NLP #TimesUp Project

Jennifer Chicchi

10/01/2019

Introduction to NLP

Our fundamental understanding of the human social world rests on interpreting words and concepts that are expressed in human language. One powerful and exciting tool that can help us to understand language is NLP (Natural Language Processing) technology. Though most methods that are central to NLP have been around for decades, the recent explosion in big data technology has empowered us to use such techniques in increasingly novel and useful ways. The purpose of this project was to demonstrate how NLP and data science methods can be used to extract meaningful insights about a recent social phenomenon, the #TimesUp movement.

#TimesUp Movement

In recent years there has been spreading awareness of sexual harassment and increased activism surrounding this issue. The #MeToo movement, which began on Twitter in 2017, focused on confronting sexual assault and harassment, as well as making specific calls to action, such as asking for apology or other compensatory responses from perpetrators. The #MeToo movement has been seen as a response to issues of general sexual harassment, but there have also been more targeted efforts, like #TimesUp, which has specifically focused on sexual harassment in the workplace. The #TimesUp movement began in 2018 and has been mostly understudied compared to the #MeToo movement. What I wanted to know from this analysis was whether and how NLP techniques could be used to aid in our collective understanding of this recent social movement. While I didn't enter this analysis with a specific hypothesis in mind, I wanted to see what we could learn from using exploratory techniques, spanning simple data visualizations all the way through advanced machine learning clustering algorithms. My hope in doing so was to learn more about the movement and possibly uncover some insights that could lead to more specific hypothesis for future analyses.

Analysis Outline

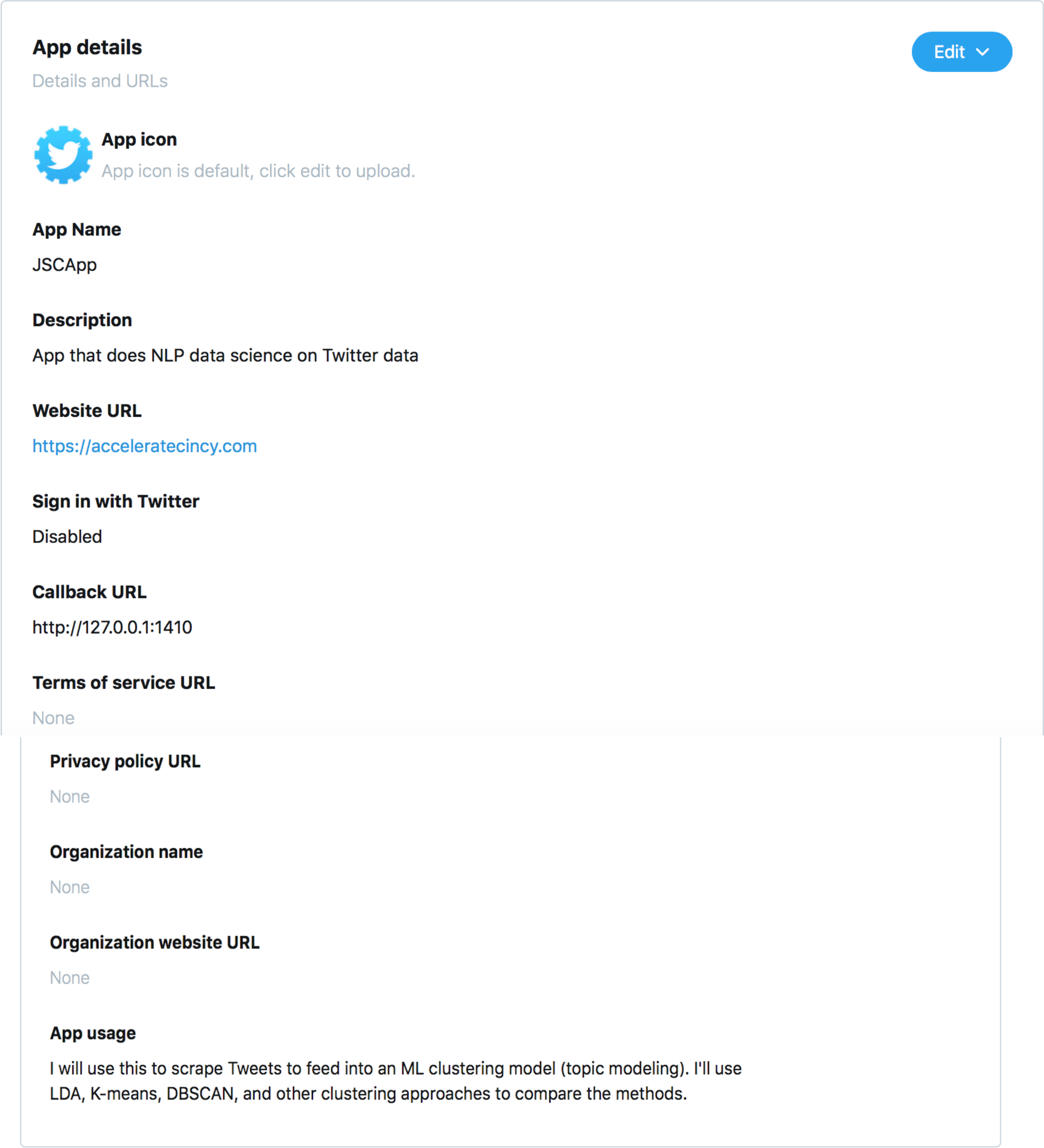
1. Data acquisition via Twitter API
2. Data cleaning using regular expressions
3. Feature engineering including text metadata, lubridate package, and sentiment
4. Exploratory analysis using dplyr and ggplot2
5. Machine learning NLP preprocessing techniques
6. Unsupervised machine learning methods for cluster analysis

1. Data acquisition via Twitter API

Having chosen the topic that I wanted to explore in this data science project, the next step was to decide how to gather the data. There are many social networks and apps that have their own interface that programmers can work with. These interfaces are called APIs (Application Programming Interface). Here, I chose to focus on the Twitter API. Here are the steps I followed to set up the Twitter API:

1. I visited the Twitter Developer's Site at: http://dev.twitter.com

2. I signed in with my Twitter account
3. I visited Twitter's app website at: http://apps.twitter.com
4. I created a new application
5. I filled out the application details with: Name, Description, Website, Callback URL. The results appeared as follows:



App details

Details and URLs

App icon

App icon is default, click edit to upload.

App Name

JSCApp

Description

App that does NLP data science on Twitter data

Website URL

https://acceleratecincy.com

Sign in with Twitter

Disabled

Callback URL

http://127.0.0.1:1410

Terms of service URL

None

Privacy policy URL

None

Organization name

None

Organization website URL

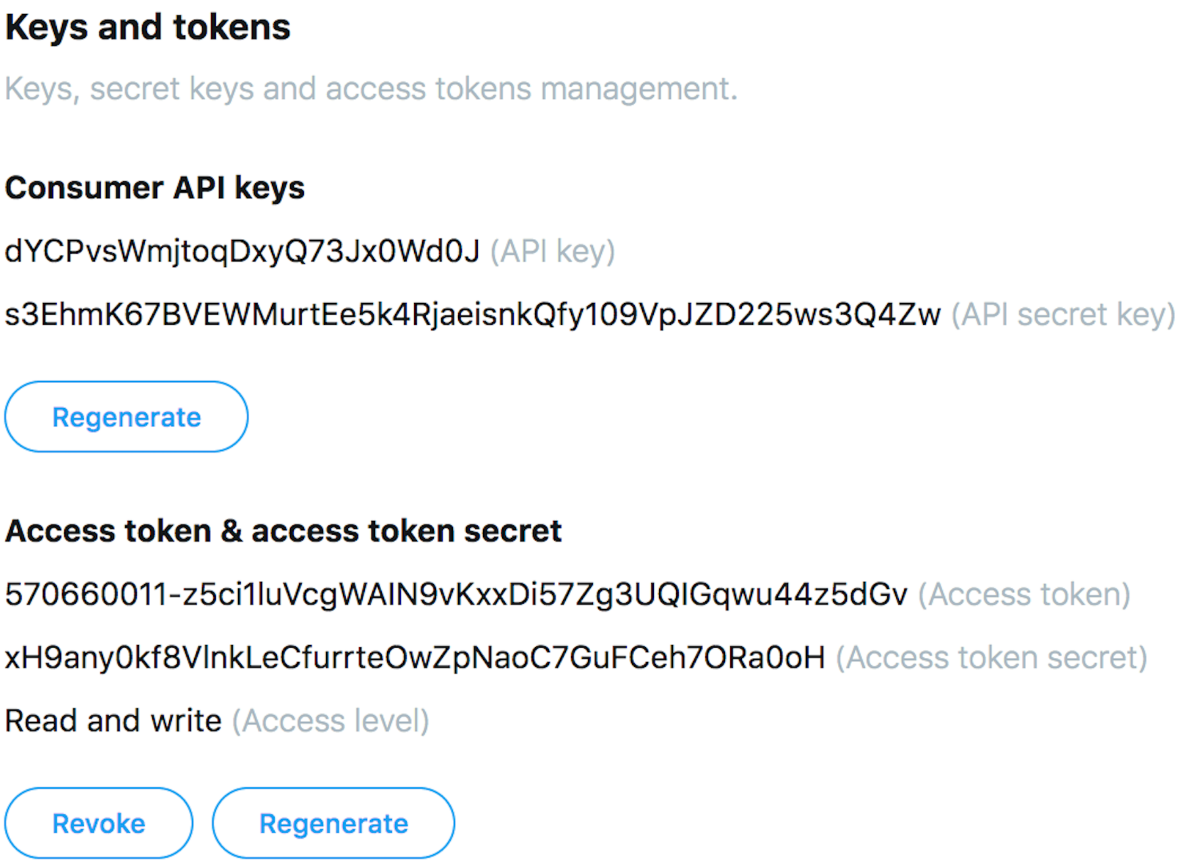
None

App usage

I will use this to scrape Tweets to feed into an ML clustering model (topic modeling). I'll use LDA, K-means, DBSCAN, and other clustering approaches to compare the methods.

6. I created my access token
7. I chose the access type I needed
8. I made a note of my OAuth Settings, as seen below:

9. I then pasted the Consumer Key, Consumer Secret, OAuth Access Token and OAuth Access Token Secret into R. Again, see below:



Keys and tokens

Keys, secret keys and access tokens management.

Consumer API keys

dYCPvsWmjtoqDxyQ73Jx0Wd0J (API key)

s3EhmK67BVEWMurtEe5k4RjaeisnkQfy109VpJZD225ws3Q4Zw (API secret key)

Access token & access token secret

570660011-z5ci1luVcgWAIN9vKxxDi57Zg3UQIGqwu44z5dGv (Access token)

xH9any0kf8VlnkLeCfurrteOwZpNaoC7GuFCeh7ORa0oH (Access token secret)

Read and write (Access level)

Next, I was ready to code. First I set the API credentials and connected to Twitter Oauth.

```r
library(twitteR)
```

```
## Warning: package 'twitteR' was built under R version 3.6.1
```

```r
consumer_key        <- "dYCPvsWmjtoqDxyQ73Jx0Wd0J"
consumer_secret     <- "s3EhmK67BVEWMurtEe5k4RjaeisnkQfy109VpJZD225ws3Q4Zw"
access_token        <- "570660011-z5ci1luVcgWAIN9vKxxDi57Zg3UQIGqwu44z5dGv"
access_token_secret <- "xH9any0kf8VlnkLeCfurrteOwZpNaoC7GuFCeh7ORa0oH"
setup_twitter_oauth(consumer_key, consumer_secret, access_token, access_token_secret)
```

```
## [1] "Using direct authentication"
```

Then I extracted up to 20,000 Tweets containing the search term #TimesUp.

```
times_up_tweets <- searchTwitter("#TimesUp", n=20000, lang ="en")
```

Next I converted the .json output to an R data frame

```
tweets_df <- twListToDF(times_up_tweets)
```

I ran simple descriptive analytics on the data frame.

```
summary(tweets_df[,c(3:6)])
```

```
##  favorited       favoriteCount              replyToSN                      created    
##  Mode :logical   Min.   :   0.000   realDonaldTrump:  17   2019-06-18 18:11:37:   3  
##  FALSE:7609      1st Qu.:   0.000   TIMESUPNOW     :  11   2019-06-19 16:06:50:   3  
##                  Median :   0.000   MiraSorvino    :   8   2019-06-25 02:04:03:   3  
##                  Mean   :   2.676   CollectiveShout:   6   2019-06-16 19:52:58:   2  
##                  3rd Qu.:   0.000   NYMag          :   6   2019-06-17 02:15:44:   2  
##                  Max.   :1933.000   (Other)        : 575   2019-06-17 15:43:01:   2  
##                                     NA's           :6986   (Other)            :7594  
```

And we can see a sample of our Tweets

```
print(substr(tweets_df[1, 2], 0, 80))
```

```
## [1] "RT @Killtime2k: Final roster for WR is set ready for the WR open tourney\n\nPg: KT"
```

At this point our Tweets were not clean. In the next section we'll address this.

