BA515

Section 003

Group Project

Social Media Analytics

Group - 42

Hongyuan(Lion) Li (44123396)

Samual Chan (62741640)

Varshitha Narahari (13778527)

Yash Gupta (33169418)

Introduction:

People actively express their opinions in social media platforms such as Twitter, Facebook, Instagram, WeChat, Snapchat, etc. As analytics enthusiasts, our team wanted to take advantage of this opportunity by using our Python skills to conduct social media analysis. The process started by selecting two related key words and doing analysis on the five thousand tweets collected.

Part A: The keywords selected were "Brexit" and "Boris Johnson" as the key words for collecting twitter feed data for our project, as they are the most frequented topics in the UK. The process involved the following steps:

Process:

Step A: Create a developer account on Twitter to collect the tweets
Step B: Install the Twitter API Python package: *TwythonStreamer* (Appendix-A)
Step C: Import ba515twitter and JSON input credentials (Appendix-B)
Step D: Enter keyword "Bexit" and "Boris Johnson" and collect 5000 tweets each (Appendix-C)
Step E: Collect the twitter data as JSON files named: tweet_stream_Boris Johnson_5000.json and
tweet_stream_Brexit_5000.json (Appendix-D)

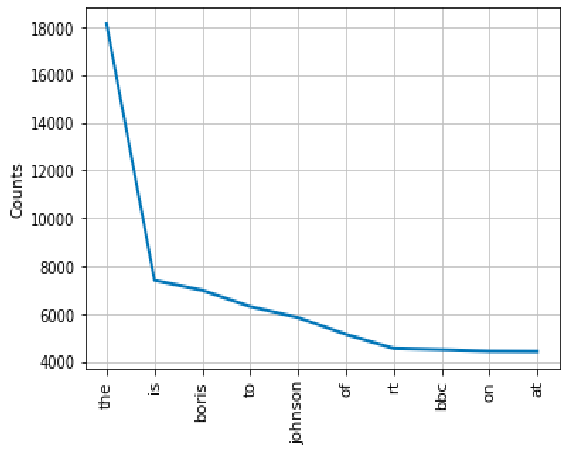
Part B: Preliminary Analysis

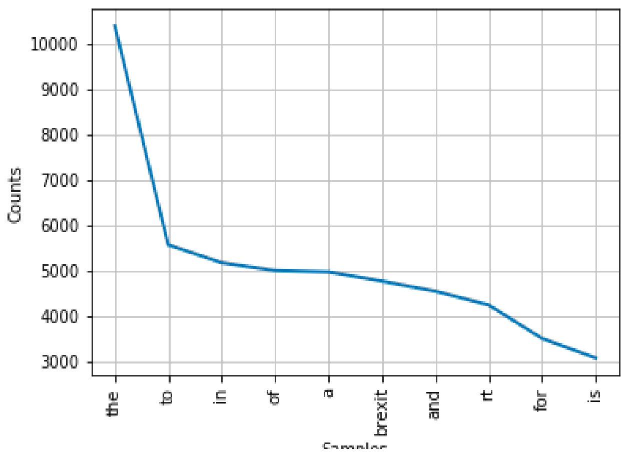
Process:

Step A: Imported all the required libraries and packages associated with the analysis (Appendix-E)
Step B: Set up the table for NLP (Appendix-F)
Step C: Create a library of stop-words (Appendix-F)
Note: The dictionary was modified and words – "https" and "rt" were added to the list
Step D: Use Tweet Parser to capture the quotes and retweets (Appendix-G)
Note: Tweet parser will parse each tweet and check if it is a quote, retweet, or just a regular tweet. It will also check to see the status of the relevant text section to see if it is truncated. If so, it will fetch the extended. Finally, it will put all the strings into a list of lists.
Step E: Parse the JSON Files and store them as lists of lists (Appendix-H)
Step F: Split the test using the code in Appendix - I

B.1: Ten most popular words with and without stop words

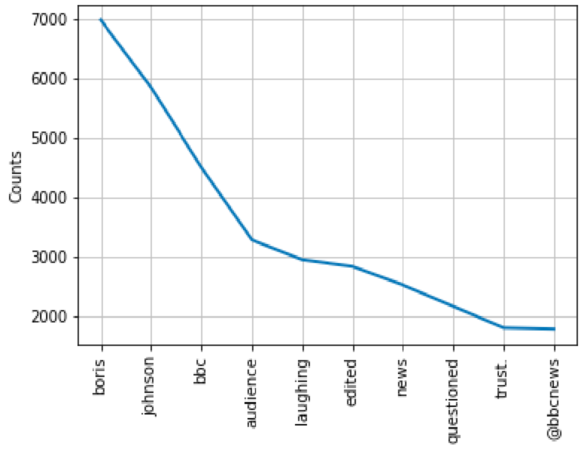
The ten most popular words without stop words are analysed using the code in Appendix-J.

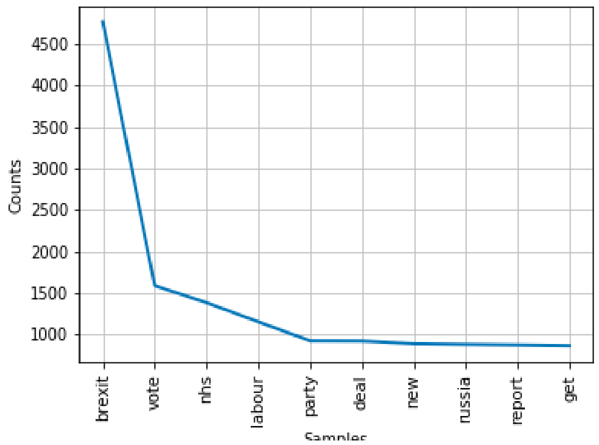




The most popular word without stopwords for both “Brexit” and “Boris Johnson” as keyword is “the” which is an article and does not relay much information or insights. Other similar not so informative words are prepositions and verbs- is, to, of etc. The nouns or information words from “Brexit” is brexit and from”Boris Johnson” are boris, johnson and bbc.

The ten most popular words with stop words are analysed using the code in Appendix-J.

