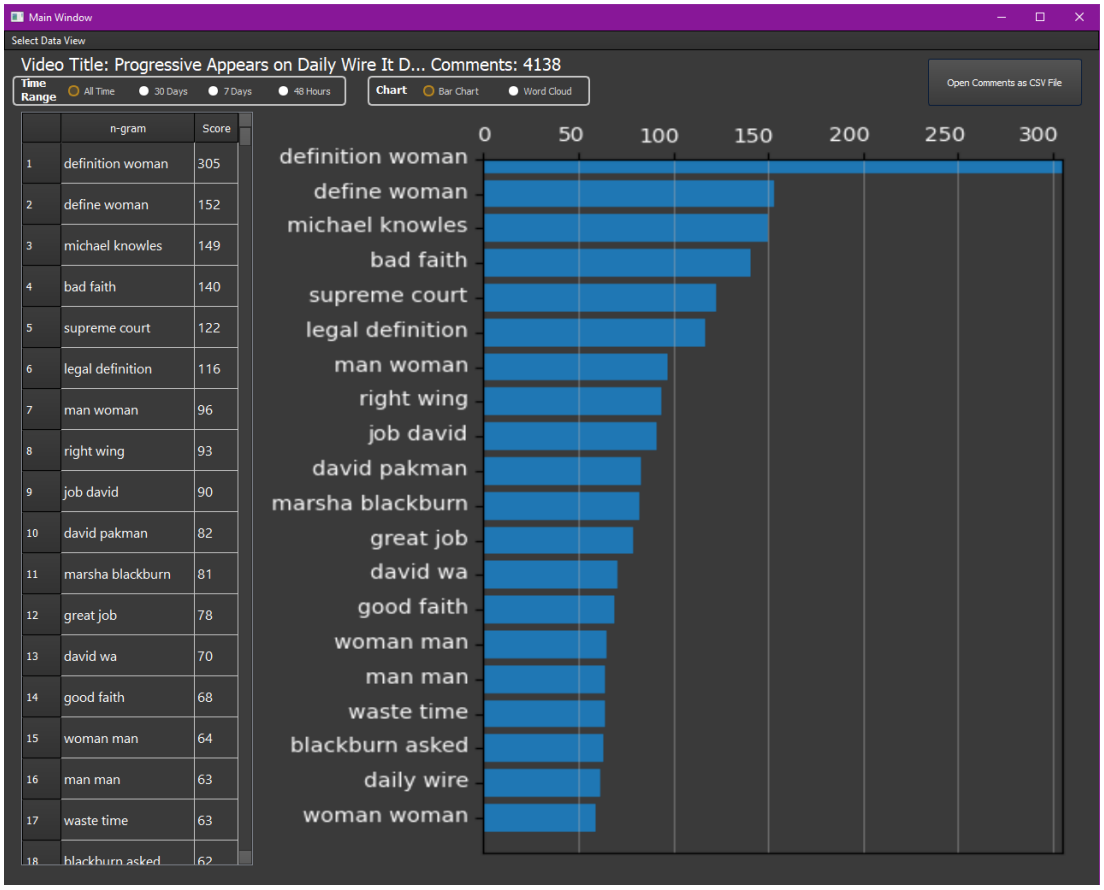
### A.1.21 Figure 5.4 Test 2 GUI screens



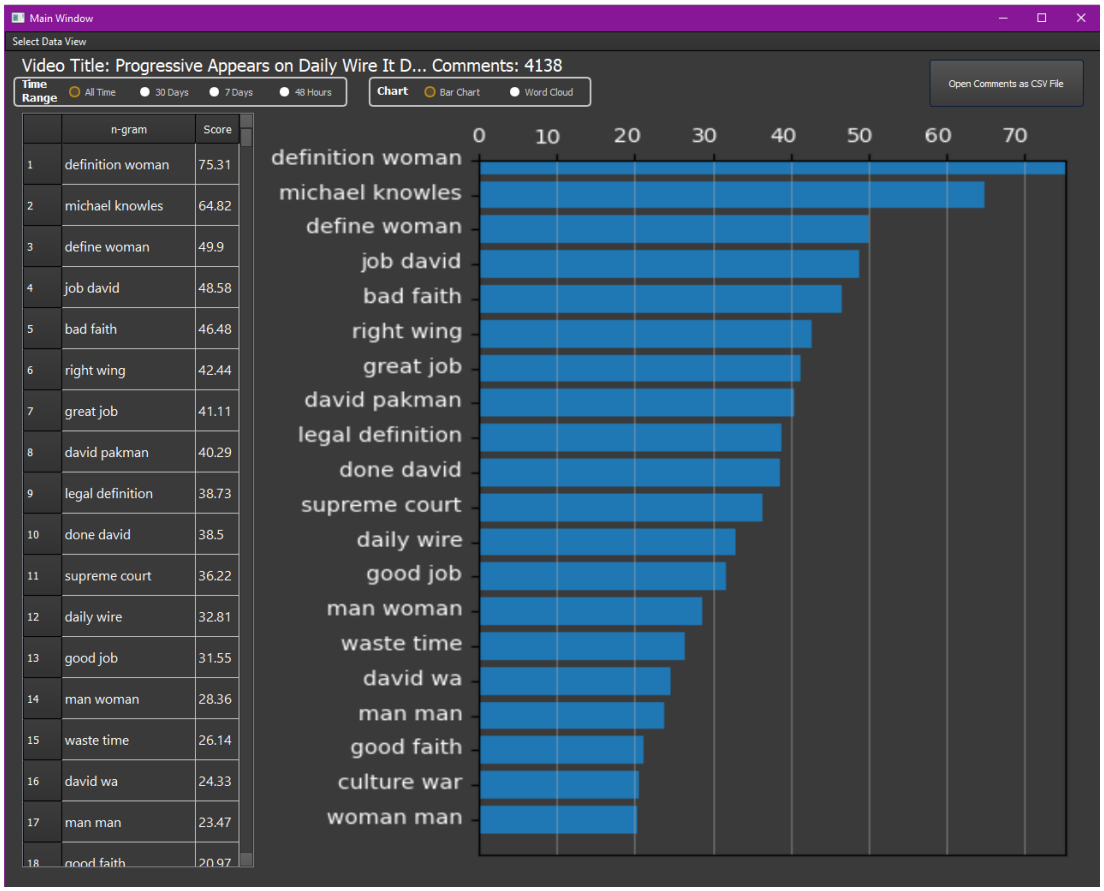

(a) Unigram frequency bar chart screen



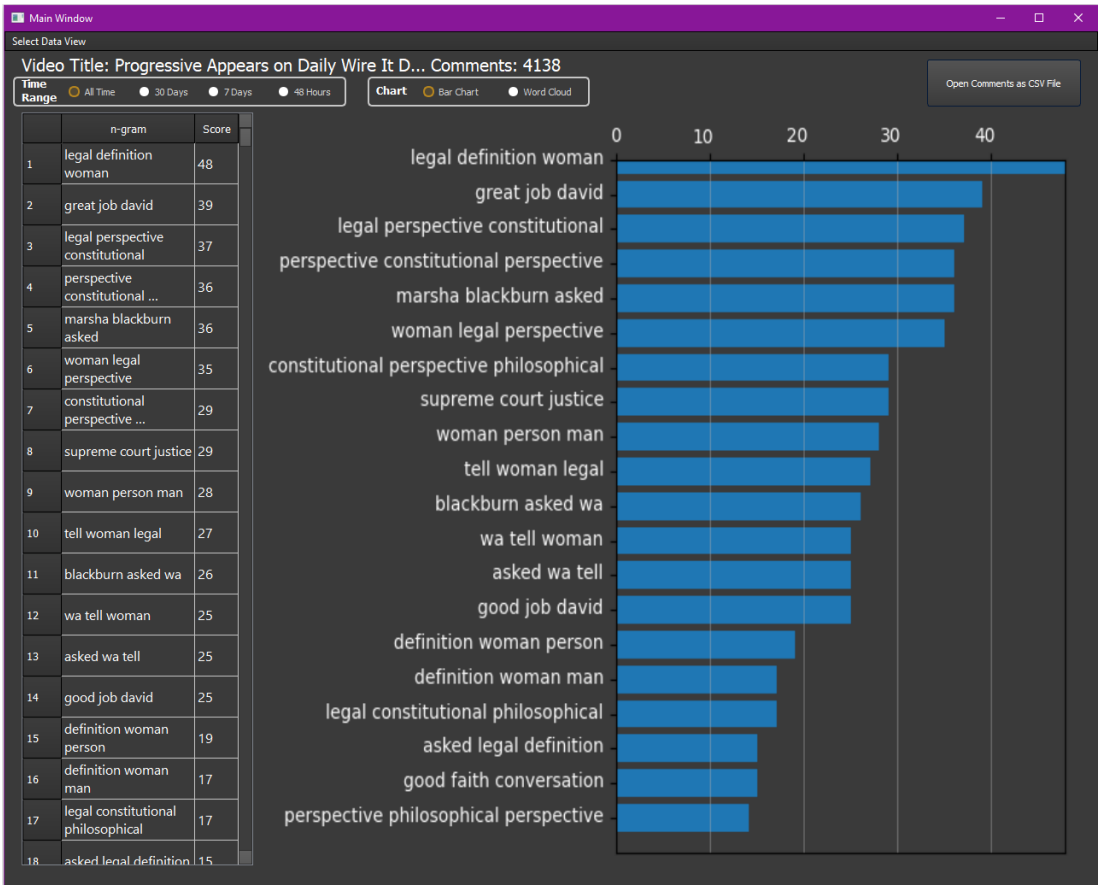

(b) Unigram frequency word cloud screen



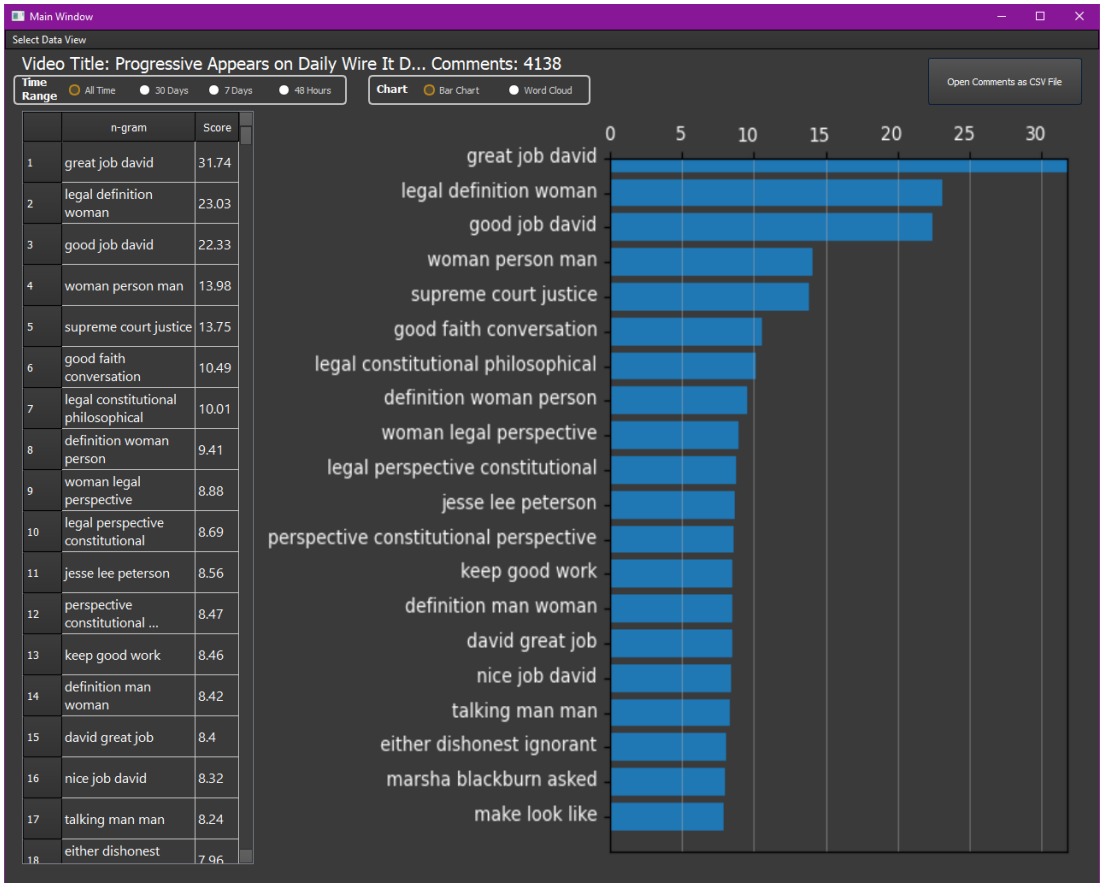

(c) Bigram frequency screen




(d) Bigram importance screen

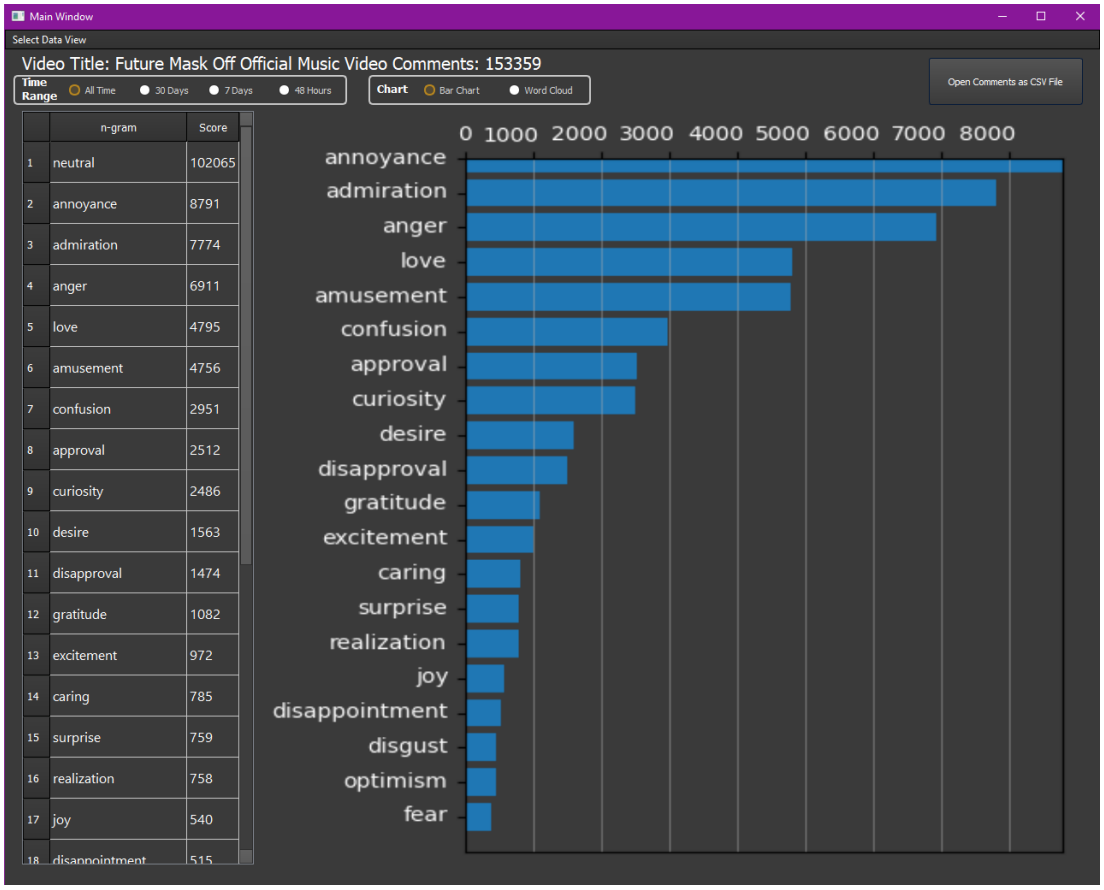



(e) Trigram frequency screen

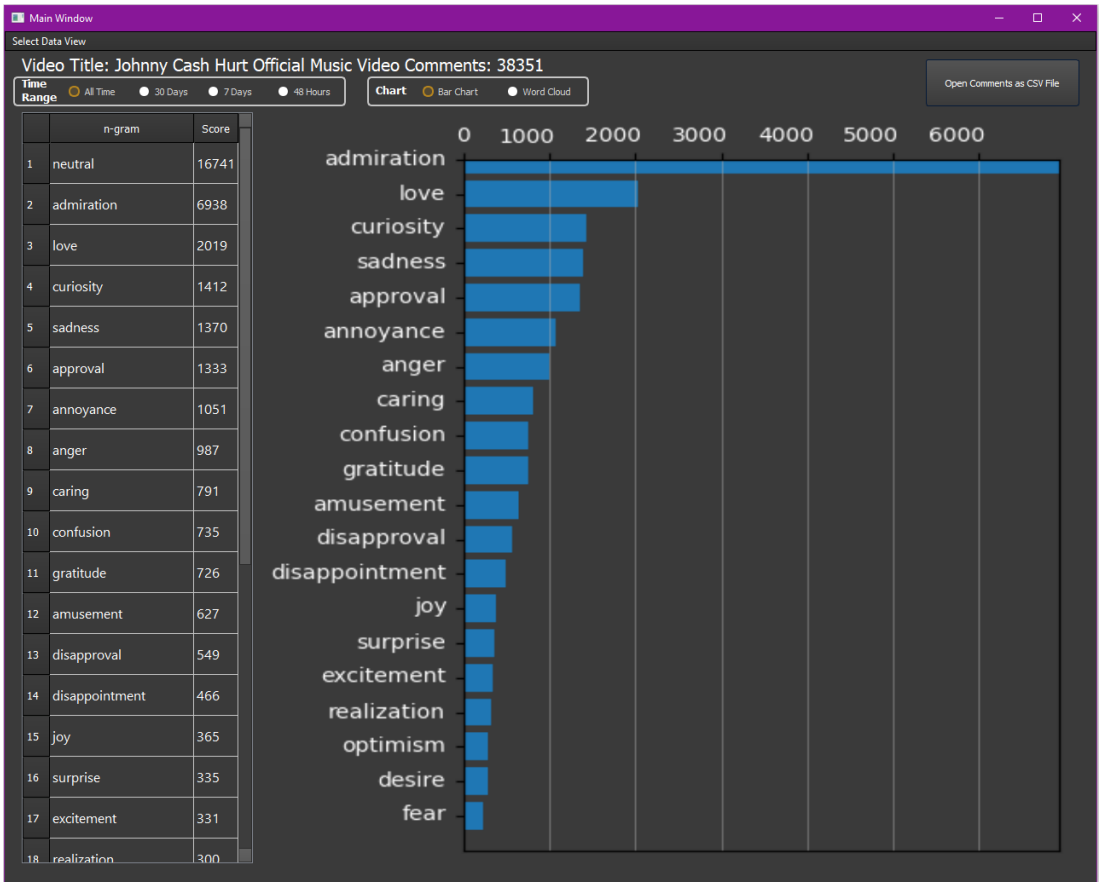



(f) Trigram importance screen

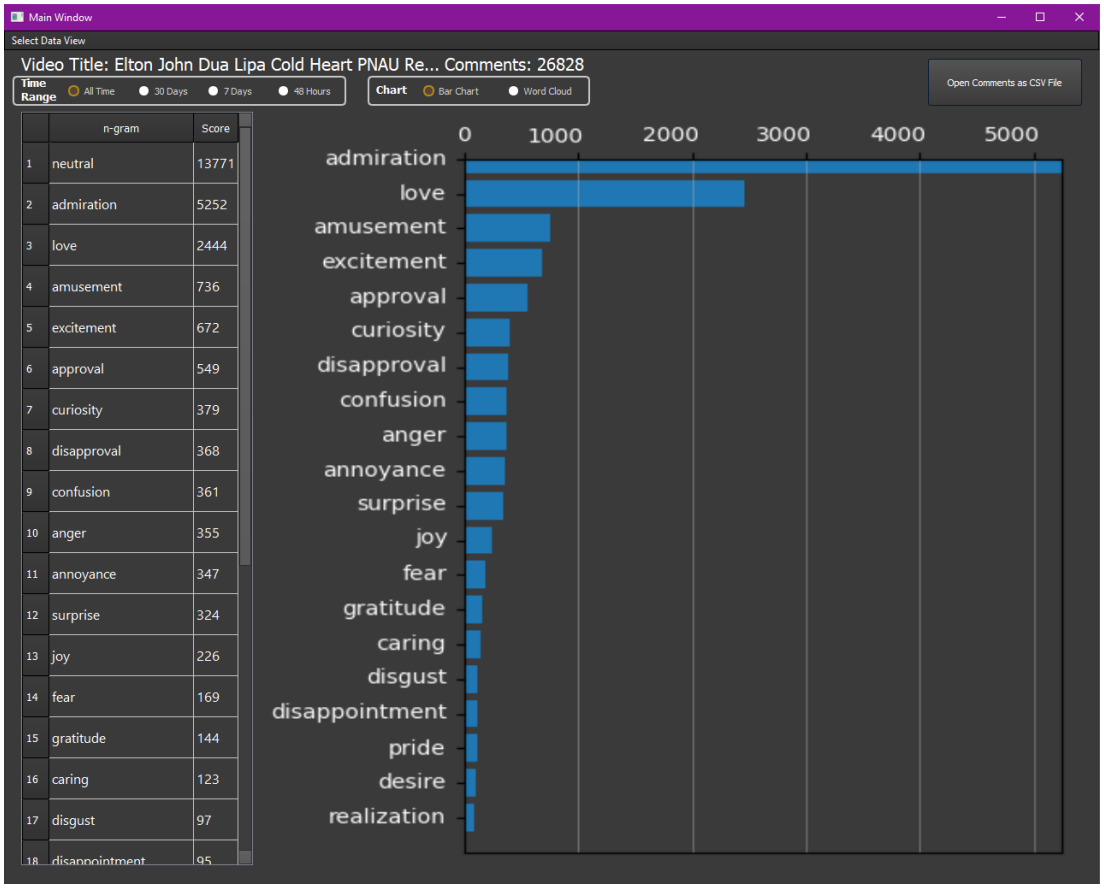
### A.1.22 Figure 5.5 Test 3, 4, 5 emotional analysis screens




