University of Westminster

Web and social media analysis

7BUIS009W

CourseWork: Social Media Assignment

Prepared for

Dr Philip Worrall

Name: Mohamed Banihani

Student ID: W1743591

Word count: 1985

26 April 2020

TABLE OF CONTENTS

Introduction:

Nowadays, social media platforms have become tremendously shaping the world around us, and because of its unconventional benefits has been adopted by many organisations. More and more companies are utilising social media data to have a better understanding of users. Based on statista website, there are 3.08 Billion users who are using social media platforms(statista, 2020). As a result, a massive amount of data is being generated each day. This report aims to use Twitter API to collect data related to Bill Gates by identifying 5 keywords, preparing the data by cleaning it and finally analysing the data to gain more in-depth insights.

1) Selected famous person

Bill Gates is one of the most well-known businessmen in the world. In 1975, he co-founded Microsoft. Microsoft is considered the 3rd most happiest account on Twitter (TwitterBlog, 2019). Bill Gates is the world's second-richest man (Coudriet, 2019). Apart from being the co-founder of Microsoft, Bill Gates is also co-chair of the Bill & Melinda Foundation, which is the world's wealthiest charity (Gates foundation, 2019). In most of his recent public interviews, he had warned that global catastrophe wasn't a war but a highly infectious virus which now is strongly perceived as accurate (FT, 2020). As a result, many tweets published recently highlights how Bill Gates predicted the ongoing scenario. For this reason, I picked Bill Gates to analyse the overall tweets and discover how people are precisely reacting on Twitter. Key word select is “bill gates” “melinda gates” “microsoft” “covid19” “pandemic.”

2) API's

API (Application Programming Interface) is a program that allows applications to communicate and exchange data with each other. It consists of a set of rules which enables an application to request and receive data (Davis, 2019). Most websites offer an open API that enables the user to access and obtain data. In this report, two data collection APIs, the Standard API and the Premium API are discussed.

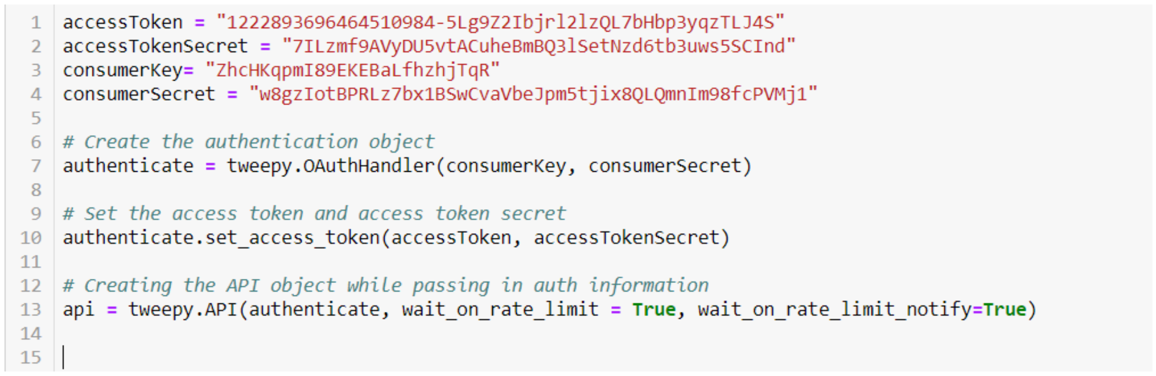
The standard API contains a Streaming API and REST API, which offers a collection of tweets related to the specific query (Developer.twitter, 2020). The main difference is that the Streaming API gives access to a sample of all tweets published on Twitter. It allows collection of live tweets data whereas REST API is more suitable for singular searches, such as searching historic tweets up to seven days, reading user profile information or posting Tweets.

Premium search API: Premium search API, also known as pay as you go mainly includes reliable and affordable versions of enterprise APIs, helping businesses to use the services allowed efficiently. The premium search contains both data and counts endpoints, providing functionality outside what's available in standard search/tweets endpoint, including more tweets per request, improved rate limits, more sophisticated queries, metadata enhancements, such as extended URLs and accurate geo-information profiling(Developer.twitter, 2020). Premium search API endpoints are of two types: 30-day endpoint and full-archive endpoint. These search API endpoints share a similar design and documentation. The only dissimilarity in the two APIs is the search time frame, either the previous 30 days or tweets from as early as 2006. Time frames can be defined with minute granularity.

Although the premium APIs share standard method/parameter details with the enterprise APIs, they have different authentication method. Premium APIs benefits through offering low-latency, full-fidelity, query-based access to the tweet archive. Tweets are shown in reverse chronological order, starting with the most recent Tweet. Premium search API pricing starts at $149/month.

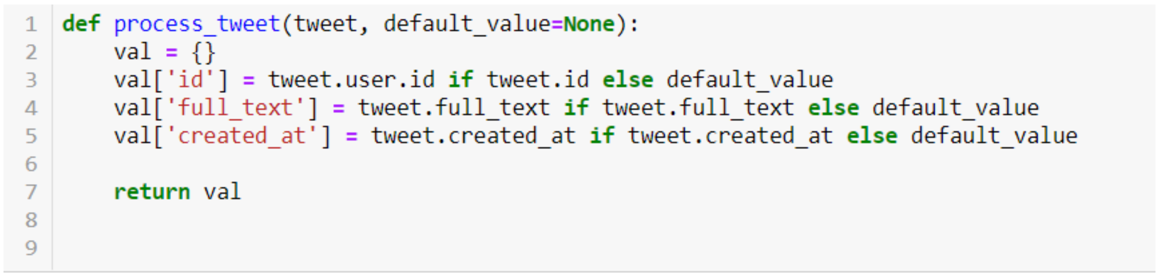
2) Data collection

Data collection is considered the first step in analysing tweeter. This process requires having Twitter API keys and installing tweepy library.



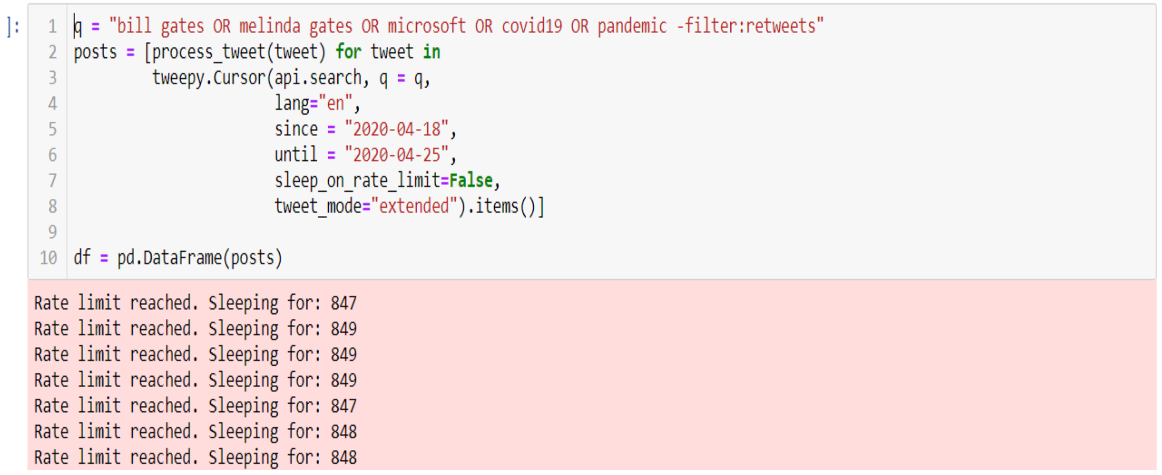
```
accessToken = "1222893696464510984-5Lg9Z2Ibjrl2lzQL7bHbp3yqzTLJ4S"
accessTokenSecret = "7ILzmf9AVyDU5vtACuheBmBQ3lSetNzd6tb3uws5SCInd"
consumerKey= "ZhcHKqpmI89EKEBaLfhzhjTqR"
consumerSecret = "w8gzIotBPRLz7bx1BSwCvaVbeJpm5tjix8QLQmnIm98fcPVMj1"

# Create the authentication object
authenticate = tweepy.OAuthHandler(consumerKey, consumerSecret)

# Set the access token and access token secret
authenticate.set_access_token(accessToken, accessTokenSecret)

# Creating the API object while passing in auth information
api = tweepy.API(authenticate, wait_on_rate_limit = True, wait_on_rate_limit_notify=True)

```

the above code identifies the keys to access twitter API and create an API variable that contains the authentication. The methods wait_on_rate_limit and wait_on_rate_limit_notify help in fetching a vast amount of data without running the code every 15 mintues.



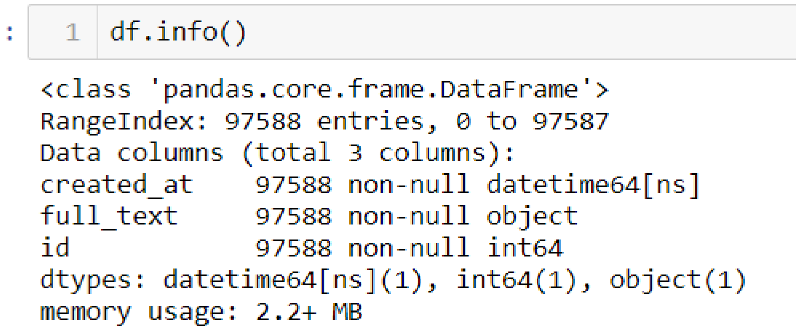
```python
def process_tweet(tweet, default_value=None):
    val = {}
    val['id'] = tweet.user.id if tweet.id else default_value
    val['full_text'] = tweet.full_text if tweet.full_text else default_value
    val['created_at'] = tweet.created_at if tweet.created_at else default_value

    return val


```

the above code creates a function which stores only the user id, the full tweet text and the date as these three columns are essential for the analysis.



```python
]:  q = "bill gates OR melinda gates OR microsoft OR covid19 OR pandemic -filter:retweets"
    posts = [process_tweet(tweet) for tweet in
            tweepy.Cursor(api.search, q = q,
                           lang="en",
                           since = "2020-04-18",
                           until = "2020-04-25",
                           sleep_on_rate_limit=False,
                           tweet_mode="extended").items()]

    df = pd.DataFrame(posts)
```

```
Rate limit reached. Sleeping for: 847
Rate limit reached. Sleeping for: 849
Rate limit reached. Sleeping for: 849
Rate limit reached. Sleeping for: 849
Rate limit reached. Sleeping for: 847
Rate limit reached. Sleeping for: 848
Rate limit reached. Sleeping for: 848
```



```python
:  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97588 entries, 0 to 97587
Data columns (total 3 columns):
created_at    97588 non-null datetime64[ns]
full_text     97588 non-null object
id            97588 non-null int64
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 2.2+ MB
```

After having the keys and running the function process_tweet, next is to query the Twitter API and request the data using tweepy.cursor. As shown above, six parameters were added to be able to fetch the valid data throughout an entire week.

