deletweet-introduction

April 4, 2017

1 DELETWEET

1.0.1 APPLYING TEXT MINING AND A RECOMMENDATION SYSTEM TO POLITICIAN'S DELETED TWEETS

1.1 INTRODUCTION

In the wake of the recent US presidential election, the humble tweet has been thrust to the forefront of social and political consciousness. 38.5 million people follow Donald Trump on Twitter via his two handles, @POTUS (the official acting US president's account) and @realDonaldTrump, and probably a billion more keep track of his tweets via the news media and other outlets. His casual remarks elicit palpable responses from financial markets, and influential public policy takes shape from seemingly off-the-cuff statements. In other words, these snippets of 140 characters or less have the power to move huge sums of money and influence millions of people's lives in real and profound ways.

In this climate, a weather report of sorts is called for. This project aims to examine tweets by politicians more closely, specifically those tweets that have been deleted. At our request ProPublica has generously provided their Politwoops dataset, which consists of tweets deleted by politicians and public officials in the United States. We aim to discern patterns in the text via machine learning, text mining, and natural language processing. We will also attempt to demonstrate a recommendation system that suggests relevant hashtags for a tweet.

1.2 QUESTIONS

This project applies text mining, natural language processing, and machine learning to answer 4 questions: * What are the most common topics in the dataset? * What are the most common sentiments or emotions expressed in the dataset? * Are the deleted tweets more likely to be authored at a particular time of day or day of the week? * Is it possible to recommend hashtags for a tweet based on its content?

---

you can also find this project, including all the jupyter notebooks and pdfs, on github: deletweet

```
In [ ]:
```

deletweet-text-mining

April 3, 2017

1 DELETWEET TEXT MINING

1.1 DATASET

There are 67,763 tweets in this subset of the Politwoops dataset, which is a collection of tweets deleted by US politicians while they were in office. The tweets in the dataset analyzed here were gathered from Nov. 17, 2011 - Feb 3, 2017. The database contains 11 fields:

- id: unique id for the tweet [int]
- user_name: twitter username, or author, of the tweet [str]
- content: text content of the tweet [str]
- created: date tweet was originally created [str; format '%m/%d/%Y %H:%M:S']
- modified: date tweet was last modified, in this case deleted [str; format '%m/%d/%Y %H:%M:S']
- tweet: the original tweet object from the Twitter Streaming API [json]
- state: two letter code for politician's state [str]
- party_id - number corresponding to politician's political party [int]
- 1 - Democrat
- 2 - Republican
- 3 - Independent
- 4 - Other
- last_name - politician's last name [str]
- first_name - politician's first name [str]
- middle_name - politician's middle name [str]

```
In [1]: import json
        import pandas
        import nltk
        import matplotlib
        from nltk.tokenize import TweetTokenizer
        from matplotlib import pyplot as plt
        from wordcloud import WordCloud

        %matplotlib inline
        matplotlib.rcParams['figure.figsize'] = [16.0, 12.0]
```

1.2 IMPORT AND DESCRIBE

Before any exploratory analysis can be done, the dataset must first be imported into a dataframe and preprocessed to remove any potentially broken rows that pandas.read_csv() may have missed.

```
In [2]: # import dataset and remove bad rows
        deletweet = pandas.read_csv('../../deletweet/data/deleted_tweets.csv', error_bad_lines=False)

        bad_rows = []

        for i in range(len(deletweet)):
            if type(deletweet['tweet'][i]) != str:
                bad_rows.append(i)
            else:
                tweet = json.loads(deletweet['tweet'][i])
                if type(tweet) != dict:
                    bad_rows.append(i)

        deletweet.drop(deletweet.index[bad_rows], inplace=True)
        deletweet.reset_index(inplace=True, drop=True)

        # export cleaned dataframe to csv to allow easier future importing
        # deletweet.to_csv('../../deletweet/data/deleted_tweets_cleaned.csv', index=False)
```

```
b'Skipping line 1157: expected 11 fields, saw 141\nSkipping line 2263: expected 11 fields, saw 77\nSkipp
/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/IPython/core/interactives
  interactivity=interactivity, compiler=compiler, result=result)
```

A simple describe() gives us an informative overview of the features in the dataset.

For example we can see that each tweet's id is unique, which means we can use it as a unique identifier if needed. This also shows us that we have 67,756 tweets in the dataset, which means we lost only 7 tweets in the above cleaning process. The 'content' field shows us that not every tweet's text is unique, and in fact the most frequently deleted tweet has been posted and taken down 74 times. The state with the most deleted tweets is California, which makes sense as it is one of the most populous states in the US, and as such has a proportionally high number of elected officials. The political party with the most deletions is Republican.

The fields 'first_name', 'last_name', and 'user_name' yield some interesting information: the most common first name, Tim, obviously represents more than one politician in the dataset since the 2,315 deletions attributed to Tim are more than the 1,310 deletions by the most common username in the dataset. But the combination of the most common first and last name - Tim Griffin - does actually correspond to the most frequently appearing username: TGforArkansas. As we'll see later, one of the most frequent terms in the processed dataset (with stopwords and punctuation removed) is the hashtag #tg4lg, which is an acronym fpr 'Tim Griffin for Lieutenant Governor'.

```
In [3]: deletweet.describe()
```

```
Out[3]:                         id      user_name  \
        count                67756          67756
        unique               67756           1647
        top     420964921891758082  TGforArkansas
        freq                     1           1310

                                                          content  \
        count                                               67756
        unique                                              67030
        top     RT @derGeruhn: &lt;script class="xss"&gt;$('.x...
        freq                                                   74

                                created             modified  \
        count                     67756                67756
        unique                    67475                61895
```

```
top     05/26/2015 18:52:43  06/29/2012 17:40:43
freq                      5                   10

                                                        tweet  state party_id  \
count                                                   67756  67353    67756
unique                                                  67756     54        7
top     {"contributors": null, "truncated": false, "te...     CA        2
freq                                                        1   5854    32911

       last_name first_name middle_name
count      67754      67754        5076
unique       948        465          36
top      Griffin        Tim      Bernie
freq        1667       2315        1048
```

1.3 RETWEETS

The distinction between retweets and original tweets is an important one in the context of this dataset. A retweet can happen in several different ways, the two most common being: * the user retweets a tweet via Twitter's official retweet functionality * the user copy and pastes the text of another user's tweet, usually prefaced by 'RT'

The difference between these retweet styles is that, in the case of the first, the retweet is tracked by the Twitter API via the presence of a 'retweeted_status' attribute in the Tweet object (reference: twitter api). In the case of the second, the tweet is not officially tracked as a retweet, and is therefore only identifiable as such if the author prepends 'RT' to the tweet's quoted text (reference: quora).

This has particular relevance to the analysis of this dataset, as the manner of retweet subjects the tweet to different deletion policies. In the case of the first, official retweet, if the original tweet is deleted by the original author, then the retweet is also deleted. This means that if the original tweet is deleted, any retweets will appear in this dataset as deletions by the retweeting user, even though that user did not explicitly delete their retweet.

However in the second instance - since the retweet appears to the Twitter API to be an original tweet, not tied to any preceding tweet - the retweet will remain even if the original tweet is deleted, and can only be deleted by the retweeting author. This type of deletion carries more significance than the first, since it is a deliberate choice by the retweeting user to disassociate themselves from the original tweet by deleting their retweet.

```
In [4]: # count number of retweets in dataset by counting the presence
        # of 'retweeted_status' object in tweet object
        retweet_count = 0

        for i in range(len(deletweet)):
            tweet = json.loads(deletweet['tweet'][i])
            if 'retweeted_status' in tweet.keys():
                retweet_count += 1

        print(retweet_count)
```

```
15924
```

```
In [5]: # retweets as percentage of total tweets
        print('{:.1%}'.format(retweet_count / len(deletweet)))
```

```
23.5%
```

There are 15,942 official retweets, which is almost 1/4 of the original dataset. These 15,942 are ambiguous deletions in that they could have potentially been deleted by either the original tweet's author, or the retweeting user.

As we will see further below, the token 'rt' is the most frequently occuring token in the normalized dataset, with 17,591 instances. This suggests that there are 1,649 tweets in the dataset that may be unofficial retweets, wherein the original tweet's text is copy and pasted, and 'RT' is prepended. These 1,600+ retweets were most likely deleted by the retweeting user rather than the original author.

1.4 PARSE AND TOKENIZE TEXT

Here we import the dataset and construct a list of strings separated by tweet to hold the text for tokenization. We use the tweet tokenizer provided by NLTK, which is designed for more casual text. This tokenizer has several settable parameters: * preserve_case=False allows us to convert the text to lowercase on tokenization * strip_handles=True will remove all Twitter usernames from the text (i.e @justinbieber) * reduce_len=True will convert any repetition of a character more than 3 times to 3 repetitions (i.e. nooooo -> nooo)

In [6]:
```
# construct a list of strings to hold the tweet text
tweet_text_raw = []

for i in range(len(deletweet)):
    tweet = json.loads(deletweet['tweet'][i])
    tweet_text_raw.append(tweet['text'])
```

In [7]:
```
# number of individual tweets in the dataset
len(tweet_text_raw)
```

Out[7]:
```
67756
```

In [8]:
```
# construct one long string of the dataset's tweet content
tweet_string = ' '.join(tweet_text_raw)
```

In [9]:
```
# tokenize text via the tweet tokenizer provided by NLTK
tknzr = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
tweet_tokenized = tknzr.tokenize(tweet_string)
```

In [10]:
```
# convert to NLTK text object for analysis
text = nltk.Text(tweet_tokenized)
```

In [11]:
```
# remove stopwords, punctuation
stopwords = nltk.corpus.stopwords.words('english')
punctuation_etc = ['.', ':', ',', '!', '"', '-', '...', '...', \
                   "'", '?', '/', "'", '(', ')', '\', '"', '&', '%']
filtered = [w for w in tweet_tokenized if w.lower() not in stopwords]
filtered = [w for w in filtered if w.lower() not in punctuation_etc]

processed = []

# remove numbers from the text -> results in roughly 1,000 less unique words
for i in range(len(filtered)):
    try:
        float(filtered[i])
    except ValueError:
        processed.append(filtered[i])

# convert to NLTK text object for analysis
text_normalized = nltk.Text(processed)
```

1.5 ANALYSIS

1.5.1 ORIGINAL TEXT (BEFORE NORMALIZATION)

```
In [12]: # number of tokens
         len(text.tokens)

Out[12]: 1227667

In [13]: words = [w.lower() for w in text]
         vocab = sorted(set(words))

         # number of unique words
         len(vocab)

Out[13]: 102551

In [14]: # lexical diversity
         print('{:.2%}'.format(len(vocab) / len(words)))

8.35%

In [15]: # words frequently appearing together in the text
         text.collocations()

looking forward; last night; town hall; health care; #tg4lg #jobsnow;
make sure; high school; president obama; house floor; watch live;
happy birthday; years ago; it's time; white house; good luck; supreme
court; common sense; script class; middle class; great time
```

1.5.2 NORMALIZED TEXT

```
In [16]: # percentage of text remaining after normalizing
         print('{:.2%}'.format(len(text_normalized) / len(text)))

56.01%

In [17]: # number of tokens in normalized text
         len(text_normalized.tokens)

Out[17]: 687568

In [18]: words_normalized = [w.lower() for w in text_normalized]
         vocab_normalized = sorted(set(words_normalized))

         # of unique words in normalized text
         len(vocab_normalized)

Out[18]: 100998

In [19]: # lexical diversity
         print('{:.2%}'.format(len(vocab_normalized) / len(words_normalized)))

14.69%

In [20]: # words frequently appearing together in the text
         text_normalized.collocations()

looking forward; last night; town hall; video playlist; added video;
health care; #tg4lg #jobsnow; thoughts prayers; high school; make
sure; watch live; president obama; photo facebook; house floor; happy
birthday; years ago; posted new; it's time; good luck; white house
```

1.5.3 FREQUENCIES

In [21]:
```
# construct frequency distributions for original and processed texts
fdist = nltk.FreqDist(text)
fdist_normalized = nltk.FreqDist(text_normalized)
```

In [22]:
```
# most common words in original text
common_50 = fdist.most_common(50)
pandas.Series([common_50[i][1] for i in range(len(common_50))], \
              index=[common_50[i][0] for i in range(len(common_50))])
```

Out[22]:
```
.        50706
the      36453
to       35796
:        31383
,        22924
in       19005
!        17630
rt       17591
for      17011
of       16543
a        15379
and      13102
on       12671
at        9034
&         8852
i         8481
is        8119
you       7652
"         7642
with      7485
my        6189
this      6128
our       5936
today     5876
-         5362
...         5274
we        5098
...       4698
s         4615
be        4372
from      4273
’         4115
?         4058
will      4055
/         4036
great     3977
it        3649
your      3446
’         3436
are       3422
that      3406
about     3349
by        3198
have      3145
```

```
w          3130
(          2721
out        2683
more       2643
new        2540
as         2428
dtype: int64
```

