Sentimental Analysis of Twitter Data using API

Project Report

Pratik P. Patil
(202118023)
Course: Python (IT 606)

M.Sc. Data Science
Dhirubhai Ambani Institute of Informationa
and Communiction Technology
- Gandhinagar

Index

❖ *Problem Definition:*

Twitter is one of the world’s biggest social media platforms. It generates a huge amount of data every day, which comprises a variety of data from news to memes and personal opinions to trolls.
This project comprises of the sentimental analysis of this data using Natural Language Processing libraries in Python.

❖ *Problem Explanation:*

This project aims to get user input from the user for what Topic or Person they want to analyze along with the number of recent tweets they want to use for the analysis. This project proves to be very promising for getting a brief idea of public sentiment on any given topic, which could be used for several research purposes.

❖ *Project Description:*

- *Definition:*

"Sentiment analysis is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information."
In short, we can say that **Sentimental Analysis is the Process of detecting positive or negative sentiments in text.**

In this Project I have made a program for the detailed Sentimental Analysis of Twitter data, on any Keyword (Person/Topic) given by the user.

This is a detailed exploratory analysis which uses Twitter API for extracting data and then getting inference of public sentiment on any given Topic/Person.

Various factors required for basic analysis of any data are been covered and results are obtained using Python programming and a bunch of its Libraries.

❖ *Methodology / Analysis Performed:*

1) Installing and importing all the necessary libraries.

2) Setting up Twitter API Authentication credentials.

3) Taking user input on a favorable topic of analysis and the number of Tweets to be analyzed.

4) Finding the number of Positive, Negative and Neutral tweets along with their polarity. (Using the TextBlob and NLTK libraries)

5) Visualizing the sentiments using Pie chart.

6) Data cleaning – removing Tags, Retweets, Hashtags and Punctuation from the data.

7) Calculating individual Polarity of every single tweet.

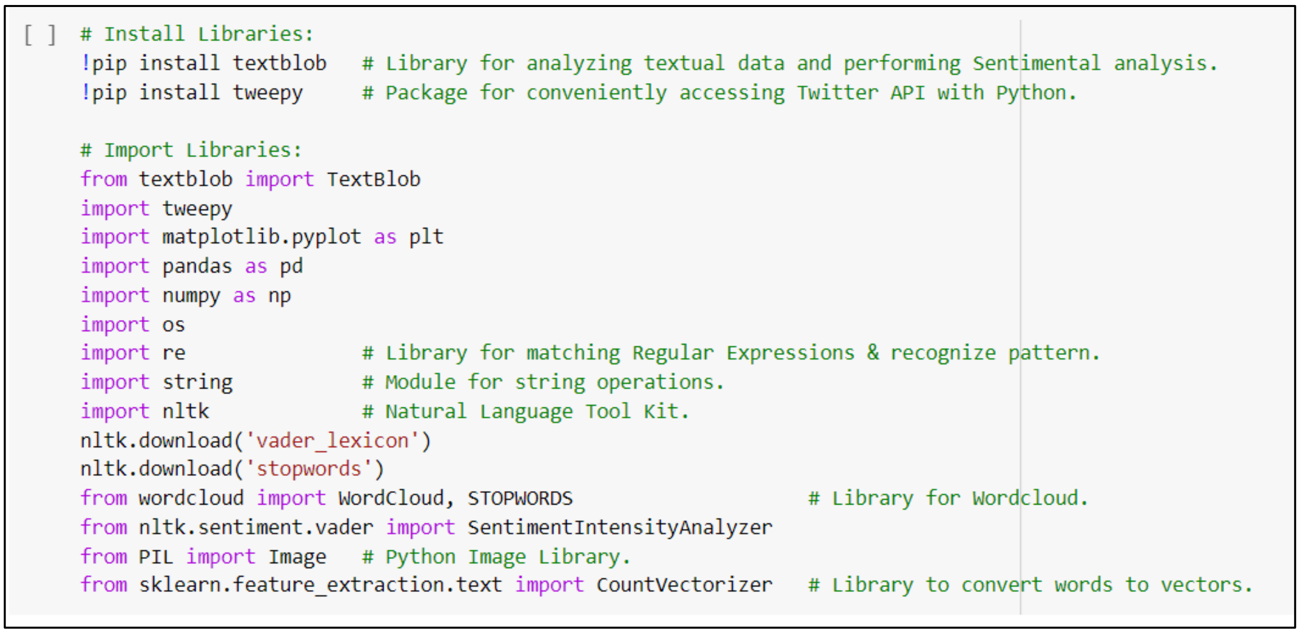
8) Visualizing the cleaned data after removing duplicates.

9) Creating Word-Cloud for Positive, Negative and Neutral sentiments to get good idea of which words are used more for describing the topic in hand.

10) Visualizing the average number of words for tweets of different sentiment.

11) Applying Tokenization, Stemmer and removing Stopwords from data and finding the most used words by using Countvectorizer.

❖ *Python Libraries Used:*



```
[ ] # Install Libraries:
    !pip install textblob   # Library for analyzing textual data and performing Sentimental analysis.
    !pip install tweepy     # Package for conveniently accessing Twitter API with Python.

    # Import Libraries:
    from textblob import TextBlob
    import tweepy
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np
    import os
    import re               # Library for matching Regular Expressions & recognize pattern.
    import string           # Module for string operations.
    import nltk             # Natural Language Tool Kit.
    nltk.download('vader_lexicon')
    nltk.download('stopwords')
    from wordcloud import WordCloud, STOPWORDS                    # Library for Wordcloud.
    from nltk.sentiment.vader import SentimentIntensityAnalyzer
    from PIL import Image   # Python Image Library.
    from sklearn.feature_extraction.text import CountVectorizer   # Library to convert words to vectors.
```

Main Libraries Used:

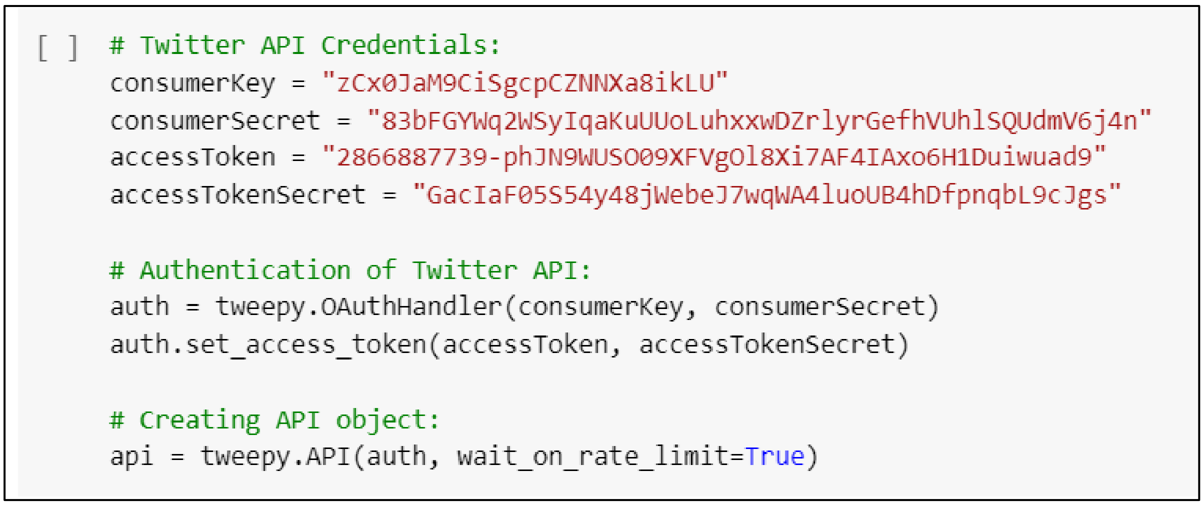
- *tweepy:* This package is used for conveniently accessing Twitter API through Python.
- *textblob:* Library for analysing textual data and performing sentimental analysis. It gives Polarity and Subjectivity of the text.
- *nltk:* Natural Language Tool Kit is an important library for sentimental analysis as it gives sentimental scores for textual data.
- *wordcloud:* Used to create a word-cloud for any given data.
- *re:* Regular Expression is used to recognize patterns in any String, and here it is used for data cleaning.

Other Libraries Used:

- pandas: Library for data manipulation and analysis, here it is used for converting arrays into dataframes.
- numpy: Library for processing arrays.
- matplotlib: Used for plotting graphs and charts.
- os: Operating System - Python’s standard utility module.
- string: Used for processing standard Python strings.
- PIL: Python Image Library
- CountVectorizer: It is from the sklearn library. Used to convert textual data into vectors based on their frequencies and processing them.

❖ *Code and Analysis:*

1. Initializing the API:



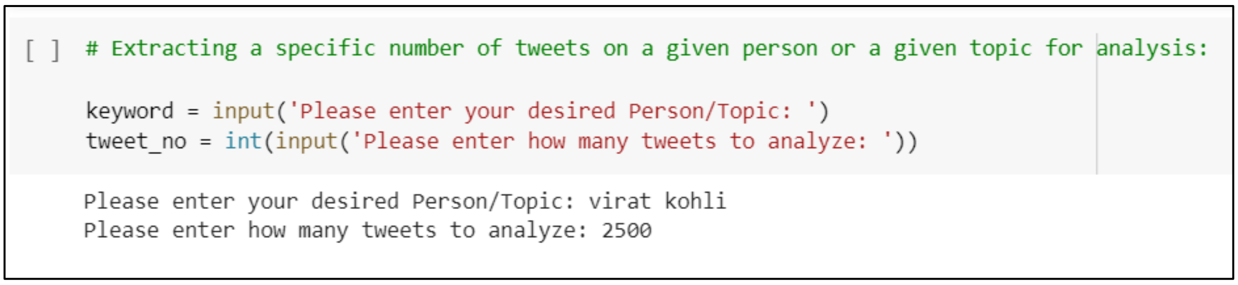
```
[ ]  # Twitter API Credentials:
     consumerKey = "zCx0JaM9CiSgcpCZNNXa8ikLU"
     consumerSecret = "83bFGYWq2WSyIqaKuUUoLuhxxwDZrlyrGefhVUhlSQUdmV6j4n"
     accessToken = "2866887739-phJN9WUSO09XFVgOl8Xi7AF4IAxo6H1Duiwuad9"
     accessTokenSecret = "GacIaF05S54y48jWebeJ7wqWA4luoUB4hDfpnqbL9cJgs"

     # Authentication of Twitter API:
     auth = tweepy.OAuthHandler(consumerKey, consumerSecret)
     auth.set_access_token(accessToken, accessTokenSecret)

     # Creating API object:
     api = tweepy.API(auth, wait_on_rate_limit=True)
```

As the first and the most important step, setting up the API credentials play an important role. The consumerKey and accessToken are generated from the 'Twitter Developer website' and the Authentication is done by using the *tweepy* library.

tweepy.OAuthHandler() function gives us 'Open Authentication' or open access to all the tweets on Twitter from any Python IDE.

tweepy.API() function sets the connection between Python and Twitter API using the Authentication Keys.

