RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Project Report
COVID-19 Twitter Sentiment Analysis using NLP

Course: Problem Solving with Data
Prof. Dr. Souvick Ghosh
Spring 2020

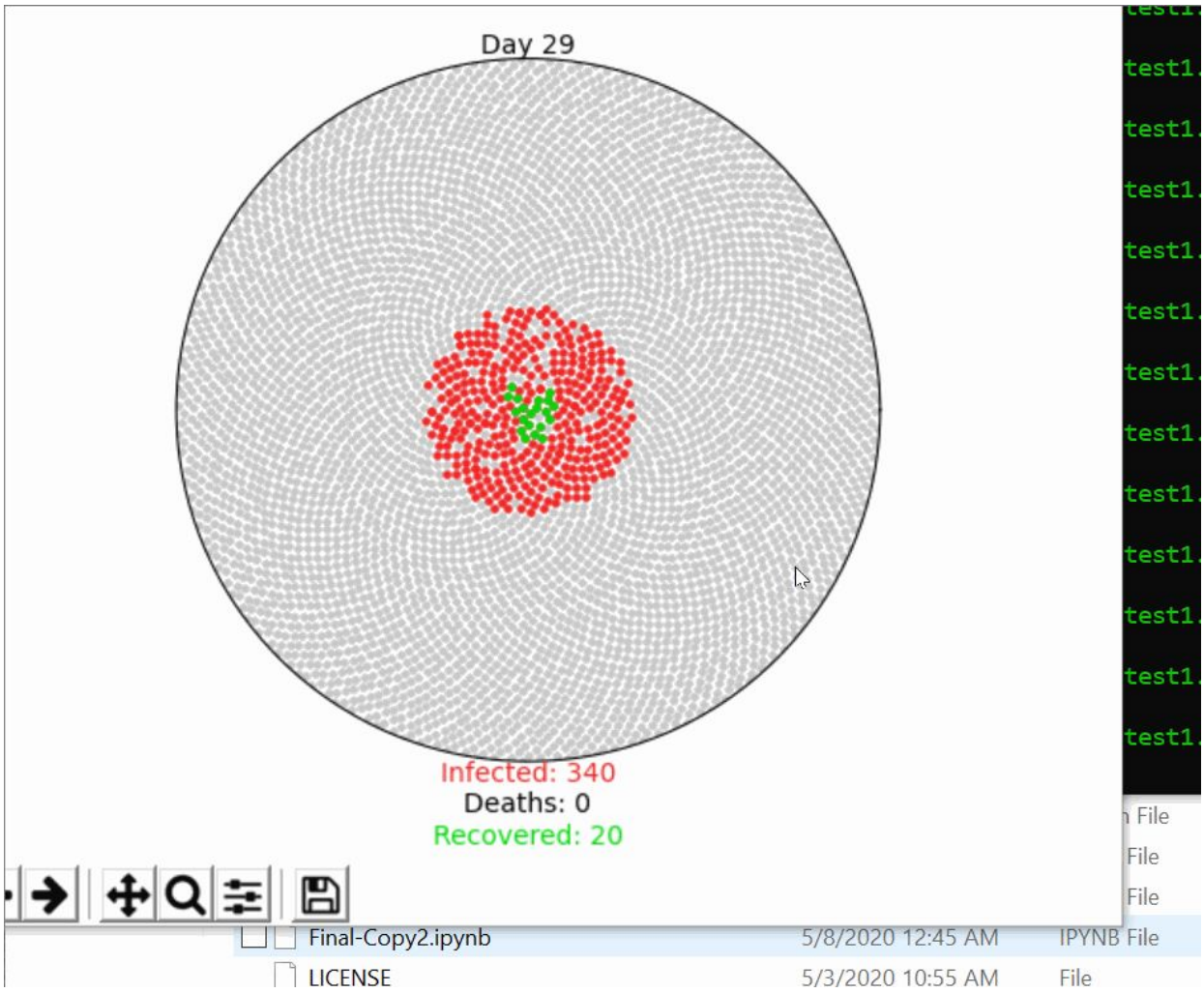
By
Harsha Mangnani
Anthony Bedi

Table of Contents

```
COVID19_PARAMS = {
    "r0": 2.3,
    "incubation": 5,
    "percent_mild": 0.8,
    "mild_recovery": (7, 14),
    "percent_severe": 0.2,
    "severe_recovery": (21, 42),
    "severe_death": (14, 56),
    "fatality_rate": 0.034,
    "serial_interval": 7
}
```

