2021/22

MSc Web and Social Media Analytics

twitter notes

Tutorial Pack 8
(90 minutes)

(To be completed during LW11 tutorial)

LEARNING OBJECTIVES

- **To perform exploratory analysis using WordClouds and the Python Counter class**
- **To practice applying the k-means clustering algorithm to group tweets into themes**

LEARNING OUTCOMES

- **By the end of this tutorial students will have; ○ Revised key Python programming concepts**
  - **Practiced interacting with the V2 Twitter API using both the streaming and historic methods.**
  - **Developed experience writing API queries using rulesets ○ Explored the use of regular expressions to aid formatting of data files**
  - **Conducted an initial exploratory analysis to identify prominent keywords and phrases**
- **Applied the k-means approach to group tweets by their TF-IDF scores**

RESOUCRES AND TOOLS REQUIRED

- **Python 3 via repl.it**
- **Lecture Slides for LW 10**
- **Rstudio (from apps anywhere)**

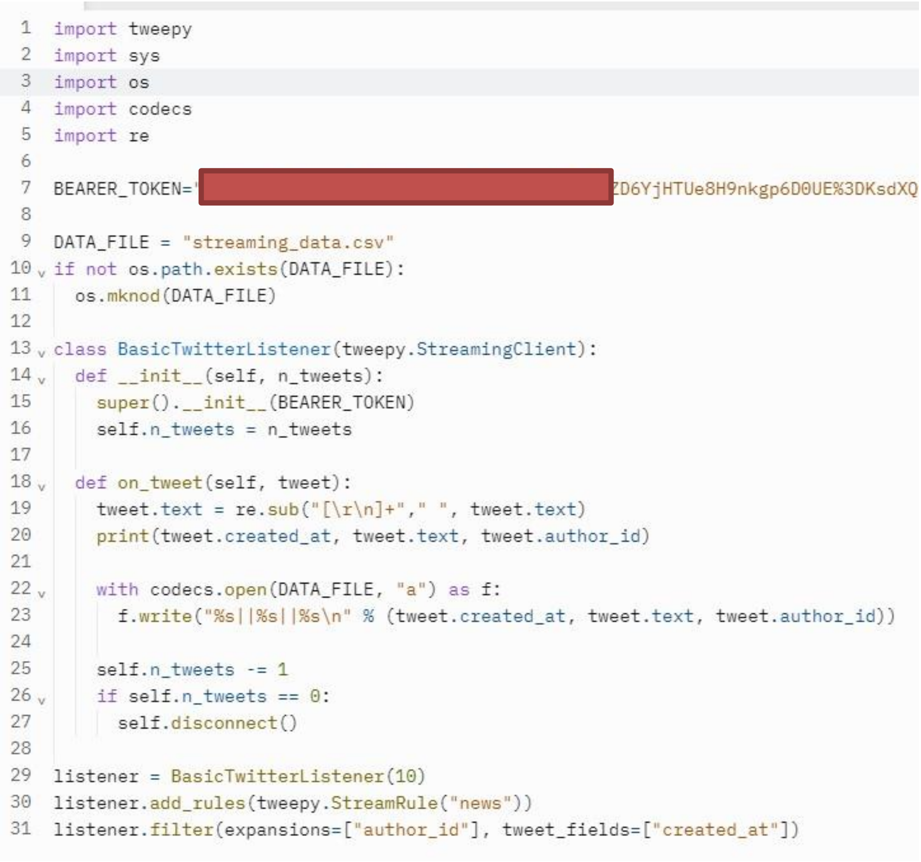
IMPORTANT:
This pack is designed for you to go at your own speed. At the end of each section there is a series of practice questions and exercises. You should attempt to answer **all questions**.

<u>Any questions you do not complete today should be completed before your next tutorial</u>.

You may find this tutorial pack and the exercises useful preparation for the coursework.

1. Streaming API example (10 mins)

The following code demonstrates how to listen for tweets using the streaming APIv2 twitter #1



```
 1  import tweepy
 2  import sys
 3  import os
 4  import codecs
 5  import re
 6
 7  BEARER_TOKEN='[redacted]ZD6YjHTUe8H9nkgp6D0UE%3DKsdXQ
 8
 9  DATA_FILE = "streaming_data.csv"
10  if not os.path.exists(DATA_FILE):
11    os.mknod(DATA_FILE)
12
13  class BasicTwitterListener(tweepy.StreamingClient):
14    def __init__(self, n_tweets):
15      super().__init__(BEARER_TOKEN)
16      self.n_tweets = n_tweets
17
18    def on_tweet(self, tweet):
19      tweet.text = re.sub("[\r\n]+"," ", tweet.text)
20      print(tweet.created_at, tweet.text, tweet.author_id)
21
22      with codecs.open(DATA_FILE, "a") as f:
23        f.write("%s||%s||%s\n" % (tweet.created_at, tweet.text, tweet.author_id))
24
25      self.n_tweets -= 1
26      if self.n_tweets == 0:
27        self.disconnect()
28
29  listener = BasicTwitterListener(10)
30  listener.add_rules(tweepy.StreamRule("news"))
31  listener.filter(expansions=["author_id"], tweet_fields=["created_at"])
```

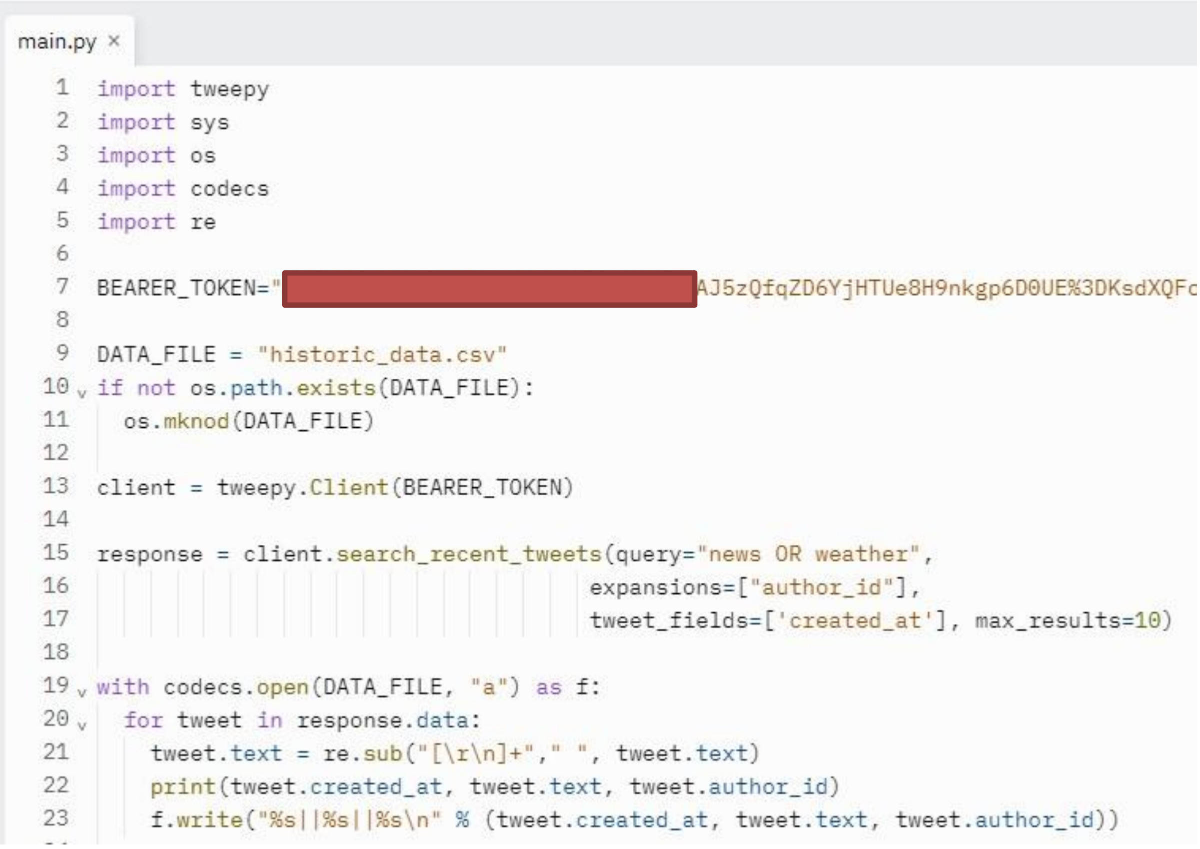
Study the above code and attempt to answer the following questions.