(a) Test 3 emotional analysis bar chart screen

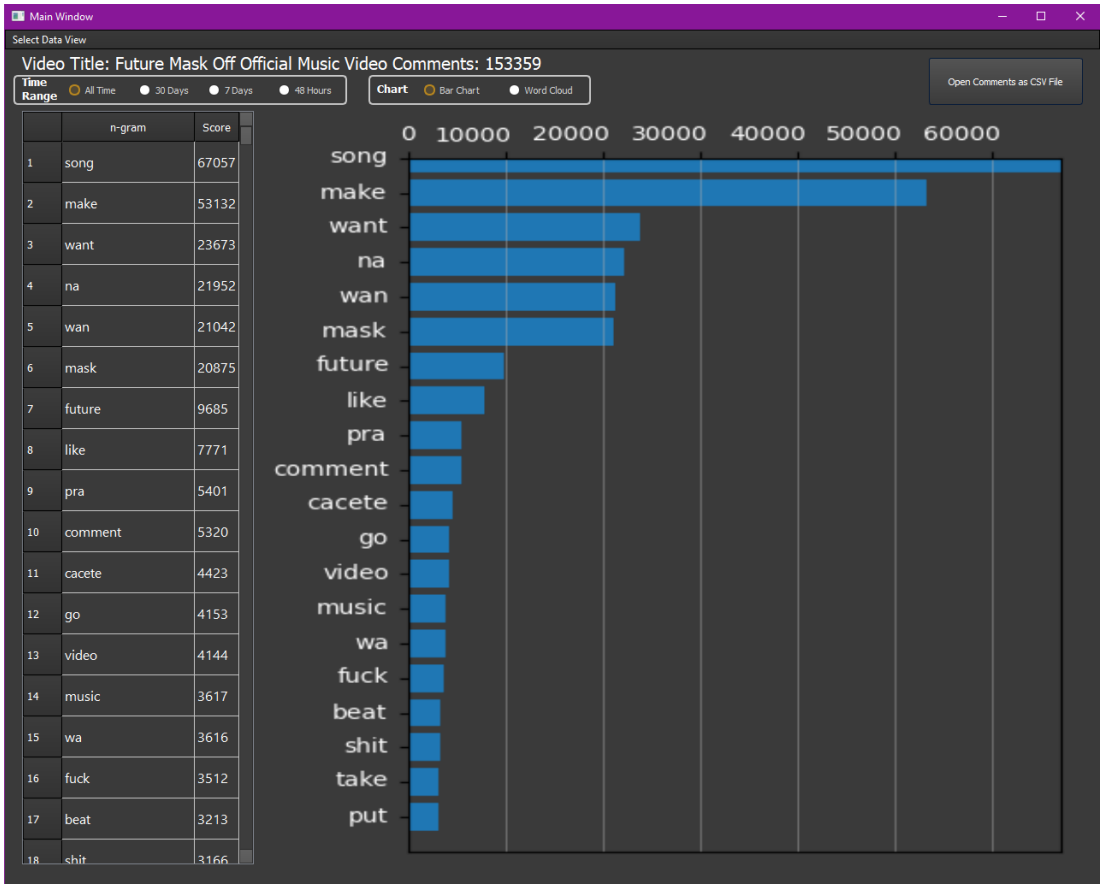



(b) Test 4 emotional analysis bar chart screen

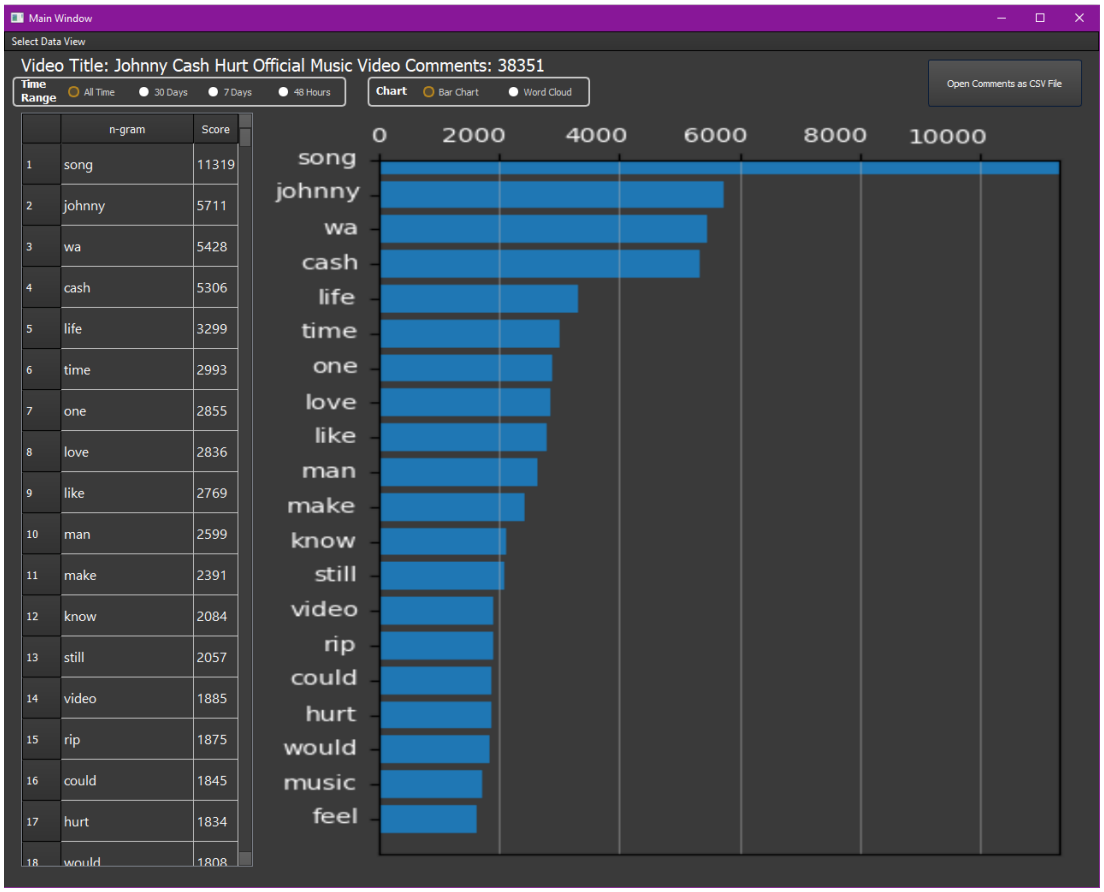



(c) Test 5 emotional analysis bar chart screen

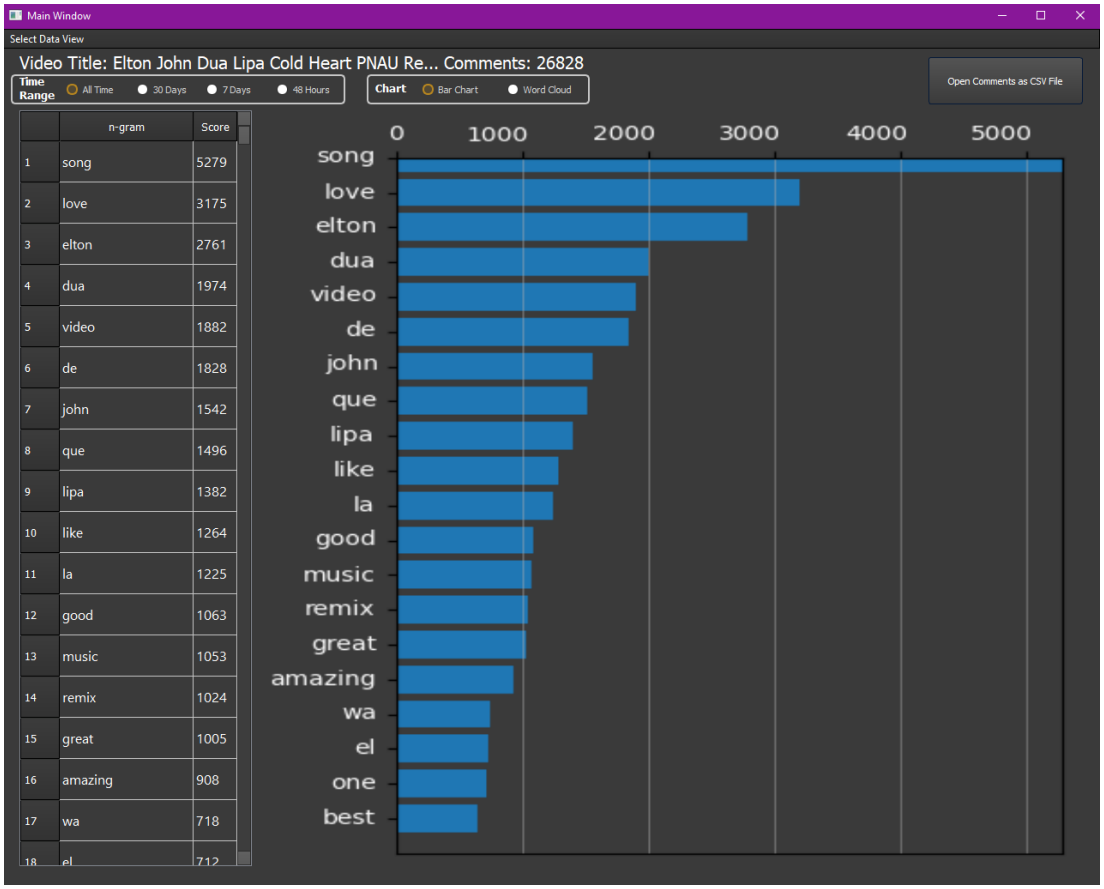
### A.1.23 Figure 5.6 Test 3, 4, 5 unigram frequency screens