Pandemic Spread on a hypothetical population of 4500. This can be modified to simulate any virus given the following information.

https://www.washingtonpost.com/graphics/2020/health/coronavirus-how-epidemics-spread-and-end/



Tutorial

Research Questions

Our data analysis consisted of three main research areas. First, we performed a number of different natural language processing and textual analysis techniques on a large dataset of tweets related to Covid. Our research questions here included “Which terms are used most frequently along with Covid?”, “Which pairs of words appear the most in positive and negative tweets?” and the hypothesis that “With increase in number of days since the pandemic started, there will be an increase in the number of negative tweets” The second area consisted of analyzing the account creation date and word counts of tweets. Our main research questions related to these topics were “Do newer accounts have more polar tweets?” and “Are wordier tweets more polar?”. Next, we moved to analyzing the average polarity of tweets in a state vs the number of total Covid cases. Our main research question in this area was “Do states with more confirmed cases have more polar tweets?”. In the following sections, we will provide further explanations of our hypothesis, along with describing our data collection technique and results.

Natural Language Processing

Sentiment analysis (also known as opinion mining) is one of the many applications of Natural Language Processing. It is a set of methods and techniques used for extracting subjective information from text or speech, such as opinions or attitudes. In simple terms, it involves classifying a piece of text as positive, negative or neutral.

Hypothesis

For this dataset, we believed that the global quarantine efforts would lead to a higher percentage of negative tweets due to the associated struggles. Other questions include:

- What are the most common words in the entire dataset?
- What are the most common words in the dataset for negative and positive tweets, respectively?
- How many hashtags are there in a tweet?

Data Collection

In total, we collected around 12,300 tweets for this dataset. The search terms used were ‘covid-19’ , ‘coronavirus’, and ‘pandemic’.

In any natural language processing task, cleaning raw text data is an important step. It helps in getting rid of the unwanted words and characters which helps in obtaining more consistent results. If we skip this step then there is a higher chance that the data would be too noisy to work with.. The objective of this step is to remove punctuation, special characters, numbers, and terms which do not contribute to the sentimentality.

Data Cleaning:

Remove unwanted patterns from tweets:

Remove user handles (@user): We removed the twitter handles as they are already masked as @user due to privacy concerns. These twitter handles do not provide any information about the nature of the tweet.

Remove Punctuations, Numbers and Special characters: We got rid of the punctuations, numbers and even special characters since they wouldn’t help in differentiating different types of tweets. We removed numbers as well because covid-19 or covid, doesn’t matter. We removed punctuations because we will use TextBlob to assign sentiment. TextBlob, unlike VADER, doesn’t take into account the punctuations.

Remove Short Words: Most of the smaller words do not add much value. Examples of these include: ‘pdx’, ‘his’, ‘all’. We removed these. We have to be a little careful here in selecting the length of the words which we want to remove. So, we decided to remove all the words having 3 or less characters.

Convert to Lower Case: Here we converted the text to all lower case so that terms such as Covid and covid are treated similarly.

Remove Stopwords: Stop words are a set of commonly used words in a language. Examples of stop words include: “a”, “we”, “the”, “is”, and “are”. The idea behind removing stop words is

that we can focus on the important words instead. We can either create a custom list of stopwords ourselves (based on use case) or we can use predefined libraries.

Removing most common words: This is optional. Depending on the most common words, we can remove the words if unnecessary. We did not remove common words as the words are of importance to our hypothesis.

Remove rare words: Once again, this is optional. This is very intuitive, as some of the words that are very unique in nature like names, brands, product names, and some of the noise characters, such as html left outs, also need to be removed for different natural language processing tasks. We removed these unnecessary words as they had the lowest frequency.