2. User Input for Keywords:

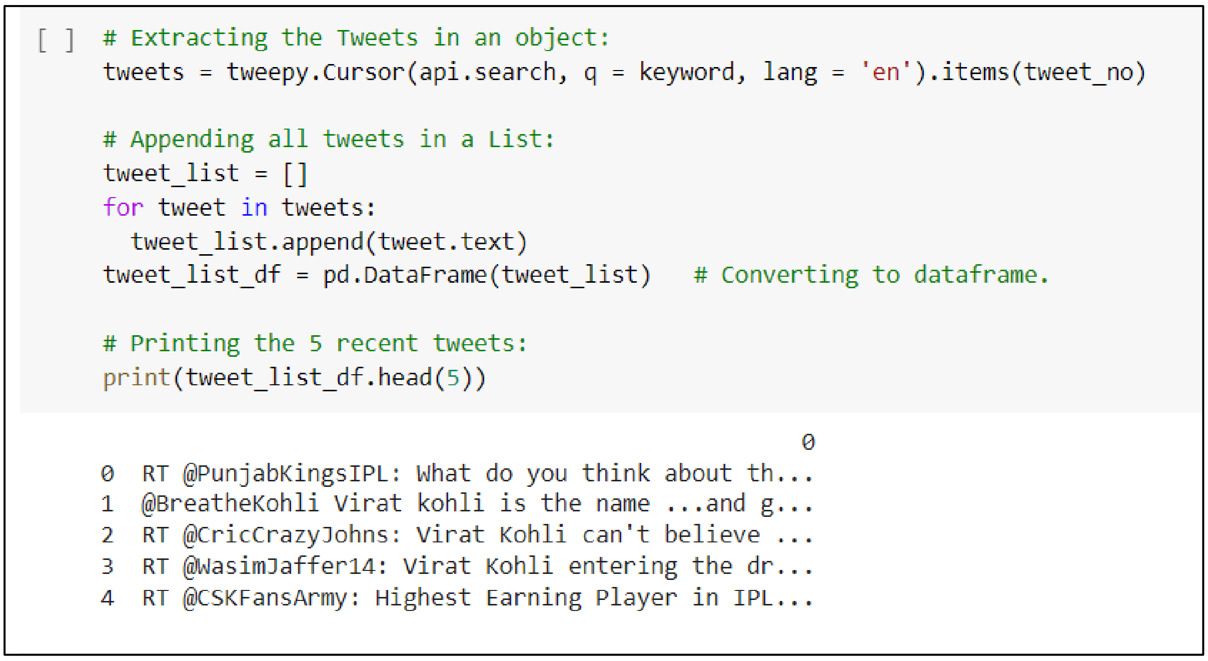


```
[ ]  # Extracting a specific number of tweets on a given person or a given topic for analysis:

     keyword = input('Please enter your desired Person/Topic: ')
     tweet_no = int(input('Please enter how many tweets to analyze: '))

     Please enter your desired Person/Topic: virat kohli
     Please enter how many tweets to analyze: 2500
```

I have taken two objects as user input: 'keyword' and 'tweet_no', where keyword is any topic/person of the user's interest on whom the analysis will be performed and tweet_no is the number of recent tweets the user needs to analyze.

For instance, I have taken here the keyword as 'virat kohli' and I'm analysing 2500 tweets.

3. Extracting Tweets from the API:



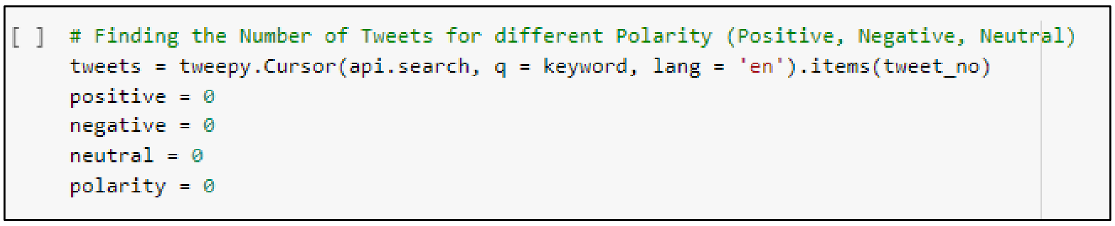
```
[ ]  # Extracting the Tweets in an object:
     tweets = tweepy.Cursor(api.search, q = keyword, lang = 'en').items(tweet_no)

     # Appending all tweets in a List:
     tweet_list = []
     for tweet in tweets:
       tweet_list.append(tweet.text)
     tweet_list_df = pd.DataFrame(tweet_list)   # Converting to dataframe.

     # Printing the 5 recent tweets:
     print(tweet_list_df.head(5))

                                                0
     0  RT @PunjabKingsIPL: What do you think about th...
     1  @BreatheKohli Virat kohli is the name ...and g...
     2  RT @CricCrazyJohns: Virat Kohli can't believe ...
     3  RT @WasimJaffer14: Virat Kohli entering the dr...
     4  RT @CSKFansArmy: Highest Earning Player in IPL...
```

Here, I have extracted the Tweets from Twitter by the *tweet.Cursor()* command, which performed an API search for the given keyword in English language and extracted the given number of tweets in the object named 'tweets'.

Thereafter, I have used a for loop to append the tweets in an empty array called 'tweet_list', after which I converted it into a dataframe by using the *pd.DataFrame()* command.

In this example, you can see the head of the dataframe which has 2500 tweets related to 'virat kohli'.

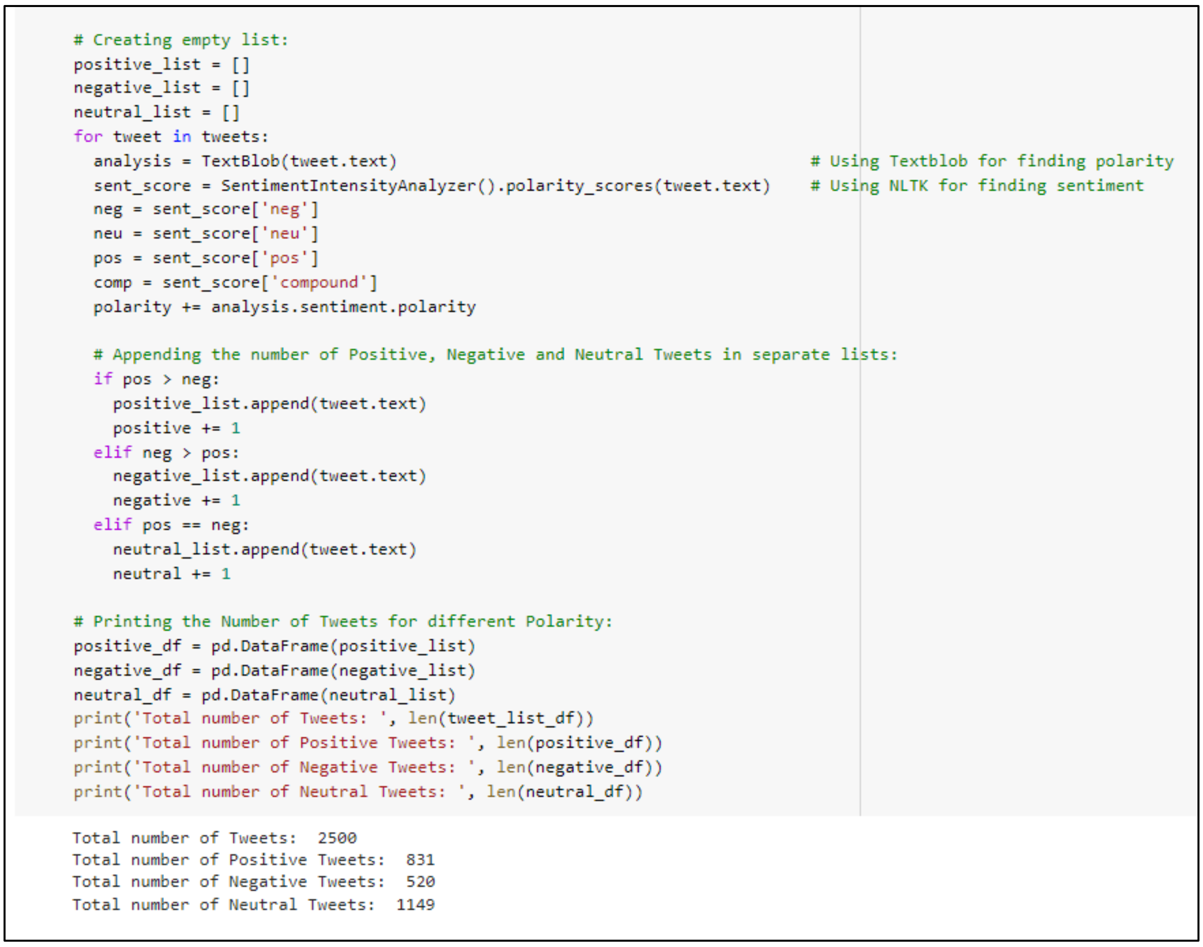
4. Number of Tweets for different Sentiments:



```
[ ]  # Finding the Number of Tweets for different Polarity (Positive, Negative, Neutral)
     tweets = tweepy.Cursor(api.search, q = keyword, lang = 'en').items(tweet_no)
     positive = 0
     negative = 0
     neutral = 0
     polarity = 0
```

For initialization, I have taken three types of sentiments namely Positive, Negative and Neutral, and initializer their values as 0.

Further I calculated how many Positive, Negative or Neutral tweets are present in our extracted data.



```
# Creating empty list:
positive_list = []
negative_list = []
neutral_list = []
for tweet in tweets:
  analysis = TextBlob(tweet.text)                                          # Using Textblob for finding polarity
  sent_score = SentimentIntensityAnalyzer().polarity_scores(tweet.text)    # Using NLTK for finding sentiment
  neg = sent_score['neg']
  neu = sent_score['neu']
  pos = sent_score['pos']
  comp = sent_score['compound']
  polarity += analysis.sentiment.polarity

  # Appending the number of Positive, Negative and Neutral Tweets in separate lists:
  if pos > neg:
    positive_list.append(tweet.text)
    positive += 1
  elif neg > pos:
    negative_list.append(tweet.text)
    negative += 1
  elif pos == neg:
    neutral_list.append(tweet.text)
    neutral += 1

# Printing the Number of Tweets for different Polarity:
positive_df = pd.DataFrame(positive_list)
negative_df = pd.DataFrame(negative_list)
neutral_df = pd.DataFrame(neutral_list)
print('Total number of Tweets: ', len(tweet_list_df))
print('Total number of Positive Tweets: ', len(positive_df))
print('Total number of Negative Tweets: ', len(negative_df))
print('Total number of Neutral Tweets: ', len(neutral_df))
```

```
Total number of Tweets:  2500
Total number of Positive Tweets:  831
Total number of Negative Tweets:  520
Total number of Neutral Tweets:  1149
```

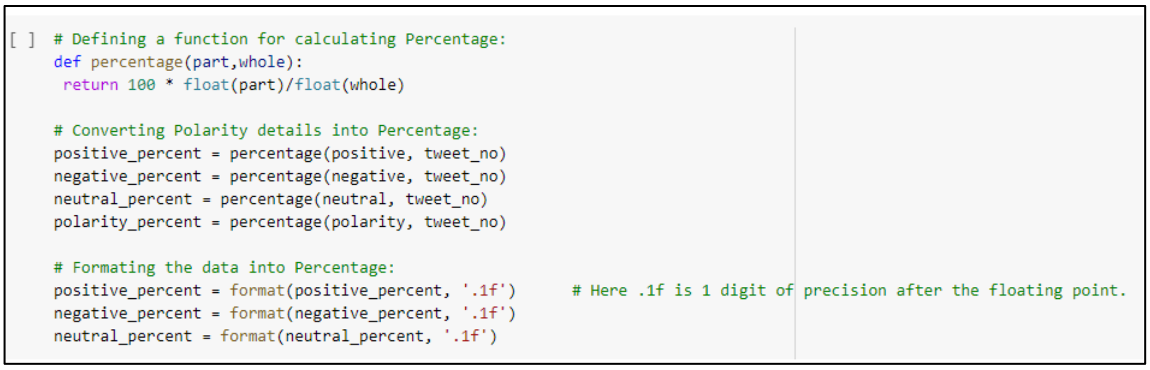
I have also created empty lists for the three sentiments, where I can store each type of tweet separately.

In the *for* loop, I have used the *TextBlob()* command to find the Polarity of every single tweet. After which I have used *SentimentIntensityAnalyzer().polarity_scores()* function for calculating individual polarity score for each tweet, which will be stored in the objects 'pos', 'neu' and 'neg' for comparison in the *If else* statement.

From the *If else* statement, I assigned each tweet in the list they belong to and then printed the length of the lists to get conclusion on the number of tweets per sentiments for the giver keyword.

In this particular example, you can see that there are high 'Neutral' and 'Positive' tweets compared to 'Negative'.