(a) Test 3 unigram frequency bar chart screen

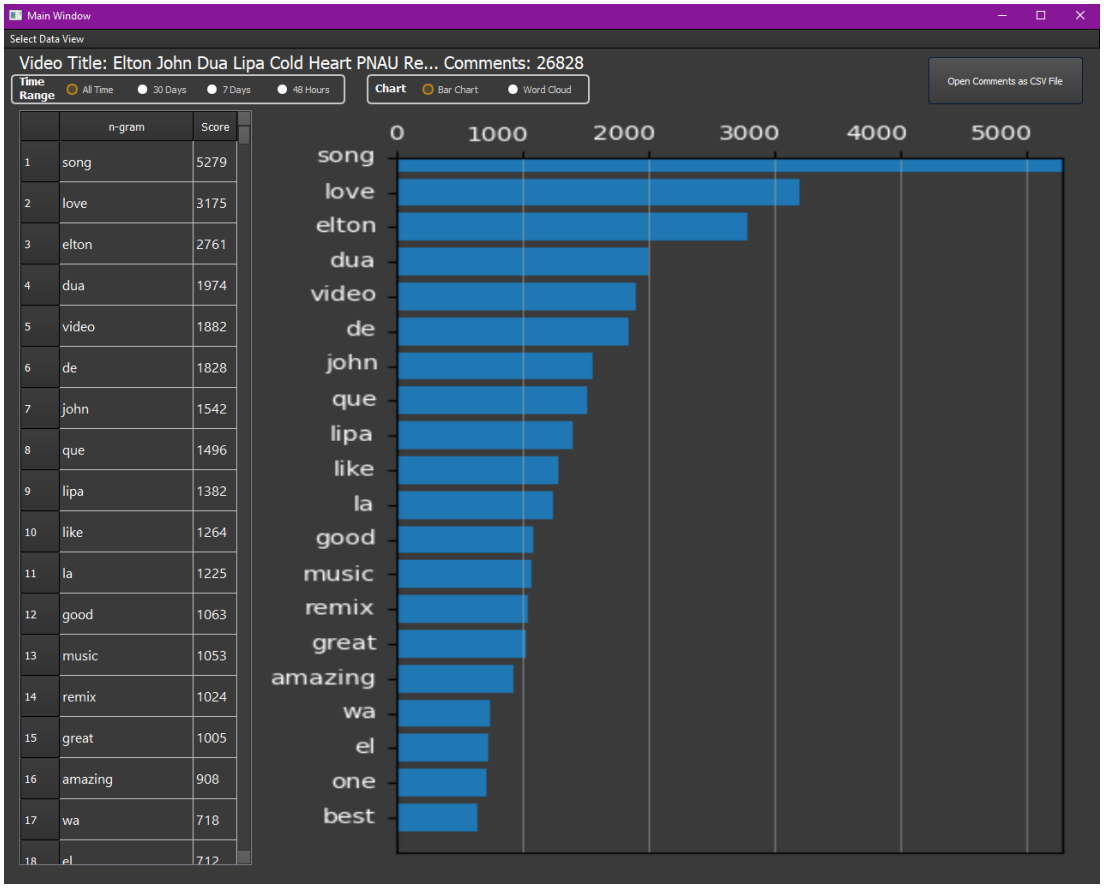



(b) Test 4 unigram frequency bar chart screen

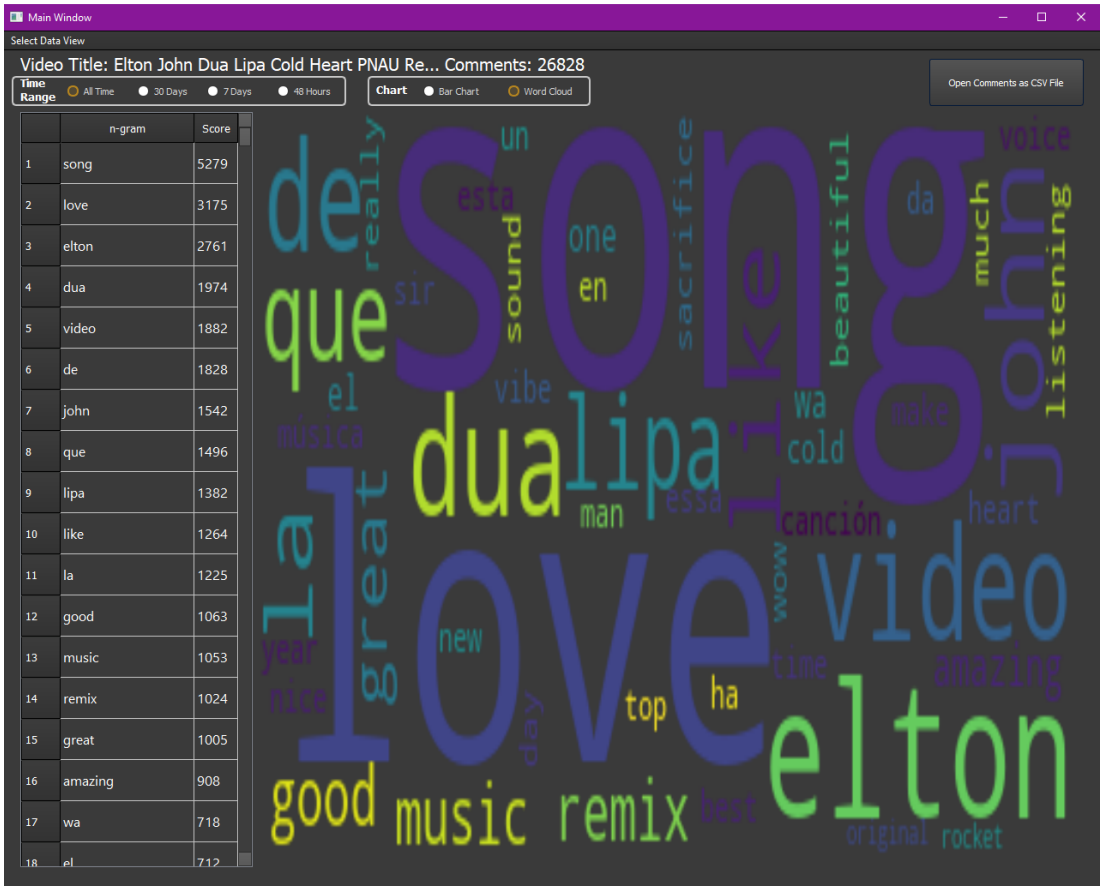



(c) Test 5 unigram frequency bar chart screen

### A.1.24 Figure 5.7 Test 5 unigram word frequency screens

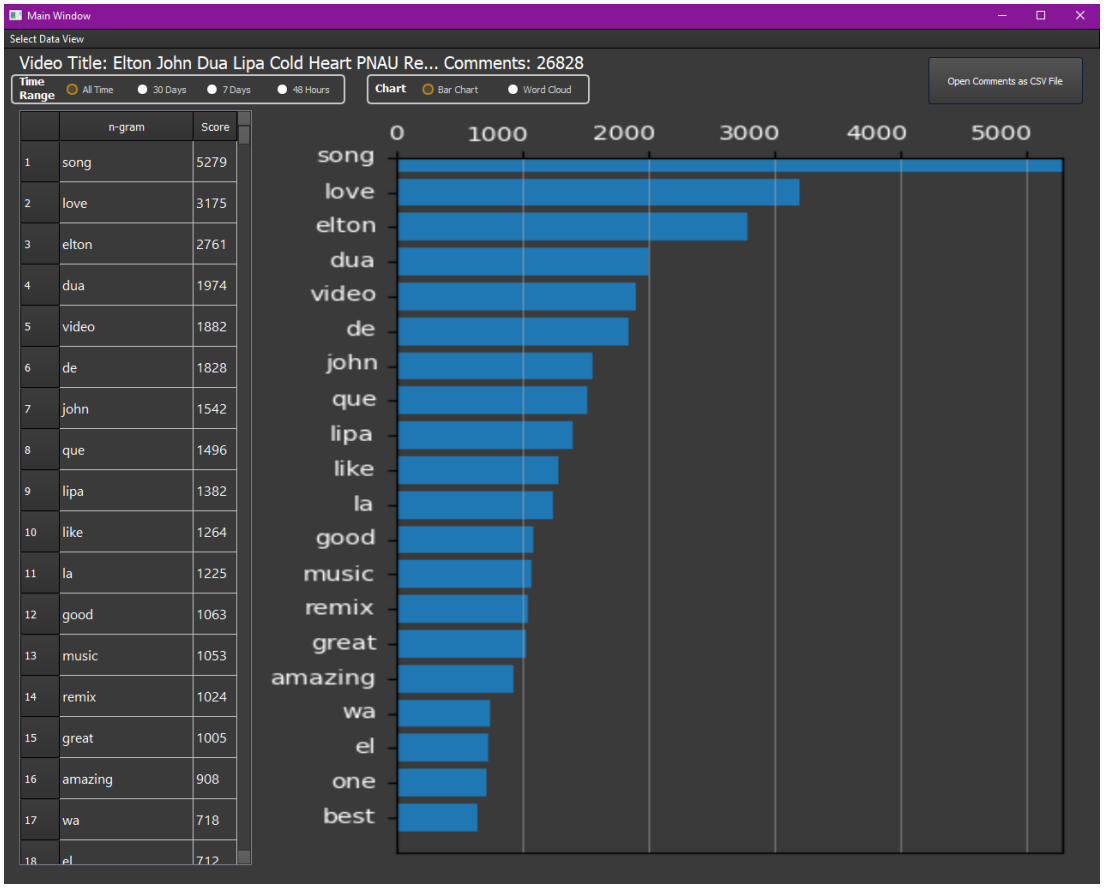



(a) Test 5 unigram frequency bar chart screen

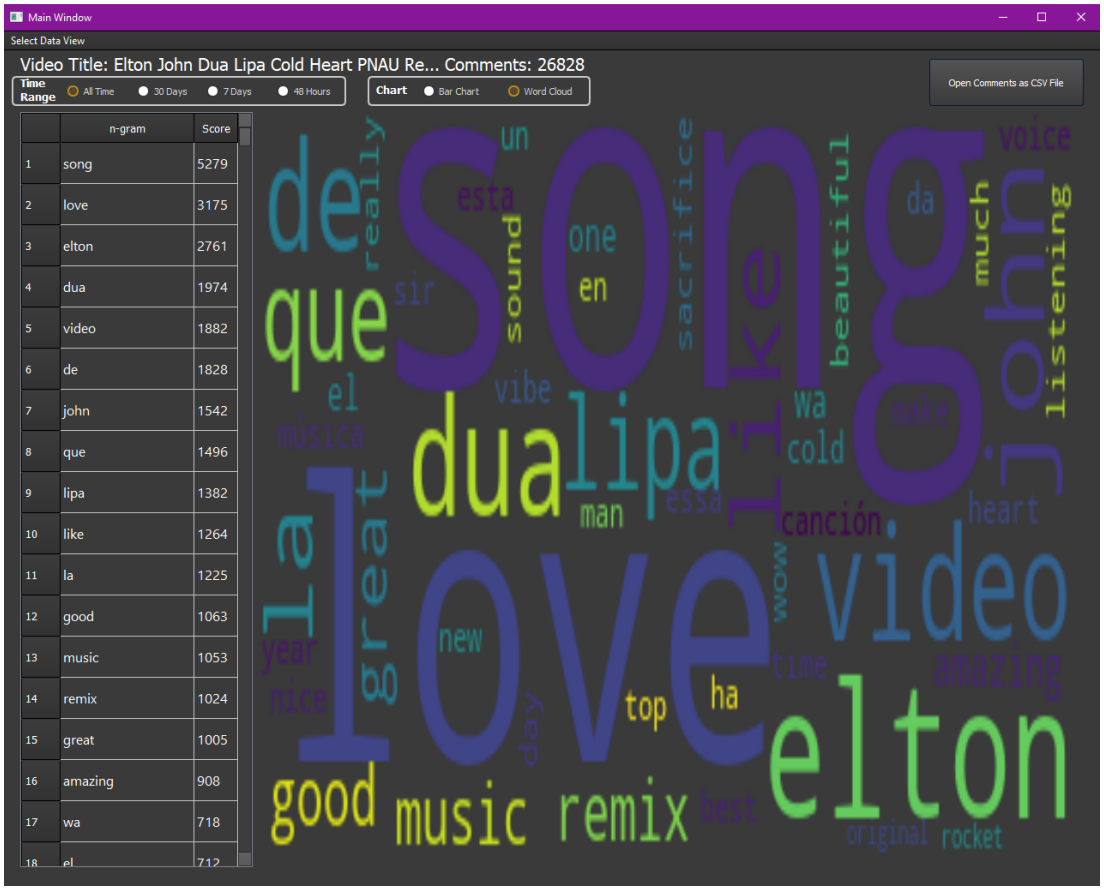



(b) Unigram frequency word cloud screen

### A.1.25 Figure 5.7 Test 5 unigram word frequency screens

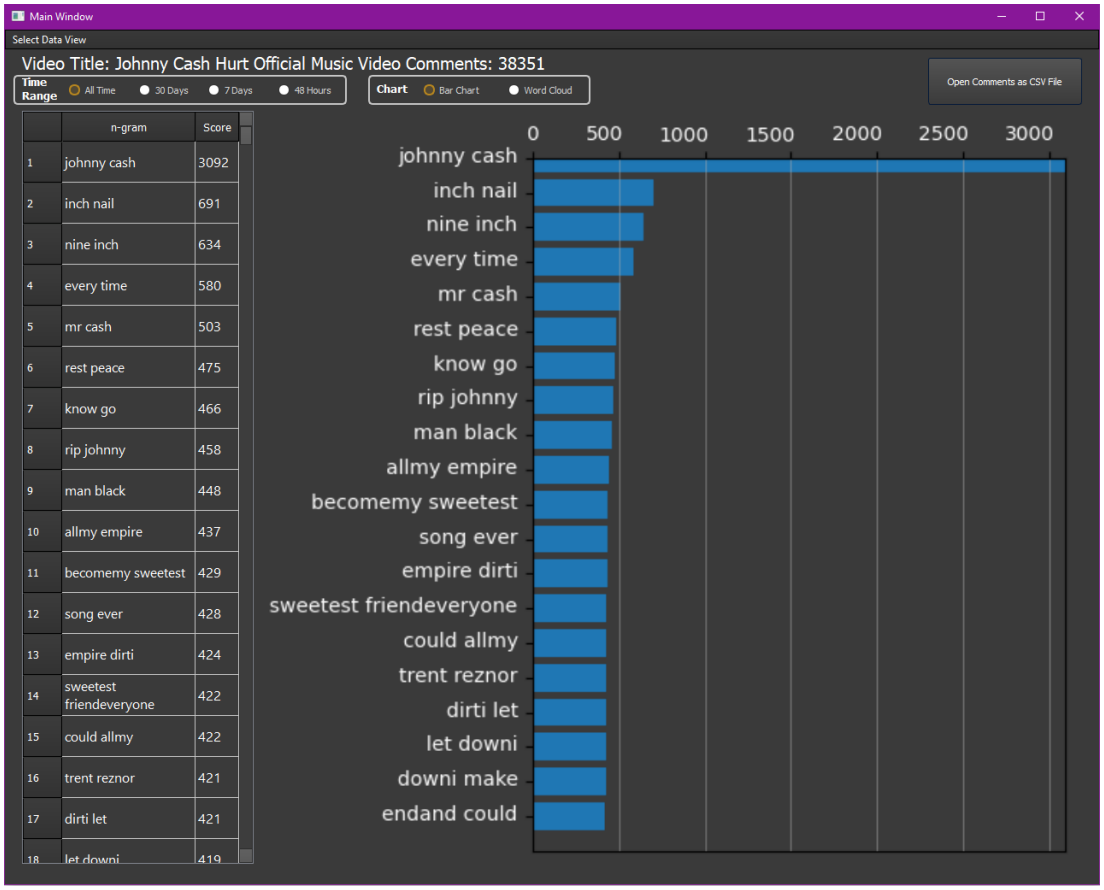



(a) Test 5 unigram frequency bar chart screen

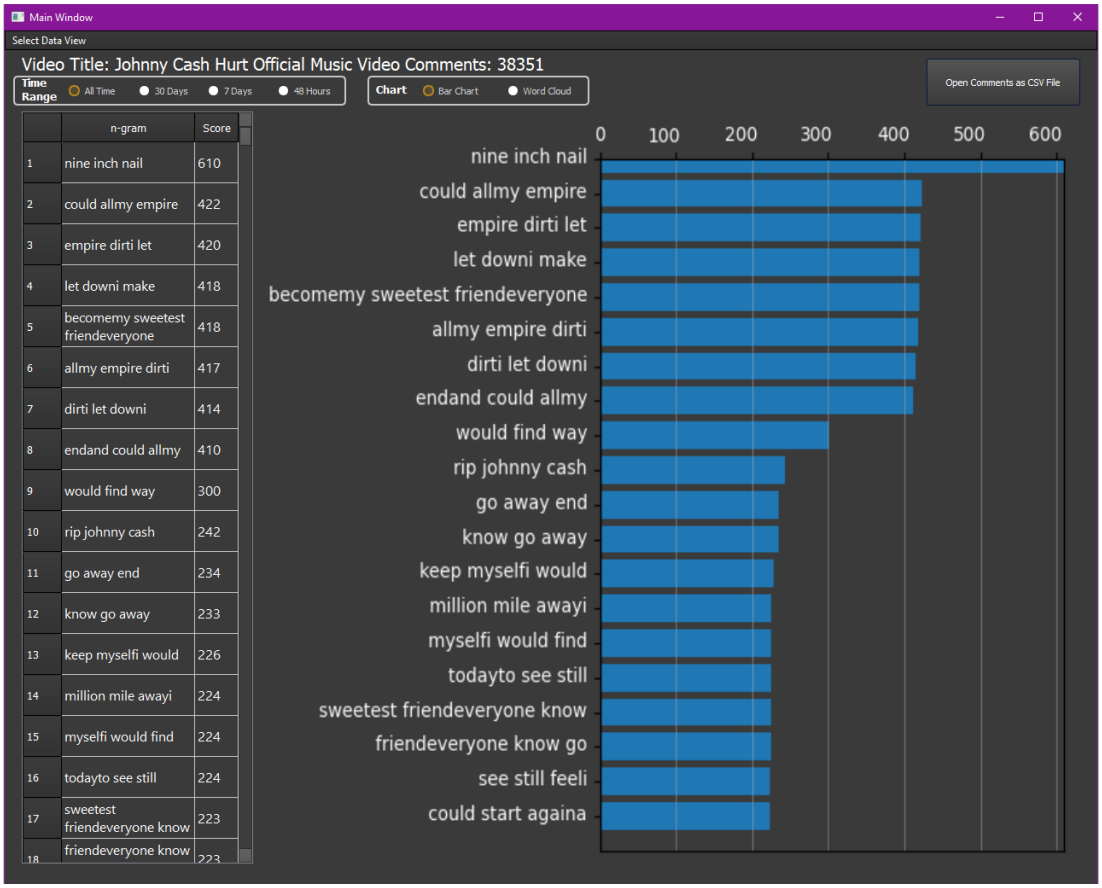



(b) Unigram frequency word cloud screen

### A.1.26 Figure 5.8 Test 4 bigram and trigram word frequency screens

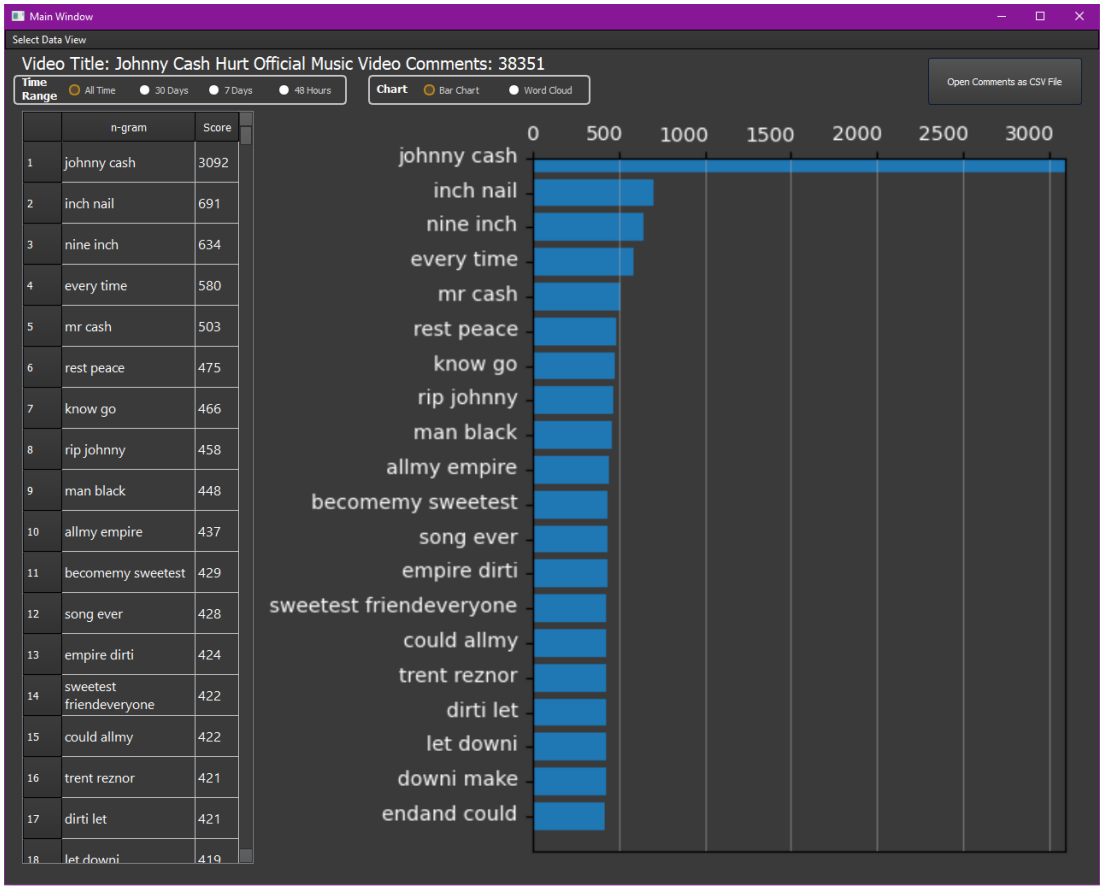



(a) Test 4 bigram frequency screen

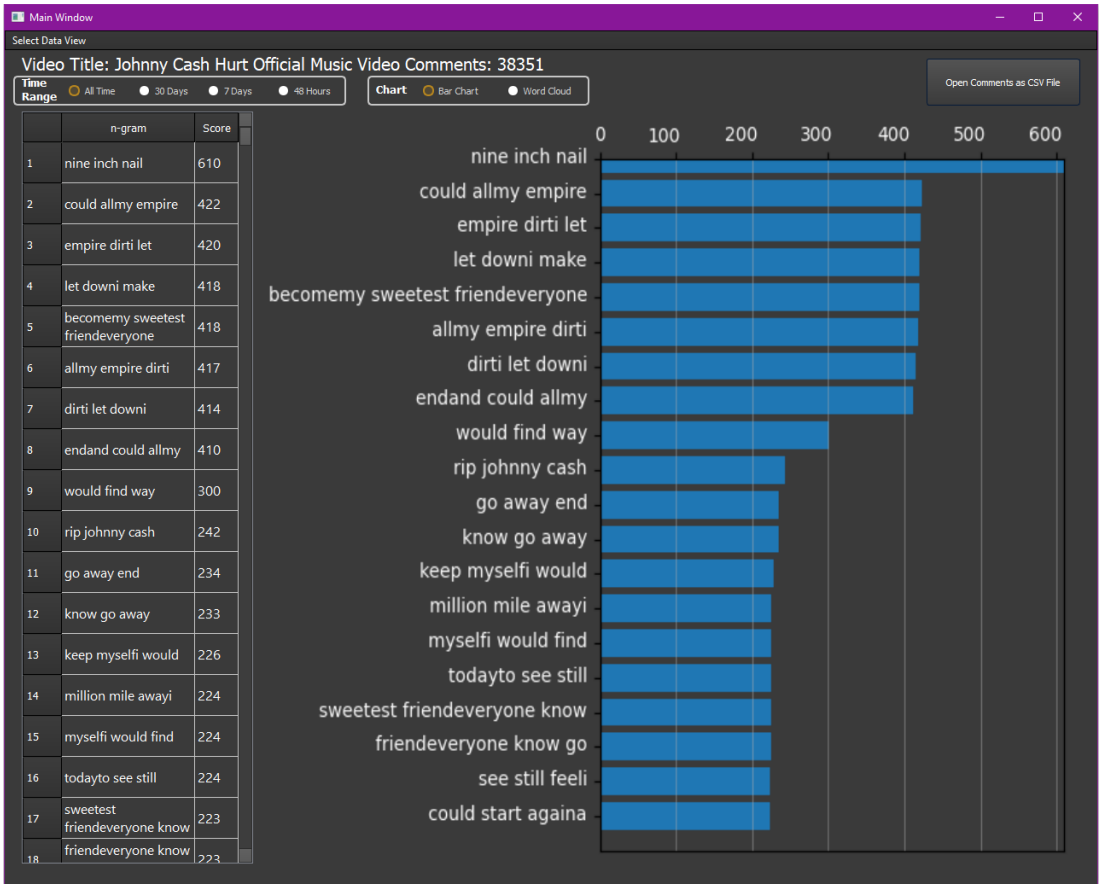



(b) Test 4 trigram frequency screen

### A.1.27 Figure 5.9 Test 4 bigram and trigram word importance screens




(a) Test 4 bigram importance screen




(b) Test 4 trigram importance screen

## A.2 Tables

### A.2.1 Table 4.1 Each of the pandas DataFrame columns, along with their respective functions and an example comment at that stage

| Column Name | Section | Function Name | Example |
|---|---|---|---|
| Comment | 4.1 | retrieve | Does his lips* mvoing mean he's LYING? 😂 https://www.dictionary.com/browse/lying |
| no_symbols | 4.2.2 | remove_symbols | Does his lips mvoing mean hes LYING? http-swww.dictionary.combrowselying |
| no_punctuation_and_symbols | 4.2.2 | remove_punctuation_and_symbols | Does his lips mvoing mean hes LYING http-swwwdictionarycombrowselying |
| no_urls | 4.2.3 | remove_urls | Does his lips mvoing mean hes LYING? |
| no_urls2 | 4.2.3 | remove_urls | Does his lips mvoing mean hes LYING |
| lower_case | 4.2.4 | lower_case | does his lips mvoing mean hes lying |
| expanded_contractions | 4.2.5 | expand_contractions | does his lips mvoing mean he is lying |
| expanded_contractions2 | 4.2.5 | expand_contractions | Does his lips mvoing mean he is LYING? |
| spelling | 4.2.6 | correct_spelling | does his lips moving mean he is lying |
| spelling2 | 4.2.6 | correct_spelling | Does his lips moving mean he is LYING? |
| tokens | 4.2.7 | tokenise | ['does', 'his', 'lips', 'moving', 'mean', 'he', 'is', 'lying'] |
| lemmas | 4.2.8 | lemmatize | ['doe', 'his', 'lip', 'moving', 'mean', 'he', 'is', 'lying'] |
| lemmas_no_stopwords | 4.2.9 | remove_stopwords | ['doe', 'lip', 'moving', 'mean', 'lying'] |

Each of the pandas DataFrame columns, along with their respective functions and an example comment at that stage

### A.2.2 Table 4.2 All files generated for each video processed

| **Name** | **Type** | **Description** |
|---|---|---|
| title | pickle | Title of the video |
| data | CSV | Raw comments with timestamps |
| cleaned_data | CSV | Cleaned comments |
| emotional_analysis | pickle | Emotional analysis |
| report | CSV | Report |
| processed_all | CSV | Processed comments |
| processed_30_days | CSV | Processed comments from last 30 days |
| processed_7_days | CSV | Processed comments from last 7 days |
| processed_48_hours | CSV | Processed comments from last 48 hours |
| vectorizer_data | pickle | Vectorizer data |
| Charts | Folder with PNGs | Folder with bar charts and word clouds |

All files generated for each video processed

### A.2.3 Table 4.3 MainWindow classes controller variables

| **Variable** | **Type** | **Purpose (assuming the appropriate screen is selected)** |
|---|---|---|
| *comments* | DataFrame | Current subset of comments being displayed to the user |
| *screen* | String | Current screen being displayed to the user |
| *chart* | String | Current chart being displayed to the user |
| *filter* | String | Current time range filter |
| *table* | List of lists | Current vectorizer data being displayed to the user |
| *sort_by* | String | Current sort metric (by sentiment, subjectivity or post date-time) |

**MainWindow** classes controller variables

### A.2.4 Table 5.1 Music videos used to test the application

| No. | Title | Creator | Views | Main Comments | URL | Sentiment Assigned | Emotion Assigned | Offence Assigned |
|---|---|---|---|---|---|---|---|---|
| 1 | Should she have been FIRED? – Debating Destiny | Bastiat | 4000 | 26 | https://www.youtube.com/watch?v=KlbDJcsNTNo | 69% | 31% | 23% |
| 2 | Progressive Appears on Daily Wire, It Doesn't Go Well | David Pakman | 340,000 | 4138 | https://www.youtube.com/watch?v=dwOSlwtyA_M | 87% | 65% | 43% |
| 3 | Mask Off | Future | 535,000,000 | 153,359 | https://www.youtube.com/watch?v=xvZqHgFz51I | 56% | 33% | 13% |
| 4 | Hurt | Johnny Cash | 153,000,000 | 38,351 | https://www.youtube.com/watch?v=8AHCfZTRGiI | 74% | 56% | 29% |
| 5 | Cold Heart (PNAU Remix) | Elton John, Dua Lipa | 241,000,000 | 26,828 | https://www.youtube.com/watch?v=qod03PVTLqk | 68% | 49% | 14% |

Music videos used to test the application

### A.2.5 Table 5.2 Examples of some of the most positive and negative comments from the second testing video

| Comment | Polarity |
|---|---|
| Masterful work Pak! | 1 |
| Love 💝it 🤣😂🤣😂 Pakman the Best 😀👍💯 | 0.9822 |
| You missed in this one. It was easy. Just challenge him to do the same thing and then take it apart. 'Woman' means different things in different contexts. Is there a hard cut off for when a girl becomes a woman? Is it legal for intercourse, legal for marriage, able to vote, menarche, 'deflowered', some feminist idea of emancipation, having a Bat Mitzvah?" And 'the' dictionary definition? The fact that there are more than one dictionary, each containing more than one entry gives the lie to that. The debate of prescriptive/descriptive only exists in the public mind. Prescriptive went out with the Victorians and Confucius. Language does not have a spec (see the use of 'preferred' in the Chicago Manual)."And there are more than one type of dictionary– popular, medical, biological, anthropological. If you mean the colloquial definition, you'd better go look up the squishiness of the word Colloquial. As well as Dialect, Ethnicity, and I don't know, maybe Wittgenstein.' (*Comment goes on for another 542 words...*) | 0.9952 |
| Her 'base' ?!? She's not up for election | -1 |
| Progressives hate women nothing more. This is the ultimate form of misogyny. Allowing men to cosplay as women. Making mockery of women. Any woman that can just go along with this foolery that anything can be a woman dispite the biological difference you are part of the problem. Progressive policies will eventually undermine the same feminist that so called fought for these rights. Facts don't care about feelings sorry not sorry. A man can't be a woman and a woman can't be a man. You have gender disphoria you need help. Mutilating your bodies trying to indoctrinate children in your weird fantasies to get approval and validation of your existence. It's disgusting and down right evil. The same women who want to protect women's sports women's spaces are the same ones allowing it to get destroyed. You refuse to acknowledge facts over emotions. Hard data over how you feel one day or the next. | -0.9786 |
| Just for a change I'd like to hear one of these conservative Talking Heads talk about something other than culture issues. You know, policies. Financial policies, foreign policies, just any policies that they stand for other than these stupid culture War issues. But that's all they've got. They have no policies, and frankly I'm sick of hearing about the culture War. I just don't care about it. It's all really quite meaningless. | -0.9734 |

Examples of some of the most positive and negative comments from the second testing video

### A.2.6 Table 5.3 Music videos used to test the application

| No. | Title | Creator | Views | Main Comments | URL | Sentiment Assigned | Emotion Assigned | Offence Assigned |
|---|---|---|---|---|---|---|---|---|
| 1 | Should she have been FIRED? – Debating Destiny | Bastiat | 4000 | 26 | https://www.youtube.com/watch?v=KlbDJcsNTNo | 69% | 31% | 23% |
| 2 | Progressive Appears on Daily Wire, It Doesn't Go Well | David Pakman | 340,000 | 4138 | https://www.youtube.com/watch?v=dwOSlwtyA_M | 87% | 65% | 43% |
| 3 | Mask Off | Future | 535,000,000 | 153,359 | https://www.youtube.com/watch?v=xvZqHgFz51I | 56% | 33% | 13% |
| 4 | Hurt | Johnny Cash | 153,000,000 | 38,351 | https://www.youtube.com/watch?v=8AHCfZTRGiI | 74% | 56% | 29% |
| 5 | Cold Heart (PNAU Remix) | Elton John, Dua Lipa | 241,000,000 | 26,828 | https://www.youtube.com/watch?v=qod03PVTLqk | 68% | 49% | 14% |

Music videos used to test the application

# Appendix B

# User Guide

## B.1 Instructions

You must provide an adequate user guide for your software. The guide should provide easily understood instructions on how to use your software. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all of the features of your package. Technical details of how the package works are rarely required. Keep the guide concise and simple. The extensive use of diagrams, illustrating the package in action, can often be particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better included in an appendix to the main report.

# YouTube Comments Analyser

USER GUIDE

Daniel

# Contents

# Introduction

The YouTube Comments Analyser is an application that can be used to analyse comments posted on any YouTube video. Various natural language processing and sentiment analysis techniques are performed by the application, where a variety of statistics and charts about a video's comments are displayed to the user once processing has been completed.

The application has been developed in Python and converted into an executable, meaning that no external libraries are needed to use the application.

# Installing

## System Requirements

Windows 10 is required.

### Minimum Recommendations

| | |
|---|---|
| OS | Windows 10 (64-bit) |
| RAM | 4GB |
| Storage | 2GB Available Space (to store application) |

For the open CSV file button to function correctly, Microsoft Excel needs to be installed.

## How to Run

The application can be run by opening the main directory and opening the "YouTube Comments Analyser" file.

It is recommended to use the application without changing the window's resolution or maximizing the window. If, however, some of the application is cut off due to a small monitor, press "Windows key + up arrow" to full screen the application.

**The application may take up to 10 minutes to launch if the machine has a poor-performing CPU.**

**If the "desktop.ini" file shows up in the drop box, do not try and select this file, as it will crash the application.**

## File Structure Of The Application

All of the files needed and created during the execution of the program are stored in the main "*Python*" folder.

In the "*Python*" folder, there are three folders: "*StyleSheets*", "*Corpora*", and "*Videos*".

The "*StyleSheets*" folder contains all of the stylesheets used to decorate the GUI.

The "*Corpora*" folder stores all of the corpora used for the classification of comments into categories. The "Cyber Troll" and "Emotions" folders are both stored here, both of which represent their respective corpora. Inside each of these folders are the raw corpus, the cleaned version of the corpus, and data that represents classifiers that have been trained using the cleaned corpora.

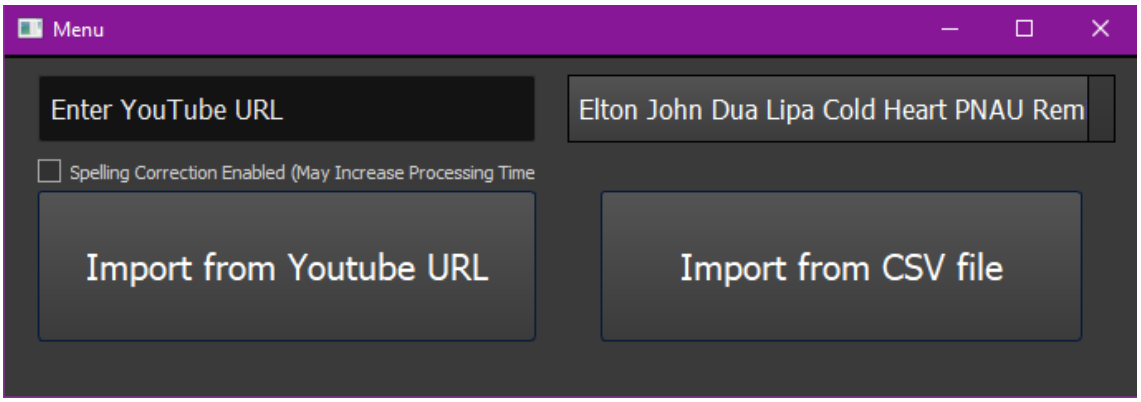
The "*Videos*" folder stores all of the videos processed by the application. The application comes with some datasets pre-downloaded – these datasets are the ones that have been used to test the application, described in the main paper.

Each video's folder is titled that video's name on YouTube. Each of these files stores the raw and processed data of each video – each file is described in more detail in the table below.

| **Name** | **Type** | **Description** |
|---|---|---|
| title | pickle | Title of the video |
| data | CSV | Raw comments with timestamps |
| cleaned_data | CSV | Cleaned comments |
| emotional_analysis | pickle | Emotional analysis |
| report | CSV | Report |
| processed_all | CSV | Processed comments |
| processed_30_days | CSV | Processed comments from the last 30 days |
| processed_7_days | CSV | Processed comments from the last 7 days |
| processed_48_hours | CSV | Processed comments from the last 24 hours |
| vectorizer_data | pickle | Vectorizer data |
| Charts | Folder with PNGs | Folder with bar charts and word clouds |

# GUI

## Menu

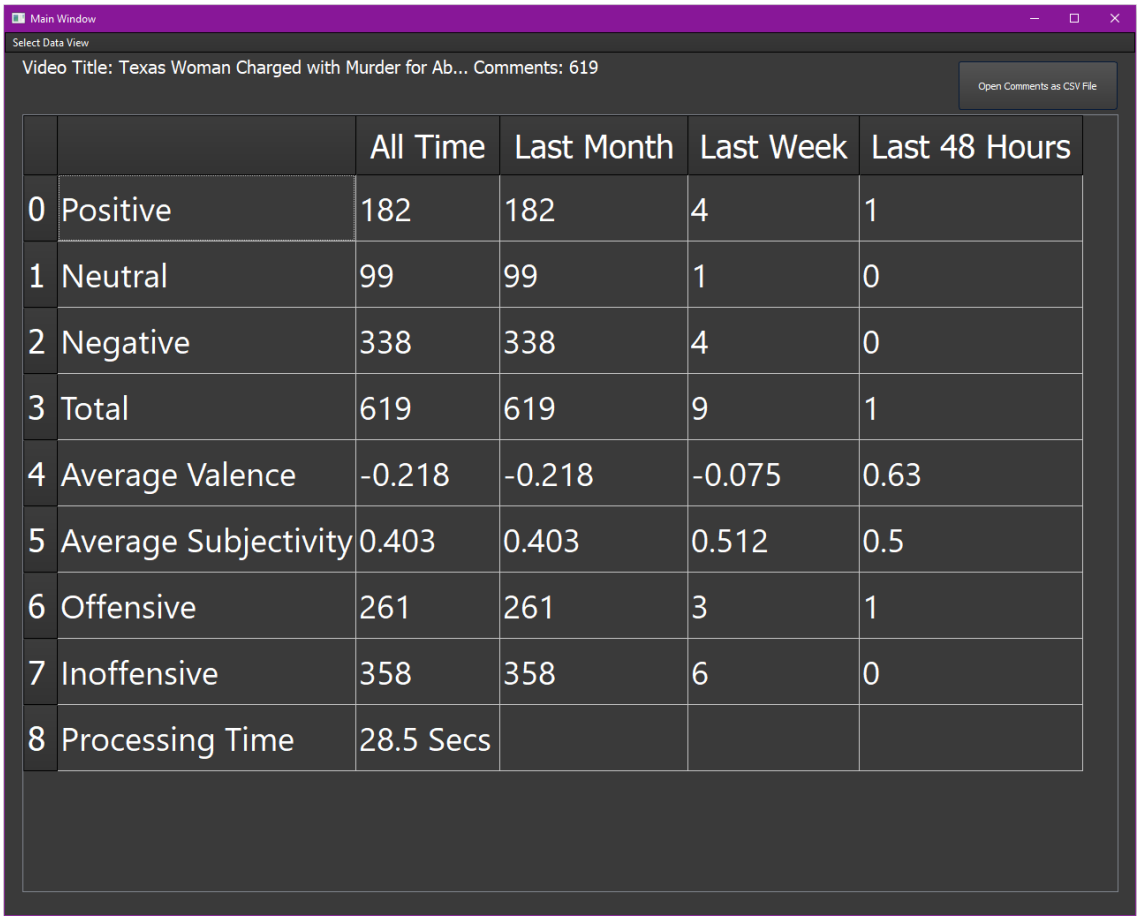



The menu is the first window displayed to the user once the program is run. The menu gives the user two options: either type in a URL and import the comments from the respective video or select a CSV file and import the comments from there instead. The menu comprises two buttons, a text box for entering a URL, a drop-down list for selecting the name of the CSV file/video, and a check box that allows the user to turn the spell correction feature on or off.