QNum	Question	Answer
1	On which lines is a file called "streaming_data.csv" created if it doesn't already exist?	
2	Which line numbers contain code that controls the number of tweets that are collected?	
3	What happens when the requested number of tweets are collected?	
4	On which line is data about each tweet written to a file?	
5	When a new tweet arrives, which three fields are saved to the "streaming_data.csv" file?	
6	Which line number controls the tweets that will be listened for?	

2. Historic API example (5 mins)

The following code demonstrates how to listen for tweets using the Historic APIv2

twitter #2



main.py ×

```
1   import tweepy
2   import sys
3   import os
4   import codecs
5   import re
6
7   BEARER_TOKEN="[redacted]AJ5zQfqZD6YjHTUe8H9nkgp6D0UE%3DKsdXQFc
8
9   DATA_FILE = "historic_data.csv"
10  if not os.path.exists(DATA_FILE):
11    os.mknod(DATA_FILE)
12
13  client = tweepy.Client(BEARER_TOKEN)
14
15  response = client.search_recent_tweets(query="news OR weather",
16                                         expansions=["author_id"],
17                                         tweet_fields=['created_at'], max_results=10)
18
19  with codecs.open(DATA_FILE, "a") as f:
20    for tweet in response.data:
21      tweet.text = re.sub("[\r\n]+"," ", tweet.text)
22      print(tweet.created_at, tweet.text, tweet.author_id)
23      f.write("%s||%s||%s\n" % (tweet.created_at, tweet.text, tweet.author_id))
```

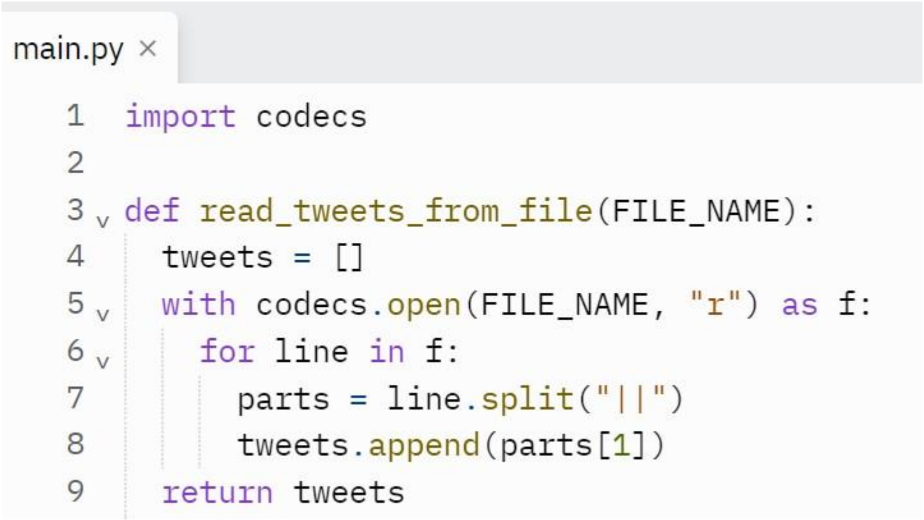
Study the above code and attempt to answer the following questions.

QNum	Question	Answer
7	Which two keywords are used to search for relevant tweets?	
8	On which line number can the number of tweets returned be controlled?	
9	How many tweets will appear n the historic_data.csv file if the above code is run twice?	
10	What pattern does the regular expression on line 21 match? Why might it be used here?	

3. Reading back collected tweets (5 mins)

In the two examples seen previously, the tweet text, author id and the date are fields retrieved and saved to a file name specified by the variable DATA_FILE.

The format of this file is as follows: each line represents a single tweet in the form <<created_at>>||<<tweet.text>>||<<tweet.author_id>>. The following function demonstrates how the **tweet text** of all tweets can be extracted from this file.



```
1  import codecs
2
3  def read_tweets_from_file(FILE_NAME):
4      tweets = []
5      with codecs.open(FILE_NAME, "r") as f:
6          for line in f:
7              parts = line.split("||")
8              tweets.append(parts[1])
9      return tweets
```

Study the above code and attempt to answer the following questions.

QNum	Question	Answer
11	How many parameters does the above function accept?	
12	What data type does the above function return?	
13	Explain the purpose of lines 5 and 6	
14	How might the above code be modified to extract only the author_ids?	

Exercise 1: (20 mins)

- Using the streaming API, collect a sample of 100 tweets on a subject of your choice and save them to a file called “historic_data.csv”. *Hint:* you will need to modify the sample code provided in the appendix.
- Using the streaming API listen for tweets on a topic of your choice until you have gathered 500 tweets. Save your tweets to “streaming_data.csv”

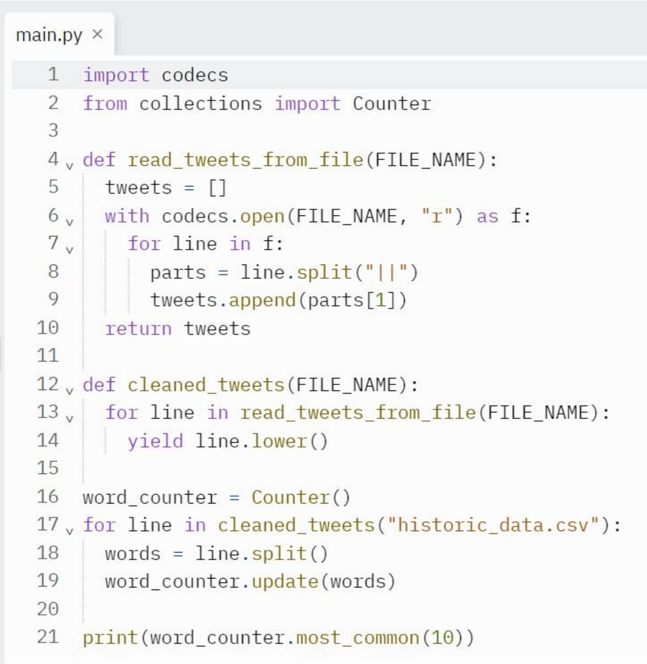
Exercise 2: (10 mins)

- Create a new repl and upload the data file obtained from exercise 1. Verify you can read in and print your collected tweets using the example code on page 4.
- Modify you code so that tweets are lowercased before being printed i.e., apply a lowercase pre-processing step.

4. Exploratory analysis

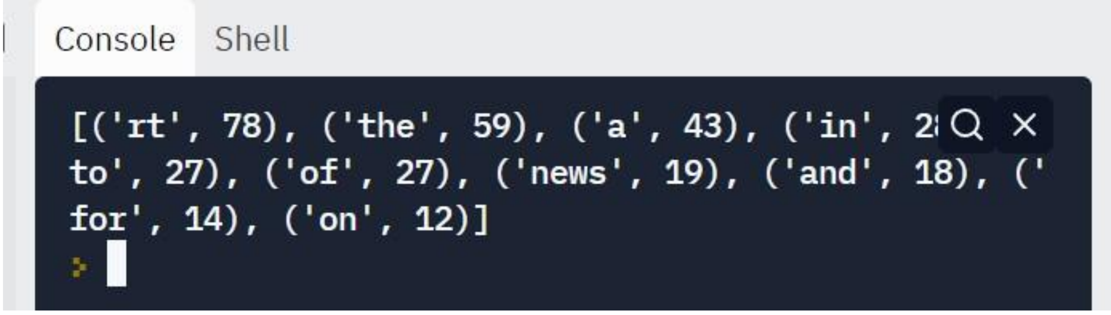
An initial exploratory analysis can tell us whether we have applied sufficient preprocessing and indeed chosen relevant keywords. A useful Python class for this purpose is the *Counter* class, which can be found in the collections package.