2. Data cleaning using regular expressions

Please note, prior to doing this analysis, I studied how to write for loops, if/else logic, and regular expression patterns. All of these techniques are necessary for performing advanced text analytics. The details of this work can be found here. https://github.com/jchicchi/NLP-Project-TimesUp/

Context

Regular expressions are character-matching algorithms that allow us to identify specific key phrases or other patterns in our text data. They can be used to perform targeted searches, like if you want to find all instances of the word 'TimesUp', all hashtags, all urls, etc. They also can be used for removing certain patterns from the text, such as URL's or special characters, as is common in data cleaning.

Methodology

I used a series of regular expressions for data cleaning, as detailed below. I also performed common data cleaning techniques, such as removing trailing and leading whitespace from the text and lowercasing all the words.

Code

Here, we can use regex patterns to find certain types of text, such as URL's.

```r
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 3.6.1
```

```r
# Extract all URL's
urlp<-"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+"
str_extract_all(tweets_df$text[1:100], urlp)[0:5]
```

```
## [[1]]
## character(0)
##
## [[2]]
## character(0)
##
## [[3]]
## [1] "https://t.co/A36hlpiwlT"
##
## [[4]]
## character(0)
##
## [[5]]
## character(0)
```

Next we'll remove various sources of noise in our text.

```r
# Remove all URL's
tweets_df$cleaned_text <- str_remove_all(tweets_df$text, urlp)

# Remove RTs
retweet_pattern <- "(RT|via)((?:\\b\\W*@\\w+)+)"
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, retweet_pattern)

# Remove @ shoutouts
shoutout_pattern <- "@\\w+"
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, shoutout_pattern)

# Remove \n
newline_pattern <- '\n'
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, newline_pattern)

# Remove &amp
amp_pattern <- '&amp;'
```

```r
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, amp_pattern)

# Remove hashtags
hashtag_pattern <- "#\\S+"
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, hashtag_pattern)

# Remove emojis
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, "[^[:ascii:]]")

# Remove all remaining punctuation
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, "[[:punct:]]")
```

Finally, we will perform some extra cleaning steps, such as lowercasing words and adding a final regex pattern to remove special characters the previous patterns missed. We will also trim any excess white space on either the front or end of the strings.

```r
# Lowercase all words
tweets_df$cleaned_text <- str_to_lower(tweets_df$cleaned_text)

# Extra regex step
tweets_df$cleaned_text <- str_remove_all(tweets_df$cleaned_text, 'http')

# Trim trailing and leading white spaces
tweets_df$cleaned_text <- str_squish(tweets_df$cleaned_text)
```

Now our text data is much cleaner and we are ready to explore further.

```r
print(substr(tweets_df$cleaned_text[1], 0, 80))
```

```
## [1] "final roster for wr is set ready for the wr open tourneypg ktsg ktsf ktpf ktc kt"
```

3. Feature engineering including text metadata, lubridate package, and sentiment

Context

In machine learning and analysis in general, oftentimes you can extract more signal by creating new features based on the information you already have. This step is called feature engineering. Features are the variables found in the given problem set that can strongly/sufficiently help us build an accurate predictive model. Due to its inherent richness as a data source, text offers us much opportunity for creating features.

Methodology

I used the lubridate package to create columns based on the date and time of the tweets. I also summarized each tweet in terms of its word count, its number of characters, and a metric I created called word sophistication. Finally, I performed sentiment analysis on the tweets to examine how emotionally positive or negative the text was.

I start with some basic date conversions.

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 3.6.1
```

```
# Convert date to yyyy/mm/dd hh:mm:ss format
tweets_df$created <- ymd_hms(tweets_df$created)

# Converts from Coordinated Universal Time UTC/Greenwich Mean Time to Mountain Time Zone MDT
tweets_df$created <- with_tz(tweets_df$created)
```

Next I engineer some features

```
tweets_df$hour <- hour(tweets_df$created)
tweets_df$month <- month(tweets_df$created)
tweets_df$weekday <- wday(tweets_df$created)  #which weekday a Tweet was on
```

Next I extract the sentiment (emotion) of the Tweet

```
library(syuzhet)
```

```
## Warning: package 'syuzhet' was built under R version 3.6.1
```

```
word.df <- as.vector(tweets_df$cleaned_text)

# Calls the NRC sentiment dictionary to calculate the presence of eight different emotions
#and their corresponding valence in a text file.
emotion.df <- get_nrc_sentiment(word.df)

str(emotion.df)
```

```
## 'data.frame':    7609 obs. of  10 variables:
##  $ anger       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ anticipation: num  1 1 1 0 0 0 1 0 0 0 ...
##  $ disgust     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ fear        : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ joy         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ sadness     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ surprise    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ trust       : num  0 0 0 0 3 0 0 0 0 0 ...
##  $ negative    : num  0 0 0 0 0 0 0 2 0 0 ...
##  $ positive    : num  0 0 0 0 3 0 0 0 0 0 ...
```

We can use the NRC sentiment scores to explore our Tweets by their emotion.

Find angry tweets

```
angry_index <- emotion.df$anger > 0
print(substr(tweets_df$cleaned_text[angry_index][4], 0, 80))
```

```
## [1] "johnny depp the real victim"
```

Find positive tweets

```
pos_index <- emotion.df$positive > 0
print(substr(tweets_df$cleaned_text[pos_index][6], 0, 80))
```

```
## [1] "the evolution is here"
```

Find negative tweets

```
neg_index <- emotion.df$negative > 0
print(substr(tweets_df$cleaned_text[neg_index][18], 0, 30))
```

```
## [1] "tarantino abused women on set "
```

Let's add all the NRC Sentiment classifications to our data frame

```
tweets_df <- cbind(tweets_df, emotion.df)
head(tweets_df[,22:26])
```

```
##   anger anticipation disgust fear joy
## 1     0            1       0    0   0
## 2     0            1       0    0   0
## 3     0            1       0    0   0
## 4     0            0       0    0   0
## 5     0            0       0    0   0
## 6     0            0       0    0   0
```

Finally we'll add a column for numeric sentiment

```
sent.value <- get_sentiment(word.df)
tweets_df$sent <- sent.value
```

We'll add a feature that captures the number of characters in the tweets

```
# Number of characters
tweets_df$nchar_raw <- nchar(as.character(tweets_df$text))
tweets_df$nchar_clean <- nchar(as.character(tweets_df$cleaned_text))
```

To obtain the word counts in the tweets, first we must tokenize our text. This means splitting each tweet into a list of words.

```
library(tokenizers)
```

```
## Warning: package 'tokenizers' was built under R version 3.6.1
```

```
tokens <- tokenize_words(tweets_df$cleaned_text)
```

Create feature for word counts

```
word_counts <- numeric()
for (i in 1:nrow(tweets_df)){
        word_counts[i] <- length(tokens[[i]])
}
tweets_df$word_count <- word_counts
print(tweets_df$word_count[1:5])
```

```
## [1] 21 21 12 15 19
```

Create feature for unique word counts

```
unique_word_counts <- numeric()
for (i in 1:nrow(tweets_df)){
        unique_word_counts[i] <- length(unique(tokens[[i]]))
}
tweets_df$unique_word_counts <- unique_word_counts
print(tweets_df$unique_word_counts[1:5])
```

```
## [1] 19 19 11 14 18
```

We can now leverage these two features to create a new feature called word sophistication (#distinct words/#total words)

```
tweets_df$sophistication <- tweets_df$unique_word_counts/tweets_df$word_count
print(tweets_df$sophistication[1:5])
```

```
## [1] 0.9047619 0.9047619 0.9166667 0.9333333 0.9473684
```

Now that we have created features, we are now ready to begin exploring our data.

4. Exploratory analysis using dplyr and ggplot2

Context

After having created features and cleaning the data, I was now ready for exploratory analysis. Here, although I didn't have any specific hypotheses in mind yet about the nature of the #TimesUp movement, I saw this as an opportunity to uncover some interesting findings that might tell a story or converge on a common theme.

Methodology

I used the dplyr package to partition our data in various ways and summarize the partitions. I also used ggplot2 to visualize some patterns in the data.

Let's view the proportion of retweets across hours in the day

```
tweets_df %>%
        group_by(hour) %>%
        summarize(prop_retweets = mean(isRetweet)) %>%
        as.data.frame()
```

```
##    hour prop_retweets
## 1     0     0.7482993
## 2     1     0.7229730
## 3     2     0.7482993
## 4     3     0.6556291
## 5     4     0.7159091
## 6     5     0.6908397
## 7     6     0.7036011
## 8     7     0.7506775
## 9     8     0.6776860
## 10    9     0.6325967
## 11   10     0.5710145
## 12   11     0.6198157
## 13   12     0.6419214
## 14   13     0.6821516
## 15   14     0.6221080
## 16   15     0.6951754
## 17   16     0.7341513
## 18   17     0.7121212
## 19   18     0.7562814
## 20   19     0.7920228
## 21   20     0.7343284
## 22   21     0.7219512
## 23   22     0.8053435
## 24   23     0.7959184
```

There appears to be a greater proportion of retweets around 10:00 pm.

Let's also check the sentiment by hours in the day

```
tweets_df %>%
        group_by(hour) %>%
        summarize(mean_sent = mean(sent),
                  median_sent = median(sent)) %>%
        as.data.frame()
```

```
##    hour    mean_sent median_sent
## 1     0  0.028231293       0.000
## 2     1 -0.237837838      -0.275
## 3     2  0.003401361       0.000
## 4     3 -0.071523179       0.000
## 5     4  0.105965909       0.000
## 6     5 -0.167366412      -0.250
## 7     6 -0.129362881       0.000
## 8     7  0.055691057       0.000
## 9     8  0.084710744       0.000
## 10    9  0.125552486       0.000
## 11   10 -0.070579710       0.000
## 12   11  0.139861751       0.000
## 13   12  0.237991266       0.050
## 14   13  0.016136919       0.000
## 15   14  0.061439589       0.000
## 16   15 -0.019078947       0.000
## 17   16 -0.113292434      -0.400
## 18   17 -0.230303030      -0.400
## 19   18 -0.010804020      -0.025
## 20   19 -0.014529915       0.000
## 21   20 -0.055223881       0.000
## 22   21 -0.132439024       0.000
## 23   22  0.170992366       0.550
## 24   23  0.125255102       0.225
```

The most positive tweets come between 11am and 12pm. The most negative tweets come at 1 am and also at 5pm.

Finally let's see if there are any differences in tweet properties depending on whether it was retweeted.