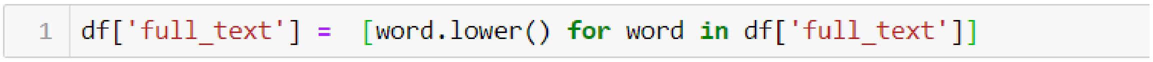
The result from the query is a panda's data frame (*df)* that contains three columns and 97,587 rows. It is worth mentioning that the running time for the above query was 7 hours.

4) Pre-Processing

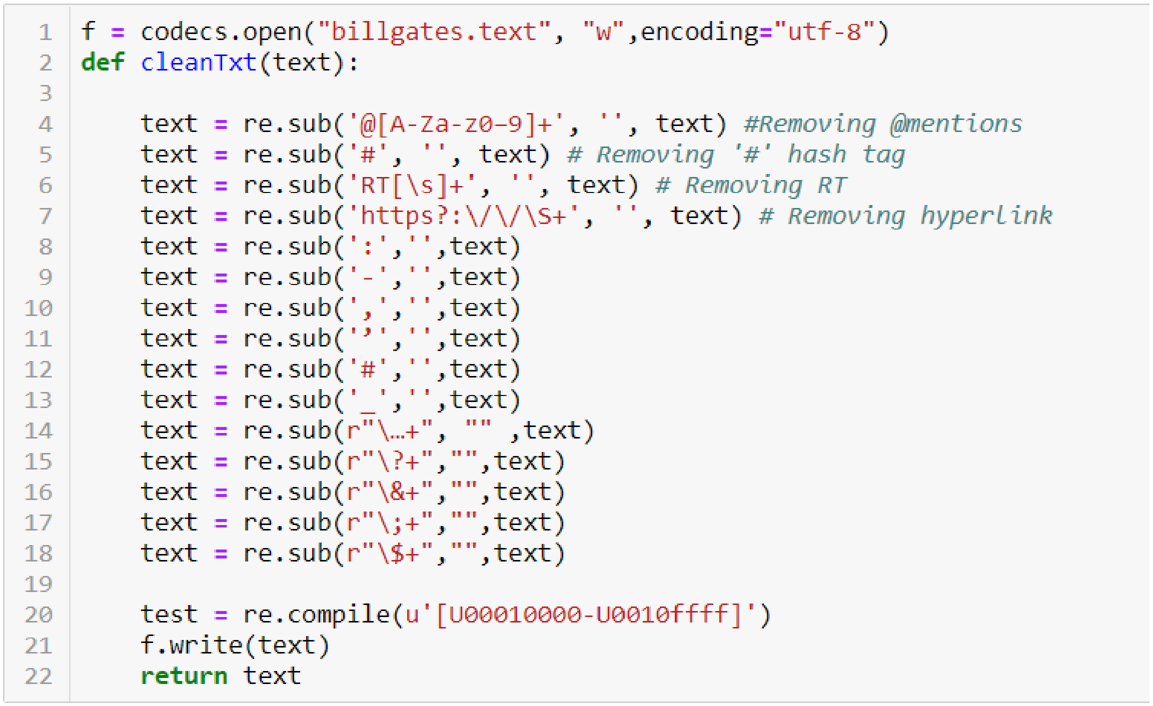
Pre-processing is one of the critical steps for any type of analysis. It helps in cleaning and removing all unnecessary data to enhance the accuracy of the results.

The three main pre-processing steps taken to achieve pre-processing aims are as follows:

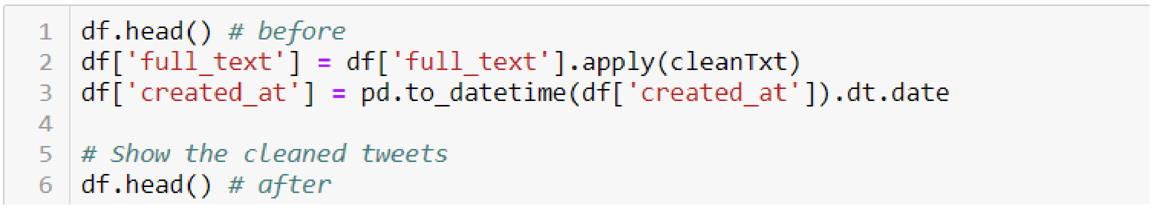
1. Convert all text into lower cases by using .lower function
2. Replace all unwanted characters from the tweets by using *re* library
3. Remove the hours, mintues and seconds from the date attribute using pandas to_datetime.



```
df['full_text'] =  [word.lower() for word in df['full_text']]
```

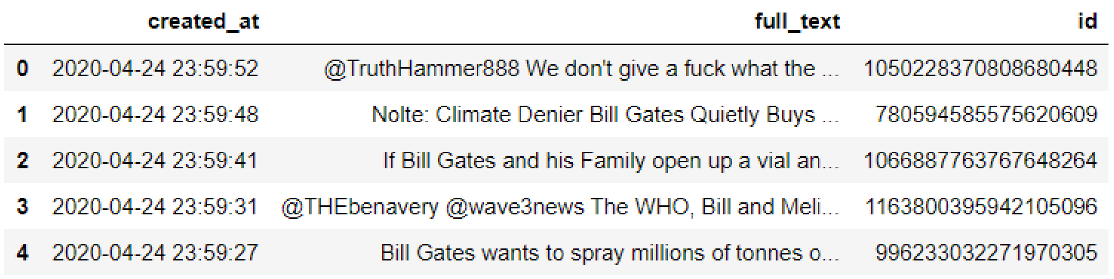


```
f = codecs.open("billgates.text", "w",encoding="utf-8")
def cleanTxt(text):

    text = re.sub('@[A-Za-z0-9]+', '', text) #Removing @mentions
    text = re.sub('#', '', text) # Removing '#' hash tag
    text = re.sub('RT[\s]+', '', text) # Removing RT
    text = re.sub('https?:\/\/\S+', '', text) # Removing hyperlink
    text = re.sub(':','',text)
    text = re.sub('-','',text)
    text = re.sub(',','',text)
    text = re.sub('’','',text)
    text = re.sub('#','',text)
    text = re.sub('_','',text)
    text = re.sub(r"\…+", "" ,text)
    text = re.sub(r"\?+","",text)
    text = re.sub(r"\&+","",text)
    text = re.sub(r"\;+","",text)
    text = re.sub(r"\$+","",text)

    test = re.compile(u'[U00010000-U0010ffff]')
    f.write(text)
    return text
```



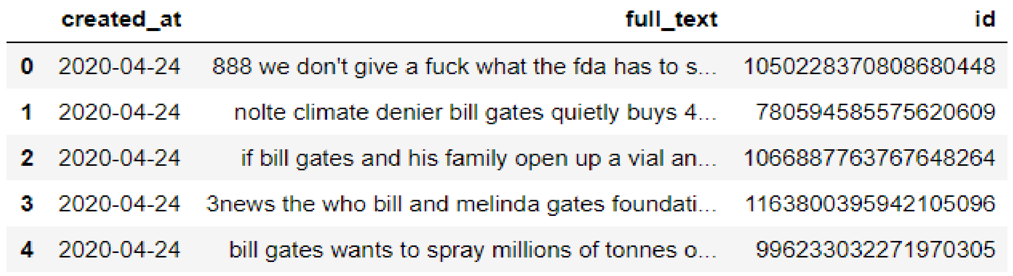
```
df.head() # before
df['full_text'] = df['full_text'].apply(cleanTxt)
df['created_at'] = pd.to_datetime(df['created_at']).dt.date

# Show the cleaned tweets
df.head() # after
```

Before



	created_at	full_text	id
0	2020-04-24 23:59:52	@TruthHammer888 We don't give a fuck what the ...	1050228370808680448
1	2020-04-24 23:59:48	Nolte: Climate Denier Bill Gates Quietly Buys ...	780594585575620609
2	2020-04-24 23:59:41	If Bill Gates and his Family open up a vial an...	1066887763767648264
3	2020-04-24 23:59:31	@THEbenavery @wave3news The WHO, Bill and Meli...	1163800395942105096
4	2020-04-24 23:59:27	Bill Gates wants to spray millions of tonnes o...	996233032271970305

After

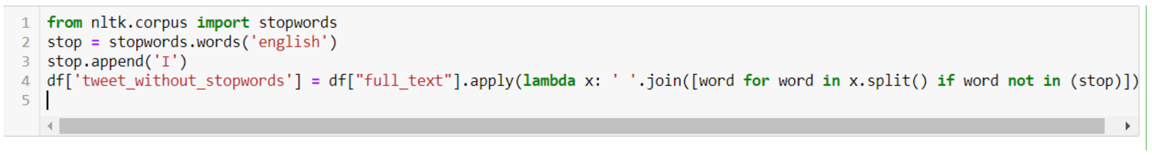


	created_at	full_text	id
0	2020-04-24	888 we don't give a fuck what the fda has to s...	1050228370808680448
1	2020-04-24	nolte climate denier bill gates quietly buys 4...	780594585575620609
2	2020-04-24	if bill gates and his family open up a vial an...	1066887763767648264
3	2020-04-24	3news the who bill and melinda gates foundati...	1163800395942105096
4	2020-04-24	bill gates wants to spray millions of tonnes o...	996233032271970305

As shown in the sample data above, full_text has been cleaned by removing unnecessary data, and created_at (date) converted to show only the date without timings.

