Analyzing people's opinion toward coronavirus

Chenchen Mao, Yikun Wu, Keyang Li, Jiaxin Song, Qinwei Huang, Yang Gan,Jialiang Wang

Abstract

Since the breaking of COVID-19, the epidemic situation has become the most conspicuous thing in the world. Millions of people are infected and the trend of growth shows no sign of abating. Social distance, home quarantine order, remote office work, online course, lack of medical resources, unemployment, death… These kinds of words related to this worldwide epidemic occupy all kinds of social websites' hotpoint.

This report is aiming to COVID-19. how people think about the COVID-19, whether their opinions are varied from the deterioration of the situation, what thing has the most outstanding impact on people's attitudes to coronavirus, whether people in cities which are in worse situation tend to have worse judgement compared to the people in cities which are not as badly hit? We crawl tweets from Twitter and analyze these data. The answer to these questions above are given in this report.

We use twitter api to collect data from Twitter. The collected information is then input into a sentiment analysis model where data are labelled positive, negative and neutral. Twitter api is also used to specify the location and release time of tweets. The data are sorted by geography and time, then we are able to figure out people's attitudes towards coronavirus.

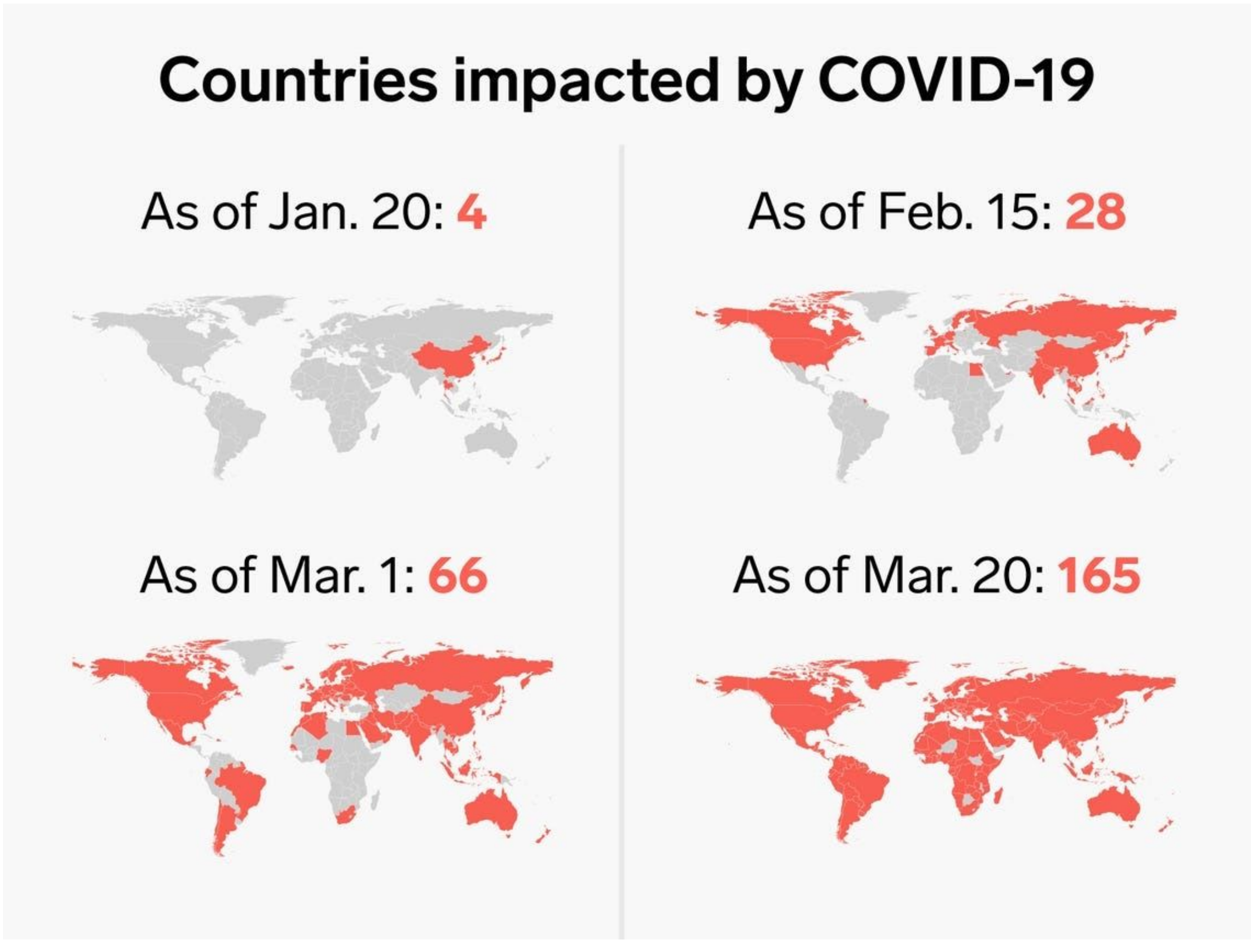
The attitudes of people can reflect to what extent the virus affects people's daily life. In some cases, it can also be used to predict the growing trend of the infected people. For an infectious disease that is so transmissible, the less it is taken seriously, the worse it will be. With that in mind, analyzing people's opinion towards coronavirus is meaningful.

Contents

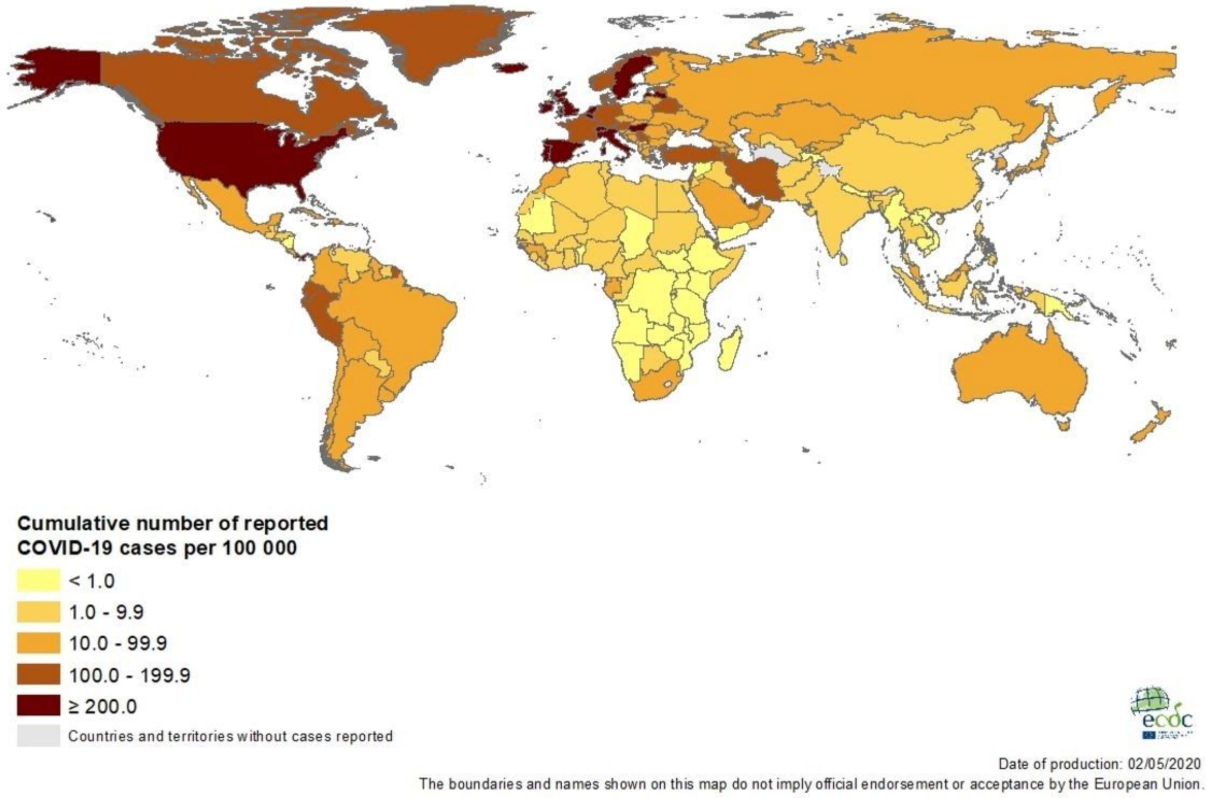
1. Introduction

1.1 COVID-19

If we had to sum up 2020 in one word, that word must be COVID-19. In January, COVID-19 hit China, although China has tried their best to tackle this epidemic and warn the whole world about the dangerousness of this virus, more and more countries are still affected and being hit hard. As time goes on, the trend of this epidemic shows no signs of abating and appears to be getting worse. As of May, the number of infected people has reached 3471884 and it has caused 244104 death cases. (https://www.worldometers.info/coronavirus/)



(https://www.businessinsider.com/map-tracks-novel-coronavirus-spread-in-countries-around-the-world-2020-3)


Cumulative number of reported COVID-19 cases per 100 000
< 1.0
1.0 - 9.9
10.0 - 99.9
100.0 - 199.9
≥ 200.0
Countries and territories without cases reported
ecdc
Date of production: 02/05/2020
The boundaries and names shown on this map do not imply official endorsement or acceptance by the European Union.

(https://www.ecdc.europa.eu/en/geographical-distribution-2019-ncov-cases)

On social media platforms, forums and new agencies, the epidemic has been occupying the most popular position. Thound of information about COVID-19 are released each day. Regarding this tough time people are facing now and the frequency of discussions going online, our team chose COVID-19 as our project topic.

1.2 Twitter

The data in our project all comes from Twitter. There are two reasons why we choose twitter as our data source. On the one hand, twitter is one of the most popular social media platforms with users from all over the world. The quantity and diversity of data are guaranteed. On the other hand, Twitter provides rich and powerful developer tools and apis, which makes the crawling of data easier.

Our goal is showing people's attitude as time goes and how location affects people's opinions. This report shows how our group implemented knowledge we learned in class to achieve the goal.

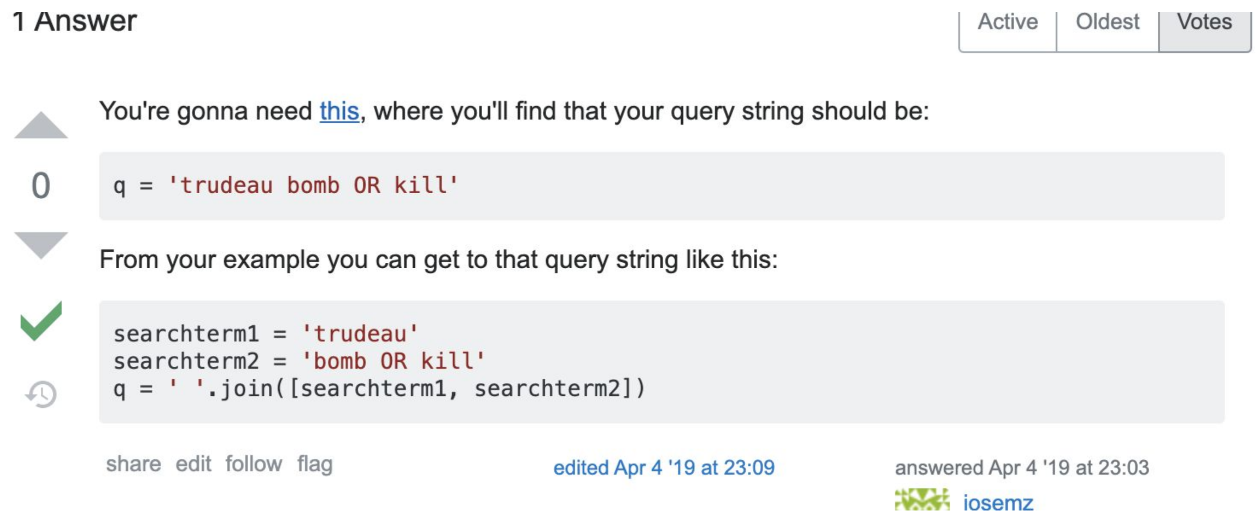
The following content will introduce our project in detail.