```
tweets_df %>%
        group_by(isRetweet) %>%
        summarize(n_tweets=n(),
                  avg_nchar_tweet = mean(nchar_raw),
                  avg_nchar_clean_tweet = mean(nchar_clean),
                  avg_sent = mean(sent))
```

```
## # A tibble: 2 x 5
##   isRetweet n_tweets avg_nchar_tweet avg_nchar_clean_tweet avg_sent
##   <lgl>        <int>           <dbl>                 <dbl>    <dbl>
## 1 FALSE         2291            131.                  73.0  -0.0551
## 2 TRUE          5318            143.                  97.9   0.0272
```

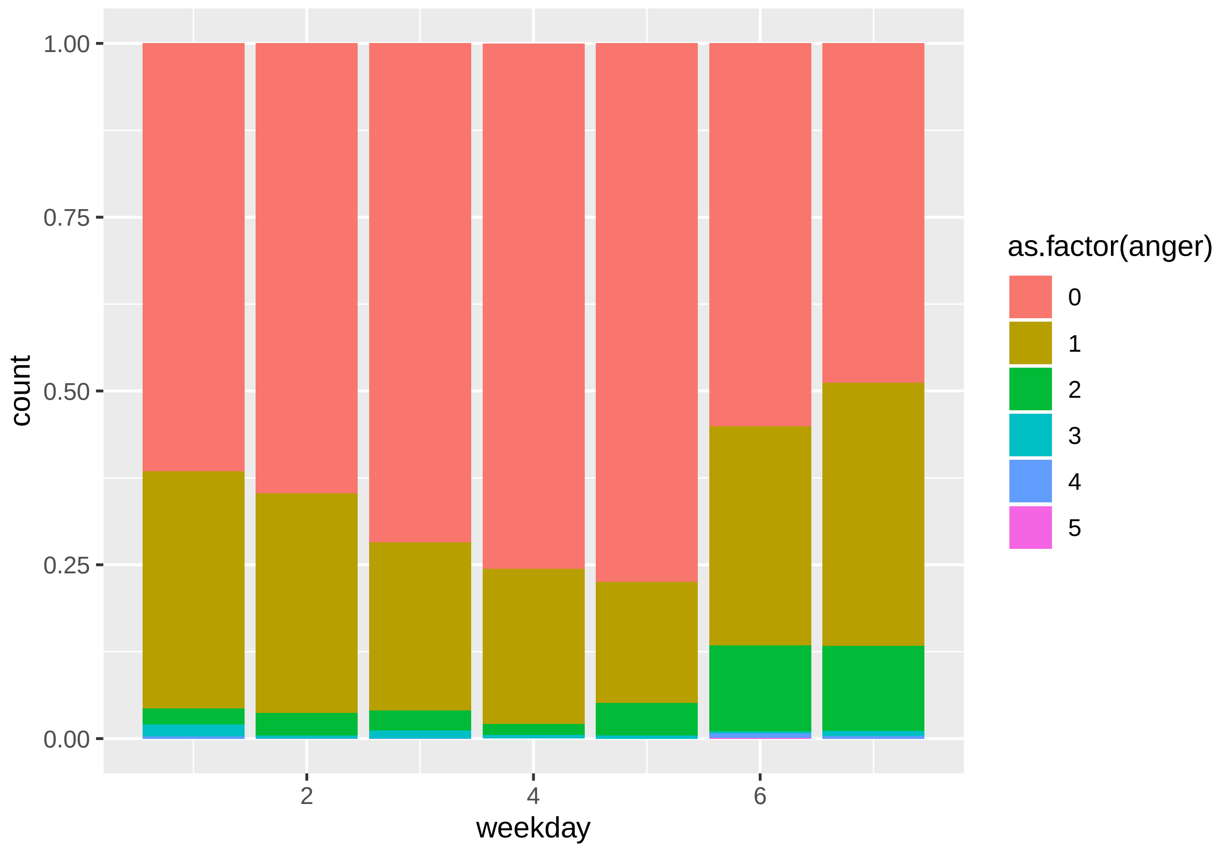
Here we can see some interesting differences. Retweets tend to have more characters and higher (more positive) sentiment.

Let's see if overall tweet emotions, specifically anger, vary across the week.

```
tweets_df %>%
        group_by(weekday) %>%
        summarize(m=mean(anger), n=n())
```

```
## # A tibble: 7 x 3
##   weekday     m     n
##     <dbl> <dbl> <int>
## 1       1 0.452   934
## 2       2 0.395  1318
## 3       3 0.335  1375
## 4       4 0.271  1214
## 5       5 0.282  1065
## 6       6 0.602   955
## 7       7 0.660   748
```

```
ggplot(data=tweets_df, aes(weekday))+
  geom_bar(aes(fill=as.factor(anger)), position="fill")
```



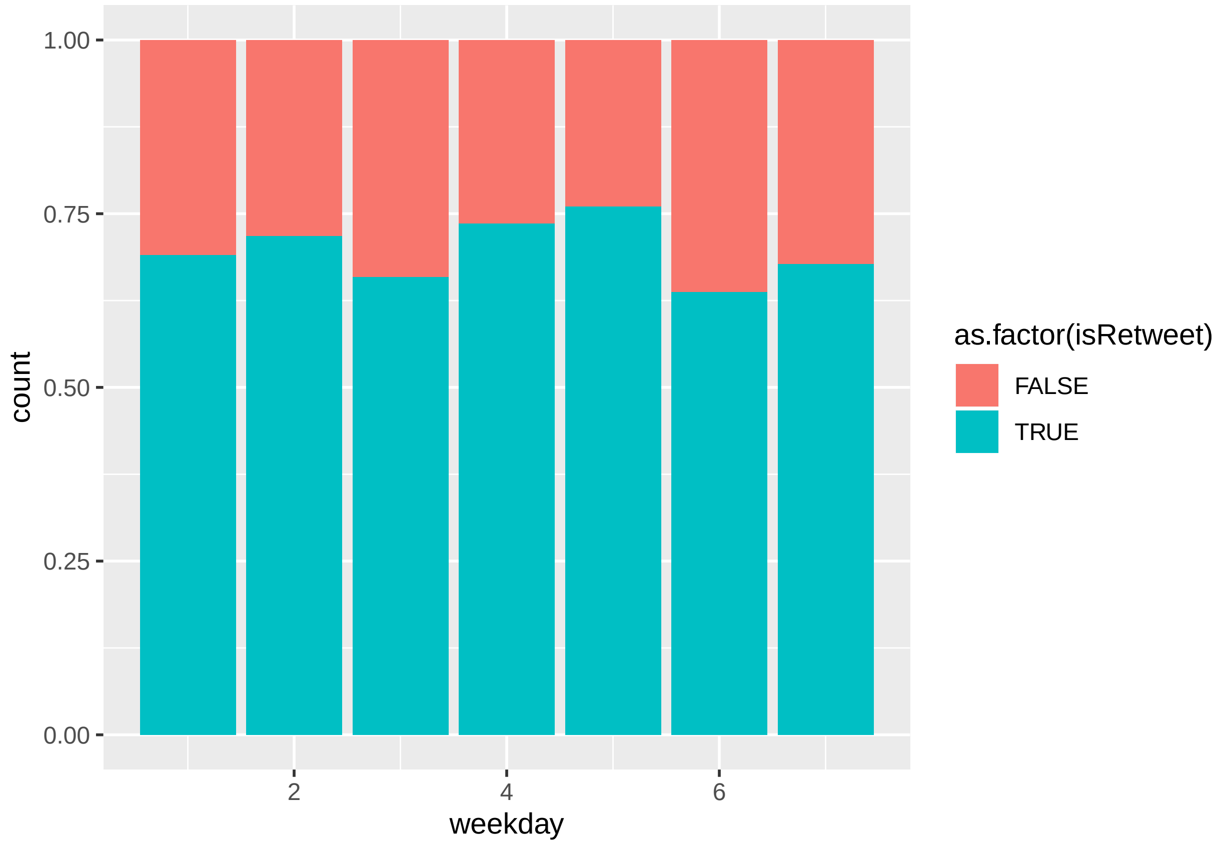
This table suggests that weekend tweets tend to be angrier. One could ask whether any other properties of tweets vary by weekday. Let's try checking the proportion of retweets across the week.

```
tweets_df %>%
        group_by(weekday) %>%
        summarize(m=mean(isRetweet), n=n())
```

```
## # A tibble: 7 x 3
##   weekday     m     n
##     <dbl> <dbl> <int>
## 1       1 0.691   934
## 2       2 0.719  1318
## 3       3 0.659  1375
## 4       4 0.736  1214
## 5       5 0.761  1065
## 6       6 0.638   955
## 7       7 0.678   748
```

This table suggests that there are relatively fewer retweets on weekends. Connecting the two above patterns, perhaps then, angry tweets are less likely to be retweeted. Let's explore that possibility.

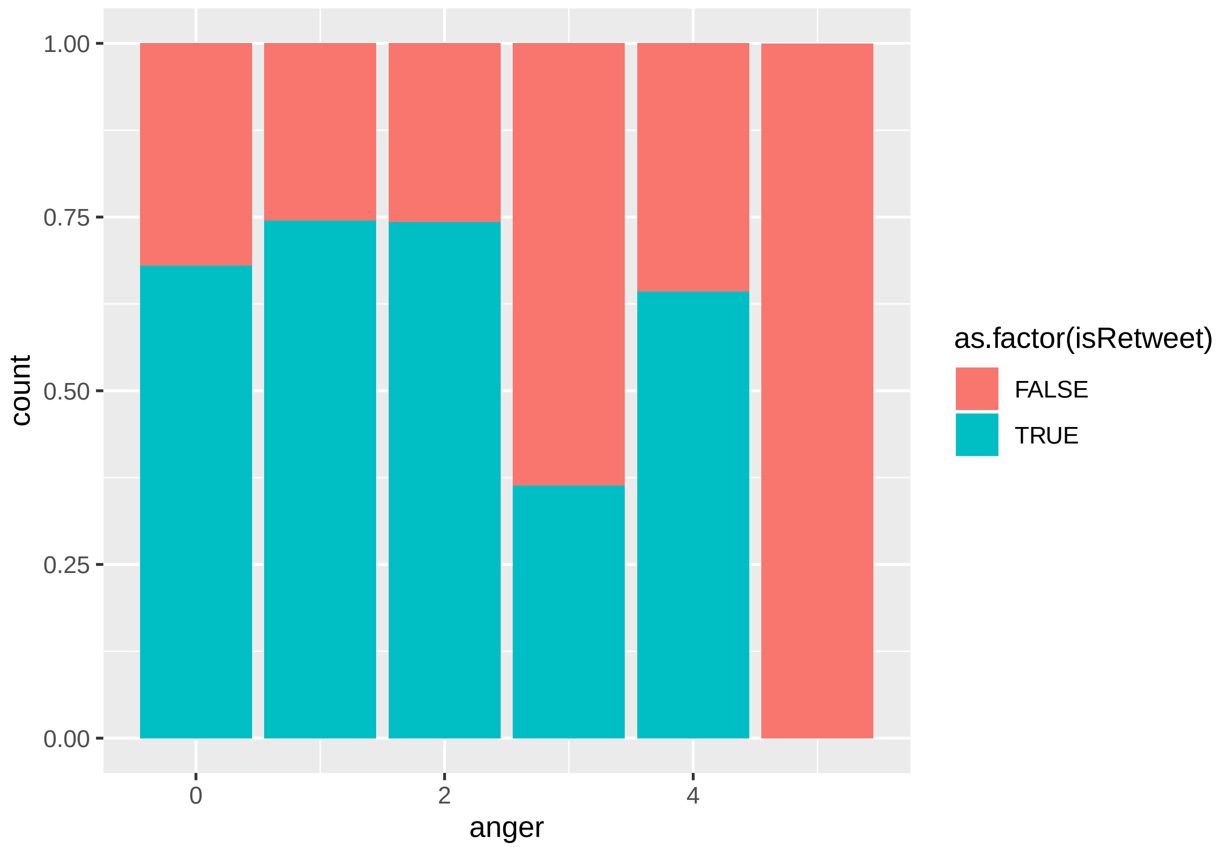
```r
ggplot(data=tweets_df, aes(weekday))+
  geom_bar(aes(fill=as.factor(isRetweet)), position="fill")
```



```r
tweets_df %>%
        group_by(anger) %>%
        summarize(m=mean(isRetweet), n=n())
```

```
## # A tibble: 6 x 3
##   anger     m     n
##   <dbl> <dbl> <int>
## 1     0 0.680  5048
## 2     1 0.745  2106
## 3     2 0.743   385
## 4     3 0.364    55
## 5     4 0.643    14
## 6     5 0         1
```