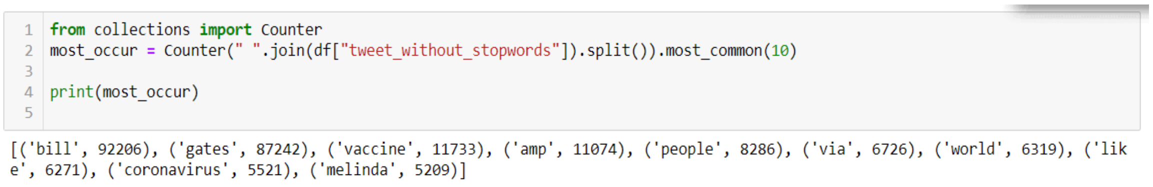
5) Top 10 word count

After having the dataset cleaned and ready, the next step is to count the top 10 occurring words. Firstly before running counting words query, removing stop words is deemed as an essential process towards counting words that do not pose negative consequences onto the final model.



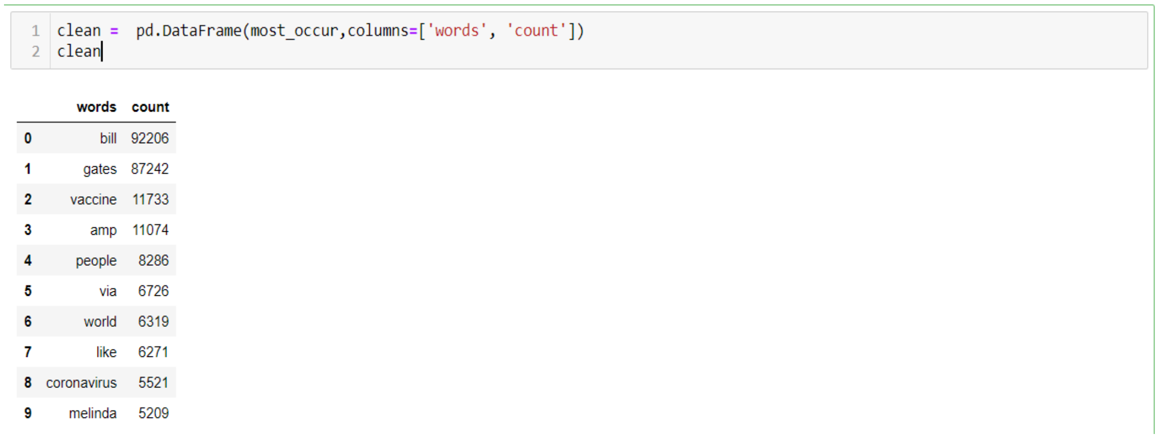
```
from nltk.corpus import stopwords
stop = stopwords.words('english')
stop.append('I')
df['tweet_without_stopwords'] = df["full_text"].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))

```

The above code results in creating a new column named *tweer_without_stopwords*, helping in counting the top 10 most tweeted words.



```
from collections import Counter
most_occur = Counter(" ".join(df["tweet_without_stopwords"]).split()).most_common(10)

print(most_occur)

```

```
[('bill', 92206), ('gates', 87242), ('vaccine', 11733), ('amp', 11074), ('people', 8286), ('via', 6726), ('world', 6319), ('lik
e', 6271), ('coronavirus', 5521), ('melinda', 5209)]
```

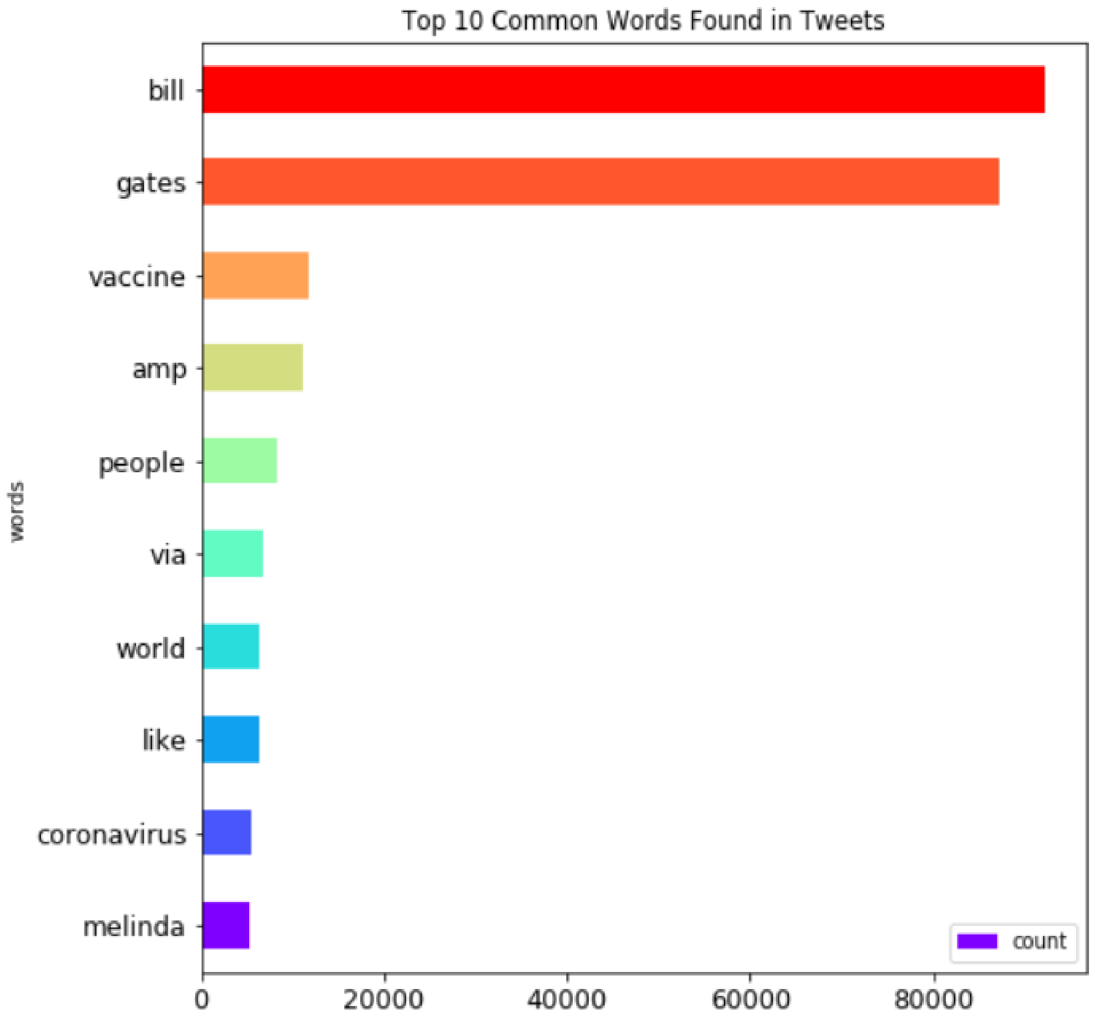


```
clean =  pd.DataFrame(most_occur,columns=['words', 'count'])
clean
```

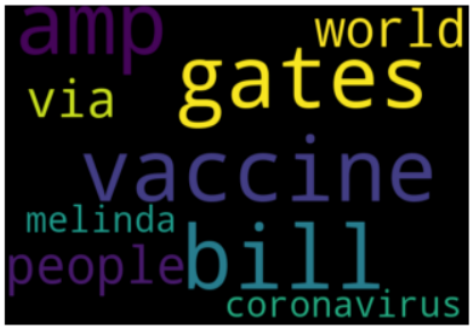
	words	count
0	bill	92206
1	gates	87242
2	vaccine	11733
3	amp	11074
4	people	8286
5	via	6726
6	world	6319
7	like	6271
8	coronavirus	5521
9	melinda	5209

