THE MAKING OF A TEXT ANALYSIS

NICHOLAS CATANI

SEATTLE PACIFIC
UNIVERSITY

TABLE OF CONTENT

DATABASE AND SOFTWARE CHOICE 3

INTRODUCTION 4

DATA EXPLORATION 5

DATA PREPARATION 8

DATA RESULTS 16

WHAT DID GO WELL? 21

WHAT DID GO WRONG? 22

WHAT I WOULD DO DIFFERENTLY? 23

DATABASE AND SOFTWARE CHOICE

South Park is an American animated sitcom created by Trey Parker and Matt Stone and developed by Bryan Graden for Comedy Central. The sitcom centers around four boys: Stan Marsh, Kyle Broflovski, Eric Cartman, and Kenny McCormick - see the picture on the front page for reference. The boys live in a fictional small town of South Park, located within the real-life South Park basin in the Rocky Mountains of central Colorado, approximately a one-hour drive from Denver. The town is also home to an assortment of other characters, including students, families, elementary school staff, and various residents.

For this project, I've decided to look for a South Park dataset that would allow me to conduct a text analysis based on characters' lines from hours of the sitcom's episodes. I've stumbled into a dataset on GitHub, containing scripts from seasons 1 to 18. (All-seasons.csv)

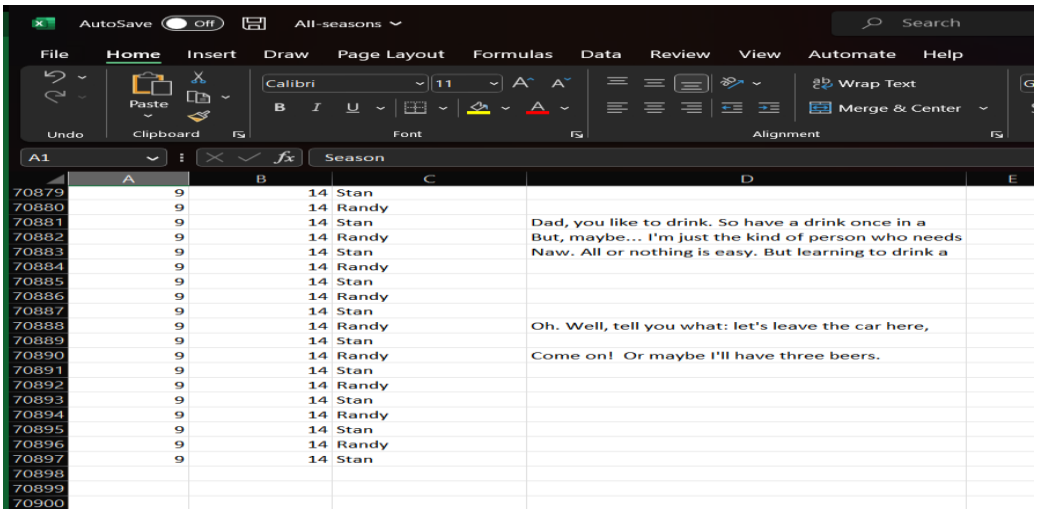
GitHub - BobAdamsEE/SouthParkData: .csv files containing script information including season, episode, character, & line.

As far as the software I'm concerned. I ended up using RStudio since I've never done text mining with it.

Posit | The Open-Source Data Science Company

INTRODUCTION

The dataset contains more than seventy thousand lines said by all the characters from season 1 to season 18. I believe that this is a perfect set to use on text analysis because of its profanity and dark, surreal humor that satirizes a substantial amount of subject matter – eventually, I've boiled everything down to sentiment analysis due to its huge amount of emotional content. The dialogues are spread out by season number, episode, and speaker.



	A	B	C	D
70879	9	14	Stan	
70880	9	14	Randy	
70881	9	14	Stan	Dad, you like to drink. So have a drink once in a
70882	9	14	Randy	But, maybe... I'm just the kind of person who needs
70883	9	14	Stan	Naw. All or nothing is easy. But learning to drink a
70884	9	14	Randy	
70885	9	14	Stan	
70886	9	14	Randy	
70887	9	14	Stan	
70888	9	14	Randy	Oh. Well, tell you what: let's leave the car here,
70889	9	14	Stan	
70890	9	14	Randy	Come on! Or maybe I'll have three beers.
70891	9	14	Stan	
70892	9	14	Randy	
70893	9	14	Stan	
70894	9	14	Randy	
70895	9	14	Stan	
70896	9	14	Randy	
70897	9	14	Stan	
70898				
70899				
70900				

There are a couple of reasons why I chose it. First, I used to watch the sitcom several years back when I was a teenager. Second, it would be exciting to discover the connotation of each dialogue using Data Mining software and check it with the episode. More specifically, I'm seeking answers to Who is the character who says the most negative words? Which words are we talking about? What is the season with the most positive words in it?