2. Collecting Data

2.1 Tweepy

When we got the task, the first thing that came to our mind was using Cookbook twitter since the last assignment that we did uses Cookbook twitter. But this approach did not work since we cannot find a way to apply location,keywords and time as filters at the same time. So we switched to use tweepy(api.search). We first tried to make several searches and checked the results. However, the result was not what we expected since it returned some non-english tweets. Therefore, we added a language filter to get only english tweets. Since we wanted to get all tweets about COVID-19 and there are different ways to call COVID-19. Some people call it coronavirus on twitter, and some use pandemic to refer to this outbreak. we came up with multiple keywords, such as pandemic, COVID-19, COVID and etc. As long as a tweet contains any keyword from the list, the tweet should be stored. So we googled how to achieve this goal and find an answer from stackoverflow.

```
keyword="coronavirus OR COVID-19 OR Pandemic OR epidemic OR outbreak OR COVID19 OR COVID 19 OR corona virus OR COVID "
```


1 Answer

Active Oldest Votes

0

You're gonna need this, where you'll find that your query string should be:

```
q = 'trudeau bomb OR kill'
```

From your example you can get to that query string like this:

```
searchterm1 = 'trudeau'
searchterm2 = 'bomb OR kill'
q = ' '.join([searchterm1, searchterm2])
```

share edit follow flag   edited Apr 4 '19 at 23:09   answered Apr 4 '19 at 23:03 iosemz

After finalizing keywords, We started to think about cities that we are going to analyze. Our first choice was `cities=["Denver","Los Angeles","Seattle","New York City", "Syracuse","Chicago", "Rico","San Francisco","Salt Lake City","Dallas","Atlanta","Miami"]`
In order to get tweets from each city, looping each city became our top solution. `for i in cities:`
After we run code, we realized that tweepy api search cannot filter tweets by city name. Therefore, we had to use each city's geocode. A new problem arose since we did not know the city's name from geocode so we created a dictionary to store the city name and geocode.

```
citydic={"Denver":"39.73034,-104.98958,50km","Los Angeles":"34.04015,-118.29757,50km","Seattle":"47.62405,-122.33279,50km","New York City":"40.74823,-73.98583,50km","Syracuse":"43.088947,
-76.154480,50km","Chicago":"41.85013,-87.68808,50km","Rico":"37.692095,
-108.030839,50km","San Francisco":"37.66873,-122.40438,50km","Salt Lake
City":"40.75186,-111.88270,50km","Dallas":"32.80963,-96.78387,50km","Atlanta":"
33.77729,-84.39015,50km","Miami":"25.76847,-80.28968,50km"}
```

We first looked at each city name from the list and used the name as a key to get the city geocode from the dictionary. `tweepy.Cursor(api.search, q=keyword,geo=citydic[i],lang="en",tweet_mode='extended',since = '2020-04-18',until='2020-04-19').items(1000)`

However, some results did not have full text because they were retweeted. Storing tweet text in two ways is the way to solve the problem.

```
                if 'retweeted_status' in dir(tweet):
                    dic["text"]=tweet.retweeted_status.full_text
                    print(tweet.retweeted_status.full_text,i)
                else:
                    dic["text"]=tweet.full_text
```

Though getting a full tweet text problem is solved, the program crushed when the rate limit reached. In order to solve this problem, we used try and exception.

```
       try:
            for tweet in tweepy.Cursor(api.search,
q=keyword,geo=citydic[i],lang="en",tweet_mode='extended',since =
'2020-04-18',until='2020-04-19').items(1000):

                dic = {}
                if 'retweeted_status' in dir(tweet):
                    dic["text"]=tweet.retweeted_status.full_text
                    print(tweet.retweeted_status.full_text,i)
                else:
                    dic["text"]=tweet.full_text
                    print(tweet.full_text,i)
                print(tweet.created_at)
                dic["created_at"]=tweet.created_at
                dic["location"]=i
                result=result+[dic]

            break
        except tweepy.error.TweepError as e:
            print(str(e))
```

After finishing all the preparation,we started collecting data and analyzing data. But we realized that we only collected seven days' data due we used api.search.Thus, we abandoned tweepy and used another way to collect data.

2.2 GetOldTweets

Using GetOldTweets allowed us to collect data before seven days. We added all the conditions into GetOldTweets.

```
        tweetCriteria = got.manager.TweetCriteria().setQuerySearch(keyword)\
                                                .setSince(x)\
                                                .setUntil(y)\
                                                .setMaxTweets(120)\
                                                .setNear(i)\
                                                .setLang("en")
```

There were still two problems that bothered us. The first problem was that we could not get a lot of data. When GetOldTweets had a rate limit problem, it would terminate. Hence, we had to modify the program by makeing program sleep instead terminate.

```
        except Exception as e:
            print("test")
            #print("An error occured during an HTTP request:", str(e))
            #print("Try to open in browser: https://twitter.com/search?q=%s&src=typd" %
urllib.parse.quote(urlGetData))
            print('Encountered 429 Error (Rate Limit Exceeded)', file=sys.stderr)
            print("Retrying in 15 minutes...ZzZ...", file=sys.stderr)
            sys.stderr.flush()
            #made change here to deal with rate limit exceeded probelm
            #make program sleep 6 mintues
            time.sleep(60*6)
            print('...ZzZ...Awake now and trying again.', file=sys.stderr)
            return 2
```

The second problem was json = TweetManager.getJsonResponse return int object sometime, which would cause a crush problem since we expected to get a string object. We finally used try and except to solve this problem.

```
               json = TweetManager.getJsonResponse(tweetCriteria, refreshCursor, cookieJar,
proxy, user_agent, debug=debug)
               try:#sometime json is not string we have to handle this exception
                   tmp=json['items_html']
                   if len(tmp.strip()) == 0:
                       break
               except:
                   #print(json['items_html'])
                   break
```

Besides, we realized all the cities we chose are big cities, which may not be subjective enough to show the result. We added Syracuse and Rico into our city list.

```
cities=["Denver,United States","Los Angeles,United States","Seattle,United
States","New York City,United States", "Syracuse,United States","Chicago,United
States","San Francisco,United States","Salt Lake City,United States","Dallas,United
States","Atlanta,United States","Miami,United States","Rico,United States"]
```

Therefore, we got around 70,000 tweets from different cities and dates.

3. Twitter Sentiment Analysis using Python

Sentiment Analysis, also called opinion mining or emotion AI, is the process of determining whether a piece of writing is positive, negative, or neutral. It is widely used when predicting people's attitude towards some hot topic. In our project, we are interested in getting people's attitude towards COVID-19. Also, we want to get the information about how their attitude changed with timeline or location.

After getting tweets from different locations and time, we tried to analyze people's attitudes hidden in those tweets. There are a lot of packages that could be used in analysis like NLTK and TextBlob.

In the step of extracting sentiment, we tried 2 different ways:

1. Lexicon-based methods

2. Machine Learning-based methods.

3.1 Machine Learning-based Methods

At first, even though this way is not suitable, we wanted to utilize the NLTK package to implement our machine learning-based classifier model.

For the training set, we used a downloadable training set which is twitter_samples in NLTK package. It contains 5000 tweets with negative sentiments, 5000 tweets with positive sentiments and 20000 tweets with no sentiments. All tweets of which were all labeled as positive or negative, depending on the content. Then we should also use a pre-model to accelerate the training of the final model.

With those labelled tweets, before inputting it as training data, we need to make a clean. Words are the most important part. However, when it comes to things like punctuation, you cannot get the sentiment from punctuation. Therefore, punctuation does not matter to Sentiment Analysis. Moreover, tweet components like images, videos, URLs, usernames, emojis, etc. do not contribute to the polarity (whether it is positive or negative) of the tweet. We expect to find a way to add those images into sentiment analysis in the future.

Here is the clean function:

```
def remove_noise(tweet_tokens, stop_words = ()):
   cleaned_tokens = []

   for token, tag in pos_tag(tweet_tokens):
       token = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\
                      '(?:%[0-9a-fA-F][0-9a-fA-F]))+','', token)
       token = re.sub("(@[A-Za-z0-9_]+)","", token)
```

```
        if tag.startswith("NN"):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'

        lemmatizer = WordNetLemmatizer()
        token = lemmatizer.lemmatize(token, pos)

        if len(token) > 0 and token not in string.punctuation and token.lower() not
in stop_words:
            cleaned_tokens.append(token.lower())
    return cleaned_tokens
```

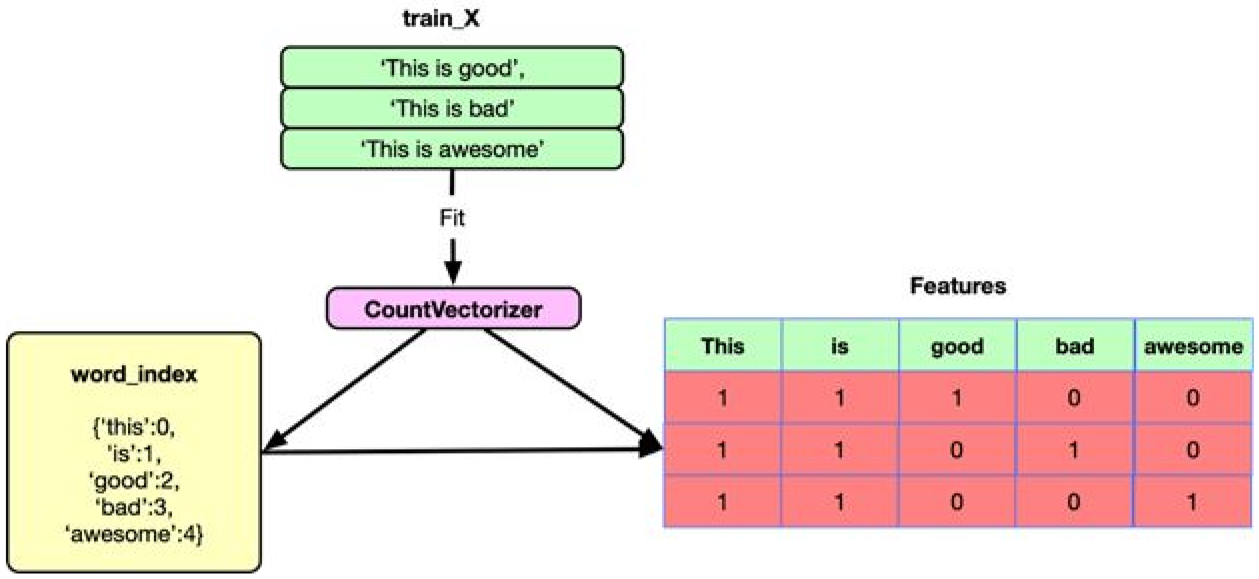
Words have different forms—for instance, “ran”, “runs”, and “running” are various forms of the same verb, “run”. Depending on the requirement of your analysis, all of these versions may need to be converted to the same form, “run”. Normalization in NLP is the process of converting a word to its canonical form.