As shown above, the counter method is used to count each word and save them in a variable called *most_occur.* Next, the variable is converted to pandas dataframe and finally visualised by using matplotlib and word cloud.

```
import matplotlib.cm as cm
fig, ax = plt.subplots(figsize=(8, 8))
colors = cm.rainbow(np.linspace(0, 1, 10))
# Plot horizontal bar graph
clean.sort_values(by='count').plot.barh(x='words',
                      y='count',
                      ax=ax,
                      color=colors)

ax.set_title("Top 10 Common Words Found in Tweets")

plt.show()
```



```
allWords = ' '.join([twts for twts in clean['words']])
wordCloud = WordCloud(width=600, height=400, random_state=21, max_font_size=110).generate(allWords)


plt.imshow(wordCloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



6.1) Number of tweets posted per day

This part of the report aims to answer how many tweets are posted per day. In pursuing to find an answer for this specific question, group by function is used that groups each day with its right number of tweets.

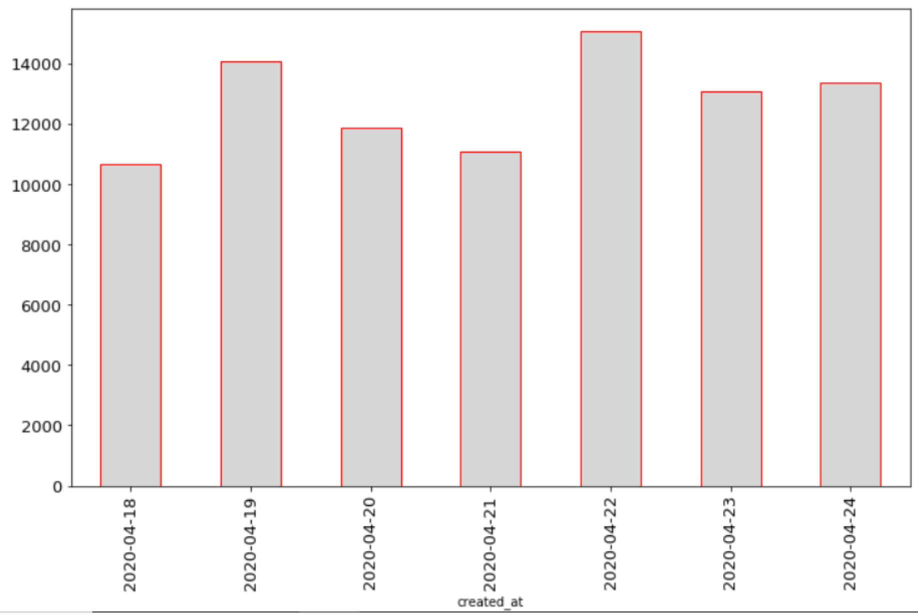
```
gp1 =  df.groupby('created_at')['full_text'].nunique()
gp1
```

```
created_at
2020-04-18    10645
2020-04-19    14073
2020-04-20    11847
2020-04-21    11084
2020-04-22    15065
2020-04-23    13083
2020-04-24    13374
Name: full_text, dtype: int64
```

As shown in the above, date 22-04-2020 has the highest number of tweets which fetched 15065 compared with 18-04-2020, which fetched 10645, the lowest amount. Lastly, a bar plot is used to visualise the findings.

```
fig, ax = plt.subplots(figsize=(12,7))

df.groupby('created_at')['full_text'].nunique().plot(kind = 'bar',color=(0.2, 0.2, 0.2, 0.2),  edgecolor='red')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b3ce3f6f60>
```



6.2) the number of unique user per day

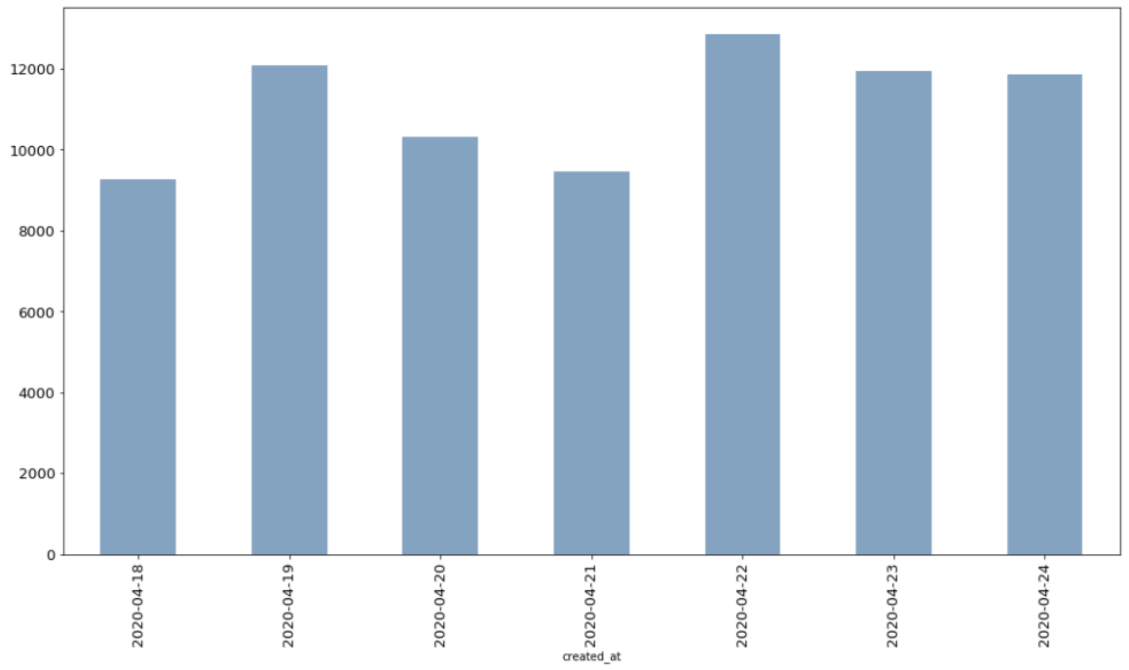
```
gp2 = df.groupby('created_at')['id'].nunique()
gp2
```

```
created_at
2020-04-18     9250
2020-04-19    12069
2020-04-20    10316
2020-04-21     9467
2020-04-22    12863
2020-04-23    11942
2020-04-24    11859
Name: id, dtype: int64
```

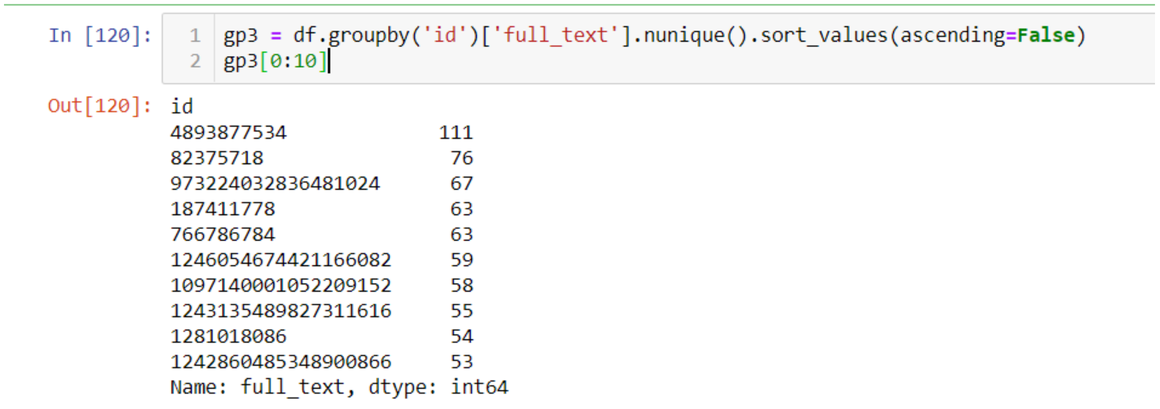
As shown in the above, date 22-04-2020 has the highest number of tweets which reached 12863 compared with 18-04-2020, which fetched 9250, the lowest amount. Lastly, a bar plot is used to visualise the findings.

```
fig, ax = plt.subplots(figsize=(17,9))

df.groupby('created_at')['id'].nunique().plot(kind = 'bar',color=(0.2, 0.4, 0.6, 0.6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b3ce0154a8>
```



6.3) top 10 active users



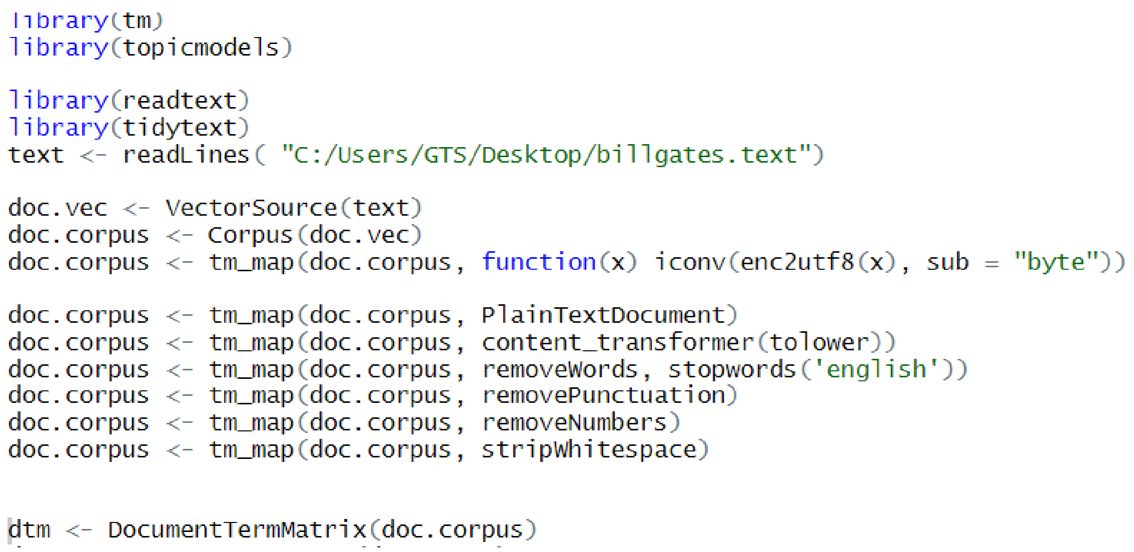
```
In [120]: gp3 = df.groupby('id')['full_text'].nunique().sort_values(ascending=False)
          gp3[0:10]