DATA EXPLORATION

Normally I map the dataset to have an idea of how to approach the problem and what it looks like. The first thing I do is to load the All-season.csv in the terminal and check the structure using the following script.

```
## LOADING THE CSV FILE AND CHECK THE DATASET

dataset <- read.csv("C:/Users/Nicho/Desktop/All-seasons.csv")
head(dataset, 10)
```

The green text on quote, represents the file path, simply look at the property file and copy/paste the path in the directory. Running the head script gives the following output:

Season	Episode	Character	Line
10	1	Stan	You guys, you guys! Chef is going away.
10	1	Kyle	Going away? For how long?
10	1	Stan	Forever.
10	1	Chef	I'm sorry boys.
10	1	Stan	Chef said he's been bored, so he joining a group called the Super Adventure Club.
10	1	Chef	Wow!
10	1	Mrs. Garrison	Chef?? What kind of questions do you think adventuring around the world is gonna answer?!
10	1	Chef	What's the meaning of life? Why are we here?

All right, the dataset contains four attributes Season, Episode, Character, and Line, and has 70896 records stored – check pic in the introduction section or run the script **summary()**. Now, I would like to know how many characters play a role in the sitcom.

```

## CHECK HOW MANY CHARACTERS ARE IN THE SET

length(unique(dataset$Character))

```

The output should be 3950. Great, there are enough people to populate a small town! Still unsatisfied, who are the main 25 characters who speak the most? Let's find out…

```

library(dplyr)
library(magrittr)
dataset %>%
  count(Character) %>%
  arrange(desc(n)) %>%
  top_n(25)

```

I loaded two libraries that would allow me to cluster the characters to pick the top ones and arrange them in descending order. The above script should produce the following output:

Character	n
Cartman	9,774
Stan	7,680
Kyle	7,099
Butters	2,602
Randy	2,467
Mr. Garrison	1,002
Chef	917
Kenny	881
Sharon	862
Mr. Mackey	633
Gerald	626
Jimmy	597
Wendy	585
Liane	582
Sheila	566
Jimbo	556

It seems that 4 out of 4 main characters of the series are in the top 25 list as being the most loquacious around, with Cartman, Stan, and Kyle standing in the first three positions.

DATA PREPARATION

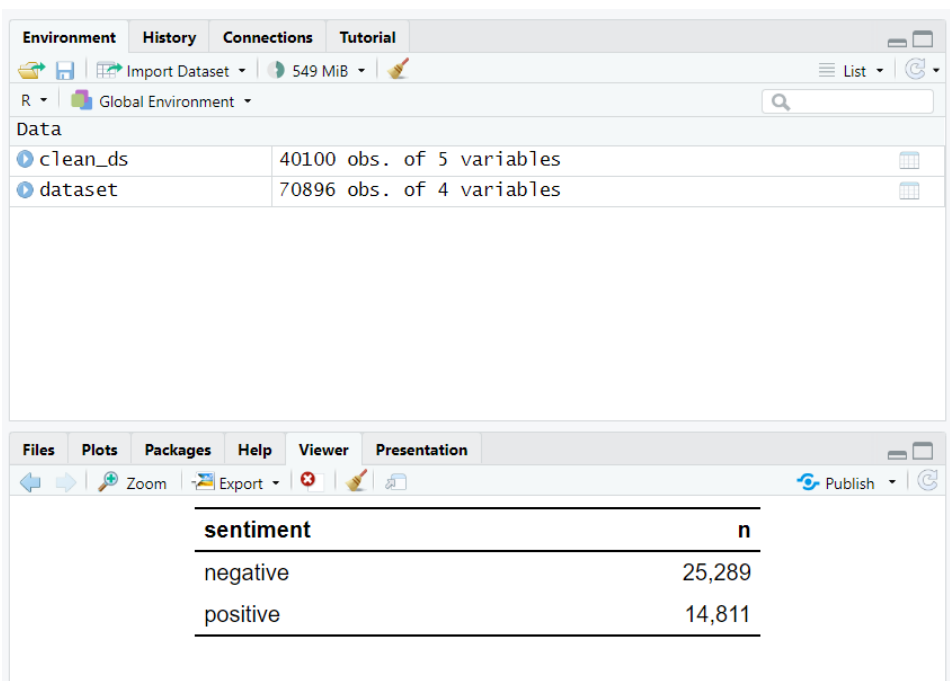
Now that the dataset has been broadly examined, let's jump into the following step and prepare the data for sentiment analysis. The first action that should be taken is to separate all those dialogues into single words. The TOKENIZATION operator does it by transforming words into attributes for further analysis. RStudio has a function in the library (tidytext) that does the trick perfectly, called "unnest_tokens". In essence, it takes two arguments, the first name of the new column containing the words, and the second column from which these words will come – output & input. The second action is to use an "inner_join" algorithm that would extract sentiment words from the dataset and compare them with Bing lexicon – a dictionary entitled to categorize words into either positive or negative categories. In the R manual, 2 Sentiment analysis with tidy data | Text Mining with R (tidytextmining.com), there are two other lexicons based on unigrams. The "AFINN" assigns words with a score that runs between -5 and 5, depending on the score, it would mean either negative sentiment or a positive one. The "NRC" clusters words binarily depending on the emotion spurred – disgust, fear, joy, sadness, trust.