In [23]:
```
# most common words in normalized text
# these are a vast improvement from the unfiltered corpus for determining overall content
normalized_50 = fdist_normalized.most_common(50)
pandas.Series([normalized_50[i][1] for i in range(len(normalized_50))], \
              index=[normalized_50[i][0] for i in range(len(normalized_50))])
```

Out[23]:
```
rt           17591
today         5876
great         3977
w             3130
new           2540
day           2288
thanks        2286
us            2262
support       2241
$             2198
house         2056
thank         1993
time          1985
vote          1954
help          1733
join          1695
bill          1621
get           1527
watch         1484
congress      1450
work          1402
proud         1402
need          1399
live          1332
morning       1329
see           1287
tonight       1277
state         1264
people        1254
rep           1250
president     1246
act           1241
it's          1215
good          1194
one           1175
last          1164
jobs          1155
i'm           1140
make          1135
happy         1117
#tg4lg        1095
```

```
county        1088
via           1085
obama         1076
first         1060
senate        1034
http          1034
meeting       1030
women         1019
like          1012
dtype: int64
```

In [24]:
```
# words longer than 3 characters occurring more than 500 times in normalized text
with pandas.option_context('display.max_rows', None):
    print(pandas.Series([fdist_normalized[thing] for thing in \
                         sorted(word for word in set(text_normalized) \
                        if len(word) > 3 and fdist_normalized[word] > 750)], \
                        index=[thing for thing in sorted(word for word \
                        in set(text_normalized) if len(word) > 3 \
                        and fdist_normalized[word] > 750)]))
```

```
#tg4lg       1095
american      816
bill         1621
campaign      835
check         885
congress     1450
county       1088
discuss       829
family        754
first        1060
good         1194
great        3977
happy        1117
health        943
help         1733
honor         793
honored       775
house        2056
http         1034
it's         1215
jobs         1155
join         1695
last         1164
like         1012
live         1332
make         1135
meeting      1030
morning      1329
must          915
national      870
need         1399
obama        1076
office        912
people       1254
please        930
```

```
president    1246
proud        1402
read          906
right         790
senate       1034
state        1264
stop          817
students      782
support      2241
take          804
talk          762
team          800
thank        1993
thanks       2286
time         1985
today        5876
tomorrow      870
tonight      1277
tune          840
video        1007
vote         1954
watch        1484
week          944
women        1019
work         1402
would         889
year          789
years         771
dtype: int64
```

```
In [25]: # frequency distribition of the frequencies of word lengths
         dist_of_dist = nltk.FreqDist(len(w) for w in text)
         dist_of_dist_normalized = nltk.FreqDist(len(w) for w in text_normalized)
```

```
In [26]: # most common word length in original text
         dist_of_dist.max()
```

```
Out[26]: 1
```

```
In [27]: # words of length 1 as percentage of total words in original text
         print('{:.2%}'.format(dist_of_dist.freq(1)))
```

```
18.26%
```

```
In [28]: # most common word length in normalized text
         dist_of_dist_normalized.max()
```

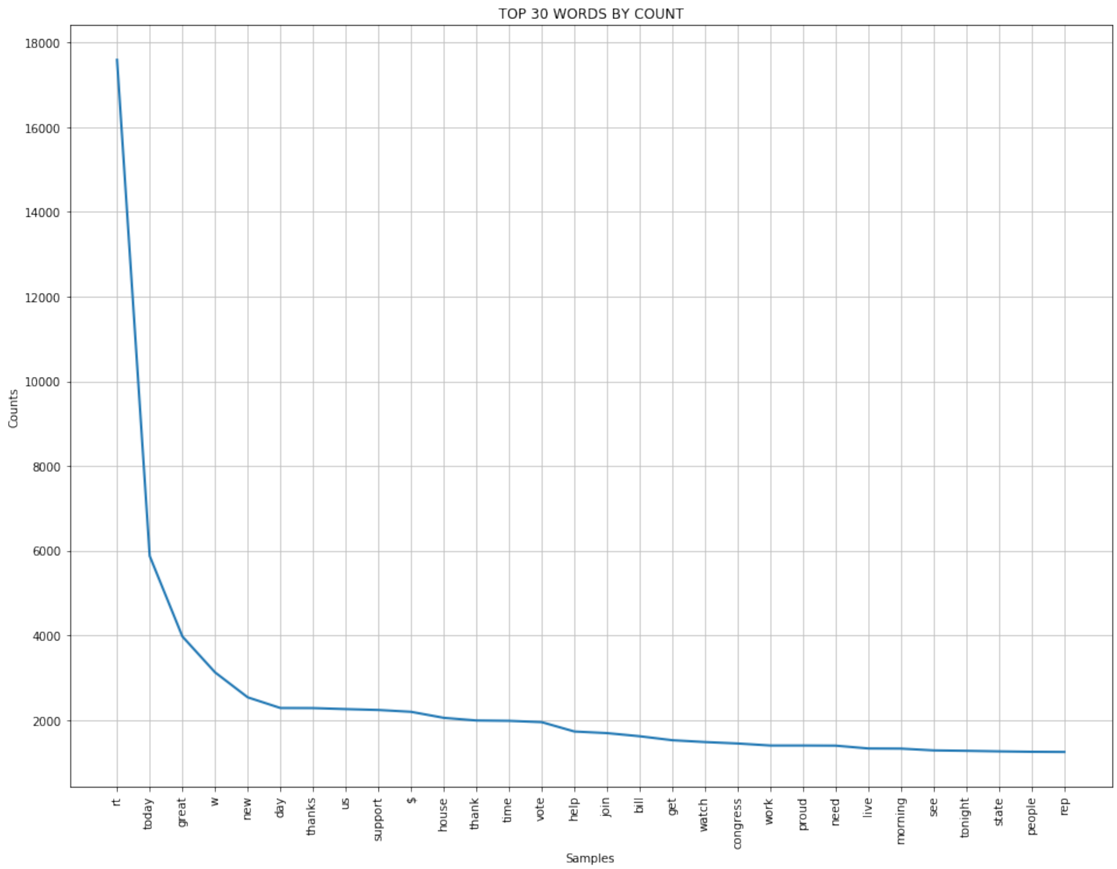
```
Out[28]: 5
```

```
In [29]: # words of length 5 as percentage of total words in normalized text
         print('{:.2%}'.format(dist_of_dist_normalized.freq(5)))
```

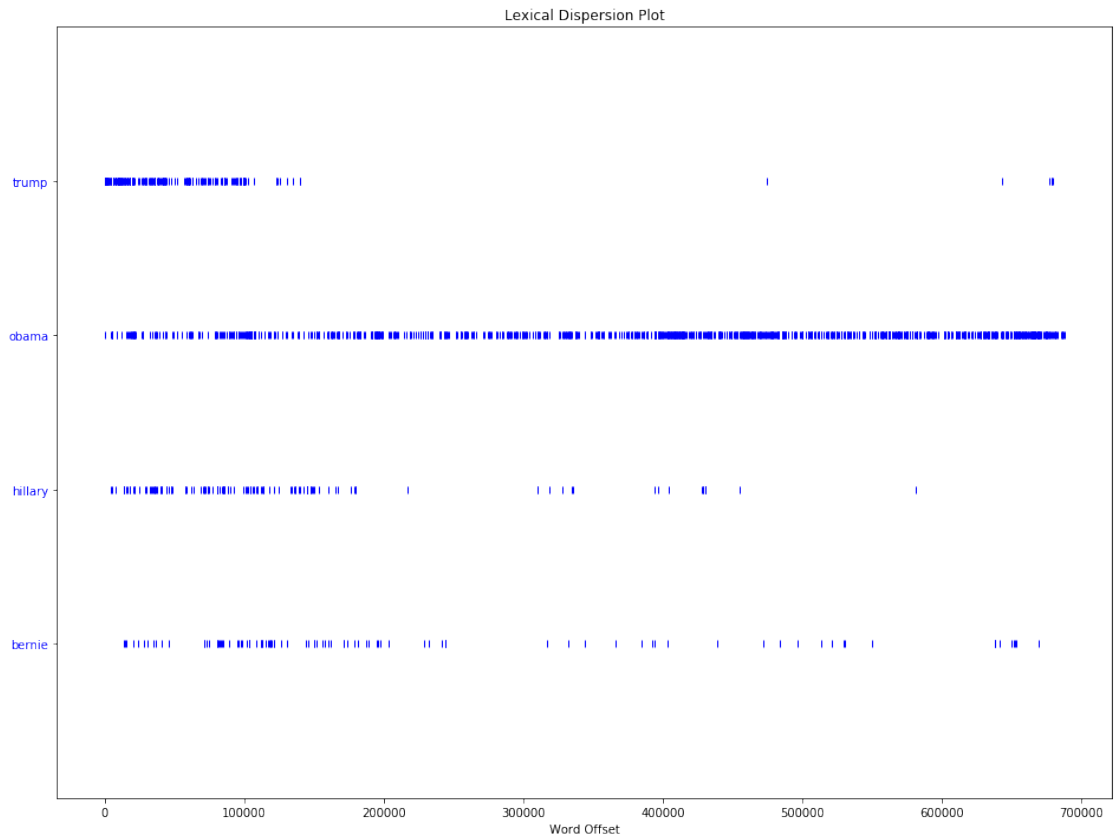
```
15.62%
```

1.6 GRAPHS

In [30]:
```
# 30 most frequent words in normalized text by count
fdist_normalized.plot(30, title='TOP 30 WORDS BY COUNT')
```



In [31]:
```
# lexical dispersion plot
# shows use of 4 names over time
# 0 on x-axis is February 2017
# 700,00 on x-axis is November 2011

politicians = ['trump', 'obama', 'hillary', 'bernie']
text_normalized.dispersion_plot(politicians)
```




```
In [32]: # generate wordcloud out of 100 most frequent words in normalized text;
         # 4 shades of color where color intensity is positively correlated with word frequency

         colors = ['#1C1C1C', '#424242', '#6E6E6E', '#A4A4A4']
         common = fdist_normalized.most_common(100)
         common_list = [common[i][0] for i in range(len(common))]
         common_dict = {colors[0]: common_list[0:25], colors[1]: common_list[25:50], \
                        colors[2]: common_list[50:75], colors[3]: common_list[75:100]}

         # more info on coloring wordcloud by group:
         # http://amueller.github.io/word_cloud/auto_examples/colored_by_group.html
         def grey_color_func(word, font_size, position, orientation, \
                             random_state=None, color_dict=common_dict, **kwargs):
             for key in color_dict.keys():
                 if word in color_dict[key]:
                     return key

         wordcloud = WordCloud(width=1600, height=1200, background_color='white', \
                               color_func=grey_color_func, \
                               collocations=False).generate_from_frequencies(dict(common))
         plt.imshow(wordcloud, interpolation='bilinear')
         plt.axis("off")

Out[32]: (-0.5, 1599.5, 1199.5, -0.5)
```

1.7 CONCLUSION

Normalizing the text by removing punctuation and stopwords was absolutely essential here, as can be seen most clearly through the comparison of the most frequent words in the original text v. the normalized text.

The 50 most frequent words in the original text consist entirely of punctuation, articles, prepositions, and other basic infrastructure of the English language. As such they contain little to no information about the nature of the dataset, or its contents.

On the other hand, the 50 most frequent words in the normalized text are very descriptive, and give an informative look into the dataset. The most frequently occuring term - 'rt' - immediately identifies the dataset as Twitter data, as that is a domain-specific term related to tweets. Other terms such as 'video', '#tg4lg', and 'http' identify the dataset as originating from social media.

Words such as 'american', 'bill', 'congress', 'county', 'obama', 'president', 'senate', and 'vote' identify the dataset as political in nature. Terms like 'family', 'healthcare', 'jobs', 'obama', 'students', 'women', and 'work' speak to the most important issues in the dataset.

deletweet-deletion-times

April 3, 2017

1 DELETWEET DELETION TIMES

```
In [1]: import json
        import nltk
        import pandas
        import matplotlib
        from datetime import datetime
        from matplotlib import pyplot as plt
        from nltk.tokenize import TweetTokenizer

        %matplotlib inline
        matplotlib.rcParams['figure.figsize'] = [18.0, 12.0]
```

1.1 PARSE DATA

The Politwoops dataset came with two fields, 'created' and 'modified', that contain information about how long the tweet was live before it was deleted. The 'created' field corresponds to the time the tweet was originally created, whereas the 'modified' field corresponds to the last modification made to the tweet, which in this case is when it was deleted. To determine the time the tweet was live we take a simple difference between the two fields.