5. Calculating Percentage and Visualization:

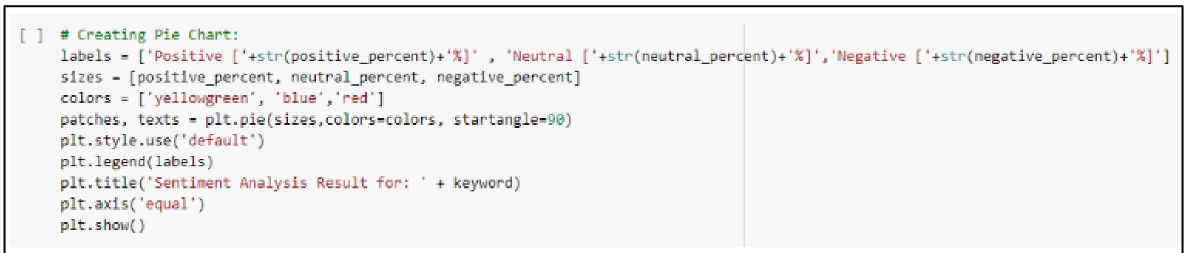


```
[ ] # Defining a function for calculating Percentage:
    def percentage(part,whole):
     return 100 * float(part)/float(whole)

    # Converting Polarity details into Percentage:
    positive_percent = percentage(positive, tweet_no)
    negative_percent = percentage(negative, tweet_no)
    neutral_percent = percentage(neutral, tweet_no)
    polarity_percent = percentage(polarity, tweet_no)

    # Formating the data into Percentage:
    positive_percent = format(positive_percent, '.1f')      # Here .1f is 1 digit of precision after the floating point.
    negative_percent = format(negative_percent, '.1f')
    neutral_percent = format(neutral_percent, '.1f')
```

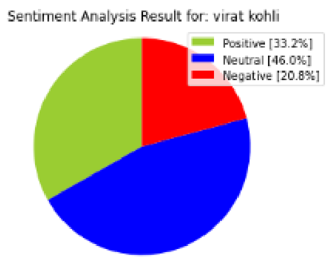
I have defined a simple function for finding the percentage of the 'number of tweets' for all the sentiment types and it will return a float value.

After calculating percentages, I have formatted the output using the *format()* function, where the output will give only one value after the decimal point.

This data of percentage will be used to plot a Pie chart after this.



```
[ ] # Creating Pie Chart:
    labels = ['Positive ['+str(positive_percent)+'%]' , 'Neutral ['+str(neutral_percent)+'%]','Negative ['+str(negative_percent)+'%]']
    sizes = [positive_percent, neutral_percent, negative_percent]
    colors = ['yellowgreen', 'blue','red']
    patches, texts = plt.pie(sizes,colors=colors, startangle=90)
    plt.style.use('default')
    plt.legend(labels)
    plt.title('Sentiment Analysis Result for: ' + keyword)
    plt.axis('equal')
    plt.show()
```

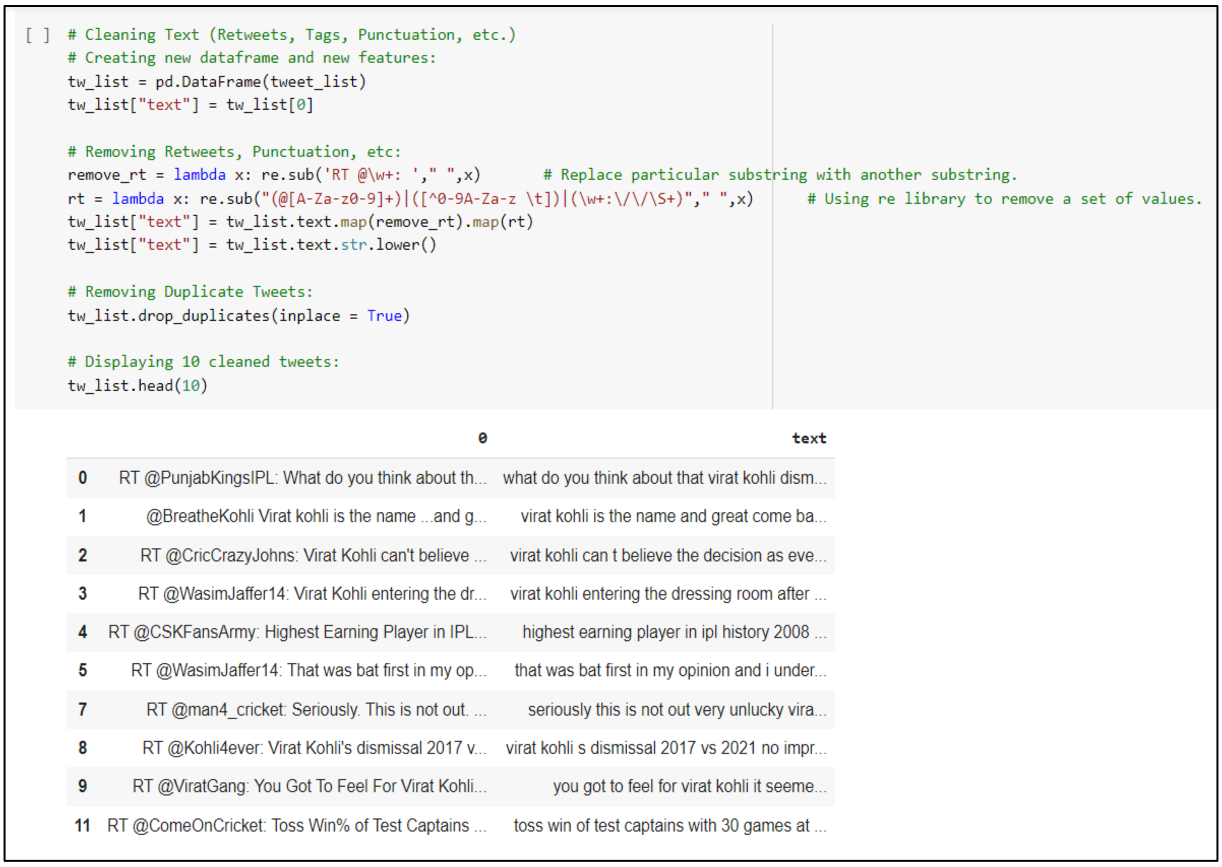


In this example, I have plotted this Pie chart for the given keyword by using the above calculated percentage values.

This plot has been created by using the matplotlib.pyplot library.

As we can see, this figure shows that the highest number of tweets are of Neutral sentiment, followed by Positive and then Negative sentiment.

6. Data Cleaning for further analysis:



```
[ ] # Cleaning Text (Retweets, Tags, Punctuation, etc.)
    # Creating new dataframe and new features:
    tw_list = pd.DataFrame(tweet_list)
    tw_list["text"] = tw_list[0]

    # Removing Retweets, Punctuation, etc:
    remove_rt = lambda x: re.sub('RT @\w+: '," ",x)       # Replace particular substring with another substring.
    rt = lambda x: re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)"," ",x)      # Using re library to remove a set of values.
    tw_list["text"] = tw_list.text.map(remove_rt).map(rt)
    tw_list["text"] = tw_list.text.str.lower()

    # Removing Duplicate Tweets:
    tw_list.drop_duplicates(inplace = True)

    # Displaying 10 cleaned tweets:
    tw_list.head(10)
```

```
                                                  0                                          text
 0   RT @PunjabKingsIPL: What do you think about th...  what do you think about that virat kohli dism...
 1        @BreatheKohli Virat kohli is the name ...and g...     virat kohli is the name and great come ba...
 2      RT @CricCrazyJohns: Virat Kohli can't believe ...   virat kohli can t believe the decision as eve...
 3      RT @WasimJaffer14: Virat Kohli entering the dr...   virat kohli entering the dressing room after ...
 4   RT @CSKFansArmy: Highest Earning Player in IPL...      highest earning player in ipl history 2008 ...
 5     RT @WasimJaffer14: That was bat first in my op...    that was bat first in my opinion and i under...
 7        RT @man4_cricket: Seriously. This is not out. ...    seriously this is not out very unlucky vira...
 8      RT @Kohli4ever: Virat Kohli's dismissal 2017 v...  virat kohli s dismissal 2017 vs 2021 no impr...
 9      RT @ViratGang: You Got To Feel For Virat Kohli...        you got to feel for virat kohli it seeme...
11   RT @ComeOnCricket: Toss Win% of Test Captains ...   toss win of test captains with 30 games at ...
```

This part of the project is important because until now, we performed all the analysis on the raw data that we extracted from the Twitter API.

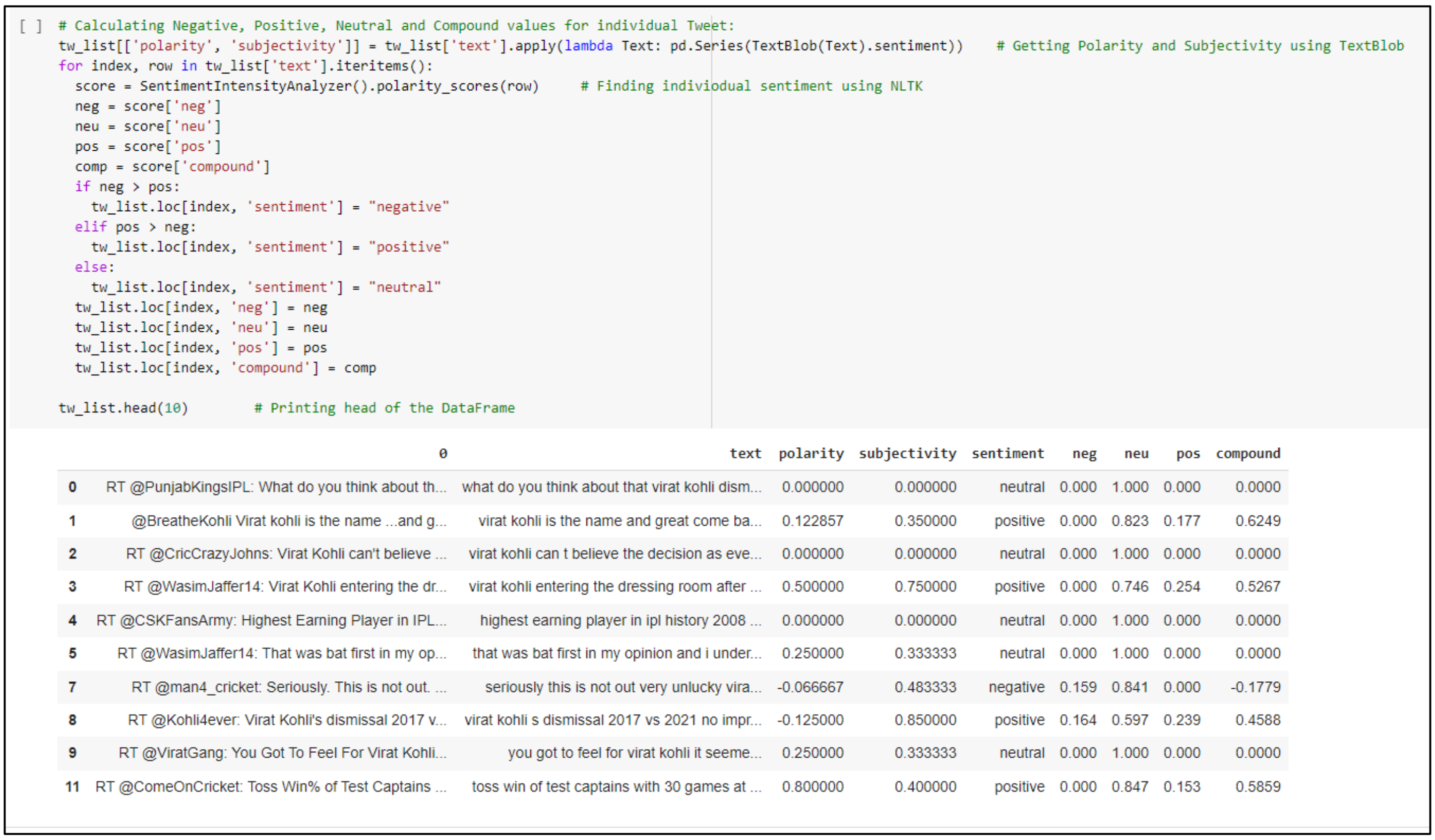
Now, by using the *re* library, we find all the patterns related to Re-Tweets, punctuations and Hashtags thereby replacing all of that with a space (“ ”).

I have also converted all the strings in the tweets in this dataframe in lower case for the ease of analysis.

And finally after cleaning the data, I have also removed all the duplicate tweets (if there were any) with the help of *drop_duplicates()* function.

