Proposal

This capstone project involves the use of analyzing natural language and being able to access the information when needed. This project is seeking to solve the problem of creating new dialogue and interactions of established characters solely based on their previous data. One of the best ways of analyzing characters over the long term and also having enough data to create an accurate model are television shows. Unlike movies, television shows have multiple episodes spanning across multiple seasons that can adequately define a character and their interactions with other characters. Of all the shows that are currently on the air, South Park is one of the few shows that been around for over twenty years and would best serve the purpose of getting in-depth analyzes for individual characters.

For a show that has been running for as long as it has, the main difficulty is the amount of material that needs to be produced in order to keep the show on the air. Teams of writers need to be constantly producing new material to keep up with the demand for new content that is just as good if not better than what has already been aired. Sometimes entire new teams need to be hired to replace ex-employees which can sometimes change the quality of material produced and might even change the personality of the characters that have already been established. By implementing an NLP language model we can access work that abides by the parameters of the characters that have previously been determined. Our client would be the creators and producers of the show, and our job is to convince them that using our model can work by checking that new dialogue created would match the personality of the character.

The data that we will be using will be the scripts for every show starting from the first season and ending with Season 18. We will also be looking at the main characters of the show so we will need our data to have every line these characters have had for these episodes. After we have processed this data our goal will be to create separate chatbots that will be able to give responses when queried by the user. Not only can they interact with the user, but they will also be able to interact will each other. Asking the user a question will match their input with the best response from the scripts and that response can then be applied to the other chatbots until we have a couple of lines of dialogue reacted to any type of situation.

Preprocessing

This dataset was acquired from the scripts for the first eighteenth seasons of the show which were posted on the official South Park website. All the scripts were combined into a single CSV file with four columns of data being the Season, Episode, Character, and the Line. This is the dialogue spoken by every character in sequential order and since we will be focusing on the main characters of the show we will be filtering out most of the entries of the data. The main characters and the individuals with the most lines of dialogue are Cartman, Kyle, and Stan. This means that we will be creating three separate chatbots based on the dialogue of these characters and since most of their lines involve talking to each other we should have a large amount of relevant data to look through. While it is nice that we have a record of which season and episode the line is from the show, we will only need the specific for either Cartman, Kyle, and Stan while everything will be removed.

Since this project deals with text and nothing numerical or categorical, we need to go through various stages of preprocessing the dialogue so that it can be analyzed into a model. Preprocessing is actually implemented twice in this project as the lines of dialogue as well as

the question the user inputs both go through the same steps in order to compare similarities between the two. The order in which preprocessing is as follows: expanded contractions, lowercase the letters, remove numbers, spell check, remove stop words, remove punctuations, tokenize, and finally lemmatize. These are in no way all the methods of preprocessing for Natural Language Processing but the most appropriate for the given dataset.

Expanding contractions is probably the least common preprocessing method but since the dialogue is filled with contractions and we are using the Natural Language Toolkit (NLTK), it will treat these as single words and not two separate words. Next, we need to remove any uppercase words by having all words be lowercase because if we don't then they will be treated as separate tokens when capitalized. Next, we need to remove any numbers since we will match words and phrases. Next, we will be employed a spellchecker so fix any typographical errors that may have occurred in the scripts and is the most time-consuming step as it takes a couple of minutes to run. Next, we will be removing stop words which are basically commonly used words that don't contribute to the identification of a sentence or phrase. By removing stop words we decrease the amount of time needed to retrieve results with little to no difference in results. Next, we will be removing all punctuations since we are once again searching by words and phrases. Despite the fact that we are asking the user to input a question, we don't actually need the question mark to implement our search.

The final two steps are tokenization and lemmatization which are essential steps needed to be able to apply a model to our data. All the preprocessing that we have done up to this point has involved the deletion or modification of a string of words but has not actually changed the data itself. Tokenization is the process of breaking down words, phrases, sentences, and paragraphs into simple units called tokens. There are different types of tokenization but for this project, we will be using word tokenization where a single word will be expressed as a single token. All the steps of preprocessing are done to get better results through tokenization so that we remove noise from the data. Lemmatization is an algorithm used directly after tokenization that cuts suffixes and prefixes from a word. By identifying the part of speech a particular word has, it can resolve words to their dictionary form.

Preprocessing is now complete and what we are left with is a list of words for each line of dialogue. We also have converted this into a string for certain processing purposes but this is how we prepare data to be used for modeling.

Exploration

Now that we have preprocessed all the lines of dialogue for the three main characters of the show, we can now find out which words are used most frequently for each. To give a little more context for the show, these characters are third graders in elementary school that frequently use explicit language in their conversation. While the show is not completely uncensored, it is expected that these types of words will be the most used and observed in the data.

Our first step of data exploration involves using Wordcloud to create figures that show the most frequently used words for each character. Instead of just listing how many times a word is encountered, the size of the word will correspond to how frequently it is used. This means the most common word used will be the largest in the figure while the smaller words displayed are used less frequently.