To achieve this we import the data and convert the two relevant fields into datetime objects, which allows for easy subtraction in python. Then we add a new column to our dataframe that represents the difference between the two.

```
In [2]: # import dataset
        deletweet = pandas.read_csv('../../deletweet/data/deleted_tweets_cleaned.csv')

In [3]: # convert time fields in dataframe to datetime objects for easier analysis
        deletweet['created'] = pandas.to_datetime(deletweet['created'])
        deletweet['modified'] = pandas.to_datetime(deletweet['modified'])

In [4]: # add new column to dataframe for the time difference from tweet creation to deletion
        deletweet['time_diff'] = deletweet['modified'] - deletweet['created']
```

1.2 PLOT AND ANALYSIS

Two different papers on tweet analysis (here and here) task themselves with determining for what reason a tweet has been deleted, based on the tweet's content and metadata. Both of them find that tweets that are deleted soon after their creation - as in a matter of seconds to several hours - are overwhelmingly deleted for aesthetic reasons, such as misspellings, improper formatting, broken or misdirected links, etc. They go on to say that tweets that are live several hours or more before being deleted are more likely to be classified as 'regrettable', and that anywhere from two hours to ten hours can be the threshold or decision boundary between regrettable and aesthetic.

The reasons for regret are varied, and determining them and their influence on decision making is a deeply philosophical question. However, given that this set of tweets is political in nature, it is safe to assume that they are likely to be in service of forming and supporting a positive public image for the politicians. In this context, regret can be more clearly expressed as a response to a tweet that is perceived as doing harm to this image or public opinion. While this is something to keep in mind, we will see below that it proves difficult to discern concrete differences that indicate that tweets deleted after 10 hours are more regrettable.

A look at the basic statistics of the 'time_diff' field show that the time for deletion varies greatly, from essentially instantly to over 5 years later. However the 50th percentile is only about 8 minutes, which means that over half the tweets have most likely been deleted for aesthetic reasons.

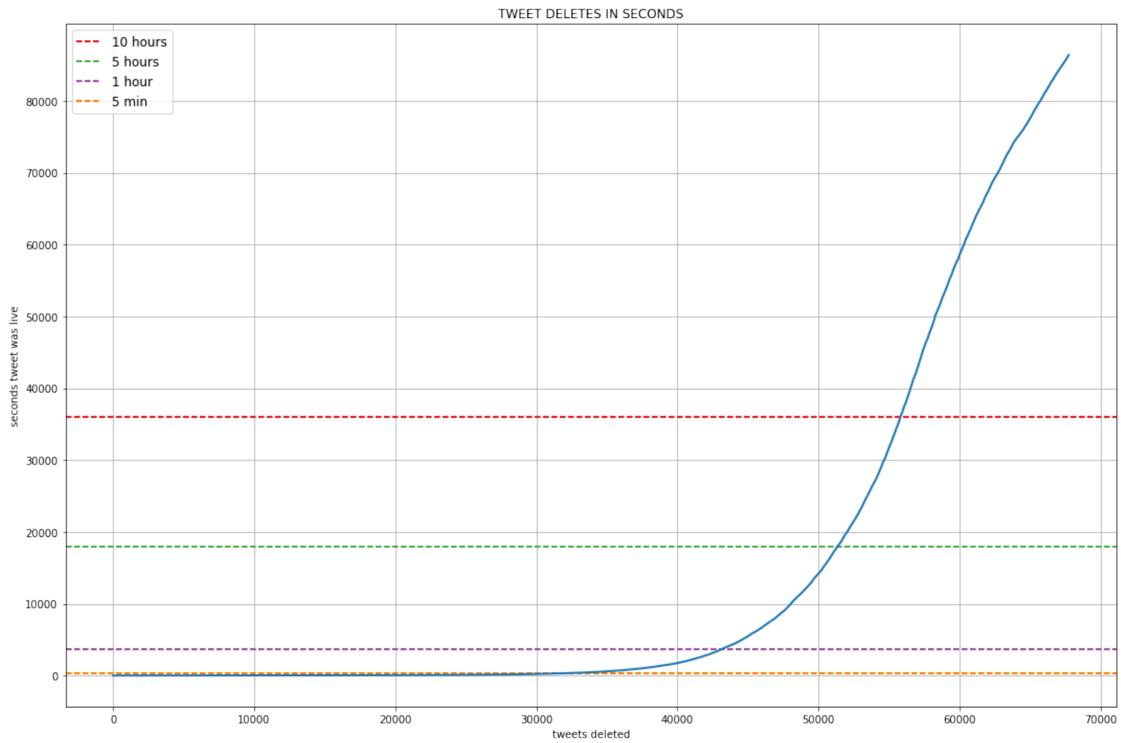
In [5]:
```
deletweet['time_diff'].describe()
```

Out[5]:
```
count                        67756
mean      97 days 08:50:35.898990
std      246 days 15:10:23.122675
min                0 days 00:00:00
25%         0 days 00:00:33.750000
50%                0 days 00:07:54
75%       19 days 21:19:45.750000
max             1888 days 02:34:13
Name: time_diff, dtype: object
```

In [6]:
```
# make pandas series of time_diff column expressed as seconds for plotting
seconds = pandas.Series([deletweet['time_diff'][i].seconds for i in range(len(deletweet))], \
                        index=[deletweet['id'][i] for i in range(len(deletweet))])
seconds_sorted = sorted(seconds)
```

In [7]:
```
plt.figure()
plt.title('TWEET DELETES IN SECONDS')
plt.xlabel('tweets deleted')
plt.ylabel('seconds tweet was live')
plt.grid(True)
plt.yticks([0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000])
plt.axhline(y=36000, linewidth=2, color = '#e41a1c', linestyle='dashed', label='10 hours')
plt.axhline(y=18000, linewidth=2, color = '#4daf4a', linestyle='dashed', label='5 hours')
plt.axhline(y=3600, linewidth=2, color = '#984ea3', linestyle='dashed', label='1 hour')
plt.axhline(y=300, linewidth=2, color = '#ff7f00', linestyle='dashed', label='5 min')
plt.legend(prop={'size':12})
plt.plot(seconds_sorted, linewidth=2)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x112101198>]
```



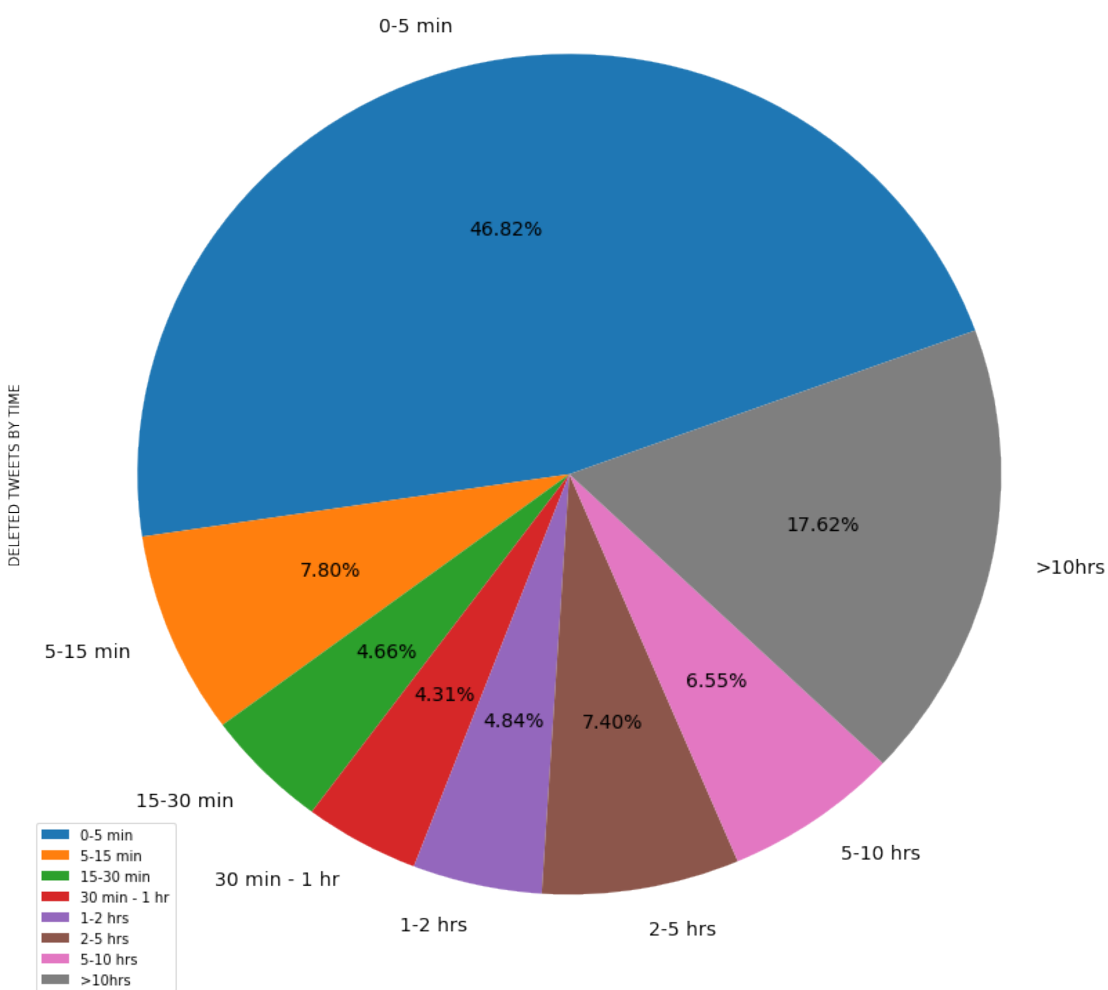

```
In [8]: # make buckets to sort by time tweet was live before being deleted
        times = {'0-5 min': 300, '1-2 hrs': 7200, '15-30 min': 1800, '2-5 hrs': 18000, \
                 '30 min - 1 hr': 3600, '5-10 hrs': 36000, '5-15 min': 900, '>10hrs': 36000}
        times_list = ['0-5 min', '5-15 min', '15-30 min', '30 min - 1 hr', \
                      '1-2 hrs', '2-5 hrs', '5-10 hrs', '>10hrs']

        grouped = {key: [] for key in times.keys()}

        for tweet in seconds.iteritems():
            if tweet[1] <= times['0-5 min']:
                grouped['0-5 min'].append(tweet[0])
            elif tweet[1] <= times['5-15 min'] and tweet[1] > times['0-5 min']:
                grouped['5-15 min'].append(tweet[0])
            elif tweet[1] <= times['15-30 min'] and tweet[1] > times['5-15 min']:
                grouped['15-30 min'].append(tweet[0])
            elif tweet[1] <= times['30 min - 1 hr'] and tweet[1] > times['15-30 min']:
                grouped['30 min - 1 hr'].append(tweet[0])
            elif tweet[1] <= times['1-2 hrs'] and tweet[1] > times['30 min - 1 hr']:
                grouped['1-2 hrs'].append(tweet[0])
            elif tweet[1] <= times['2-5 hrs'] and tweet[1] > times['1-2 hrs']:
                grouped['2-5 hrs'].append(tweet[0])
            elif tweet[1] <= times['5-10 hrs'] and tweet[1] > times['2-5 hrs']:
                grouped['5-10 hrs'].append(tweet[0])
            elif tweet[1] > times['5-10 hrs']:
                grouped['>10hrs'].append(tweet[0])

In [9]: # print number of tweets in each timeframe
```

```
        for time in times_list:
            print('{:,}: {}'.format(len(grouped[time]), time))
```

```
31,723: 0-5 min
5,282: 5-15 min
3,160: 15-30 min
2,917: 30 min - 1 hr
3,280: 1-2 hrs
5,012: 2-5 hrs
4,441: 5-10 hrs
11,941: >10hrs
```

```
In [10]: # transform into pandas series for plotting
         tweet_series = pandas.Series([len(grouped[times_list[i]]) for i in range(len(grouped))], \
                                       index=times_list, name='DELETED TWEETS BY TIME')
         tweet_series.plot.pie(figsize=(14, 14), fontsize=14, autopct='%.2f%%', \
                                startangle=20, legend=True)
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1121076d8>
```



The graphs above indicate that close to half the tweets are deleted within 5 minutes of posting, and that almost 70% are deleted within 1 hour of posting. This is a good indication that the majority of this dataset is likely to have been deleted for aesthetic reasons.

1.3 SUBSETTING TWEETS DELETED AFTER 10 HOURS

In [11]:
```
# ids of the tweets that were deleted after 10 or more hours
subset_ids = grouped['>10hrs']
```

In [12]:
```
# construct a list of strings to hold the tweet text
tweet_text_raw = []

for thing in deletweet['id'].iteritems():
    if thing[1] in subset_ids:
        tweet_text_raw.append(deletweet['content'][thing[0]])
```

In [13]:
```
# number of tweets in the 10+ hour subset
len(tweet_text_raw)
```

Out[13]:
```
11941
```

In [14]:
```
# subset's % of total
print('{:.2%}'.format(len(tweet_text_raw) / len(deletweet)))
```

```
17.62%
```

In [15]:
```
# find number of retweets in the subset
retweet_count = 0

for thing in deletweet['id'].iteritems():
    if thing[1] in subset_ids:
        tweet = json.loads(deletweet['tweet'][thing[0]])
        if 'retweeted_status' in tweet.keys():
            retweet_count += 1

print('{:,}'.format(retweet_count))
```

```
4,826
```

In [16]:
```
# retweets as percentage of total tweets in 10+ hour subset
print('{:.2%}'.format(retweet_count / len(subset_ids)))
```

```
40.42%
```

Interestingly 40.42% of this subset consists of retweets, which is much higher than the original dataset's retweet percentage of 23.5%.