Out[120]: id
          4893877534             111
          82375718                76
          973224032836481024      67
          187411778               63
          766786784               63
          1246054674421166082     59
          1097140001052209152     58
          1243135489827311616     55
          1281018086              54
          1242860485348900866     53
          Name: full_text, dtype: int64
```

As shown above, id number “4893877534” is the most active user between the dataset which tweeted 111 tweets in 7 days.

7) Topic modelling:

Since the early 1990s, the growth rate of the Internet in general and associated technologies have proliferated phenomenally, which in effect has resulted in a plethora of online traffic and data (Cho, 2019). Topic modelling is a frequently used text-mining tool which scans through a set of documents which help in discovering hidden semantic structures within the extensive body of the text(). Topic modelling is also described as probabilistic topic modelling, which refers to statistical algorithms used for the discovery of latent semantic. LDA is one of the famous kind of topic modelling used for topic extraction (Kulshrestha, 2019). This is done by linking each word in a given to a specified number of topic models. LDA performs a parallel task where the aim is to assign each word with topics of varying interpretations accurately.

In this report, the LDA method is performed by using Rstudio as follows:



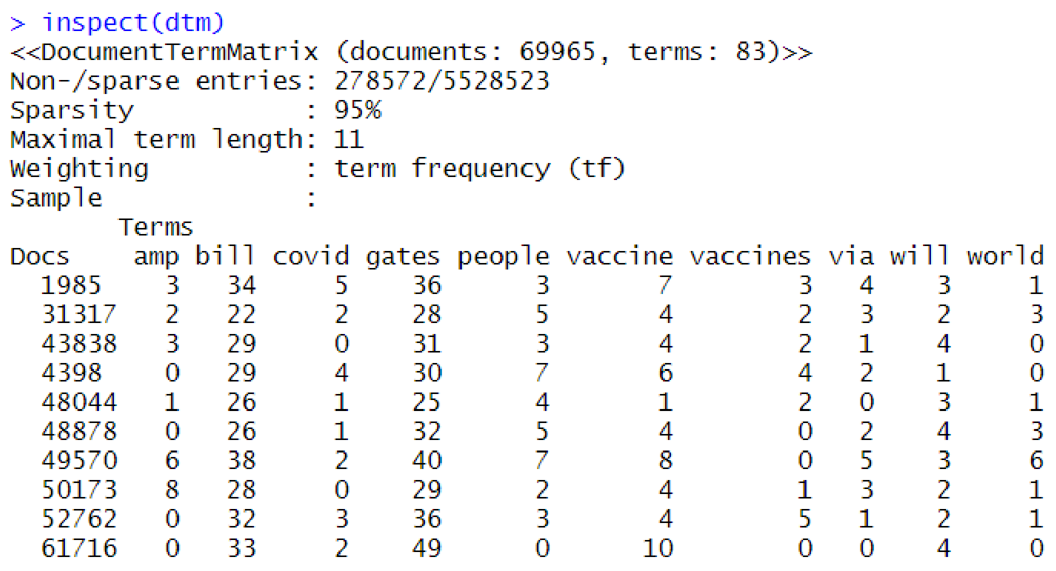
```
library(tm)
library(topicmodels)

library(readtext)
library(tidytext)
text <- readLines( "C:/Users/GTS/Desktop/billgates.text")

doc.vec <- VectorSource(text)
doc.corpus <- Corpus(doc.vec)
doc.corpus <- tm_map(doc.corpus, function(x) iconv(enc2utf8(x), sub = "byte"))

doc.corpus <- tm_map(doc.corpus, PlainTextDocument)
doc.corpus <- tm_map(doc.corpus, content_transformer(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)
```

The above R code initially reads the text file and converts it to a vector. The code contains functions that perform a series of tasks related to cleaning the dataset such as removing numbers, punctuation, stopwords and whitespace. This is a necessary step action towards creating a better and accurate LDA model. The cleaned dataset is positioned into a DocumentTermMatrix. DocumentTermMatrix helps to create a numerical representation of each word and place each word in a matrix (table).

After obtaining a DocumentTermMatrix, the last step involves removing terms that do not occur in at least 2% of all tweets.

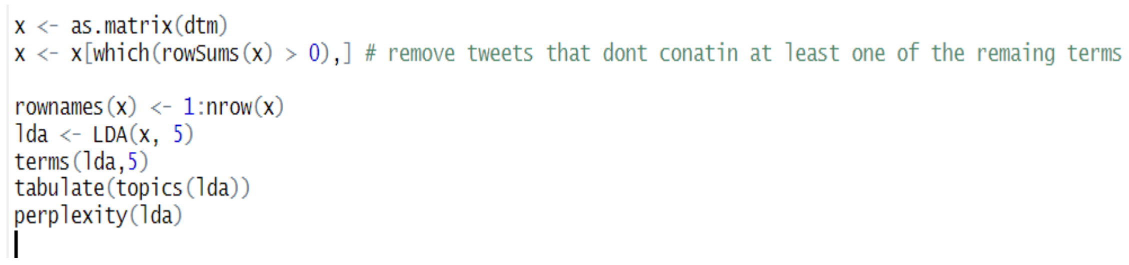
dtm <- removeSparseTerms(dtm, 0.98)
inspect(dtm)

The output of the above code is as follows:



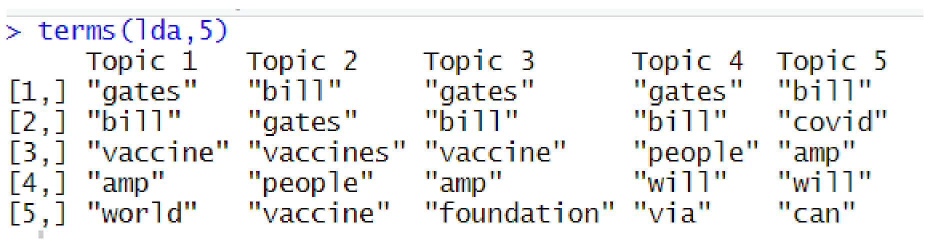
```
> inspect(dtm)
<<DocumentTermMatrix (documents: 69965, terms: 83)>>
Non-/sparse entries: 278572/5528523
Sparsity           : 95%
Maximal term length: 11
Weighting          : term frequency (tf)
Sample             :
       Terms
Docs    amp bill covid gates people vaccine vaccines via will world
  1985    3   34     5    36      3       7        3   4    3     1
  31317   2   22     2    28      5       4        2   3    2     3
  43838   3   29     0    31      3       4        2   1    4     0
  4398    0   29     4    30      7       6        4   2    1     0
  48044   1   26     1    25      4       1        2   0    3     1
  48878   0   26     1    32      5       4        0   2    4     3
  49570   6   38     2    40      7       8        0   5    3     6
  50173   8   28     0    29      2       4        1   3    2     1
  52762   0   32     3    36      3       4        5   1    2     1
  61716   0   33     2    49      0      10        0   0    4     0
```

After having the Document term matrix ready, the next step is to apply the DTM to the LDA model using five topics



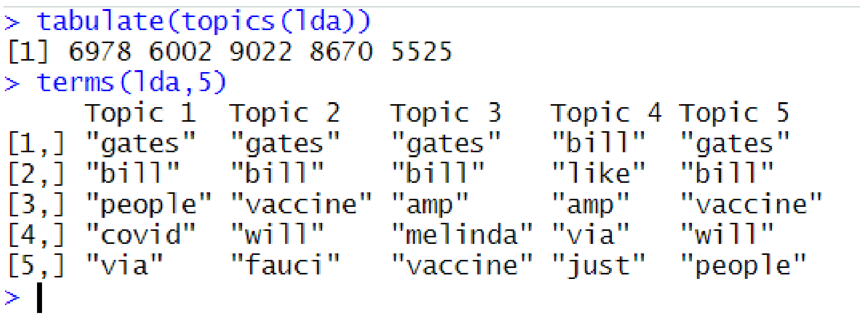
```
x <- as.matrix(dtm)
x <- x[which(rowSums(x) > 0),] # remove tweets that dont conatin at least one of the remaing terms

rownames(x) <- 1:nrow(x)
lda <- LDA(x, 5)
terms(lda,5)
tabulate(topics(lda))
perplexity(lda)
```

The output of the above code is shown below:



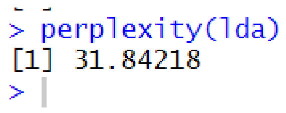
```
> terms(lda,5)
     Topic 1   Topic 2    Topic 3      Topic 4  Topic 5
[1,] "gates"   "bill"     "gates"      "gates"  "bill"
[2,] "bill"    "gates"    "bill"       "bill"   "covid"
[3,] "vaccine" "vaccines" "vaccine"    "people" "amp"
[4,] "amp"     "people"   "amp"        "will"   "will"
[5,] "world"   "vaccine"  "foundation" "via"    "can"
```

To get more details about the total amount of words in each of the five topics, *tabulate* function is used to highlight the desired results optimally.



```
> tabulate(topics(lda))
[1] 6978 6002 9022 8670 5525
> terms(lda,5)
     Topic 1  Topic 2   Topic 3   Topic 4 Topic 5
[1,] "gates"  "gates"   "gates"   "bill"  "gates"
[2,] "bill"   "bill"    "bill"    "like"  "bill"
[3,] "people" "vaccine" "amp"     "amp"   "vaccine"
[4,] "covid"  "will"    "melinda" "via"   "will"
[5,] "via"    "fauci"   "vaccine" "just"  "people"
>
```

The above result shows that topic 3 is the most talked-about is the dataset which reached to 9022 tweets throughout the entire period. Additionally, to measure how well the probability distribution of the model has occurred, the perplexity of the LDA model is determined. Lower value perplexity is preferred, and perplexity that is less than the number of topics specified indicates model over-fitting.