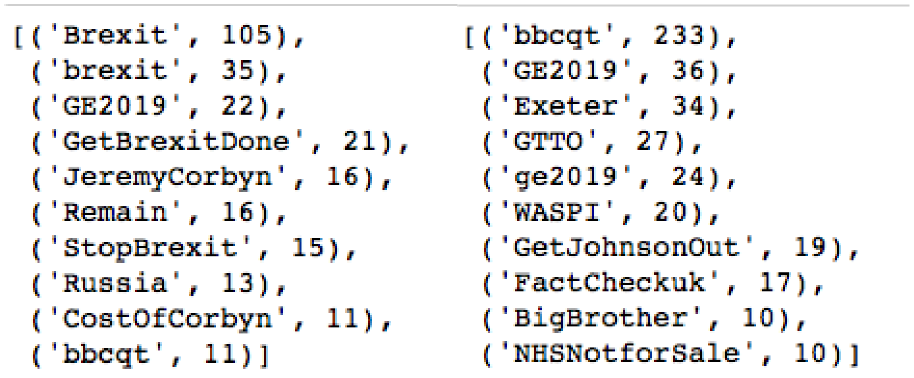




The most popular word with stopwords for both “Brexit” and “Boris Johnson” relay much more information or insights as compared to without using stop words.
The ten most popular words using the keyword “Brexit” are brexit , vote, nhs, labour, party, new, russia, report and get and the ten most popular words using the keyword ”Boris Johnson” are boris, johnson, bbc, audience, laughing, edited, news, questioned, trust and @bbcnews.

B.2: Ten most popular hashtags (#hashtag)

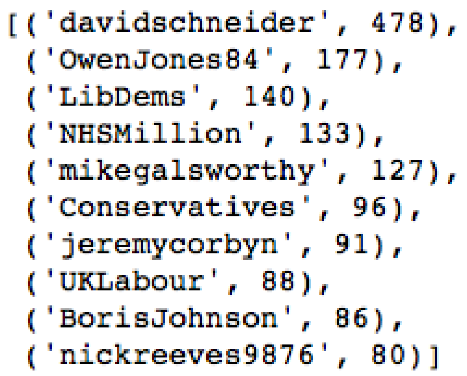
The ten most popular hashtags are analysed using the code in Appendix-K.



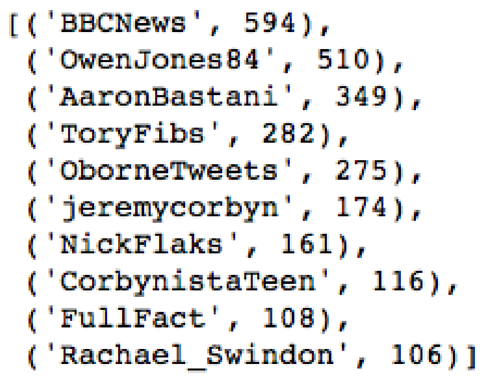
```
[('Brexit', 105),          [('bbcqt', 233),
 ('brexit', 35),            ('GE2019', 36),
 ('GE2019', 22),            ('Exeter', 34),
 ('GetBrexitDone', 21),     ('GTTO', 27),
 ('JeremyCorbyn', 16),      ('ge2019', 24),
 ('Remain', 16),            ('WASPI', 20),
 ('StopBrexit', 15),        ('GetJohnsonOut', 19),
 ('Russia', 13),            ('FactCheckuk', 17),
 ('CostOfCorbyn', 11),      ('BigBrother', 10),
 ('bbcqt', 11)]             ('NHSNotforSale', 10)]
```

B.3: Ten most frequently mentioned usernames (@username)

The ten most frequently mentioned usernames are analysed using the code in Appendix-L.



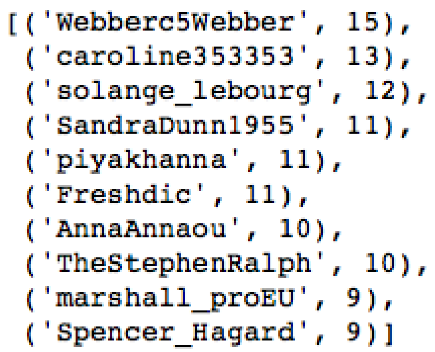
```
[('davidschneider', 478),
 ('OwenJones84', 177),
 ('LibDems', 140),
 ('NHSMillion', 133),
 ('mikegalsworthy', 127),
 ('Conservatives', 96),
 ('jeremycorbyn', 91),
 ('UKLabour', 88),
 ('BorisJohnson', 86),
 ('nickreeves9876', 80)]
```



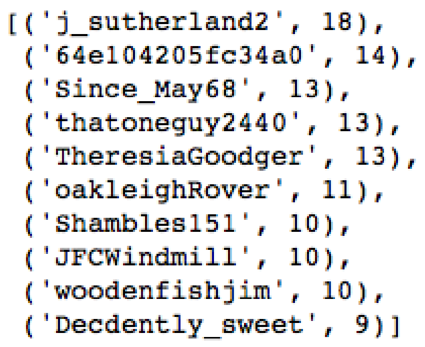
```
[('BBCNews', 594),
 ('OwenJones84', 510),
 ('AaronBastani', 349),
 ('ToryFibs', 282),
 ('OborneTweets', 275),
 ('jeremycorbyn', 174),
 ('NickFlaks', 161),
 ('CorbynistaTeen', 116),
 ('FullFact', 108),
 ('Rachael_Swindon', 106)]
```

B.4: The most vocal person on the keyword which means the most frequently tweeting person (tweet author) on the keyword

The ten most frequent tweeters about “Brexit” and “Boris Johnson” are analysed using the code in Appendix-M.