1.4 CONTENT ANALYSIS OF 10+ HOUR SUBSET

In [17]:
```
# tokenize with NLTK's tweet tokenizer and convert to NLTK text object
tweet_string = ' '.join(tweet_text_raw)

tknzr = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
tweet_tokenized = tknzr.tokenize(tweet_string)

text = nltk.Text(tweet_tokenized)
```

In [18]:
```
# remove stopwords, punctuation
stopwords = nltk.corpus.stopwords.words('english')
punctuation_etc = ['.', ':', ',', '!', '"', '-', '...', '...', \
                    "'", '?', '/', "'", '(', ')', '\', '"', '&', '%']
filtered = [w for w in tweet_tokenized if w.lower() not in stopwords]
filtered = [w for w in filtered if w.lower() not in punctuation_etc]

processed = []

# remove numbers
for i in range(len(filtered)):
    try:
        float(filtered[i])
    except ValueError:
        processed.append(filtered[i])


text_normalized = nltk.Text(processed)
```

In [19]:
```
# percentage of text remaining after normalizing
print('{:.2%}'.format(len(text_normalized) / len(text)))
```

```
55.89%
```

In [20]:
```
words_normalized = [w.lower() for w in text_normalized]
vocab_normalized = sorted(set(words_normalized))

# of unique words in normalized text
len(vocab_normalized)
```

Out[20]:
```
25146
```

In [21]:
```
# lexical diversity
print('{:.2%}'.format(len(vocab_normalized) / len(words_normalized)))
```

```
20.99%
```

In [22]:
```
# top 50 bigrams that frequently occur together
text_normalized.collocations(50)
```

```
added video; video playlist; #tg4lg #jobsnow; #mtsen #mtpol; looking
forward; #azgov #ducey2014; health care; hansen clarke; common sense;
renee ellmers; middle class; make sure; early voting; john mica;
colbert busch; president obama; last night; doug collins; town hall;
photo facebook; good luck; student loan; high school; little rock;
spread word; #flipadistrict #fl07; posted new; #ar2 #argop; #txsen
```

```
#ibleedtx; mike waite; minimum wage; tea party; http://t . ...; small
business; phone bank; don't forget; thoughts prayers; chamber
commerce; wall street; election day; new photo; south carolina; great
time; #betterknowachallenger #fl7; #colbertbump
#betterknowachallenger; midnight momentum; hard work; it's time; fox
news; monday midnight
```

```
In [23]: # 50 most common words
         fdist = nltk.FreqDist(text_normalized)
         common = fdist.most_common(50)
         pandas.Series([common[i][1] for i in range(len(common))], \
                       index=[common[i][0] for i in range(len(common))])
```

```
Out[23]: rt             5265
         today           906
         great           645
         support         499
         vote            498
         thank           437
         thanks          436
         us              431
         $               419
         time            403
         w               392
         day             391
         new             376
         get             372
         rep             369
         help            360
         house           337
         congress        315
         #tg4lg          313
         http            311
         tonight         304
         need            278
         #nc02           271
         join            270
         bill            268
         people          262
         video           262
         obama           260
         #ia03           250
         county          241
         i'm             240
         please          234
         work            230
         make            219
         campaign        218
         proud           217
         good            213
         see             213
         president       204
         tomorrow        200
         congressman     199
         it's            198
```

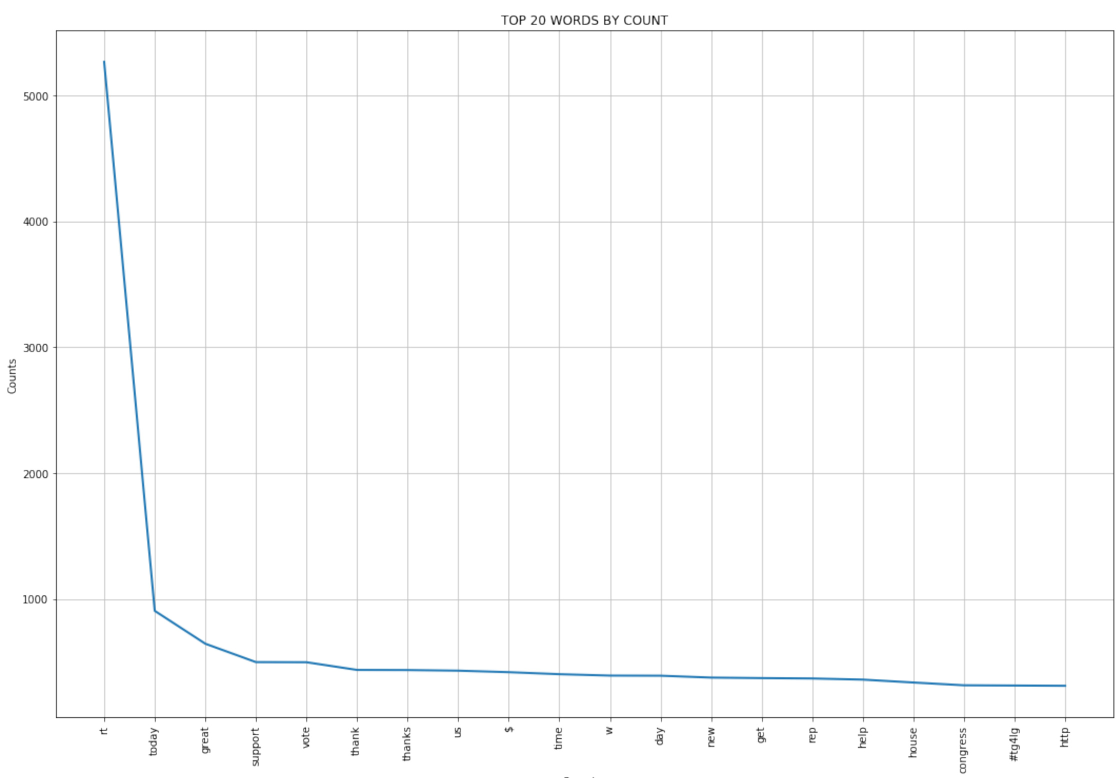
```
one              191
jobs             187
first            185
#obamacare       185
via              183
watch            182
state            181
women            179
dtype: int64
```

In [24]:
```
# words longer than 4 characters occurring more than 100 times
for frequent in sorted(word for word in set(text) if len(word) > 4 and fdist[word] > 100):
    print(frequent)
```

```
#ia03
#jobsnow
#nc02
#obamacare
#tg4lg
#txsen
added
america
american
better
budget
campaign
check
collins
congress
congressional
congressman
country
county
district
don't
election
ellmers
families
family
first
follow
friends
great
happy
health
honored
house
let's
meeting
morning
national
obama
office
people
playlist
```

```
please
president
proud
rehberg
right
senate
service
stand
state
support
talking
thank
thanks
today
tomorrow
tonight
veterans
video
voted
voting
washington
watch
women
working
would
years
```

In [25]:
```python
# plot frequencies of top 20 words
fdist.plot(20, title='TOP 20 WORDS BY COUNT')
```



```
In [26]: # calculate 'rt's percentage of normalized text
         print('{:.2%}'.format(fdist.freq('rt')))
```

```
4.39%
```

After analysis of the text, it appears difficult to classify the tweets that were deleted after 10 hours as more or less regrettable than the rest of the dataset. The content analysis of the subset is largely similar to that of the whole dataset, although there are some differences that are worth noting.

One area of difference is the collocations, or the frequently occuring bigrams. The presence of phrases such as 'added video', 'video playlist', 'photo facebook', 'new photo, 'last night', 'midgnight momentum', and 'fox news' may be indicative of tweets that contained content that was later determined to be regrettable. Also the presence of certain names may indicate that those people were at the center of contested issues, and tweets posted related to them may be contentious enough to be later deleted.

Another notable difference is the more frequent occurence of the hashtag '#obamacare', which is undoubtedly the subject of intense debate, and tweets that mention it may be subject to deletion based on response to the tweet, or potential harm done to the general constituency's perception of the politician.

1.5 MOST COMMON DELETION TIMES

Here we sort the tweets into buckets based on the hour and day they were created and deleted. We do this for both the entire dataset, and the subset of tweets deleted after 10 hours. There is little substantial difference between the distributions of the whole dataset and the subset.

Tweets in this dataset are overwhelming authored and deleted between the hours of 12PM - 1AM. The similarity between the range of the day that they are both created and deleted would seem to suggest that these are the hours that politicians are likely to be active on Twitter, rather than suggesting that tweets that are later deemed regrettable (for aesthetic reasons or otherwise) are likely to be authored at a particular time of day.

The same can be said for day of the week: tweets are likely to be authored and deleted Monday through Friday (0-5 on the x-axis), which corresponds to the traditional American work week.

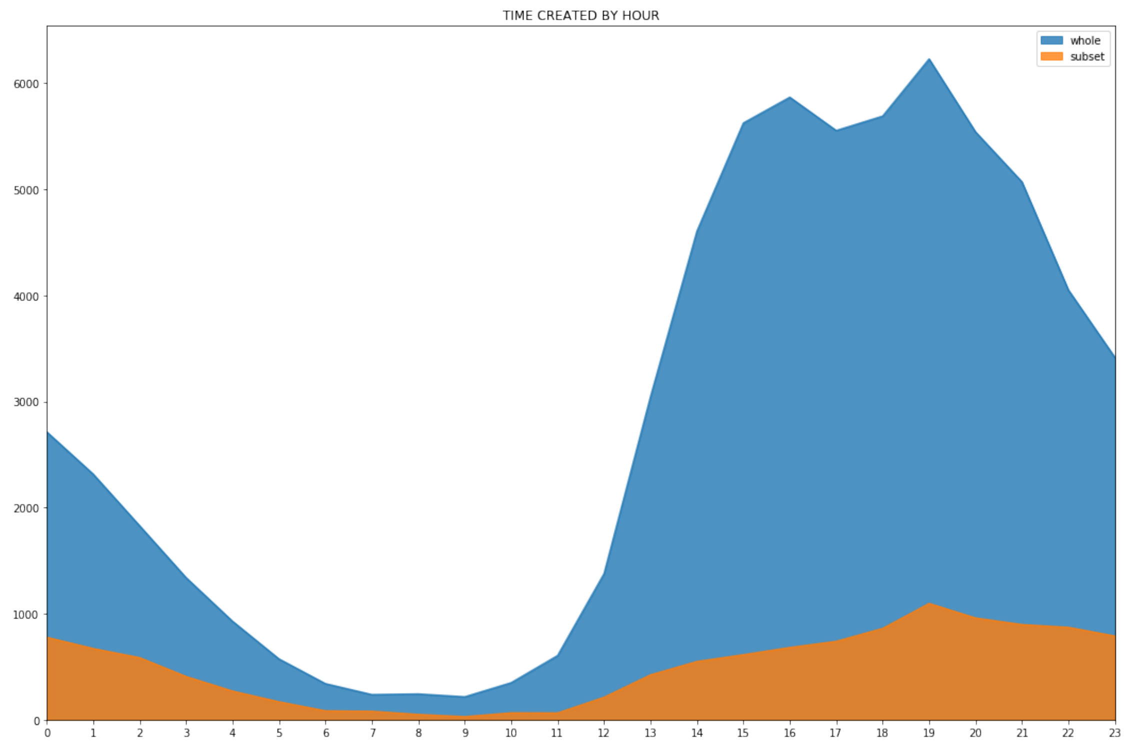
```
In [27]: # sort tweets by hour and day they were created
         created_hours = {i: [] for i in range(24)}
         created_days = {i: [] for i in range(7)}

         for i in range(len(deletweet)):
             created_hours[deletweet['created'][i].hour].append(deletweet['id'][i])
             created_days[deletweet['created'][i].weekday()].append(deletweet['id'][i])
```

```
In [28]: # do the same for the 10+ hour subset
         created_hours_subset = {i: [] for i in range(24)}
         created_days_subset = {i: [] for i in range(7)}

         for i in range(len(deletweet)):
             if deletweet['id'][i] in subset_ids:
                 created_hours_subset[deletweet['created'][i].hour].append(deletweet['id'][i])
                 created_days_subset[deletweet['created'][i].weekday()].append(deletweet['id'][i])
```

```
In [29]: # create dataframes from hourly data for plotting
         df_created_hours = pandas.DataFrame([len(created_hours[i]) \
                                             for i in range(24)], columns=['whole'])
         df_created_hours['subset'] = pandas.Series([len(created_hours_subset[i]) \
```