```
> perplexity(lda)
[1] 31.84218
>
```

8) Sentiment modelling:

Sentiment modelling or opinion mining has been an active research area in recent years (Pang and Lee 2008).

Researchers have adequately studied modelling constraints at the document level, sentence level, and characters level to determine the sentiment polarity (Pang and Lee, 2008). The critical information used in the majority of existing sentiment analysis techniques is a list of sentiment words or opinion words.

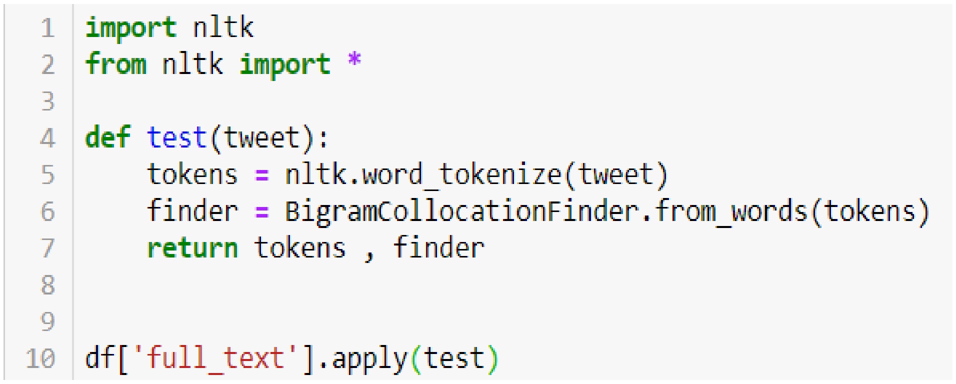
Positive sentiment words are words that help in uncovering desired states and undesirable states. Since these words bear sentiments by itself, it becomes a key to frequently use them in sentiment modelling tasks. However, it is also worthy to note that sentiment analysis based only on these words can result in biased results. There are various other types of expressions that do not expose any sentiments on their own, but when they put into particular contexts, they potentially bear sentiments. Liu (2020) explains several such expressions and their corresponding sentiment rules that are therefore introduced. All expressions must be extracted and associated problems solved prior sentiment analysis to achieve higher accuracy in deciphering the true sentiments declared.

Sentiment analysis can be defined as contextual text mining which identifies and extracts subjective information in documents or sentences and helping a business to understand the sentiment of their brand, product or service while determining whether the user’s sentiment is positive, negative, or neutral. This type of analysis requires either labelling of existing data or the use of trained classifiers. Classifiers are built using a variant of the bag-of-words or by adding custom ones.

For the purpose of this report, two methods of sentiment analysis will be taken

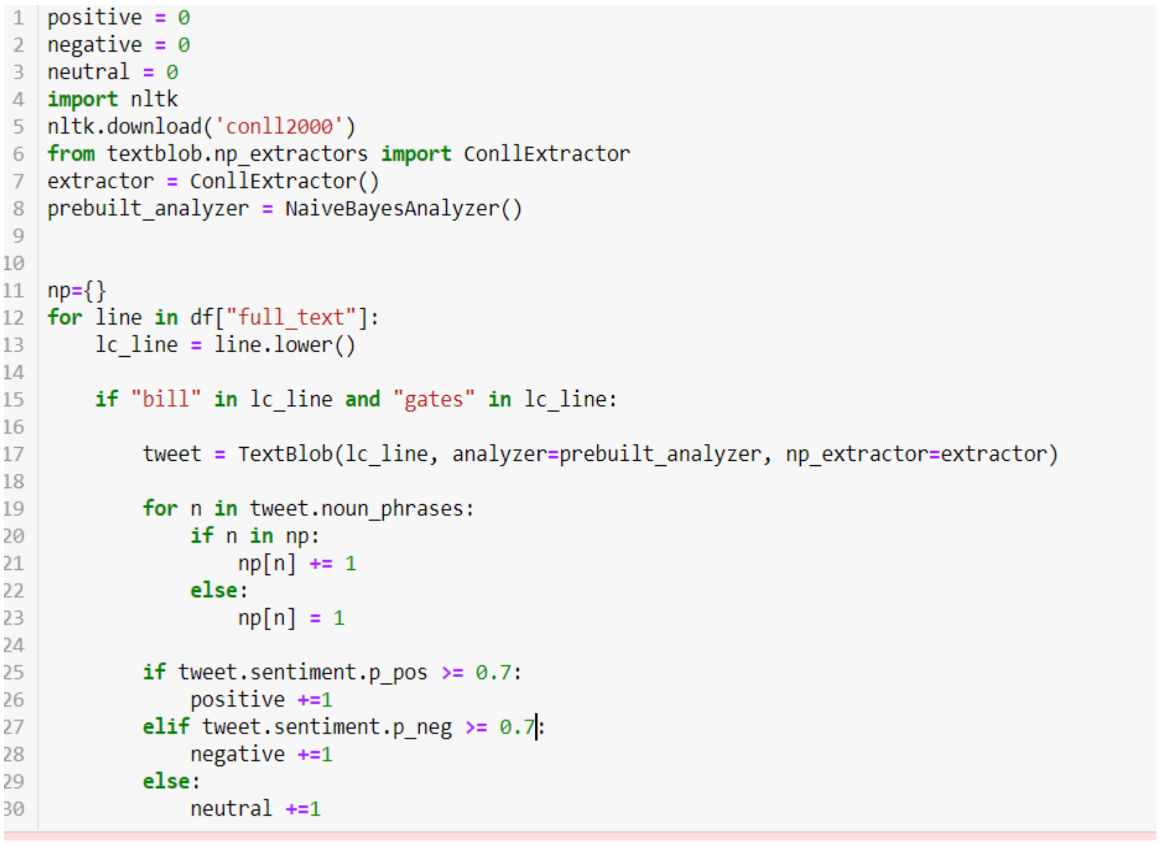
1) Using Naïve Bayes classifier:

   The first step is to tokenise the data. Tokens data helps in splitting up a larger body of text into smaller lines or words



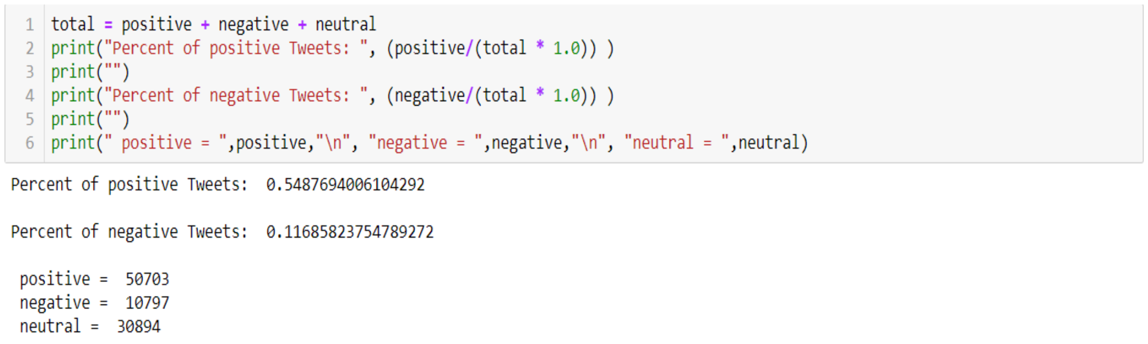
```
import nltk
from nltk import *

def test(tweet):
    tokens = nltk.word_tokenize(tweet)
    finder = BigramCollocationFinder.from_words(tokens)
    return tokens , finder


df['full_text'].apply(test)
```

   After having the data ready next step is to calculate the positive, negative and neutral tweets based on NaiveBayes classifier with p_pos and p_neg equals to 0.7.



```
positive = 0
negative = 0
neutral = 0
import nltk
nltk.download('conll2000')
from textblob.np_extractors import ConllExtractor
extractor = ConllExtractor()
prebuilt_analyzer = NaiveBayesAnalyzer()


np={}
for line in df["full_text"]:
    lc_line = line.lower()

    if "bill" in lc_line and "gates" in lc_line:

        tweet = TextBlob(lc_line, analyzer=prebuilt_analyzer, np_extractor=extractor)

        for n in tweet.noun_phrases:
            if n in np:
                np[n] += 1
            else:
                np[n] = 1

        if tweet.sentiment.p_pos >= 0.7:
            positive +=1
        elif tweet.sentiment.p_neg >= 0.7:
            negative +=1
        else:
            neutral +=1
```

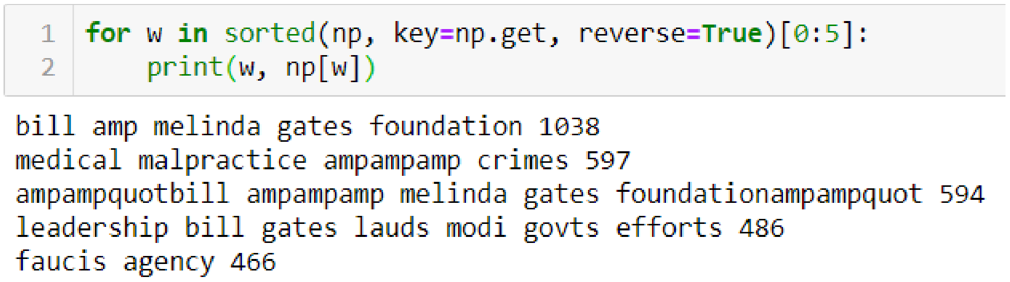
The above code creates Naïve Bayes classifier and picks two-word bill and gates to detect the positive negative and neutral tweets by using 70 per cent both in positive and negative. After having the model ready, the next step is to calculate the percentage of positive and negative value



```
total = positive + negative + neutral
print("Percent of positive Tweets: ", (positive/(total * 1.0)) )
print("")
print("Percent of negative Tweets: ", (negative/(total * 1.0)) )
print("")
print(" positive = ",positive,"\n", "negative = ",negative,"\n", "neutral = ",neutral)
```