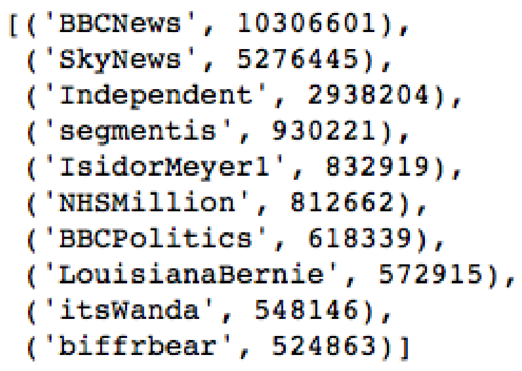
```
[('Webberc5Webber', 15),
 ('caroline353353', 13),
 ('solange_lebourg', 12),
 ('SandraDunn1955', 11),
 ('piyakhanna', 11),
 ('Freshdic', 11),
 ('AnnaAnnaou', 10),
 ('TheStephenRalph', 10),
 ('marshall_proEU', 9),
 ('Spencer_Hagard', 9)]
```



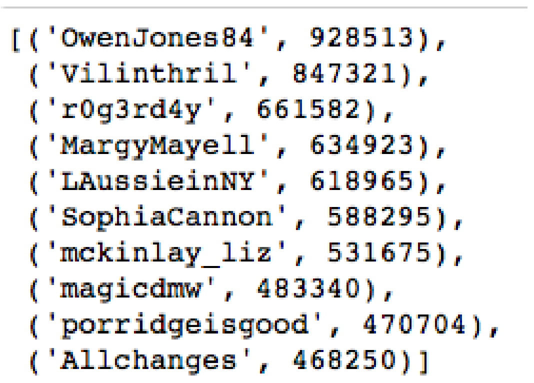
```
[('j_sutherland2', 18),
 ('64e104205fc34a0', 14),
 ('Since_May68', 13),
 ('thatoneguy2440', 13),
 ('TheresiaGoodger', 13),
 ('oakleighRover', 11),
 ('Shambles151', 10),
 ('JFCWindmill', 10),
 ('woodenfishjim', 10),
 ('Decdently_sweet', 9)]
```

B.5: The most influential person on the keyword

The ten most influential persons are analysed using the code in Appendix-N. A user’s influence score is defined as the sum of “followers_count”, “friends_count”, and “favourites_count” in the “user” sub-dictionary.



```
[('BBCNews', 10306601),
 ('SkyNews', 5276445),
 ('Independent', 2938204),
 ('segmentis', 930221),
 ('IsidorMeyer1', 832919),
 ('NHSMillion', 812662),
 ('BBCPolitics', 618339),
 ('LouisianaBernie', 572915),
 ('itsWanda', 548146),
 ('biffrbear', 524863)]
```



```
[('OwenJones84', 928513),
 ('Vilinthril', 847321),
 ('r0g3rd4y', 661582),
 ('MargyMayell', 634923),
 ('LAussieinNY', 618965),
 ('SophiaCannon', 588295),
 ('mckinlay_liz', 531675),
 ('magicdmw', 483340),
 ('porridgeisgood', 470704),
 ('Allchanges', 468250)]
```

Part C: Word Cloud

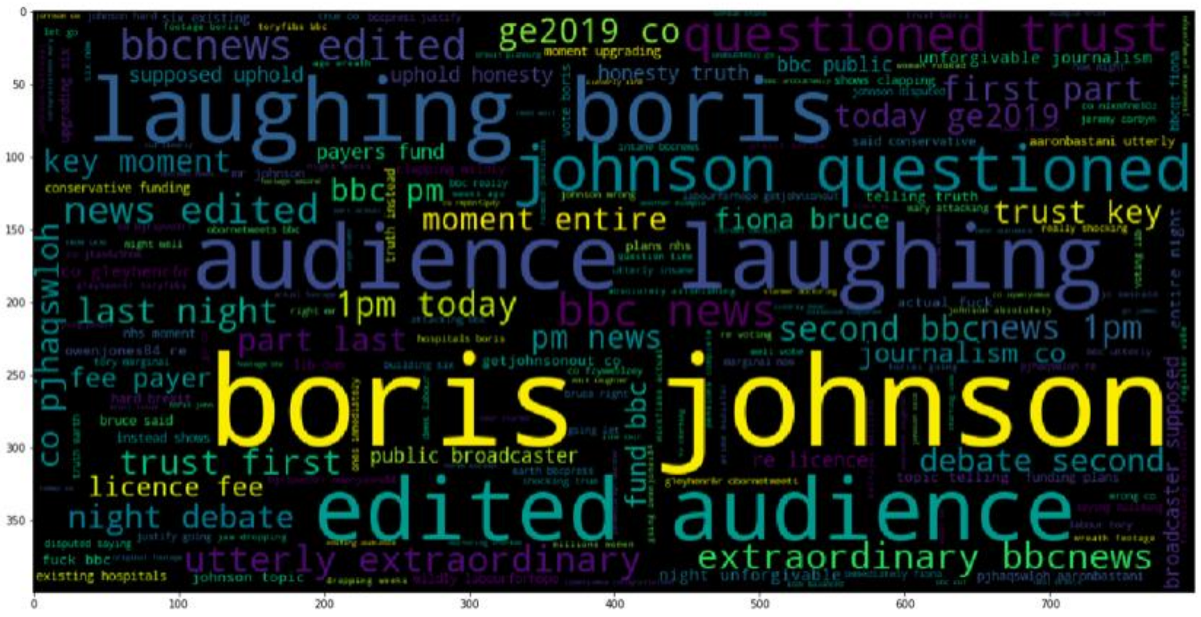
The word cloud is an excellent data presentation in python that enables the user to present data in a creative form. Word clouds are a fast, engaging and low-cost alternative for analyzing text from online surveys. They generators by breaking the text down into component words and counting how frequently they appear in the body of text Hein (2019). The bigger the word appears on the word cloud, the higher its frequency of appearance in the tweets.