In this Image you can see that there are two columns created named ‘0’ and ‘text’ respectively, where the column ‘0’ contains all the original tweets whereas the column ‘text’ contains all the cleaned tweets after removing duplicates and both the columns are mapped to each other for maintaining the number of rows in both.

7. Analysis of Cleaned data:



```
[ ] # Calculating Negative, Positive, Neutral and Compound values for individual Tweet:
    tw_list[['polarity', 'subjectivity']] = tw_list['text'].apply(lambda Text: pd.Series(TextBlob(Text).sentiment))    # Getting Polarity and Subjectivity using TextBlob
    for index, row in tw_list['text'].iteritems():
      score = SentimentIntensityAnalyzer().polarity_scores(row)     # Finding individual sentiment using NLTK
      neg = score['neg']
      neu = score['neu']
      pos = score['pos']
      comp = score['compound']
      if neg > pos:
        tw_list.loc[index, 'sentiment'] = "negative"
      elif pos > neg:
        tw_list.loc[index, 'sentiment'] = "positive"
      else:
        tw_list.loc[index, 'sentiment'] = "neutral"
      tw_list.loc[index, 'neg'] = neg
      tw_list.loc[index, 'neu'] = neu
      tw_list.loc[index, 'pos'] = pos
      tw_list.loc[index, 'compound'] = comp

    tw_list.head(10)        # Printing head of the DataFrame
```

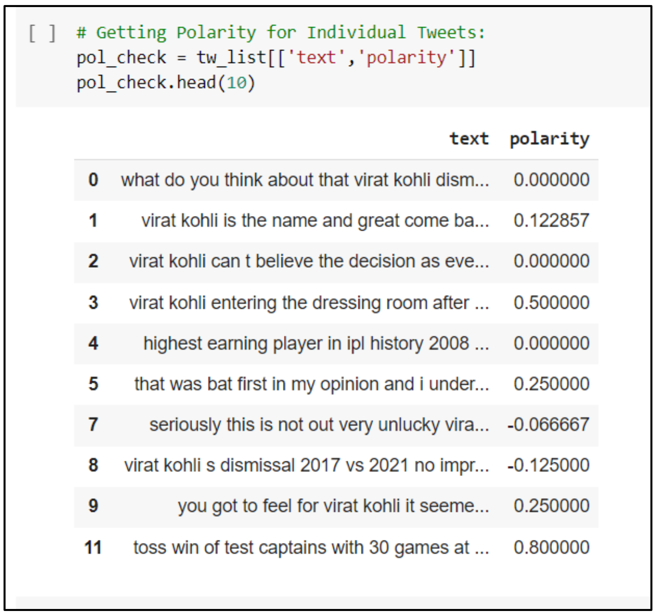
	0	text	polarity	subjectivity	sentiment	neg	neu	pos	compound
0	RT @PunjabKingsIPL: What do you think about th...	what do you think about that virat kohli dism...	0.000000	0.000000	neutral	0.000	1.000	0.000	0.0000
1	@BreatheKohli Virat kohli is the name ...and g...	virat kohli is the name and great come ba...	0.122857	0.350000	positive	0.000	0.823	0.177	0.6249
2	RT @CricCrazyJohns: Virat Kohli can't believe ...	virat kohli can t believe the decision as eve...	0.000000	0.000000	neutral	0.000	1.000	0.000	0.0000
3	RT @WasimJaffer14: Virat Kohli entering the dr...	virat kohli entering the dressing room after ...	0.500000	0.750000	positive	0.000	0.746	0.254	0.5267
4	RT @CSKFansArmy: Highest Earning Player in IPL...	highest earning player in ipl history 2008 ...	0.000000	0.000000	neutral	0.000	1.000	0.000	0.0000
5	RT @WasimJaffer14: That was bat first in my op...	that was bat first in my opinion and i under...	0.250000	0.333333	neutral	0.000	1.000	0.000	0.0000
7	RT @man4_cricket: Seriously. This is not out. ...	seriously this is not out very unlucky vira...	-0.066667	0.483333	negative	0.159	0.841	0.000	-0.1779
8	RT @Kohli4ever: Virat Kohli's dismissal 2017 v...	virat kohli s dismissal 2017 vs 2021 no impr...	-0.125000	0.850000	positive	0.164	0.597	0.239	0.4588
9	RT @ViratGang: You Got To Feel For Virat Kohli...	you got to feel for virat kohli it seeme...	0.250000	0.333333	neutral	0.000	1.000	0.000	0.0000
11	RT @ComeOnCricket: Toss Win% of Test Captains ...	toss win of test captains with 30 games at ...	0.800000	0.400000	positive	0.000	0.847	0.153	0.5859

In this part of the project, once again I used the *TextBlob()* and *SentimentIntensityAnalyzer().polarity_check()* commands to determine the Polarity, Subjectivity, Sentiment and individual sentiment scores along with their Compound [addition of all sentiment scores (pos + neu + neg) and then normalizing it between 0 and 1] for all the tweets individually and creating separate columns for these factors in the dataframe.

This data gives an idea of how the Sentiment scores and Polarity have a relationship in determining the sentiment of any given statement (Tweets in this case).

By looking at each single tweet, we can determine our own intelligence, what type of words or sentences used in each tweet can change its polarity and sentiment.

8. Visualizing the Polarity:

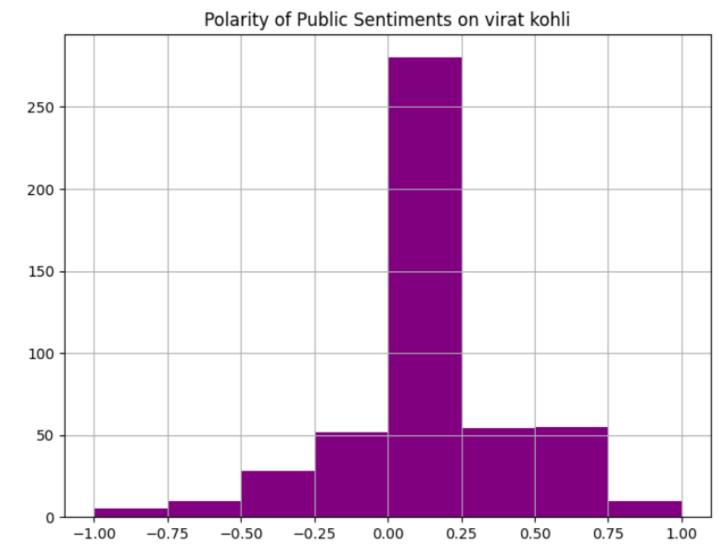


```
[ ] # Getting Polarity for Individual Tweets:
    pol_check = tw_list[['text','polarity']]
    pol_check.head(10)
```

	text	polarity
0	what do you think about that virat kohli dism...	0.000000
1	virat kohli is the name and great come ba...	0.122857
2	virat kohli can t believe the decision as eve...	0.000000
3	virat kohli entering the dressing room after ...	0.500000
4	highest earning player in ipl history 2008 ...	0.000000
5	that was bat first in my opinion and i under...	0.250000
7	seriously this is not out very unlucky vira...	-0.066667
8	virat kohli s dismissal 2017 vs 2021 no impr...	-0.125000
9	you got to feel for virat kohli it seeme...	0.250000
11	toss win of test captains with 30 games at ...	0.800000

In this example, I have taken all the tweets related to our keyword 'virat kohli' and their individual polarities for visualizing them.

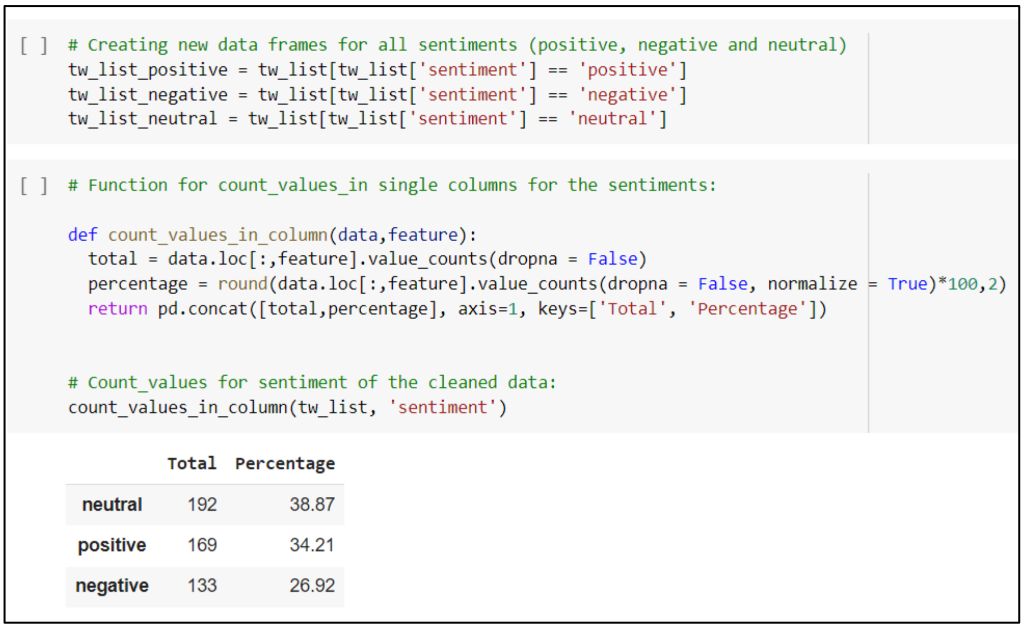
```
[ ] # Plotting Bar graph for visualizing the distribution of Polarity:
    fig, ax = plt.subplots(figsize = (8,6))
    pol_check.hist(bins=[-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1],
                   ax=ax, color = 'purple')
    plt.title('Polarity of Public Sentiments on ' + keyword)
    plt.show()
```



This histogram is created by using *matplotlib.pyplot* library's *hist()* function.

Here we can easily see that the region near 0 on x-axis has high frequency which means that most of the tweets are neutral in nature. Then the next high frequency is seen between 0 and 1 which are the positive tweets and the least i.e., 0 to -1 are the negative tweets.

9. Visualizing the cleaned Tweets:



```
[ ] # Creating new data frames for all sentiments (positive, negative and neutral)
    tw_list_positive = tw_list[tw_list['sentiment'] == 'positive']
    tw_list_negative = tw_list[tw_list['sentiment'] == 'negative']
    tw_list_neutral = tw_list[tw_list['sentiment'] == 'neutral']
```

```
[ ] # Function for count_values_in single columns for the sentiments:

    def count_values_in_column(data,feature):
      total = data.loc[:,feature].value_counts(dropna = False)
      percentage = round(data.loc[:,feature].value_counts(dropna = False, normalize = True)*100,2)
      return pd.concat([total,percentage], axis=1, keys=['Total', 'Percentage'])


    # Count_values for sentiment of the cleaned data:
    count_values_in_column(tw_list, 'sentiment')
```