Normalization helps group together words with the same meaning but different forms. Without normalization, “ran”, “runs”, and “running” would be treated as different words, even though you may want them to be treated as the same word. Stemming is a process of removing affixes from a word. Stemming, working with only simple verb forms, is a heuristic process that removes the ends of words.

One more important point is determining word density. The most basic form of analysis on textual data is to take out the word frequency. A single tweet is too small of an entity to find out the distribution of words, hence, the analysis of the frequency of words would be done on all positive tweets. We used a function to count the frequency of all words:

```
def get_all_words(cleaned_tokens_list):
    for tokens in cleaned_tokens_list:
        for token in tokens:
            yield token
```

After the training set is done, we should start to build a classifier model. Our classifier model is Naive Bayes Classifier which is the same as the classifier we used in the second case (TextBlob part). First, a vocabulary will be needed to store all the words resident in our training data set. Then, read the tweet text part (cleaned version) and compare every word in tweet with vocabulary. Sometimes words in a tweet might not be listed in the vocabulary. So, we have to give them a label to present whether they are in the list. Next, we need to build a feature vector for each word.



This	is	good	bad	awesome
1	1	1	0	0
1	1	0	1	0
1	1	0	0	1

Words to vectors

Finally, after all the words being translated into vectors, we should call the Naïve Bayes Classifier to build the model with a training set.

However, because the dataset of our project is huge (about 65000). If we use this way to analyze those tweets, it will take a considerable time for our hardware to not be good enough. Then, we start to try the second way.

3.2 Lexicon-based Methods

Lexicon based methods define a list of positive and negative words, with a valence — (eg 'nice': +2, 'good': +1, 'terrible': -1.5 etc). The algorithm looks up a text to find all known words. It then combines their individual results by summing or averaging. Some extensions can check some grammatical rules, like negation or sentiment modifier (like the word "but", which weights sentiment values in text differently, to emphasize the end of text).

In this method, we utilized TectBlob and its default corpus. However, there are other libraries like NLTK and spaCy. Why should we choose TextBlob? NLTK and spaCy Library are good libraries for performing Natural Language Processing tasks. But when it comes to NLU (Natural Language Understanding) which is a subset of NLP we need to work on unstructured data.

Both NLTK and spaCy are not able to perform in this scenario and thus TextBlob library comes under spotlight. TextBlob is designed to handle both structured and unstructured forms of data.

In TextBlob these aspects are vital evaluation standard:

1. Polarity: determined by phase of emotions expressed in the analyzed sentence. It ranges from -1 to 1 and can be divided into 3 categories: Positive, Negative, Neutral.

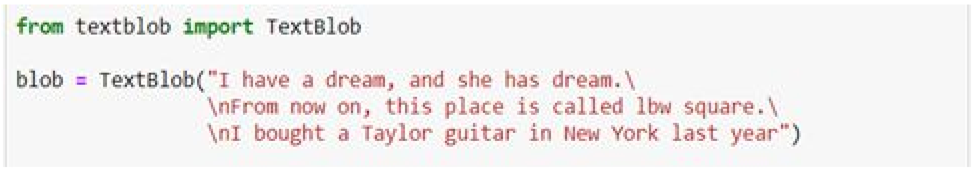
2. Subjective: check whether the analyzed sentence is describing features or opinions.

3. Intensity: it defines the strength of emotional expression in a checked sentence.

3.3 Sentiment Analysis by TextBlob

In our project, we mainly used TextBlob in 5 ways: Part-of-Speech Tagging & Filter Words by Tags, words inflection and lemmatization, generating N-grams Lists, sentiment analysis and spelling correction.

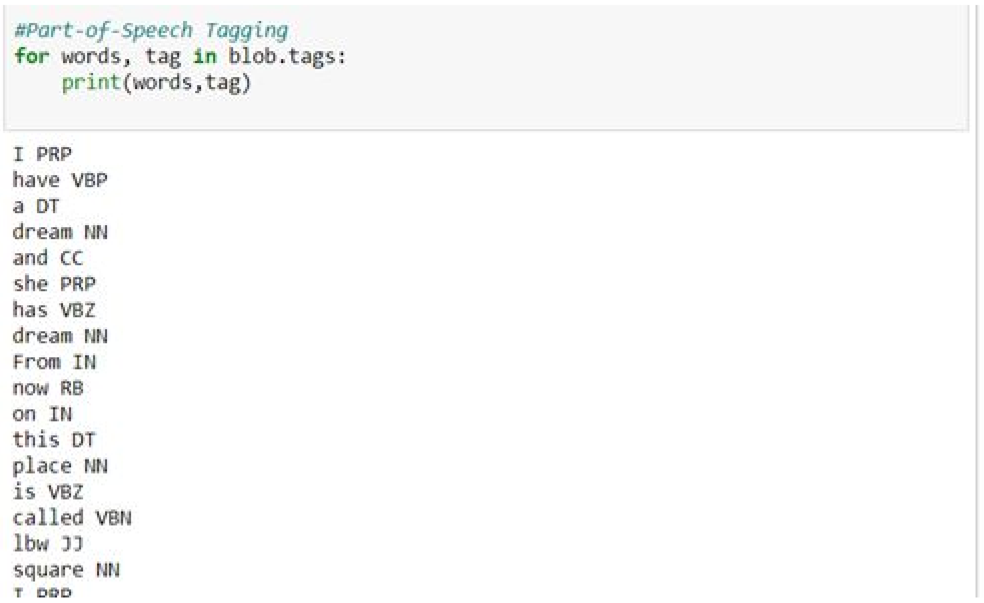
Before handling the data, we need to give the input to TextBlob, and it will help us divide the sentence into tokens, which shows in the figure:



```
from textblob import TextBlob

blob = TextBlob("I have a dream, and she has dream.\
                \nFrom now on, this place is called lbw square.\
                \nI bought a Taylor guitar in New York last year")
```

3.3.1 Part-of-Speech Tagging & Filter Words by Tags

Stop words are very common in English, such as a, an, and, or, of, at, the, and so on. These words carry extremely limited substantive information. Therefore, what we need to do is to remove the stop words from the text during the NLP analysis. The advantage of this is that we reduce our vocabulary and thus reduce the dimension of our eigenvectors.

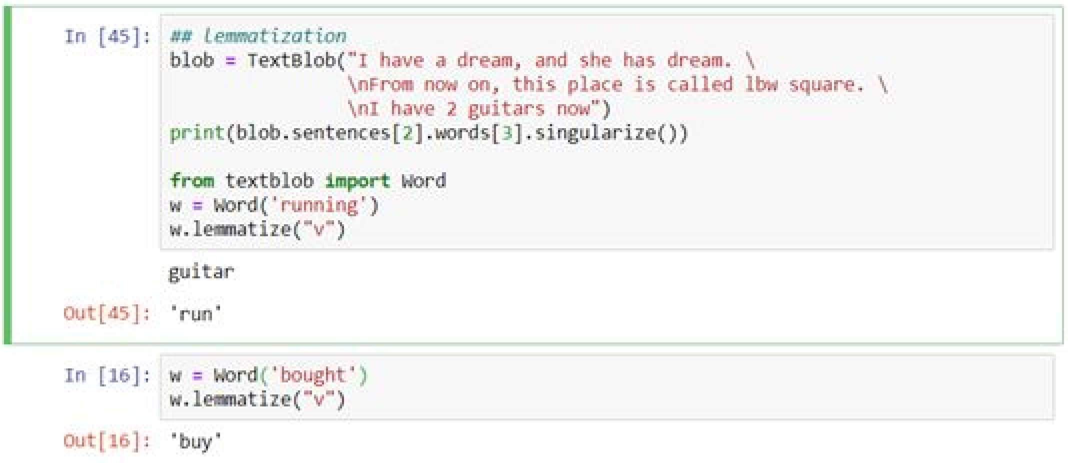


```
#Part-of-Speech Tagging
for words, tag in blob.tags:
    print(words,tag)
```

```
I PRP
have VBP
a DT
dream NN
and CC
she PRP
has VBZ
dream NN
From IN
now RB
on IN
this DT
place NN
is VBZ
called VBN
lbw JJ
square NN
I PRP
```

Method "tag" in TextBlob helps us identify each word by its part-of-speech. This method returns the type of input words. For example, "NN" means noun, "PRP" means pronoun, "DT" means determiner including "this" and "that".

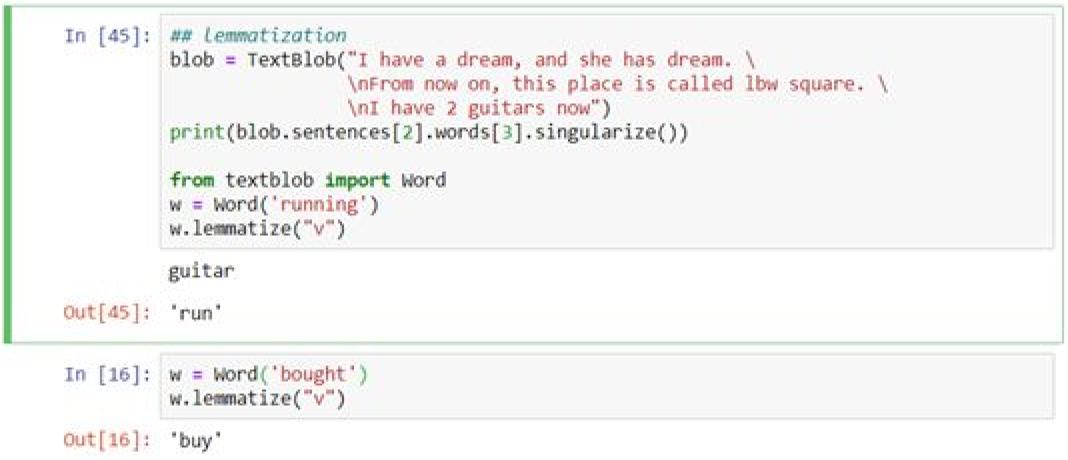
Before doing NLP tasks, we need to filter the input by the word type we need to improve performance and accuracy.



```
In [45]: ## Lemmatization
         blob = TextBlob("I have a dream, and she has dream. \
                         \nFrom now on, this place is called lbw square. \
                         \nI have 2 guitars now")
         print(blob.sentences[2].words[3].singularize())

         from textblob import Word
         w = Word('running')
         w.lemmatize("v")

         guitar

Out[45]: 'run'

In [16]: w = Word('bought')
         w.lemmatize("v")

Out[16]: 'buy'
```