In the example below I demonstrate how to use the counter class by looping over all the collected tweets, splitting them into their respective words and updating the counter class. At the end of the process I can access the most commonly used words by calling the most_common() method.



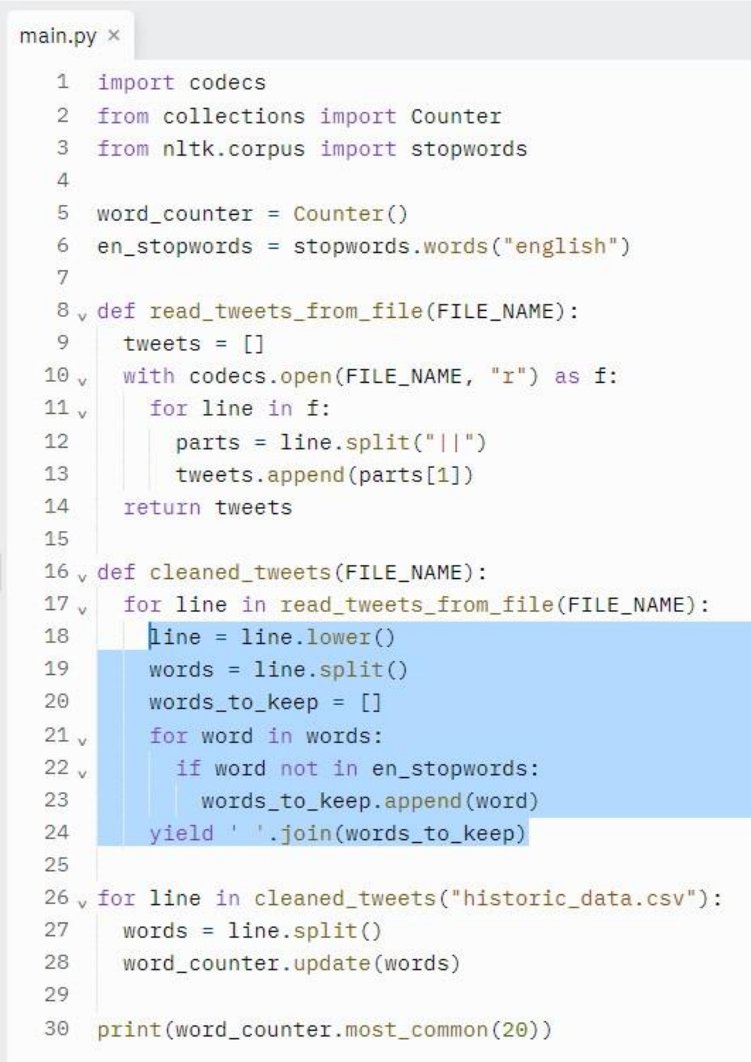
```python
import codecs
from collections import Counter

def read_tweets_from_file(FILE_NAME):
  tweets = []
  with codecs.open(FILE_NAME, "r") as f:
    for line in f:
      parts = line.split("||")
      tweets.append(parts[1])
  return tweets

def cleaned_tweets(FILE_NAME):
  for line in read_tweets_from_file(FILE_NAME):
    yield line.lower()

word_counter = Counter()
for line in cleaned_tweets("historic_data.csv"):
  words = line.split()
  word_counter.update(words)

print(word_counter.most_common(10))
```

The output of the counter is shown below. Among the top 10 most used words we can see the words “rt”, “the” and “a”. This indicates a weakness in our preprocessing step since we have not dealt with the issue of stopwords.



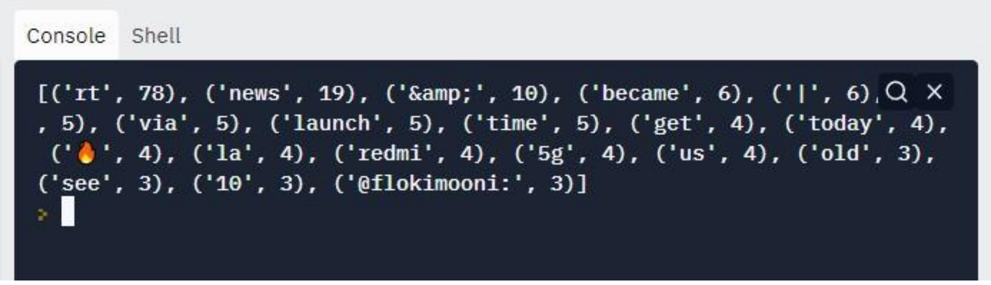
```
[('rt', 78), ('the', 59), ('a', 43), ('in', 2
to', 27), ('of', 27), ('news', 19), ('and', 18), ('
for', 14), ('on', 12)]
> 
```

We have already seen in LW9 that lists of stopwords can be obtained from the NTLK (natural language toolkit) package. In the example below, I have updated my code so that as part of the cleaned_tweets function, each word is checked against the list of English stopwords. If a word is found in the list of English stopwords it is excluded.



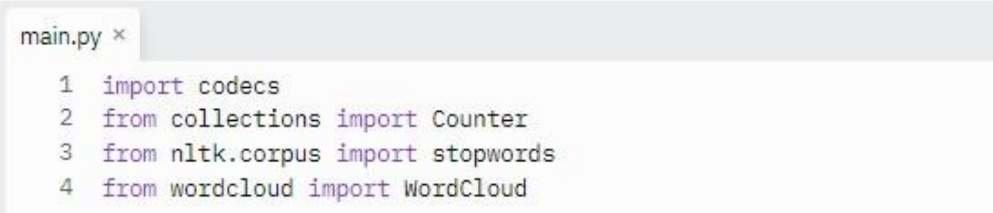
```python
import codecs
from collections import Counter
from nltk.corpus import stopwords

word_counter = Counter()
en_stopwords = stopwords.words("english")

def read_tweets_from_file(FILE_NAME):
    tweets = []
    with codecs.open(FILE_NAME, "r") as f:
        for line in f:
            parts = line.split("||")
            tweets.append(parts[1])
    return tweets

def cleaned_tweets(FILE_NAME):
    for line in read_tweets_from_file(FILE_NAME):
        line = line.lower()
        words = line.split()
        words_to_keep = []
        for word in words:
            if word not in en_stopwords:
                words_to_keep.append(word)
        yield ' '.join(words_to_keep)

for line in cleaned_tweets("historic_data.csv"):
    words = line.split()
    word_counter.update(words)

print(word_counter.most_common(20))
```

We can see in the output of the top 20 most used words all the stopwords have now been removed. However, this is not to say that all pre-processing has necessarily been completed since we still observe a high incidence of the rt symbol and the ampersand sign relative to other popular terms. Both of which could be candidates for removal in further pre-processing steps.