Process

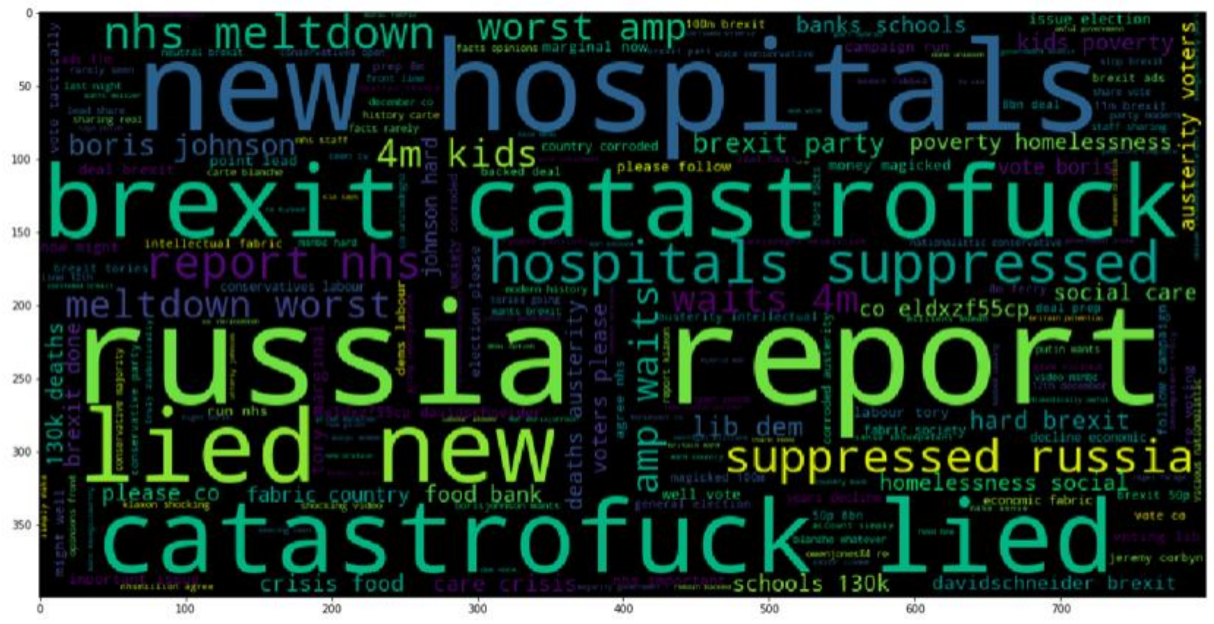
Step A: Imported all the required libraries and packages associated with the analysis (Appendix-E)
Step B: Create a library of stop-words (Appendix-F)
Note: The dictionary was modified and words – "https" and "rt" were added to the list
Step C: Create lists for both keywords without stop words. Use the list of individual words created for 'brexit' and 'boris johnson' as explained above in B.1.
Step D: Parse this list using the for loop and append only those words not in the stopwords list into a new list brexit_stopwords/bj_stopwords. (Appendix-G)
Step E: To increase code readability, create copies of these lists as 'brexit_wordcloud' and 'bj_wordcloud'.
Step F: Lower the max font size and generate the wordcloud as 'wordcloud = WordCloud(width=800, height=400).generate(brexit_wordcloud.translate(table_p))'
Step G: Display the generated wordcloud image using the matplotlib.pyplot package.

- Set the figure size as (20,10).
- Use 'plt.imshow' to display the image wordcloud.
- Turn on the axis lines as 'plt.axis("on")' .
- Save the wordcloud both as png and pdf using 'plt.savefig'.
- Show the word cloud in the Jupyter notebook using 'plt.show'.

The WordCloud for keyword 'Boris Johnson' is shown below. The most frequent words here as "boris" and "johnson". Some other words in decreasing order of frequency are laughing, boris, audience, edited, questioned and trust.



The WordCloud for keyword ‘Brexit’ is shown below. The most frequent words for this keyword are “russia” and “report”. Some other words in decreasing frequency are new, hospitals, lied, brexit and catastrofuck.



Part D: Sentiment Analysis

Sentiment analysis gives us a gauge on the emotions of the tweeters. Sentiment is measured by two parameters viz., polarity and subjectivity. Polarity tells us if the text is positive, negative or neutral. Subjectivity estimates how objective or subjective the text is.

D1: Average polarity and subjectivity scores for each keyword

Process

Step A: Install textblob and import TextBlob from textblob library analysis (Appendix-E)
Step B: Create a list of tweets for both keywords by making the text of each tweet an element of the list.
Step C: Create a library of stop-words (Appendix-F)
Note: The dictionary was modified and words – “https” and “rt” were added to the list
Step D: Loop each element in list of tweets and call the class TextBlob as ‘tb = TextBlob(str(<tweet>))’
Step E: Extract the polarity and subjectivity of the tweet as ‘ tb.sentiment.subjectivity’ and ‘tb.sentiment.polarity’.
Step F: Append these into lists _sub_list and _pol_list respectively.
Step G: Mean function on these lists gives the average subjectivity and polarity scores for each keywords.

Output

Average Subjectivity score Brexit:	0.45777069608768023
Average Polarity score Brexit:	0.02312315172734414
Average Subjectivity score Boris Johnson:	0.47158841721841954

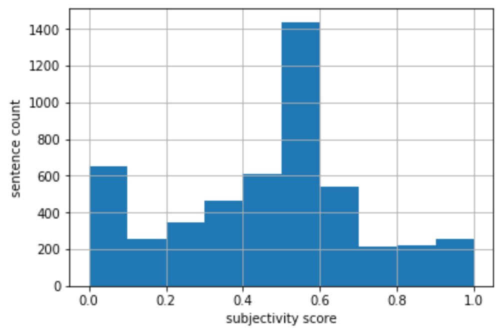
Average Polarity score Boris Johnson: 0.03324636912858966

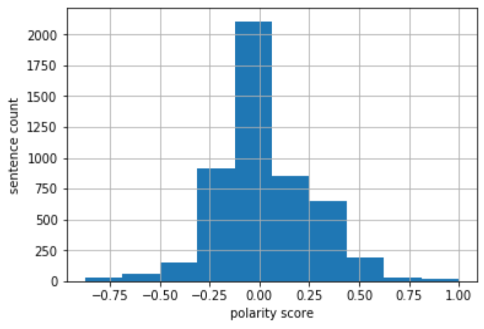
D2: Polarity and subjectivity score distribution visualization using histograms

Process

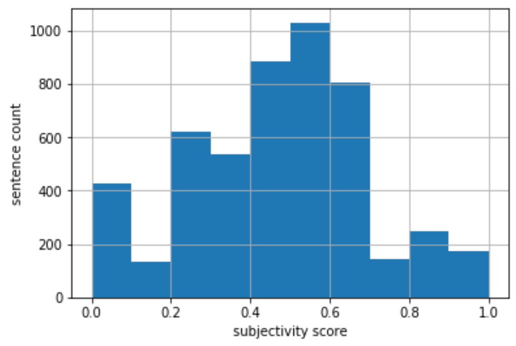
Step A: Import ‘matplotlib.pyplot’ package as plt
Step B: Plot the histogram of _sub_list and _pol_list lists using ‘plt.hist()’ function.
Step C: Set xlabel, ylabel and enable grid.
Step D: Save the histograms as pdf using ‘plt.savefig()’ function.
Step E: Use ‘plt.show()’ function to display the histogram on the Jupyter notebook.