```r
ggplot(data=tweets_df, aes(anger))+
  geom_bar(aes(fill=as.factor(isRetweet)), position="fill")
```



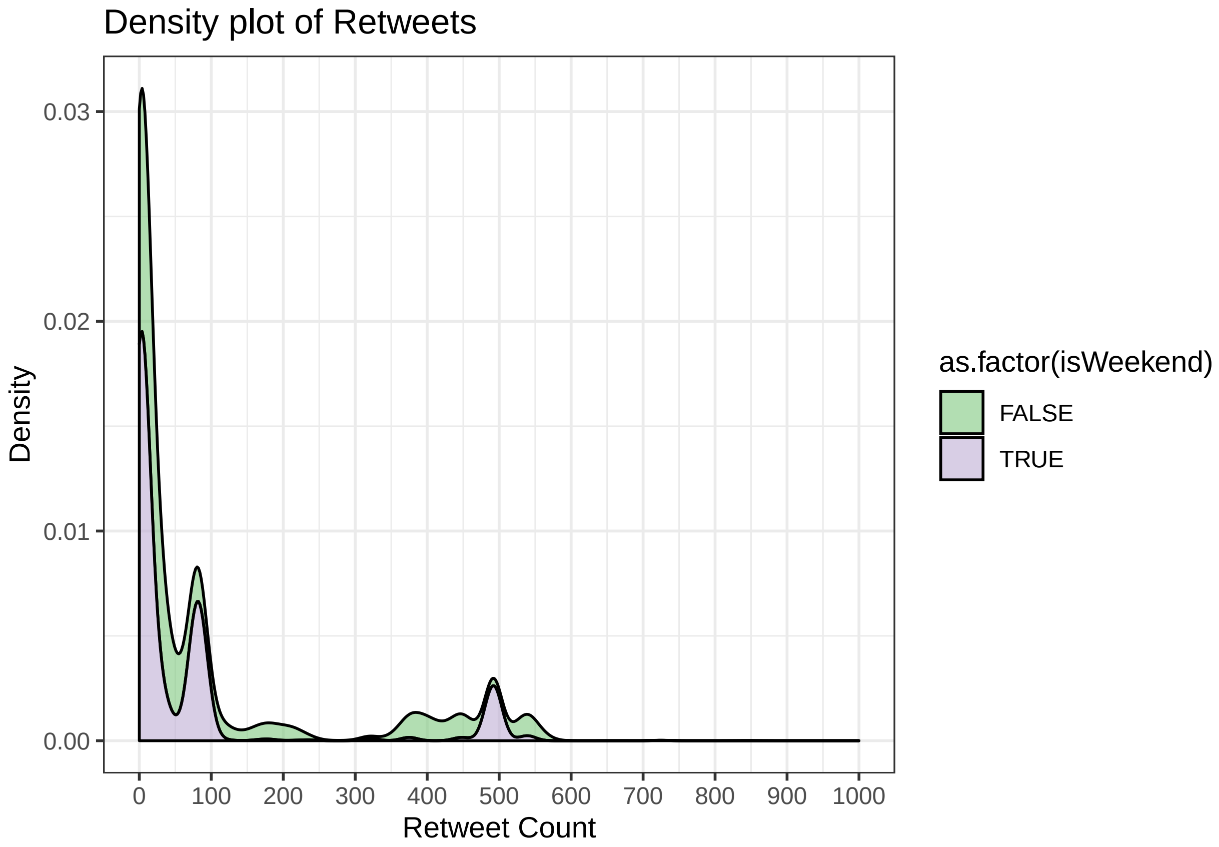
Here, we see that relative to unangry tweets (having an anger score of 0), tweets become more likely to be shared when there is a slight anger in them (with a score of 1 or 2), but much less likely to be shared when there is extreme anger (having a score of 3 or 4 or 5).

Perhaps an effective strategy for getting one's voice circulated on Twitter is to express slight, but not extreme anger.

Next let's see the densities of retweets by whether they occurred on a weekend.

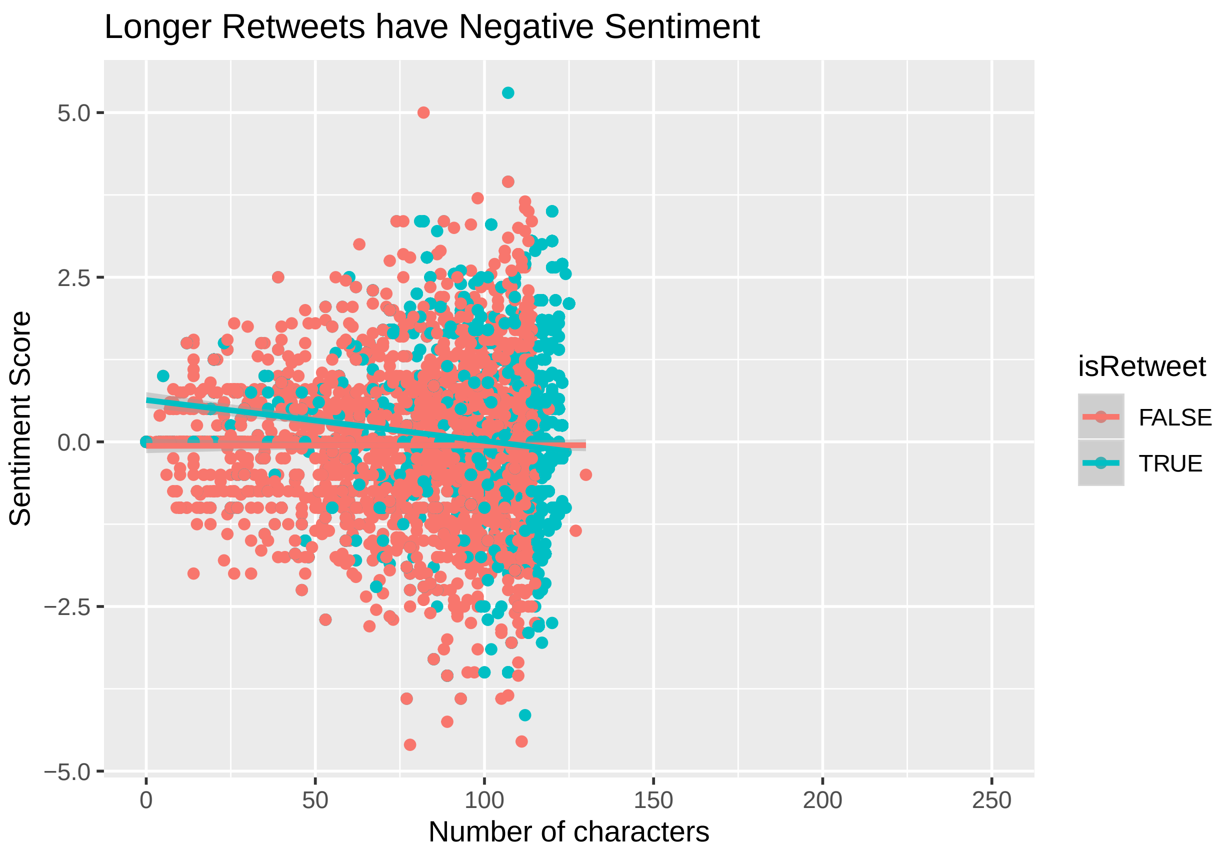
```
tweets_df$isWeekend <- as.factor(tweets_df$weekday %in% c(6, 7, 8))
```

```
## Warning: Removed 9 rows containing non-finite values (stat_density).
```



Here we can see the densities of retweet counts. Notice an overall similar pattern on weekends vs weekdays.

Next, let's check to see if there are any interesting linear relationships in the data.



It looks like there is a linear relationship between sentiment and the number of tweet characters. However, this may depend on whether it was a retweet, with retweets showing the negative linear trend, and non-retweets showing no linear trend. Let's see the summary report of the linear regression model on just retweets.

```
mod <- lm(data = subset(tweets_df, isRetweet=="TRUE"), sent~nchar_clean)
summary(mod)
```

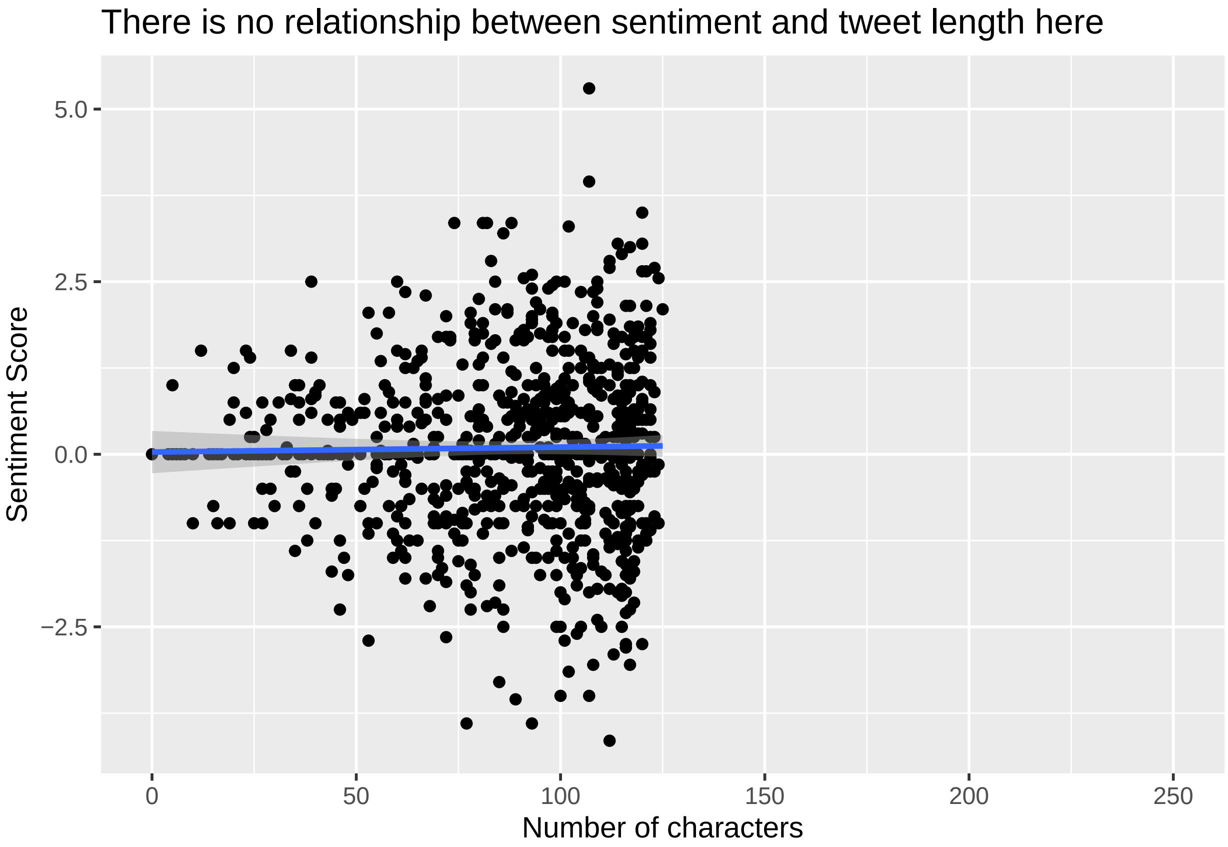
```
##
## Call:
## lm(formula = sent ~ nchar_clean, data = subset(tweets_df, isRetweet ==
##     "TRUE"))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0896 -0.7938 -0.1213  0.6039  5.3294
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.6347984  0.0612784   10.36   <2e-16 ***
## nchar_clean -0.0062072  0.0006059  -10.24   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 1.122 on 5316 degrees of freedom
## Multiple R-squared:  0.01936,    Adjusted R-squared:  0.01917
## F-statistic: 104.9 on 1 and 5316 DF,  p-value: < 2.2e-16
```

Indeed, the linear relationship is highly significant with a p-value of $< .0001$. However, the R-squared close to 0 indicates that the variables probably don't have a linear relationship. Furthermore, one core assumption of regression is that rows are independent. Here, we have many duplicate rows due to the fact that these are retweets. So let's analyze the relationship again after subsetting just to unique tweets.

```
deduped.data <- unique(subset(tweets_df, isRetweet=="TRUE")[ , c('nchar_clean','sent')] )
mod2 <- lm(data=deduped.data, sent~nchar_clean)
summary(mod2)
```

```
##
## Call:
## lm(formula = sent ~ nchar_clean, data = deduped.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2617 -0.8420 -0.0527  0.7664  5.1918
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.0314795  0.1558290   0.202    0.840
## nchar_clean 0.0007166  0.0016932   0.423    0.672
##
## Residual standard error: 1.306 on 728 degrees of freedom
## Multiple R-squared:  0.000246,   Adjusted R-squared:  -0.001127
## F-statistic: 0.1791 on 1 and 728 DF,  p-value: 0.6723
```



This suggests the previous linear trend could have been driven by a tweet or tweets that was frequently retweeted and also had few characters and positive sentiment.

Conclusions

Taken together, we have learned quite a bit about how people are engaging with the #TimesUp movement on Twitter.

- Around 10 pm, there is a slightly greater proportion of retweets, which suggests that Twitter users who are interested in sharing TimesUp content might be more of the night-owl variety.
- The most positive tweets come between 11am and 12pm. The most negative tweets come at 1am, as well as at 5pm. Although there could be multiple explanations, one possible reason for this effect is that people are stressed out immediately after work hours, as well as late at night, when they experience insomnia.
- Retweets tend to have more characters and higher (more positive) sentiment. This could suggest a strategy for people wanting their tweet to go viral. Namely, it could pay off to develop a more thought-out tweet and to keep the sentiment more positive than negative.
- I also found that weekend tweets tend to be angrier and also retweeted less often. More nuanced breakdown of the retweet by anger relationship suggests there is an optimal level of anger, such that tweets that are slightly, but not extremely angry, are more likely to be shared than unangry tweets. Considering this, to maximize the chance of a retweet, it may be well-advised to express slight, but not extreme anger.

5. Cluster analysis to detect common themes in the text

Cluster analysis is a form of unsupervised machine learning in which the algorithms will find common latent structures in the data. Here, in this case, we will apply clustering algorithms to find common text structures, like semantic themes. We'll try different clustering algorithms, such as K-Means and Latent Dirichlet Allocation.