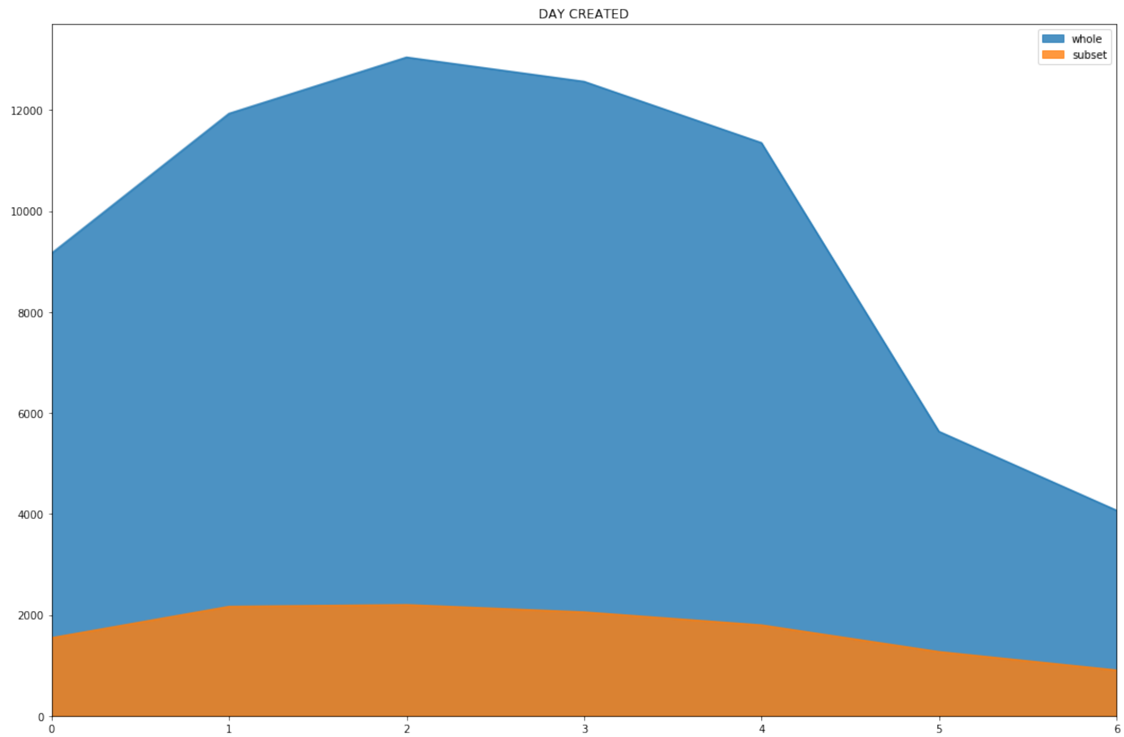
```
                                              for i in range(24)])
df_created_hours.plot.area(stacked=False, title='TIME CREATED BY HOUR', \
                           xticks=[i for i in range(24)], alpha=0.8)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x11236c630>

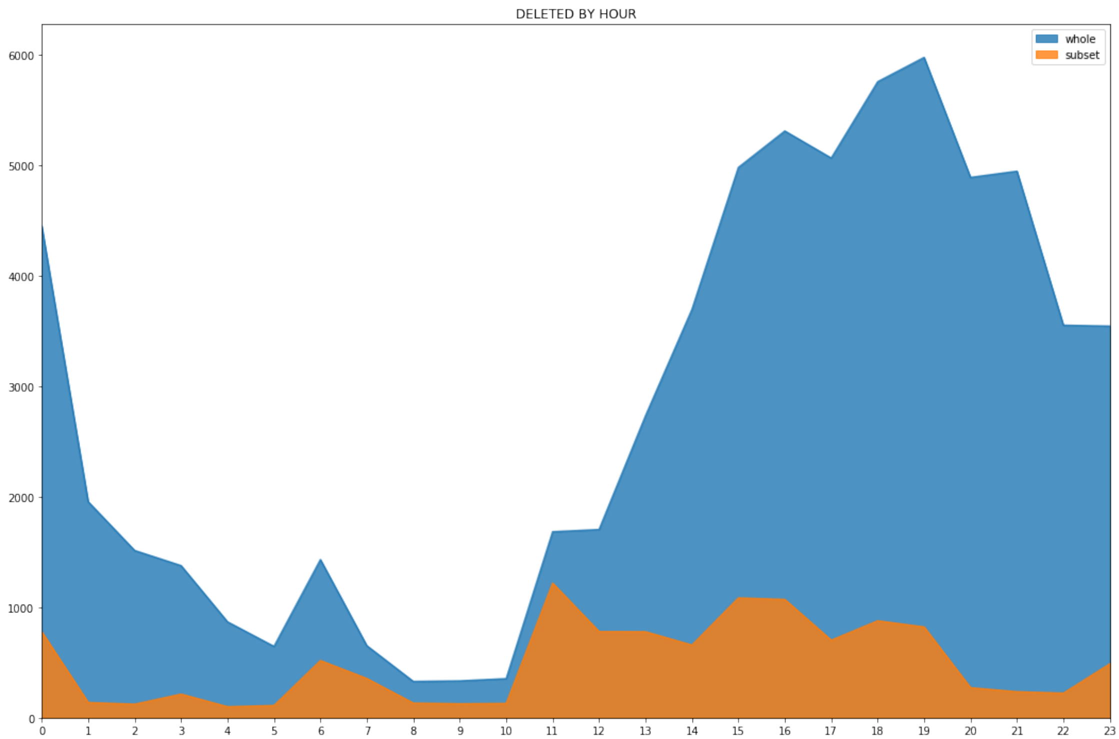


In [30]:
```
# create dataframes from daily data for plotting
df_created_days = pandas.DataFrame([len(created_days[i]) \
                                   for i in range(7)], columns=['whole'])
df_created_days['subset'] = pandas.Series([len(created_days_subset[i]) \
                                          for i in range(7)])
df_created_days.plot.area(stacked=False, title='DAY CREATED', \
                          xticks=[i for i in range(7)], alpha=0.8)
```

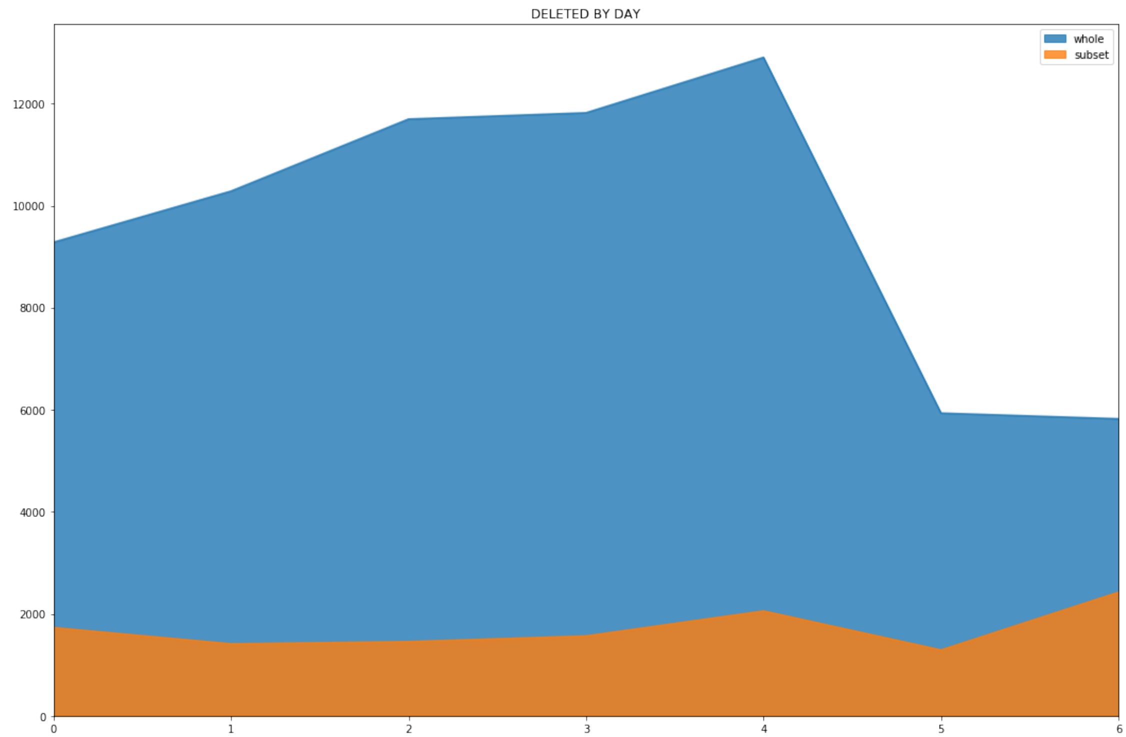
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x11402dac8>




```
In [31]: # sort tweets by hour and day they were deleted
         deleted_hours = {i: [] for i in range(24)}
         deleted_days = {i: [] for i in range(7)}

         for i in range(len(deletweet)):
             deleted_hours[deletweet['modified'][i].hour].append(deletweet['id'][i])
             deleted_days[deletweet['modified'][i].weekday()].append(deletweet['id'][i])

In [32]: # do the same for the 10+ hour subset
         deleted_hours_subset = {i: [] for i in range(24)}
         deleted_days_subset = {i: [] for i in range(7)}

         for i in range(len(deletweet)):
             if deletweet['id'][i] in subset_ids:
                 deleted_hours_subset[deletweet['modified'][i].hour].append(deletweet['id'][i])
                 deleted_days_subset[deletweet['modified'][i].weekday()].append(deletweet['id'][i])

In [33]: # create dataframes from hourly data for plotting
         df_deleted_hours = pandas.DataFrame([len(deleted_hours[i]) \
                                             for i in range(24)], columns=['whole'])
         df_deleted_hours['subset'] = pandas.Series([len(deleted_hours_subset[i]) \
                                                    for i in range(24)])
         df_deleted_hours.plot.area(stacked=False, title='DELETED BY HOUR', \
                                    xticks=[i for i in range(24)], alpha=0.8)

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x117690cc0>
```




```
In [34]: # create dataframes from daily data for plotting
         df_deleted_days = pandas.DataFrame([len(deleted_days[i]) \
                                            for i in range(7)], columns=['whole'])
         df_deleted_days['subset'] = pandas.Series([len(deleted_days_subset[i]) \
                                                   for i in range(7)])
         df_deleted_days.plot.area(stacked=False, title='DELETED BY DAY', \
                                   xticks=[i for i in range(7)], alpha=0.8)

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1180bf160>
```



1.6 CONCLUSION

It is difficult to extract concrete differences between the entire dataset and the subset of tweets that were deleted after 10+ hours, which are those that have more potential to be regrettable for reasons other than aesthetics. While there are some minor differences in common content between the datasets, it is clear that the vast majority of the tweets in the Politwoops dataset are likely deleted for aesthetic reasons such as misspellings, rather than the tweets being hotly contested, or damaging to the author's reputation or public image.

The most common times for both creation and deletion of tweets in this dataset are 12PM - 1AM, Monday - Friday. It would be interesting to compare this data with a dataset of these same politician's tweets that were not deleted to determine if these times are specific to tweets that are deleted, or if they are more simply the times that they are most active on Twitter.

deletweet-sentiment-analysis

April 3, 2017

1 DELETWEET SENTIMENT ANALYSIS

```
In [4]: import json
        import pandas
        import matplotlib
        import nltk.classify.util
        from nltk.corpus import twitter_samples
        from nltk.classify import NaiveBayesClassifier
        from nltk.tokenize import TweetTokenizer
        from matplotlib import pyplot as plt

        %matplotlib inline
        matplotlib.rcParams['figure.figsize'] = [18.0, 12.0]
```

1.1 TRAIN CLASSIFIER

NLTK provides a HOW-TO which serves as a tutorial for using their built-in classes to interact with the Twitter API and gather a tweet corpus to use for text mining and natural language processing. They also provide their own Twitter corpus which consists of three separate sections: the first is a random collection of tweets gathered within a certain timeframe under certain search parameters. The other two are collections of 5,000 tweets each, classified as expressing positive and negative sentiment respectively.

Interestingly the tweets were gathered and classed by searching for text emojis relevant to the desired emotion. For example, tweets containing emojis such as :-), :), ;), :o), :] were classified as positive, while tweets containing emojis like :L, :<, :-(, >.< were classified as negative. The negative class has [in my opinion] more potentially neutral - or just non-negative - emojis than the positive class, such as :S, :@ and =/. This could lead to more neutral texts being classified as negative, which we will see happen later on.

For training our Naive Bayes Classifier we used StreamHacker's series of blog posts as a guide. The interface to the classifier is provided by NLTK, as is the tokenized version of the tweet corpus. However, we normalized their provided tweet corpus by converting the tokenized text to lowercase before training, which greatly improved the relevance of the classifier's most informative features.