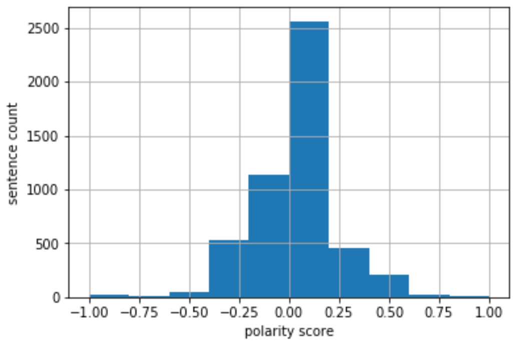
Output: For Brexit





Output: For Boris Johnson





D3. Most positive and negative tweets on each keyword

Process

Step A: Create empty variables min and max and initialize to 0.
Step B: Enumerate through the polarity list and compare the current polarity value with max variable until the highest value is found. Similarly, compare with min variable until the lowest value is found.
Step C: Obtain the indices of these max and min polarities.
Step D: Use the same indices on ‘brexit_tweets’ and ‘bj_tweets’ to obtain the text of the corresponding polarity values.

Output

Most positive tweet about Brexit

"RT @MichaelRosenYes: Until Corbyn tells me which way to vote on Brexit, I'm in pieces. I have no means to make up my own mind. He-e-e-lp!!!!"
Polarity: 1.0

The author seems to be confused and slightly sarcastic. However, the polarity is calculated as 1.0. The reason could be that there are no negative words coupled with Brexit directly present in the text.

Most negative tweet about Brexit

"@EmmaKennedy It's not going to happen. Swinson will support a Johnson govt if she can, and give these two reasons: (1) Lib Dems can mitigate worst aspects of Brexit, and (2) The Union: Johnson has promised not to grant #indyref2.

Stopping Brexit is not her priority."
Polarity: -0.875

The author uses a lot of negative words like not. The author does visibly seem to be angry.

Most positive tweet about Boris Johnson:

@smartysue Who did best on #bbcqt... ?

Jeremy Corbyn 12%
Nicola Sturgeon 9%
Jo Swinson 27%
Boris Johnson 52%

YouGov,
Polled 1,342 on 22 Nov, between 6pm-7pm.RT @CrankyOutrider: @smartysue Who did best on #bbcqt... ?

Jeremy Corbyn 12%
Nicola Sturgeon 9%
Jo Swinson 27%
Boris Johnson 52%

YouGov,…
Polarity: 1.0

This is a poll conducted. The author has retweeted the poll results. The polarity is calculated 1.0 as there are no negative words coupled with Boris Johnson. Also, he has the largest percent of votes. This could be assumed to be positive until context is given.

Most negative tweet about Boris Johnson:

“On Boris Johnson

"All an Eton education teaches you, is to tell outrageous lies in a posh accent, with some Latin words thrown in"

#GE2019 https://t.co/pIEHGJB0zTRT @Doozy_45: On Boris Johnson

"All an Eton education teaches you, is to tell outrageous lies in a posh accent, with some Latin words thro…”
Polarity: -1.0

This tweet seems to reflect anger. Negative words like outrageous might have resulted in the polarity -1. The author seems to be criticizing Boris Johnson.

Part E: Insights

Big data and analytics are the driving forces behind the total disruption in the ecosystem we live in. Today, we have more data, insights, and statistics at our fingertips than ever before, and continuously generating humongous data daily through smartphones, social media, and many other platforms and applications. Organizations can tap on the tremendous potential offered by the big-data and turn it into a competitive advantage.

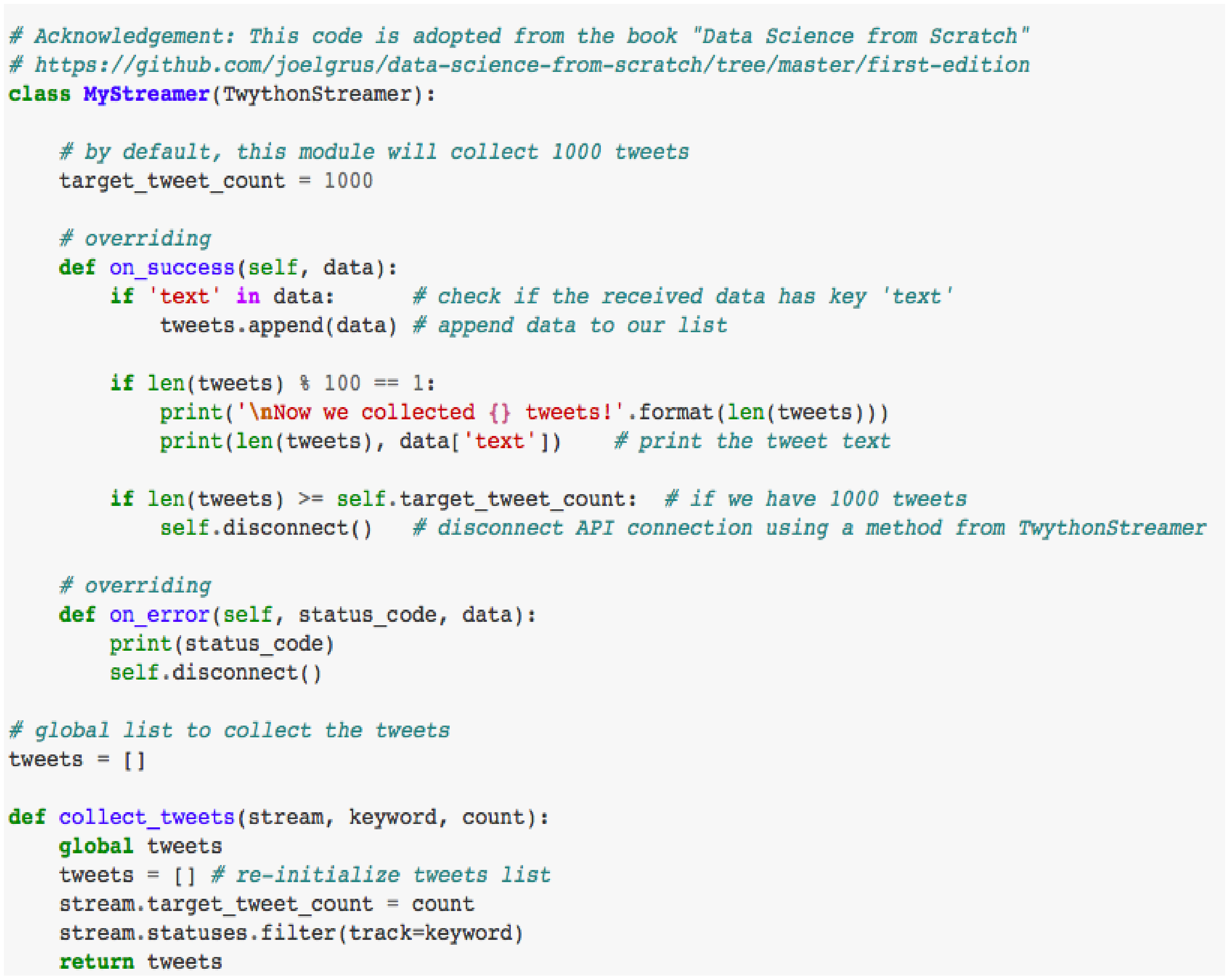
It is no secret that data insights from customers can reveal a lot of information about customers. The 2016 Trump election campaign results is a perfect example of the same. Cambridge Analytica is believed to have played a crucial role in Trump campaign as the able to create profiles of the voters using the information shared by them on Facebook and then use targeted digital ads to nudge their behaviour.