From the looks of it, the dataset does not contain highly sophisticated words associated with the "NRC" lexicon nor did want words' connotations to be classified with numbers. Thus, the "Bing" lexicon suits the most in the analysis.

Having said all of that, let's proceed and find out how many words with negative or positive connotations are present in the dataset.

```
library(tidytext)

clean_ds <- dataset %>%
  unnest_tokens(word, Line) %>%
  inner_join(get_sentiments("bing")) %>%
  anti_join(stop_words, by = "word")

clean_ds %>%
  count(sentiment)

```

Running the script above, the result should be the following…



sentiment	n
negative	25,289
positive	14,811

There are over 25000 negative words and slightly lower than 15000 positive ones in all 18 seasons. It is important to notice how these two algorithms, not only tokenize words, but handle lower case or upper case, and stop word elimination, concurrently. The dataset shrunk almost in half, from having 70000 to 40000 records, adding one more attribute to the table.

South-Park Text Analysis.R* × clean_ds ×

Filter

	Season	Episode	Character	word	sentiment
1	10	1	Stan	bored	negative
2	10	1	Stan	super	positive
3	10	1	Chef	wow	positive
4	10	1	Cartman	miss	negative
5	10	1	Cartman	miss	negative
6	10	1	Mayor McDaniels	miss	negative
7	10	1	Randy	super	positive
8	10	1	Gerald	wow	positive
9	10	1	Gerald	super	positive
10	10	1	Mrs. Garrison	super	positive
11	10	1	Randy	super	positive
12	10	1	Randy	super	positive
13	10	1	Randy	happy	positive
14	10	1	Stan	happier	positive

Showing 1 to 15 of 40,100 entries, 5 total columns

At this point, the dataset is not perfect yet because it contains word repetitions that might alter the analysis. Thus, the intention is to group all the records by word root.

```
lexic_ds <- clean_ds %>%
  count(word, sentiment)

top_20 <- lexic_ds %>%
  group_by(sentiment) %>%
  top_n(20) %>%
  ungroup() %>%
  mutate(word = reorder(word, n))

```

Filter

	word	sentiment	n
320	clique	negative	1
321	cloud	negative	19
322	clueless	negative	1
323	clumsy	negative	1
324	cocky	negative	1
325	cold	negative	107
326	collapse	negative	3
327	colorful	positive	2
328	combust	negative	8
329	comfort	positive	3