Converting Emojis to Text: Twitter text also contains a good amount of emoticons. In sentiment analysis, emojis express an emotion. Hence, removing them might not be a good solution.

Remove URLs: Recall the presence of URLs in the 15 most common words. To remove the URLs, we can also utilize Beautiful Soup Library.

Before and After Preprocessing:

```
#Before and After Preprocessing
df[['text', 'tidy_tweet']].head()
```

|:

	text	tidy_tweet
0	b'@realDonaldTrump Trump believes killing 70,0...	trump believes killing americans destroying am...
1	b'RT @bessbell: My dad is an ICU doctor treati...	doctor treating covid patients past week never...
2	b'RT @Thomas1774Paine: A leaked dossier conclu...	leaked dossier concludes china lied deliberate...
3	b'Free COVID-19 testing is available to ALL Lo...	free covid testing available angeles county re...
4	b'RT @ClevelandPolice: NEW SCAMS TO WATCH OUT ...	scams watch nantigen testing emails advertisin...

Tokenization: Tokens are individual terms or words, and tokenization is the process of splitting a string of text into tokens.

	tidy_tweet	token_tweet
3610	#covid intel report assess chinese government ...	[, covid, intel, report, assess, chinese, gove...
4996	guardian reporting tests sent care homes carri...	[guardian, reporting, tests, sent, care, homes...
4475	#ifnotfortacha twitter look smile kill kill covid	[, ifnotfortacha, twitter, look, smile, kill, ...
10761	amid #coronavirus crisis must continue focus s...	[amid, coronavirus, crisis, must, continue, fo...

Stemming and Lemmatization:

Lemmatization is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors. Here we have used nltk's PorterStemmer() function to stem the tweets.

	token_tweet	stem_tweet
1142	[covid, testing, cimas, nstop, nonsense, covid...	covid test cima nstop nonsens covid public hea...
2056	[cmany, people, survive, disease, going, rehab...	cmani peopl surviv diseas go rehabilit need ex...
11227	[, covid, news, nih, funded, study, incidence,...	covid news nih fund studi incid novel coronav...
9553	[april, secretary, alex, azar, says, keeps, ev...	april secretari alex azar say keep everyon bio...
8851	[unprepared, part, answer, pandemic, trump, st...	unprepar part answer pandem trump stockpil chi...