```
Percent of positive Tweets:  0.5487694006104292

Percent of negative Tweets:  0.11685823754789272

 positive =  50703
 negative =  10797
 neutral =  30894
```

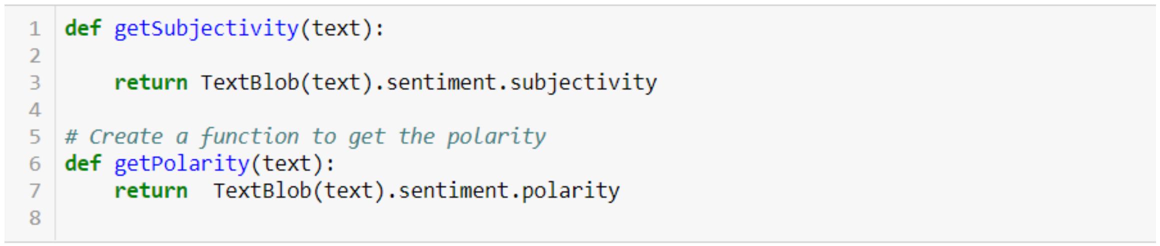
As shown above, the dataset contains 54% of positive values and 11% negative values which leave almost 35% for the neutral tweets. Next step is to show the top five most mentioned noun phrases.



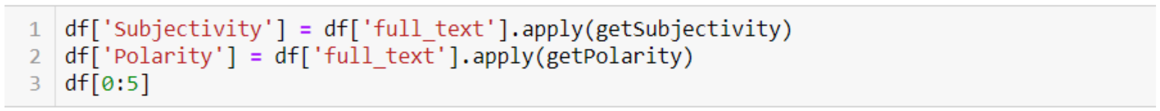
```
for w in sorted(np, key=np.get, reverse=True)[0:5]:
    print(w, np[w])
```

```
bill amp melinda gates foundation 1038
medical malpractice ampampamp crimes 597
ampampquotbill ampampamp melinda gates foundationampampquot 594
leadership bill gates lauds modi govts efforts 486
faucis agency 466
```

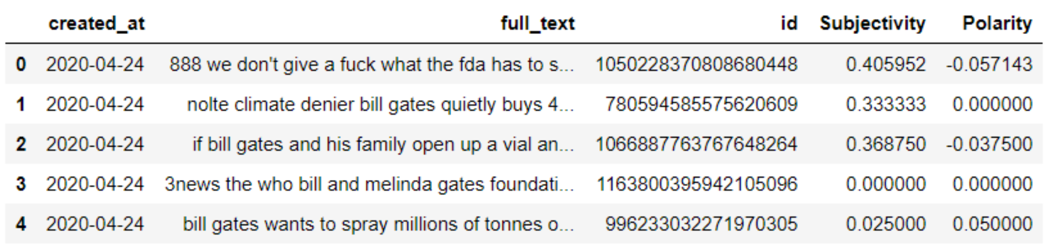
2) Using Texblob polarity method:



```
def getSubjectivity(text):

    return TextBlob(text).sentiment.subjectivity

# Create a function to get the polarity
def getPolarity(text):
    return  TextBlob(text).sentiment.polarity

```



```
df['Subjectivity'] = df['full_text'].apply(getSubjectivity)
df['Polarity'] = df['full_text'].apply(getPolarity)
df[0:5]
```



	created_at	full_text	id	Subjectivity	Polarity
0	2020-04-24	888 we don't give a fuck what the fda has to s...	1050228370808680448	0.405952	-0.057143
1	2020-04-24	nolte climate denier bill gates quietly buys 4...	780594585575620609	0.333333	0.000000
2	2020-04-24	if bill gates and his family open up a vial an...	1066887763767648264	0.368750	-0.037500
3	2020-04-24	3news the who bill and melinda gates foundati...	1163800395942105096	0.000000	0.000000
4	2020-04-24	bill gates wants to spray millions of tonnes o...	996233032271970305	0.025000	0.050000

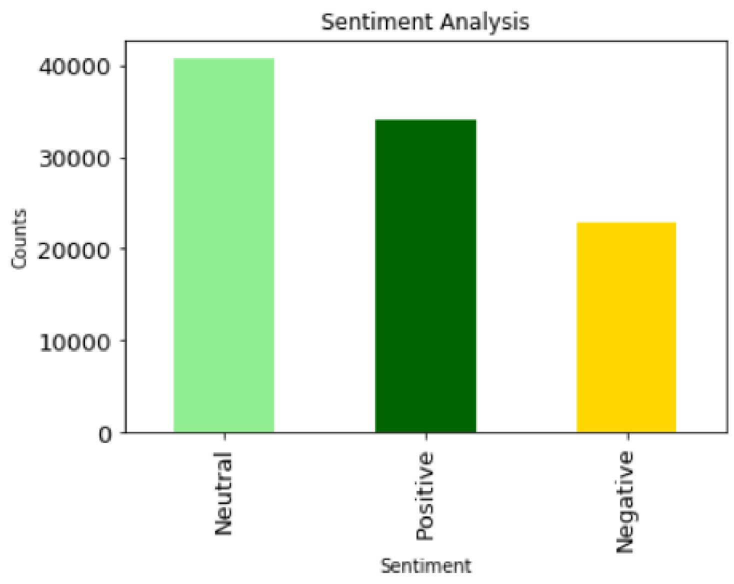
The above code creates two methods to calculate the polarity. Further, the function is called, which creates two new columns. After having the polarity value available next is to detect all positive, negative and neutral tweets.

```python
def getAnalysis(score):


    if score < 0:

        return 'Negative'
    elif score == 0:
        return 'Neutral'
    else:
        return 'Positive'


df['Analysis'] = df['Polarity'].apply(getAnalysis)

# Show the dataframe
df[0:5]
```

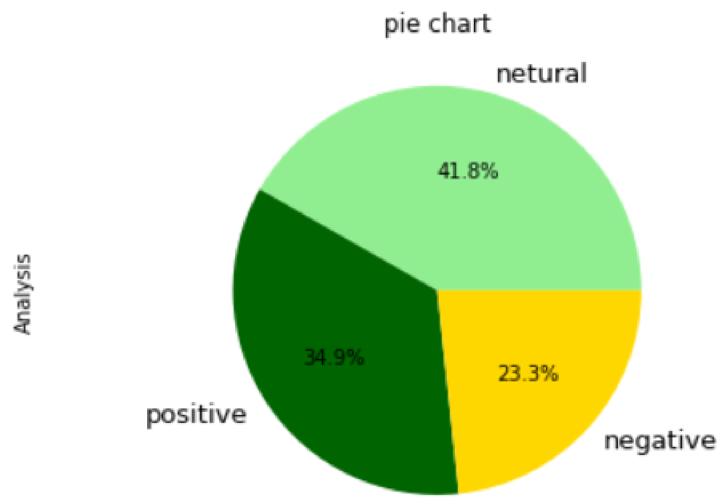
```
   created_at                                      full_text                   id  Subjectivity  Polarity  Analysis
0  2020-04-24  888 we don't give a fuck what the fda has to s...  1050228370808680448      0.405952 -0.057143  Negative
1  2020-04-24        nolte climate denier bill gates quietly buys 4...   780594585575620609      0.333333  0.000000   Neutral
2  2020-04-24       if bill gates and his family open up a vial an...  1066887763767648264      0.368750 -0.037500  Negative
3  2020-04-24  3news the who bill and melinda gates foundati...  1163800395942105096      0.000000  0.000000   Neutral
4  2020-04-24     bill gates wants to spray millions of tonnes o...   996233032271970305      0.025000  0.050000  Positive
```

The above code creates a function that detects the outcome for each tweet and saves them in a new column named *analysis. A*fter having all the data available next step is to plot the output. Two graphs shown below illustrates how may tweets are positive, negative and neutral.

```python
# Print the percentage of positive tweets
ptweets = df[df.Analysis == 'Positive']
ptweets = ptweets['full_text']
ptweets

round( (ptweets.shape[0] / df.shape[0]) * 100 , 1)

# Print the percentage of negative tweets
ntweets = df[df.Analysis == 'Negative']
ntweets = ntweets['full_text']
ntweets

round( (ntweets.shape[0] / df.shape[0]) * 100, 1)

# Show the value counts


# Plotting and visualizing the counts
plt.title('Sentiment Analysis')
plt.xlabel('Sentiment')
plt.ylabel('Counts')
df['Analysis'].value_counts().plot(kind = 'bar')
plt.show()
```



```
import matplotlib
matplotlib.rc('xtick', labelsize=13)
matplotlib.rc('ytick', labelsize=13)
colors = ['lightgreen','darkgreen', 'gold']

lable =   "netural","positive","negative"

plt.title("pie chart")

plt.axis('equal')

df['Analysis'].value_counts().plot(kind = 'pie',colors=colors,labels=lable,autopct='%1.1f%%')

plt.tight_layout()

plt.show()
```



Reference:

- Alvarez, R., 2020. *The Top 10 Brands That Make People Happier On Twitter*. [online] Blog.twitter.com. Available at: <https://blog.twitter.com/en_us/topics/insights/2019/the-top-10-brands-that-make-people-happier-on-twitter.html> [Accessed 17 April 2020].

- Cho, H., 2019. Topic Modeling. *Osong Public Health and Research Perspectives*, 10(3), pp.115-116.

- Coudriet, C., 2020. *Bill Gates Again World'S Second-Richest Person After One Day Behind Arnault*. [online] Forbes. Available at: <https://www.forbes.com/sites/cartercoudriet/2019/11/06/bill-gates-second-richest-arnault-bezos/#3d8d76df4e7c> [Accessed 12 April 2020].

- Davis, T., 2020. *What Is An API And How Does It Work?*. [online] Medium. Available at: <https://towardsdatascience.com/what-is-an-api-and-how-does-it-work-1dccd7a8219e> [Accessed 18 April 2020].

- Developer.twitter. 2020. *Standard Search API*. [online] Available at: <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets> [Accessed 16 April 2020].

- Developer.twitter.com. 2020. *Enterprise Search Apis*. [online] Available at: <https://developer.twitter.com/en/docs/tweets/search/api-reference/enterprise-search> [Accessed 17 April 2020].

- Developer.twitter. 2020. *Things Every Developer Should Know*. [online] Available at: <https://developer.twitter.com/en/docs/basics/things-every-developer-should-know> [Accessed 16 April 2020].

- Kulshrestha, R., 2020. *Latent Dirichlet Allocation(LDA)*. [online] Medium. Available at: <https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2> [Accessed 19 April 2020].
- Liu, B., 2020. *Sentiment Analysis And Opinion Mining* | *Synthesis Lectures On Human Language Technologies*. [online] Morganclaypool.com.

Available at: <https://www.morganclaypool.com/doi/abs/10.2200/s00416ed1v01y201204hlt016> [Accessed 20 April 2020].

- Statista. 2020. *Number Of Social Media Users Worldwide 2010-2021* | *Statista*. [online] Available at: <https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/> [Accessed 15 April 2020].

- Pang, B. and Lee, L., 2008. Opinion Mining and Sentiment Analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2), pp.1-135.

- Ft. 2020. *Transcript: Bill Gates Speaks To The FT About The Global Fight Against Coronavirus*. [online] Available at: <https://www.ft.com/content/13ddacc4-0ae4-4be1-95c5-1a32ab15956a?shareType=nongift> [Accessed 16 April 2020]. David

- Gatesfoundation.org. 2020. *Bill Gates*. [online] Available at: <https://www.gatesfoundation.org/who-we-are/general-information/leadership/executive-leadership-team/bill-gates> [Accessed 14 April 2020].

Appendix:

```
import tweepy

from textblob import TextBlob

from wordcloud import WordCloud

import pandas as pd

import numpy as np

import re

import matplotlib.pyplot as plt

import codecs

%matplotlib inline
```

```
accessToken = "1222893696464510984-5Lg9Z2Ibjrl2lzQL7bHbp3yqzTLJ4S"

accessTokenSecret = "7ILzmf9AVyDU5vtACuheBmBQ3lSetNzd6tb3uws5SCInd"

consumerKey= "ZhcHKqpmI89EKEBaLfhzhjTqR"

consumerSecret = "w8gzIotBPRLz7bx1BSwCvaVbeJpm5tjix8QLQmnIm98fcPVMj1"

# Create the authentication object

authenticate = tweepy.OAuthHandler(consumerKey, consumerSecret)

# Set the access token and access token secret

authenticate.set_access_token(accessToken, accessTokenSecret)

# Creating the API object while passing in auth information

api = tweepy.API(authenticate, wait_on_rate_limit = True, wait_on_rate_limit_notify=True)
```

```
def process_tweet(tweet, default_value=None):

    val = {}

    val['id'] = tweet.user.id if tweet.id else default_value

    val['full_text'] = tweet.full_text if tweet.full_text else default_value

    val['created_at'] = tweet.created_at if tweet.created_at else default_value
    return val
```

```
q = "bill gates OR melinda gates OR microsoft OR covid19 OR pandemic -filter:retweets"

posts = [process_tweet(tweet) for tweet in

        tweepy.Cursor(api.search, q = q,

                    lang="en",

                    since = "2020-04-18",

                    until = "2020-04-25",

                    sleep_on_rate_limit=False,

                    tweet_mode="extended").items()]


df = pd.DataFrame(posts)
```

```
df['full_text'] =  [word.lower() for word in df['full_text']]

f = codecs.open("billgates.text", "w",encoding="utf-8")

def cleanTxt(text):


    text = re.sub('@[A-Za-z0–9]+', '', text) #Removing @mentions

    text = re.sub('#', '', text) # Removing '#' hash tag

    text = re.sub('RT[\s]+', '', text) # Removing RT

    text = re.sub('https?:\/\/\S+', '', text) # Removing hyperlink

    text = re.sub(':','',text)

    text = re.sub('-','',text)

    text = re.sub(',','',text)

    text = re.sub('’','',text)

    text = re.sub('#','',text)

    text = re.sub('_','',text)

    text = re.sub(r"\…+", "" ,text)

    text = re.sub(r"\?+","",text)
    text = re.sub(r"\&+","",text)

    text = re.sub(r"\;+","",text)

    text = re.sub(r"\$+","",text)

    text = re.sub(r"\'+","",text)

    test = re.compile(u'[U00010000-U0010ffff]')

    f.write(text)

    return text

df.head() # before

df['full_text'] = df['full_text'].apply(cleanTxt)

df['created_at'] = pd.to_datetime(df['created_at']).dt.date


# Show the cleaned tweets

df.head() # after
```

```
def getSubjectivity(text):

    return TextBlob(text).sentiment.subjectivity

# Create a function to get the polarity

def getPolarity(text):

    return  TextBlob(text).sentiment.polarity
```

```
df['Subjectivity'] = df['full_text'].apply(getSubjectivity)

df['Polarity'] = df['full_text'].apply(getPolarity)

df[0:5]
```

```
def getAnalysis(score):

    if score < 0:


        return 'Negative'

    elif score == 0:
        return 'Neutral'
    else:
        return 'Positive'


df['Analysis'] = df['Polarity'].apply(getAnalysis)


# Show the dataframe
df[0:5]
```

```
print('Printing positive tweets:\n')
j=1
sortedDF = df.sort_values(by=['Polarity']) #Sort the tweets
for i in range(0, sortedDF.shape[0] ):


    if( sortedDF['Analysis'][i] == 'Positive'):


        print(str(j) + ') '+ sortedDF['full_text'][i])
        print()
        j= j+1
```

```
print('Printing negative tweets:\n')
j=1
sortedDF = df.sort_values(by=['Polarity'],ascending=False) #Sort the tweets
for i in range(0, sortedDF.shape[0] ):
      if( sortedDF['Analysis'][i] == 'Negative'):
        print(str(j) + ') '+sortedDF['full_text'][i])
        print()
        j=j+1
```

```
ptweets = df[df.Analysis == 'Positive']

ptweets = ptweets['full_text']

ptweets

round( (ptweets.shape[0] / df.shape[0]) * 100 , 1)

# Print the percentage of negative tweets

ntweets = df[df.Analysis == 'Negative']

ntweets = ntweets['full_text']

ntweets

round( (ntweets.shape[0] / df.shape[0]) * 100, 1)

# Show the value counts

# Plotting and visualizing the counts

plt.title('Sentiment Analysis')

plt.xlabel('Sentiment')

plt.ylabel('Counts')

df['Analysis'].value_counts().plot(kind = 'bar',color = ['lightgreen','darkgreen', 'gold'])

plt.show()
```

```
import matplotlib

matplotlib.rc('xtick', labelsize=13)

matplotlib.rc('ytick', labelsize=13)

colors = ['lightgreen','darkgreen', 'gold']

lable =   "netural","positive","negative"

plt.title("pie chart")

plt.axis('equal')

df['Analysis'].value_counts().plot(kind = 'pie',colors=colors,labels=lable,autopct='%1.1f%%')

plt.tight_layout()

plt.show()
```

```
from nltk.corpus import stopwords

stop = stopwords.words('english')

stop.append('I')

df['tweet_without_stopwords'] = df["full_text"].apply(lambda x: ' '.join([word for word in
x.split() if word not in (stop)]))
```

```
from collections import Counter

most_occur = Counter(" ".join(df["tweet_without_stopwords"]).split()).most_common(10)

print(most_occur)


clean =  pd.DataFrame(most_occur,columns=['words', 'count'])

clean
```

```
import matplotlib.cm as cm

fig, ax = plt.subplots(figsize=(8, 8))

colors = cm.rainbow(np.linspace(0, 1, 10))

# Plot horizontal bar graph

clean.sort_values(by='count').plot.barh(x='words',

                     y='count',

                     ax=ax,

                     color=colors)

ax.set_title("Top 10 Common Words Found in Tweets")

plt.show()

allWords = ' '.join([twts for twts in clean['words']])

wordCloud = WordCloud(width=600, height=400, random_state=21,
max_font_size=110).generate(allWords)

plt.imshow(wordCloud, interpolation="bilinear")

plt.axis('off')

plt.show()
```

```
gp1 =  df.groupby('created_at')['full_text'].nunique()

gp1


gp2 = df.groupby('created_at')['id'].nunique()

gp2


gp3 = df.groupby('id')['full_text'].nunique().sort_values(ascending=False)

gp3[0:10]
```

```
fig, ax = plt.subplots(figsize=(12,7))

df.groupby('created_at')['full_text'].nunique().plot(kind = 'bar',color=(0.2, 0.2, 0.2, 0.2),
edgecolor='red')
```

```
fig, ax = plt.subplots(figsize=(17,9))

df.groupby('created_at')['id'].nunique().plot(kind = 'bar',color=(0.2, 0.4, 0.6, 0.6))
```

```
from textblob.sentiments import NaiveBayesAnalyzer

prebuilt_analyzer = NaiveBayesAnalyzer()

for tweet in df['full_text']:

   x = TextBlob(tweet,analyzer=prebuilt_analyzer)
```

```
positive = 0

negative = 0

neutral = 0

import nltk

nltk.download('conll2000')

from textblob.np_extractors import ConllExtractor

extractor = ConllExtractor()

prebuilt_analyzer = NaiveBayesAnalyzer()
```

```
np={}
for line in df["full_text"]:
    lc_line = line.lower()

    if "bill" in lc_line and "gates" in lc_line:

        tweet = TextBlob(lc_line, analyzer=prebuilt_analyzer, np_extractor=extractor)

        for n in tweet.noun_phrases:
            if n in np:
                np[n] += 1
            else:
                np[n] = 1

        if tweet.sentiment.p_pos >= 0.7:
            positive +=1
        elif tweet.sentiment.p_neg >= 0.7:
            negative +=1
        else:
            neutral +=1
```

```
for w in sorted(np, key=np.get, reverse=True)[0:5]:
    print(w, np[w])
```

```
total = positive + negative + neutral
print("Percent of positive Tweets: ", (positive/(total * 1.0)) )
print("")
print("Percent of negative Tweets: ", (negative/(total * 1.0)) )

print("")

print(" positive = ",positive,"\n", "negative = ",negative,"\n", "neutral = ",neutral)
```