Showing 320 to 329 of 2,472 entries, 3 total columns

Filter

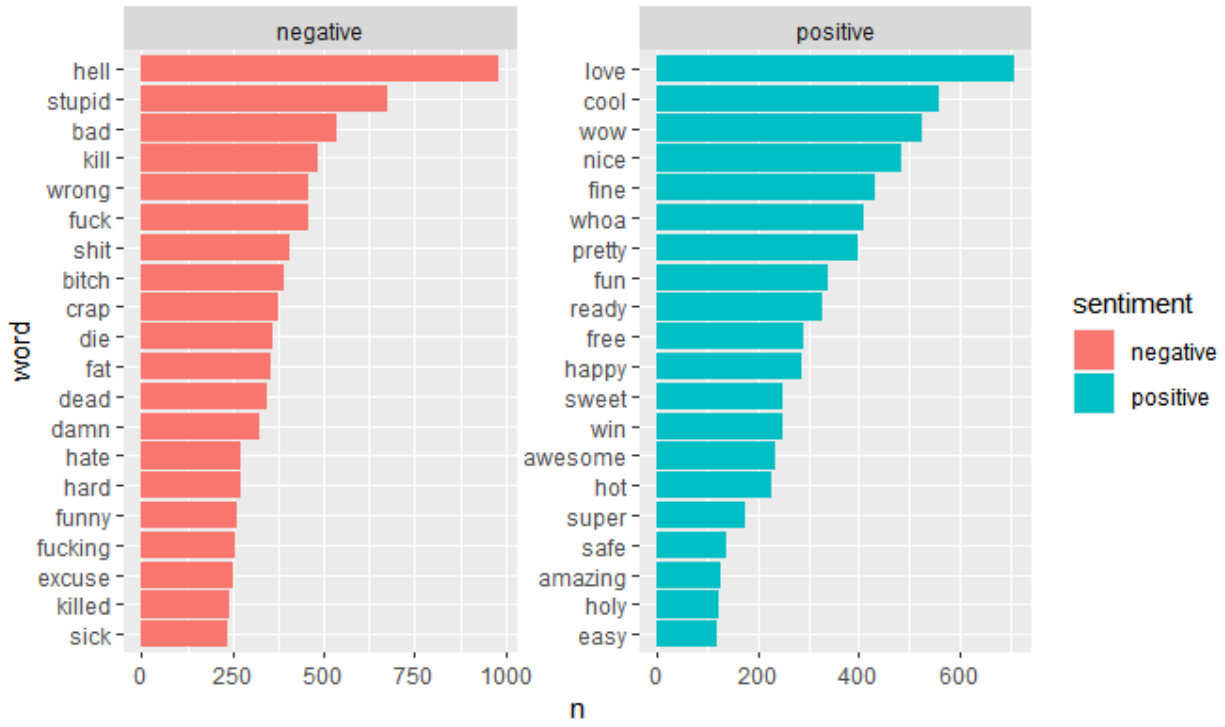
	word	sentiment	n
1	hell	negative	981
2	love	positive	708
3	stupid	negative	678
4	cool	positive	559
5	bad	negative	537
6	wow	positive	527
7	nice	positive	486
8	kill	negative	482
9	wrong	negative	458
10	fuck	negative	457
11	fine	positive	434
12	shit	negative	409
13	whoa	positive	408
14	pretty	positive	399
15	bitch	negative	393
16	crap	negative	378
17	die	negative	359
18	fat	negative	353
19	dead	negative	347
20	fun	positive	339

Showing 1 to 20 of 40 entries, 3 total columns

There are close to 3000 distinct words in the dialogue. With the top 20 (passive and negative) are illustrated in the chart above.

Also, there is a fancy way to represent the most frequent words said in the series. By running the following script, it is possible to create a bar chart that displays the outcome.

```
library(ggplot2)
ggplot(top_20, aes(word, n, fill = sentiment)) +
  geom_col() +
  facet_wrap(~sentiment, scales = "free") + coord_flip()
```