	Total	Percentage
neutral	192	38.87
positive	169	34.21
negative	133	26.92

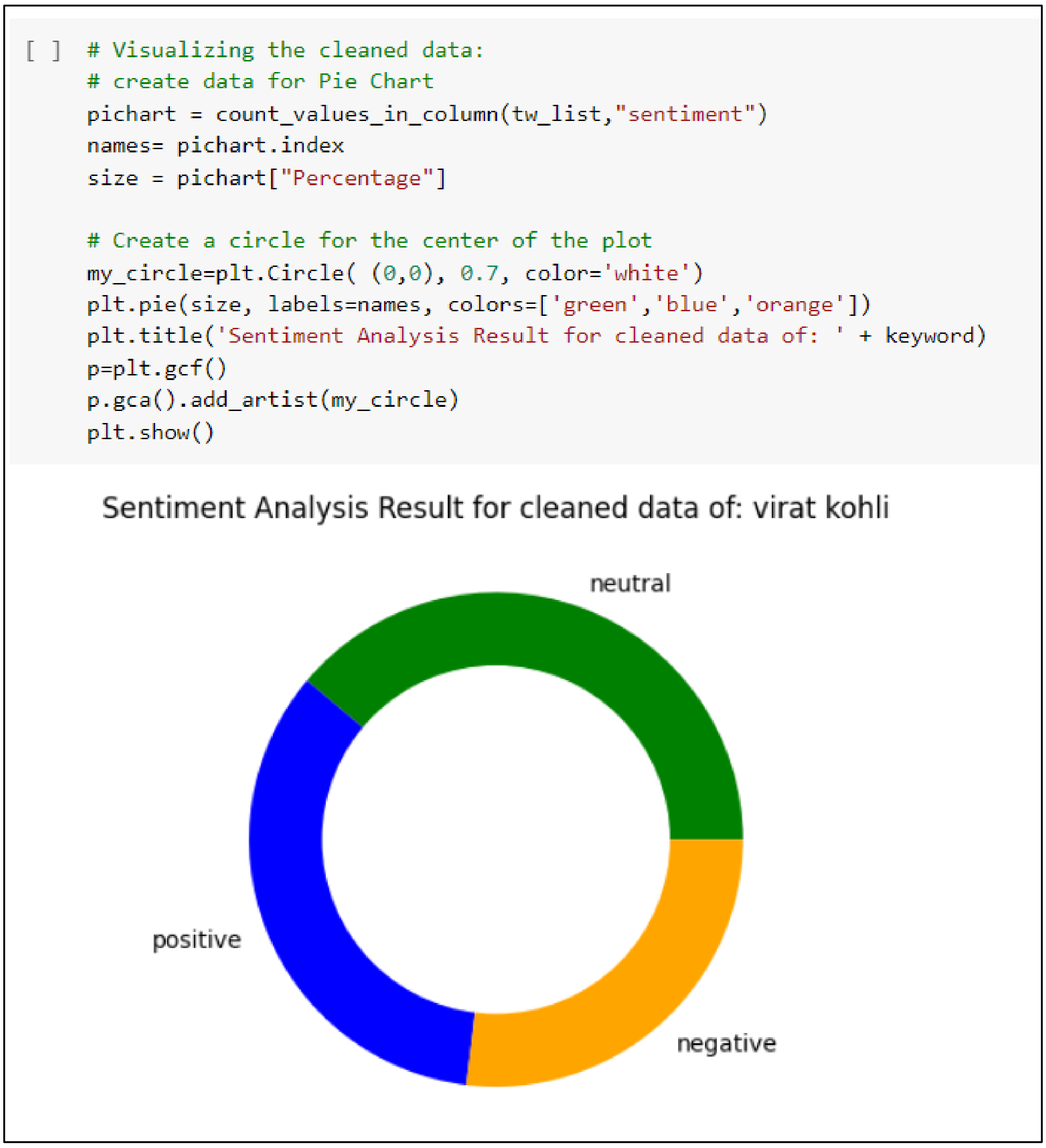
This part is about the cleaned data, where we have removed all the duplicate tweets and now, we will see whether the results have changed.

Comparing this result in the example with the result for raw data, we see that the number of tweets for each sentiment have reduced by a significant number, through which we can say that there were a huge number of duplicate tweets in this data.

Furthermore, by looking at the ‘Percentage’ column we can say that still after removing a huge quantity of unwanted data, there is a high percentage of Neutral tweets, then Positive tweets on the second position and as the previous result Negatives are the least.

This data can be said to more accurately represent the public sentiment on our selected keyword.

After this we visualize this data using a donut graph:

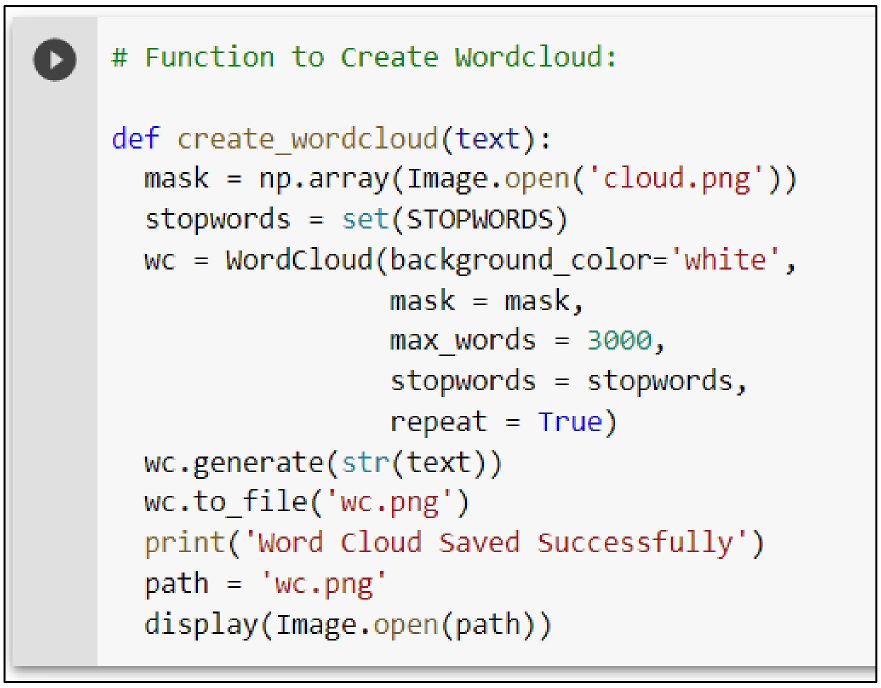


```
[ ]  # Visualizing the cleaned data:
     # create data for Pie Chart
     pichart = count_values_in_column(tw_list,"sentiment")
     names= pichart.index
     size = pichart["Percentage"]

     # Create a circle for the center of the plot
     my_circle=plt.Circle( (0,0), 0.7, color='white')
     plt.pie(size, labels=names, colors=['green','blue','orange'])
     plt.title('Sentiment Analysis Result for cleaned data of: ' + keyword)
     p=plt.gcf()
     p.gca().add_artist(my_circle)
     plt.show()
```

Again, this graph is created using *matplotlib* library.

And we can see the variation in the size of the donut sizes for each colour Blue, Green and Orange representing Positive, Neutral and Negative respectively for the given keyword (i.e., 'virat kohli' in this example)

10. Creating Wordcloud:



```
# Function to Create Wordcloud:

def create_wordcloud(text):
  mask = np.array(Image.open('cloud.png'))
  stopwords = set(STOPWORDS)
  wc = WordCloud(background_color='white',
                 mask = mask,
                 max_words = 3000,
                 stopwords = stopwords,
                 repeat = True)
  wc.generate(str(text))
  wc.to_file('wc.png')
  print('Word Cloud Saved Successfully')
  path = 'wc.png'
  display(Image.open(path))
```

“Word Clouds (also known as wordle, word collage or tag cloud) are visual representations of words that give greater prominence to words that appear more frequently.”

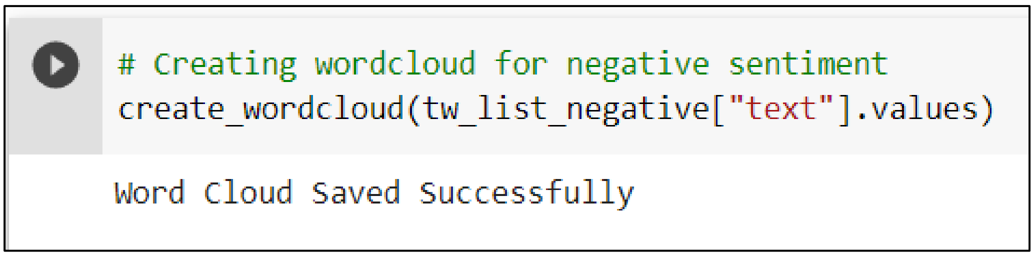
Overall, wordclouds are a very effective way to represent the trend of a huge amount of data and show us what type of words people use while describing someone/something/some event.

In this function, I have used a cloud mask named ‘clod.png’ which is a simple image of a cloud and would act as a mask while creating the wordcloud, which would result in the wordcloud being created in the shape of an actual cloud.

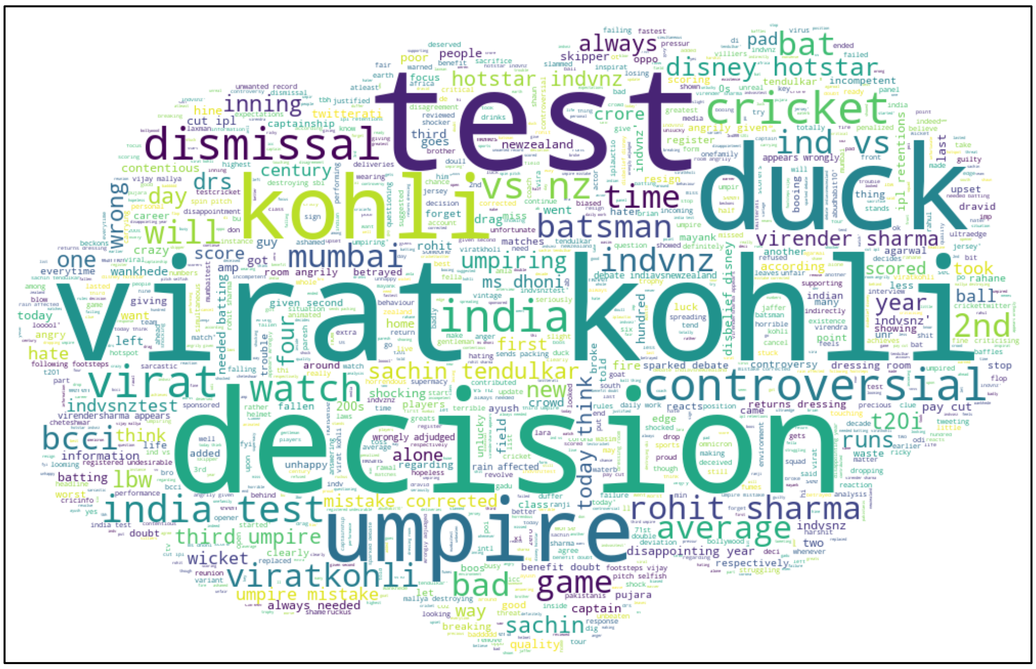
Then there are *STOPWORDS* also included from the wordcloud library, so that most of the stopwords (most common words used in our language in daily life, e.g., the, an, and, if, what, that, etc.) are removed from the data and these unimportant words don’t affect our analysis.

Moving on, I have created four wordclouds each for ‘All the tweets’, ‘Positive tweets’, ‘Negative tweets’ and ‘Neutral tweets’.

I will explain an example for the wordcloud here:



```
# Creating wordcloud for negative sentiment
create_wordcloud(tw_list_negative["text"].values)

Word Cloud Saved Successfully
```

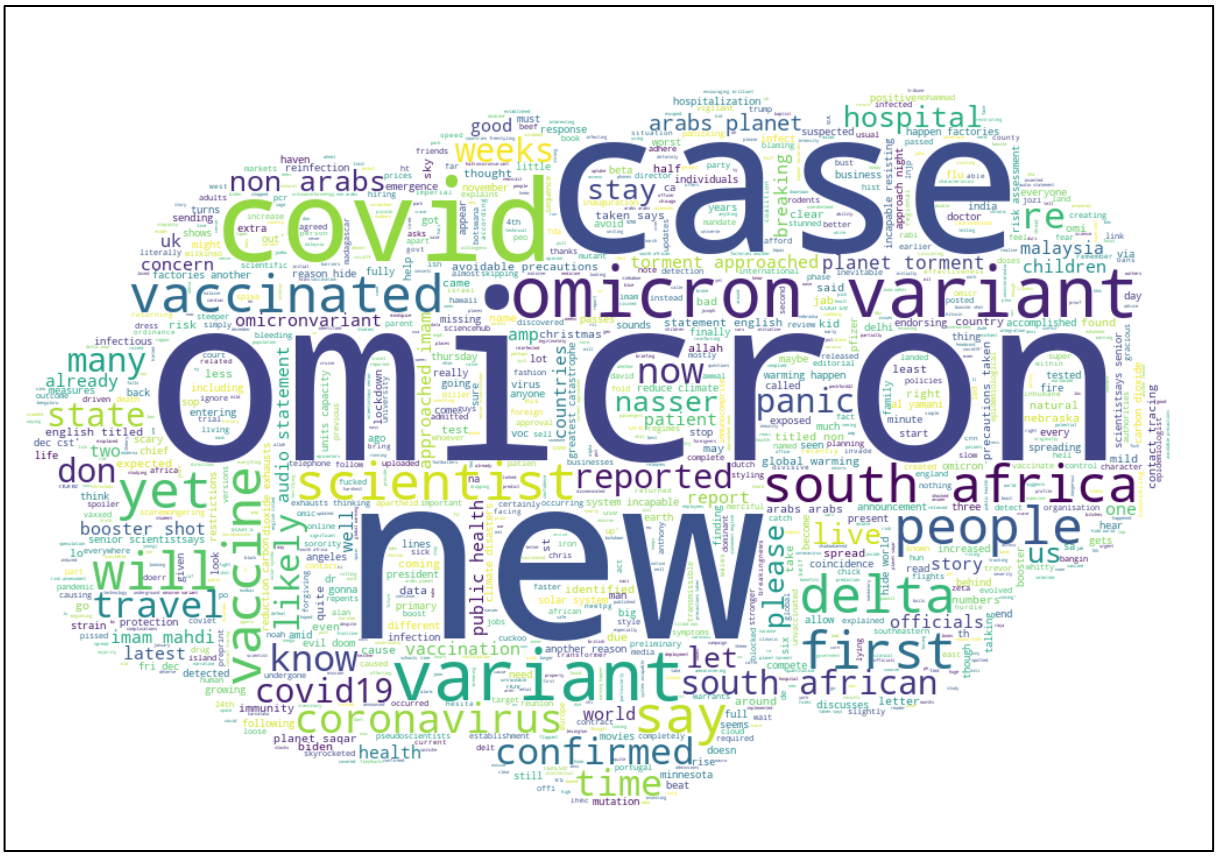


In this example, this wordcloud represents all the 'Negative tweets' for the keyword 'virat kohli' on the day this data was extracted.