Figure 1: Cartman Wordcloud

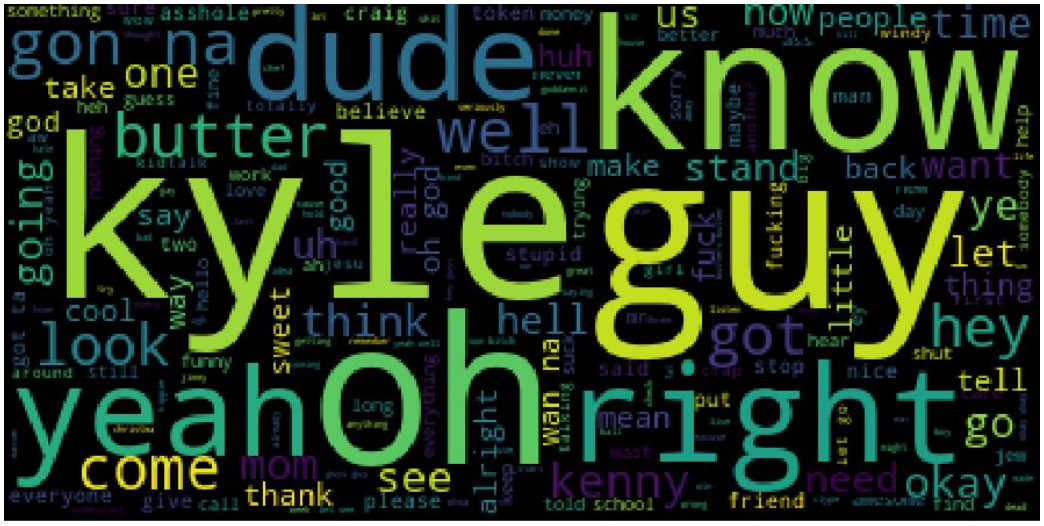


Figure 2: Kyle Wordcloud