3.3.2 Words Inflection and Lemmatization

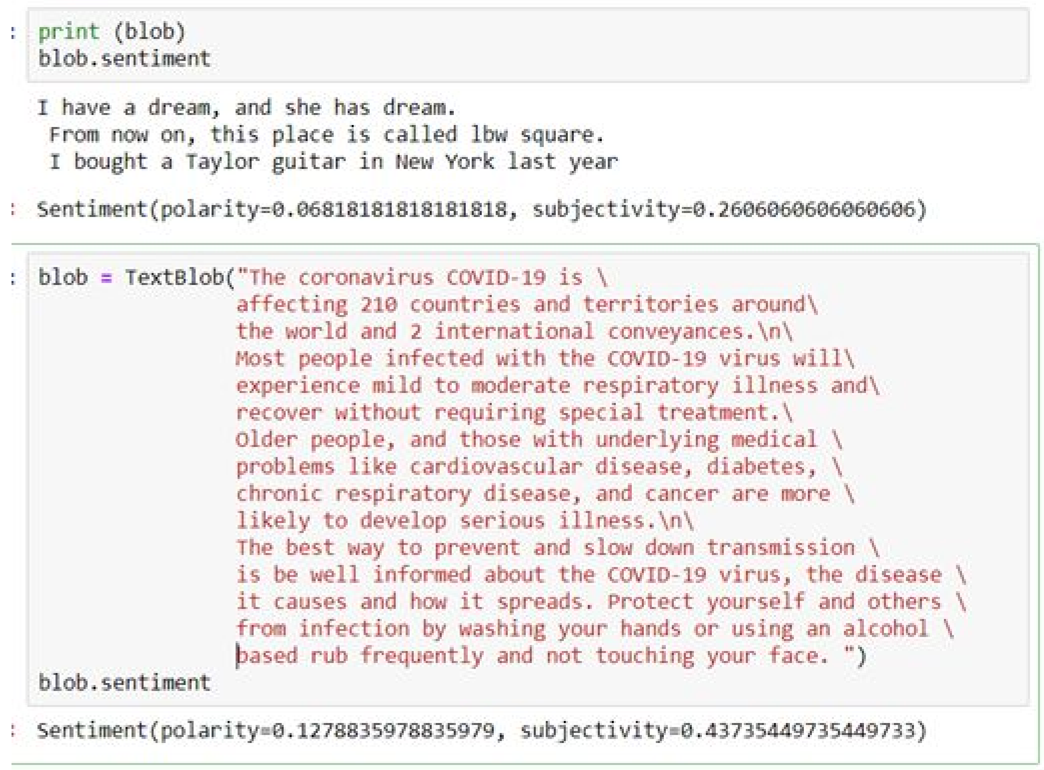
In the figure above, we can find an issue, same as the verb, “have” and “has” are marked differently by TextBlob, which are “VBP” and “VBZ”. Also, Singular and plural nouns also affect our analysis, so we use lemmatization in TextBlob to finish this step, as the following figure:



```
In [45]: ## Lemmatization
         blob = TextBlob("I have a dream, and she has dream. \
                         \nFrom now on, this place is called lbw square. \
                         \nI have 2 guitars now")
         print(blob.sentences[2].words[3].singularize())

         from textblob import Word
         w = Word('running')
         w.lemmatize("v")

         guitar

Out[45]: 'run'

In [16]: w = Word('bought')
         w.lemmatize("v")

Out[16]: 'buy'
```

3.3.3 Sentiment Analysis

Sentiment method helps calculate the value of polarity and subjectivity, which describes whether the input sentence is positive or negative and public opinion or fact.



```
: print (blob)
  blob.sentiment

  I have a dream, and she has dream.
   From now on, this place is called lbw square.
   I bought a Taylor guitar in New York last year

: Sentiment(polarity=0.06818181818181818, subjectivity=0.2606060606060606)

: blob = TextBlob("The coronavirus COVID-19 is \
                  affecting 210 countries and territories around\
                  the world and 2 international conveyances.\n\
                  Most people infected with the COVID-19 virus will\
                  experience mild to moderate respiratory illness and\
                  recover without requiring special treatment.\
                  Older people, and those with underlying medical \
                  problems like cardiovascular disease, diabetes, \
                  chronic respiratory disease, and cancer are more \
                  likely to develop serious illness.\n\
                  The best way to prevent and slow down transmission \
                  is be well informed about the COVID-19 virus, the disease \
                  it causes and how it spreads. Protect yourself and others \
                  from infection by washing your hands or using an alcohol \
                  based rub frequently and not touching your face. ")
  blob.sentiment

: Sentiment(polarity=0.1278835978835979, subjectivity=0.43735449735449733)
```

3.3.4 N-gram Implementation

However, this function has its own shortcomings. For example, the phrase "work hard" will be recognized as negative sentiment only because of the word "hard".

It is true that "hard" is a negative word in many circumstances, but it is neutral when it goes after the word "work". If the probability of "hard" appears after "work" is a, then we need to remove this part.

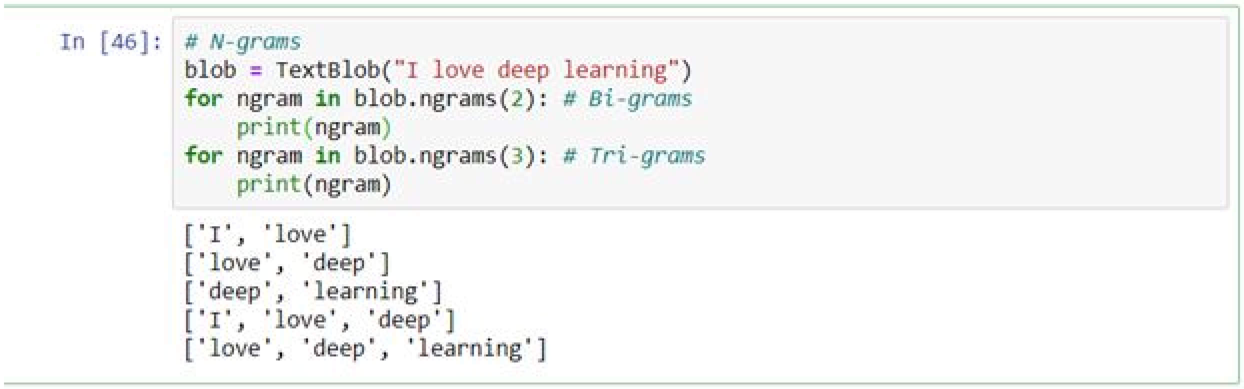
Here comes to the concept of N-gram.

The N-gram Model is a Language Model, which is a probabilistic discriminant Model. Its input is a sentence (the sequence of words), and its output is the probability of the sentence, namely the joint probability of these words.

For example, we use N-gram to analyze the sentence "I love ice coffee", we can think in the other way: If we have the input words {I, love, ice, coffee}, what is the possibility that the input is "I love ice coffee"? (In this circumstance, we only consider Bi-gram, which means the probability of word A's appearance only depends on the word before A)

So the formula comes out:

$P\{\text{I love ice coffee}\} = P\{I\}P\{love|I\}P\{ice|love\}P\{ice|coffee\}$



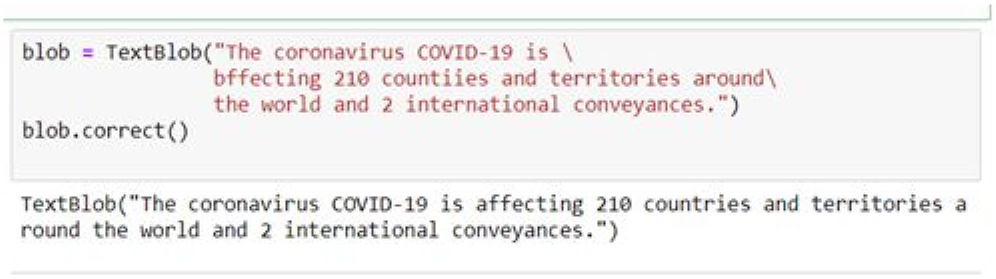
```
In [46]: # N-grams
         blob = TextBlob("I love deep learning")
         for ngram in blob.ngrams(2): # Bi-grams
             print(ngram)
         for ngram in blob.ngrams(3): # Tri-grams
             print(ngram)

         ['I', 'love']
         ['love', 'deep']
         ['deep', 'learning']
         ['I', 'love', 'deep']
         ['love', 'deep', 'learning']
```

Although N-grams model can let us better to split out have better semantic identifier, and then let us do further textual analysis, but the downside is also obvious, that is using N-grams model may make our vocabulary into exponential growth, and not all the Bigram contain useful information, and in this situation even in Tri-gram or Quad-gram contain more separate characters such as N-grams model will be more serious.

In this project, we only used Bi-gram.

3.3.5 Spelling Collection

Spelling collection is a small part of data cleaning, but it is important since it somewhat influences the result of our analysis because our target is tweets. For convenience, users always use abbreviations or even misspellings in their tweets, so that we need to firstly clean the data before doing any analysis.



```
blob = TextBlob("The coronavirus COVID-19 is \
                 bffecting 210 countiies and territories around\
                 the world and 2 international conveyances.")
blob.correct()

TextBlob("The coronavirus COVID-19 is affecting 210 countries and territories a
round the world and 2 international conveyances.")
```

There is another case that the user misspelled A to B but B is also fit in his tweet, even if it causes ambiguity. This issue can be fixed by the N-gram model which has been introduced in 4.

3.4 Utilizing TextBlob when Collecting Tweets

In our project, we call package TextBlob when getting tweets data. Therefore, sentiment analysis can be done when data collection completes. This method saves us a lot of time in doing sentiment analysis.

First, to get the tweet object, we should extract the text part of the tweet. Then, clean the full text data with clean function:

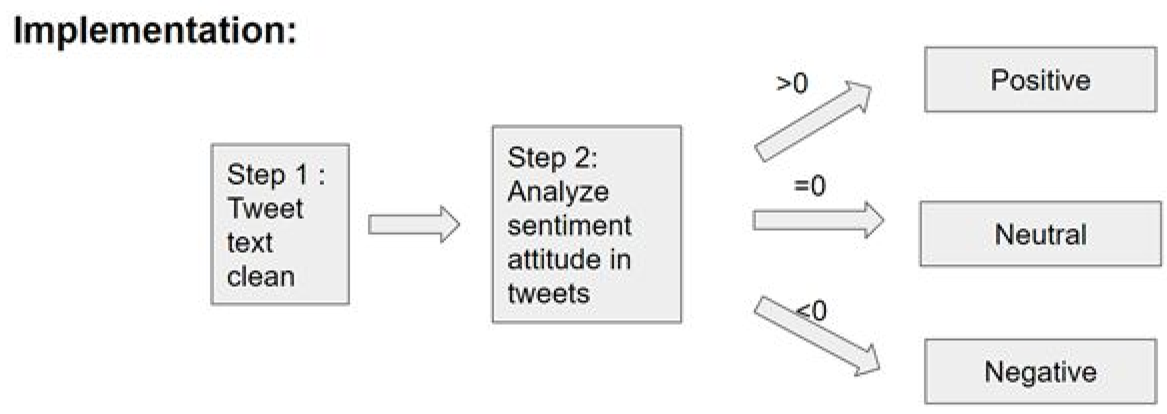
```
def clean_tweet(tweet):
   '''
   Utility function to clean tweet text by removing links, special characters
   using simple regex statements.
   '''
   return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ",
tweet).split())
```

After cleaning the symbols, we can call the function TextBlob and use the cleaned text data as input.

```
#anaylze opinion
analysis = TextBlob(clean_tweet(j.text))
```

The result returned by TextBlob is a textblob object which contains the properties mentioned before.