Checking the spell correction box will make the application attempt to rectify any spelling errors found within the dataset.

## Main Window

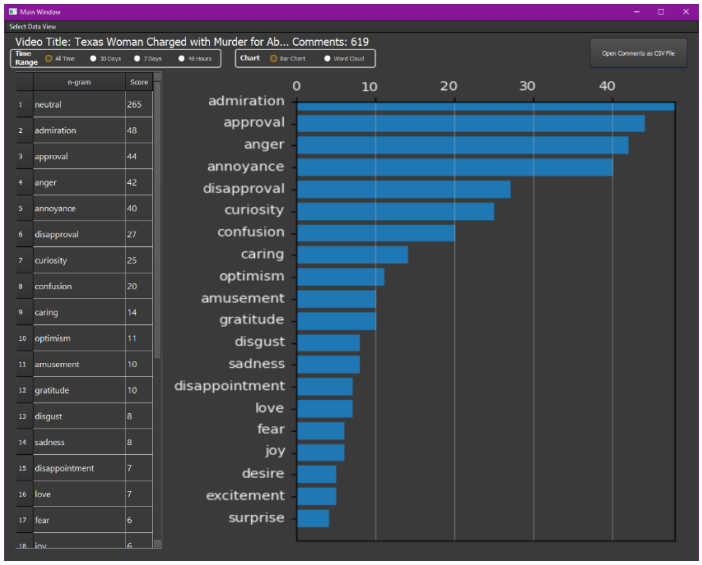
### Report

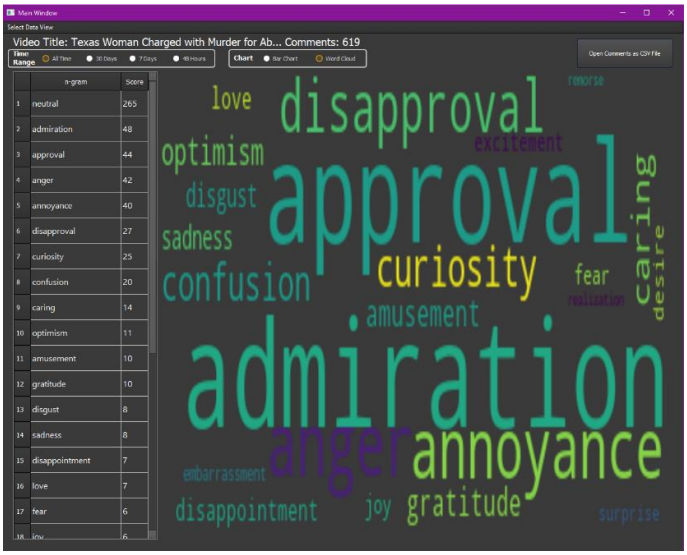



| | | All Time | Last Month | Last Week | Last 48 Hours |
|---|---|---|---|---|---|
| 0 | Positive | 182 | 182 | 4 | 1 |
| 1 | Neutral | 99 | 99 | 1 | 0 |
| 2 | Negative | 338 | 338 | 4 | 0 |
| 3 | Total | 619 | 619 | 9 | 1 |
| 4 | Average Valence | -0.218 | -0.218 | -0.075 | 0.63 |
| 5 | Average Subjectivity | 0.403 | 0.403 | 0.512 | 0.5 |
| 6 | Offensive | 261 | 261 | 3 | 1 |
| 7 | Inoffensive | 358 | 358 | 6 | 0 |
| 8 | Processing Time | 28.5 Secs | | | |

Once a video's comments have either been processed or loaded from a CSV file, the report screen is displayed to the user. This screen gives the user a general overview of the sentiment and subjectivity of the video's comments. This comprises the total number of positive, neutral and negative comments in each of the time ranges (all time, last 30 days, last 7 days, last 48 hours), as well as the total number of comments, and the average valence and average subjectivity of the comments in each of the time ranges. The report also shows how many comments were classified as offensive compared to inoffensive.

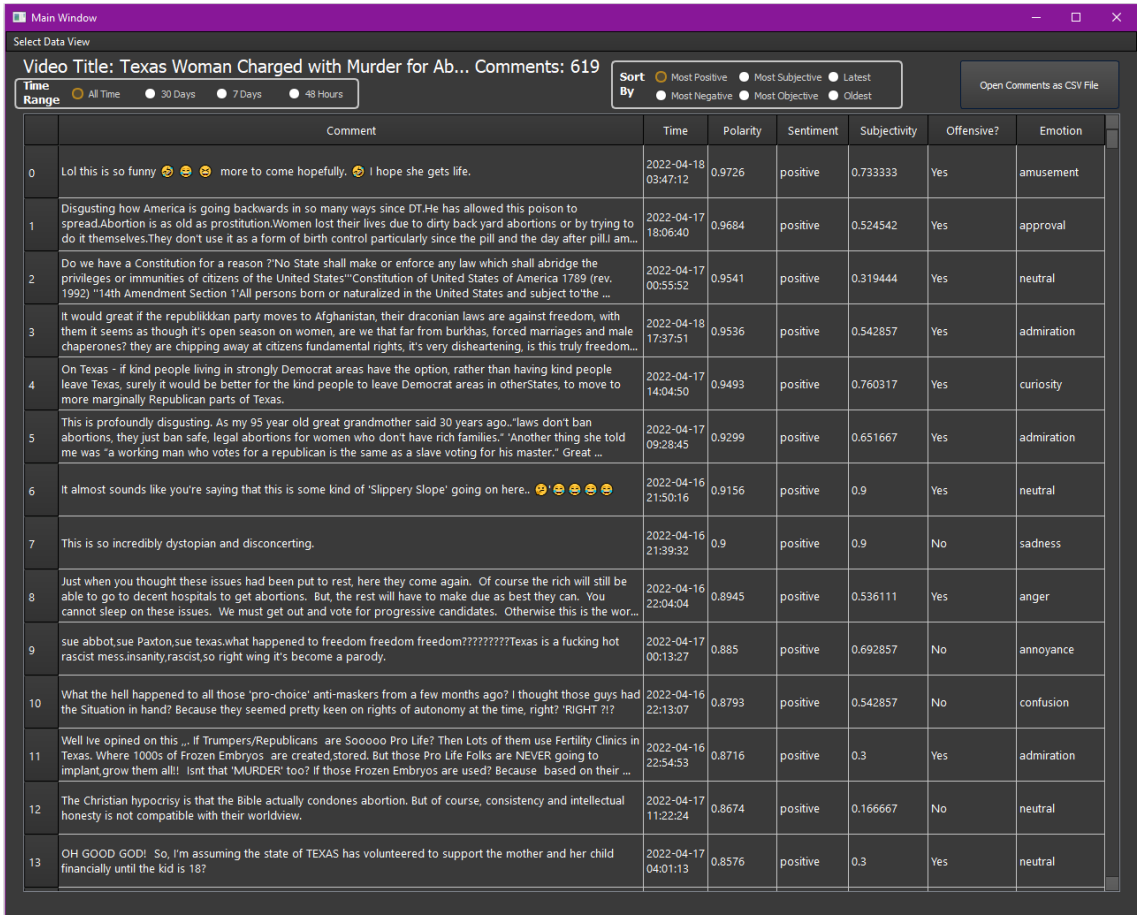
## Emotional Analysis




The emotional analysis screen shows the user a breakdown of the emotional classification performed on the dataset with the help of the GoEmotions corpus. 28 emotion categories have been used, where each comment has been assigned to a single category. The frequency of each category across the dataset has been calculated and displayed to the user on this screen. This data is also represented as a bar chart and a word cloud. The data can also be filtered for specific time ranges using the buttons at the top of the screen.

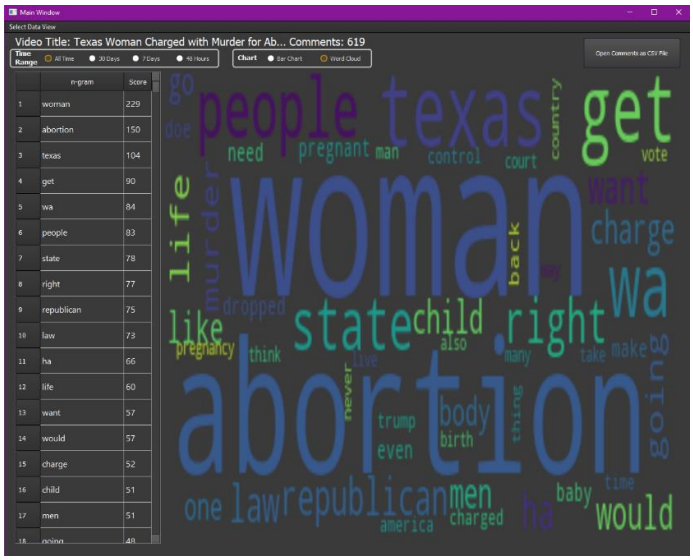
## List of Comments

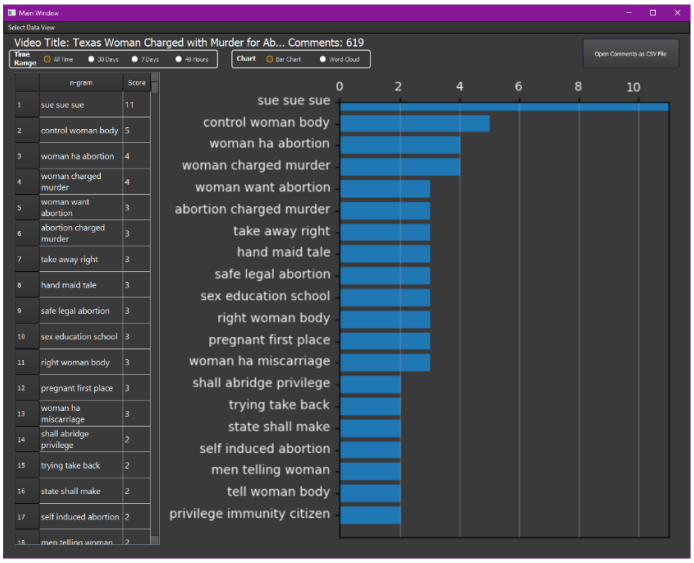



| | Comment | Time | Polarity | Sentiment | Subjectivity | Offensive? | Emotion |
|---|---|---|---|---|---|---|---|
| 0 | Lol this is so funny 🤣 😂 😭 more to come hopefully. 🤣 I hope she gets life. | 2022-04-18 03:47:12 | 0.9726 | positive | 0.733333 | Yes | amusement |
| 1 | Disgusting how America is going backwards in so many ways since DT.He has allowed this poison to spread.Abortion is as old as prostitution.Women lost their lives due to dirty back yard abortions or by trying to do it themselves.They don't use it as a form of birth control particularly since the pill and the day after pill.I am... | 2022-04-17 18:06:40 | 0.9684 | positive | 0.524542 | Yes | approval |
| 2 | Do we have a Constitution for a reason ?'No State shall make or enforce any law which shall abridge the privileges or immunities of citizens of the United States'"Constitution of United States of America 1789 (rev. 1992) "14th Amendment Section 1'All persons born or naturalized in the United States and subject to'the ... | 2022-04-17 00:55:52 | 0.9541 | positive | 0.319444 | Yes | neutral |
| 3 | It would great if the republikkkan party moves to Afghanistan, their draconian laws are against freedom, with them it seems as though it's open season on women, are we that far from burkhas, forced marriages and male chaperones? they are chipping away at citizens fundamental rights, it's very disheartening, is this truly freedom... | 2022-04-18 17:37:51 | 0.9536 | positive | 0.542857 | Yes | admiration |
| 4 | On Texas - if kind people living in strongly Democrat areas have the option, rather than having kind people leave Texas, surely it would be better for the kind people to leave Democrat areas in otherStates, to move to more marginally Republican parts of Texas. | 2022-04-17 14:04:50 | 0.9493 | positive | 0.760317 | Yes | curiosity |
| 5 | This is profoundly disgusting. As my 95 year old great grandmother said 30 years ago.."laws don't ban abortions, they just ban safe, legal abortions for women who don't have rich families." 'Another thing she told me was "a working man who votes for a republican is the same as a slave voting for his master." Great ... | 2022-04-17 09:28:45 | 0.9299 | positive | 0.651667 | Yes | admiration |
| 6 | It almost sounds like you're saying that this is some kind of 'Slippery Slope' going on here.. 🤔😂😂😂😂 | 2022-04-16 21:50:16 | 0.9156 | positive | 0.9 | Yes | neutral |
| 7 | This is so incredibly dystopian and disconcerting. | 2022-04-16 21:39:32 | 0.9 | positive | 0.9 | No | sadness |
| 8 | Just when you thought these issues had been put to rest, here they come again. Of course the rich will still be able to go to decent hospitals to get abortions. But, the rest will have to make due as best they can. You cannot sleep on these issues. We must get out and vote for progressive candidates. Otherwise this is the wor... | 2022-04-16 22:04:04 | 0.8945 | positive | 0.536111 | Yes | anger |
| 9 | sue abbot,sue Paxton,sue texas.what happened to freedom freedom freedom?????????Texas is a fucking hot rascist mess.insanity,rascist,so right wing it's become a parody. | 2022-04-17 00:13:27 | 0.885 | positive | 0.692857 | No | annoyance |
| 10 | What the hell happened to all those 'pro-choice' anti-maskers from a few months ago? I thought those guys had the Situation in hand? Because they seemed pretty keen on rights of autonomy at the time, right? 'RIGHT ?!? | 2022-04-16 22:13:07 | 0.8793 | positive | 0.542857 | No | confusion |
| 11 | Well Ive opined on this ,,. If Trumpers/Republicans are Sooooo Pro Life? Then Lots of them use Fertility Clinics in Texas. Where 1000s of Frozen Embryos are created,stored. But those Pro Life Folks are NEVER going to implant,grow them all!! Isnt that 'MURDER' too? If those Frozen Embryos are used? Because based on their ... | 2022-04-16 22:54:53 | 0.8716 | positive | 0.3 | Yes | admiration |
| 12 | The Christian hypocrisy is that the Bible actually condones abortion. But of course, consistency and intellectual honesty is not compatible with their worldview. | 2022-04-17 11:22:24 | 0.8674 | positive | 0.166667 | No | neutral |
| 13 | OH GOOD GOD! So, I'm assuming the state of TEXAS has volunteered to support the mother and her child financially until the kid is 18? | 2022-04-17 04:01:13 | 0.8576 | positive | 0.3 | Yes | neutral |

The list of comments screen allows the user to view the comments of the selected video all in one scrollable table. The user can sort the table by the most positive comments, the most negative comments, the most subjective comments, the least subjective comments, the most recently posted comments, and the oldest comments. The data can also be filtered for specific time ranges using the buttons at the top of the screen.

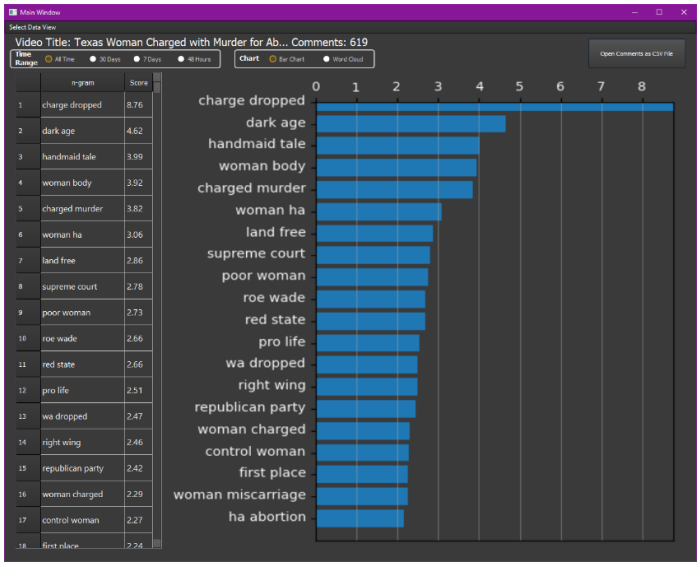
## Frequency Screens

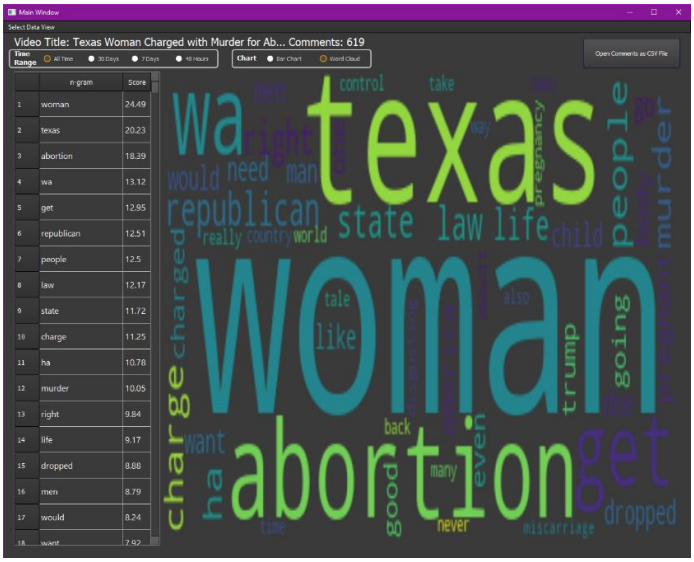



There are three frequency screens used in this application – one for single words (unigrams), one for two-word pairs (bigrams), and one for three-word pairs (trigrams). These screens display the frequency of each n-gram in the dataset in a table on the left side of the screen, with the most frequent n-grams at the top of the table, as well as either a bar chart or a word cloud on the right of the screen. To switch between charts, select either of the chart buttons. The data can also be filtered for specific time ranges using the buttons at the top of the screen.

## Importance Screens




Similarly to the frequency screens, three importance screens for unigrams, bigrams and trigrams have been developed. Here the importance of each n-gram has been calculated and assigned a score, with the most important n-grams displayed at the top of the table. Switching between charts and filtering between time ranges works in the same way.

## Warnings/Troubleshooting

The application may take up to 10 minutes to launch if the machine has a poor-performing CPU.

If the “desktop.ini” file shows up in the drop box, do not try and select this file, as it will crash the application.

It is recommended to use the application without changing the window’s resolution or maximizing the window. If, however, some of the application is cut off due to a small monitor, press “Windows key + up arrow” to full screen the application.

Processing time for videos with an extremely large number of comments can be substantial. As an example, a video with over 150,000 comments was used to test the application, which had a processing time of just over 40 minutes.

A limit of 200,000 comments downloaded per day is enforced by the Google Cloud Services for this application. This limit resets each day at 00:00 Pacific Time. If the limit is reached, the application will display an “Invalid URL” error message even if the URL is valid. Change the API key in the “API Key.txt” file in the main directory. To obtain a new API Key, follow these steps:

1. Create a Google account.
2. Visit “https://console.cloud.google.com”.
3. Create a new project by pressing the drop-down list of projects, and pressing ”New Project”.
4. Enter the term ”YouTube Data API v3” into the search bar
5. Press ”YouTube Data API v3”.
6. Press ”Enable”.
7. Press ”Credentials”.
8. Press “Create Credentials” was pressed.
9. Press “API key”.
10. Press “Show key”.
11. Copy the key into the “API Key.txt” file in the main directory.

# Appendix C

# Source Code

**Originality Avowal**

I verify that I am the sole author of the source code files, except where explicitly stated to the contrary.