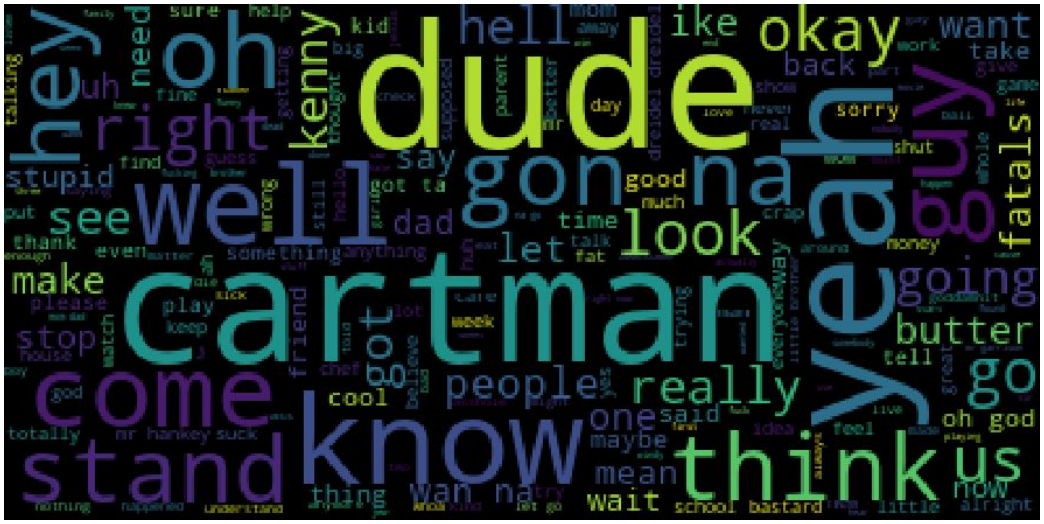
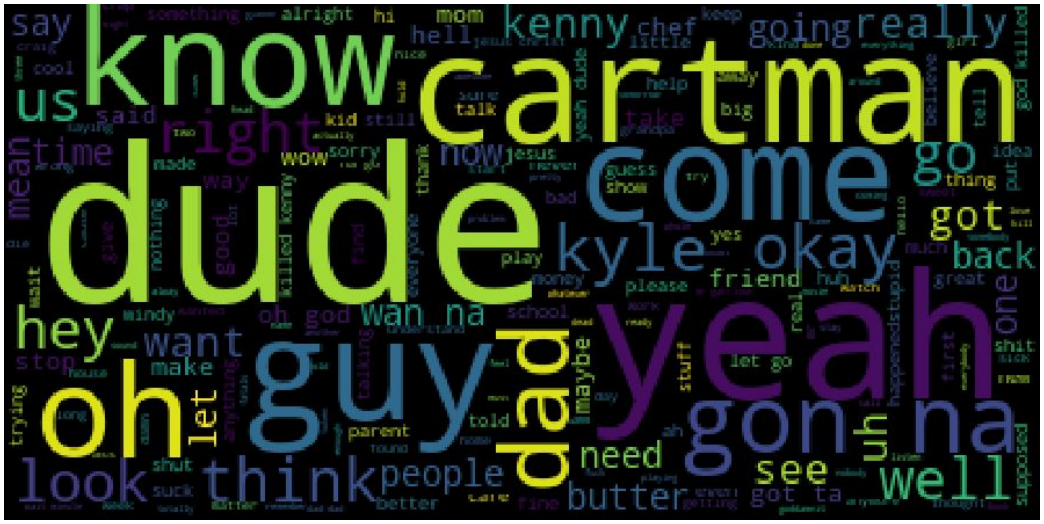
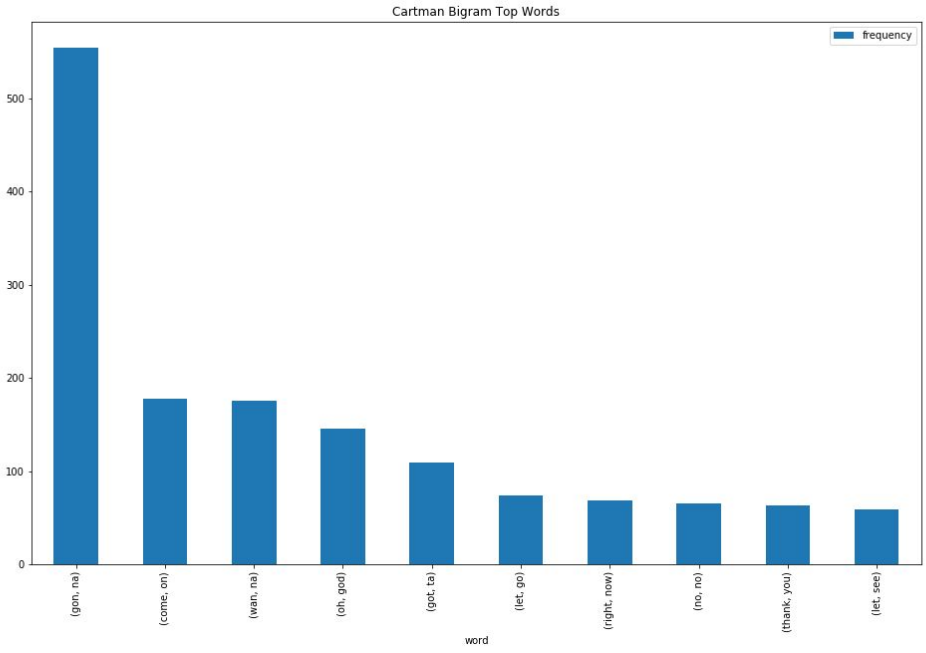