Next, we can call the polarity under sentiment property which is an integer variable to get the score of analyzed tweet text. We decided to divide the sentiment of tweet into 3 categories: positive, negative and neutral according to their sentiment score given by textblob.

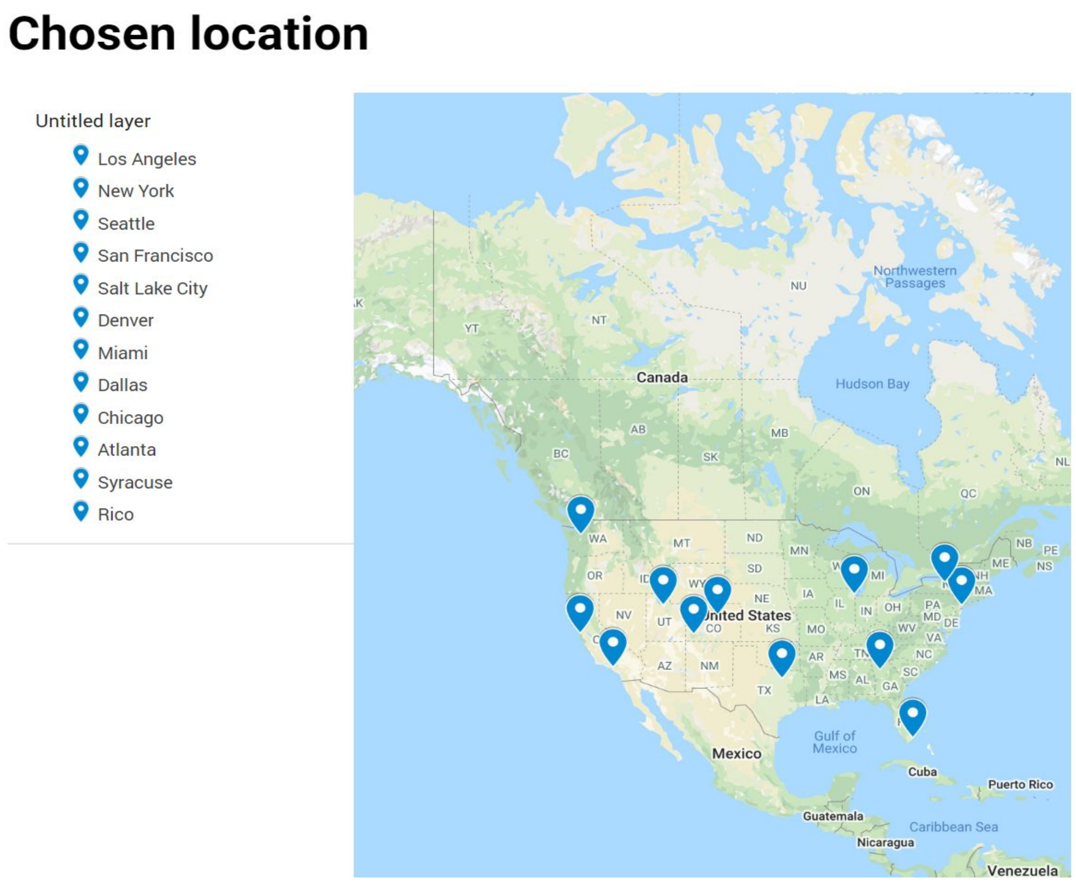

Implementation:
Step 1 : Tweet text clean
Step 2: Analyze sentiment attitude in tweets
>0
=0
<0
Positive
Neutral
Negative

Finally, we can add the ‘sentiment’ tag into our own tweet dictionary with the analysis result for later work.

4. Location Analysis

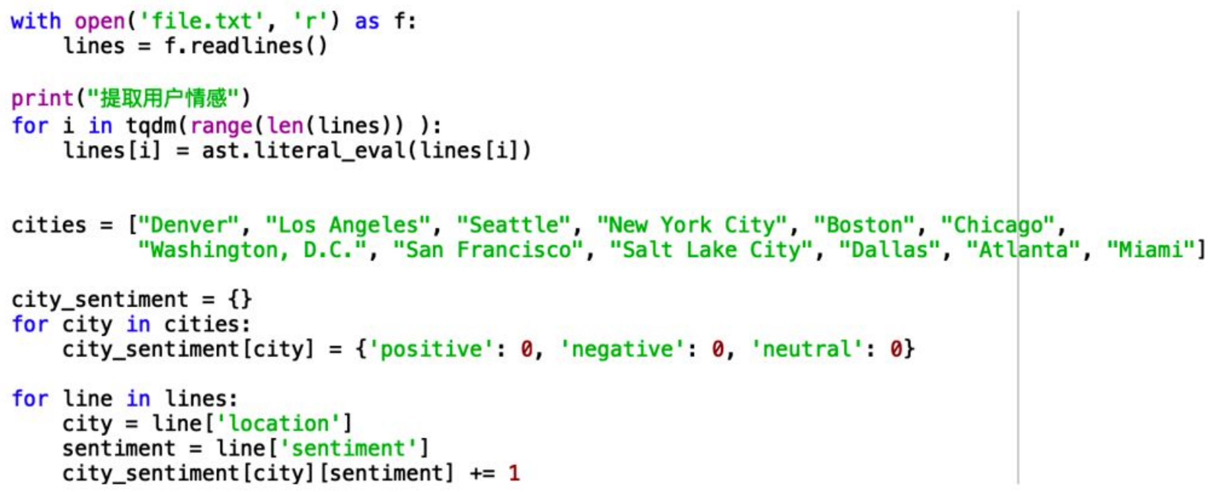
4.1 Sentiment Attitude

We want to see whether location would influence people's attitude towards this issue. So, to research the US attitudes to coronavirus based on the location. According to the user activation numbers of twitter, we choose12 cities that cover the whole mainland of the US, which are "Denver", "Los Angeles", "Seattle", "New York City", "Syracuse", "Chicago","Rico.", "San Francisco", "Salt Lake City", "Dallas", "Atlanta" and "Miami" just as the map shows.



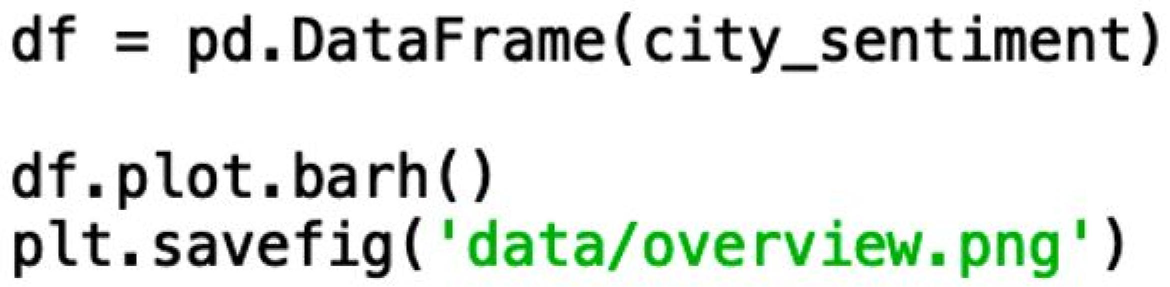
After collecting the data and analysis, we divide the attitude into three parts, positive, negative and neutral.

At first, we read the data, and do some basic setting.



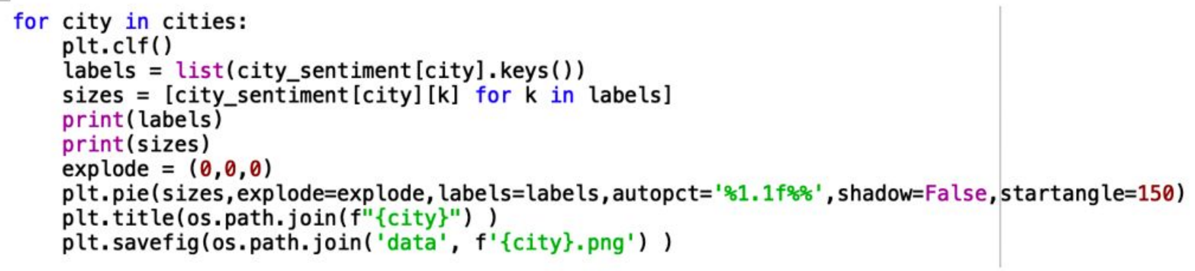
```python
with open('file.txt', 'r') as f:
    lines = f.readlines()

print("提取用户情感")
for i in tqdm(range(len(lines)) ):
    lines[i] = ast.literal_eval(lines[i])


cities = ["Denver", "Los Angeles", "Seattle", "New York City", "Boston", "Chicago",
          "Washington, D.C.", "San Francisco", "Salt Lake City", "Dallas", "Atlanta", "Miami"]

city_sentiment = {}
for city in cities:
    city_sentiment[city] = {'positive': 0, 'negative': 0, 'neutral': 0}

for line in lines:
    city = line['location']
    sentiment = line['sentiment']
    city_sentiment[city][sentiment] += 1
```

We use the dataframe to make a form and use barh to make a horizontal bar plot.

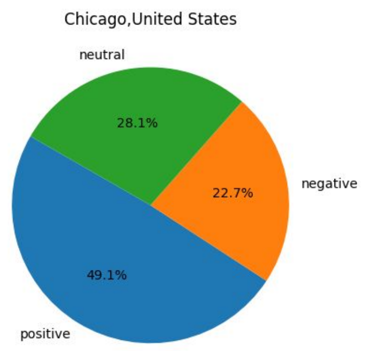


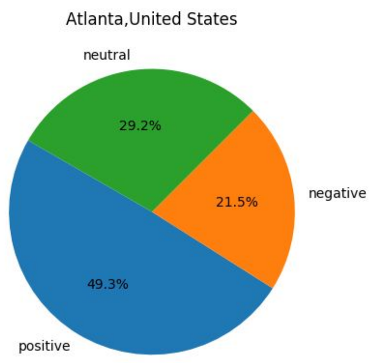
```python
df = pd.DataFrame(city_sentiment)

df.plot.barh()
plt.savefig('data/overview.png')
```

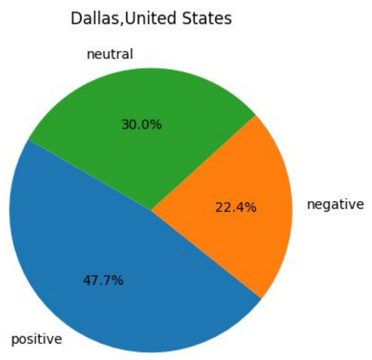
Then we use the function in plt to make the pie chart for each city and the label as showing below..