This scenario is related to the Indian Cricket player and captain Virat Kohli, when he was given LBW out on the 1st ball and 0 runs by the umpire, which happens to be a wrong decision made by the umpire and went on to become a controversial topic.

We can clearly see the highlighted words in the wordcloud like 'test', 'duck', 'virat kohli', 'decision', 'umpire', 'controversial', etc. words which represent the negative side of this particular event and how the public reacted to it.

Another example:

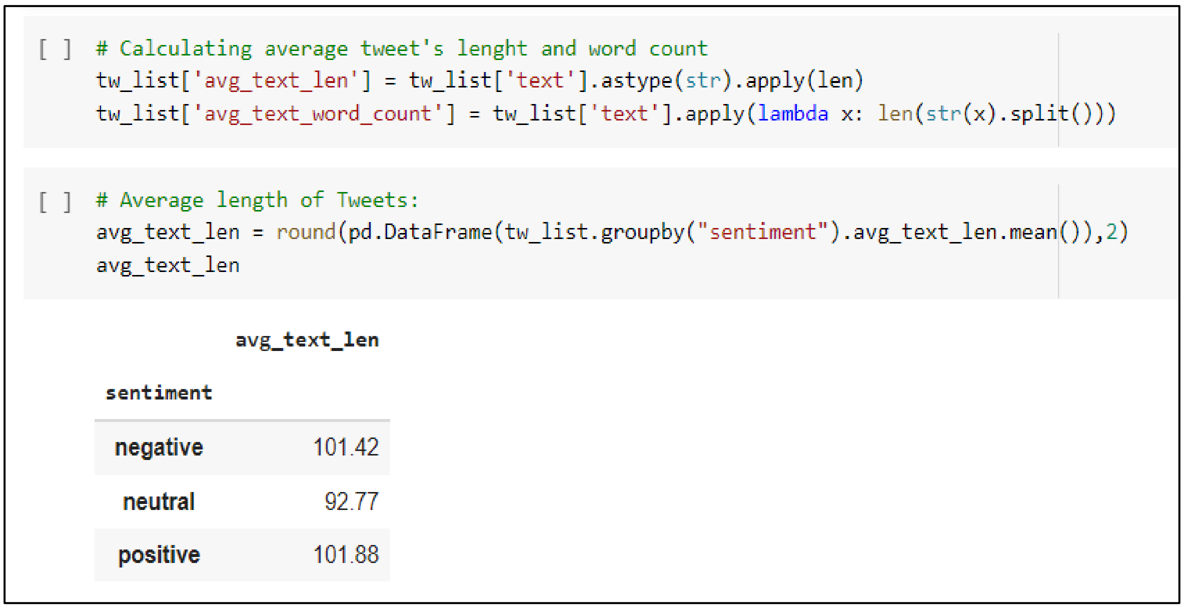


This is another wordcloud that was created by me while analysing the keyword 'omicron' which is a new variant of COVID-19 Corona Virus that was found in South Africa in November 2021 and was declared a variant of concern by the WHO soon after.

In this wordcloud we can clearly see that the words 'omicron', 'covid', 'cases', 'new', 'variant', 'south africa', 'covid19', 'coronavirus', 'confirmed', etc. are visible in large fonts as these were the most majorly used words to describe this particular event.

Thus, we can conclude that wordcloud is certainly a very effective way for describing the topic of our concern in an easy-to-understand way.

11. Average tweet length and word count:



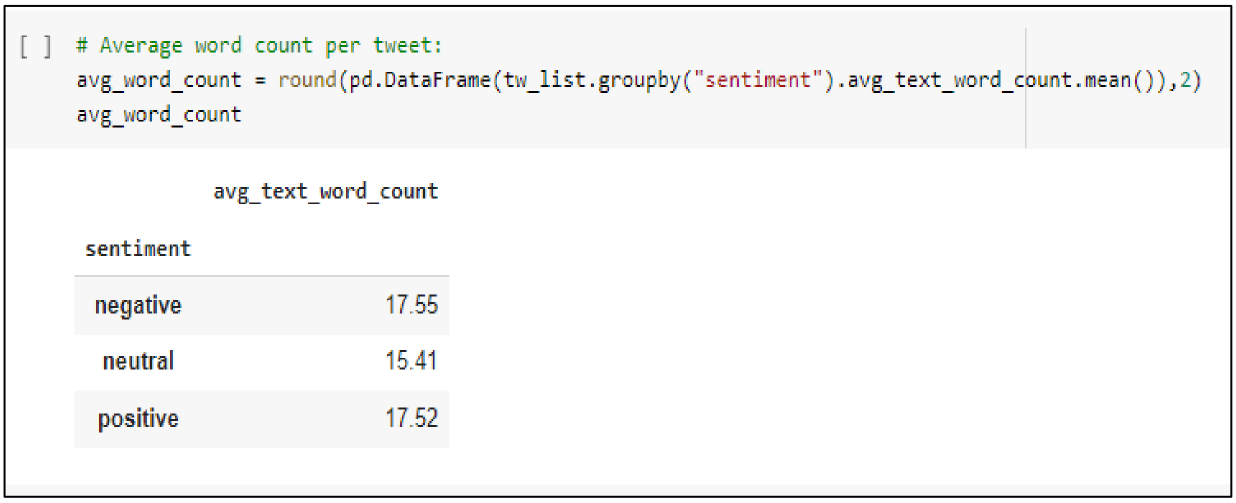
```
[ ] # Calculating average tweet's lenght and word count
    tw_list['avg_text_len'] = tw_list['text'].astype(str).apply(len)
    tw_list['avg_text_word_count'] = tw_list['text'].apply(lambda x: len(str(x).split()))
```

```
[ ] # Average length of Tweets:
    avg_text_len = round(pd.DataFrame(tw_list.groupby("sentiment").avg_text_len.mean()),2)
    avg_text_len
```

sentiment	avg_text_len
negative	101.42
neutral	92.77
positive	101.88

Here, I have calculated the ‘Average text length’ for all the tweets.

This is important to find out, as there can be instances where we could see any specific pattern that people tend to type longer tweets in positive or negative sentiment on any trending topic. This data could be useful to observe the behaviour and extremist reaction towards a specific topic.



```
[ ] # Average word count per tweet:
    avg_word_count = round(pd.DataFrame(tw_list.groupby("sentiment").avg_text_word_count.mean()),2)
    avg_word_count
```

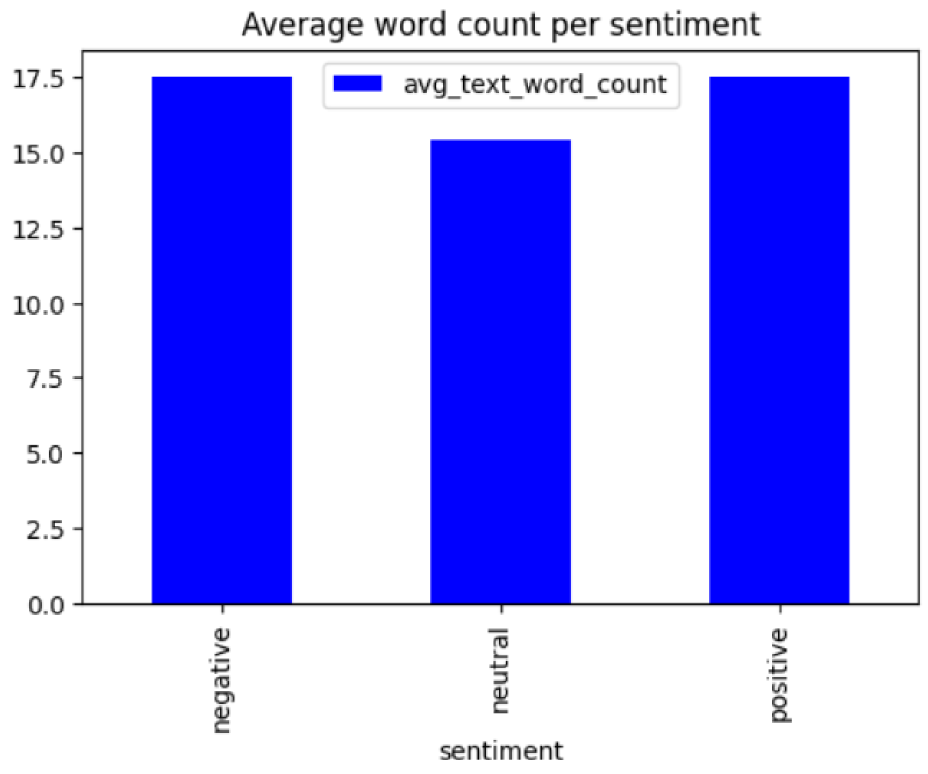
sentiment	avg_text_word_count
negative	17.55
neutral	15.41
positive	17.52

Similarly, the ‘Average word count’ is also a good indicator of the public feelings.

We can also visualize this data with the help of the following bar chart:

```
[ ]  # Plotting Bar chart:
     avg_word_count.plot(kind = 'bar', figsize = [6,4], color ='blue')
     plt.title('Average word count per sentiment')
```

```
Text(0.5, 1.0, 'Average word count per sentiment')
```



In this particular example, we can clearly say that there is no specific trend and both positive and negative tweets have around the same ‘average text length’ as well as same ‘average word count’.

Hence, we can conclude that as the previous analysis we can say that this topic was delt neutrally by most of the people as there is a constant high level of Neutral sentiment and almost negligible variation in Positive and Negative sentiments.

12. Applying Tokenization, Stemmer, removing Stopwords and CountVectorizer:

Definitions:

*Tokenization: Tokenization is the process of breaking text documents apart into those pieces. In text analytics, tokens are most frequently just words.

*Stemmer: Stemming is the process of reducing a word to its root form. This ensures variants of a word match during a search. For example, walking and walked can be stemmed to the same root word: walk.

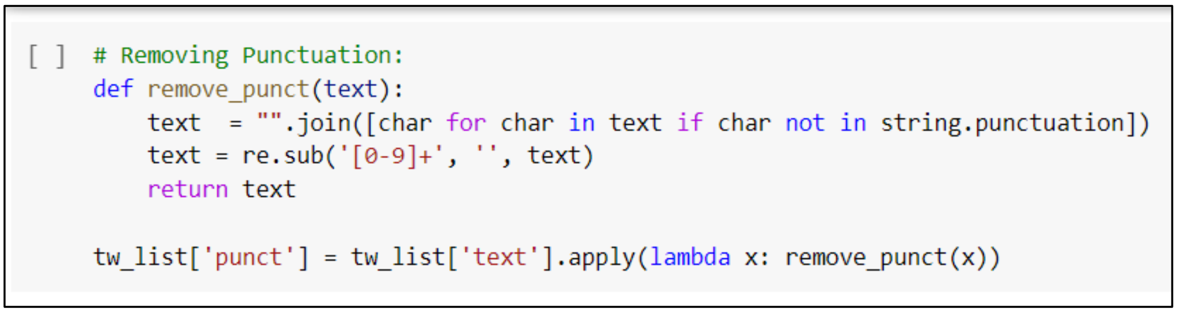
*Stopwords: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence.