It was fascinating how twitter data analysis we conducted revealed a lot of similar information – we were able to gauge the sentiment of a particular word and the feelings and trend associated with it. The polarity results tell us if the text associations with a particular word are positive, negative or neutral. The average Polarity score of Brexit is 0.02, which means it is pretty neutral. The same was reflected in the Brexit results which swayed by a thing margin in favor of Brexit. Similarly, the average Polarity score of Boris Johnson is 0.03, which means it is pretty neutral. The support for Boris Johnson and his views was neutral in the UK. Subjectivity estimates how objective or subjective the text is. Subjective sentences generally refer to personal opinion, emotion or judgment whereas objective refers to factual information. Brexit has an average subjectivity score of 0.46 and average subjectivity score for Boris Johnson is 0.47. This means that the tweets are fairly opinionated and express a fair amount of opinion, emotion or judgement.

The companies can find so much about a “Brand” by analyzing the market sentiment associated with that word. The popularity index can help identify the influences that can help spread a positive sentiment about the brand. The companies can also identify the closest competitors and market leaders by doing a similar analysis across all the social media platforms.

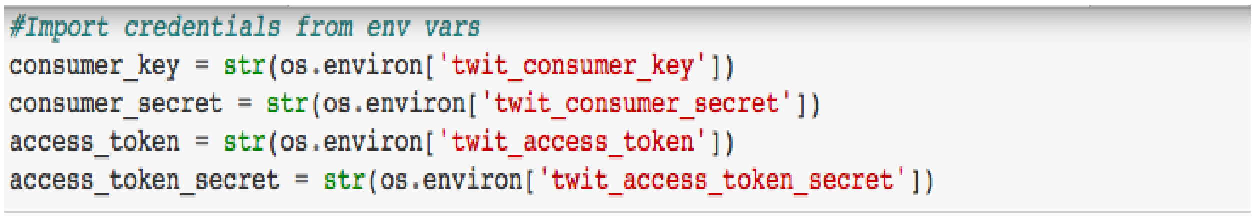
The analysis of tweets also helps the organizations to identify the problems that the customers are facing with their products which can help in two ways. First, it can be used a platform for problem resolution for the most common issues. Second, it can be used as inputs for product enhancement and development. If tapped optimally, the data can do wonders for an organization. No wonder, data is the oil of the next century.

Appendix-A: Twython Streamer Code

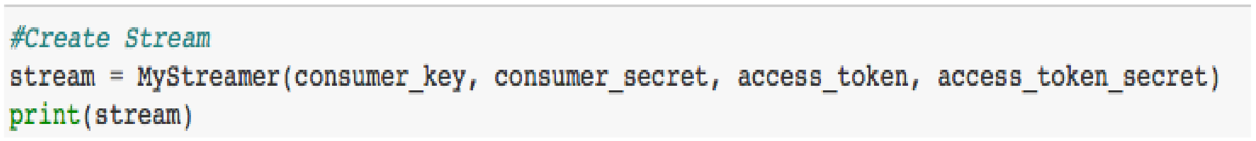


```python
# Acknowledgement: This code is adopted from the book "Data Science from Scratch"
# https://github.com/joelgrus/data-science-from-scratch/tree/master/first-edition
class MyStreamer(TwythonStreamer):

    # by default, this module will collect 1000 tweets
    target_tweet_count = 1000

    # overriding
    def on_success(self, data):
        if 'text' in data:      # check if the received data has key 'text'
            tweets.append(data) # append data to our list

        if len(tweets) % 100 == 1:
            print('\nNow we collected {} tweets!'.format(len(tweets)))
            print(len(tweets), data['text'])     # print the tweet text

        if len(tweets) >= self.target_tweet_count:  # if we have 1000 tweets
            self.disconnect()   # disconnect API connection using a method from TwythonStreamer

    # overriding
    def on_error(self, status_code, data):
        print(status_code)
        self.disconnect()

# global list to collect the tweets
tweets = []

def collect_tweets(stream, keyword, count):
    global tweets
    tweets = [] # re-initialize tweets list
    stream.target_tweet_count = count
    stream.statuses.filter(track=keyword)
    return tweets
```

Appendix-B: Code for importing ba515twitter and JSON input credentials



```python
#Import credentials from env vars
consumer_key = str(os.environ['twit_consumer_key'])
consumer_secret = str(os.environ['twit_consumer_secret'])
access_token = str(os.environ['twit_access_token'])
access_token_secret = str(os.environ['twit_access_token_secret'])
```



```python
#Create Stream
stream = MyStreamer(consumer_key, consumer_secret, access_token, access_token_secret)
print(stream)
```