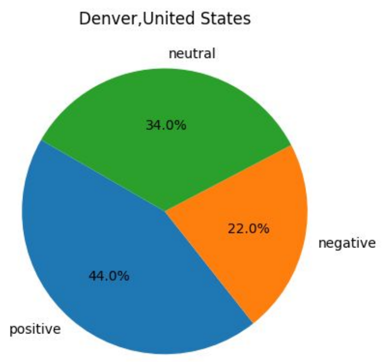


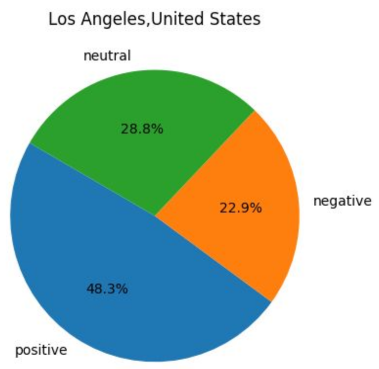
```python
for city in cities:
    plt.clf()
    labels = list(city_sentiment[city].keys())
    sizes = [city_sentiment[city][k] for k in labels]
    print(labels)
    print(sizes)
    explode = (0,0,0)
    plt.pie(sizes,explode=explode,labels=labels,autopct='%1.1f%%',shadow=False,startangle=150)
    plt.title(os.path.join(f"{city}") )
    plt.savefig(os.path.join('data', f'{city}.png') )
```

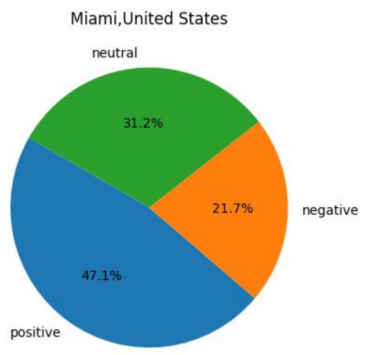

Chicago,United States
neutral
28.1%
22.7%
negative
49.1%
positive

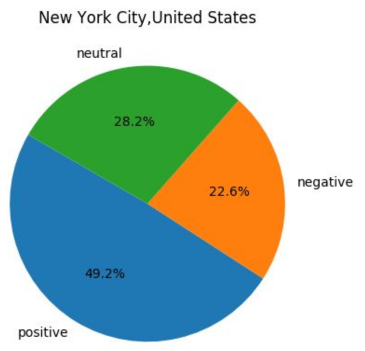

Atlanta,United States
neutral
29.2%
21.5%
negative
49.3%
positive

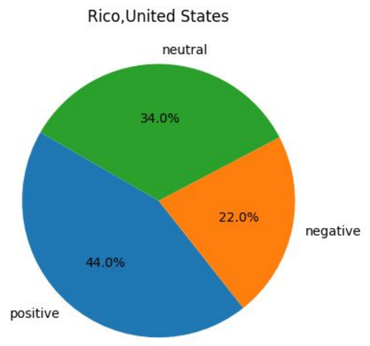

Dallas,United States
neutral
30.0%
22.4%
negative
47.7%
positive

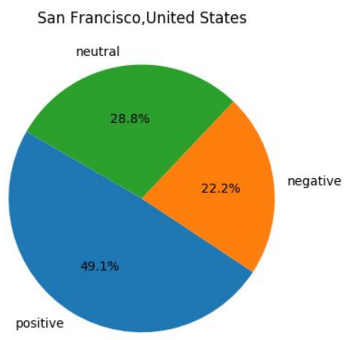

Denver,United States
neutral
34.0%
22.0%
negative
44.0%
positive

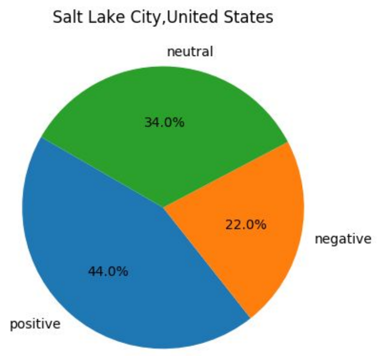

Los Angeles,United States
neutral
28.8%
22.9%
negative
48.3%
positive

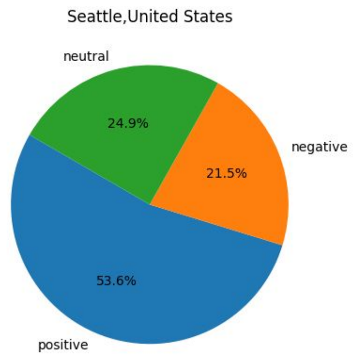

Miami,United States
neutral
31.2%
21.7%
negative
47.1%
positive

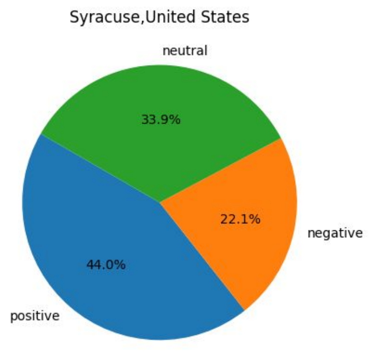






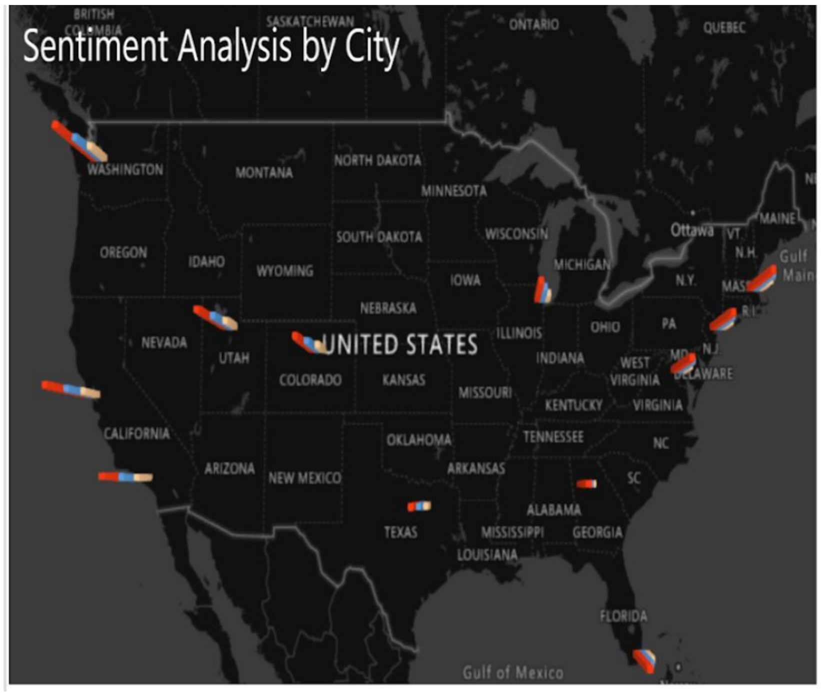


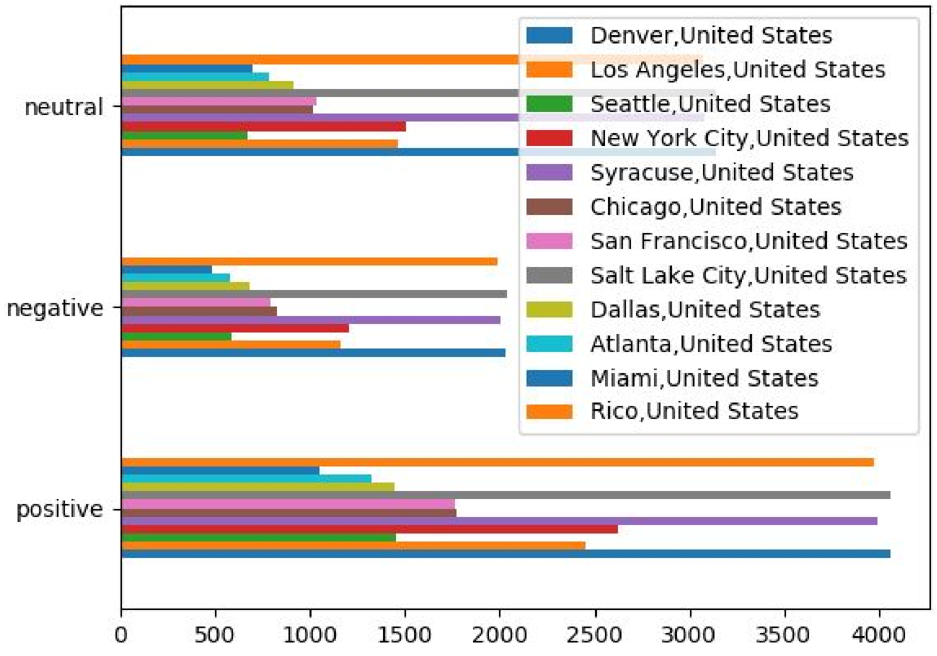




According to the pie chart, we can see that there are some differences in the rate of attitude of these cities. The users of Denver, Rico and Salt Lake City have the most rate of neutral attitude. Similarly, the users of Los Angeles have the most negative attitude. And Seattle users have the most positive attitude.
And among all the selected cities, a positive attitude still dominates even in the most severe region like New York city.

Then we generated the histogram and bar graph to try to get more information about the data.

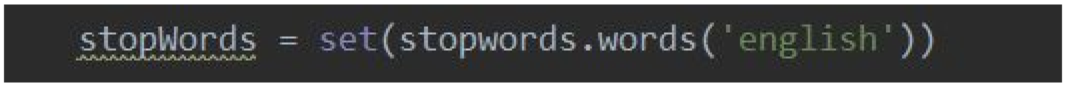




According to this bar graph, we can see that Salt Lake City has the most positive tweets and also the most negative numbers, while Denver has the most neutral ones.

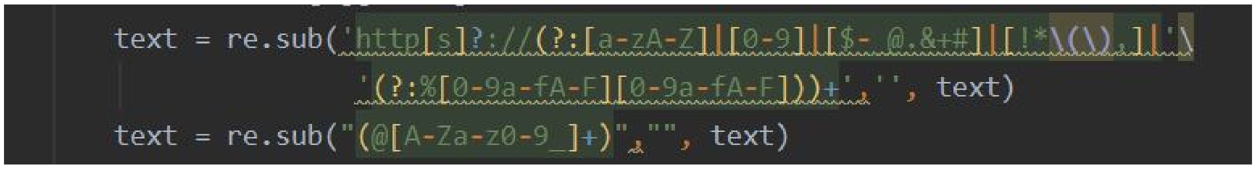
4.2 Word Cloud

After finishing the sentiment analysis for those cities, we would like to take further research not only on their emotional attitudes, but the word they spread most frequently. To do so, we made a word cloud through the python API.