*CountVectorizer: What is CountVectorizer in Python?

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

- **Removing Punctuations:**

Code:

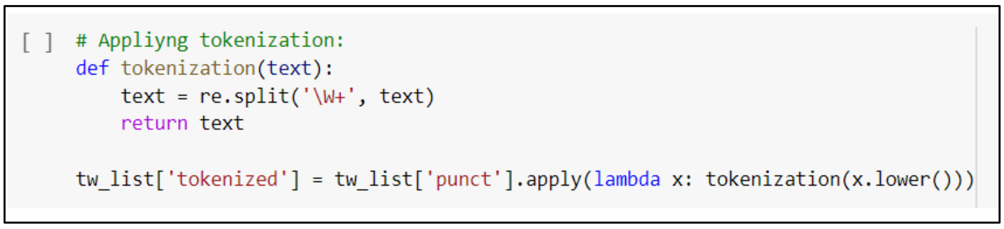


```
[ ]  # Removing Punctuation:
     def remove_punct(text):
         text  = "".join([char for char in text if char not in string.punctuation])
         text = re.sub('[0-9]+', '', text)
         return text

     tw_list['punct'] = tw_list['text'].apply(lambda x: remove_punct(x))
```

Here, I have added a column named *‘punct’* which comprises of all the cleaned tweets for any punctuation marks.

- **Applying Tokenization:**

Code:

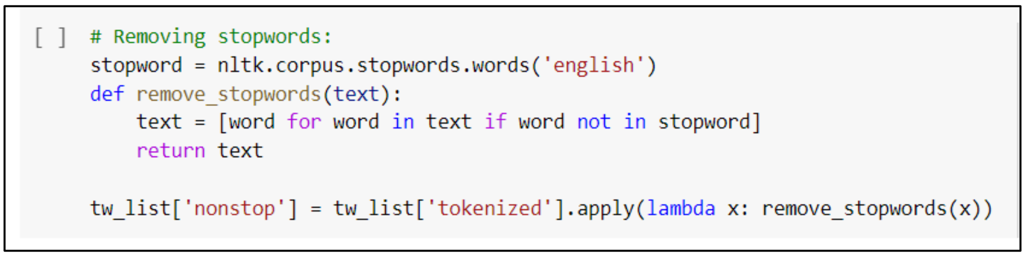


```
[ ]  # Appliyng tokenization:
     def tokenization(text):
         text = re.split('\W+', text)
         return text

     tw_list['tokenized'] = tw_list['punct'].apply(lambda x: tokenization(x.lower()))
```

Here, I have created a function for applying Tokenization to the tweets and breaking the sentences into individual words separated by commas (,).And then added a column named *‘tokenized’* to the dataframe.

- **Removing Stopwords:**

Code:

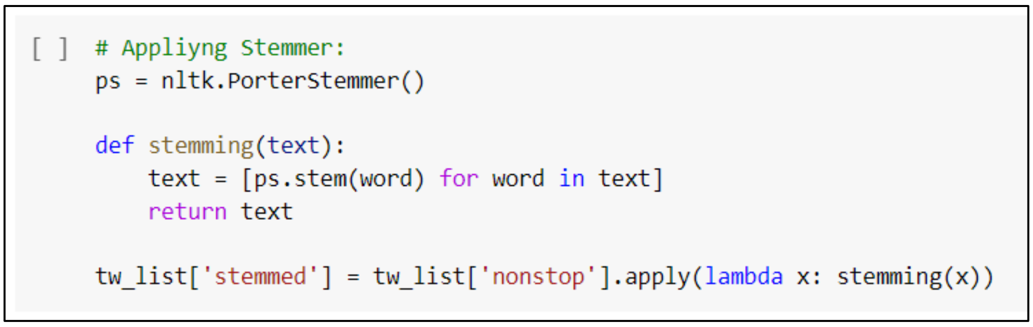


```
[ ]  # Removing stopwords:
     stopword = nltk.corpus.stopwords.words('english')
     def remove_stopwords(text):
         text = [word for word in text if word not in stopword]
         return text

     tw_list['nonstop'] = tw_list['tokenized'].apply(lambda x: remove_stopwords(x))
```

Here, I created a function to remove all the stopwords from our tweets and used this function on the *‘tokenized’* column to remove all the unwanted words from it and added a column *‘nonstop’* to the dataframe, which consists of all the individual tweet words after removing stopwords.

- **Applying Stemmer:**

Code:

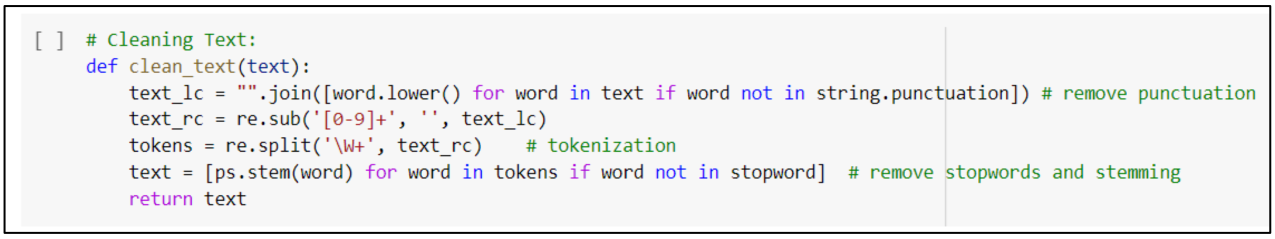


```
[ ]  # Appliyng Stemmer:
     ps = nltk.PorterStemmer()

     def stemming(text):
         text = [ps.stem(word) for word in text]
         return text

     tw_list['stemmed'] = tw_list['nonstop'].apply(lambda x: stemming(x))
```

Here, I added a new column to the dataframe in which all the words from *‘stemmed’* column are converted to their root form.

- **Function for cleaning text:**

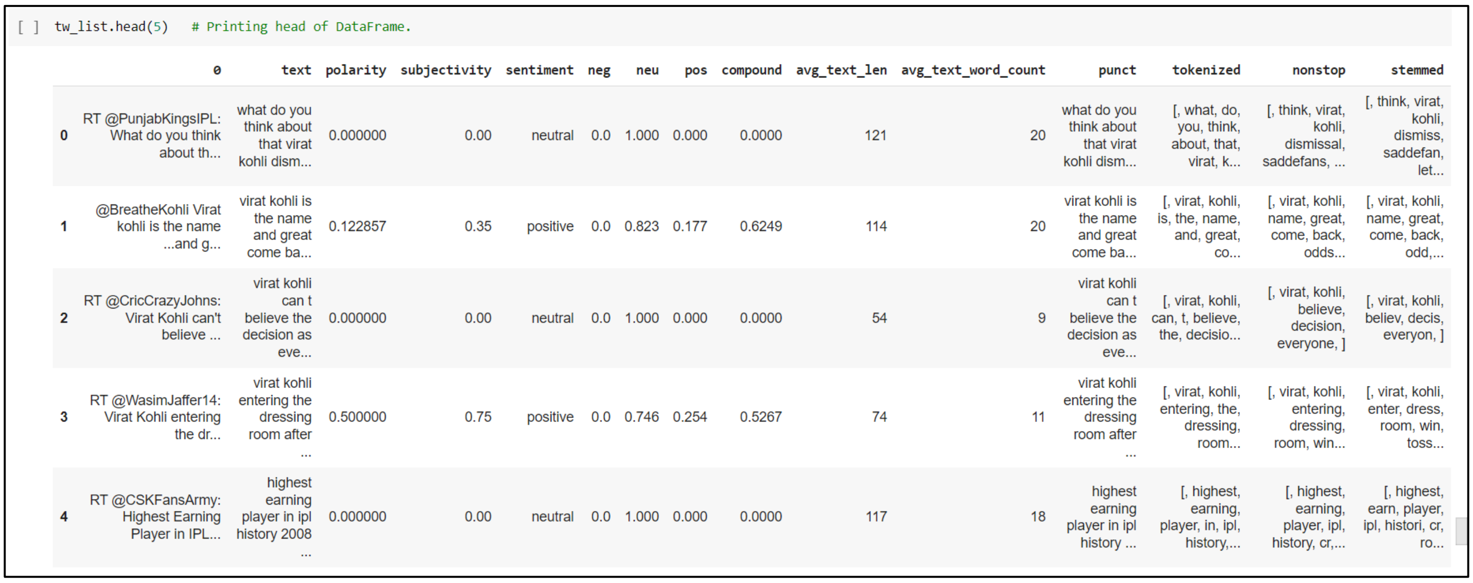
Code:



```
[ ]  # Cleaning Text:
     def clean_text(text):
         text_lc = "".join([word.lower() for word in text if word not in string.punctuation]) # remove punctuation
         text_rc = re.sub('[0-9]+', '', text_lc)
         tokens = re.split('\W+', text_rc)    # tokenization
         text = [ps.stem(word) for word in tokens if word not in stopword]  # remove stopwords and stemming
         return text
```

This function will be used in CountVectorizer.

- **Viewing Dataframe:**



```
[ ]  tw_list.head(5)   # Printing head of DataFrame.
```

	0	text	polarity	subjectivity	sentiment	neg	neu	pos	compound	avg_text_len	avg_text_word_count	punct	tokenized	nonstop	stemmed
0	RT @PunjabKingsIPL: What do you think about th...	what do you think about that virat kohli dism...	0.000000	0.00	neutral	0.0	1.000	0.000	0.0000	121	20	what do you think about that virat kohli dism...	[, what, do, you, think, about, that, virat, k...	[, think, virat, kohli, dismissal, saddefans, ...	[, think, virat, kohli, dismiss, saddefan, let...
1	@BreatheKohli Virat kohli is the name ...and g...	virat kohli is the name and great come ba...	0.122857	0.35	positive	0.0	0.823	0.177	0.6249	114	20	virat kohli is the name and great come ba...	[, virat, kohli, is, the, name, and, great, co...	[, virat, kohli, name, great, come, back, odds...	[, virat, kohli, name, great, come, back, odd,...
2	RT @CricCrazyJohns: Virat Kohli can't believe ...	virat kohli can t believe the decision as eve...	0.000000	0.00	neutral	0.0	1.000	0.000	0.0000	54	9	virat kohli can t believe the decision as eve...	[, virat, kohli, can, t, believe, the, decisio...	[, virat, kohli, believe, decision, everyone,]	[, virat, kohli, believ, decis, everyon,]
3	RT @WasimJaffer14: Virat Kohli entering the dr...	virat kohli entering the dressing room after ...	0.500000	0.75	positive	0.0	0.746	0.254	0.5267	74	11	virat kohli entering the dressing room after ...	[, virat, kohli, entering, the, dressing, room...	[, virat, kohli, entering, dressing, room, win...	[, virat, kohli, enter, dress, room, win, toss...
4	RT @CSKFansArmy: Highest Earning Player in IPL...	highest earning player in ipl history 2008 ...	0.000000	0.00	neutral	0.0	1.000	0.000	0.0000	117	18	highest earning player in ipl history ...	[, highest, earning, player, in, ipl, history,...	[, highest, earning, player, ipl, history, cr,...	[, highest, earn, player, ipl, histori, cr, ro...

This is the output of the head of our dataframe. Here we can see the columns *'punct'*, *'tokenized'*, *'nonstop'* and *'stemmed'* in the output.

For this example, view the 1st row of this output, here in the *'punct'* column we cannot see any punctuations of any type.