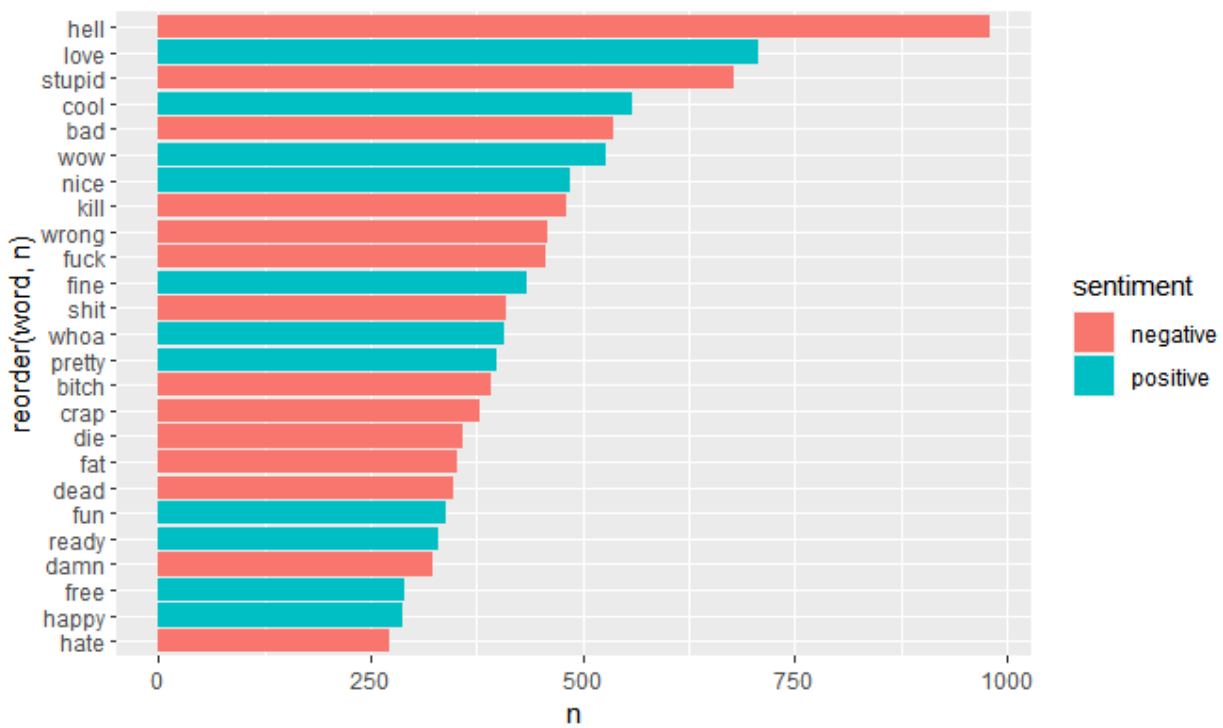
An alternative version is achievable by using word cloud - two additional libraries are required to get the task done.

```
library(reshape2)
library(wordcloud)

lexic_ds %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(max.words = 70)
```

The orange words are positive, and the green ones are negative - forgot to insert a legend.

Rather than having two distinct bar charts for either positive or negative, it is possible to cluster them together and form a single one.

```
count_words <- clean_ds %>%
  count(word, sentiment, sort = TRUE)

top_25 <- head(count_words, 25)
ggplot(top_25, aes(x = reorder(word, n), n, fill = sentiment)) + geom_col() + coord_flip()
```



DATA RESULTS

It is time to answer the following questions – which is the objective of the analysis.

- **Who are the characters with the most positive and negative words said?**

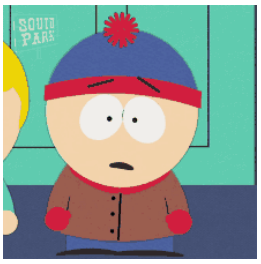
```
clean_ds$Character <- sapply(clean_ds$Character, tolower)

clean_ds %>%
  count(Character, sentiment) %>%
  arrange(desc(n)) %>%
  top_n(30)
```

	Character	sentiment	n
1	cartman	negative	4480
2	cartman	positive	2356
3	kyle	negative	2168
4	stan	negative	2131
5	stan	positive	1102
6	kyle	positive	981
7	butters	negative	804
8	randy	negative	799
9	butters	positive	541
10	randy	positive	538
11	mr. garrison	negative	421
12	mr. mackey	negative	310
13	chef	negative	305
14	jimmy	positive	257
15	jimbo	negative	229
16	mr. garrison	positive	228
17	jimmy	negative	217
18	sharon	negative	215
19	chef	positive	207
20	announcer	negative	200
21	kenny	negative	197
22	wendy	negative	196
23	announcer	positive	170
24	gerald	negative	167
25	narrator	negative	167
26	mrs. garrison	negative	161
27	sheila	negative	157
28	reporter	negative	156
29	mr. mackey	positive	152
30	singers	negative	148

Eric Cartman leads on the board for both being the most illiterate character of the series with 4480 words and the politest character with 2356 words.