Appendix-C: Code to collect 5000 tweets

```
# keyword = 'Boris Johnson'
# #keyword = ''
# print('My keywords is ' + keyword)
```

```
# start collecting 5000 tweets with keyword
# tweets = collect_tweets(stream, keyword, 5000)
# print(len(tweets), 'tweets are collected')
```

```
# with open('tweet_stream_{}_{}.json'.format(keyword, len(tweets)), 'w') as outfile:
#     json.dump(tweets, outfile, indent=4)
```

```
keyword = 'Brexit'
#keyword = ''
print('My keywords is ' + keyword)
```

```
My keywords is Brexit
```

```
# # start collecting 5000 tweets with keyword
#  tweets = collect_tweets(stream, keyword, 5000)
# print(len(tweets), 'tweets are collected')
```

```
# with open('tweet_stream_{}_{}.json'.format(keyword, len(tweets)), 'w') as outfile:
#     json.dump(tweets, outfile, indent=4)
```

Appendix-D: Code to save the tweets as JSON files

```
# Open files
with open('tweet_stream_Brexit_5000.json') as infile:
    brex_data = json.load(infile)

with open('tweet_stream_Boris Johnson_5000.json') as infile:
    bj_data = json.load(infile)

# Put both datsets into a list for easier handling
data = {'brex_data' : brex_data, 'bj_data' : bj_data}
```

Appendix-E: Code to import the required libraries and packages

```
%matplotlib inline
import os
import json
from pprint import pprint
import matplotlib.pyplot as plt

from twython import TwythonStreamer
from collections import Counter

import string
import nltk

from wordcloud import WordCloud
from textblob import TextBlob
from statistics import mean
```

Appendix-F: Code to set up the table for Natural Language Processing (NLP)

```
# Set trans table for NLP
puncs = string.punctuation
dgts = string.digits
table_p = str.maketrans(puncs, len(puncs) * " ")

# Create stopword Dict
stopwords = nltk.corpus.stopwords.words('english')
stopwords.append('rt')
stopwords.append('https')
```

Appendix-G: Code to parse the tweets

```
def tweet_text_parse(tweet_json,tweet_list):
    """
    Takes in a twitter API JSON file and an empty list, then finds the longest version of the retw
eet,
    quote, and regular text file.  It will then remove any duplicates among the three and then app
end
    the results to the list.
    """
    for count, x in enumerate(tweet_json):
        #Reset Variables
        quote = ''
        rt = ''
        text = ''

        # Check if this tweet is a quote
        # some tweets include is_quote_status but not quoted_status
        if 'is_quote_status' and 'quoted_status' in x.keys():
                if x['quoted_status']['truncated']:
                    quote = (x['quoted_status']['extended_tweet']['full_text'])
                else:
                    quote = (x['quoted_status']['text'])

        # Check if this tweet is a RT
        if 'retweeted_status' in x.keys():
            if x['retweeted_status']['truncated']:
                rt = (x['retweeted_status']['extended_tweet']['full_text'])
            else:
                rt = (x['retweeted_status']['text'])
        if x['truncated']:
            text = (x['extended_tweet']['full_text'])
        else:
            text = (x['text'])

        # Remote Duplicate Text
        if quote in text:
            quote = ''
        if rt in text:
            rt = ''
        tweet_list.append([quote]+[rt]+[text])
```

Appendix-H: Code to **parse the JSON Files and store them as lists of lists**

```
tweet_list_brexit = []
tweet_list_boris_johnson = []

tweet_text_parse(data['brex_data'],tweet_list_brexit)
tweet_text_parse(data['bj_data'],tweet_list_boris_johnson)
```

Appendix-I: Text Split and stop words code

```
brexit_words =[]
brexit_stopwords = []

for tweet in tweet_list_brexit:
    for text in tweet:
        brexit_words += text.lower().split()

for words in brexit_words:
    if words not in stopwords and len(words) > 1:
        brexit_stopwords.append(words)
```

```
bj_words =[]
bj_stopwords = []

for tweet in tweet_list_boris_johnson:
    for text in tweet:
        bj_words += text.lower().split()

for words in bj_words:
    if words not in stopwords and len(words) > 1:
        bj_stopwords.append(words)
```

Appendix-J: Code to analyze the most popular words

```
brexit_words =[]
brexit_stopwords = []

for tweet in tweet_list_brexit:
    for text in tweet:
        brexit_words += text.lower().split()

for words in brexit_words:
    if words not in stopwords and len(words) > 1:
        brexit_stopwords.append(words)
```

```
bj_words =[]
bj_stopwords = []

for tweet in tweet_list_boris_johnson:
    for text in tweet:
        bj_words += text.lower().split()

for words in bj_words:
    if words not in stopwords and len(words) > 1:
        bj_stopwords.append(words)
```

```
# Boris Johnson word frequency without stopwords
freq = nltk.FreqDist(bj_stopwords)
freq.plot(10)
```

```
# Boris Johnson word frequency with stopwords
freq = nltk.FreqDist(bj_words)
freq.plot(10)
```

```
# Brexit word frequency without stopwords
freq = nltk.FreqDist(brexit_stopwords)
freq.plot(10)
```

```
# Brexit word frequency
freq = nltk.FreqDist(brexit_words)
freq.plot(10)
```

Appendix-K: Code to identify the most popular hashtags

```
brexit_hashtag_count = []
brexit_mention_list = []
for x, y in enumerate(data['brex_data']):
    hashtag = data['brex_data'][x]['entities']['hashtags']
    if len(hashtag) > 0:
        brexit_hashtag_count.append(hashtag[0]['text'])
    mentions = data['brex_data'][x]['entities']['user_mentions']
    for ment in mentions:
        brexit_mention_list.append(ment['screen_name'])


bj_hashtag_count = []
bj_mention_list =[]
for x, y in enumerate(data['bj_data']):
    hashtag = data['bj_data'][x]['entities']['hashtags']
    if len(hashtag) > 0:
        bj_hashtag_count.append(hashtag[0]['text'])
    mentions = data['bj_data'][x]['entities']['user_mentions']
    for ment in mentions:
        bj_mention_list.append(ment['screen_name'])
```

```
ht = Counter(brexit_hashtag_count)
ht.most_common(10)
```

```
htbj = Counter(bj_hashtag_count)
htbj.most_common(10)
```