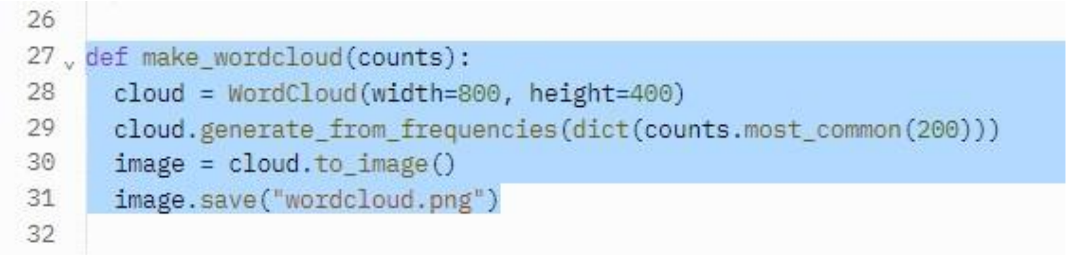
```
[('rt', 78), ('news', 19), ('&amp;', 10), ('became', 6), ('|', 6),
, 5), ('via', 5), ('launch', 5), ('time', 5), ('get', 4), ('today', 4),
 ('🔥', 4), ('la', 4), ('redmi', 4), ('5g', 4), ('us', 4), ('old', 3),
('see', 3), ('10', 3), ('@flokimooni:', 3)]
> 
```

One way this data can be explored visually is using a word cloud. A word cloud displays all words on a chart in relation to their frequency across a dataset. In Python we can create wordclouds using the wordclouds package. First, we import the wordCloud object from the wordcloud package.



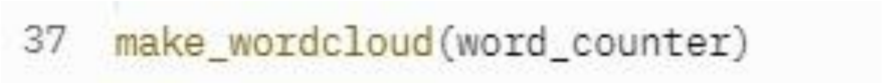
```python
import codecs
from collections import Counter
from nltk.corpus import stopwords
from wordcloud import WordCloud
```

Next we add a function to create a wordcloud and save it to a file called "wordcloud.png". This function accepts one parameter, the counter we created earlier that tracks the frequencies of words, to determine the relative sizes of words. In this case, we are drawing the wordcloud based on the top 200 most used words in the dataset.



```python

def make_wordcloud(counts):
  cloud = WordCloud(width=800, height=400)
  cloud.generate_from_frequencies(dict(counts.most_common(200)))
  image = cloud.to_image()
  image.save("wordcloud.png")

```

To draw the wordcloud we must call the make_wordcloud function, passing the word_counter variable as a parameter, at the end of the program when all words have been counted.



```python
make_wordcloud(word_counter)
```

Table A. Python code used to create a wordcloud

```
import codecs from collections
import Counter from nltk.corpus
import stopwords
from wordcloud import WordCloud

word_counter = Counter()
en_stopwords = stopwords.words("english")

def read_tweets_from_file(FILE_NAME):
tweets = []   with
codecs.open(FILE_NAME, "r") as f:     for
line in f:       parts = line.split("||")
tweets.append(parts[1])   return tweets

def cleaned_tweets(FILE_NAME):
  for line in read_tweets_from_file(FILE_NAME):
    line = line.lower()     words =
line.split()     words_to_keep = []
for word in words:       if word not
in en_stopwords:
words_to_keep.append(word)
yield ' '.join(words_to_keep)

def make_wordcloud(counts):   cloud =
WordCloud(width=800, height=400)
  cloud.generate_from_frequencies(dict(counts.most_common(200)))
image = cloud.to_image()
  image.save("wordcloud.png")

for line in cleaned_tweets("historic_data.csv"):
  words = line.split()
  word_counter.update(words)

make_wordcloud(word_counter)
```

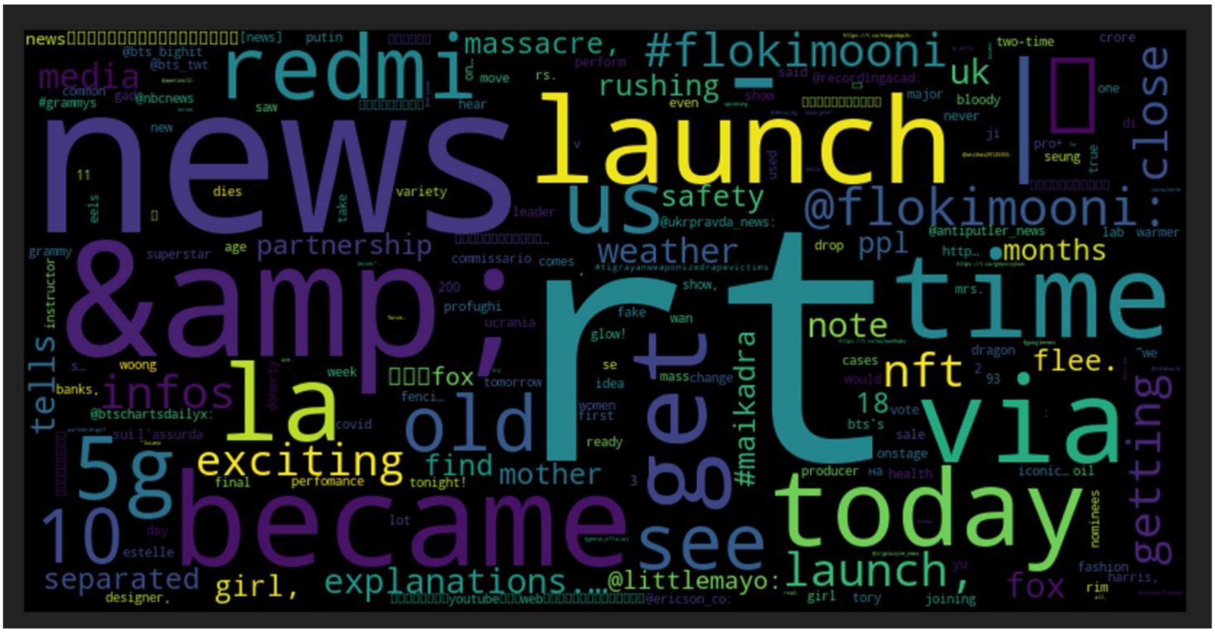


Figure 1 The resulting wordcloud downloaded as a PNG image

Exercise 3: (10 mins)

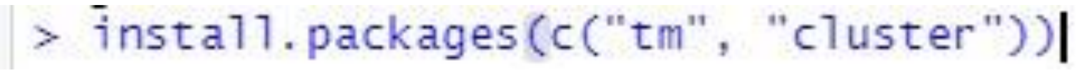
- Using the example code as a guide, determine the top 25 most common words present in your dataset.
- Explore your dataset using a wordcloud drawn for the top 200 most used words.

5. Exploring important dimensions and themes of discussion

In the LW10 lecture we saw how clustering can be applied to group a set of observations using a suitable unsupervised learning algorithm. In this case we used the k-means approach to assign individual text documents (or indeed tweets) to one of N groups according to their closeness to the centre of each group.

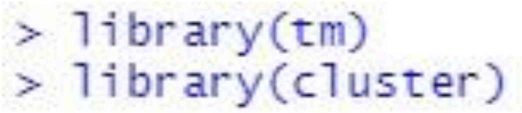
Recall that, as we are working with textual data, the documents were first recoded into a quantitative representation using the TF-IDF scores. With information about average word usage in each group (the groups centre) and word usage across groups, we can determine the usefulness of given word dimension. This approach can also be used to approximate “themes” of discussion taking place.

To be able to conduct this analysis in R we first need to install and load two important text-mining packages.



```
> install.packages(c("tm", "cluster"))
```

The following code will load the two libraries into the working environment.



```
> library(tm)
> library(cluster)
```

As a case study I am going to use the previously collected tweets surrounding the release of the Gravity movie. They are contained in a text file called “gravity-filmtweets.txt” that is placed in the working directory of my R installation.

In the following code I read in the tweets into a new document corpus and perform some initial pre-processing. If the data has already been processed when it was collected (i.e. from Python) these additional steps are not strictly necessary.