```
## Warning: package 'tm' was built under R version 3.6.1
```

```
## Warning: package 'wordcloud' was built under R version 3.6.1
```

```r
input_text <- tweets_df$cleaned_text
```

Context

To do various machine learning operations, we will need to preprocess the text data further than we already have. The first step in R is to create a Corpus object from the tm package.

```r
# make a corpus object
Corpus <- Corpus(VectorSource(input_text))
```

One common preprocessing step is to remove words that occur too frequently to be meaningful, such as 'a' and 'the'. These are called 'stopwords'

```r
Corpus = tm_map(Corpus, removeWords, stopwords("english"))
```

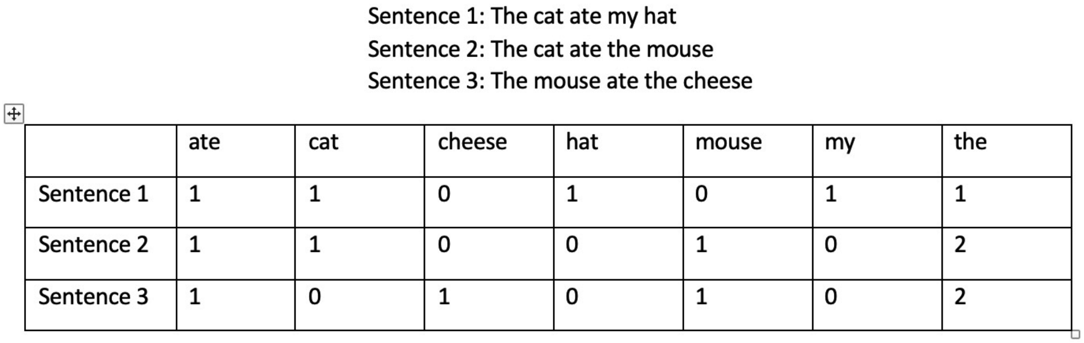
We can also define our own unique stopword list and remove all words from that list.

```r
customStopwords <- c("can", "say","one","way","use",
                     "also","howev","tell","will",
                     "much","need","take","tend","even",
                     "like","particular","rather","said",
                     "get","well","make","ask","come","end",
                     "first","two","help","often","may",
                     "might","see","someth","thing","point",
                     "post","look","right","now","think","’ve ","’re ")

#remove custom stopwords
Corpus <- tm_map(Corpus, removeWords, customStopwords)
```

Next, we can also stem the documents. This will reduce various versions of a word, such as 'loving', 'loves', 'loved' to just the stem 'lov'. This will help us condense our signal as we prepare to build a model.

```r
Corpus = tm_map(Corpus, stemDocument)
```

#####DTM Now that we have preprocessed the data, we can take the next step to leverage more sophisticated techniques by representing the text itself numerically. To this end, a standard NLP method is to convert the documents (in this case, tweets) into a sparse vector of numbers. The most common version of this method is called ‘Bag of Words’, in which each row represents a tweet and each column represents a word in the Corpus vocabulary. It can be demonstrated as follows:



Sentence 1: The cat ate my hat
Sentence 2: The cat ate the mouse
Sentence 3: The mouse ate the cheese

	ate	cat	cheese	hat	mouse	my	the
Sentence 1	1	1	0	1	0	1	1
Sentence 2	1	1	0	0	1	0	2
Sentence 3	1	0	1	0	1	0	2

```r
# Create sparse document/term matrix
DTM <- DocumentTermMatrix(Corpus)
inspect(DTM)
```

```
## <<DocumentTermMatrix (documents: 7609, terms: 4434)>>
## Non-/sparse entries: 62781/33675525
## Sparsity           : 100%
## Maximal term length: 33
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   adrienn bennett black harass master meet plumber sexual woman women
##   1188       0       0     0      1      0    0       0      1     0     0
##   1501       0       0     0      0      0    0       0      2     0     0
##   1536       0       0     0      0      0    0       0      0     0     1
##   1829       0       0     0      0      0    0       0      0     0     2
##   2234       0       0     0      0      0    0       0      0     0     2
##   3021       0       0     0      0      0    0       0      0     0     2
##   3228       0       0     0      0      0    0       0      0     0     2
##   3278       0       0     0      0      0    0       0      0     0     2
##   3428       0       0     0      0      0    0       0      0     0     1
##   3602       0       0     0      0      0    0       0      0     0     1
```

We can now call functions on our DTM like to find the most frequent words.

```r
findFreqTerms(DTM, 1000)
```

```
## [1] "sexual" "harass" "women"
```

#####KMeans Next, to prepare our data for the K-means clustering algorithm, we must convert our DTM to a standard matrix format.

```r
m <- as.matrix(DTM)
```

Then we must create a matrix that stores the pairwise distances among all our document-term vectors.

```r
d <- dist(m)
```

First we'll test our clustering algorithm with just two clusters.

```r
kfit <- kmeans(d, centers=2, nstart=1)
```

Here we can see properties of our iteration, such as the total within sums of squares and between sums of squares.

```
## [1] "The within SS values for our 2 clusters are 5374785.07689646 and 10009257.1817719"
```

```
## [1] "The between SS for our clusters is 6867058.01116788"
```

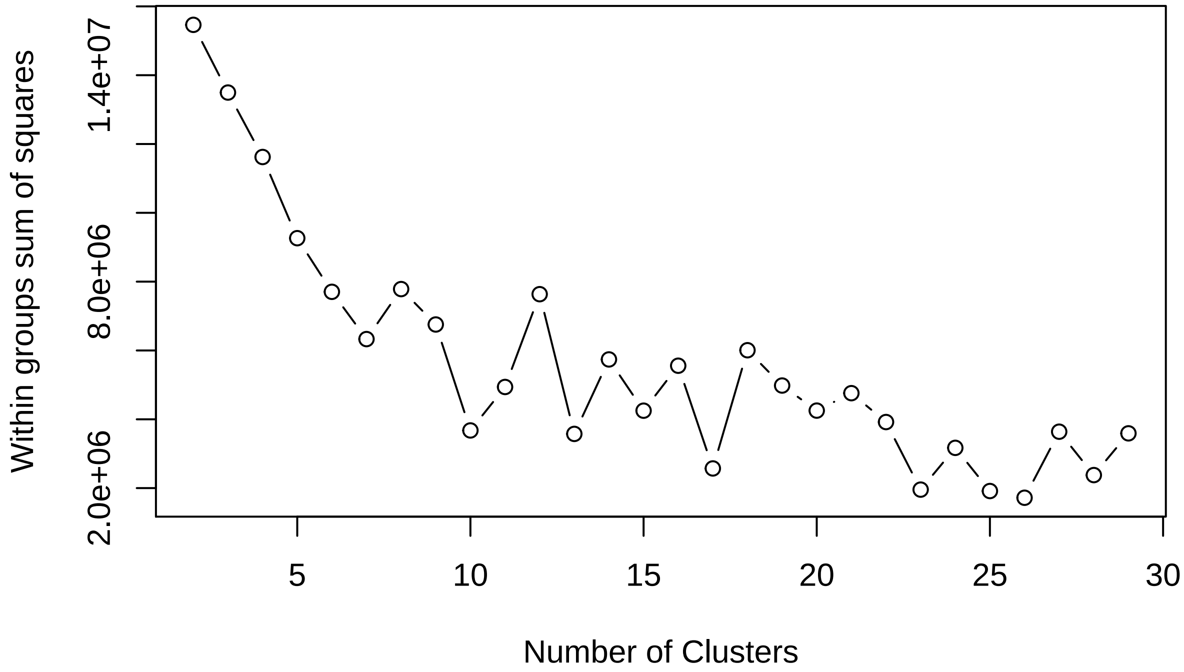
But we can specify any number of n possible clusters for n points in our data. How can we tell which is the optimal amount? One thing we can do is to try many iterations varying K each iteration and keep track of which K gives us the best tradeoff of low K values and low within SS.

We will now test our K-means algorithm across 28 different levels of K. We will store our results in a results bin on each iteration.

```r
k_results_bin <- data.frame(k=2:29,
                            withinss=numeric(28))

for (i in 2:29) {
        print(paste('On iteration:', i))
        kfit <- kmeans(d,centers=i,nstart=1)
        k_results_bin[i-1, 'withinss'] <- sum(kfit$withinss)
}
```

Next we can visualize our results using an Elbow plot

```r
plot(2:29, k_results_bin[1:28, 'withinss'],
     type="b", xlab="Number of Clusters",ylab="Within groups sum of squares")
```



Here, it is possible to visually determine the best value for K. Along the line chart, there appears to be a strong inflection point at K=9, therefore it looks like K = 9 is a reasonable choice for our optimal number of clusters. A copy of the line chart that is easier to view and less compressed horizontally can be found here: https://github.com/jchicchi/NLP-Project-TimesUp/

```
set.seed(42)
best_kfit <- kmeans(d,centers=9,nstart=1)
```

Let's see how many documents belong to each cluster.

```
table(best_kfit$cluster)
```

```
##
##    1    2    3    4    5    6    7    8    9
##  233 2004  283  293 1241   65  291 2520  679
```

Next let's assign each tweet to its corresponding cluster. This way we can more easily describe the themes in our clusters.

```
tweets_df$clusters <- as.numeric(best_kfit$cluster)
```

We can now filter the data to just tweets belonging to each cluster.

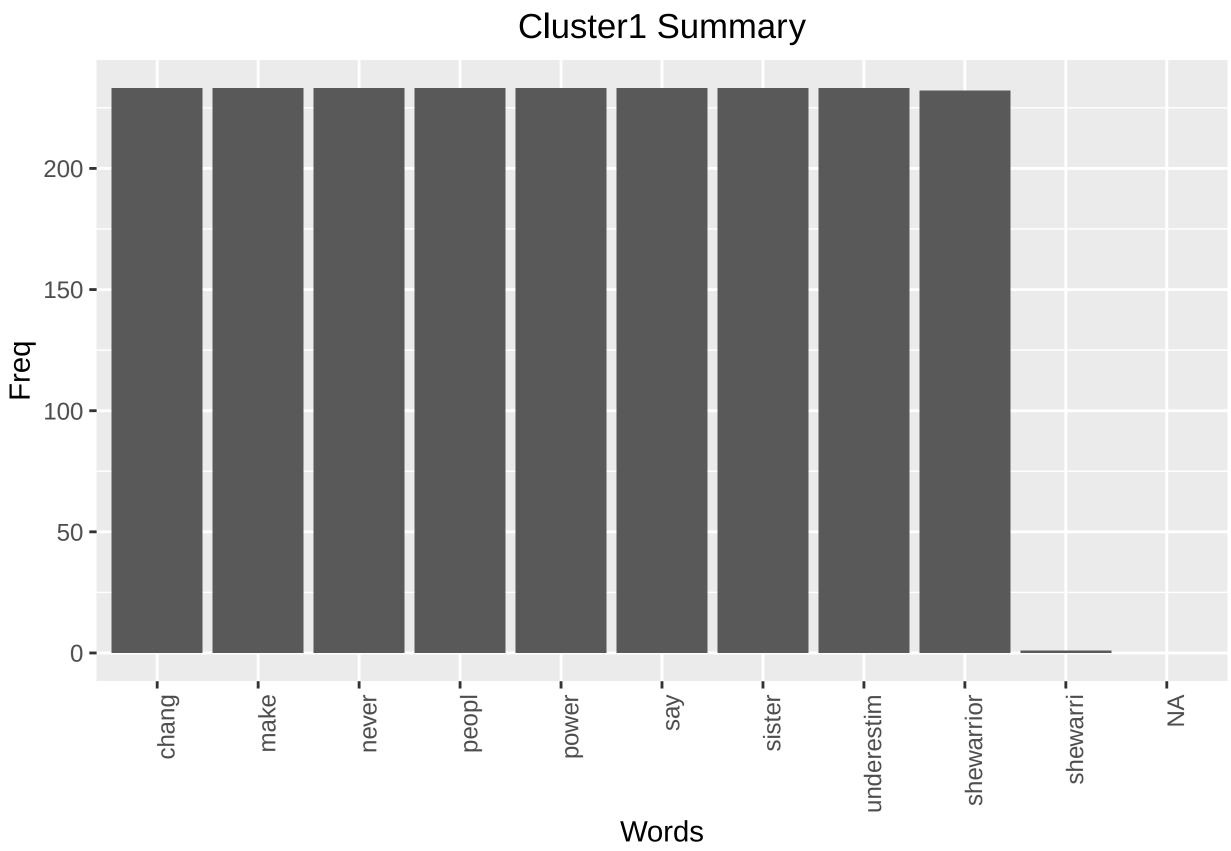
```
clust1 <- tweets_df[tweets_df$clusters == 1, ]
clust2 <- tweets_df[tweets_df$clusters == 2, ]
clust3 <- tweets_df[tweets_df$clusters == 3, ]
clust4 <- tweets_df[tweets_df$clusters == 4, ]
clust5 <- tweets_df[tweets_df$clusters == 5, ]
clust6 <- tweets_df[tweets_df$clusters == 6, ]
clust7 <- tweets_df[tweets_df$clusters == 7, ]
clust8 <- tweets_df[tweets_df$clusters == 8, ]
clust9 <- tweets_df[tweets_df$clusters == 9, ]
```

Next, to describe our clusters, let's build a function that will return the top n words within a text column. We can then apply this function across our cluster segments.