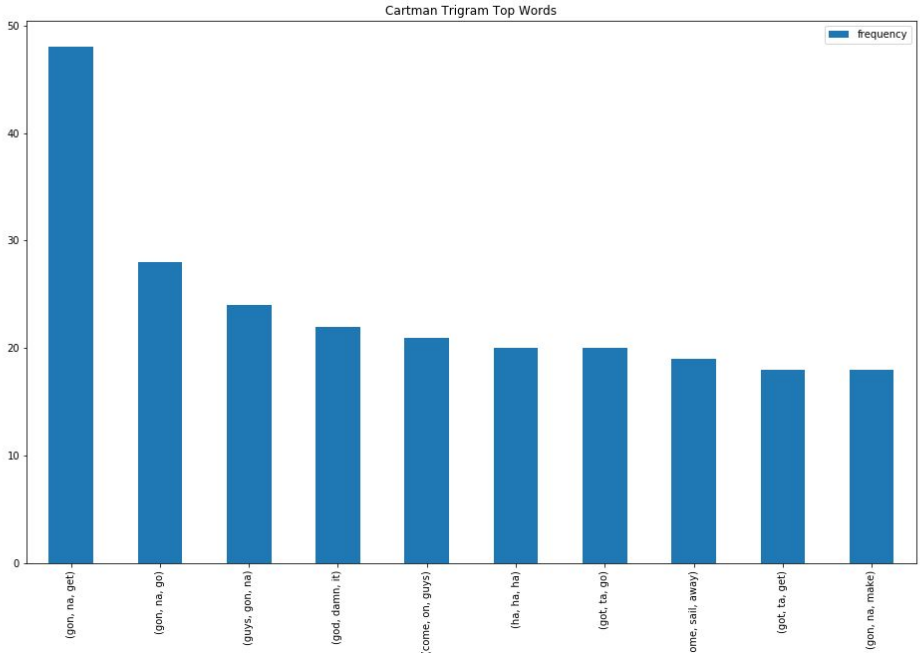
Figure 3: Stan Wordcloud

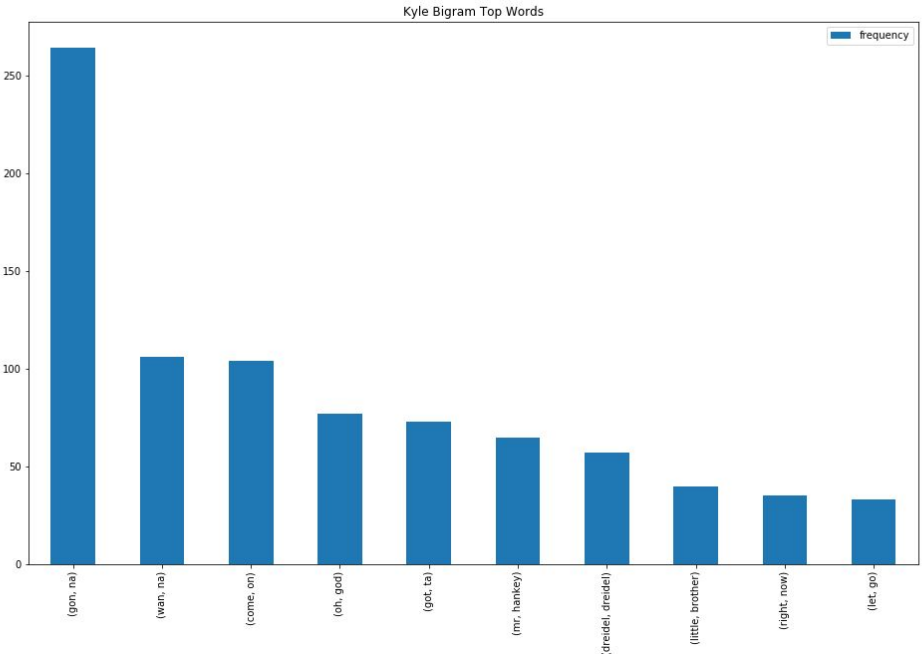


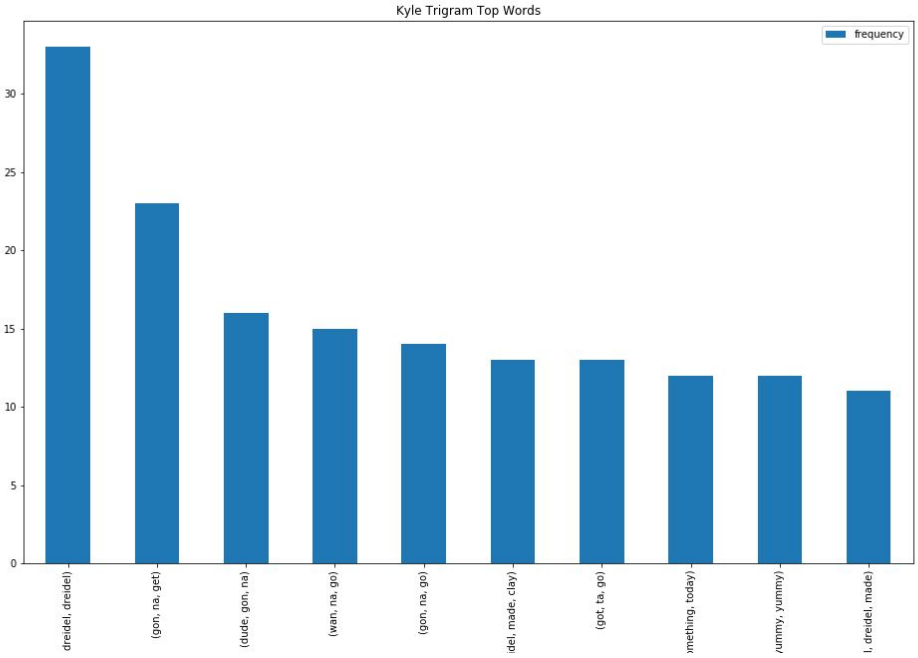
What we see here is that the dialogue for Cartman, Kyle, and Stan show that they most frequently use the word dude, yeah, and each other's name in conversation. It is very difficult to find any kind of vulgar language in any of the Wordclouds which is surprising and shows how they weren't used as frequently as everyone assumes. Another thing that pops up is the word "gon" and "na" which means that our preprocessing split up a single word of gonna into two separate words. This would have to be related to the stage of lemmatization where it reduces the word to its simplest form and in this case, did so incorrectly with the slang term of the phrase going to. There are possibly other slang words that this happened to and gonna is a word that is used frequently and is the most apparent.