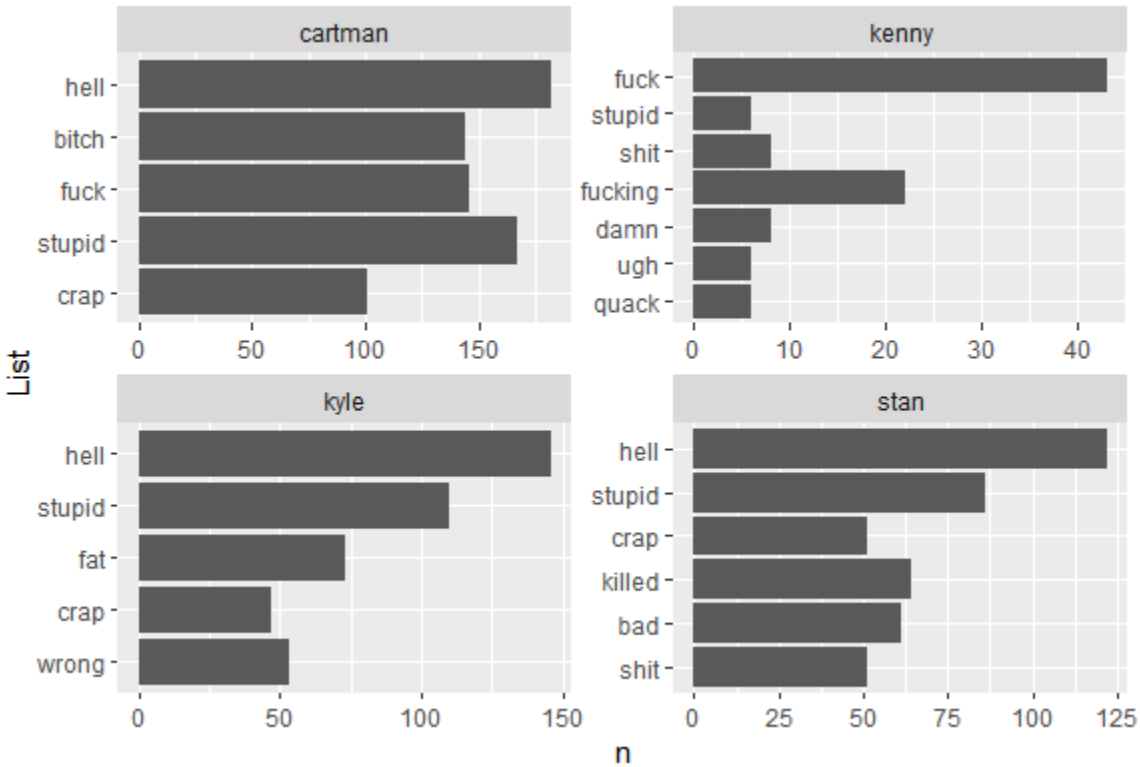
Followed by Kyle...

And Stan…



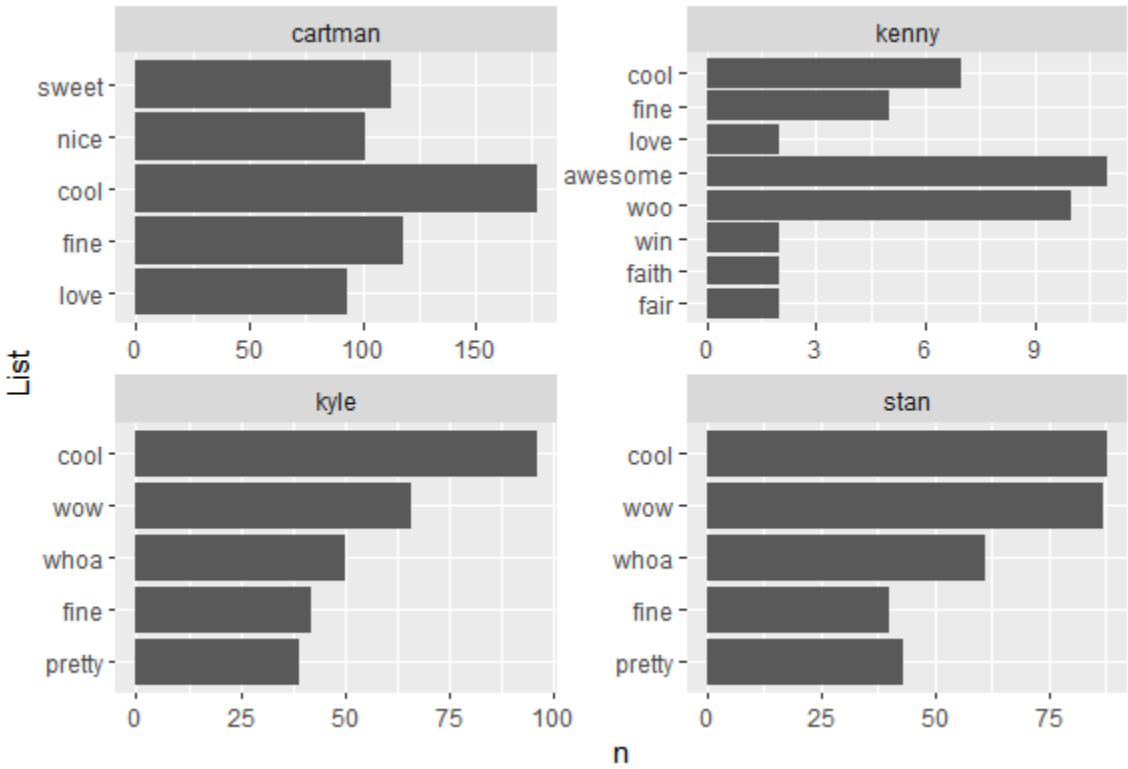
- **What are the most common negative words said by the main characters? – Cartman, Kenny, Stan, and Kyle.**

```
char_neg_words <- clean_ds %>%
  inner_join(get_sentiments("bing")) %>%
  filter(Character %in% c("stan", "kenny", "cartman", "kyle")) %>%
  filter(sentiment == "negative") %>%
  count(word, Character) %>%
  group_by(Character) %>%
  top_n(5) %>%
  ungroup() %>%
  mutate(List = reorder(paste(word), n))

ggplot(char_neg_words, aes(List, n)) + geom_col(show.legend = FALSE) +
  facet_wrap(~Character, nrow = 2, scales = "free") + coord_flip()
```