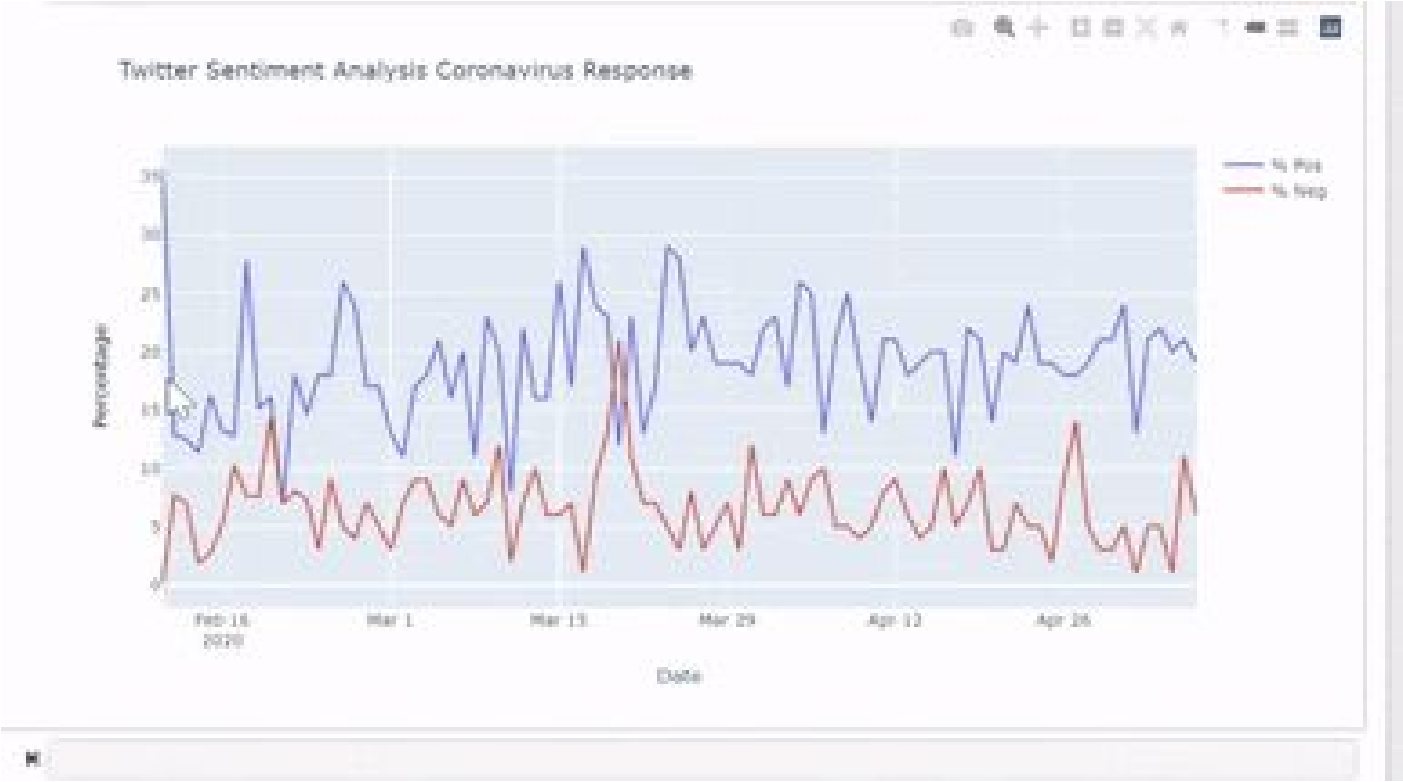
We have only lemmatized the tweets. We need to provide the POS tag of the word along with the word for lemmatizer in NLTK. Depending on the POS (Part of Speech), the lemmatizer may return different results.

	tidy_tweet	lemm_tweet
8338	monday pandemic https sczpcdnvmc	monday pandemic http sczpcdnvmc
1719	twitter policy enough contain disinformation r...	twitter policy enough contain disinformation r...
9530	hahahaha pandemic caused alot panic	hahahaha pandemic cause alot panic
823	president today office presidential villa abuj...	president today office presidential villa abuj...
2314	breaking news president duterte distributes re...	break news president duterte distribute relief...

Sentiment Analysis using TextBlob:

We then went ahead and collected 100 tweets for each day starting from Feb 11, 2020 until May 07, 2020. We tried collecting more tweets but had to interrupt the kernel. We did some research to find a work around and we found that Kafka Producer and Consumer might help. I (Harsha) am still learning it, so we stuck with 100 tweets.

Our hypothesis was based on the fact that the impacts of COVID such as quarantine and job layoffs (all over LinkedIn) would lead to a higher percentage of negative sentiment tweets. From the data we collected, we found this:



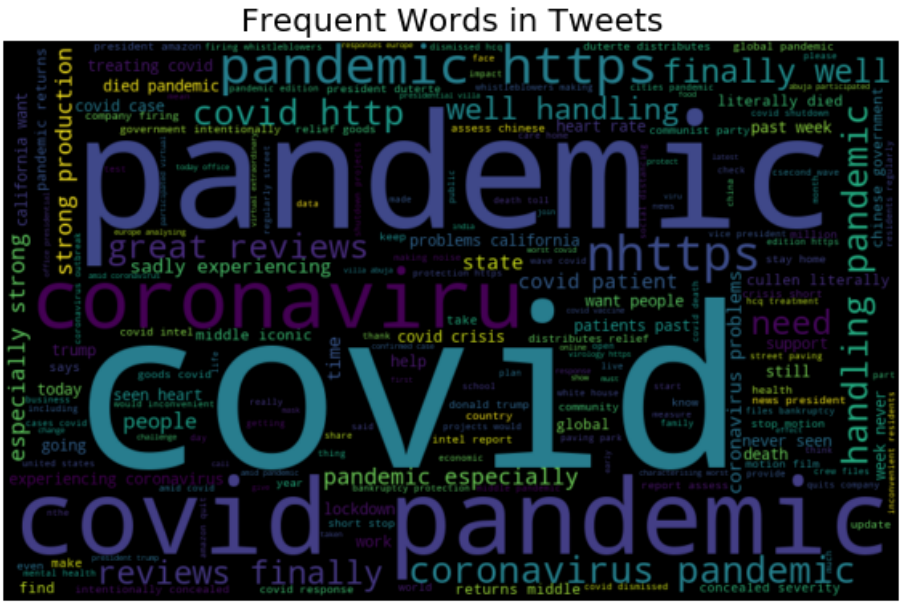
Clearly, there is more of a positive trend going on. There are some hikes in negative tweets around mid-march, the hike might be a result of the lockdown announcement all over the world, which might have caused panic. Otherwise, we have an overall positive trend. **Hence, we reject our hypothesis.** We believed that the global quarantine efforts would lead to a higher percentage of negative tweets due to the associated struggles, but sentiment on twitter suggests otherwise. To dig in further, we checked the words associated with positive and negative tweets.

Story Generation and Visualization from Tweets

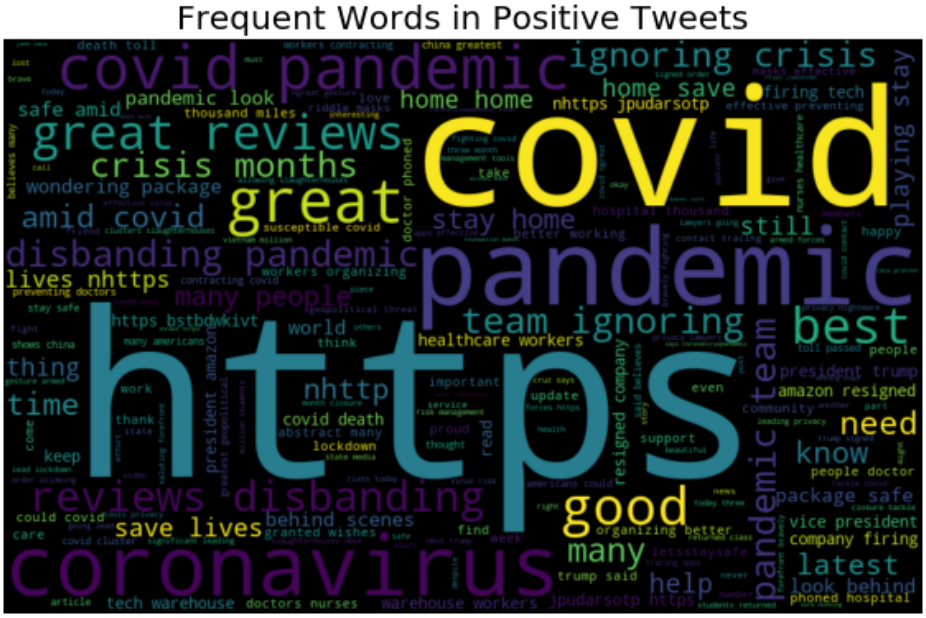
In this section, we explored the cleaned tweets. Exploring and visualizing data, no matter whether its text or any other data, is an essential step in gaining insights.

WordCloud: A word cloud is a visualization wherein the most frequent words appear in large size and the less frequent words appear in smaller sizes.