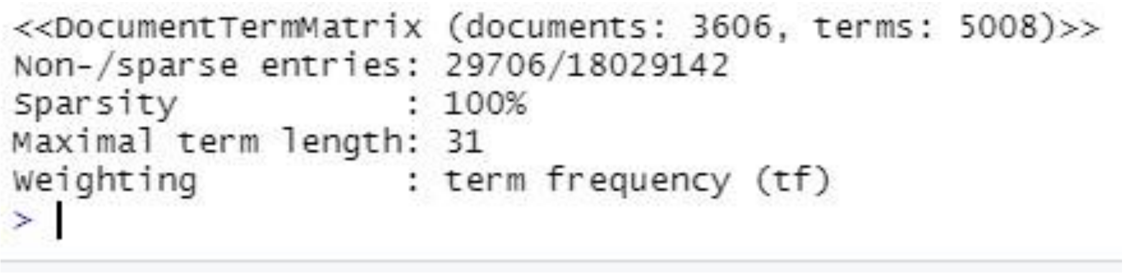
Table 2 Example code use to load in the gravity film tweets into R

```
text <- readLines("gravity-film-tweets.txt", encoding="UTF-8")
doc.vec <- VectorSource(text) doc.corpus <-
Corpus(doc.vec)
doc.corpus <- tm_map(doc.corpus, tolower)
doc.corpus <- tm_map(doc.corpus, removeWords,
stopwords('english')) doc.corpus <- tm_map(doc.corpus,
removePunctuation) doc.corpus <- tm_map(doc.corpus,
removeNumbers) doc.corpus <- tm_map(doc.corpus, stripWhitespace)
```

In the next step I convert the corpus (collection of text documents) into a document term matrix. This is essentially the bag of words model we saw in LW9.

```
dtm <- DocumentTermMatrix(doc.corpus)
dtm
```

From the output below we see we have a total of 3606 tweets that use 5008 unique words (our dimensions). The longest word used is 31 characters. The high level of sparsity implies that individual tweets use only a tiny proportion of available words (as expected)

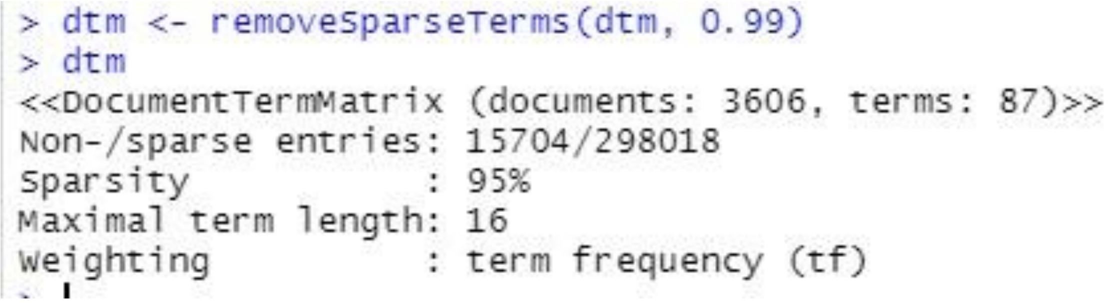


```
<<DocumentTermMatrix (documents: 3606, terms: 5008)>>
Non-/sparse entries: 29706/18029142
Sparsity           : 100%
Maximal term length: 31
Weighting          : term frequency (tf)
> |
```

Since high levels are sparsity can weaken statistical relationships, we next remove terms that don’t appear in at least 1% of our dataset.

```
dtm <- removeSparseTerms(dtm, 0.99) dtm
```

The output below shows that following a removal of sparse terms, the number of unique dimensions has dropped dramatically to just 87. Thus 99% of all tweets contain one or more of these dimensions.

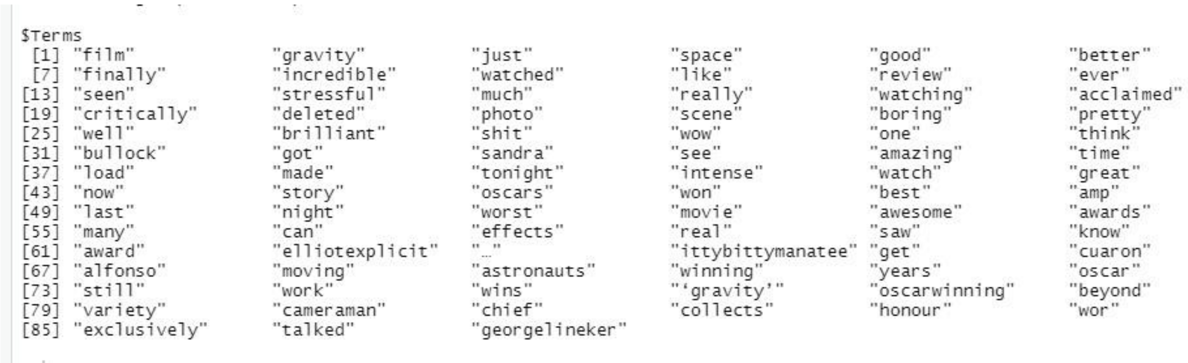


```
> dtm <- removeSparseTerms(dtm, 0.99)
> dtm
<<DocumentTermMatrix (documents: 3606, terms: 87)>>
Non-/sparse entries: 15704/298018
Sparsity           : 95%
Maximal term length: 16
Weighting          : term frequency (tf)
```

We can access these remaining 87 words using the dimnames property of the document term matrix.

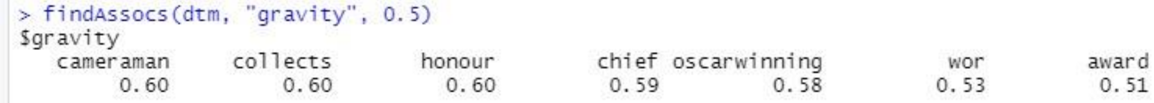
```
dtm$dimnames
```

And obtain the following list of terms



```
$Terms
 [1] "film"          "gravity"        "just"           "space"              "good"         "better"
 [7] "finally"       "incredible"     "watched"        "like"               "review"       "ever"
[13] "seen"          "stressful"      "much"           "really"             "watching"     "acclaimed"
[19] "critically"    "deleted"        "photo"          "scene"              "boring"       "pretty"
[25] "well"          "brilliant"      "shit"           "wow"                "one"          "think"
[31] "bullock"       "got"            "sandra"         "see"                "amazing"      "time"
[37] "load"          "made"           "tonight"        "intense"            "watch"        "great"
[43] "now"           "story"          "oscars"         "won"                "best"         "amp"
[49] "last"          "night"          "worst"          "movie"              "awesome"      "awards"
[55] "many"          "can"            "effects"        "real"               "saw"          "know"
[61] "award"         "elliotexplicit" "…"              "ittybittymanatee"   "get"          "cuaron"
[67] "alfonso"       "moving"         "astronauts"     "winning"            "years"        "oscar"
[73] "still"         "work"           "wins"           "‘gravity’"          "oscarwinning" "beyond"
[79] "variety"       "cameraman"      "chief"          "collects"           "honour"       "wor"
[85] "exclusively"   "talked"         "georgelineker"
```

Clearly the films name is an import term, but what other terms are most often used in the same tweet. For this analysis we can use a function called **findAssocs**(), which identifies all terms that have a higher than specified correlation with the given term.