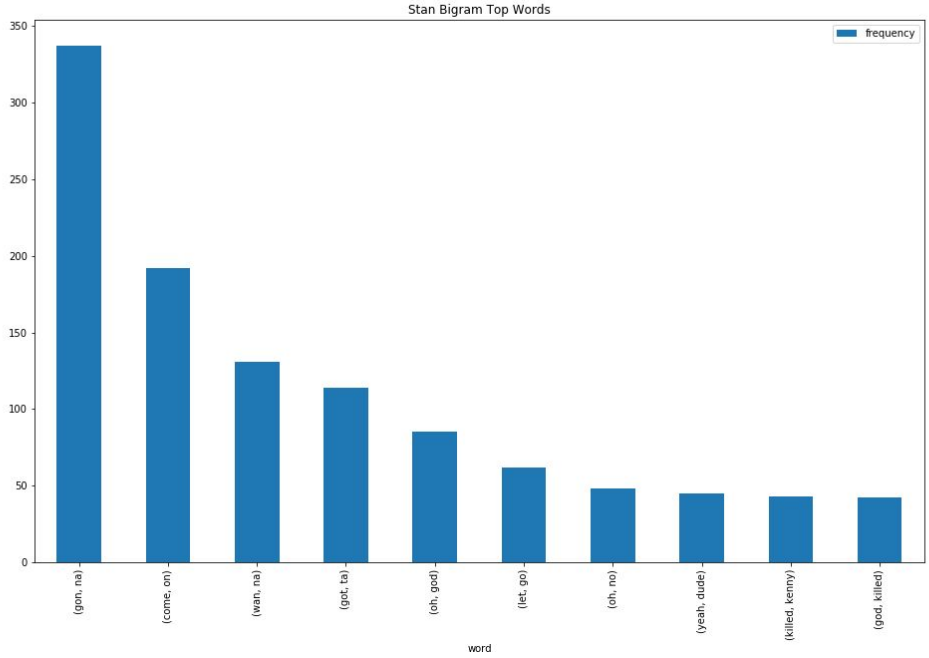
Our next phase of data exploration involves looking at n-grams which are sets of words that occur together sequentially. Unlike Worldcloud which displays single word frequency, we will be looking at the most common phrases of two and three-word phrases called bigrams and trigrams respectively. This means that order matters for n-grams and will not display additional n-grams that are out of sequence.

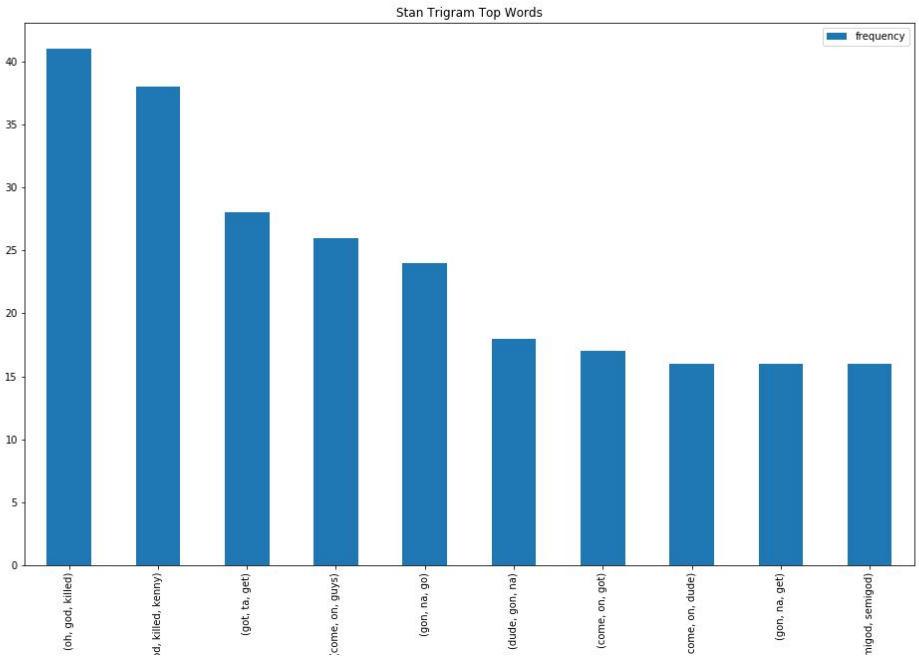



Cartman Trigram Top Words
frequency
50
40
30
20
10
0
(gon, na, get)
(gon, na, go)
(guys, gon, na)
(god, damn, it)
come, on, guys)
(ha, ha, ha)
(got, ta, go)
ome, sail, away)
(got, ta, get)
(gon, na, make)


Kyle Bigram Top Words
frequency
250
200
150
100
50
0
(gon, na)
(wan, na)
(come, on)
(oh, god)
(got, ta)
(mr, hankey)
dreidel, dreidel)
(little, brother)
(right, now)
(let, go)


Kyle Trigram Top Words
frequency
30
25
20
15
10
5
0
dreidel, dreidel)
(gon, na, get)
(dude, gon, na)
(wan, na, go)
(gon, na, go)
idel, made, clay)
(got, ta, go)
mething, today)
/ummy, yummy)
l, dreidel, made)


Stan Bigram Top Words
frequency
350
300
250
200
150
100
50
0
(gon, na)
(come, on)
(wan, na)
(got, ta)
(oh, god)
(let, go)
(oh, no)
(yeah, dude)
(killed, kenny)
(god, killed)
word



Once again we see our problem with our informal contractions showing up again with the words gonna, wanna, and gotta showing up in our n-grams because they are treated as two words. For example, the trigram of “gon, na, go” should actually be a bigram. Besides this, there are expected phrases show up that are used frequently during the show. By removing stop words they are missing the more common words but we are still able to identify what the phrase is pertaining to.