```
# Write a function for the above syntax
summarize_text <- function(a_col, nwords=20){
        xCorpus <- Corpus(VectorSource(a_col)) # make a corpus object
        xCorpus = tm_map(xCorpus, removeWords, stopwords("english"))
        xCorpus = tm_map(xCorpus, stemDocument)
        xDTM <- DocumentTermMatrix(xCorpus)
        text_summary <- sort(slam::col_sums(xDTM), decreasing=TRUE)[1:nwords]

        return(text_summary)
}
```

Here we can see the top 20 words in cluster 1

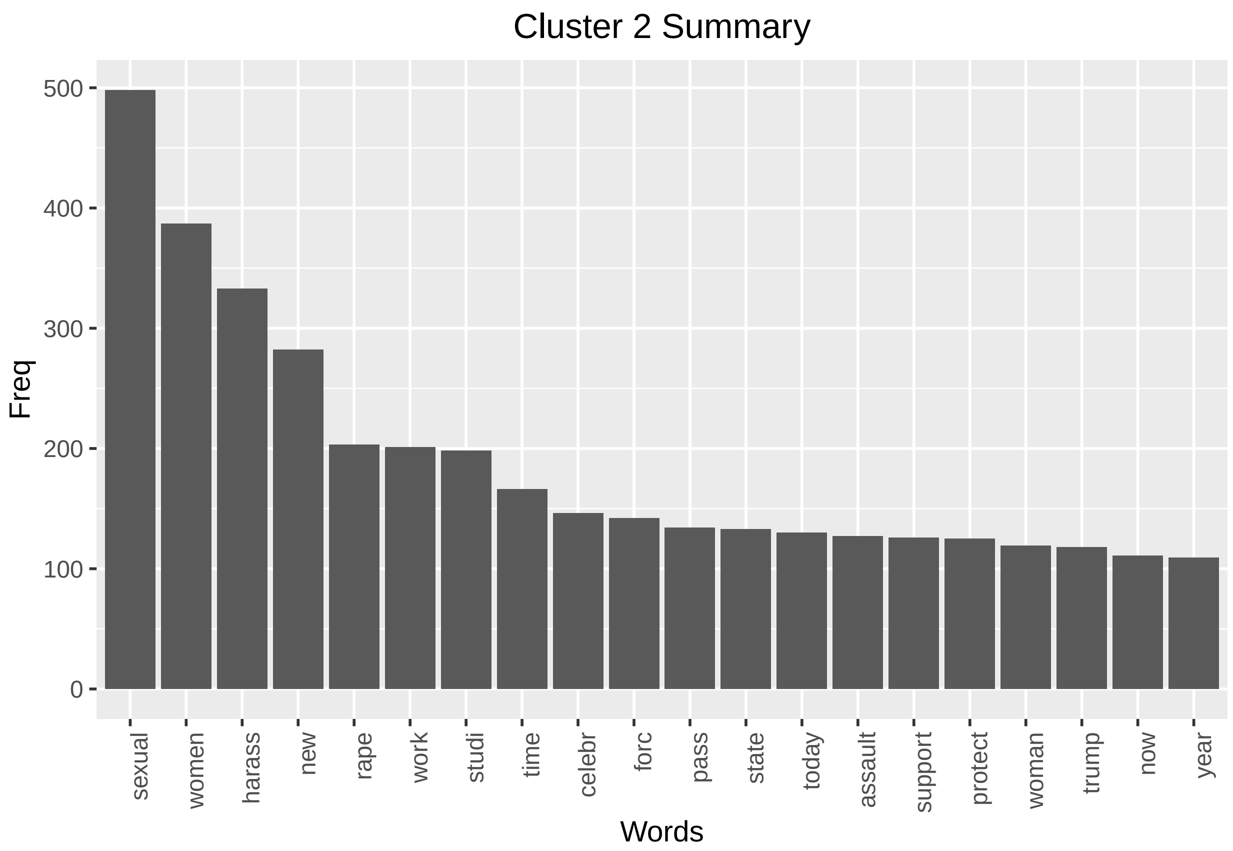
```
clust1_summary <- summarize_text(a_col=clust1$cleaned_text, nwords=20)
clust1_df <- data.frame(words=names(clust1_summary),
                        freq=clust1_summary)

ggplot(clust1_df, aes(x = reorder(words, -freq), y = freq)) +
        geom_bar(stat = "identity") +
        theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
        labs(x='Words', y='Freq', title='Cluster1 Summary') +
        theme(plot.title = element_text(hjust = 0.5))
```



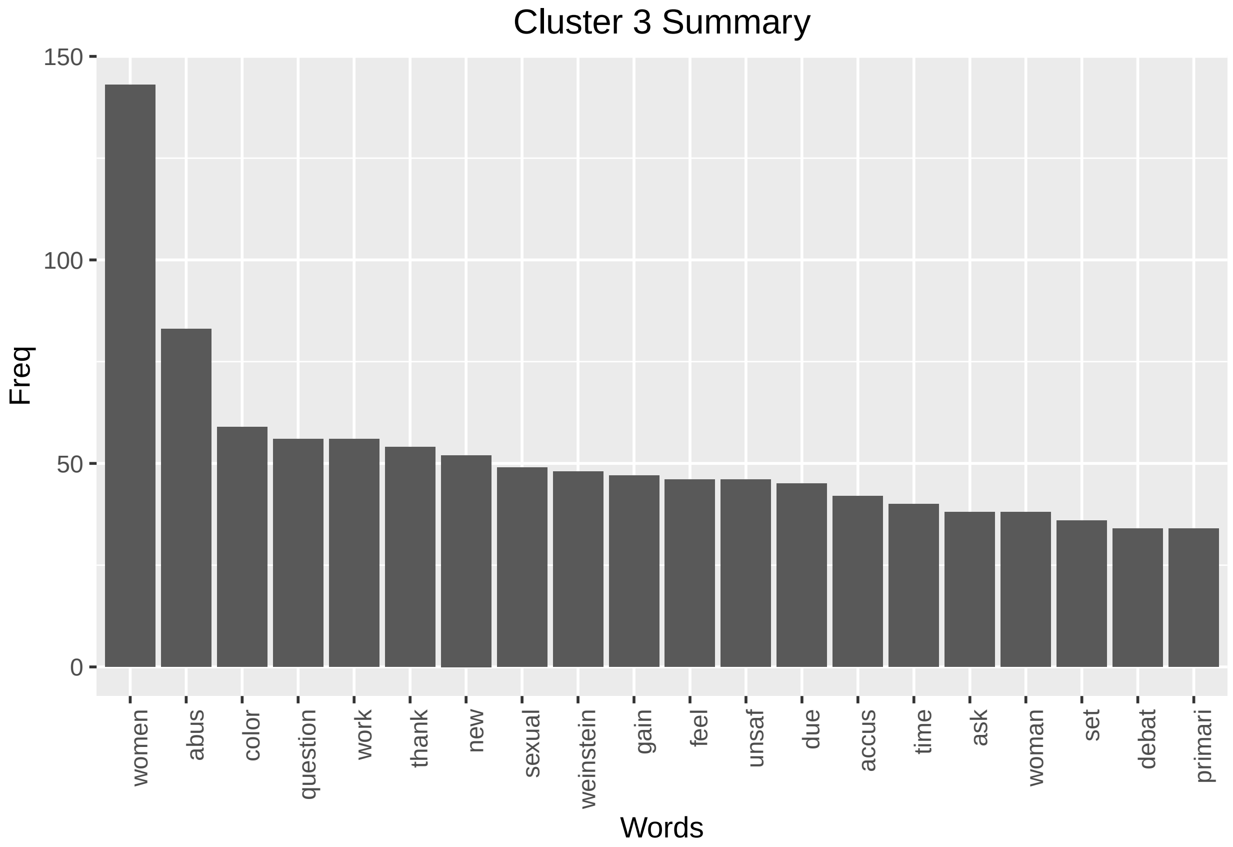
This cluster seems to be about strong women and empowerment.

We can wrap the above code into a function and apply it to the remaining clusters.

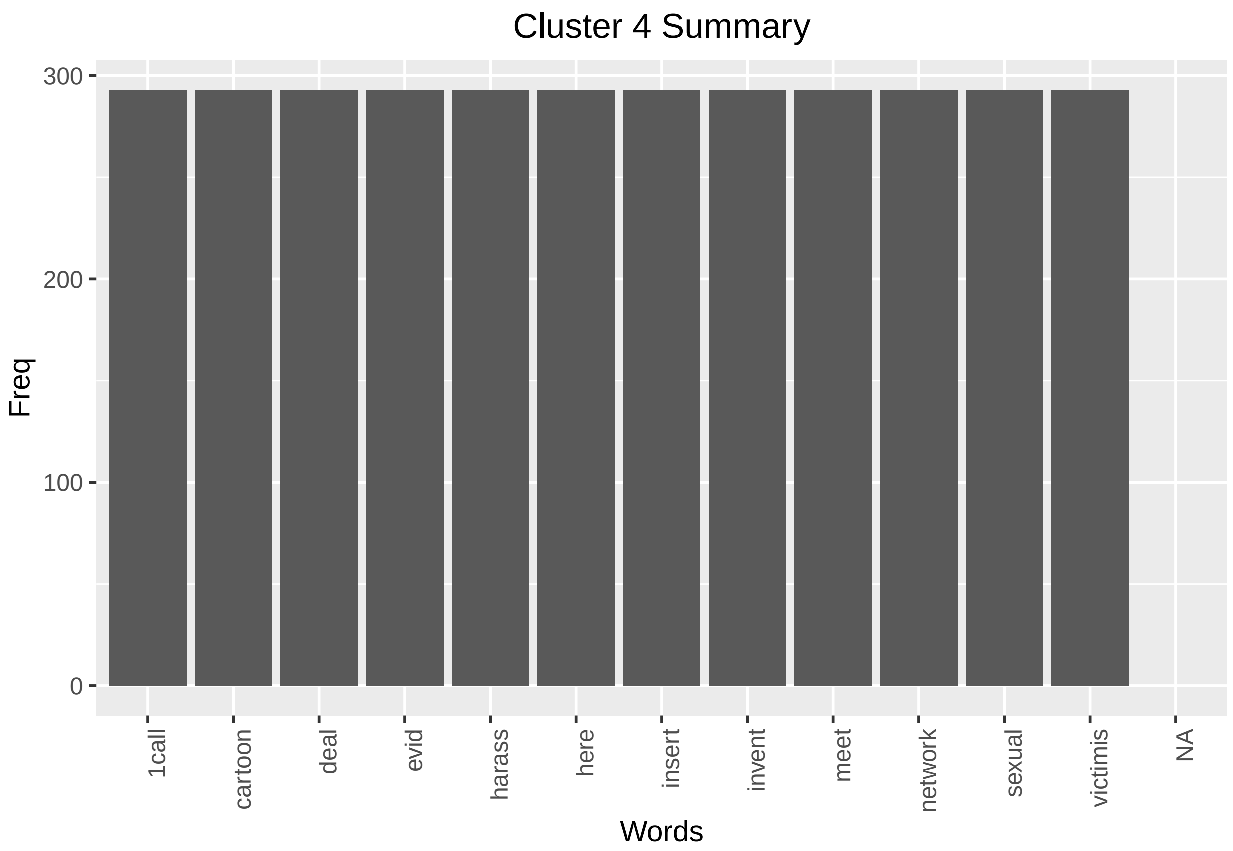
```
plot_cluster_summary <- function(a_col, clust_num){
        clust_sum <- summarize_text(a_col=a_col, nwords=20)
        clust_df <- data.frame(words=names(clust_sum),
                               freq=clust_sum)