Daniel Van Cuylenburg

April 29, 2022

## C.1 Python Files

### C.1.1 main.py

```python
# main.py: Main classes used in YouTube Comments Analyser application.
#
# Written by Daniel Van Cuylenburg (k19012373)
#

# Imports.
# GUI Classes.
from gui import *
# Dictionary of (unexpanded contraction) -> (expanded contraction) pairs.
from contractions import contractions

# The Python Standard Library.
from concurrent.futures import process
from sys import argv
from time import time
from datetime import datetime
from os import getcwd, mkdir
from os.path import exists, join
from re import sub
from urllib import request, parse
from json import loads
from pickle import dump, load
from ast import literal_eval
from collections import Counter
# cgitb used for error printing during implementation stage.
import cgitb
# cgitb.enable(format = "text")

from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox

from pandas import DataFrame, read_csv, concat, Series

from numpy import isnan

from googleapiclient.discovery import build

from emoji import UNICODE_EMOJI

from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords, wordnet as wn, sentiwordnet as swn
from nltk.tag import pos_tag

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob import TextBlob

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
# Classifiers used.
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
# Classifiers tested but not used in final version.
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.ensemble import GradientBoostingClassifier
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.naive_bayes import MultinomialNB

from pylab import figure, close

```

```
from wordcloud import WordCloud

from PIL import Image

# Google Cloud Service API keys. If the daily quota is reached,
# use another API key. Here are some alternative API keys.
# API_KEY = "AIzaSyChlJ40_GmFw8n9Sw-2YJdyo4ury3kS2bw"
# API_KEY = "AIzaSyC_jbsqsPiCZZcfdac4DU6AVjcC54FuQ6Y"
# API_KEY = "AIzaSyBDVpl0jBBZj27WmCrGOH7HF-OkCCnwRek"

# Current directory path.
CURRENT_PATH = join(getcwd(), "Python")

# Max features to be used by count vectorizer and TF-IDF vectorizer.
MAX_FEATURES = 2000


class CommentFetcher:
    """Class that downloads comments from a given YouTube URL.

    Attributes:
        url (str): URL entered by user.
        title (str): Title of the video.
        comments (pandas DataFrame): Video's comments.
    """

    def __init__(self, url):
        """Inits CommentFetcher class with url."""
        print("Downloading Video's Comments")

        # Removes any excess whitespace in the URL.
        self.url = url.replace(" ", "")

        # Fetches the Video's title.
        query = parse.urlencode({"format": "json", "url": self.url})
        response = request.urlopen("https://www.youtube.com/oembed?" +
                                   query).read()
        data = loads(response.decode())
        self.title = data["title"]
        self.title = CommentCleaner().remove_punctuation_and_symbols(
            self.title).replace("   ", " ")

        # Removes any excess whitespace at the end of a video's title.
        while self.title[-1] == " ":
            self.title = self.title[:-1]

        self.retrieve_comments()

    def retrieve_comments(self):
        """Downloads a YouTube video's comments. Saves them in a DataFrame."""
        # Stores the URL's video ID.
        url_parsed = parse.urlparse(self.url)
        qsl = parse.parse_qs(url_parsed.query)
        video_id = qsl["v"][0]

        # Fetches the current API key from a text file.
        with open(join(CURRENT_PATH, "API Key.txt"), "r") as file:
            api_key = file.read().splitlines()

        # Creates a youtube resource object.
```

```
        youtube = build("youtube", "v3", developerKey=api_key)
        video_response = youtube.commentThreads().list(
            part="snippet", videoId=video_id).execute()

        fetched_comments = [[], []]
        # Iterates through the video's response.
        while video_response:
            # Extracts the comment's text and timestamp from each result object.
            for item in video_response["items"]:
                # Comment's text.
                fetched_comments[0].append(item["snippet"]["topLevelComment"]
                                           ["snippet"]["textDisplay"])
                # Comments date-time stamp.
                fetched_comments[1].append(item["snippet"]["topLevelComment"]
                                           ["snippet"]["publishedAt"])

            # If there are more comments that have not been downloaded,
            # go to the next results page.
            if "nextPageToken" in video_response:
                p_token = video_response["nextPageToken"]
                video_response = youtube.commentThreads().list(
                    part="snippet", videoId=video_id,
                    pageToken=p_token).execute()
            else:  # If all of the comments have been downloaded, stop.
                break

        # Removes any HTML tags from the dataset.
        for comment in range(len(fetched_comments[0])):
            new_comment = sub(
                '''<a href="|</a>|<br>|<b>|</b>|&quot;|&amp;|&#39;''', """'""",
                fetched_comments[0][comment])
            fetched_comments[0][comment] = new_comment

        # Saves the comments as a pandas DataFrame.
        self.comments = DataFrame({
            "Comment": fetched_comments[0],
            "Time": fetched_comments[1]
        })

        self.format_comments()

    def format_comments(self):
        # Replaces any new lines in the dataset with whitespaces.
        self.comments = self.comments.replace("\n", " ", regex=True)

        # Option of removing duplicate comments. Not used.
        # self.comments = self.comments.drop_duplicates(subset = "Comment")

        # Creates a folder for the current video.
        # Folder's name is the title of the video.
        path = join(CURRENT_PATH, "Videos", str(self.title))
        if not exists(join(CURRENT_PATH, "Videos", self.title)):
            mkdir(path)
            mkdir(join(path, "Charts"))

        # Saves the title of the video.
        with open(join(path, "title.pkl"), "wb") as file:
            dump((self.title), file)

        # Saves the raw comments of the video.
```

```
        self.comments.to_csv(join(path, "data.csv"), index=False)


class CommentCleaner:
    """Class that cleans the comments dataset passed into it.

    Attributes:
        comments (pandas DataFrame, optional): Video's comments.
            Defaults to None.
        spell_check (bool): If spelling correction should be
            performed on misspelled words. Defaults to None.
        corpus (bool): If the comments are a corpuses comments.
    """

    def __init__(self, comments=None, spell_check=False, corpus=False):
        """Inits CommentCleaner with comments."""
        self.comments = comments
        self.spell_check = spell_check
        self.corpus = corpus

    def clean(self):
        """Makes calls to each function in the class to clean the dataset."""
        # If the dataset is a corpus, removes additional Twitter tags.
        if self.corpus:
            print("Cleaning Corpus")
            self.remove_twitter_tags_comments()
        else:
            print("Cleaning Data")

        self.remove_excess_whitespace()
        self.remove_symbols_comments()
        self.remove_punctuation_and_symbols_comments()
        self.remove_urls_comments()
        self.lower_case_comments()
        self.remove_excess_whitespace()
        self.expand_contractions_comments()
        self.correct_spelling_comments()
        self.tokenise_comments()
        self.lemmatize_comments()
        self.remove_stopwords_comments()
        self.remove_excess_whitespace()

        # If the dataset is not a corpus, reformats the date-time stamps.
        if not self.corpus:
            self.format_date_time()

        return self.comments

    def remove_twitter_tags_comments(self):
        """Calls remove_twitter_tags() on each comment.

        Creates a new column with the returned data.
        """
        self.comments["Comment"] = self.comments["Comment"].apply(
            self.remove_twitter_tags)

    def remove_twitter_tags(self, comment):
        """Removes any Twitter tags from a string."""
        return sub(r"\@user|\@USER", "", comment)

```

```
    def remove_excess_whitespace(self):
        """Removes any rows from the data that contain whitespace only."""
        self.comments = self.comments.dropna()

    def remove_symbols(self, comment):
        """Removes symbols from a string.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            str: Comment with no symbols.
        """
        return "".join([
            x for x in comment
            if (x.isalpha() or x in [" ", ".", ",", "!", "?"] or
                x not in UNICODE_EMOJI)
        ])  # or x not in emoji.UNICODE_EMOJI

    def remove_symbols_comments(self):
        """Calls remove_symbols() on each comment

        Creates a new column with the returned data.
        """
        self.comments["no_symbols"] = self.comments["Comment"].apply(
            self.remove_symbols)

    def remove_punctuation_and_symbols(self, comment):
        """Removes punctuation and symbols from a string.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            str: Comment with no punctuation and symbols.
        """
        return "".join([x for x in comment if (x.isalpha() or x == " ")])

    def remove_punctuation_and_symbols_comments(self):
        """Calls remove_punctuation_and_symbols() on each comment

        Creates a new column with the returned data.
        """
        self.comments["no_punctuation_and_symbols"] = self.comments[
            "Comment"].apply(self.remove_punctuation_and_symbols)

    def remove_urls(self, comment):
        """Removes URLs from a string.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            str: Comment with no URLs.
        """
        return sub(r"\S*https?\S*", "", comment)

    def remove_urls_comments(self):
        """Calls remove_urls() on each comment

```

```
        Creates a new column with the returned data.
        """
        self.comments["no_urls"] = self.comments["no_symbols"].apply(
            self.remove_urls)
        self.comments["no_urls2"] = self.comments[
            "no_punctuation_and_symbols"].apply(self.remove_urls)

    def lower_case(self, comment):
        """Makes all characters in a string lower case.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            str: Comment with all characters in lower case.
        """
        lower = ""
        for character in comment:
            if character.isalpha():
                lower += character.lower()
            else:
                lower += character
        return lower

    def lower_case_comments(self):
        """Calls lower_case() on each comment

        Creates a new column with the returned data.
        """
        self.comments["lower_case"] = self.comments["no_urls2"].apply(
            self.lower_case)

    def expand_contractions(self, comment):
        """Expands all contractions in a string.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            str: Comment with expanded contractions.
        """
        expanded = ""
        for word in comment.split():
            if word in contractions:
                expanded += contractions[word] + " "
            else:
                expanded += word + " "
        return expanded

    def expand_contractions_comments(self):
        """Calls expand_contractions() on each comment

        Creates a new column with the returned data.
        """
        self.comments["expanded_contractions"] = self.comments[
            "lower_case"].apply(self.expand_contractions)
        self.comments["expanded_contractions2"] = self.comments[
            "no_urls"].apply(self.expand_contractions)

    def correct_spelling(self, comment):
```

```
        """Corrects the spelling of words in a string.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            str: Comment with corrected spelling.
        """
        new_comment = []
        for word in comment.split():
            try:
                new_comment.append(str(TextBlob(word).correct()))
            except:
                new_comment.append(word)
        return " ".join(new_comment)

    def correct_spelling_comments(self):
        """Calls correct_spelling() on each comment if the user has selected the
        spell correction option; creates a new column with the returned data.
        """
        if self.spell_check:
            print("Starting spelling correction. May take a while if the dataset is large. A statement will print when spelling correction finishes.")
            self.comments["spelling"] = self.comments[
                "expanded_contractions"].apply(self.correct_spelling)
            self.comments["spelling2"] = self.comments[
                "expanded_contractions2"].apply(self.correct_spelling)
            print("Spelling correction finished.")
        else:
            self.comments["spelling"] = self.comments["expanded_contractions"]
            self.comments["spelling2"] = self.comments["expanded_contractions2"]

    def tokenise(self, comment):
        """Turns a string into a list, splitting at whitespaces.

        Args:
            comment (list): A single comment from the dataset.

        Returns:
            list: Comment as a list.
        """
        return word_tokenize(comment)

    def tokenise_comments(self):
        """Calls tokenise() on each comment

        Creates a new column with the returned data.
        """
        self.comments["tokens"] = self.comments["spelling"].apply(self.tokenise)

    def lemmatize(self, comment):
        """Turns a string into its dictionary form, if it has one.

        Args:
            comment (list): A single comment from the dataset.

        Returns:
            list: Lemmatized comment.
        """
        lemmas = []
```

```
        for word in comment:
            lemmas.append(WordNetLemmatizer().lemmatize(word))
        return lemmas

    def lemmatize_comments(self):
        """Calls lemmatize() on each comment

        Creates a new column with the returned data.
        """
        self.comments["lemmas"] = self.comments["tokens"].apply(self.lemmatize)

    def remove_stopwords(self, comment):
        """Removes stop words from a string.

        Args:
            comment (list): A single comment from the dataset.

        Returns:
            list: Comment with no stop words.
        """
        new_comment = []
        for word in comment:
            if not word in stopwords.words("english"):
                new_comment.append(word)
        if new_comment == []:
            return None
        else:
            return new_comment

    def remove_stopwords_comments(self):
        """Calls remove_stopwords() on each comment

        Creates a new column with the returned data.
        """
        self.comments["lemmas_no_stopwords"] = self.comments["lemmas"].apply(
            self.remove_stopwords)

    def format_date_time(self):
        """Changes each comments timestamp into a more readable format."""
        self.comments["Time"] = self.comments["Time"].apply(
            lambda x: (x[0:10] + "\n" + x[11:19]))


class CommentSentiment:
    """Class that assigns sentiment and subjectivity scores to each comment.

    Attributes:
        comments (pandas DataFrame): Video's comments.
        vader_analyzer (SentimentIntensityAnalyzer): VADER lexicon analyser.
    """

    def __init__(self, comments):
        """Inits CommentSentiment class with comments."""
        self.comments = comments
        self.vader_analyzer = SentimentIntensityAnalyzer()

    def analyse(self):
        """Calls assign_sentiment_comments() andassign_subjectivity_comments()."""
        print("Assigning Sentiment to Comments")
        self.assign_sentiment_comments()
```

```
        self.assign_subjectivity_comments()
        return self.comments

    def assign_sentiment_comments(self):
        """Assigns sentiment scores and tags to each comment."""
        self.comments["Polarity"] = self.comments["spelling2"].apply(
            self.assign_valence)
        for _, row in self.comments.iterrows():  # For each comment
            # If the VADER and Textblob methods was not able to assign a
            # polarity to the current comment, tries the SentiWordNet method.
            if row["Polarity"] == 0.0:
                self.apply_swn(row)

        # Tags each comment with a sentiment label based on its polarity score.
        self.comments["Sentiment"] = self.comments["Polarity"].apply(
            self.assign_sentiment)

    def assign_valence(self, comment):
        """Assign a valence score to the comment."""
        vader_sentiment_dict = self.vader_analyzer.polarity_scores(comment)
        # If VADER was able to assign a polarity, returns this score.
        if vader_sentiment_dict["compound"] != 0:
            return vader_sentiment_dict["compound"]
        else:  # If VADER could not assign a polarity score.
            # Assigns a polarity score with TextBlob.
            return self.assign_textblob_valence(comment)

    def assign_textblob_valence(self, comment):
        """Assigns and returns a polarity score using the TextBlob lexicon."""
        return TextBlob(comment).sentiment.polarity

    def apply_swn(self, row):
        """Assigns a polarity score using the SentiWordNet method to a comment."""
        sentiment = 0.0
        # For each word and its POS tag in the comment.
        for word, tag in pos_tag(row["tokens"]):
            # Converts the POS tag into a WordNet POS tag.
            wn_tag = self.penn_to_wn(tag)
            # Lemmatizes the word.
            lemma = WordNetLemmatizer().lemmatize(word, pos=wn_tag)
            # Generates a subset of synonyms for the given word and its POS tag.
            synsets = wn.synsets(lemma, pos=wn_tag)
            if synsets:  # If a synset was found for the word.
                # Takes the first, most common sense (synset) of the word.
                synset = synsets[0]
                # Retrieves the sentiment of the word,
                # compound this to the total sentiment of the comment.
                swn_synset = swn.senti_synset(synset.name())
                sentiment += swn_synset.pos_score() - swn_synset.neg_score()
        row["Polarity"] == sentiment

    def penn_to_wn(self, tag):
        """Converts a POS tag to a WordNet POS tag."""
        if tag.startswith("J"):
            return wn.ADJ
        elif tag.startswith("N"):
            return wn.NOUN
        elif tag.startswith("R"):
            return wn.ADV
        elif tag.startswith("V"):
```

```
            return wn.VERB
        return wn.NOUN

    def assign_subjectivity_comments(self):
        """Calls assign_subjectivity() on each comment."""
        self.comments["Subjectivity"] = self.comments["spelling2"].apply(
            self.assign_subjectivity)

    def assign_subjectivity(self, comment):
        """Assigns a subjectivity score using the TextBlob lexicon.

        Args:
            comment (str): A single comment from the dataset.

        Returns:
            float: Subjectivity score of the comment.
        """
        return TextBlob(comment).sentiment.subjectivity

    def assign_sentiment(self, comment):
        """Assigns a sentiment label based on the given comments polarity score.

        Args:
            comment (list): A single comment from the dataset.

        Returns:
            str: Comment's sentiment label.
        """
        if comment >= 0.05:
            return "positive"
        elif comment <= -0.05:
            return "negative"
        else:
            return "neutral"


class CommentVectorizer:
    """Class that vectorizes the dataset passed into it.

    Attributes:
        comments (pandas DataFrame): Video's comments.
        frequencies (list): Count vectorized data for each n-gram.
        importance (list): TF-IDF vectorized data for each n-gram.
    """

    def __init__(self, comments):
        """Inits CommentVectorizer with comments."""
        self.comments = comments

    def process(self):
        """Calls vectorize() and tfidf()."""
        self.vectorize()
        self.tfidf()

    def vectorize(self):
        """Performs count vectorization on the dataset for unigrams, bigrams,
trigrams."""
        self.frequencies = []
        ngrams = [(1, 1), (2, 2), (3, 3)]
        for ngram in ngrams:  # For each n-gram.
```

```
            count_vectorizer = CountVectorizer(ngram_range=ngram,
                                               max_features=MAX_FEATURES)
            count = count_vectorizer.fit_transform(
                self.comments["lemmas_no_stopwords"].apply(
                    lambda x: " ".join(x)))
            df = DataFrame(count.toarray(),
                           columns=count_vectorizer.get_feature_names_out())
            # Generates a list of n-gram-frequency pairs, sorted by most frequent.
            df = df.T.sum(axis=1).sort_values(ascending=False)
            df.columns = ["Word", "Frequency"]
            self.frequencies.append(df)

    def tfidf(self):
        """Performs TF-IDF vectorization on the dataset for unigrams, bigrams,
trigrams."""
        self.importance = []
        ngrams = [(1, 1), (2, 2), (3, 3)]
        for ngram in ngrams:  # For each n-gram.
            tfidf_vectorizer = TfidfVectorizer(use_idf=True,
                                               smooth_idf=False,
                                               ngram_range=ngram,
                                               max_features=MAX_FEATURES)
            score = tfidf_vectorizer.fit_transform(
                self.comments["lemmas_no_stopwords"].apply(lambda x: " ".join(x)))
            df = DataFrame(score.toarray(),
                           columns=tfidf_vectorizer.get_feature_names_out())
            # Generates a list of n-gram-importance pairs, sorted by most important.
            self.importance.append(
                df.T.sum(axis=1).sort_values(ascending=False))


class CommentClassifier:
    """Class that classifies the dataset into categories.

    Attributes:
        comments (pandas DataFrame): Video's comments.

        offensive_path (str): File path of OffensEval corpus.
        emotion_path (str): File path of GoEmotions corpus.

        offensive_corpus (pandas DataFrame): OffensEval corpus.
        emotions_corpus (pandas DataFrame): GoEmotions corpus.
    """

    def __init__(self, comments):
        """Inits CommentClassifier with comments; saved cleaned corpus as CSV"""
        self.comments = comments

        self.offensive_path = join(CURRENT_PATH, "Corpora", "OffensEval")
        self.emotion_path = join(CURRENT_PATH, "Corpora", "Emotions")

        self.clean_corpora()

    def clean_corpora(self):
        """If the corpus has not been cleaned, clean it and save this as a CSV
file."""
        # If the OffensEval corpus has not yet been cleaned, cleans it.
        if not exists(join(self.offensive_path, "cleaned_offense_corpus.csv")):
            corpus = read_csv(join(self.offensive_path, "offence_corpus.csv"))
            corpus["Comment"] = corpus["Comment"].apply(str)
```

```
            corpus = CommentCleaner(corpus, False, True).clean()
            corpus.to_csv(join(self.offensive_path,
                               "cleaned_offense_corpus.csv"),
                               index=False)
        # Reads the cleaned OffensEval corpus into a DataFrame.
        self.offensive_corpus = read_csv(
            join(self.offensive_path, "cleaned_offense_corpus.csv"))

        # If the GoEmotions corpus has not yet been cleaned, cleans it.
        if not exists(join(self.emotion_path, "cleaned_emotions_corpus.csv")):
            corpus = read_csv(join(self.emotion_path, "goemotions_1.csv"))
            corpus = concat([corpus,
                             read_csv(join(self.emotion_path, "goemotions_2.csv"))],
                             ignore_index=True)
            corpus = concat(
                [corpus,
                 read_csv(join(self.emotion_path, "goemotions_3.csv"))],
                 ignore_index=True)
            corpus = corpus.rename(columns={'text': 'Comment'})
            corpus["Category"] = corpus.apply(self.assign_emotion, axis=1)
            corpus = CommentCleaner(corpus, False, True).clean()
            corpus[["Comment", "Category", "lemmas_no_stopwords"
                   ]].to_csv(join(self.emotion_path,
                                  "cleaned_emotions_corpus.csv"),
                                  index=False)
        # Reads the cleaned GoEmotions corpus into a DataFrame.
        self.emotions_corpus = read_csv(
            join(self.emotion_path, "cleaned_emotions_corpus.csv"))

    def assign_emotion(self, row):
        """When cleaning the GoEmotions corpus, assign an emotion to each comment.

        Args:
            row (pandas DataFrame single row): Row of a comment from the corpus.

        Returns:
            str: Assigned emotion category.
        """
        emotions = [
            "admiration", "amusement", "anger", "annoyance", "approval",
            "caring", "confusion", "curiosity", "desire", "disappointment",
            "disapproval", "disgust", "embarrassment", "excitement", "fear",
            "gratitude", "grief", "joy", "love", "nervousness", "optimism",
            "pride", "realization", "relief", "remorse", "sadness", "surprise",
            "neutral"
        ]
        for emotion in emotions:
            if row[emotion] == 1:
                return emotion
        return "neutral"

    def classify(self):
        """Classifies the comments.

        Returns:
            pandas DataFrame: Classified comments dataset.
        """
        print("Classifying Data")
        # Elements needed for each corpus.
        elements = [[self.offensive_corpus, self.emotions_corpus],
```

```
                    ["offence_svm", "emotions_decision_tree"],
                    [DecisionTreeClassifier(), SVC(kernel="linear", gamma="auto")],
                    [self.offensive_path, self.emotion_path],
                    ["Offensive?", "Emotion"]]

        for n in range(len(elements[0])):  # For each of the corpora.
            # Transforms the labels into numbers.
            encoder = LabelEncoder()
            labels = encoder.fit_transform(elements[0][n]["Category"])

            # Performs count vectorization on the cleaned corpus.
            count_vectorizer = CountVectorizer(max_features=5000)
            count_vectorizer.fit(
                self.emotions_corpus["lemmas_no_stopwords"].apply(
                    lambda x: " ".join(literal_eval(x))))
            classifier_input = count_vectorizer.transform(
                self.comments["lemmas_no_stopwords"].apply(
                    lambda x: " ".join(x)))

            # If the classifier has not been trained on the corpus previously, do
so.
            if not exists(join(elements[3][n], elements[1][n] + ".pkl")):
                training_data = count_vectorizer.transform(
                    self.emotions_corpus["lemmas_no_stopwords"].apply(
                        lambda x: " ".join(literal_eval(x))))
                # Fits the training data into the classifier.
                classifier = elements[2][n]
                classifier.fit(training_data, labels)
                # Saves the trained classifier as a pickle file.
                with open(join(elements[3][n], elements[1][n] + ".pkl"), "wb") as
file:
                    dump((classifier), file)
            else:  # If the classifier has already been trained, loads this data.
                with open(join(elements[3][n], elements[1][n] + ".pkl"), "rb") as
file:
                    classifier = load(file)

            # Assigns a category to each comment in the user input dataset.
            predictions = classifier.predict(classifier_input)
            # Converts the numerical assigned categories to their labels.
            labels = encoder.inverse_transform(predictions)
            self.comments[elements[4][n]] = labels

        return self.comments


class Main:
    """Main class. Makes calls to all other classes.

    Time ranges refer to the subset of comments posted in the last
    30 days, 7 days, 48 hours, and all time.

    Attributes:
        app (QApplication): PyQt5 application used to display windows.
        menu (Menu): Menu window.
        start_time (time): Start of processing time, used when calculating
            the total processing time.
        file_name (str): Name of video's file.
        file_path (str): File path to the current video.