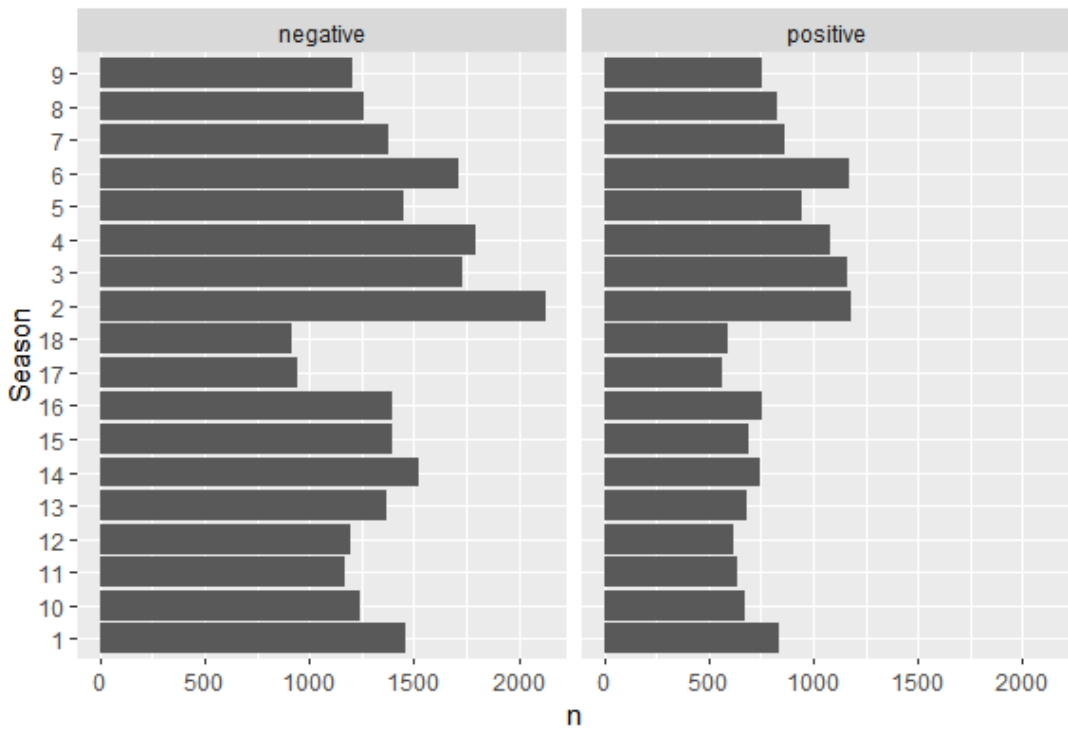
- **What are the most common negative words said by the main characters? – Cartman, Kenny, Stan, and Kyle.**

```
char_pos_words <- clean_ds %>%
  inner_join(get_sentiments("bing")) %>%
  filter(Character %in% c("stan", "kenny", "cartman", "kyle")) %>%
  filter(sentiment == "positive") %>%
  count(word, Character) %>%
  group_by(Character) %>%
  top_n(5) %>%
  ungroup() %>%
  mutate(List = reorder(paste(word), n))

ggplot(char_pos_words, aes(List, n)) + geom_col(show.legend = FALSE) +
  facet_wrap(~Character, nrow = 2, scales = "free") + coord_flip()
```