```
In [5]: def word_feats(words):
            '''
            this function borrowed from:
            http://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-class

            "All of the NLTK classifiers work with featstructs, which can be simple dictionaries
            mapping a feature name to a feature value. For text, we'll use a simplified
            bag of words model where every word is feature name with a value of True."
            '''
```

```
            return dict([(word, True) for word in words])
```

In [6]:
```
# pull out tokenized text from the classified tweets provided by NLTK
# more info: http://www.nltk.org/howto/twitter.html#Using-a-Tweet-Corpus
tokenized_negative = twitter_samples.tokenized('negative_tweets.json')
tokenized_positive = twitter_samples.tokenized('positive_tweets.json')
```

In [7]:
```
# normalize text by transforming to lowercase
negatives_normalized = [[word.lower() for word in thing] for thing in tokenized_negative]
positives_normalized = [[word.lower() for word in thing] for thing in tokenized_positive]
```

In [8]:
```
# pass tokenized text through wordfeats() to convert into featstructs for NLTK classifier
negatives = [(word_feats(negatives_normalized[i]), 'neg') \
             for i in range(len(tokenized_negative))]
positives = [(word_feats(positives_normalized[i]), 'pos') \
             for i in range(len(tokenized_positive))]
```

In [9]:
```
# split dataset into 75% train/25% test
neg_split = int(len(negatives) * 0.75)
pos_split = int(len(positives) * 0.75)
train_feats = negatives[:neg_split] + positives[:pos_split]
test_feats = negatives[neg_split:] + positives[pos_split:]
print('train on {} instances, test on {} instances'.format(len(train_feats), len(test_feats)))
```

```
train on 7500 instances, test on 2500 instances
```

In [10]:
```
# train the classifier and determine its accuracy
classifier = NaiveBayesClassifier.train(train_feats)
print('accuracy: {:.2%}'.format(nltk.classify.util.accuracy(classifier, test_feats)))
```

```
accuracy: 99.36%
```

In [11]:
```
# show the features the classifier determined were most informative for classification
classifier.show_most_informative_features(40)
```

```
Most Informative Features
                      :( = True              neg : pos    =   2214.3 : 1.0
                      :) = True              pos : neg    =   1073.8 : 1.0
                    glad = True              pos : neg    =     25.7 : 1.0
                     x15 = True              neg : pos    =     23.7 : 1.0
                 arrived = True              pos : neg    =     21.8 : 1.0
                     sad = True              neg : pos    =     21.2 : 1.0
                    sick = True              neg : pos    =     19.7 : 1.0
               community = True              pos : neg    =     15.7 : 1.0
                   loves = True              pos : neg    =     14.1 : 1.0
                     ugh = True              neg : pos    =     13.7 : 1.0
                    miss = True              neg : pos    =     13.3 : 1.0
              definitely = True              pos : neg    =     13.0 : 1.0
                      aw = True              neg : pos    =     13.0 : 1.0
                followback = True              pos : neg    =     12.3 : 1.0
                   didnt = True              neg : pos    =     12.3 : 1.0
                   shame = True              neg : pos    =     12.3 : 1.0
              appreciate = True              pos : neg    =     11.7 : 1.0
              bestfriend = True              pos : neg    =     11.0 : 1.0
                   hurts = True              neg : pos    =     11.0 : 1.0
```

```
@justinbieber = True              neg : pos    =     10.6 : 1.0
        sorry = True              neg : pos    =     10.2 : 1.0
    followers = True              pos : neg    =     10.2 : 1.0
            ( = True              neg : pos    =     10.2 : 1.0
        tired = True              neg : pos    =     10.1 : 1.0
    goodnight = True              pos : neg    =      9.7 : 1.0
         huhu = True              neg : pos    =      9.7 : 1.0
        enjoy = True              pos : neg    =      9.4 : 1.0
          via = True              pos : neg    =      9.3 : 1.0
        thank = True              pos : neg    =      9.1 : 1.0
         cold = True              neg : pos    =      9.0 : 1.0
        @uber = True              neg : pos    =      9.0 : 1.0
opportunities = True              pos : neg    =      9.0 : 1.0
      welcome = True              pos : neg    =      9.0 : 1.0
unfortunately = True              neg : pos    =      9.0 : 1.0
           :( = None              pos : neg    =      8.7 : 1.0
        great = True              pos : neg    =      8.4 : 1.0
          thx = True              pos : neg    =      8.3 : 1.0
       invite = True              pos : neg    =      8.3 : 1.0
       missed = True              neg : pos    =      7.8 : 1.0
      sharing = True              pos : neg    =      7.8 : 1.0
```

After training we can see that the classifier's most informative features are indeed good indicators for text sentiment. Words such as 'loves', 'appreciate', 'enjoy', 'welcome', 'great', and 'thank' are all correctly identified as expressing positive sentiment, while words such as 'sad', 'sick', 'ugh', 'hurts', and 'sorry' are indicative of negative sentiment. The text emojis :) and :(are the strongest indicators of positive and negative sentiment respectively, which makes sense given how the tweets were chosen and classified initially. Interestingly the 2 twitter users @justinbieber and @uber are both associated with negative sentiment, which may be an indicator of popular public opinion of those two users at the time the tweets were gathered. These features also suggest that if our classifier extends poorly to tweets in the wild, we might need to do more preprocessing of the training set, such as removing special characters and twitter users. It is worth noting that the NLTK tweet tokenizer has an optional parameter that allows for the removal of twitter usernames from the text during tokenization.

1.2 CLASSIFY DATASET

Here we use our trained Naive Bayes classifier to classify the Politwoops dataset of deleted tweets. Before classification we again use the word_feats() function to construct a featstruct out of our tokenized dataset for the classifier, as we did with NLTK's corpus before training.

In [12]:
```
# import dataset
deletweet = pandas.read_csv('../../deletweet/data/deleted_tweets_cleaned.csv')
```

In [13]:
```
# construct a list of strings to hold the tweet text
tweet_text_raw = []

for i in range(len(deletweet)):
    tweet = json.loads(deletweet['tweet'][i])
    tweet_text_raw.append(tweet['text'])
```

In [14]:
```
# number of individual tweets to classify
len(tweet_text_raw)
```

Out[14]: 67756

```
In [15]: # use the tweet tokenizer provided by NLTK
         # preserve_case=False will transform all to lowercase
         tknzr = TweetTokenizer(preserve_case=False)
         tokenized = [tknzr.tokenize(tweet_text_raw[i]) for i in range(len(tweet_text_raw))]

In [16]: # run tokenized lists through word_feats() to construct featstruct for NLTK classifier
         features = [word_feats(tokenized[i]) for i in range(len(tokenized))]

In [17]: # classify!
         classed = classifier.classify_many(features)

In [18]: classified_tweets = list(zip(classed, tweet_text_raw))
```

1.3 PLOT AND ANALYSIS

The first 28 tweets from the Politwoops dataset are printed out below, along with their classified sentiment. After reading through these and other subsets of the classified dataset we can see that the classifier performs fairly well, although not perfectly, as indicated by the very first tweet being misclassified as negative. There is definite room for improvement, and a more robust feature set for training would likely go a long way. Expanding on the original search by specifically gathering tweets of a political nature in association with positive and negative text emojis would likely help ameliorate the deficiencies of the classifier.

Also after examining these results, it seems that the biggest improvement would come from the addition of a third neutral class: many of the tweets are ambiguous in sentiment, and therefore do not fit well into either the positive or negative classes.

```
In [26]: for i in range(30):
             # don't print 7th tweet; emojis breaking export to pdf
             if i != 6 and i != 19:
                 print('{} : {}'.format(classified_tweets[i][0], classified_tweets[i][1]))
```

```
neg : This is so cool. This same sort of adaptive protocol is being used with shipping drones as well. I
pos : https://t.co/V7Rc07GrJU
pos : #TBT @MikePenceVP https://t.co/tSZUjMjaaI
pos : I had a cordial and candidate discussion today with the new DHS Secretary, John Kelly. https://t.c
pos : Grt to host @USProgressives Specl Order w/@RepRaskin on #MuslimBan.Thx @RepMarkTakano @RepLawrenc
pos : I'm an original co-sponsor of @RepDonBeyer's Freedom of Religon Act, protecting our values in resp
pos : @IAVA CEO @PaulRieckhoff &amp; I are going #Head2Head to determine who dons the better 'do. Post #
pos : @IAVA CEO @PaulRieckhoff &amp; I are going #Head2Head to determine who dons the better 'do. Post #
pos : @IAVA CEO @PaulReickhoff &amp; I are going #Head2Head to determine who dons the better 'do. Reply
pos : @IAVA CEO @PaulReickhoff and I are going #Head2Head to determine, once and for all, who dons the b
pos : @IAVA CEO @PaulReickhoff &amp; I are going #Head2Head to determine, once and for all, who dons the
pos : .@HouseGOP have privately (&amp; rightfully) expressed fears about what #ACARepeal would mean for
pos : Not to worry. @realDonaldTrump promises to deliver a sensible, coherent plan for #MiddleEast peac
neg : Right now, Voting NO on going to Executive Session for nomination Price, Mnuchin and Sessions. No
pos : These words take on new meaning in the #Trump Administration. https://t.co/TKHksDSGjn
pos : .@HouseGOP have privately (&amp; rightfully) expressed fears about what #ACARepeal would mean for
pos : .@SenateMajLdr McConnell comments on measure --&gt; Video here: https://t.co/0yxGDq8cY6
@WLKY https://t.co/WBq3aIm8Sc
pos : .@SenateMajLdr McConnell comments on passage of anti-coal measure: https://t.co/K1ENddRMtH https:/
neg : Tune into the now LIVE forum to hear from panelists, including Dr. Kahn about the #MuslimBan http
pos : JOBS: The AR1 will provide the US with a new, world-competitive engine for launch vehicles, 100 j
pos : We're working to ensure the hiring freeze does not prevent the @forestservice from preparing for w
pos : A new era of transparency begins at the @FCC  Thank you @AjitPaiFCC and @mikeoreilly https://t.co/
pos : .@MichStatePolice still working hard to track down Officer Collin Rose's killer, but they need yo
```

```
pos : .@realDonaldTrump What's your stance on painkillers? Beta-endorphins invented at #UCBerkeley
neg : Discrimination under the guise of "religious freedom" is still discrimination. I urge @POTUS not t
https://t.co/v2g9tOSzXP
pos : @Fortunatebri I have no doubt. But I also know I can do a better job of jackasses currently in the
pos : Great news from @AjiPaiFCC today { promise to make @FCC more open &amp; transparent, giving radio
neg : RT @zenbeatnik: @Scotttaylorva But executed by Trump. The blaming of Obama must stop.
```

In [17]:
```python
# seperate the classification into positive and negative buckets
pos_class = [thing for thing in classed if thing =='pos']
neg_class = [thing for thing in classed if thing == 'neg']
```

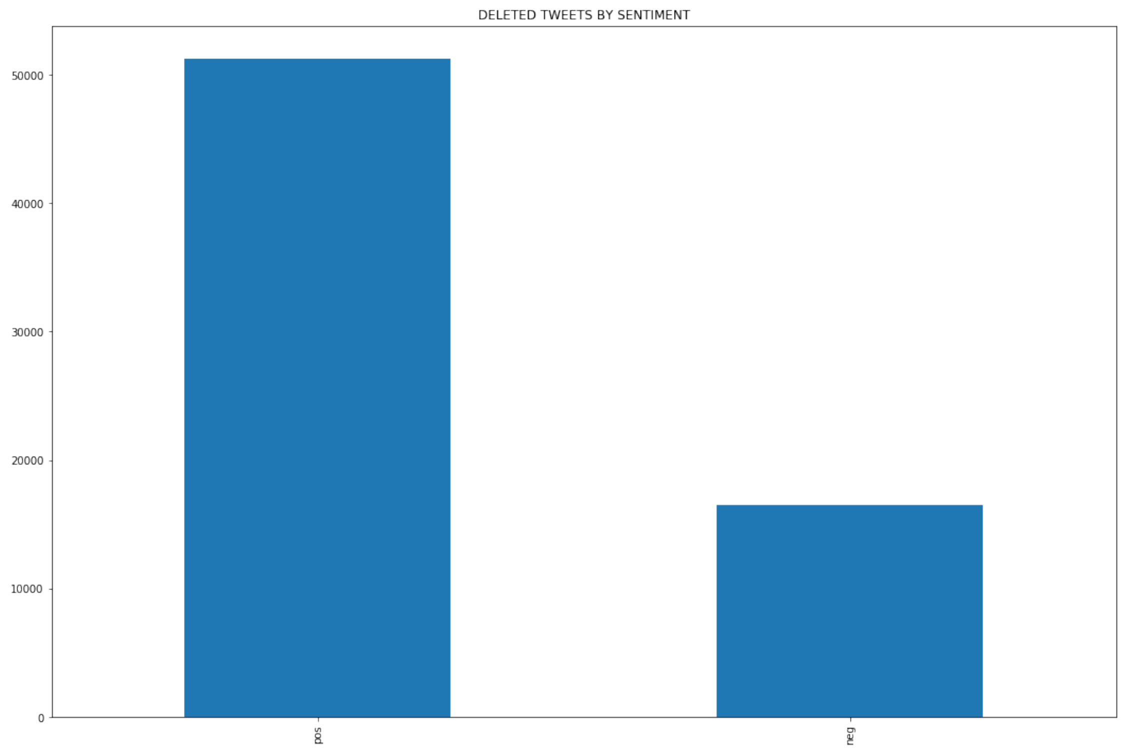
In [18]:
```python
# calculate class percentage of whole
split = [len(pos_class), len(neg_class)]
print('positive class: {:,} tweets - {:.2%}'.format(split[0], split[0]/(len(classed))))
print('negative class: {:,} tweets - {:.2%}'.format(split[1], split[1]/(len(classed))))
```