Appendix-L: Code to determine the most popular udernames

```
brexit_hashtag_count = []
brexit_mention_list = []
for x, y in enumerate(data['brex_data']):
    hashtag = data['brex_data'][x]['entities']['hashtags']
    if len(hashtag) > 0:
        brexit_hashtag_count.append(hashtag[0]['text'])
    mentions = data['brex_data'][x]['entities']['user_mentions']
    for ment in mentions:
        brexit_mention_list.append(ment['screen_name'])


bj_hashtag_count = []
bj_mention_list =[]
for x, y in enumerate(data['bj_data']):
    hashtag = data['bj_data'][x]['entities']['hashtags']
    if len(hashtag) > 0:
        bj_hashtag_count.append(hashtag[0]['text'])
    mentions = data['bj_data'][x]['entities']['user_mentions']
    for ment in mentions:
        bj_mention_list.append(ment['screen_name'])
```

```
b = Counter(brexit_mention_list)
b.most_common(10)
```

```
bj = Counter(bj_mention_list)
bj.most_common(10)
```

Appendix-M: Code to identify the frequent tweeters

```
tweet_count_brex = {}
for tweet in data['brex_data']:
    person_id = tweet['user']['screen_name']
    if person_id not in tweet_count_brex.keys():
        tweet_count_brex[person_id] = 1
    else:
        tweet_count_brex[person_id] += 1

tweet_count_bj = {}
for tweet in data['brex_data']:
    person_id = tweet['user']['screen_name']
    if person_id not in tweet_count_bj.keys():
        tweet_count_bj[person_id] = 1
    else:
        tweet_count_bj[person_id] += 1
```

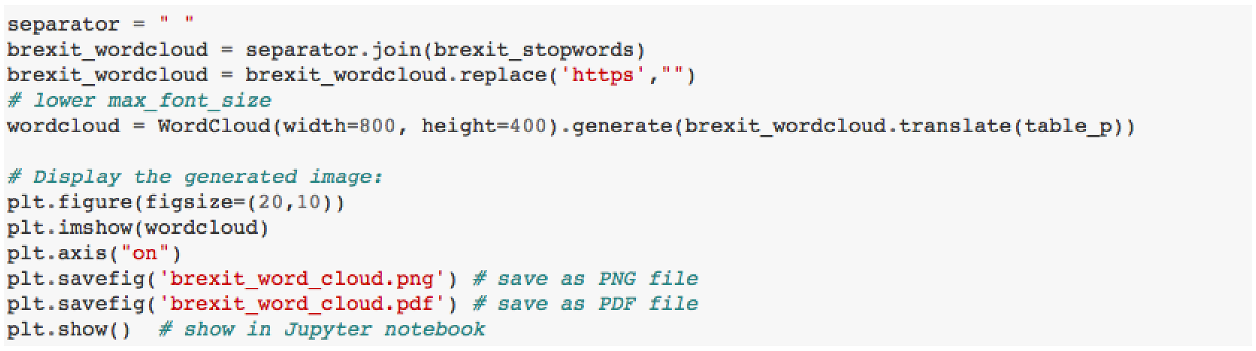
```
c = Counter(tweet_count_bj)
c.most_common(10)
```

```
c = Counter(tweet_count_brex)
c.most_common(10)
```

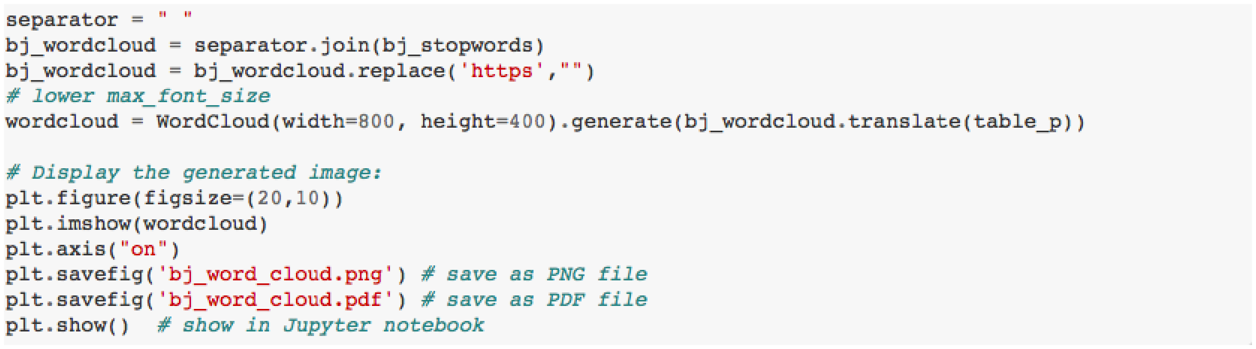
Appendix-N: Code to identify the most influential persons

```
person_score_brex = dict()

for tweet in data['brex_data']:
    person_id = tweet['user']['screen_name']
    follower_count = tweet['user']['followers_count']
    friend_count = tweet['user']['friends_count']
    favourites_count = tweet['user']['favourites_count']
    person_score_brex[person_id] = follower_count + friend_count + favourites_count

person_score_bj = dict()

for tweet in data['bj_data']:
    person_id = tweet['user']['screen_name']
    follower_count = tweet['user']['followers_count']
    friend_count = tweet['user']['friends_count']
    favourites_count = tweet['user']['favourites_count']
    person_score_bj[person_id] = follower_count + friend_count + favourites_count
```

Appendix-O: Code for word cloud



```
separator = " "
brexit_wordcloud = separator.join(brexit_stopwords)
brexit_wordcloud = brexit_wordcloud.replace('https',"")
# lower max_font_size
wordcloud = WordCloud(width=800, height=400).generate(brexit_wordcloud.translate(table_p))

# Display the generated image:
plt.figure(figsize=(20,10))
plt.imshow(wordcloud)
plt.axis("on")
plt.savefig('brexit_word_cloud.png') # save as PNG file
plt.savefig('brexit_word_cloud.pdf') # save as PDF file
plt.show()  # show in Jupyter notebook
```



```
separator = " "
bj_wordcloud = separator.join(bj_stopwords)
bj_wordcloud = bj_wordcloud.replace('https',"")
# lower max_font_size
wordcloud = WordCloud(width=800, height=400).generate(bj_wordcloud.translate(table_p))

# Display the generated image:
plt.figure(figsize=(20,10))
plt.imshow(wordcloud)
plt.axis("on")
plt.savefig('bj_word_cloud.png') # save as PNG file
plt.savefig('bj_word_cloud.pdf') # save as PDF file
plt.show()  # show in Jupyter notebook
```

References:

Hein, R. (2019, April 24). The Pros and Cons of Word Clouds as Visualizations. Retrieved from https://www.visioncritical.com/blog/pros-and-cons-word-clouds-visualizations.