- **Which season contains more negative than positive words? What about more positive than negative ones?**

```
season <- clean_ds %>%
  inner_join(get_sentiments("bing")) %>%
  group_by(Season, sentiment) %>%
  count(Season, sentiment)

ggplot(season, aes(Season, n, fill - sentiment)) + geom_col() + facet_grid(~sentiment) + coord_flip()

```



WHAT DID GO WELL?

The dataset did not require a lot of data cleaning. It is well known that data accuracy is a key attribute of high-quality data – not to mention the impact on the overall increase in productivity and the availability of the right information during the results. The dataset contained 4 attributes and more than 70000 records without having missing values. All these premises made the task easier during data preparation, where I had to tokenize words, remove stop words, adjust the lowercase, and lemmatize – which is the process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the dictionary form.

The second thing I may think of is perhaps the programming language itself. I had the chance to use RStudio since Undergrad - when I was a freshman at the University of Rome. In my opinion, RStudio is designed to make it easy to write scripts compared to Python programming, which requires more critical thinking and logic. In R, as soon as a user creates a script, the window within the session adjusts automatically. Also, it can call up potential syntax options while the user is writing just by using the tab key. On top of that, the programming language makes graphics more accessible for a casual user. Scientists may easily click back and forth between plots, change the sizes of the plot without rerunning the script, and export/copy plots to include in documents.

WHAT DID GO WRONG?

Perhaps pointing the finger toward my analysis. Even though I've learned a lot in this project, "as I mentioned earlier, I never had the chance to use RStudio for text analysis", it took me some time to implement and master the sentiment analysis. There were two major reasons for my non-extremely detailed analysis: First, the limited amount of time I had to conduct deeper research and study. I've experienced that having 9 credits (3 graduate courses) plus working, is extremely challenging to deal with, especially if the student wants to achieve high grades. – even for a hardcore reader. During the last two weeks, I felt overwhelmed by the final projects, up to a point when my mind started to wonder how to cut corners – Fortunately, I am mature enough to not fall into this behavior. Having said that, the second reason is just a mere consequence of the first one. Given the limited amount of time I had at my disposal, as mentioned earlier, there were a few scripts where I had to spend some extra time to pull that off. I wanted to run a second analysis using topic modeling, but unfortunately, I failed.

I'm dealing with study burnout. Thus, I've decided to spend the spring holidays away from any kind of book. Last few days I felt overwhelmed and unable to meet constant demands, up to a point, where I started to begin to lose interest or motivation which led me to take on certain roles in the first place. I did a quick search on the internet to see if there was such a thing, to burn out because tired of studying, and I figured out that is common. It happens when a person is stressed by studies for a long time. I'm not a procrastinator, I tend to get the job done quickly, but sometimes I fall into the perfectionism trap, which costs me hours of precious sleep at nighttime.

WHAT I WOULD DO DIFFERENTLY?

By the time I realized what I would do differently, I was halfway through the project. I guess the thought was to change the dataset. I am a graduate student at Seattle Pacific University, a Christian School, with professors who might feel awkward reading this analysis. The sitcom is well known for its profanity and dark, surreal humor that satirizes a substantial amount of subject matter - something that might go against the Bible. Since the beginning, I thought about how good it could be to run a sentiment analysis on the South Park dataset, because of its consistent content of swearing and dirty words, but I failed to consider how a Christian reader would perceive the report. I am not Christian, despite being born and raised in Rome "my apartment is just two metro stops away from the Vatican City, where the Pope lives" but my grandma is. As such, I'll end this document citing a small verse … "Watch the way you talk. Let nothing foul or dirty come out of your mouth. Say only what helps, each word a gift" (Ephesians 4:29)