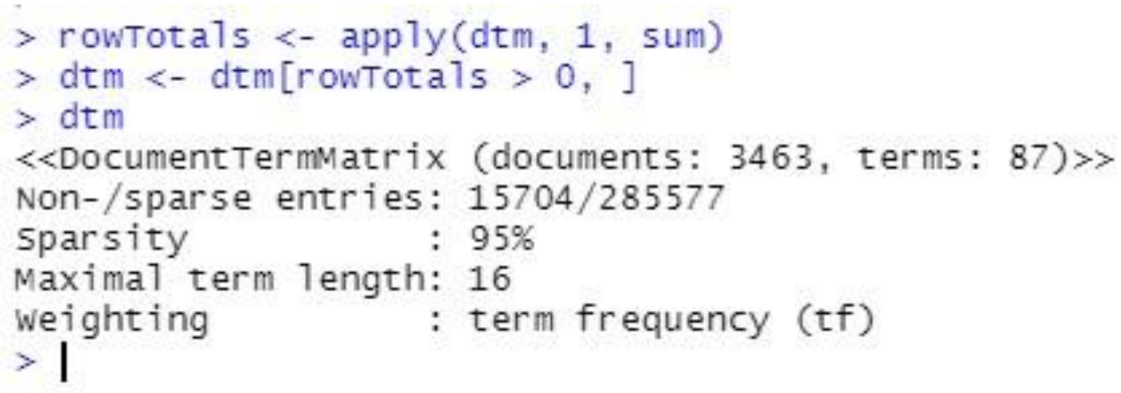
```
> findAssocs(dtm, "gravity", 0.5)
$gravity
   cameraman     collects       honour        chief oscarwinning          wor        award
        0.60         0.60         0.60         0.59         0.58         0.53         0.51
```

Previously, when we removed sparse terms, we deleted all terms that didn’t appear in at least 1% of our collected tweets. As a result, we may now have tweets in our collection that contain a row full of zeros (since all their words were deleted). We will now need to remove these documents since they don’t use any of the remaining terms. A simple sum of each row to check that it adds to a positive number will suffice.

```
rowTotals <- apply(dtm, 1, sum) dtm <-
dtm[rowTotals > 0, ]
```

```
dtm
```

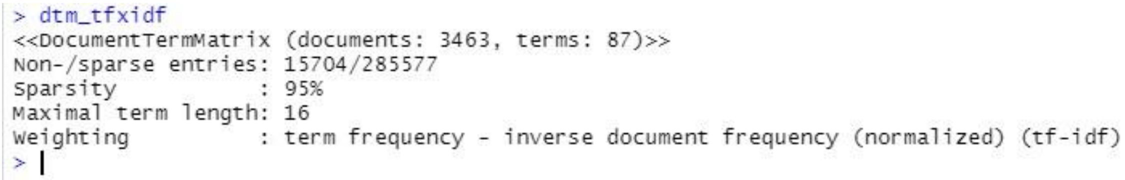
From the output below we found approximately (3606-3463 = 143) of such documents.



```
> rowTotals <- apply(dtm, 1, sum)
> dtm <- dtm[rowTotals > 0, ]
> dtm
<<DocumentTermMatrix (documents: 3463, terms: 87)>>
Non-/sparse entries: 15704/285577
Sparsity           : 95%
Maximal term length: 16
Weighting          : term frequency (tf)
> |
```

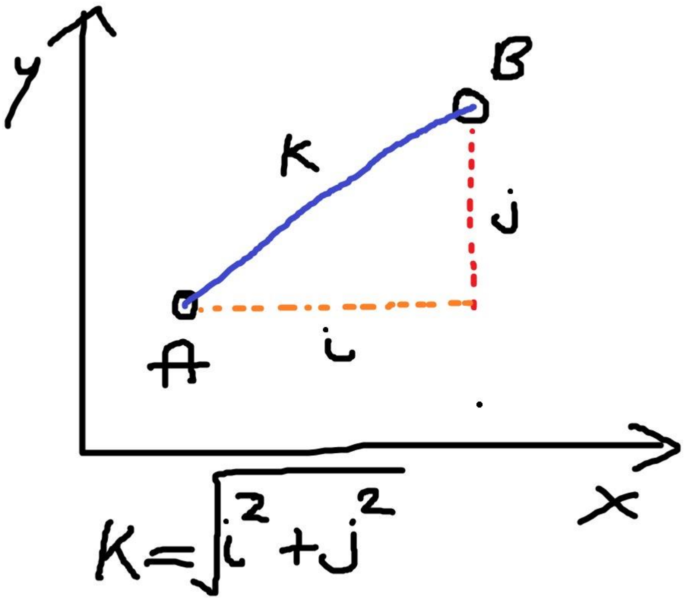
Next we create the TF-IDF matrix to replace the term frequencies and calculate the normalised Euclidean distance matrix. The distance matrix measures how far each document is from all other documents as the “crow flies”. This measure is used by the k-means algorithm to determine whether any two documents should be placed in the same cluster.

```
dtm_tfxidf <- weightTfIdf(dtm) m
<- as.matrix(dtm_tfxidf)
rownames(m) <- 1:nrow(m)
norm_eucl <- function(m) m/apply(m, 1, function(x) sum(x^2)^.5) m_norm
<- norm_eucl(m)
```



```
> dtm_tfxidf
<<DocumentTermMatrix (documents: 3463, terms: 87)>>
Non-/sparse entries: 15704/285577
Sparsity           : 95%
Maximal term length: 16
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
> |
```

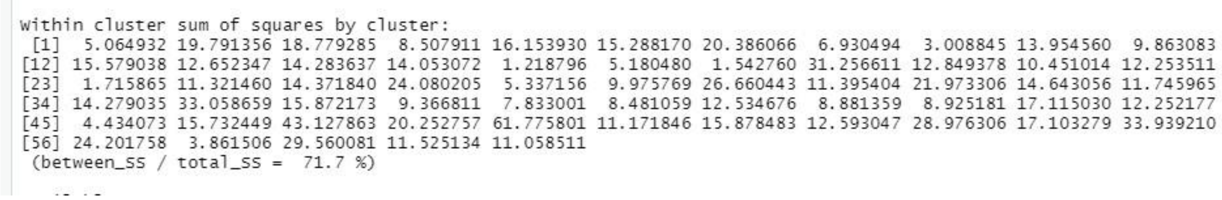
The Euclidean distance comes from Pythagoras’s famous theorem. In the twodimensional case, the Euclidean distance between points A and B is given by the vector k. It is found using the square root of the sum of the squares of both dimensions i and j (i.e. A^2+B^2 = C^2)



The k-means method can now be applied. As a starting point we choose the number of clusters to be the square root of the number of text documents in our dataset. This can be fine tuned later using an elbow approximation or other suitable method.

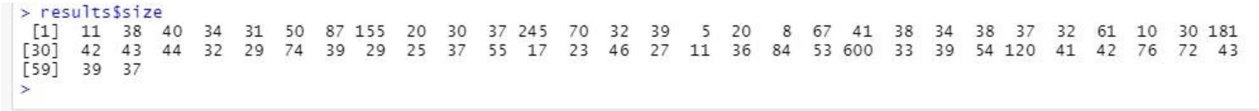
```
results <- kmeans(m_norm, as.integer(floor(length(text)^0.5)))
```

The results variable can now be inspected to check the results following model fitting. In this case we observe a ratio of the BSS to the TSS of 71.8% which is a reasonable initial fit.