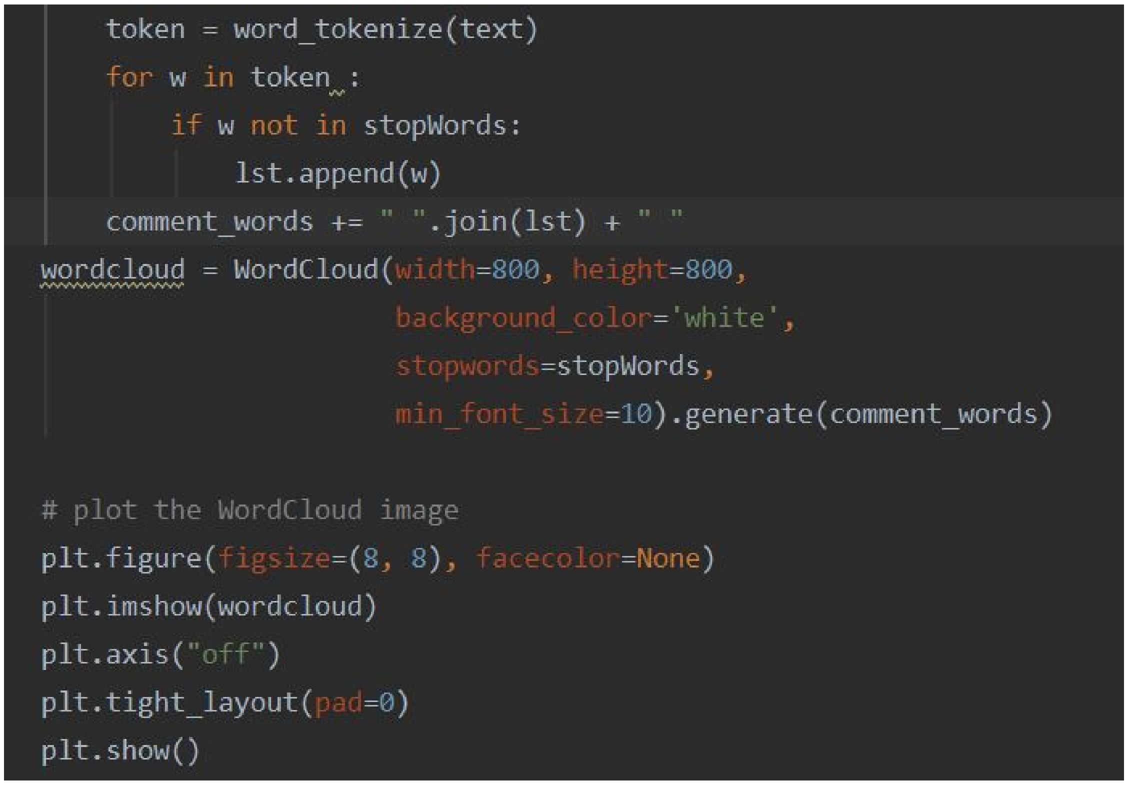
```python
stopWords = set(stopwords.words('english'))
```

We set the stopwords first, since they could be treated like the useless information, which means we should filter them.



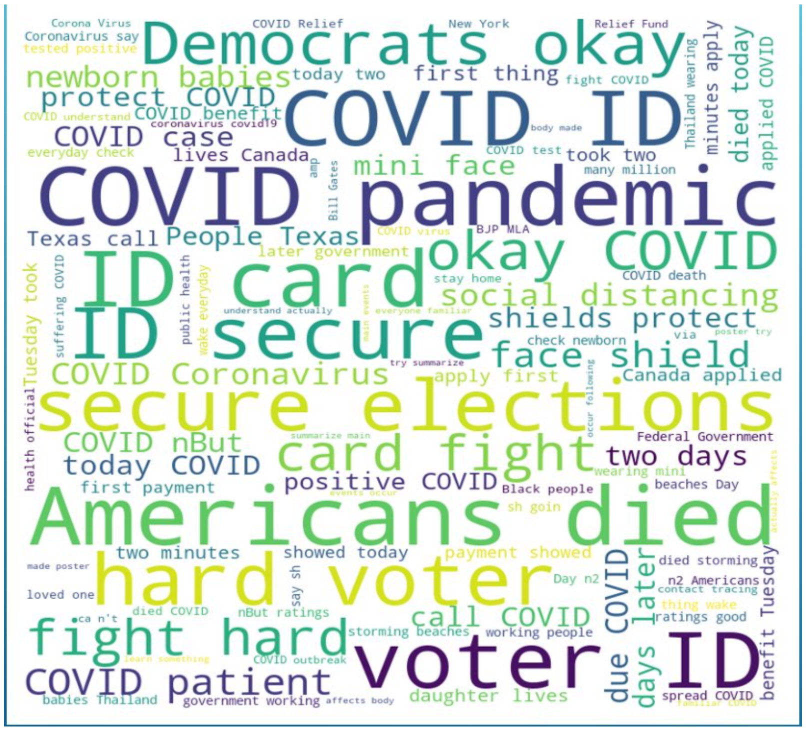
```python
text = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\
              '(?:%[0-9a-fA-F][0-9a-fA-F]))+','', text)
text = re.sub("(@[A-Za-z0-9_]+)","", text)
```

Then further clean the twitter text by deleting those symbols.



```python
    token = word_tokenize(text)
    for w in token :
        if w not in stopWords:
            lst.append(w)
    comment_words += " ".join(lst) + " "
wordcloud = WordCloud(width=800, height=800,
                      background_color='white',
                      stopwords=stopWords,
                      min_font_size=10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```

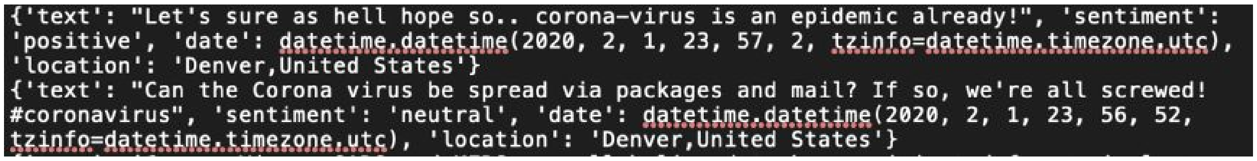
After that, we tokenized our text and added them as a whole list, then we could start our analysis through the WordCloud API, through putting those parameters, we could get a most frequent terms plot as shown below.



With this picture depicted, words people used to talk related to this issue are most about the virus itself, secure and death.

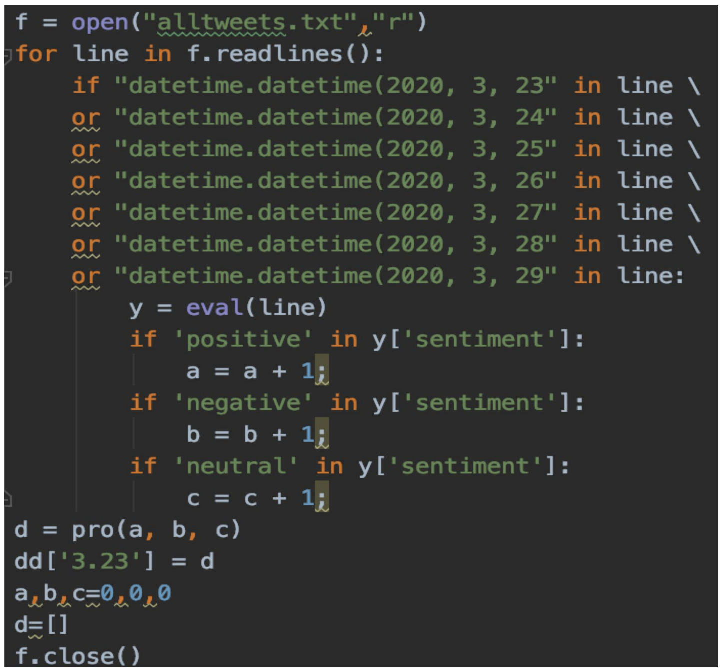
5. Time Analysis

5.1 Data Processing