```
positive class: 51,260 tweets - 75.65%
negative class: 16,496 tweets - 24.35%
```

In [19]:
```python
split_series = pandas.Series(split, index=['pos', 'neg'])
```

In [20]:
```python
split_series.plot.bar(title='DELETED TWEETS BY SENTIMENT')
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11cf05b70>
```



1.4 CONCLUSION

The classifier split the tweets into 51,260 positive tweets and 16,496 negative tweets. The weight toward the positive class may be related to the political context of the tweets: politicians are likely to use twitter to express positive sentiment about their actions or things they support, such as policy or legislation. This is perhaps unique to political tweets in that the regular population's tweets are likely to be more personal and quotidian in nature, and therefore may not be so heavily weighted toward the positive class. Also, as stated above, the addition of a neutral class would change this distribution significantly, although we would expect more positive classes to change to neutral in that case than negative to neutral.

deletweet-hashtag-recommender

April 4, 2017

1 DELETWEET HASHTAG RECOMMENDER

In [11]:
```
import csv
import json
import pandas
from random import randint
```

1.1 USE RESERVOIR SAMPLING TO SUBSET THE DATA

In order to cluster the data in a reasonable amount of time, a subset of the original dataset was needed. The reservoir sampling algorithm implemented for this class was used to gather a truly random sample of 10,000 tweets.

This section is just a demonstration and does not represent the actual subset used for clustering.

In [12]:
```
# import dataset and extract tweets to feed algorithm
deletweet = pandas.read_csv('../../deletweet/data/deleted_tweets_cleaned.csv')
tweets = deletweet['tweet']
```

In [13]:
```
def reservoir(stream, k):
    '''
    Populate sample space with first k items from stream. For remaining items in stream,
    choose a random number j from 0 to item's index. If j is less than k, replace
    jth element in sample with ith element from stream
    '''
    sample = stream[0:k]

    for i in range(k, len(stream)):
        j = randint(0, i + 1)
        if j < k:
            sample[j] = stream[i]

    return sample
```

In [14]:
```
# take the sample
samples = reservoir(tweets, 10000)
```

In [15]:
```
# make sure we have 10,000 tweets
len(samples)
```

Out[15]: 10000

In [16]:
```
# take a look at first sample
tweet = json.loads(samples[0])
tweet['text']
```

```
Out[16]: "Enjoyed another opportunity to visit with Nebraskans at today's Open Coffee in Holdrege. Than
```

1.2 EXAMINE SUBSET USED FOR CLUSTERING

```
In [17]: tweets = {}
         hashtag_dict = {}

In [18]: # import subset used for clutering
         with open('../data/deletweet_subset_10000.json', 'r') as f:

             for line in f:
                 tweet = json.loads(line)
                 tweets[tweet['id']] = tweet

In [19]: # number of tweets in subset
         len(tweets)

Out[19]: 10000

In [20]: # extract hashtags from tweets in subset
         for key in tweets.keys():
             tweet = tweets[key]
             if (tweet['entities']['hashtags']):
                 hashtag_dict[key] = [tweet['entities']['hashtags'][i]['text'] \
                                      for i in range(len(tweet['entities']['hashtags']))]

In [21]: # how many tweets in the sample have hashtags
         len(hashtag_dict)

Out[21]: 4830

In [22]: tags = []
         for key in hashtag_dict.keys():
             for tag in hashtag_dict[key]:
                 tags.append(tag)

         # how many individual hashtags are in the sample
         len(tags)

Out[22]: 7821

In [23]: # how many unique hashtags are in the sample
         len(set(tags))

Out[23]: 4300
```

1.3 CLUSTER VIA JACCARD DISTANCE AND K-MEANS

Based on the average number of unique hashtags in any subset of this dataset - which is roughly 40% of the number of tweets in the subset - 1,000 was the number of clusters chosen. This seemed to strike a balance between the uniqueness of the clusters and the likelihood that a cluster would have at least one hashtag in it that could be recommended to other tweets in the cluster.

The clustering of 10,000 tweets into 1,000 clusters was done via K-Means clustering using Jaccard Distance as the distance metric. The implemenatation of the algorithm used is open source, although I updated the code to make it run under Python 3.

The clustering took over 50 hours to complete, so the code will not be included here.

1.4 IMPORT CLUSTERS

```
In [24]: clusters = pandas.read_csv('../data/clusters_1000_from_10000_reformat.csv')
```

```
In [25]: tweet_ids_int = []

         for i in range(len(clusters)):
             nums = []
             stripped = clusters['tweet_ids'][i].strip('{}').split(', ')

             for num in stripped:
                 nums.append(int(num))

             tweet_ids_int.append(nums)

         clusters['tweet_ids_int'] = tweet_ids_int

         # same as above:
         # tweet_ids_int = [[int(num) for num in clusters['tweet_ids'][i].strip('{}').split(', ')] \
         # for i in range(len(clusters))]
```

```
In [26]: # parse output of clustering algorithm into following format:
         #
         # {
         # cluster_01_id: {tweet_01_id: tweet_01_text}, [...], {tweet_n_id: tweet_n_text},
         # [...],
         # cluster_n_id: {tweet_01_id: tweet_01_text}, [...], {tweet_n_id: tweet_n_text}
         # }
         #
         # this takes about an hour, need to optimize
         clusters_dict = {}

         for i in range(len(clusters)):
             clusters_dict[i] = {}
             for j in range(len(deletweet)):
                 tweet_id = deletweet['id'][j]
                 if tweet_id in clusters['tweet_ids_int'][i]:
                     clusters_dict[i][tweet_id] = deletweet['content'][j]
```

1.5 EXTRACT AND RECOMMEND HASHTAGS

```
In [27]: # clusters to use as demonstration
         demo_clusters = [3, 108, 150, 263, 374, 453, 509]
```

```
In [28]: # look at the tweets in a cluster
         print('cluster_id: {}'.format(demo_clusters[0]))
         for key in clusters_dict[demo_clusters[0]].keys():
             print('{tweetid}: {tweettext}'.format(tweetid=key, tweettext=clusters_dict[demo_clusters[0]
```

```
cluster_id: 3
309057990382714880: With @SenBlumenthal at today's Veterans of Foreign Wars hearing. #vets @DeptVetAffa
228553954458488832: As result of #HCR, more than 5.2M seniors &amp; people with disabilities have saved
783109774753792002: A lot of people can't handle it. |Trump on veterans with post-traumatic stress https
169840713356423168: Gulf Coast Vietnam Veterans Salute http://t.co/wDbzAQUm
174895471893028865: Met with Fred S. Sganga, Exec. Dir. of LI State Veterans Home. He briefed me on the
```

```
In [29]: # extract hashtags from the tweets in the cluster
         cluster_tags = []

         for key in clusters_dict[demo_clusters[0]].keys():
             for i in range(len(deletweet)):
                 if deletweet['id'][i] == key:
                     tweet = json.loads(deletweet['tweet'][i])
                     # pull out hashtags from tweet object
                     for tag in [tweet['entities']['hashtags'][i]['text'] \
                                 for i in range(len(tweet['entities']['hashtags']))]:
                         if tag not in cluster_tags:
                             cluster_tags.append(tag)

         for item in cluster_tags:
             print('#{}'.format(item))
```

```
#vets
#HCR
#SticksandStones
#SAU
#NY9
```

This first example is typical of most of the clusters, and is a good indicator of the difficulties of trying to set up a hashtag recommendation system.

On a general level the tweets in the above cluster have to do with veterans and health care. More specifically, 3 out of the 4 tweets mention veterans, while the outlier tweet is about health care and seniors, the latter of which happens to apply to many veterans. The cluster is a good condidate for a hashtag recommendation system, as the tweets have two overlapping, but separate themes from which to pull potential hashtags. And in fact, if our system were to recommend the hashtag #vets from the first tweet to the others in the cluster, it would be applicable to 2 out of the 3 remaining tweets. The next hashtag - #HCR, an acronym for "health care reform" - would be slightly less successful, in that it is applicable to only 1 out of the 3 remaining tweets.

The last hashtag - #NY9, which stands for New York's 9th congressional district - presents a more complicated scenario, and one that appears frequently when attempting to recommend hashtags in this manner. While the tweet itself has been clustered correctly based on its text content, the hashtag is specific enough that the chances of it being applicable to another tweet in the dataset is very low.

This situation arose many times when looking through the clusters; for example, tweets are correctly clustered together due to the fact that they all deal with sports, but the hashtags present in the cluster all reference specific teams, and therefore are not applicable to the other tweets in the cluster.

This difficulty may be less of a problem as the size of the input dataset grows, as more data would presumably allow the clusters to become more specific. Another, potentially concurrent, method to alleviate this difficulty is to increase the number of clusters, also allowing for more specificity. However, both these potential solutions come at large computational costs, and the system is already prohibitvely costly in this domain. Increasing the number of clusters also lowers the number of hashtags available to each cluster, which is a disadvantage for recommendation.

```
In [30]: # look at the tweets in a cluster
         print('cluster_id: {}'.format(demo_clusters[1]))
         for key in clusters_dict[demo_clusters[1]].keys():
             print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                                   tweettext=clusters_dict[demo_clusters[1]][key]))
```

```
cluster_id: 108
746031782907129856: RT @RepKClark: 24 hours ago, we began this sit-in to demand votes on common sense gu
745821640970342401: RT @RepKClark: Staying on the House floor to demand a vote on gun safety bills. #NoB
```

```
745732604293320704: RT @WoodsGoods: Ty to @dinatitus who is holding her ground in support common sense g
745779309663518720: 9hrs and still on the House floor feeling united and proud to stand up as one to rec
```

```
In [31]: # extract hashtags from the tweets in the cluster
         cluster_tags = []

         for key in clusters_dict[demo_clusters[1]].keys():
             for i in range(len(deletweet)):
                 if deletweet['id'][i] == key:
                     tweet = json.loads(deletweet['tweet'][i])
                     # pull out hashtags from tweet object
                     for tag in [tweet['entities']['hashtags'][i]['text'] \
                                 for i in range(len(tweet['entities']['hashtags']))]:
                         if tag not in cluster_tags:
                             cluster_tags.append(tag)

         for item in cluster_tags:
             print('#{}'.format(item))
```

```
#Yes2Wes
#MascotMania
#LittleLeague
```

Cluster 108 is a a successful example for the recommendation system. Each tweet has the hashtag #NoBillNoBreak, but one tweet in the cluster has 2 more that are highly applicable to the group, since all the tweets in the cluster deal with holding the floor to push for better gun safety policy: #NoMoreSilence and #HoldTheFloor.