        p1 <- ggplot(clust_df, aes(x = reorder(words, -freq), y = freq)) + geom_bar(stat = "identity") +

        p1
}
```



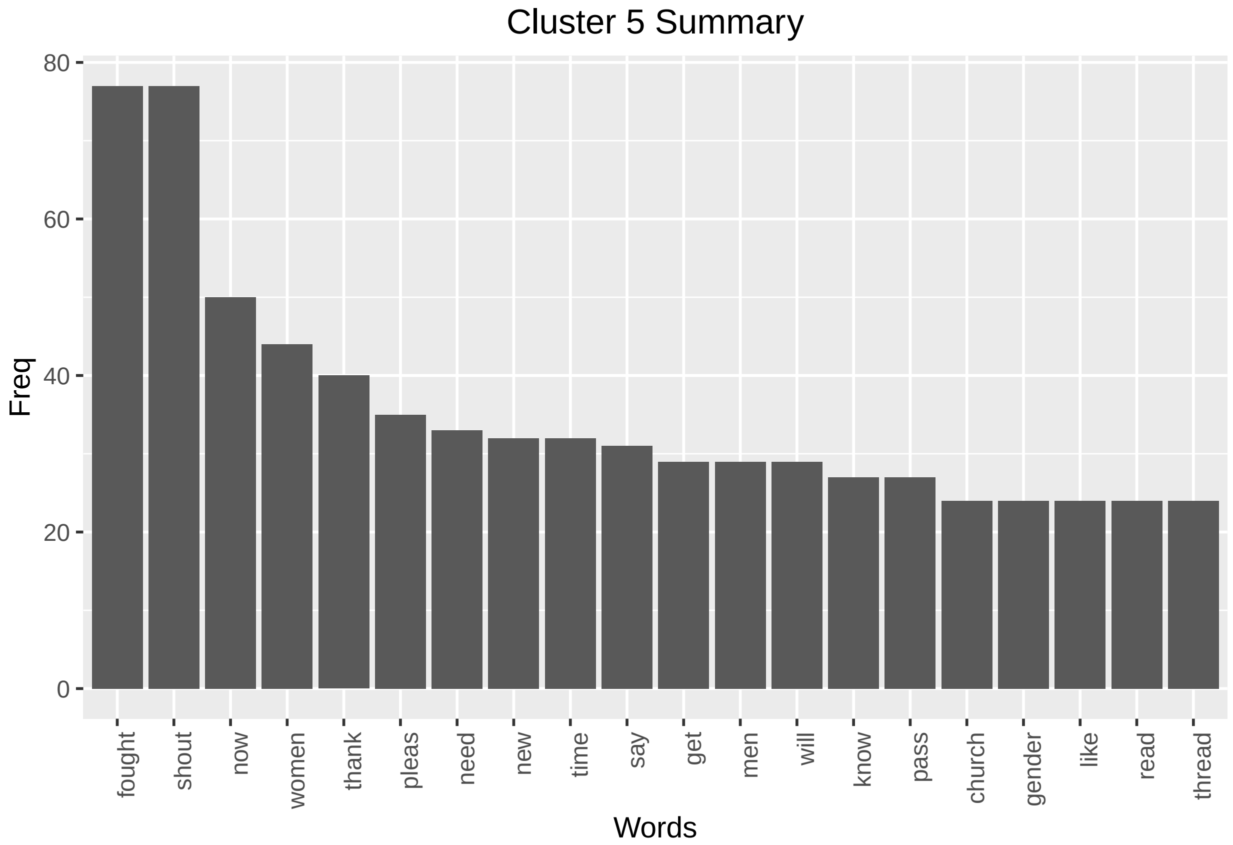
This cluster seems to reflect general themes about the TimesUp movement, like ‘women’, ‘sexual’, ‘harass’, ‘work’, ‘rape’, and ‘assault’, etc. It also has ‘Trump’ in it, presumably because of some past comments he has made about women.



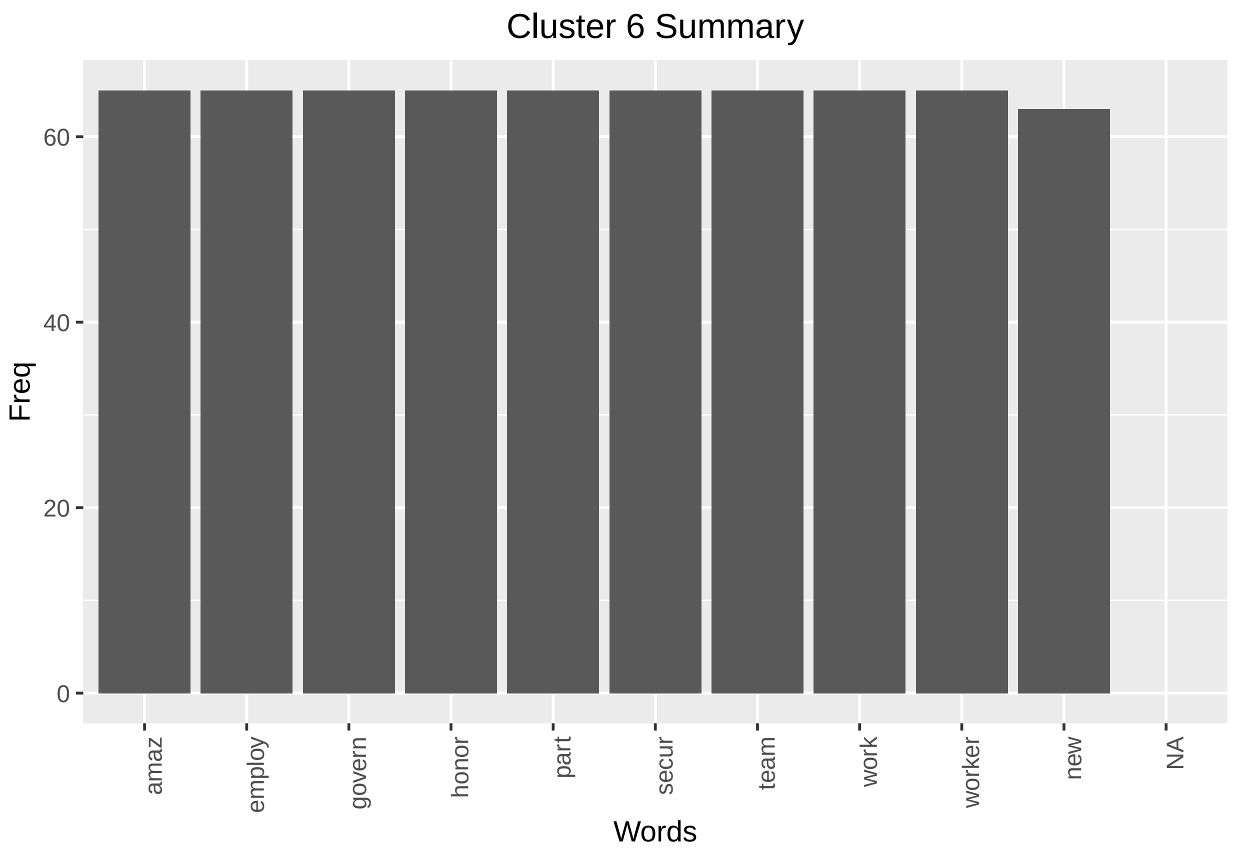
This cluster theme is less clear. Perhaps it is reflecting a discussion of alleged perpetrators, such as ‘Weinstein’.



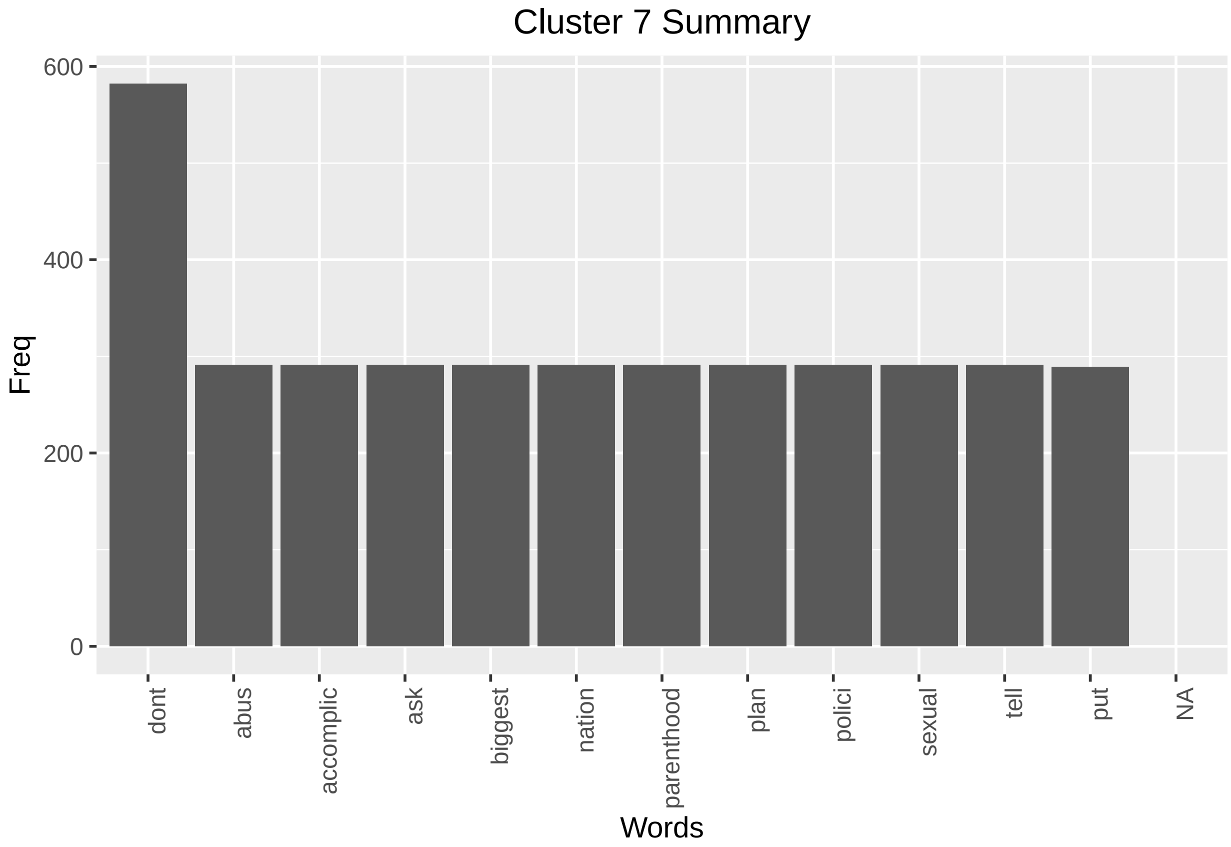
The frequency distribution of this cluster's words is uniform, suggesting that it perhaps reflects one retweet.



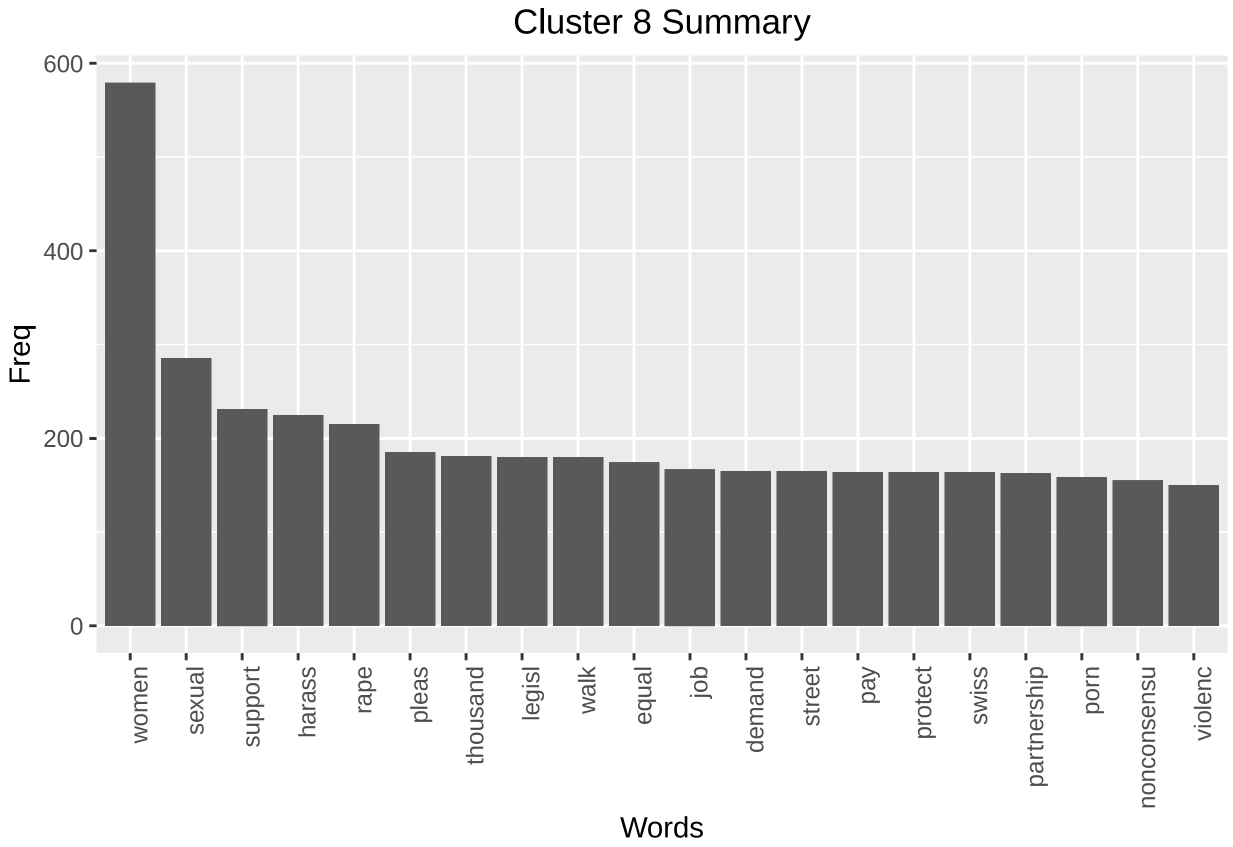
This cluster seems to reflect calls to action, such as ‘fought’ and ‘shout’.



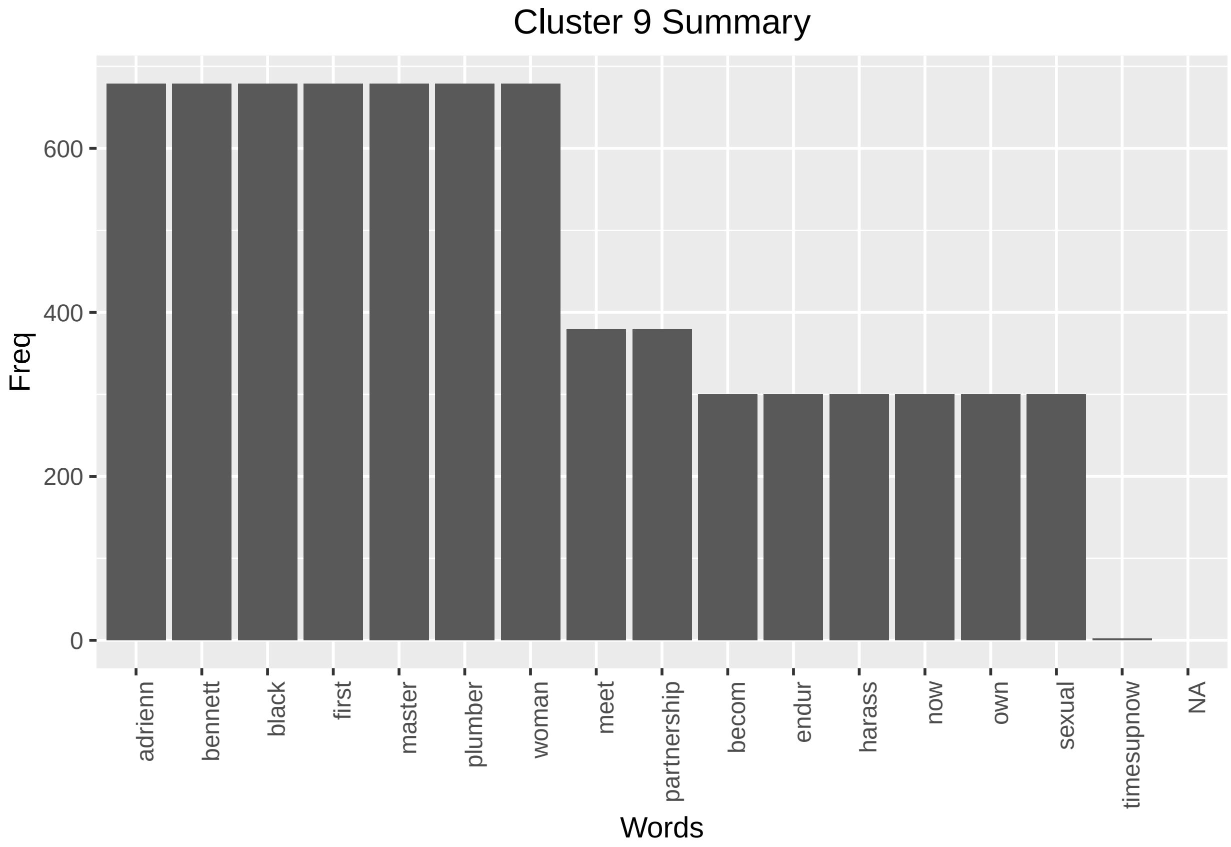
This cluster seems to reflect a political component, with the word 'govern appearing here.



This cluster is picking up tweets related to Planned Parenthood.



This cluster theme is also not immediately clear, but appears to tap themes about equal pay and jobs.



This cluster is picking up tweets related to Adrienne Bennett, the first black female master plumber in Michigan.

Summary

In sum, we can use cluster analysis like K-means to find common themes in our tweet data.

Topic Modeling (LDA)