Modeling

We will be using two different types of vectorized models to arrange our tokens in a way that is usable for our chatbots. They are Bag of Words and TF-IDF (Term Frequency-Inverse Document Frequency).

Bag of Words is the more simplistic of the two and works and takes the unique words in the dataset and expresses their occurrence as a vector or string of numbers. Each entry will either have a 1 or 0 value showing that it is contained or missing in that specific entry. It is called a bag of words because any information about the order or structure of words in the dataset is discarded and focuses on just whether or not it is there.

TF-IDF involves the use of two separate mathematical equations to find the numerical value of a word on a scale of 0.000 to 1.000. The first equation is Term Frequency and divides a specific word by the total number of words in the entry. The second equation of Inverse Document Frequency takes the logarithm of the total number of entries a data set has and divides it by the number of times a certain word is found in the entries. This means that the least frequently occurring word will have a large score when encountered comparing to more commonly used ones. TF and IDF are then multiplied by each other to find its specific score. This means that words we have been seeing pop up most frequently from our previous figures will be getting low scores for IDF and have lower TF-IDF scores.

Figure 1: Cartman Bag of Words Top Words

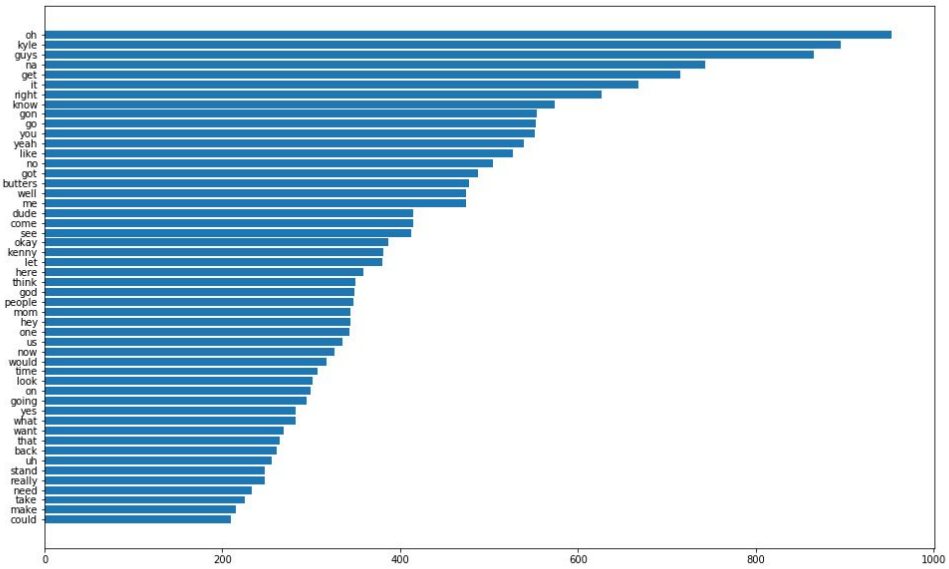


Figure 2: Cartman TF-IDF Top Words

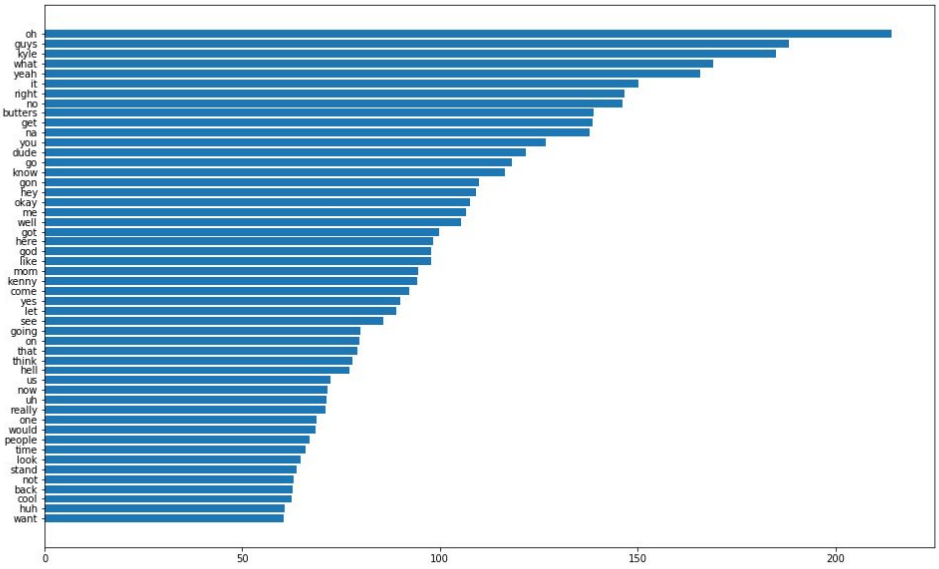


Figure 3: Kyle Bag of Words Top Words

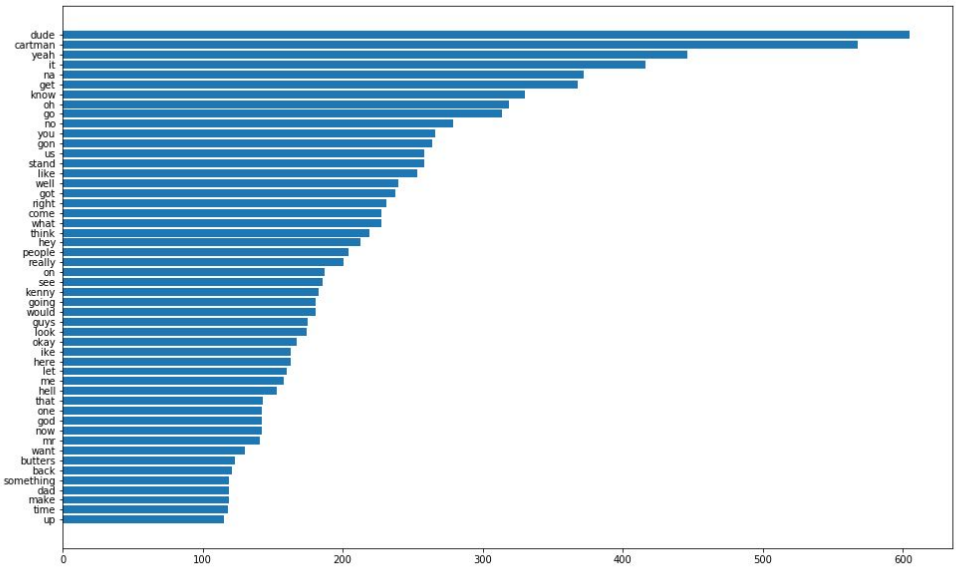


Figure 4: Kyle TF-IDF Top Words

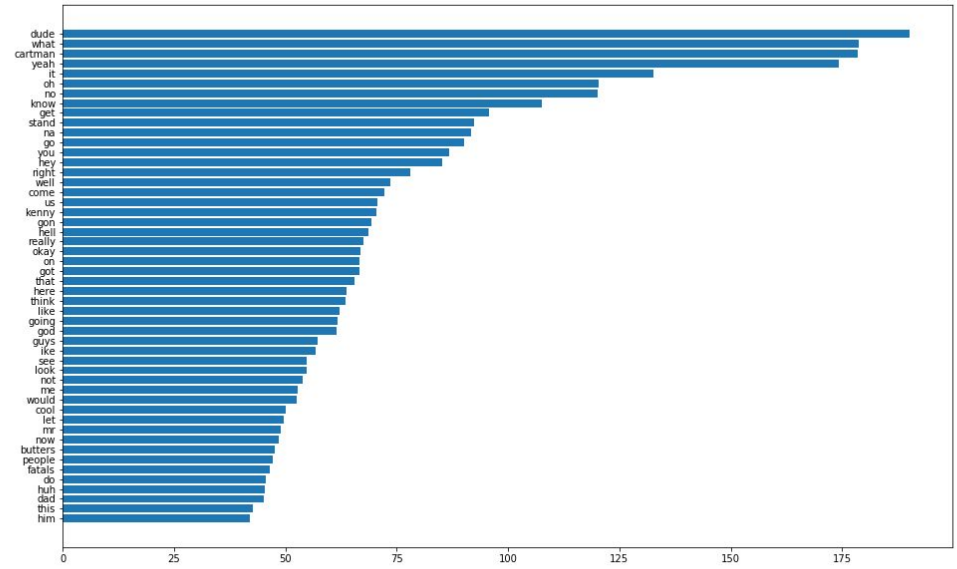


Figure 5: Stan Bag of Words Top Words

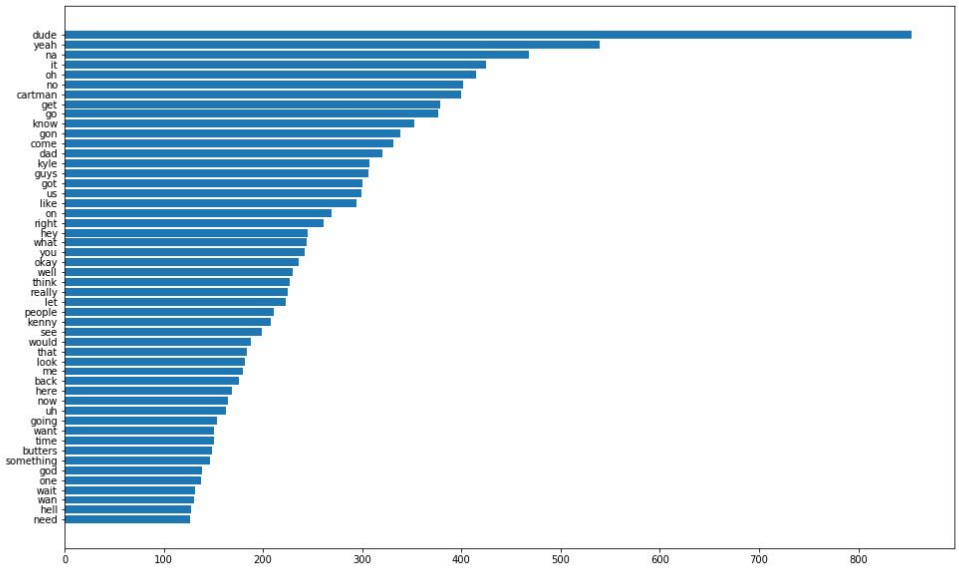
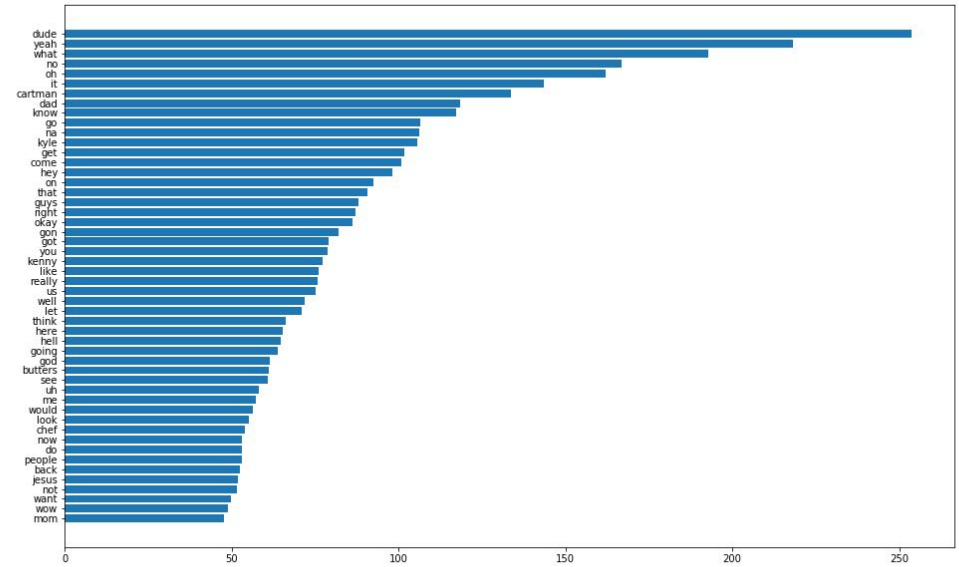


Figure 6: Stan TF-IDF Top Words



As we can see by comparing the Bag of Words and TF-IDF the list of top words is not the same. In our next section of implementing chatbots, we will be seeing if this translates to having different responses chosen for a particular question. While Bag of Words might be the

more simplistic model and TF-IDF is more mathematical in its approach, it is unclear now which is the best in responding.

Chatbots

Chatbots are created by defining a function that applies our same preprocessing steps from the dataset instead used for the user input. This means that it doesn't matter if the query is a single word or a complete sentence with punctuation since it works by comparing the similarity between the remaining words that weren't filtered out. The chatbots are creating for each of the three main characters and also using the Bag of Words model and TF-IDF model making it six in total. Since we are comparing arrays for similarities the steps that we took for analyzing our dataset will be the same exact steps used for the processing of the user input. First, we perform the processing to get rid of any unneeded information such as punctuation, uppercase, stop words, and numbers. Next, we will be vectorizing in order to transform the query into an array. Next, we will be using cosine similarities to find which response will match up the best with the question posed. The value returned will be the one with the largest value and will return the original line of dialogue.

When comparing the difference between the Bag of Words and TF-IDF models we can see that they perform identically when given a single word input. This tells us that when we have only one word to compare with both methods will find the exact same line of dialogue. We see a difference when we have a question with multiple words that need to be compared against. Bag of Words will match its response with the line that matches the most words while TF-IDF will match the single word with the highest score and give a line based on that one word. The problem with using dialogue between characters is that we end up with lines of dialogue with only a couple of words. When we tried to have the chatbots talk to each other we have situations where all three chatbots will output the same word since that has the best match but not necessarily the most appropriate. TF-IDF also will match the highest scoring word in the phrase and ignore the other words because they have a lower score. Bag of Words will match a query with the simplest response needed to fulfill the requirements.