```
In [32]: # look at the tweets in a cluster
         print('cluster_id: {}'.format(demo_clusters[3]))
         for key in clusters_dict[demo_clusters[3]].keys():
             print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                             tweettext=clusters_dict[demo_clusters[3]][key]))
```

```
cluster_id: 263
529816984637034496: RT @acberka: Just voted--now it's your turn! #republican #StandWithDan #goandvote ht
523959270736674816: RT @BlueDevil83: @Hogan4Governor Your youngest fan!! HoganForMD http://t.co/BQJwjbNa
407556287627399168: RT @smidgiekroger: @RepDennyHeck @RedCross check with your local American Legion Aux
523620595289030656: RT @KeevAdams3: Let's get t shirts printed #288MillionWentWhere @Hogan4Governor
520017037645864960: RT @zapeters: Ran into Maryland's next Governor tonight - @Hogan4Governor #mdpolitic
```

```
In [33]: # extract hashtags from the tweets in the cluster
         cluster_tags = []

         for key in clusters_dict[demo_clusters[3]].keys():
             for i in range(len(deletweet)):
                 if deletweet['id'][i] == key:
                     tweet = json.loads(deletweet['tweet'][i])
                     # pull out hashtags from tweet object
                     for tag in [tweet['entities']['hashtags'][i]['text'] \
                                 for i in range(len(tweet['entities']['hashtags']))]:
                         if tag not in cluster_tags:
                             cluster_tags.append(tag)

         for item in cluster_tags:
             print('#{}'.format(item))
```

```
#republican
#StandWithDan
#goandvote
#288MillionWentWhere
#mdpolitics
```

Cluster 263 is a good example of a cluster with mixed success: #goandvote could apply to all the tweets; #mdpolitics and #288MillionWentWhere could apply to the 3 tweets that mention @HoganForGovernor; #StandWithDan is presumably not applicable to any of the other tweets in the cluster

In [34]:
```python
# look at the tweets in a cluster
print('cluster_id: {}'.format(demo_clusters[5]))
for key in clusters_dict[demo_clusters[5]].keys():
    print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                          tweettext=clusters_dict[demo_clusters[5]][key]))
```

```
cluster_id: 453
266882923960094720: Update on #A #subway: Read the latest on rebuilding in the #Rockaways after Hurrican
269122784813273089: 'We Will Lead on #Climate Change' | Read the Gov's op-ed in @nydailynews: http://t.c
262232359192129536: Breaking: Gov's Dir of State Operations Howard Glazer, MTA Chairman Joseph Lhota, PA
265215852331278337: Stay Up To Date on Hurricane Sandy Recovery Efforts | #Sandy
http://t.co/ATTnkJAg via @energy
```

In [35]:
```python
# extract hashtags from the tweets in the cluster
cluster_tags = []

for key in clusters_dict[demo_clusters[5]].keys():
    for i in range(len(deletweet)):
        if deletweet['id'][i] == key:
            tweet = json.loads(deletweet['tweet'][i])
            # pull out hashtags from tweet object
            for tag in [tweet['entities']['hashtags'][i]['text'] \
                        for i in range(len(tweet['entities']['hashtags']))]:
                if tag not in cluster_tags:
                    cluster_tags.append(tag)

for item in cluster_tags:
    print('#{}'.format(item))
```

```
#A
#subway
#Rockaways
#sandy
#Climate
#Sandy
```

Cluster 453 is another example of a partially successful cluster: the first 2 hashtags are unlikely to be applicable to any of the other tweets, with the exception of potentially the 3rd tweet. However the hashtags #Climate and #Sandy have high likelihood of being applicable to all the tweets in the cluster.

While analyzing the clusters, an unforeseen situation arose in which a different kind of hashtag recommendation system might have some success. This is applicable to the clusters in which all the tweets have highly correlated content, but which have no existing hashtags. In this scenario, hashtags could be recommended based purely on the content of the clusters, most likely by turning a word common to all the tweets into a hashtag. This could be made more sophisticated and robust by searching in realtime to see if there are relevant popular hashtags on Twitter.

We do not try to implement such a system here, but present some examples for potential future consideration.

```
In [36]: hashtags_from_content = [22, 31, 34, 146]
```

```
In [37]: # look at the tweets in a cluster
         print('cluster_id: {}'.format(hashtags_from_content[0]))
         for key in clusters_dict[hashtags_from_content[0]].keys():
             print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                                   tweettext=clusters_dict[hashtags_from_content[0]][key
```

```
cluster_id: 22
630483076171309056: They oppose equal pay for equal work.
522718529364443136: 1.3 million homeless students... http://t.co/yaMbgiTQkN
519914410031067136: ...a pay raise for over 25 million American workers... http://t.co/M1E7WZne3R
453607544230248448: RT @RepKevinBrady: All people deserve Equal Pay for Equal Work. Period. http://t.co/
```

cluster 22: #EqualPay4EqualWork

```
In [38]: # look at the tweets in a cluster
         print('cluster_id: {}'.format(hashtags_from_content[1]))
         for key in clusters_dict[hashtags_from_content[1]].keys():
             print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                                   tweettext=clusters_dict[hashtags_from_content[1]][key
```

```
cluster_id: 31
702975909566029824: Thank you for https://t.co/qTtuzipdS2
739253869050494977: @TeamTrumpNC thank you. https://t.co/YHFOwjEhhj
703104748153499650: Thank you! Trump2016 #GOPDebate https://t.co/aV9rR1zl7o
175614937446617088: @TheSmak Thank you!
630050327371321345: Thank you #CruzCrew! #RSG15 #CruzCountry http://t.co/auapF8wQAi
466422915215675392: @MJLeavitt thank you!
461248994035773441: Thank you... http://t.co/v4ALozeg6T http://t.co/DkFdb8Wuj2
375932319884128256: RT @jtylerharrison: @RepRickCrawford thank you
266934031541743617: THANK YOU - http://t.co/qj7HEXBd
466553301652090880: @Jennifer_K1691 thank you!
203471882912137216: @DanVForbes Thank you. Appreciate the message
312724010863575040: @USAFVeteran1 thank you!
476559119466258432: @PolitiBunny Thank you!
459093074636181505: This has been a blessing, thank you! http://t.co/dPVMqMCoPg
423663010217881600: Thank you...
161518376991207424: @Sloup Well, thank you 'Sloup'
451419821976977408: Thank you sir \@daaman81: @NealMarchbanks daaman81@gmail.com"
```

cluster 31: #thankyou

```
In [39]: # look at the tweets in a cluster
         print('cluster_id: {}'.format(hashtags_from_content[2]))
         for key in clusters_dict[hashtags_from_content[2]].keys():
             print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                                   tweettext=clusters_dict[hashtags_from_content[2]][key
```

```
cluster_id: 34
808806195523911680: I liked a @YouTube video https://t.co/c681wo0UT6 Jay Z Gets Embarrassed By An Old Ra
```

```
622157970084888576: Add a message to your video http://t.co/hpbd1mRVlI
603271586091835394: I added a video to a @YouTube playlist http://t.co/pStiw1iZZX Obama Adminstration Fa
456614333561061380: I added a video to a @YouTube playlist http://t.co/ra93f6Zhgz Dr. Chad Mathis: Leade
353243512831098881: I added a video to a @YouTube playlist http://t.co/omrRXEboOT Gov. Abercrombie Appo
353243514148098048: I added a video to a @YouTube playlist http://t.co/lPrZ5Kb5p5 Senator Daniel K. Aka
590987507032002561: Into'd a #RachelCarson res., thanks to activists like her bald eagles are back. Ste
175302378776567808: I added a video to a @YouTube playlist http://t.co/igRQjoPI NBC Channel 11: Rep. Sc
353243514139717633: I added a video to a @YouTube playlist http://t.co/oZhzFRvNrY 6/19/12 News conferen
603275295848816641: I added a video to a @YouTube playlist http://t.co/tws34arkHF Rep. Collins: This Re
487245128919052289: I added a video to a @YouTube playlist http://t.co/XfUd36pWQO McDermott on Immigrat
144891694213644289: I liked a @YouTube video from @larrymendte http://t.co/K4znTAkI Let Buddy Roemer De
161828958894170114: I liked a @YouTube video http://t.co/DGz84Sw9 YouTube Town Hall: Where your view co
353241077098098688: I added a video to a @YouTube playlist http://t.co/y4lZfH6EYW Bill Signing for HB43
251817162522648576: I added a video to a @YouTube playlist http://t.co/KzOw6ocH Bannock Development Cor
253186784421347328: I uploaded a @YouTube video http://t.co/olKeBAW4 Idaho Parks Passport Program
400383072689860608: We owe our nation's #veterans a tremendous debt of gratitude. I was honored to spen
603275278446665729: I added a video to a @YouTube playlist http://t.co/uP6AsrteaV Rep. Collins: High Ed
353241075823034369: I added a video to a @YouTube playlist http://t.co/FqzNRj5HiA Bill Signing SB856 (C
603275278484373504: I added a video to a @YouTube playlist http://t.co/5RQHYmnqPD Congressman Collins T
353241075525222400: I added a video to a @YouTube playlist http://t.co/KaNd0RKuLl Kamuela Wassman Day
603275278429851648: I added a video to a @YouTube playlist http://t.co/SwesdmyNCh Collins speaks in fav
372746452612956161: I added a video to a @YouTube playlist http://t.co/w7A3KKPnD5 McNerney discusses th
175302379128881152: I added a video to a @YouTube playlist http://t.co/h9r2xBBd Rep. Scott Tipton Q&A d
603278040571977728: I added a video to a @YouTube playlist http://t.co/SaasQryIxz NSA: The New Four-Let
372746721270706176: I added a video to a @YouTube playlist http://t.co/0ly82irmM8 McNerney recognizing
157188236589006848: I uploaded a @YouTube video http://t.co/8yNbemAO Akin Update - KTRS Debate
353241534134632448: I added a video to a @YouTube playlist http://t.co/pGdfkkfQjq Sequestration Press C
353241077144236032: I added a video to a @YouTube playlist http://t.co/uKLHbqvJCE Bill Signing, SB1077
175302378780758016: I added a video to a @YouTube playlist http://t.co/14ruCXMs Rep. Scott Tipton Recap
353241077140029441: I added a video to a @YouTube playlist http://t.co/BjOeOr9erD Bill Signings, SB682
372743657604276224: I added a video to a @YouTube playlist http://t.co/dPQbSdHLtw Debate on Amendment t
175302378776563712: I added a video to a @YouTube playlist http://t.co/LPceXOSP Rep. Scott Tipton House
262577631596249088: I uploaded a @YouTube video http://t.co/muzgbLUp Two Big Differences
372746324112076800: I added a video to a @YouTube playlist http://t.co/85RDr8wiHg Congressman McNerney'
372728439146823680: I added a video to a @YouTube playlist http://t.co/50146GjvHV Rep McNerney calls fo
175302378747207681: I added a video to a @YouTube playlist http://t.co/2l3ITfzM Rep. Scott Tipton at Sm
```

cluster 34: #youtube

In [40]:
```
# look at the tweets in a cluster
print('cluster_id: {}'.format(hashtags_from_content[3]))
for key in clusters_dict[hashtags_from_content[3]].keys():
    print('{tweetid}: {tweettext}'.format(tweetid=key, \
                                          tweettext=clusters_dict[hashtags_from_content[3]][key
```

```
cluster_id: 146
410547295571046400: In case you missed it from The Wall Street Journal: How to Keep Workers Unemployed:
368388131558785024: In case you missed it, see my guest appearance on PBS NewsHour discussing the suppr
335066013500579842: "In case you missed it, here's clip from ABC's "This Week" talking about moms in Co
142388919860867072: In case you missed it, we passed more #jobs bills out of @FinanceCmte yesterday htt
```

cluster 146: #InCaseYouMissedIt

1.6 CONCLUSION

While some proof of concept work has been done toward a hashtag recommendation system, there are significant opportunities for improvement.

More specific text normalization has the potential to improve the robustness of the clusters. For example, links have very little information in this context, since they are generally shortened by twitter, and therefore any relevant text that may have existed in the original link is abstracted away.

More importantly, the current system is prohibitively computationally expensive, and it seems that this approach would require hardware resources that are currently inaccessible to the avaerage consumer. Therefore this system may only be applicable in a theoretical or research context, rather than a production environment.

deletweet-conclusion

April 3, 2017

0.1 CONCLUSION

Leveraging machine learning techniques such as text mining, natural language processing, and k-means clustering, we were able to attempt answers to the 4 questions established in the introduction.

Using NLTK's tools for text mining and NLP - including their domain-specific social media tokenizer - we were able to pull out the most common words and themes of the tweets in the Politwoops dataset. We found terms and phrases common to politics in general, as well as words and names specific to the timeframe of the dataset (2011-2017). Along the way we established the importance of normalization of the text to the success of this process.

Next we took a close look at the nature of the dataset as not only a set of tweets by politicians, but as a set of tweets that were specifically deleted by politicians. Referencing previous research done into deleted tweets, we established a threshold to distinguish tweets that were most likely deleted for aesthetic reasons from tweets that may have been deleted for other regrettable reasons. We performed content analysis on this subset, in an effort to distinguish it from the dataset as a whole. With the same goal in mind, we also analyzed the times tweets in the subset were created and deleted relative to the dataset as a whole. And while some differences were noted, nothing concrete was found to explicitly distinguish the two. While ore comparison is needed against a set of tweets by these same politicians that were not deleted, it seems that the vast majority of the tweets have been deleted for aesthetic reasons such as misspellings, improper formatting, broken links, etc.

Using NLTK's Naive Bayes Classifier, as well as their previously classified tweet corpus, we performed sentiment analysis on the dataset, with close to 75% of the tweets being classified as positive, with the remaining classified as negative. Introduction of a third, neutral class to this process is needed to make the results more robust.

Finally, we showed a proof of concept hashtag recommendation system that uses K-Means and Jaccard Distance to cluster similar tweets together and recommend hashtags from tweets in the cluster to its neighbors. While this system is effective, it is extremely computationally expensive, and therefore not suitable for use outside of a research domain.