```r
library(topicmodels)
```

```
## Warning: package 'topicmodels' was built under R version 3.6.1
```

Topic modeling is another powerful clustering algorithm that decomposes our DTM into count matrices that we can use to estimate word distributions for each topic.

The broad goals of Topic Modeling (LDA): 1. Estimate a distribution of topics over our documents (K) 2. Estimate a distribution of words over topics

In LDA, we consider documents mixtures of topics. This is distinct from K-means (in which we consider a document as belonging to only 1 cluster)

We will use a process called Gibb's sampling to estimate our distributions.

```
#Set parameters for Gibbs sampling
burnin <- 4000
iter <- 2000
thin <- 500
seed<-list(2003,5,63,100001,765) # note, second param must be equal to nstart
nstart <- 5
best <- TRUE
```

We'll set our number of topics to be 10. Initially, I tried setting the number of topics to 9 to match the number of clusters in k-means, but found that I got better results with 10 topics.

```
#Number of topics
k <- 10
```

Because of stopword and stem removal, some documents have 0 words. Let's remove these before we run topic modeling

```
raw.sum=apply(DTM,1,FUN=sum) #sum by raw each raw of the table
DTM=DTM[raw.sum!=0,]
```

Here we will run the algorithm.

```
set.seed(42)
ldaOut <-LDA(DTM,k, method="Gibbs",
control=list(nstart=nstart, seed = seed, best=best, burnin = burnin, iter = iter, thin=thin))
```

We can see the count of documents by topic

```
ldaOut.topics <- as.matrix(topics(ldaOut))
```

```
print(table(ldaOut.topics))
```

```
## ldaOut.topics
##   1   2   3   4   5   6   7   8   9  10
## 764 955 873 804 570 720 840 509 798 689
```

Importantly, we can also view the word distribution per topic.

```
ldaOut.terms <- as.matrix(terms(ldaOut,20))
```

Topic 1

```
ldaOut.terms[,1]
```

```
##  [1] women       support     protect     pleas       work        legisl      porn        nonconsensu
## Levels: addit commit due huge jour kudo legisl lisa nonconsensu pleas porn prof protect public reckon
```

This topic seems to reflect social justice in general, and is slightly easier to see in the table below.

```
ldaOut.terms[,1:5]
```

```
##        Topic.1  Topic.2  Topic.3   Topic.4     Topic.5
## 1        women     time     rape      work        dont
## 2      support      new  assault   violenc        abus
## 3      protect     pass    thank    worker      sexual
## 4        pleas    state    trump     studi        plan
## 5         work     just   carrol     adopt  parenthood
## 6       legisl     know     jean     accus      nation
## 7         porn   report survivor     great         put
## 8  nonconsensu     rape   celebr      news      polici
## 9          sex     york    alleg       new     biggest
## 10         due      run    month   convent    accomplic
## 11        huge    proud   togeth      show    thousand
## 12      public   presid     step       day         pay
## 13      commit     bill    femal     today       equal
## 14        kudo    major    allow recommend         job
## 15        lisa     make    polic     guess        walk
## 16      wilson   justic  forward     chanc      demand
## 17        prof     good     forc    combat      street
## 18      reckon      rep     told    treati       swiss
## 19        jour   incred      ill   confirm partnership
## 20       addit introduc    billi   statist        keep
```

- Topic 2 could reflect something about passing a bill. It seems moderately political.
- Topic 3 appears to be more about issues of physical safety, like rape and assault.
- Topic 4 looks like it reflects more about issues at work.
- Topic 5 could be a mixture of themes, here some about Planned Parenthood, some about abuse, and some about equal pay in the workforce.

```
ldaOut.terms[,6:10]
```

```
##        Topic.6      Topic.7   Topic.8   Topic.9   Topic.10
## 1        woman       gender    sexual    victim      power
## 2        black     question    harass       men      peopl
## 3      bennett          let      meet      want      chang
## 4       master       church      here      mani      never
## 5       plumber         call      deal     world     sister
## 6       adrienn        color      evid      girl underestim
## 7   partnership      compani     1call     shame shewarrior
## 8         endur         made    insert       get     employ
## 9         becom     industri    invent     start   movement
## 10          own        stori   network      read       year
## 11       harass         issu  victimis    believ       team
## 12         meet          ask   cartoon    rapist       talk
## 13       sexual         feel    fought     still     govern
## 14     dickhead relationship discrimin contribut      speak
## 15        night         care     natur     theyr       part
## 16       thrill        debat    segreg      your       amaz
## 17        parti       friend    racial    crimin        lot
## 18  antiharass        share   stigmat    happen      honor
## 19       bright         john      hair    moment      secur
## 20      medicin        watch     shout       peo        old
```

- Topic 6 overlaps with one of our clusters from K-means, focusing on Adrienne Bennett.
- Topic 7 is interesting. It may include themes about gender and church, together or individually. Either way, this is a distinct cluster compared to what we found in K-means.
- Topic 8 may suggest themes about sexual harassment and also the 'cartoon' and '1call' tweet that showed up in Cluster 4 of K-means.
- Topic 9 is not entirely clear, but suggests themes of the trauma of victimization.
- Topic 10 suggests themes about women empowerment and social change.

Conclusions

Both K-Means and topic modeling (LDA) gave us insight into common themes that are showing up in the text data. Issues of harassment, abuse, empowerment, and social justice are all present in both analyses.

Final Conclusions

Our analysis has demonstrated many advanced text analytics methods and machine learning techniques that we can leverage to better understand the #TimesUp movement. In our exploratory data analysis, we found patterns that suggested possible actions we could take to increase one's impact in the #TimesUp movement online. The cluster analysis showed us some common themes that people are tweeting about. Taken together, one could tweet about such themes, while also leveraging the possible strategies we found in the earlier analysis.