```

```
        comments (pandas DataFrame): Video's comments.
        comments_30_days (pandas DataFrame): Video's comments from the last 30 days.
        comments_7_days (pandas DataFrame): Video's comments from the last 7 days.
        comments_48_hours (pandas DataFrame): Video's comments from the last 48
hours.
        comments_time_ranges (list): List of all of the above comments DataFrames.

        vectorizer_data (list): Vectorizer data for unigrams, bigrams, trigrams.
        report (pandas DataFrame): Report about the comments.
    """

    def __init__(self):
        """Inits Main class."""
        print("Loading Application")
        self.generate_directories()
        self.app = QApplication(argv)

        # Applies a stylesheet to the whole application.
        qss = open(join(CURRENT_PATH, "StyleSheets", "Combinear.qss"), "r").read()
        self.app.setStyleSheet(qss)

        # Initially, the user has not picked any menu option.
        option_selected = False
        # While the user has not picked an option or has entered an invalid URL.
        while not option_selected:
            self.display_window("Menu")
            if self.menu.option == "YouTube":  # If the user wants to use a YouTube
URL.
                try:  # Attempts to download comments from the specified URL.
                    self.start_time = time()  # Start of processing time tracking.
                    fetcher = CommentFetcher(self.menu.url)
                    self.file_name = fetcher.title
                    self.file_path = join(CURRENT_PATH, "Videos",
                                          self.file_name)
                    self.comments = fetcher.comments
                    if self.comments.empty:  # If the video does not have any
comments.
                        self.display_error("This video does not have any comments.
Please enter a valid YouTube URL.")
                    else:
                        self.process_comments()
                        option_selected = True
                except:  # If there was a problem when downloading or processing the
comments.
                    self.display_error("Please enter a valid YouTube URL.")
            elif self.menu.option == "CSV":  # If the user wants to view a
previously processed dataset.
                self.file_name = self.menu.file_selection
                self.file_path = join(CURRENT_PATH, "Videos", self.file_name)
                self.read_from_files()
                option_selected = True

            # If the user has selected an option and there were no problems
downloading and processing the comments.
            if self.menu.option != None and option_selected:
                # Displays the main window.
                self.display_window("Main")
                option_selected = False
            elif self.menu.option == None:  # Else if the user has pressed the close
button on the menu.
```

```
                break

    def generate_directories(self):
        """Generate any directories used by the system if they do not already
exist."""
        # If the "Videos" folder does not exist, creates it.
        if not exists(join(CURRENT_PATH, "Videos")):
            mkdir(join(CURRENT_PATH, "Videos"))

    def display_window(self, selection):
        """Displays specified window.

        Args:
            selection (str): Window to be displayed.
        """
        window = QMainWindow()
        if selection == "Menu":
            self.menu = Menu(window)
            window.show()
        else:
            # Saves the main window object as a variable so that it does
            # not go out of scope, closing the window.
            main_window = MainWindow(window, self.file_name, self.report,
                                     self.comments_time_ranges,
                                     self.emotional_analysis,
                                     self.vectorizer_data)

            # If the screen size of the current machine is small, maximizes the
window.
            if self.app.primaryScreen().size().height() < 960:
                window.showMaximized()
            else:
                window.show()
        self.app.exec_()

    def display_error(self, message):
        """Displays error message window.

        Args:
            message (str): Error message to be displayed.
        """
        error_dialog = QMessageBox()
        error_dialog.setText(message)
        error_dialog.exec()

    def generate_empty_dataframes(self):
        """Generates empty DataFrames for the different time ranges."""
        self.comments_30_days = DataFrame(columns=self.comments.columns)
        self.comments_7_days = DataFrame(columns=self.comments.columns)
        self.comments_48_hours = DataFrame(columns=self.comments.columns)

    def process_comments(self):
        """Makes calls to other functions and classes to clean and process the
comments."""
        self.comments = CommentCleaner(self.comments,
                                       self.menu.spell_check).clean()
        self.comments = CommentSentiment(self.comments).analyse()
        classifier = CommentClassifier(self.comments)
        self.comments = classifier.classify()

```

```
        self.generate_empty_dataframes()
        self.split_data_by_time_ranges()
        self.comments_time_ranges = [
            self.comments, self.comments_30_days, self.comments_7_days,
            self.comments_48_hours
        ]

        self.vectorize_data()

        self.generate_report()
        self.generate_charts()
        self.generate_files()

    def split_data_by_time_ranges(self):
        """Fills time range DataFrames with any comments that fit the specified time range."""
        for index, row in self.comments.iterrows():  # For each comment.
            current_datatime = datetime(year=int(row["Time"][0:4]),
                                        month=int(row["Time"][5:7]),
                                        day=int(row["Time"][8:10]))
            # Length of time since the current comment was posted.
            datatime_difference = (datetime.today() - current_datatime).days
            # If the comment was posted in the last 30 days,
            # appends it to the relevant DataFrame.
            if datatime_difference < 30:
                self.comments_30_days = concat([
                    self.comments_30_days,
                    self.comments.loc[index].to_frame().transpose()
                ], ignore_index=True)
                # If the comment was posted in the last 7 days,
                # appends it to the relevant DataFrame.
                if datatime_difference < 7:
                    self.comments_7_days = concat([
                        self.comments_7_days,
                        self.comments.loc[index].to_frame().transpose()
                    ], ignore_index=True)
                    # If the comment was posted in the last 48 hours,
                    # appends it to the relevant DataFrame.
                    if datatime_difference < 2:
                        self.comments_48_hours = concat([
                            self.comments_48_hours,
                            self.comments.loc[index].to_frame().transpose()
                        ], ignore_index=True)

        # Formats the data.
        for dataframe in [
                self.comments_30_days, self.comments_7_days,
                self.comments_48_hours
        ]:
            dataframe["Polarity"] = dataframe["Polarity"].apply(
                lambda x: x.item())
            dataframe["Subjectivity"] = dataframe["Subjectivity"].apply(
                lambda x: x.item())

    def vectorize_data(self):
        """Makes calls to the vectorizer for all of the data by time range."""
        print("Vectorizing Data")
        self.vectorizer_data = []
        self.emotional_data = []
        # For each time range of comments.
```

```
        for dataframe in self.comments_time_ranges:
            vectorizer = CommentVectorizer(dataframe)
            # If data exists for this time range, appends the relevant vectorizer data.
            if not dataframe.empty:
                vectorizer.process()
                self.vectorizer_data.append([
                    vectorizer.frequencies[0], vectorizer.importance[0],
                    vectorizer.frequencies[1], vectorizer.importance[1],
                    vectorizer.frequencies[2], vectorizer.importance[2]
                ])
            # If no data exists for the time range, appends empty DataFrames.
            else:
                self.vectorizer_data.append([
                    DataFrame(),
                    DataFrame(),
                    DataFrame(),
                    DataFrame(),
                    DataFrame(),
                    DataFrame()
                ])

    def generate_report(self):
        """Generates a report about the comments."""
        print("Generating Report")
        self.report = DataFrame({
            " ": [
                "Positive", "Neutral", "Negative", "Total", "Average Valence",
                "Average Subjectivity", "Offensive", "Inoffensive",
                "Processing Time"
            ]
        })
        self.emotional_analysis = [[], [], [], []]
        time_frame_labels = [
            "All Time", "Last Month", "Last Week", "Last 48 Hours"
        ]
        # For each time range subset.
        for dataframe in range(len(self.comments_time_ranges)):
            sentiment_dict = {"positive": 0, "negative": 0, "neutral": 0}
            current_subset = self.comments_time_ranges[dataframe]
            if not current_subset.empty:  # If data exists for this time range.
                # Calculates the total number of positive, negative, and neutral comments.
                for index, value in current_subset["Sentiment"].value_counts(
                ).items():
                    sentiment_dict[index] = int(value)

                # Calculates the total number of comments.
                total = 0
                for value in sentiment_dict.values():
                    total += value

                # Calculates the average polarity and subjectivity from the scores.
                comments_valence_no_0 = current_subset["Polarity"].drop(
                    current_subset["Polarity"][current_subset["Polarity"] == 0.0].index)
                averages = [
                    comments_valence_no_0.sum().item() /
                    comments_valence_no_0.count(),
                    current_subset["Subjectivity"].sum() /
```

```
                    current_subset["Subjectivity"].count()
                ]
                for n in range(len(averages)):
                    # If the value has more than there decimal places, shortens it.
                    if str(averages[n].item())[::-1].find('.') > 3:
                        averages[n] = round(averages[n], 3)
                    if isnan(averages[n]):
                        averages[n] = 0
                    else:
                        averages[n] = averages[n].item()

                # If the current subset is of time range "all time",
                # calculates the processing time.
                processing_time = " "
                if dataframe == 0:
                    processing_time = str(round(time() - self.start_time,
                                                1)) + " Secs"

                # Counts the number of offensive and inoffensive comments.
                offensive_dict = {"Yes": 0, "No": 0}
                offensive_count = current_subset["Offensive?"].value_counts()
                for label in ["Yes", "No"]:
                    try:
                        offensive_dict[label] = offensive_count[label].item()
                    except:
                        pass

                # Appends all of the statistics to the report.
                self.report[time_frame_labels[dataframe]] = [
                    sentiment_dict.get("positive"),
                    sentiment_dict.get("neutral"),
                    sentiment_dict.get("negative"), total, averages[0],
                    averages[1], offensive_dict["Yes"], offensive_dict["No"],
                    processing_time
                ]

                # Counts the frequency of each emotion in the dataset.
                emotion_dict = {"admiration": 0, "amusement": 0, "anger": 0,
                                "annoyance": 0, "approval": 0, "caring": 0,
                                "confusion": 0, "curiosity": 0, "desire": 0,
                                "disappointment": 0, "disapproval": 0,
                                "disgust": 0, "embarrassment": 0,
                                "excitement": 0, "fear": 0, "gratitude": 0,
                                "grief": 0, "joy": 0, "love": 0,
                                "nervousness": 0, "optimism": 0, "pride": 0,
                                "realization": 0, "relief": 0, "remorse": 0,
                                "sadness": 0, "surprise": 0, "neutral": 0
                }

                for index, value in
current_subset["Emotion"].value_counts().items():
                    emotion_dict[index] = int(value)
                # Sorts the dictionary by the highest frequency value.
                emotion_dict = dict(
                    sorted(emotion_dict.items(),
                           key=lambda item: item[1],
                           reverse=True))
                # Converts the dictionary into a list of key, value pairs.
                for key, value in emotion_dict.items():
                    self.emotional_analysis[dataframe].append([key, value])
```

```
                # File names for each of the emotional analysis charts.
                chart_names = [
                    "emotions bar chart all", "emotions bar chart 30",
                    "emotions bar chart 7", "emotions bar chart 48"
                ]
                cloud_names = [
                    "emotions word cloud all", "emotions word cloud 30",
                    "emotions word cloud 7", "emotions word cloud 48"
                ]
                # Charts do not include the neutral emotion category.
                emotion_dict.pop("neutral")
                emotional_subset = Series(emotion_dict)
                # If data exists in this subset, generate charts for this data.
                if not emotional_subset.empty and emotional_subset.iloc[0] != 0:
                    self.generate_chart(emotional_subset,
                                        chart_names[dataframe],
                                        cloud_names[dataframe])
            else:  # If no data exists in this time range.
                self.report[time_frame_labels[dataframe]] = [
                    0, 0, 0, 0, 0, 0, 0, 0, " "
                ]

    def generate_chart(self, subset, bar_chart_name, word_cloud_name):
        """_summary_

        Args:
            subset (pandas DataFrames): Current subset to generate charts for.
            bar_chart_name (str): File name to save the bar chart as.
            word_cloud_name (str): File name to save the word cloud as.
        """
        # Generates a bar chart for the current DataFrame.
        # Sets chart's background colour as grey.
        fig = figure(facecolor="#3a3a3a")
        ax = fig.gca()
        ax.set_autoscale_on(False)
        # Takes the first 20 values.
        ax.bar(subset.iloc[:20].index.values, subset.iloc[:20])
        ax.axis([0, 20, 0, subset.max()])
        # Rotates the axis labels 90 degrees clockwise.
        ax.tick_params(axis="x", rotation=90, labelcolor="#ffffff")
        ax.tick_params(axis="y", rotation=90, labelcolor="#ffffff")

        # Styles the charts axes.
        ax.grid(axis="y", linewidth=0.5, alpha=0.65)
        ax.set_facecolor("#3a3a3a")

        # Save the bar chart as a PNG file.
        current_image_path = join(self.file_path, "Charts",
                                  bar_chart_name + ".png")
        fig.savefig(current_image_path, bbox_inches="tight")
        close(fig)

        # Rotates the bar chart 90 degrees clockwise.
        if exists(current_image_path):
            image = Image.open(current_image_path)
            rotated = image.rotate(-90, expand=True)
            rotated.save(current_image_path, "PNG")

        # Generates a word cloud for the current DataFrame.
```

```
        # Saves the word cloud as a PNG file.
        WordCloud(background_color="#3a3a3a",
                  max_words=50).generate_from_frequencies(subset).to_file(
                      join(self.file_path, "Charts", word_cloud_name + ".png"))

    def generate_files(self):
        """Outputs any processed data to CSV files."""
        # Store cleaned comments as CSV file.
        self.comments[[
            "Comment", "no_symbols", "no_punctuation_and_symbols", "no_urls",
            "no_urls2", "lower_case", "expanded_contractions",
            "expanded_contractions2", "spelling", "spelling2", "tokens",
            "lemmas", "lemmas_no_stopwords"
        ]].to_csv(join(self.file_path, "cleaned_data.csv"), index=False)

        # Store report as CSV file.
        self.report.to_csv(join(self.file_path, "report.csv"), index=False)

        # Store processed comments as CSV files.
        file_names = [
            "processed_all", "processed_30_days", "processed_7_days",
            "processed_48_hours"
        ]
        for comments_set in range(len(self.comments_time_ranges)):
            self.comments_time_ranges[comments_set][[
                "Comment", "Time", "Polarity", "Sentiment", "Subjectivity",
                "Offensive?", "Emotion"
            ]].to_csv(join(self.file_path, file_names[comments_set] + ".csv"),
index=False)

        # Store emotional analysis data as pickle file.
        with open(join(self.file_path, "emotional_analysis.pkl"), "wb") as file:
            dump((self.emotional_analysis), file)

        # Store vectorizer data as pickle file.
        with open(join(self.file_path, "vectorizer_data.pkl"), "wb") as file:
            dump((self.vectorizer_data), file)

    def generate_charts(self):
        """Generates bar charts and word clouds based on vectorizer data."""
        print("Generating Charts")
        # Bar chart file names for each vectorizer and time range subset.
        chart_names = [
            ["frequency unigram bar chart all", "frequency unigram bar chart 30",
            "frequency unigram bar chart 7", "frequency unigram bar chart 48"],

            ["importance unigram bar chart all", "importance unigram bar chart 30",
             "importance unigram bar chart 7", "importance unigram bar chart 48"],

            ["frequency bigram bar chart all", "frequency bigram bar chart 30",
             "frequency bigram bar chart 7", "frequency bigram bar chart 48"],

            ["importance bigram bar chart all", "importance bigram bar chart 30",
             "importance bigram bar chart 7", "importance bigram bar chart 48"],

            ["frequency trigram bar chart all", "frequency trigram bar chart 30",
             "frequency trigram bar chart 7", "frequency trigram bar chart 48"],

            ["importance trigram bar chart all", "importance trigram bar chart 30",
             "importance trigram bar chart 7", "importance trigram bar chart 48"]
```

```
        ]
        # Word cloud file names for each vectorizer and time range subset.
        cloud_names = [
            ["frequency unigram word cloud all", "frequency unigram word cloud 30",
             "frequency unigram word cloud 7", "frequency unigram word cloud 48"],

            ["importance unigram word cloud all", "importance unigram word cloud
30",
             "importance unigram word cloud 7", "importance unigram word cloud 48"],

            ["frequency bigram word cloud all", "frequency bigram word cloud 30",
             "frequency bigram word cloud 7", "frequency bigram word cloud 48"],

            ["importance bigram word cloud all", "importance bigram word cloud 30",
             "importance bigram word cloud 7", "importance bigram word cloud 48"],

            ["frequency trigram word cloud all", "frequency trigram word cloud 30",
             "frequency trigram word cloud 7", "frequency trigram word cloud 48"],

            ["importance trigram word cloud all", "importance trigram word cloud
30",
             "importance trigram word cloud 7", "importance trigram word cloud 48"]
        ]
        # For each vectorizer n-gram.
        for data_type in range(len(chart_names) - 1, -1, -1):
            for time_range in range(0, 4):  # For each time range.
                # The current vectorizer n-gram and time range data subset.
                vectorizer_subset = self.vectorizer_data[time_range][data_type]
                if not vectorizer_subset.empty:  # If data exists in this subset.
                    self.generate_chart(vectorizer_subset,
                                        chart_names[data_type][time_range],
                                        cloud_names[data_type][time_range])

    def read_from_files(self):
        """Reads in a previously processed dataset."""
        with open(join(self.file_path, "title.pkl"), "rb") as file:
            self.title = load(file)  # Title of the video.
        self.comments = read_csv(join(self.file_path, "data.csv"))  # Raw comments.
        self.report = read_csv(join(self.file_path, "report.csv"))  # Report.

        self.generate_empty_dataframes()
        self.comments_time_ranges = [
            self.comments, self.comments_30_days,
            self.comments_7_days, self.comments_48_hours
        ]

        # Reads in processed comments for each time range.
        file_names = [
            "processed_all", "processed_30_days",
            "processed_7_days", "processed_48_hours"
        ]
        for comments_set in range(len(self.comments_time_ranges)):
            self.comments_time_ranges[comments_set] = read_csv(
                join(self.file_path, file_names[comments_set] + ".csv"))

        # Reads in emotional analysis data.
        with open(join(self.file_path, "emotional_analysis.pkl"), "rb") as file:
            self.emotional_analysis = load(file)

        # Reads in vectorizer data.
```

```
        with open(join(self.file_path, "vectorizer_data.pkl"), "rb") as file:
            self.vectorizer_data = load(file)


if __name__ == "__main__":
    Main()