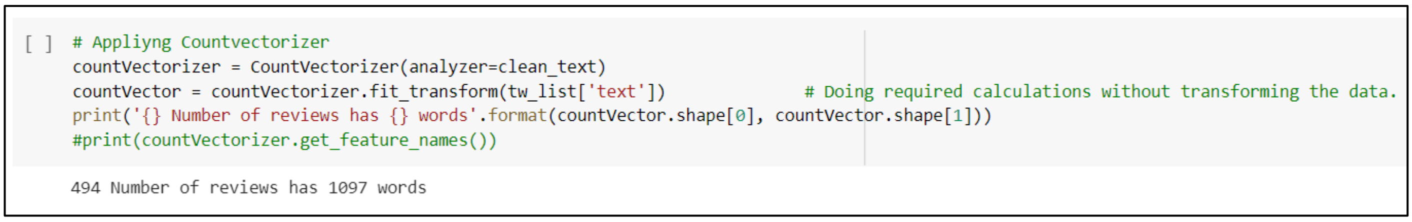
Next in the *'tokenized'* column we can see all the words from *'punct'* column separated by commas (,).

Moving forward, the words 'what', 'do', 'you', 'about', 'that' are removed from the *'nonstop'* column.

Similarly, words like 'dismissal' from *'nonstop'* column has been converted to its root form 'dismiss' in the *'stemmed'* column.

- **CountVectorizer:**

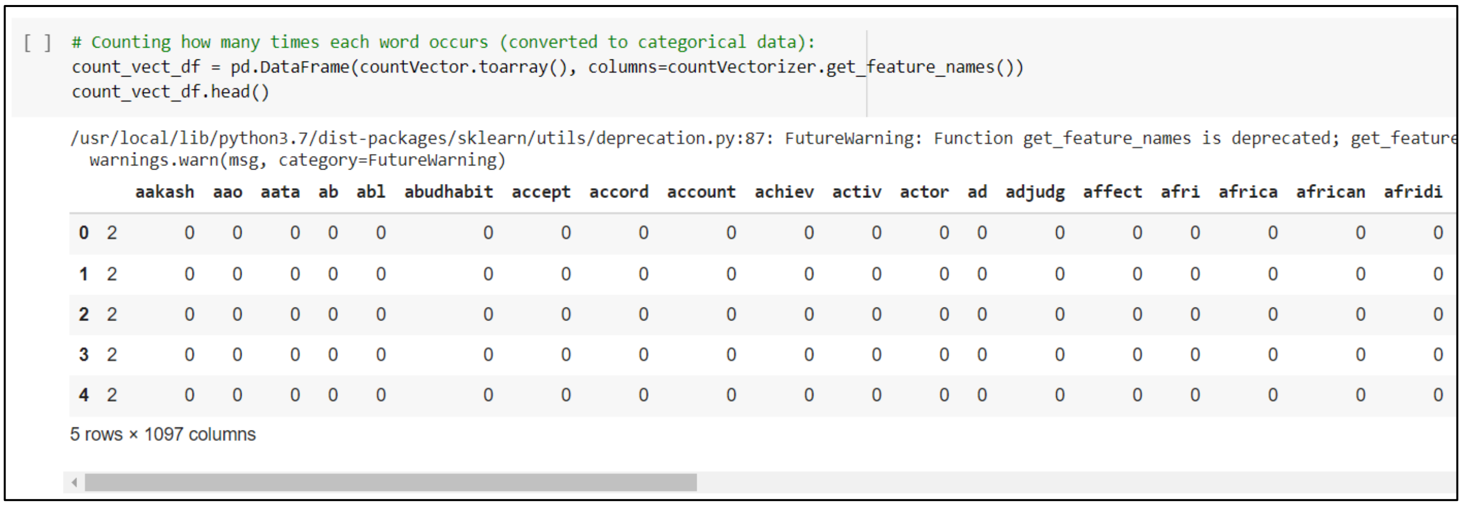
Code:



```
[ ] # Appliyng Countvectorizer
    countVectorizer = CountVectorizer(analyzer=clean_text)
    countVector = countVectorizer.fit_transform(tw_list['text'])          # Doing required calculations without transforming the data.
    print('{} Number of reviews has {} words'.format(countVector.shape[0], countVector.shape[1]))
    #print(countVectorizer.get_feature_names())

    494 Number of reviews has 1097 words
```

Here is a function to convert out tweet texts into categorical form as vectors according to the frequency of each word.

- **Viewing output of CountVectorizer:**



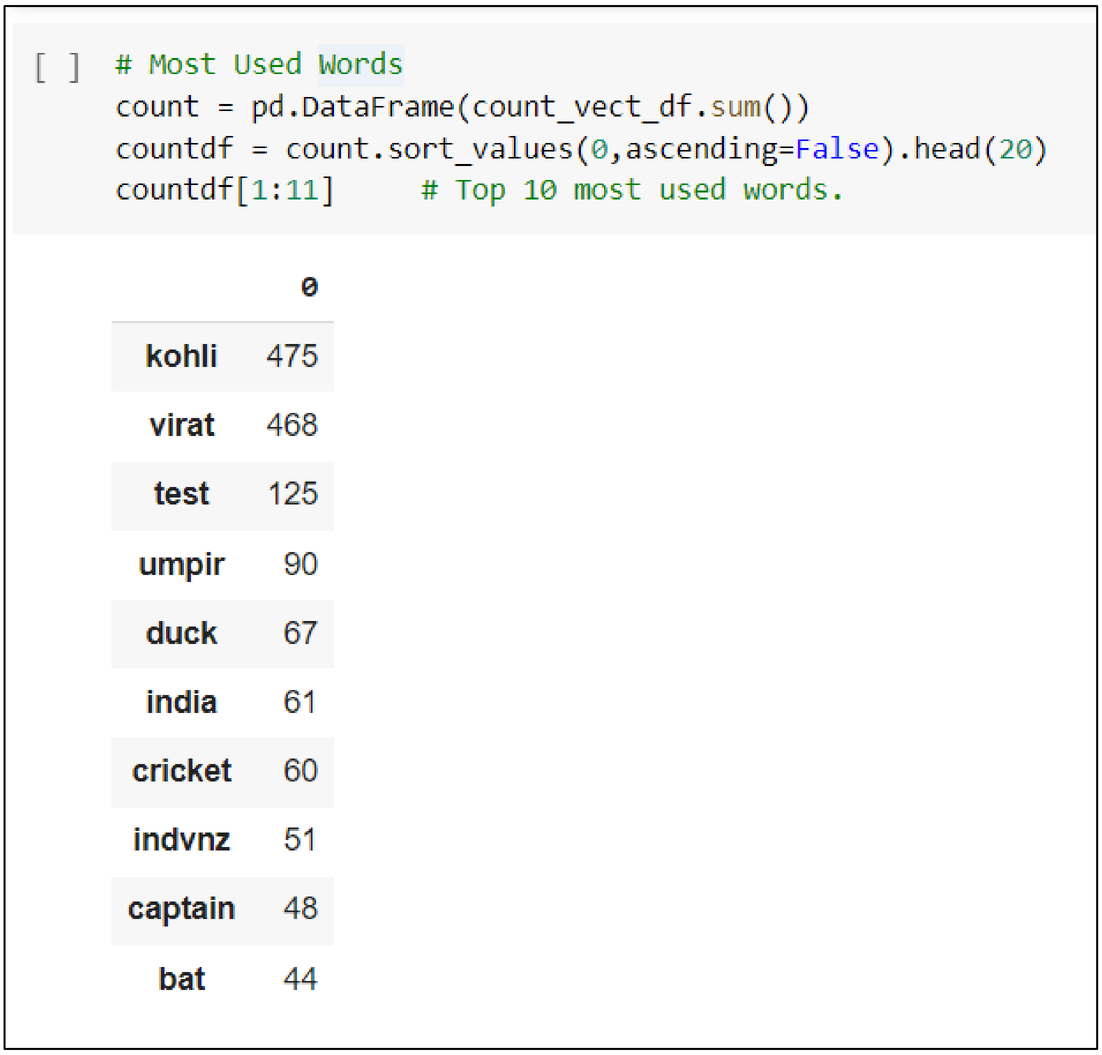
```
[ ] # Counting how many times each word occurs (converted to categorical data):
    count_vect_df = pd.DataFrame(countVector.toarray(), columns=countVectorizer.get_feature_names())
    count_vect_df.head()

    /usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature
      warnings.warn(msg, category=FutureWarning)
          aakash aao aata ab abl abudhabit accept accord account achiev activ actor ad adjudg affect afri africa african afridi
    0  2       0   0    0  0   0         0      0      0       0      0     0     0  0      0      0    0      0       0      0
    1  2       0   0    0  0   0         0      0      0       0      0     0     0  0      0      0    0      0       0      0
    2  2       0   0    0  0   0         0      0      0       0      0     0     0  0      0      0    0      0       0      0
    3  2       0   0    0  0   0         0      0      0       0      0     0     0  0      0      0    0      0       0      0
    4  2       0   0    0  0   0         0      0      0       0      0     0     0  0      0      0    0      0       0      0
    5 rows × 1097 columns
```

Here, we have now converted the countvector into a dataframe and then printed its head.

From this we can see that the function has taken all words found in all the tweets combined and arranged them from A to Z in alphabetical order.

Thereafter it has assigned 0 and 1 to all the words for each and every tweet. If the word is present in the tweet, then it will be given 1, else it will be assigned 0.

13. Most used words:



```
[ ]  # Most Used Words
     count = pd.DataFrame(count_vect_df.sum())
     countdf = count.sort_values(0,ascending=False).head(20)
     countdf[1:11]      # Top 10 most used words.
```

	0
kohli	475
virat	468
test	125
umpir	90
duck	67
india	61
cricket	60
indvnz	51
captain	48
bat	44

In this last step, I have used the dataframe of the CountVectorizer where all the values (0 and 1) assigned to each word were summed up and added to another dataframe along with the word.

Then this dataframe was sorted in descending order for getting all the words with high frequency at the top.

And then printed the head of top 10 words which occurred the maximum number of times in all the tweets from the final cleaned data.

This gives us the idea about which words actually define the topic of our interest at its root level.

❖ *Conclusion:*

The following are conclusions from this project:

- This code is dynamic in nature and can be used for the analysis of any topic/person from Twitter of the user's interest.

- This code gives detailed sentimental analysis of the chosen topic.

- There are thousands of Tweets posted every hour, so the keyword that the user wishes to analyze will be reviewed on the current scenario itself (events occurred in the recent 1 day to 1 week time span) as Twitter is a huge social media platform and trends change drastically.

  (If we wish to analyze any topic for a specific day or event, we can add an argument of date to the code.)

- Wordcloud in this analysis is an important aspect and proves to be a promising evaluation factor for giving conclusion on mass sentiments.

- We can use various types of graphs like Pie chart, Donut graph, Histogram, Bar chart and many more for visualizing a variety of data.

- Polarity of any Tweet and the type of words used in it can easily give us an idea of the sentiment behind writing it or even the feeling of the writer on any given topic.

- 'Average text length' and 'Average word count' can be used to find any certain mass trend towards the given topic.

❖ *Future Scope:*

The following can be said to be future scope for this project on Sentimental Data Analysis of Twitter:

- This code can be modified and made into a single defined function which will recall any other function whenever required.

- There is a chance of adding more than one Keyword for the analysis. Where the user could give more than one Keywords along with the number of tweets, they want to analyze and the program would give them all the outputs at once, along with a comparison between the selected Keywords and whether or not they have any correlation between them.

  For example, we can find correlation between keywords like ‘Virat Kohli’ and ‘IPL’ or even ‘Government’ and ‘Economic Policy’ or any other socio-political topic.

- We can take into consideration and add more factors to analyse the data like ‘Number of Re-Tweets’ and ‘Number of likes’ for the Tweets into consideration.

  This will definitely give us better understanding of what percentage of people support or oppose any given topic.

- There are variety of uses of this type of program:

  I. To know public sentiment on any specific topic (of social concern) like ‘Black lives matter’ or ‘Amazon Wildfires’ or ‘COVID-19 Pandemic’.

  II. In political analysis, to know the public’s opinion on any specific political party or political leader during or before elections. This could also be used to determine exit polls during elections.

  Etc.

❖ *References / Bibliography:*

- **Link:** https://en.wikipedia.org/wiki/Sentiment_analysis

- **Link:** https://getthematic.com/insights/sentiment-analysis/

- **Link:** https://textblob.readthedocs.io/en/dev/

- **Link:** https://en.wikipedia.org/wiki/Natural_Language_Toolkit#:~:text=The%20Natural%20Language%20Toolkit%2C%20or,in%20the%20Python%20programming%20language.

- **Link:** https://www.nltk.org/

- **Link:** https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/