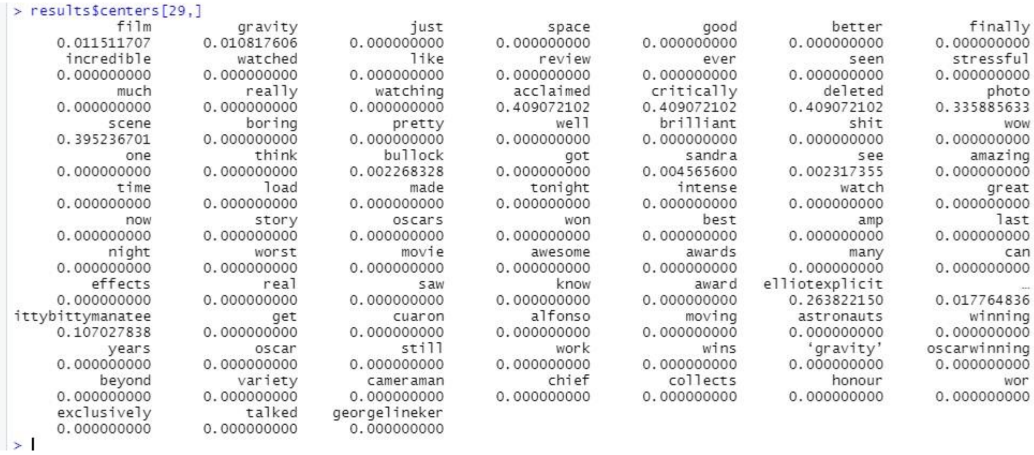
```
within cluster sum of squares by cluster:
 [1]  5.064932 19.791356 18.779285  8.507911 16.153930 15.288170 20.386066  6.930494  3.008845 13.954560  9.863083
[12] 15.579038 12.652347 14.283637 14.053072  1.218796  5.180480  1.542760 31.256611 12.849378 10.451014 12.253511
[23]  1.715865 11.321460 14.371840 24.080205  5.337156  9.975769 26.660443 11.395404 21.973306 14.643056 11.745965
[34] 14.279035 33.058659 15.872173  9.366811  7.833001  8.481059 12.534676  8.881359  8.925181 17.115030 12.252177
[45]  4.434073 15.732449 43.127863 20.252757 61.775801 11.171846 15.878483 12.593047 28.976306 17.103279 33.939210
[56] 24.201758  3.861506 29.560081 11.525134 11.058511
 (between_SS / total_SS =  71.7 %)
```

By inspection of the size property, we can see that a large proportion of tweets were placed in clusters 49 (600 tweets), cluster 12 (245 tweets) and cluster 29 (181 tweets). Cluster 16 has the lowest number of tweets assigned (5 tweets)



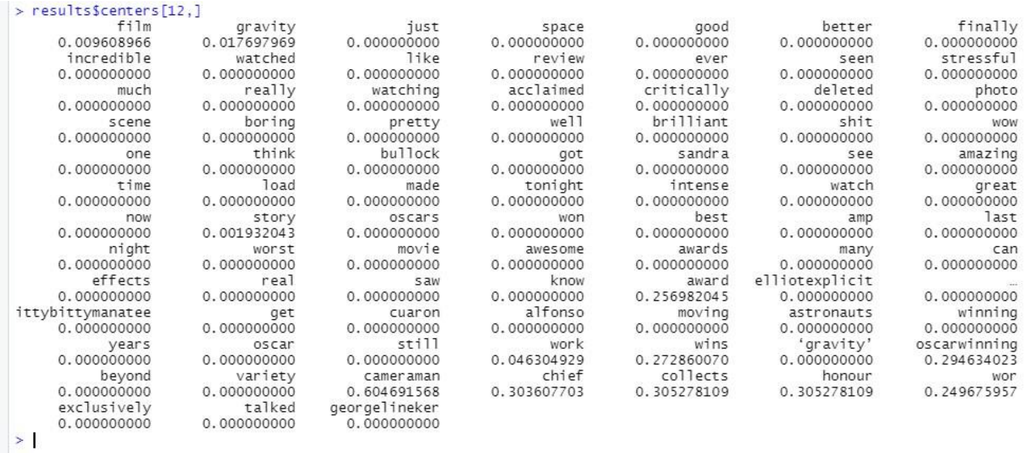
```
> results$size
 [1]  11  38  40  34  31  50  87 155  20  30  37 245  70  32  39   5  20   8  67  41  38  34  38  37  32  61  10  30 181
[30]  42  43  44  32  29  74  39  29  25  37  55  17  23  46  27  11  36  84  53 600  33  39  54 120  41  42  76  72  43
[59]  39  37
>
```

Inspecting the centres of the largest clusters we observe a potential theme of discussion around “critically”, “acclaimed”, “deleted”, “photo” and “scene” in cluster 29.



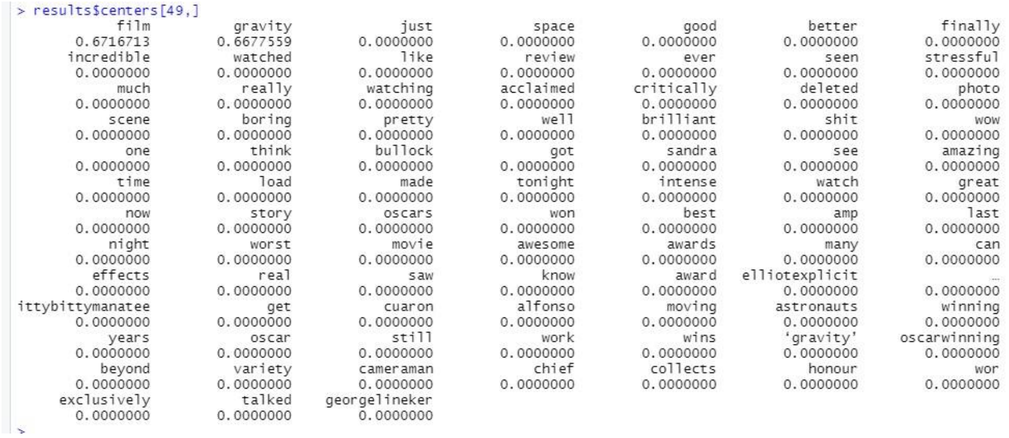
```
> results$centers[29,]
            film          gravity             just            space             good           better          finally 
     0.011511707      0.010817606      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
      incredible          watched             like           review             ever             seen        stressful 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
            much           really         watching        acclaimed       critically          deleted            photo 
     0.000000000      0.000000000      0.000000000      0.409072102      0.409072102      0.409072102      0.335885633 
           scene           boring           pretty             well        brilliant             shit              wow 
     0.395236701      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
             one            think          bullock              got           sandra              see          amazing 
     0.000000000      0.000000000      0.002268328      0.000000000      0.004565600      0.002317355      0.000000000 
            time             load             made          tonight          intense            watch            great 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
             now            story           oscars              won             best              amp             last 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
           night            worst            movie          awesome           awards             many              can 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
         effects             real              saw             know            award   elliotexplicit                … 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.263822150      0.017764836 
ittybittymanatee              get           cuaron          alfonso           moving       astronauts          winning 
     0.107027838      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
           years            oscar            still             work             wins        ‘gravity’     oscarwinning 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
          beyond          variety        cameraman            chief         collects           honour              wor 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
     exclusively           talked    georgelineker 
     0.000000000      0.000000000      0.000000000 
> |
```

Cluster 12 has high relative usage of the terms “cameraman”, “oscarwinning”, “honour”, “won”, “wins” and “award”.