```

### C.1.2 gui.py

```python
# gui.py: GUI classes used in YouTube Comments Analyser application.
#
# Menu and MainWindow classes originally created by PyQt5 UI code generator 5.15.6
# Writen and modified by Daniel Van Cuylenburg (k19012373).
#

# Imports.
from PyQt5.QtCore import QAbstractTableModel, QCoreApplication, QRect, QMetaObject,
Qt
from PyQt5.QtGui import QPixmap, QMovie, QFont, QFont
from PyQt5.QtWidgets import (QButtonGroup, QLineEdit, QWidget, QPushButton,
                             QComboBox, QMenuBar, QStatusBar, QDesktopWidget,
                             QHeaderView, QLabel, QTableView, QRadioButton,
                             QMenu, QStatusBar, QAction, QGroupBox, QGridLayout,
                             QHBoxLayout, QCheckBox)

from numpy import array, shape
from os import listdir, getcwd, startfile
from os.path import join

# Current directory path.
CURRENT_PATH = join(getcwd(), "Python")


class QLineEdit(QLineEdit):
    """QLineEdit overriding class.

    Allows the QLineEdit text prompt to dissapear when clicked.

    Attributes:
        clicked_once (bool): True if the QLineEdit has been clicked
            more than once, False otherwise.
    """

    def __init__(self, parent):
        """Inits QLineEdit class with parent object."""
        super(QLineEdit, self).__init__(parent)
        self.clicked_once = False

    def mousePressEvent(self, event):
        """If clicked for the first time, hide the text prompt.

        Args:
            event (PyQt5.QtGui.QMouseEvent): PyQt5 object storing
                mouse click information. Not used.
        """
        if not self.clicked_once:
            self.setText(QCoreApplication.translate("menu_window", ""))
            self.clicked_once = True


class Menu:
    """Menu GUI class.

    Attributes:
        menu_window (PyQt5.QtWidgets.QMainWindow): Menu window object.
        url (str): Text entered into text box.
        spell_check (bool): True if spell check check box is selected, False
otherwise.
```

```
        option (str): Tracks which button the user clicks. "YouTube" or "CSV".
        file_selection (str): Name of file currently selected in the drop box.
    """

    def __init__(self, window):
        """Inits Menu class."""
        self.menu_window = window
        self.url = ""
        self.spell_check = False
        self.option = None

        self.setup_ui()

    def from_url(self):
        """Sets youtube option & URL variables. Closes Menu window."""
        self.url = self.text_box.text()
        self.option = "YouTube"
        self.menu_window.close()

    def from_file(self):
        """Sets from CSV file option. Closes Menu window."""
        self.option = "CSV"
        self.menu_window.close()

    def set_file_selection(self):
        """Sets file_selection variable as current file selected in combo box."""
        self.file_selection = self.combo_box.currentText()

    def check_box_clicked(self, state):
        """Set spell_check variable based on if the check box is checked.

        Args:
            state (int): 2 if check box is checked, 0 otherwise.
        """
        if state == 2:  # If the check box is ticked.
            self.spell_check = True
        else:
            self.spell_check = False

    def setup_ui(self):
        """Sets up the GUI."""
        self.menu_window.setObjectName("menu_window")
        self.menu_window.resize(685, 206)  # Resize menu window.

        self.central_widget = QWidget(self.menu_window)
        self.central_widget.setObjectName("central_widget")

        # Import from video URL button object.
        self.url_button = QPushButton(self.central_widget)
        self.url_button.setEnabled(True)
        self.url_button.setGeometry(QRect(20, 80, 301, 91))
        font = QFont()  # Font object used to set text size.
        font.setPointSize(16)
        self.url_button.setFont(font)
        self.url_button.setObjectName("url_button")
        # Runs from_url() when clicked.
        self.url_button.clicked.connect(self.from_url)

        # Import from CSV file button object.
        self.csv_button = QPushButton(self.central_widget)
```

```
        self.csv_button.setGeometry(QRect(360, 80, 291, 91))
        font.setPointSize(16)
        self.csv_button.setFont(font)
        self.csv_button.setObjectName("csv_button")
        # Runs from_file() when clicked.
        self.csv_button.clicked.connect(self.from_file)

        # URL text box object.
        self.text_box = QLineEdit(self.central_widget)
        self.text_box.setGeometry(QRect(20, 10, 301, 41))
        self.text_box.setClearButtonEnabled(False)
        self.text_box.setObjectName("text_box")
        # Sets font size to 12.
        font.setPointSize(12)
        self.text_box.setFont(font)

        # Get all file names of files found in "Videos" folder.
        file_names = []
        for video_file in listdir(join(CURRENT_PATH, "Videos")):
            file_names.append(video_file)

        # Combo box object.
        self.combo_box = QComboBox(self.central_widget)
        self.combo_box.setGeometry(QRect(340, 10, 331, 41))
        self.combo_box.setObjectName("combo_box")
        # Sets font size to 12.
        font.setPointSize(12)
        self.combo_box.setFont(font)

        # Check box object.
        self.check_box = QCheckBox(self.central_widget)
        self.check_box.setGeometry(QRect(20, 60, 300, 17))
        self.check_box.setObjectName("check_box")
        self.check_box.stateChanged.connect(self.check_box_clicked)
        self.combo_box.addItems(file_names)
        # Runs set_file_selection() when clicked.
        self.combo_box.currentIndexChanged.connect(self.set_file_selection)
        # Sets current file as first file in combo box.
        self.set_file_selection()

        # Menu bar configuration
        self.menu_window.setCentralWidget(self.central_widget)
        self.menu_bar = QMenuBar(self.menu_window)
        self.menu_bar.setGeometry(QRect(0, 0, 685, 21))
        self.menu_bar.setObjectName("menu_bar")
        self.menu_window.setMenuBar(self.menu_bar)
        self.status_bar = QStatusBar(self.menu_window)
        self.status_bar.setObjectName("status_bar")
        self.menu_window.setStatusBar(self.status_bar)

        self.retranslate_ui()
        QMetaObject.connectSlotsByName(self.menu_window)

        # Move window to centre of screen.
        qt_rectangle = self.menu_window.frameGeometry()
        qt_rectangle.moveCenter(QDesktopWidget().availableGeometry().center())
        self.menu_window.move(qt_rectangle.topLeft())

    def retranslate_ui(self):
        """Set text on all objects."""
```

```
        translate = QCoreApplication.translate
        self.menu_window.setWindowTitle(translate("menu_window", "Menu"))
        self.url_button.setText(
            translate("menu_window", "Import from Youtube URL"))
        self.csv_button.setText(translate("menu_window",
                                          "Import from CSV file"))
        self.text_box.setText(translate("menu_window", "Enter YouTube URL"))
        self.combo_box.setToolTip(translate("menu_window", "Select CSV File"))
        self.check_box.setText(
            translate(
                "menu_window",
                "Spelling Correction Enabled (May Increase Processing Time)"))


class ListTableModel(QAbstractTableModel):
    """Class to display a list as a table.

    Class to display count & TF-IDF vectorizer data in a table.
    Inherits PyQt5.QtCore.QAbstractTableModel.

    Attributes:
        data (list): Vectorizer data.
        cols (list): Names of table columns.
    """

    def __init__(self, data):
        """Inits ListTableModel with data."""
        super(ListTableModel, self).__init__()
        self.data = data
        self.cols = ["n-gram", "Score"]

    def data(self, index, role):
        if role == Qt.DisplayRole:
            return self.data[index.row()][index.column()]

    def rowCount(self, index):
        return len(self.data)

    def columnCount(self, index):
        return len(self.data[0])

    def headerData(self, p_int, orientation, role):
        if role == Qt.DisplayRole:
            if orientation == Qt.Horizontal:
                return self.cols[p_int]
            elif orientation == Qt.Vertical:
                return p_int + 1
        return None


class DataframeTableModel(QAbstractTableModel):
    """Class to populate a table view with a pandas dataframe.

    Inherits PyQt5.QtCore.QAbstractTableModel.

    This class has been taken and adapted from:
    https://learndataanalysis.org/display-pandas-dataframe-with-pyqt5-qtableview-
widget/

    Attributes:
```

```
        data (DataFrame): Report or processed comments.
        cols (list): Names of table columns.
        r (int): Number of rows.
        c (int): Number of columns.
    """

    def __init__(self, data, parent=None):
        """Inits DataframeTableModel with data."""
        QAbstractTableModel.__init__(self, parent)
        self.data = array(data.values)
        self.cols = data.columns
        self.r, self.c = shape(self.data)

    def rowCount(self, parent=None):
        return self.r

    def columnCount(self, parent=None):
        return self.c

    def data(self, index, role=Qt.DisplayRole):
        if index.isValid():
            if role == Qt.DisplayRole:
                return self.data[index.row(), index.column()]
        return None

    def headerData(self, p_int, orientation, role):
        if role == Qt.DisplayRole:
            if orientation == Qt.Horizontal:
                return self.cols[p_int]
            elif orientation == Qt.Vertical:
                return p_int
        return None


class MainWindow:
    """Main Window GUI class.

    Attributes:
        comments (pandas DataFrame): Video's comments.
        screen (str): Currently displayed screen. "report" or "comments" or
            "frequency unigram", "importance unigram", "frequency bigram",
            "importance bigram", "frequency trigram", "importance trigram".
        chart (str): Currently displayed chart on frequency and
            importance screens. "bar chart" or "word cloud".
        filter (str): Currently set time range filter.
            "all" or "30" or "7" or "48".
        table (list of lists): Currently selected vectorizer data.
        sort_by (str): Currently selected sort by option on
            list of comments screen. "positive" or "negative" or

        report (pandas DataFrame): Report about comments.
    """

    def __init__(self, window, file_name, report, comments_time_ranges,
                 emotional_analysis, vectorizer_data):
        """Inits MainWindow class."""
        self.comments = None
        self.screen = "report"
        self.chart = "bar chart"
        self.filter = "all"
```

```
        self.table = None
        self.sort_by = "positive"

        self.import_data(file_name, report, comments_time_ranges,
                         emotional_analysis, vectorizer_data)

        self.setup_ui(window)

    def refresh_ui(self):
        """Display appropriate GUI elements based on current page."""
        self.hide_all()

        if self.screen == "report":
            self.main_table.setStyleSheet(self.report_table_qss)
            self.main_table.setModel(DataframeTableModel(self.report))
            header = self.main_table.horizontalHeader()
            # Resize headers.
            for i in range(0, len(header)):
                header.setSectionResizeMode(i, QHeaderView.ResizeToContents)
            # Resize row height.
            for i in range(0, len(self.report)):
                self.main_table.setRowHeight(i, 75)
            self.main_table.show()

        elif self.screen == "comments":
            self.sort_by_box.show()
            self.time_box.show()
            self.filter_buttons_controls()
            self.sort_buttons_controls()
            self.main_table.setStyleSheet(self.comments_table_qss)
            self.main_table.setModel(
                DataframeTableModel(self.comments[[
                    "Comment", "Time", "Polarity", "Sentiment", "Subjectivity",
                    "Offensive?", "Emotion"
                ]]))

            # Resize headers.
            header = self.main_table.horizontalHeader()
            header.setSectionResizeMode(0, QHeaderView.Stretch)
            header.setSectionResizeMode(1, QHeaderView.ResizeToContents)
            header.setSectionResizeMode(2, QHeaderView.ResizeToContents)
            header.setSectionResizeMode(3, QHeaderView.ResizeToContents)

            # Resize row height.
            for i in range(len(self.comments)):
                self.main_table.setRowHeight(i, 60)
            self.main_table.show()

        elif self.screen in [
                "emotions", "frequency unigram", "importance unigram",
                "frequency bigram", "importance bigram", "frequency trigram",
                "importance trigram"
        ]:
            self.chart_box.show()
            self.time_box.show()
            self.filter_buttons_controls()
            # If there are no n-grams with a frequency larger than 1.
            if self.table and self.table[0][1] == 1 and self.table[-1][1] == 1:
                self.chart_table.setText(
                    "No n-grams with a frequency larger than 1!")
```

```
            elif self.table and (self.table[0][1] != 1 or
                                 self.table[-1][1] != 1):
                self.vectorizer_table.setModel(ListTableModel(self.table))

                # Resize headers.
                header = self.vectorizer_table.horizontalHeader()
                header.setSectionResizeMode(0, QHeaderView.Stretch)
                header.setSectionResizeMode(1, QHeaderView.ResizeToContents)

                # Resize row height.
                for i in range(len(self.table)):
                    self.vectorizer_table.setRowHeight(i, 48)
                self.vectorizer_table.show()

                # Display correct chart.
                file_name = self.screen + " " + self.chart + " " + self.filter +
".png"
                self.chart_table.setPixmap(QPixmap(
                    join(CURRENT_PATH, "Videos", self.file_name,
                         "Charts", file_name)))
            else:  # If there is no data or this time range.
                self.chart_table.setText("No data for this time range!")
            self.chart_table.show()

    def hide_all(self):
        """Hide all GUI elements."""
        self.main_table.hide()
        self.vectorizer_table.hide()
        self.chart_table.hide()
        self.sort_by_box.hide()
        self.chart_box.hide()
        self.time_box.hide()

    def show_report_page(self):
        """Set current page to report page."""
        self.screen = "report"
        self.refresh_ui()

    def show_emotional_analysis(self):
        """Set current page to emotional analysis page."""
        self.screen = "emotions"
        self.refresh_ui()

    def show_comments_page(self):
        """Set current page to list of comments page."""
        self.screen = "comments"
        self.refresh_ui()

    def show_unigram_frequency_page(self):
        """Set current page to unigram frequency page."""
        self.screen = "frequency unigram"
        self.refresh_ui()

    def show_unigram_importance_page(self):
        """Set current page to unigram importance page."""
        self.screen = "importance unigram"
        self.refresh_ui()

    def show_bigram_frequency_page(self):
        """Set current page to bigram frequency page."""
```

```
416         self.screen = "frequency bigram"
417         self.refresh_ui()
418 
419     def show_bigram_importance_page(self):
420         """Set current page to bigram importance page."""
421         self.screen = "importance bigram"
422         self.refresh_ui()
423 
424     def show_trigram_frequency_page(self):
425         """Set current page to trigram frequency page."""
426         self.screen = "frequency trigram"
427         self.refresh_ui()
428 
429     def show_trigram_importance_page(self):
430         """Set current page to trigram importance page."""
431         self.screen = "importance trigram"
432         self.refresh_ui()
433 
434     def show_word_cloud(self):
435         """Set current image to word cloud."""
436         self.chart = "word cloud"
437         self.refresh_ui()
438 
439     def show_bar_chart(self):
440         """Set current image to bar chart."""
441         self.chart = "bar chart"
442         self.refresh_ui()
443 
444     def sort_by_negative(self):
445         """Set current sort option to most negative."""
446         self.sort_by = "negative"
447         self.refresh_ui()
448 
449     def sort_by_positive(self):
450         """Set current sort option to most positive."""
451         self.sort_by = "positive"
452         self.refresh_ui()
453 
454     def sort_by_latest(self):
455         """Set current sort option to most recent."""
456         self.sort_by = "latest"
457         self.refresh_ui()
458 
459     def sort_by_oldest(self):
460         """Set current sort option to oldest."""
461         self.sort_by = "oldest"
462         self.refresh_ui()
463 
464     def sort_by_most_subjective(self):
465         """Set current sort option to most subjective."""
466         self.sort_by = "most subjective"
467         self.refresh_ui()
468 
469     def sort_by_least_subjective(self):
470         """Set current sort option to least subjective."""
471         self.sort_by = "least subjective"
472         self.refresh_ui()
473 
474     def sort_buttons_controls(self):
475         """Sort comments by the current sort option selected."""
```

```
476         try:
477             if self.sort_by == "negative":
478                 self.comments = self.comments.sort_values(by="Polarity",
ascending=True)
479             elif self.sort_by == "positive":
480                 self.comments = self.comments.sort_values(by="Polarity",
ascending=False)
481             elif self.sort_by == "latest":
482                 self.comments = self.comments.sort_values(by="Time", ascending=False)
483             elif self.sort_by == "oldest":
484                 self.comments = self.comments.sort_values(by="Time", ascending=True)
485             elif self.sort_by == "most subjective":
486                 self.comments = self.comments.sort_values(by="Subjectivity",
ascending=False)
487             elif self.sort_by == "least subjective":
488                 self.comments = self.comments.sort_values(by="Subjectivity",
ascending=True)
489         except:
490             print("Sort Error.")
491             self.comments = self.comments_time_ranges[0]
492 
493     def filter_buttons_controls(self):
494         """Select the appropriate tables and list of comments to display.
495 
496         This is based on the currently selected radio buttons.
497         """
498         pageList = [
499             "comments", "emotions", "frequency unigram", "importance unigram",
500             "frequency bigram", "importance bigram", "frequency trigram",
501             "importance trigram"
502         ]
503         filterList = ["all", "30", "7", "48"]
504         tableList = [[[], [], [], []], self.emotional_analysis,
505                      self.frequencyListU, self.importanceListU,
506                      self.frequencyListB, self.importanceListB,
507                      self.frequencyListT, self.importanceListT]
508         for page in range(len(pageList)):
509             if self.screen == pageList[page]:
510                 for filter in range(len(filterList)):
511                     if self.filter == filterList[filter]:
512                         self.comments = self.comments_time_ranges[filter]
513                         self.table = tableList[page][filter]
514 
515     def filter_all_time(self):
516         """Set current filter to all time."""
517         self.filter = "all"
518         self.refresh_ui()
519 
520     def filter_thirty_days(self):
521         """Set current filter to 30 days."""
522         self.filter = "30"
523         self.refresh_ui()
524 
525     def filter_seven_days(self):
526         """Set current filter to 7 days."""
527         self.filter = "7"
528         self.refresh_ui()
529 
530     def filter_twenty_four_hours(self):
531         """Set current filter to 48 hours."""
```

```
        self.filter = "48"
        self.refresh_ui()

    def open_comments(self):
        """Open CSV file of comments."""
        try:
            startfile(join(CURRENT_PATH, "Videos", self.file_name,
"processed_all.csv"))
        except:
            print("Could not find Microsoft Excel installation!")

    def connect_menu_bar(self):
        """Connect menu bars actions to respective functions."""
        self.action_report.triggered.connect(self.show_report_page)
        self.action_emotional_analysis.triggered.connect(
            self.show_emotional_analysis)
        self.action_comments.triggered.connect(self.show_comments_page)
        self.action_unigram_frequency.triggered.connect(
            self.show_unigram_frequency_page)
        self.action_unigram_importance.triggered.connect(
            self.show_unigram_importance_page)
        self.action_bigram_frequency.triggered.connect(
            self.show_bigram_frequency_page)
        self.action_bigram_importance.triggered.connect(
            self.show_bigram_importance_page)
        self.action_trigram_frequency.triggered.connect(
            self.show_trigram_frequency_page)
        self.action_trigram_importance.triggered.connect(
            self.show_trigram_importance_page)

    def import_data(self, file_name, report, comments_time_ranges,
                    emotional_analysis, vectorizer_data):
        """Import the data used in the application.

        Args:
            file_name (str): Title of YouTube video (and file name).
            report (DataFrame): Report of average statistics about the comments.
            comments_time_ranges (list of DataFrames): List of subsets of processed
comments by time range.
            emotional_analysis (list of DataFrames): List of emotional analysis
subsets by time range.
            vectorizer_data (list of lists of DataFrames): Results of count and TF-
IDF vectorization by time range.
        """
        self.file_name = file_name
        self.comments_time_ranges = comments_time_ranges
        self.report = report
        self.emotional_analysis = emotional_analysis
        self.frequencyListU, self.importanceListU, self.frequencyListB,
self.importanceListB, self.frequencyListT, self.importanceListT = [
            [], [], [], []], [[], [], [], []], [[], [], [], []], [[], [], [], []],
[[], [], [], []], [[], [], [], []]
        ]
        list_name = [
            self.frequencyListU, self.importanceListU, self.frequencyListB,
            self.importanceListB, self.frequencyListT, self.importanceListT
        ]

        # Assigns each variable with the correct data n-gram vectorizer data.
        for type in range(len(list_name)):
```

```
            for time_range in range(0, 4):
                vectorizer_data[time_range][type] = vectorizer_data[time_range][
                    type].apply(lambda x: round(x, 2))
                for index, value in vectorizer_data[time_range][type].items():
                    list_name[type][time_range].append([index, value])

    def setup_ui(self, main_window):
        """Sets up the GUI.

        Args:
            menu_window (PyQt5.QtWidgets.QMainWindow): Main window object.
        """
        main_window.setObjectName("main_window")
        main_window.resize(1280, 1000)

        # Uncomment to disable window maximization.
        # main_window.setWindowFlags(Qt.WindowCloseButtonHint |
Qt.WindowMinimizeButtonHint)

        self.central_widget = QWidget(main_window)
        self.central_widget.setObjectName("central_widget")

        # Stylesheets.
        stylesheets_path = join(CURRENT_PATH, "StyleSheets")
        self.report_table_qss = open(
            join(stylesheets_path, "ReportTable.qss"), "r").read()
        self.comments_table_qss = open(
            join(stylesheets_path, "CommentsTable.qss"), "r").read()
        self.vectorizer_table_qss = open(
            join(stylesheets_path, "VectorizerTable.qss"), "r").read()
        self.group_box_qss = open(
            join(stylesheets_path, "GroupBox.qss"), "r").read()

        # Video title object.
        self.title_label = QLabel(self.central_widget)
        self.title_label.setGeometry(QRect(20, 0, 661, 31))
        font = QFont()  # Font object used to set text properties.
        font.setPointSize(15)
        self.title_label.setFont(font)
        self.title_label.setObjectName("title_label")
        title = self.file_name
        # If the title is longer than 38 characters, shorten it.
        if len(title) > 38:
            title = title[:38] + "..."
        self.title_label.setText(
            QCoreApplication.translate(
                "main_window", "Video Title: " + title + " Comments: " +
                str(self.comments_time_ranges[0]["Comment"].count().item())))
        self.title_label.show()

        # Tables.
        # Report/Comments table object.
        self.main_table = QTableView(self.central_widget)
        self.main_table.setGeometry(QRect(20, 70, 1241, 881))
        self.main_table.setObjectName("main_table")

        # n-gram frequency/importance table object.
        self.vectorizer_table = QTableView(self.central_widget)
        self.vectorizer_table.setGeometry(QRect(20, 72, 271, 881))
        self.vectorizer_table.setObjectName("vectorizer_table")
```

```
        self.vectorizer_table.setStyleSheet(self.vectorizer_table_qss)

        # Bar chart/word cloud image object.
        self.chart_table = QLabel(self.central_widget)
        self.chart_table.setGeometry(QRect(296, 72, 961, 881))
        self.chart_table.setObjectName("chart_table")
        self.chart_table.setScaledContents(True)
        font.setPointSize(30)
        self.chart_table.setFont(font)

        # Button boxes.
        # Sort by button box object.
        self.sort_by_box = QGroupBox(self.central_widget)
        self.sort_by_box.setGeometry(QRect(685, -10, 330, 75))
        self.sort_by_box.setObjectName("sort_by_box")
        self.sort_by_box.setStyleSheet(self.group_box_qss)
        sort_by_widget = QWidget(self.sort_by_box)
        sort_by_widget.setGeometry(QRect(50, 30, 270, 40))
        sort_by_widget.setObjectName("widget")

        # Bar chart/word cloud option button box object.
        self.chart_box = QGroupBox(self.central_widget)
        self.chart_box.setGeometry(QRect(425, 10, 260, 55))
        self.chart_box.setObjectName("chart_box")
        # self.chart_box.setStyleSheet("font-size: 13px;")
        # self.chart_box.setStyleSheet("border: 10px white;")
        self.chart_box.setStyleSheet(self.group_box_qss)
        self.chart_widget = QWidget(self.chart_box)
        self.chart_widget.setGeometry(QRect(65, 30, 191, 15))
        self.chart_widget.setObjectName("horizontalLayoutWidget")

        # Time range filter box object.
        self.time_box = QGroupBox(self.central_widget)
        self.time_box.setGeometry(QRect(10, 10, 390, 55))
        self.time_box.setObjectName("time_box")
        self.time_box.setStyleSheet(self.group_box_qss)
        self.time_widget = QWidget(self.time_box)
        self.time_widget.setGeometry(QRect(65, 30, 320, 15))
        self.time_widget.setObjectName("horizontalLayoutWidget_2")

        # Buttons.
        # Button to open CSV file object.
        self.open_csv_button = QPushButton(self.central_widget)
        self.open_csv_button.setGeometry(QRect(1080, 10, 180, 55))
        self.open_csv_button.setObjectName("open_csv_button")
        self.open_csv_button.clicked.connect(self.open_comments)

        # Word cloud button object.
        self.word_cloud_button = QRadioButton(self.chart_widget)
        self.word_cloud_button.setGeometry(QRect(550, 40, 82, 17))
        self.word_cloud_button.setObjectName("word_cloud_button")
        self.word_cloud_button.clicked.connect(self.show_word_cloud)

        # Bar chart button object.
        self.bar_chart_button = QRadioButton(self.chart_widget)
        self.bar_chart_button.setGeometry(QRect(460, 40, 82, 17))
        self.bar_chart_button.setObjectName("bar_chart_button")
        self.bar_chart_button.clicked.connect(self.show_bar_chart)
        self.bar_chart_button.setChecked(True)

```

```
        # Filter by last 7 days button object.
        self.filter_7_button = QRadioButton(self.time_widget)
        self.filter_7_button.setGeometry(QRect(160, 40, 82, 17))
        self.filter_7_button.setObjectName("filter_7_button")
        self.filter_7_button.clicked.connect(self.filter_seven_days)

        # Filter by last 48 hours button object.
        self.filter_48_button = QRadioButton(self.time_widget)
        self.filter_48_button.setGeometry(QRect(220, 40, 82, 17))
        self.filter_48_button.setObjectName("filter_48_button")
        self.filter_48_button.clicked.connect(self.filter_twenty_four_hours)

        # Filter by last 30 days button object.
        self.filter_30_button = QRadioButton(self.time_widget)
        self.filter_30_button.setGeometry(QRect(90, 40, 82, 17))
        self.filter_30_button.setObjectName("filter_30_button")
        self.filter_30_button.clicked.connect(self.filter_thirty_days)

        # Filter by all time button object.
        self.filter_all_button = QRadioButton(self.time_widget)
        self.filter_all_button.setGeometry(QRect(20, 40, 82, 17))
        self.filter_all_button.setObjectName("filter_all_button")
        self.filter_all_button.clicked.connect(self.filter_all_time)
        self.filter_all_button.setChecked(True)

        # Sort by most negative comments button object.
        self.sort_by_negative_button = QRadioButton(sort_by_widget)
        self.sort_by_negative_button.setGeometry(QRect(810, 20, 131, 17))
        self.sort_by_negative_button.setObjectName("sort_by_negative_button")
        self.sort_by_negative_button.clicked.connect(self.sort_by_negative)

        # Sort by most positive comments button object.
        self.sort_by_positive_button = QRadioButton(sort_by_widget)
        self.sort_by_positive_button.setGeometry(QRect(650, 20, 121, 17))
        self.sort_by_positive_button.setObjectName("sort_by_positive_button")
        self.sort_by_positive_button.clicked.connect(self.sort_by_positive)
        self.sort_by_positive_button.setChecked(True)

        # Sort by latest comments button object.
        self.sort_by_latest_button = QRadioButton(sort_by_widget)
        self.sort_by_latest_button.setGeometry(QRect(980, 20, 91, 17))
        self.sort_by_latest_button.setObjectName("sort_by_latest_button")
        self.sort_by_latest_button.clicked.connect(self.sort_by_latest)

        # Sort by oldest comments button object.
        self.sort_by_oldest_button = QRadioButton(sort_by_widget)
        self.sort_by_oldest_button.setGeometry(QRect(980, 40, 91, 17))
        self.sort_by_oldest_button.setObjectName("sort_by_oldest_button")
        self.sort_by_oldest_button.clicked.connect(self.sort_by_oldest)

        # Sort by most subjective comments button object.
        self.sort_by_most_subjective_button = QRadioButton(sort_by_widget)
        self.sort_by_most_subjective_button.setGeometry(QRect(650, 40, 141, 17))
        self.sort_by_most_subjective_button.setObjectName(
            "sort_by_most_subjective")
        self.sort_by_most_subjective_button.clicked.connect(
            self.sort_by_most_subjective)

        # Sort by most objective comments button object.
        self.sort_by_most_objective_button = QRadioButton(sort_by_widget)
```

```
765         self.sort_by_most_objective_button.setGeometry(QRect(810, 40, 141, 17))
766         self.sort_by_most_objective_button.setObjectName(
767             "sort_by_least_subjective")
768         self.sort_by_most_objective_button.clicked.connect(
769             self.sort_by_least_subjective)
770 
771         # Layouts.
772         # Sort by layout object.
773         self.sort_by_layout = QGridLayout(sort_by_widget)
774         self.sort_by_layout.setContentsMargins(0, 0, 0, 0)
775         self.sort_by_layout.setObjectName("sort_by_layout")
776         self.sort_by_layout.addWidget(self.sort_by_positive_button, 0, 0, 1, 1)
777         self.sort_by_layout.addWidget(self.sort_by_most_subjective_button, 0, 1,
778                                       1, 1)
779         self.sort_by_layout.addWidget(self.sort_by_latest_button, 0, 2, 1, 1)
780         self.sort_by_layout.addWidget(self.sort_by_negative_button, 1, 0, 1, 1)
781         self.sort_by_layout.addWidget(self.sort_by_most_objective_button, 1, 1,
782                                       1, 1)
783         self.sort_by_layout.addWidget(self.sort_by_oldest_button, 1, 2, 1, 1)
784 
785         # Box chart/word cloud layout object.
786         self.chart_layout = QHBoxLayout(self.chart_widget)
787         self.chart_layout.setContentsMargins(0, 0, 0, 0)
788         self.chart_layout.setObjectName("chart_layout")
789         self.chart_layout.addWidget(self.bar_chart_button)
790         self.chart_layout.addWidget(self.word_cloud_button)
791 
792         # Time range filter layout object.
793         self.time_layout = QHBoxLayout(self.time_widget)
794         self.time_layout.setContentsMargins(0, 0, 0, 0)
795         self.time_layout.setObjectName("time_layout")
796         self.time_layout.addWidget(self.filter_all_button)
797         self.time_layout.addWidget(self.filter_30_button)
798         self.time_layout.addWidget(self.filter_7_button)
799         self.time_layout.addWidget(self.filter_48_button)
800 
801         # "Sort By" text object.
802         self.sort_by_label = QLabel(self.sort_by_box)
803         self.sort_by_label.setGeometry(QRect(10, 20, 61, 51))
804         self.sort_by_label.setFont(font)
805         self.sort_by_label.setTextFormat(Qt.PlainText)
806         self.sort_by_label.setObjectName("sort_by_label")
807 
808         # "Chart" text object.
809         self.chart_label = QLabel(self.chart_box)
810         self.chart_label.setGeometry(QRect(10, 20, 61, 31))
811         self.chart_label.setFont(font)
812         self.chart_label.setTextFormat(Qt.PlainText)
813         self.chart_label.setObjectName("chart_label")
814 
815         # "Time Range" text object.
816         self.time_label = QLabel(self.time_box)
817         self.time_label.setGeometry(QRect(10, 10, 61, 51))
818         self.time_label.setFont(font)
819         self.time_label.setTextFormat(Qt.PlainText)
820         self.time_label.setObjectName("time_label")
821 
822         # Menu bar.
823         # Menu bar action objects.
824         self.action_report = QAction(main_window)
```

```
        self.action_report.setObjectName("action_report")
        self.action_emotional_analysis = QAction(main_window)
        self.action_emotional_analysis.setObjectName(
            "action_emotional_analysis")
        self.action_comments = QAction(main_window)
        self.action_comments.setObjectName("action_comments")
        self.action_unigram_frequency = QAction(main_window)
        self.action_unigram_frequency.setObjectName("action_unigram_frequency")
        self.action_unigram_importance = QAction(main_window)
        self.action_unigram_importance.setObjectName(
            "action_unigram_importance")
        self.action_bigram_frequency = QAction(main_window)
        self.action_bigram_frequency.setObjectName("action_bigram_frequency")
        self.action_bigram_importance = QAction(main_window)
        self.action_bigram_importance.setObjectName("action_bigram_importance")
        self.action_trigram_frequency = QAction(main_window)
        self.action_trigram_frequency.setObjectName("action_trigram_frequency")
        self.action_trigram_importance = QAction(main_window)
        self.action_trigram_importance.setObjectName(
            "action_trigram_importance")

        # Menu bar object configuration.
        main_window.setCentralWidget(self.central_widget)
        self.menu_bar = QMenuBar(main_window)
        self.menu_bar.setGeometry(QRect(0, 0, 1280, 21))
        self.menu_bar.setObjectName("menu_bar")
        self.menu_data = QMenu(self.menu_bar)
        self.menu_data.setObjectName("menu_data")
        main_window.setMenuBar(self.menu_bar)
        self.status_bar = QStatusBar(main_window)
        self.status_bar.setObjectName("status_bar")
        main_window.setStatusBar(self.status_bar)

        # Add menu bar actions to menu bar object.
        self.menu_data.addAction(self.action_report)
        self.menu_data.addAction(self.action_emotional_analysis)
        self.menu_data.addAction(self.action_comments)
        self.menu_data.addSeparator()
        self.menu_data.addAction(self.action_unigram_frequency)
        self.menu_data.addAction(self.action_unigram_importance)
        self.menu_data.addSeparator()
        self.menu_data.addAction(self.action_bigram_frequency)
        self.menu_data.addAction(self.action_bigram_importance)
        self.menu_data.addSeparator()
        self.menu_data.addAction(self.action_trigram_frequency)
        self.menu_data.addAction(self.action_trigram_importance)
        self.menu_data.addSeparator()
        self.menu_bar.addAction(self.menu_data.menuAction())
        self.connect_menu_bar()

        self.show_report_page()

        self.retranslate_ui(main_window)
        QMetaObject.connectSlotsByName(main_window)

        # Move window to centre of screen.
        qt_rectangle = main_window.frameGeometry()
        qt_rectangle.moveCenter(QDesktopWidget().availableGeometry().center())
        main_window.move(qt_rectangle.topLeft())