```
{'text': "Let's sure as hell hope so.. corona-virus is an epidemic already!", 'sentiment':
'positive', 'date': datetime.datetime(2020, 2, 1, 23, 57, 2, tzinfo=datetime.timezone.utc),
'location': 'Denver,United States'}
{'text': "Can the Corona virus be spread via packages and mail? If so, we're all screwed!
#coronavirus", 'sentiment': 'neutral', 'date': datetime.datetime(2020, 2, 1, 23, 56, 52,
tzinfo=datetime.timezone.utc), 'location': 'Denver,United States'}
```

The image above is the result of data mining. The total number of tweets searched was 64868. And each tweet is in the form of dictionary. The time range is from February 1st to April 17th.



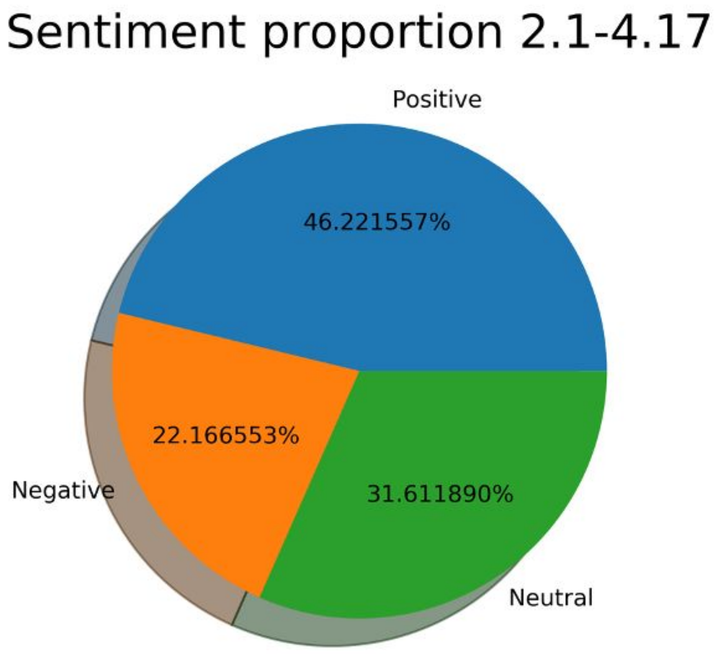
```
f = open("alltweets.txt","r")
for line in f.readlines():
    if "datetime.datetime(2020, 3, 23" in line \
    or "datetime.datetime(2020, 3, 24" in line \
    or "datetime.datetime(2020, 3, 25" in line \
    or "datetime.datetime(2020, 3, 26" in line \
    or "datetime.datetime(2020, 3, 27" in line \
    or "datetime.datetime(2020, 3, 28" in line \
    or "datetime.datetime(2020, 3, 29" in line:
        y = eval(line)
        if 'positive' in y['sentiment']:
            a = a + 1;
        if 'negative' in y['sentiment']:
            b = b + 1;
        if 'neutral' in y['sentiment']:
            c = c + 1;
d = pro(a, b, c)
dd['3.23'] = d
a,b,c=0,0,0
d=[]
f.close()
```

So just find the tweets in a week and use the eval function to make them be dictionaries. Then accumulate the whole number of each sentiment.

5.2 Making Graph

5.21 Overall Sentiment Proportion

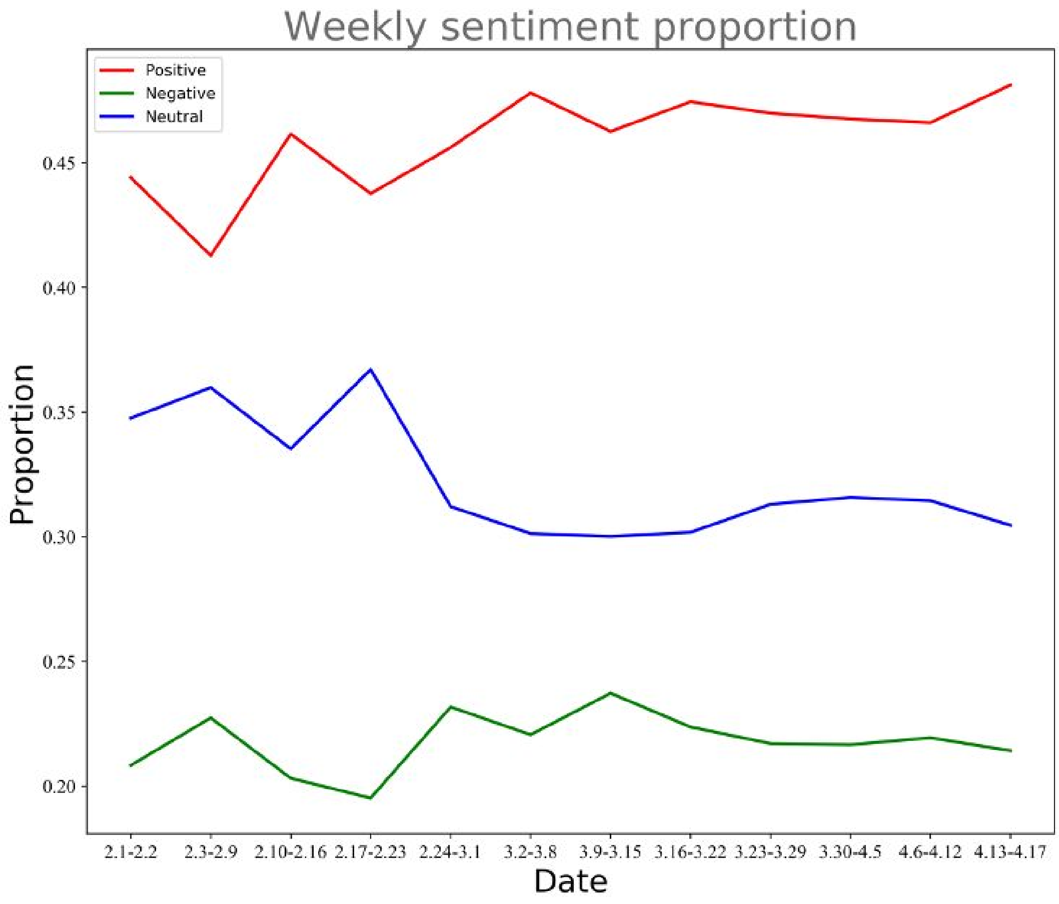
A pie chart can handle this kind of analysis. A pie chart shows a data series (data series: related data points drawn in a chart, which are derived from rows or columns of the data table. Each data series in the chart has a unique color or pattern and is in the chart legend Representation. You can draw one or more data series in the chart. The pie chart has only one data series.) The ratio of the size of each item to the sum of each item. Data points in a pie chart (data points: individual values plotted in a chart, these values are composed of bars, columns, polylines, pie or doughnut sectors, dots, and other data markers Graphical representation. Data markers of the same color form a data series.) Matplotlib is a Python drawing library. The pyplot package encapsulates many drawing functions. Matplotlib.pyplot contains a series of related functions similar to the drawing functions in MATLAB. Each function in Matplotlib.pyplot will make some modifications to the current image, for example: generate a new image, generate a new drawing area in the image, draw a line in the drawing area, mark the drawing, etc ... Matplotlib.pyplot will automatically remember the current image and drawing area, so these functions will directly act on the current image.



The overall proportion of three kinds of sentiments: positive, negative and neutral is shown above. What we can find is that positive sentiment accounts for the largest part by 46%. Neutral sentiment has the second largest proportion, which is about 31%. And unexpectedly, negative sentiment has just 22%, which is the smallest. From February 1st to April 17th, we can see that most people in America felt good about the situation or had confidence to overcome the frustration, and only 1/5 people felt bad about the novel coronavirus.

5.2.2 Weekly Sentiment Proportion

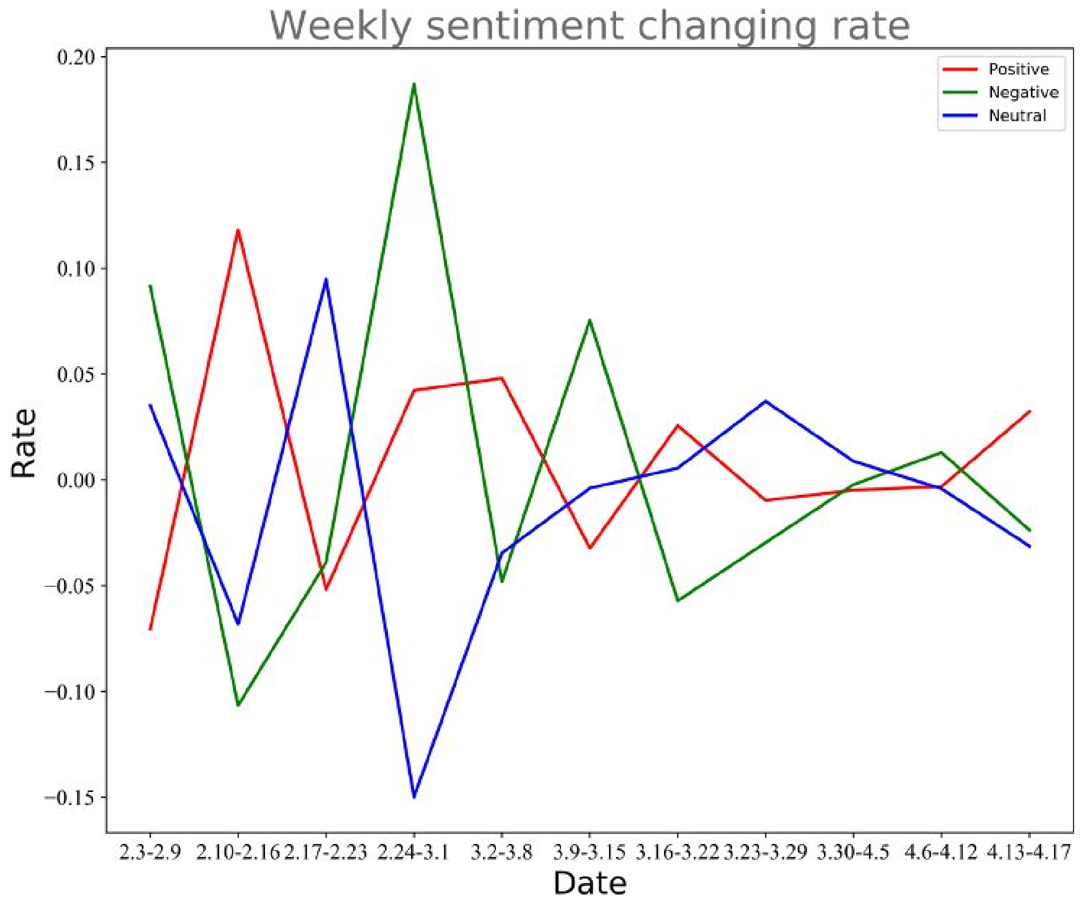
To find the sentiment change of people in America during this time period, further data processing is needed. Since the time range is so long, classification by week is suitable.
Line chart is the data arranged in the column or row of the worksheet that can be drawn into the line chart. Line charts can display continuous data that changes over time (according to common scale settings), so it is very suitable for displaying the trend of data at equal time intervals.
In order to get better analysis of the data, we chose to classify tweets using real weeks in the calendar. Since February 2nd was Sunday, the first label on the x-axis is 2.1-2.2.



As the graph above shown, the line represents positive sentiment keeps fluctuating around 45% and has a tendency to increase. The neutral one has an apparent decline from 2.17-2.23 to 2.24-3.1 and keeps stable after this decline. The negative one declines most in 2.17-2.23 but starts to increase after that. But the sentiment proportion in order still would be positive, neutral, negative every week.

5.2.3 Weekly Sentiment Changing Rate

In order to observe the change of sentiment more apparently, a changing rate is needed. So, we made another graph which named the weekly sentiment changing rate.



In this graph, positive values represent the increase of proportion compared to last week. The range of changing rate is in [-15%,20%], which is not very large. However, the increasement of negative sentiment and decline of neutral sentiment during 2.24-3.1 are very easy to find.
To find out what happened about covid-19 in America during 2.17-3.1, we searched the news on the Internet. The news displayed below are from the Timeline of the 2019–20 coronavirus pandemic in February 2020 in Wikipedia.

News:
20 February
The United States confirmed one more case in California, bringing the total number to 16.
21 February
The United States confirmed 20 more cases, bringing the total number to 35. Furthermore, the Association of Public Health Laboratories (APHL) announced that only three states were capable of testing for the corona virus: California, Nebraska, and Illinois.
24 February
The United States confirmed 18 more cases including evacuated passengers from the cruise ship, bringing the total number to 53.
26 February
The United States confirmed three new cases - one being a domestic case in California with no travel history, and the other two being former passengers aboard *Diamond Princess* - bringing the total number to 60.

28 February
The United States confirmed 4 more cases, including 2 former passengers of *Diamond Princess*. Washington state authorities later confirmed two additional presumptive cases, bringing its total to 66. One had recently returned from South Korea, and the other case was unrelated and locally acquired.
29 February
The United States confirmed its first death, a man from Washington near the Seattle area.

5.3 Summary

From the news and graph above, we can make a conclusion that the beginning of the pandemic of covid-19 and the first death in the USA made people start to worry about the situation. But the overall proportion seems not to change obviously.
So according to the information we crawled, people in the United states of America were sensitive to the news and changing situation of COVID-19, since we can map the change of sentiment to the news that happened. However, generally speaking, most people felt positive about covid-19 and the situation during 2.1 to 4.17. Only about 22% of people felt negative about it. It is hard to believe that the positive sentiment proportion was still increasing. It seems that people have not realized the seriousness of the novel coronavirus.

6. Conclusion

In this report, we show how we collect tweets containing keywords related to COVID-19, how we implement sentiment analysis on the data we collected and divide them into three groups labelled positive, negative,and neutral respectively. Finally, we try to figure out whether people's attitudes toward COVID-19 vary from time and location.

Before the result came out, our initial assumption is that as the situation of the epidemic is getting worse, the number of people who hold positive and neutral attitudes may drop significantly, while the number of people holding negative attitudes will grow quickly. We also believed that the people in cities with more infected people and death cases tend to have more negative attitudes than the cities whose situation is not that serious. However, the real situation is just in contrast to our assumption. As time goes on, the number of people holding positive attitudes are growing and the rate of people holding negative and neutral attitudes are decreasing. What's more, we find that there is no obvious difference between cities suffering worse situations and cities with better situations. We also find an interesting thing that there are several days that the number of people who hold negative attitudes increased dramatically. The reason behind these outstanding growth of negative attitudes are the news that Diamond Princess bringing the total number to 60 and the first confirmed death case in America. It seems only when danger is getting closed to people themselves and facing death, people will realize the seriousness of this epidemic. However, as the epidemic situation is not improved, people have become numb to the rise in the number of people infected.

According to our research, we think people are not cautious enough in such a terrible situation and have not realized how dangerous COVID-19 is. If people's attitudes toward COVID-19 do not change, it seems that the situation will not turn better in a short time. What's worse, the epidemic will get more serious and pose a greater threat to people's lives. We hope more and more people will change their mind and the horrible epidemic will be overcome eventually.