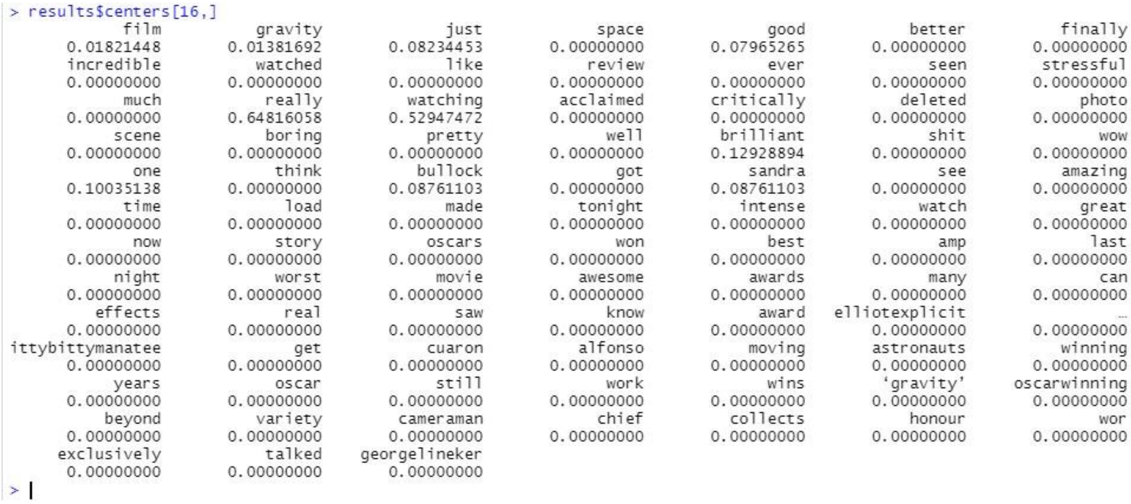
```
> results$centers[12,]
            film          gravity             just            space             good           better          finally 
     0.009608966      0.017697969      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
      incredible          watched             like           review             ever             seen        stressful 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
            much           really         watching        acclaimed       critically          deleted            photo 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
           scene           boring           pretty             well        brilliant             shit              wow 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
             one            think          bullock              got           sandra              see          amazing 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
            time             load             made          tonight          intense            watch            great 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
             now            story           oscars              won             best              amp             last 
     0.000000000      0.001932043      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
           night            worst            movie          awesome           awards             many              can 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
         effects             real              saw             know            award   elliotexplicit                … 
     0.000000000      0.000000000      0.000000000      0.000000000      0.256982045      0.000000000      0.000000000 
ittybittymanatee              get           cuaron          alfonso           moving       astronauts          winning 
     0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000      0.000000000 
           years            oscar            still             work             wins        ‘gravity’     oscarwinning 
     0.000000000      0.000000000      0.000000000      0.046304929      0.272860070      0.000000000      0.294634023 
          beyond          variety        cameraman            chief         collects           honour              wor 
     0.000000000      0.000000000      0.604691568      0.303607703      0.305278109      0.305278109      0.249675957 
     exclusively           talked    georgelineker 
     0.000000000      0.000000000      0.000000000 
> |
```

Cluster 49 has very high average usage of the words “film” and “gravity” but almost no other terms. This might reflect tweets just announcing the film rather than any discussion of it.



```
> results$centers[49,]
            film          gravity             just            space             good           better          finally 
       0.6716713        0.6677559        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
      incredible          watched             like           review             ever             seen        stressful 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
            much           really         watching        acclaimed       critically          deleted            photo 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
           scene           boring           pretty             well        brilliant             shit              wow 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
             one            think          bullock              got           sandra              see          amazing 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
            time             load             made          tonight          intense            watch            great 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
             now            story           oscars              won             best              amp             last 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
           night            worst            movie          awesome           awards             many              can 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
         effects             real              saw             know            award   elliotexplicit                … 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
ittybittymanatee              get           cuaron          alfonso           moving       astronauts          winning 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
           years            oscar            still             work             wins        ‘gravity’     oscarwinning 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
          beyond          variety        cameraman            chief         collects           honour              wor 
       0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000        0.0000000 
     exclusively           talked    georgelineker 
       0.0000000        0.0000000        0.0000000 
>
```

Looking at cluster 16, which had the lowest number of tweets assigned, we see low levels of average term usage except for the terms “really” and “watching”.



```
> results$centers[16,]
            film          gravity             just            space             good           better          finally
      0.01821448       0.01381692       0.08234453       0.00000000       0.07965265       0.00000000       0.00000000
      incredible          watched             like           review             ever             seen        stressful
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
            much           really         watching        acclaimed       critically          deleted            photo
      0.00000000       0.64816058       0.52947472       0.00000000       0.00000000       0.00000000       0.00000000
           scene           boring           pretty             well        brilliant             shit              wow
      0.00000000       0.00000000       0.00000000       0.00000000       0.12928894       0.00000000       0.00000000
             one            think          bullock              got           sandra              see          amazing
      0.10035138       0.00000000       0.08761103       0.00000000       0.08761103       0.00000000       0.00000000
            time             load             made          tonight          intense            watch            great
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
             now            story           oscars              won             best              amp             last
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
           night            worst            movie          awesome           awards             many              can
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
         effects             real              saw             know            award   elliotexplicit                …
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
ittybittymanatee              get           cuaron          alfonso           moving       astronauts          winning
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
           years            oscar            still             work             wins        ‘gravity’     oscarwinning
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
          beyond          variety        cameraman            chief         collects           honour              wor
      0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000       0.00000000
     exclusively           talked    georgelineker
      0.00000000       0.00000000       0.00000000
> |
```

Table 3 Sample code for k-means in R

```
library(tm)
library(cluster)

text <- readLines("gravity-film-tweets.txt", encoding="UTF-8")
doc.vec <- VectorSource(text) doc.corpus <-
Corpus(doc.vec)
doc.corpus <- tm_map(doc.corpus, tolower)
doc.corpus <- tm_map(doc.corpus, removeWords,
stopwords('english')) doc.corpus <- tm_map(doc.corpus,
removePunctuation) doc.corpus <- tm_map(doc.corpus,
removeNumbers) doc.corpus <- tm_map(doc.corpus, stripWhitespace)

dtm <- DocumentTermMatrix(doc.corpus) dtm

dtm <- removeSparseTerms(dtm, 0.99) dtm

rowTotals <- apply(dtm, 1, sum) dtm
<- dtm[rowTotals > 0, ]
dtm

dtm_tfxidf <- weightTfIdf(dtm) m
<- as.matrix(dtm_tfxidf)
rownames(m) <- 1:nrow(m)
norm_eucl <- function(m) m/apply(m, 1, function(x) sum(x^2)^.5) m_norm
<- norm_eucl(m)

results <- kmeans(m_norm, as.integer(floor(length(text)^0.5)))
```

Exercise 4: (15 mins)

- Using the sample code as a guide, apply the k-means approach to the dataset collected using the streaming API as part of exercise I. ○ Identify the initial number of unique dimensions (words) that are part of your dataset
  - After removing 1% sparse terms, how many word dimensions remain?
  - Take a screenshot of the remaining terms extracted from the document term matrix. ○ Comment on the ratio of the BSS/TSS ratio. Experiment with different numbers of clusters (+/- 2), how does the BSS/TSS ratio change?
  - Inspect the clusters with the highest number of tweets assigned and examine their word usage.

Appendix:

Table 4 Streaming Code

```
import tweepy
import sys
import os import
codecs
import re

BEARER_TOKEN="INSERT BEARER TOKEN HERE"

DATA_FILE = "streaming_data.csv" if
not os.path.exists(DATA_FILE):
  os.mknod(DATA_FILE)

class BasicTwitterListener(tweepy.StreamingClient):
def __init__(self, n_tweets):
    super().__init__(BEARER_TOKEN)
    self.n_tweets = n_tweets

  def on_tweet(self, tweet):     tweet.text =
re.sub("[\r\n]+"," ", tweet.text)
    print(tweet.created_at, tweet.text, tweet.author_id)

    with codecs.open(DATA_FILE, "a") as f:
      f.write("%s||%s||%s\n" % (tweet.created_at, tweet.text, tweet.author_id))

    self.n_tweets   -=   1
if self.n_tweets == 0:
      self.disconnect()

listener = BasicTwitterListener(10)
listener.add_rules(tweepy.StreamRule("news"))
listener.filter(expansions=["author_id"], tweet_fields=["created_at"])
```

Table 5 Historic Code

```
import tweepy
import sys
import os import
codecs
import re

BEARER_TOKEN="INSERT BEARER TOKEN HERE"

DATA_FILE = "historic_data.csv" if
not os.path.exists(DATA_FILE):
  os.mknod(DATA_FILE)

client = tweepy.Client(BEARER_TOKEN)

response = client.search_recent_tweets(query="news OR weather",
                                    expansions=["author_id"],
                                    tweet_fields=['created_at'], max_results=100)

with codecs.open(DATA_FILE, "a") as f:
for tweet in response.data:
    tweet.text = re.sub("[\r\n]+"," ", tweet.text)
print(tweet.created_at, tweet.text, tweet.author_id)
    f.write("%s||%s||%s\n" % (tweet.created_at, tweet.text, tweet.author_id))
```

END