```

```
885     def retranslate_ui(self, main_window):
886         """Set text on all objects."""
887         translate = QCoreApplication.translate
888         main_window.setWindowTitle(translate("main_window", "Main Window"))
889         self.word_cloud_button.setText(translate("main_window", "Word Cloud"))
890         self.bar_chart_button.setText(translate("main_window", "Bar Chart"))
891         self.sort_by_negative_button.setText(
892             translate("main_window", "Most Negative"))
893         self.sort_by_positive_button.setText(
894             translate("main_window", "Most Positive"))
895         self.sort_by_latest_button.setText(translate("main_window", "Latest"))
896         self.sort_by_oldest_button.setText(translate("main_window", "Oldest"))
897         self.sort_by_most_subjective_button.setText(
898             translate("main_window", "Most Subjective"))
899         self.sort_by_most_objective_button.setText(
900             translate("main_window", "Most Objective"))
901         self.open_csv_button.setText(
902             translate("main_window", "Open Comments as CSV File"))
903         self.filter_7_button.setText(translate("main_window", "7 Days"))
904         self.filter_48_button.setText(translate("main_window", "48 Hours"))
905         self.filter_30_button.setText(translate("main_window", "30 Days"))
906         self.filter_all_button.setText(translate("main_window", "All Time"))
907         self.menu_data.setTitle(translate("main_window", "Select Data View"))
908         self.action_report.setText(translate("main_window", "Report"))
909         self.action_emotional_analysis.setText(
910             translate("MainWindow", "Emotional Analysis"))
911         self.action_unigram_frequency.setText(
912             translate("main_window", "Unigram Frequency"))
913         self.action_unigram_importance.setText(
914             translate("main_window", "Unigram Importance"))
915         self.action_comments.setText(
916             translate("main_window", "List of Comments"))
917         self.action_bigram_frequency.setText(
918             translate("main_window", "Bigram Frequency"))
919         self.action_bigram_importance.setText(
920             translate("main_window", "Bigram Importance"))
921         self.action_trigram_frequency.setText(
922             translate("main_window", "Trigram Frequency"))
923         self.action_trigram_importance.setText(
924             translate("main_window", "Trigram Importance"))
925         self.sort_by_label.setText(translate("MainWindow", "Sort\nBy"))
926         self.chart_label.setText(translate("MainWindow", "Chart"))
927         self.time_label.setText(translate("MainWindow", "Time\nRange"))
928 
```

### C.1.3 contractions.py

```python
# contractions.py: Dictionary of (unexpanded contraction -> expanded contraction) pairs.
#
# Taken and adapted from:
# https://gist.github.com/nealrs/96342d8231b75cf4bb82

contractions = {
    "aint": "am not",
    "arent": "are not",
    "cant": "cannot",
    "cantve": "cannot have",
    "cause": "because",
    "couldve": "could have",
    "couldnt": "could not",
    "couldntve": "could not have",
    "didnt": "did not",
    "doesnt": "does not",
    "dont": "do not",
    "hadnt": "had not",
    "hadntve": "had not have",
    "hasnt": "has not",
    "havent": "have not",
    "hed": "he would",
    "hedve": "he would have",
    "hell": "he will",
    "hellve": "he will have",
    "hes": "he is",
    "howd": "how did",
    "howdy": "how do you",
    "howll": "how will",
    "hows": "how is",
    "id": "i would",
    "idve": "i would have",
    "ill": "i will",
    "illve": "i will have",
    "im": "i am",
    "ive": "i have",
    "isnt": "is not",
    "itd": "it had",
    "itdve": "it would have",
    "itll": "it will",
    "itllve": "it will have",
    "its": "it is",
    "lets": "let us",
    "maam": "madam",
    "maynt": "may not",
    "mightve": "might have",
    "mightnt": "might not",
    "mightntve": "might not have",
    "mustve": "must have",
    "mustnt": "must not",
    "mustntve": "must not have",
    "neednt": "need not",
    "needntve": "need not have",
    "oclock": "of the clock",
    "oughtnt": "ought not",
    "oughtntve": "ought not have",
    "shant": "shall not",
    "shant": "shall not",
```

```
    "shantve": "shall not have",
    "shed": "she would",
    "shedve": "she would have",
    "shell": "she will",
    "shellve": "she will have",
    "shes": "she is",
    "shouldve": "should have",
    "shouldnt": "should not",
    "shouldntve": "should not have",
    "sove": "so have",
    "sos": "so is",
    "thatd": "that would",
    "thatdve": "that would have",
    "thats": "that is",
    "thered": "there had",
    "theredve": "there would have",
    "theres": "there is",
    "theyd": "they would",
    "theydve": "they would have",
    "theyll": "they will",
    "theyllve": "they will have",
    "theyre": "they are",
    "theyve": "they have",
    "tove": "to have",
    "wasnt": "was not",
    "wed": "we had",
    "wedve": "we would have",
    "well": "we will",
    "wellve": "we will have",
    "were": "we are",
    "weve": "we have",
    "werent": "were not",
    "whatll": "what will",
    "whatllve": "what will have",
    "whatre": "what are",
    "whats": "what is",
    "whatve": "what have",
    "whens": "when is",
    "whenve": "when have",
    "whered": "where did",
    "wheres": "where is",
    "whereve": "where have",
    "wholl": "who will",
    "whollve": "who will have",
    "whos": "who is",
    "whove": "who have",
    "whys": "why is",
    "whyve": "why have",
    "willve": "will have",
    "wont": "will not",
    "wontve": "will not have",
    "wouldve": "would have",
    "wouldnt": "would not",
    "wouldntve": "would not have",
    "yall": "you all",
    "yalls": "you alls",
    "yalld": "you all would",
    "yalldve": "you all would have",
    "yallre": "you all are",
    "yallve": "you all have",
```

```
    "youd": "you had",
    "youdve": "you would have",
    "youll": "you you will",
    "youllve": "you you will have",
    "youre": "you are",
    "youve": "you have"
}

```

## C.2 StyleSheets

### C.2.1 Combinear.qss

```
/*Copyright (c) DevSec Studio. All rights reserved.

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED *AS IS*, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

/*-----QWidget-----*/
QWidget
{
  background-color: #3a3a3a;
  color: #fff;
  selection-background-color: #b78620;
  selection-color: #000;

}


/*-----QLabel-----*/
QLabel
{
  background-color: transparent;
  color: #fff;

}


/*-----QMenuBar-----*/
QMenuBar
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
rgba(57, 57, 57, 255),stop:1 rgba(50, 50, 50, 255));
  border: 1px solid #000;
  color: #fff;

}


QMenuBar::item
{
  background-color: transparent;

}
```

```
58
59
60 QMenuBar::item:selected
61 {
62   background-color: rgba(183, 134, 32, 20%);
63   border: 1px solid #b78620;
64   color: #fff;
65
66 }
67
68
69 QMenuBar::item:pressed
70 {
71   background-color: rgb(183, 134, 32);
72   border: 1px solid #b78620;
73   color: #fff;
74
75 }
76
77
78 /*-----QMenu-----*/
79 QMenu
80 {
81     background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
   rgba(57, 57, 57, 255),stop:1 rgba(50, 50, 50, 255));
82     border: 1px solid #222;
83     padding: 4px;
84   color: #fff;
85
86 }
87
88
89 QMenu::item
90 {
91     background-color: transparent;
92     padding: 2px 20px 2px 20px;
93
94 }
95
96
97 QMenu::separator
98 {
99     background-color: rgb(183, 134, 32);
100   height: 1px;
101
102 }
103
104
105 QMenu::item:disabled
106 {
107     color: #555;
108     background-color: transparent;
109     padding: 2px 20px 2px 20px;
110
111 }
112
113
114 QMenu::item:selected
115 {
116   background-color: rgba(183, 134, 32, 20%);
```

```
  border: 1px solid #b78620;
  color: #fff;

}


/*-----QToolBar-----*/
QToolBar
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(69, 69, 69, 255),stop:1 rgba(58, 58, 58, 255));
  border-top: none;
  border-bottom: 1px solid #4f4f4f;
  border-left: 1px solid #4f4f4f;
  border-right: 1px solid #4f4f4f;

}


QToolBar::separator
{
  background-color: #2e2e2e;
  width: 1px;

}


/*-----QToolButton-----*/
QToolButton
{
  background-color: transparent;
  color: #fff;
  padding: 5px;
  padding-left: 8px;
  padding-right: 8px;
  margin-left: 1px;
}


QToolButton:hover
{
  background-color: rgba(183, 134, 32, 20%);
  border: 1px solid #b78620;
  color: #fff;

}


QToolButton:pressed
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(57, 57, 57, 255),stop:1 rgba(50, 50, 50, 255));
  border: 1px solid #b78620;

}


QToolButton:checked
{
```

```
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(57, 57, 57, 255),stop:1 rgba(50, 50, 50, 255));
  border: 1px solid #222;
}


/*-----QPushButton-----*/
QPushButton
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(84, 84, 84, 255),stop:1 rgba(59, 59, 59, 255));
  color: #ffffff;
  min-width: 80px;
  border-style: solid;
  border-width: 1px;
  border-radius: 3px;
  border-color: #051a39;
  padding: 5px;

}


QPushButton::flat
{
  background-color: transparent;
  border: none;
  color: #fff;

}


QPushButton::disabled
{
  background-color: #404040;
  color: #656565;
  border-color: #051a39;

}


QPushButton::hover
{
  background-color: rgba(183, 134, 32, 20%);
  border: 1px solid #b78620;

}


QPushButton::pressed
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(74, 74, 74, 255),stop:1 rgba(49, 49, 49, 255));
  border: 1px solid #b78620;

}


QPushButton::checked
{
```

```
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(74, 74, 74, 255),stop:1 rgba(49, 49, 49, 255));
  border: 1px solid #222;

}


/*-----QLineEdit-----*/
QLineEdit
{
  background-color: #131313;
  color : #eee;
  border: 1px solid #343434;
  border-radius: 2px;
  padding: 3px;
  padding-left: 5px;

}


/*-----QPlainTExtEdit-----*/
QPlainTextEdit
{
  background-color: #131313;
  color : #eee;
  border: 1px solid #343434;
  border-radius: 2px;
  padding: 3px;
  padding-left: 5px;

}


/*-----QTabBar-----*/
QTabBar::tab
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(84, 84, 84, 255),stop:1 rgba(59, 59, 59, 255));
  color: #ffffff;
  border-style: solid;
  border-width: 1px;
  border-color: #666;
  border-bottom: none;
  padding: 5px;
  padding-left: 15px;
  padding-right: 15px;

}


QTabWidget::pane
{
  background-color: red;
  border: 1px solid #666;
  top: 1px;

}


QTabBar::tab:last
```

```
{
  margin-right: 0;

}


QTabBar::tab:first:!selected
{
  background-color: #0c0c0d;
  margin-left: 0px;

}


QTabBar::tab:!selected
{
  color: #b1b1b1;
  border-bottom-style: solid;
  background-color: #0c0c0d;

}


QTabBar::tab:selected
{
  margin-bottom: 0px;

}


QTabBar::tab:!selected:hover
{
  border-top-color: #b78620;

}


/*-----QComboBox-----*/
QComboBox
{
    background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
rgba(84, 84, 84, 255),stop:1 rgba(59, 59, 59, 255));
    border: 1px solid #000;
    padding-left: 6px;
    color: #ffffff;
    height: 20px;

}


QComboBox::disabled
{
  background-color: #404040;
  color: #656565;
  border-color: #051a39;

}


QComboBox:on
```

```
{
    background-color: #b78620;
  color: #000;

}


QComboBox QAbstractItemView
{
    background-color: #383838;
    color: #ffffff;
    border: 1px solid black;
    selection-background-color: #b78620;
    outline: 0;

}


QComboBox::drop-down
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
rgba(57, 57, 57, 255),stop:1 rgba(50, 50, 50, 255));
    subcontrol-origin: padding;
    subcontrol-position: top right;
    width: 15px;
    border-left-width: 1px;
    border-left-color: black;
    border-left-style: solid;

}


QComboBox::down-arrow
{
    image: url(://arrow-down.png);
    width: 8px;
    height: 8px;
}


/*-----QSpinBox & QDateTimeEdit-----*/
QSpinBox,
QDateTimeEdit
{
    background-color: #131313;
  color : #eee;
  border: 1px solid #343434;
  padding: 3px;
  padding-left: 5px;
    border-radius : 2px;

}


QSpinBox::up-button,
QDateTimeEdit::up-button
{
  border-top-right-radius:2px;
  background-color: #777777;
    width: 16px;
```

```
    border-width: 1px;

}


QSpinBox::up-button:hover,
QDateTimeEdit::up-button:hover
{
  background-color: #585858;

}


QSpinBox::up-button:pressed,
QDateTimeEdit::up-button:pressed
{
  background-color: #252525;
    width: 16px;
    border-width: 1px;

}


QSpinBox::up-arrow,
QDateTimeEdit::up-arrow
{
    image: url(://arrow-up.png);
    width: 7px;
    height: 7px;

}


QSpinBox::down-button,
QDateTimeEdit::down-button
{
  border-bottom-right-radius:2px;
  background-color: #777777;
    width: 16px;
    border-width: 1px;

}


QSpinBox::down-button:hover,
QDateTimeEdit::down-button:hover
{
  background-color: #585858;

}


QSpinBox::down-button:pressed,
QDateTimeEdit::down-button:pressed
{
  background-color: #252525;
    width: 16px;
    border-width: 1px;

}
```

```
466
467
468 QSpinBox::down-arrow,
469 QDateTimeEdit::down-arrow
470 {
471     image: url(://arrow-down.png);
472     width: 7px;
473     height: 7px;
474
475 }
476
477
478 /*-----QGroupBox-----*/
479 QGroupBox
480 {
481     border: 1px solid;
482     border-color: #666666;
483   border-radius: 5px;
484     margin-top: 20px;
485
486 }
487
488
489 QGroupBox::title
490 {
491     background-color: transparent;
492     color: #eee;
493     subcontrol-origin: margin;
494     padding: 5px;
495   border-top-left-radius: 3px;
496   border-top-right-radius: 3px;
497
498 }
499
500
501 /*-----QHeaderView-----*/
502 QHeaderView::section
503 {
504     background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
    rgba(60, 60, 60, 255),stop:1 rgba(50, 50, 50, 255));
505   border: 1px solid #000;
506     color: #fff;
507     text-align: left;
508   padding: 4px;
509
510 }
511
512
513 QHeaderView::section:disabled
514 {
515     background-color: #525251;
516     color: #656565;
517
518 }
519
520
521 QHeaderView::section:checked
522 {
523     background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
    rgba(60, 60, 60, 255),stop:1 rgba(50, 50, 50, 255));
```

```
    color: #fff;

}


QHeaderView::section::vertical::first,
QHeaderView::section::vertical::only-one
{
    border-top: 1px solid #353635;

}


QHeaderView::section::vertical
{
    border-top: 1px solid #353635;

}


QHeaderView::section::horizontal::first,
QHeaderView::section::horizontal::only-one
{
    border-left: 1px solid #353635;

}


QHeaderView::section::horizontal
{
    border-left: 1px solid #353635;

}


QTableCornerButton::section
{
    background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(60, 60, 60, 255),stop:1 rgba(50, 50, 50, 255));
  border: 1px solid #000;
    color: #fff;

}


/*-----QTreeWidget-----*/
QTreeView
{
  show-decoration-selected: 1;
  alternate-background-color: #3a3a3a;
  selection-color: #fff;
  background-color: #2d2d2d;
  border: 1px solid gray;
  padding-top : 5px;
  color: #fff;
  font: 8pt;

}


```

```
QTreeView::item:selected
{
  color:#fff;
  background-color: #b78620;
  border-radius: 0px;

}


QTreeView::item:!selected:hover
{
    background-color: #262626;
    border: none;
    color: white;

}


QTreeView::branch:has-children:!has-siblings:closed,
QTreeView::branch:closed:has-children:has-siblings
{
  image: url(://tree-closed.png);

}


QTreeView::branch:open:has-children:!has-siblings,
QTreeView::branch:open:has-children:has-siblings
{
  image: url(://tree-open.png);

}


/*-----QListView-----*/
QListView
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(83, 83, 83, 255),stop:0.293269 rgba(81, 81, 81, 255),stop:0.634615 rgba(79, 79, 79, 255),stop:1 rgba(83, 83, 83, 255));
    border : none;
    color: white;
    show-decoration-selected: 1;
    outline: 0;
  border: 1px solid gray;

}


QListView::disabled
{
  background-color: #656565;
  color: #1b1b1b;
    border: 1px solid #656565;

}


QListView::item
{
```

```
  background-color: #2d2d2d;
    padding: 1px;

}


QListView::item:alternate
{
    background-color: #3a3a3a;

}


QListView::item:selected
{
  background-color: #b78620;
  border: 1px solid #b78620;
  color: #fff;

}


QListView::item:selected:!active
{
  background-color: #b78620;
  border: 1px solid #b78620;
  color: #fff;

}


QListView::item:selected:active
{
  background-color: #b78620;
  border: 1px solid #b78620;
  color: #fff;

}


QListView::item:hover {
    background-color: #262626;
    border: none;
    color: white;

}


/*-----QCheckBox-----*/
QCheckBox
{
  background-color: transparent;
    color: lightgray;
  border: none;

}


QCheckBox::indicator
{
```

```
    background-color: #323232;
    border: 1px solid darkgray;
    width: 12px;
    height: 12px;

}


QCheckBox::indicator:checked
{
    image:url("./ressources/check.png");
  background-color: #b78620;
    border: 1px solid #3a546e;

}


QCheckBox::indicator:unchecked:hover
{
  border: 1px solid #b78620;

}


QCheckBox::disabled
{
  color: #656565;

}


QCheckBox::indicator:disabled
{
  background-color: #656565;
  color: #656565;
    border: 1px solid #656565;

}


/*-----QRadioButton-----*/
QRadioButton
{
  color: lightgray;
  background-color: transparent;

}


QRadioButton::indicator::unchecked:hover
{
  background-color: lightgray;
  border: 2px solid #b78620;
  border-radius: 6px;
}


QRadioButton::indicator::checked
{
  border: 2px solid #b78620;
```

```
  border-radius: 6px;
  background-color: rgba(183,134,32,20%);
  width: 9px;
  height: 9px;

}


/*-----QSlider-----*/
QSlider::groove:horizontal
{
  background-color: transparent;
  height: 3px;

}


QSlider::sub-page:horizontal
{
  background-color: #b78620;

}


QSlider::add-page:horizontal
{
  background-color: #131313;

}


QSlider::handle:horizontal
{
  background-color: #b78620;
  width: 14px;
  margin-top: -6px;
  margin-bottom: -6px;
  border-radius: 6px;

}


QSlider::handle:horizontal:hover
{
  background-color: #d89e25;
  border-radius: 6px;

}


QSlider::sub-page:horizontal:disabled
{
  background-color: #bbb;
  border-color: #999;

}


QSlider::add-page:horizontal:disabled
{
```

```
  background-color: #eee;
  border-color: #999;

}


QSlider::handle:horizontal:disabled
{
  background-color: #eee;
  border: 1px solid #aaa;
  border-radius: 3px;

}


/*-----QScrollBar-----*/
QScrollBar:horizontal
{
    border: 1px solid #222222;
    background-color: #3d3d3d;
    height: 15px;
    margin: 0px 16px 0 16px;

}


QScrollBar::handle:horizontal
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(97, 97, 97, 255),stop:1 rgba(90, 90, 90, 255));
  border: 1px solid #2d2d2d;
    min-height: 20px;

}


QScrollBar::add-line:horizontal
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(97, 97, 97, 255),stop:1 rgba(90, 90, 90, 255));
  border: 1px solid #2d2d2d;
    width: 15px;
    subcontrol-position: right;
    subcontrol-origin: margin;

}


QScrollBar::sub-line:horizontal
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(97, 97, 97, 255),stop:1 rgba(90, 90, 90, 255));
  border: 1px solid #2d2d2d;
    width: 15px;
    subcontrol-position: left;
    subcontrol-origin: margin;

}


```

```
QScrollBar::right-arrow:horizontal
{
    image: url(://arrow-right.png);
    width: 6px;
    height: 6px;

}


QScrollBar::left-arrow:horizontal
{
    image: url(://arrow-left.png);
    width: 6px;
    height: 6px;

}


QScrollBar::add-page:horizontal, QScrollBar::sub-page:horizontal
{
    background: none;

}


QScrollBar:vertical
{
    background-color: #3d3d3d;
    width: 16px;
  border: 1px solid #2d2d2d;
    margin: 16px 0px 16px 0px;

}


QScrollBar::handle:vertical
{
    background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
rgba(97, 97, 97, 255),stop:1 rgba(90, 90, 90, 255));
  border: 1px solid #2d2d2d;
    min-height: 20px;

}


QScrollBar::add-line:vertical
{
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0
rgba(97, 97, 97, 255),stop:1 rgba(90, 90, 90, 255));
  border: 1px solid #2d2d2d;
    height: 15px;
    subcontrol-position: bottom;
    subcontrol-origin: margin;

}


QScrollBar::sub-line:vertical
{
```

```
  background-color: qlineargradient(spread:repeat, x1:1, y1:0, x2:1, y2:1, stop:0 rgba(97, 97, 97, 255),stop:1 rgba(90, 90, 90, 255));
  border: 1px solid #2d2d2d;
    height: 15px;
    subcontrol-position: top;
    subcontrol-origin: margin;

}


QScrollBar::up-arrow:vertical
{
    image: url(://arrow-up.png);
    width: 6px;
    height: 6px;

}


QScrollBar::down-arrow:vertical
{
    image: url(://arrow-down.png);
    width: 6px;
    height: 6px;

}


QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical
{
    background: none;

}


/*-----QProgressBar-----*/
QProgressBar
{
    border: 1px solid #666666;
    text-align: center;
  color: #000;
  font-weight: bold;

}


QProgressBar::chunk
{
    background-color: #b78620;
    width: 30px;
    margin: 0.5px;

}


```

### C.2.2 CommentsTable.qss

```
QTableView
{
  gridline-color: #c7c7c7;
    font: 10pt;

}

QHeaderView
{
    font: 10pt;

}
```

### C.2.3 ReportTable.qss

```
QTableView
{
  gridline-color: #c7c7c7;
    font: 28pt;

}

QHeaderView
{
    font: 28pt;

}
```

### C.2.4 VectorizerTable.qss

```
QTableView
{
  gridline-color: #c7c7c7;
    font: 12pt;

}

QHeaderView
{
    font: 10pt;

}
```

### C.2.5 GroupBox.qss

```
QGroupBox
{
    border: 2px solid white;
    border-color: #c7c7c7;

}

QLabel
{
  font-size: 13px;
  font-weight: bold;

}
```