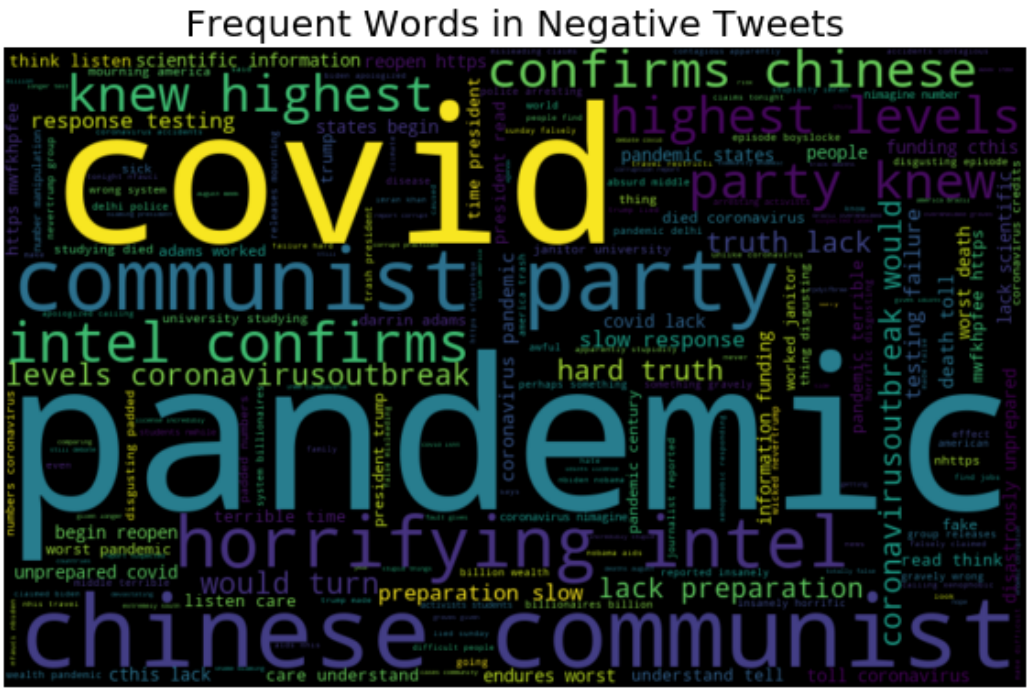
A) Most frequent words in all tweets we collected.



B) Positive Tweets have a Polarity of greater than 0.4.



C) Negative Tweets have a Polarity of less than 0.4.

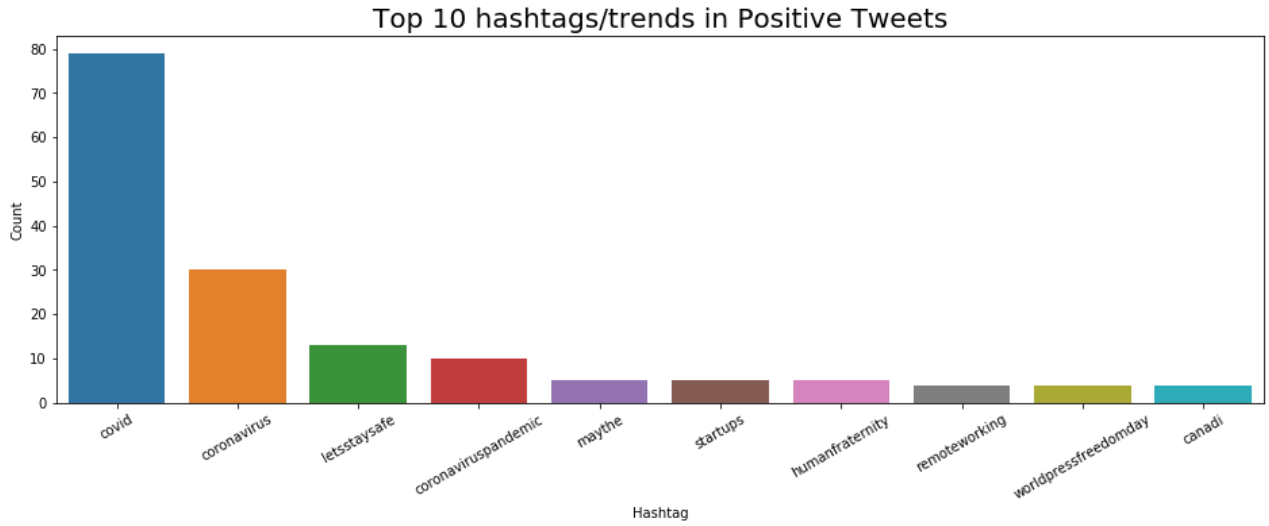


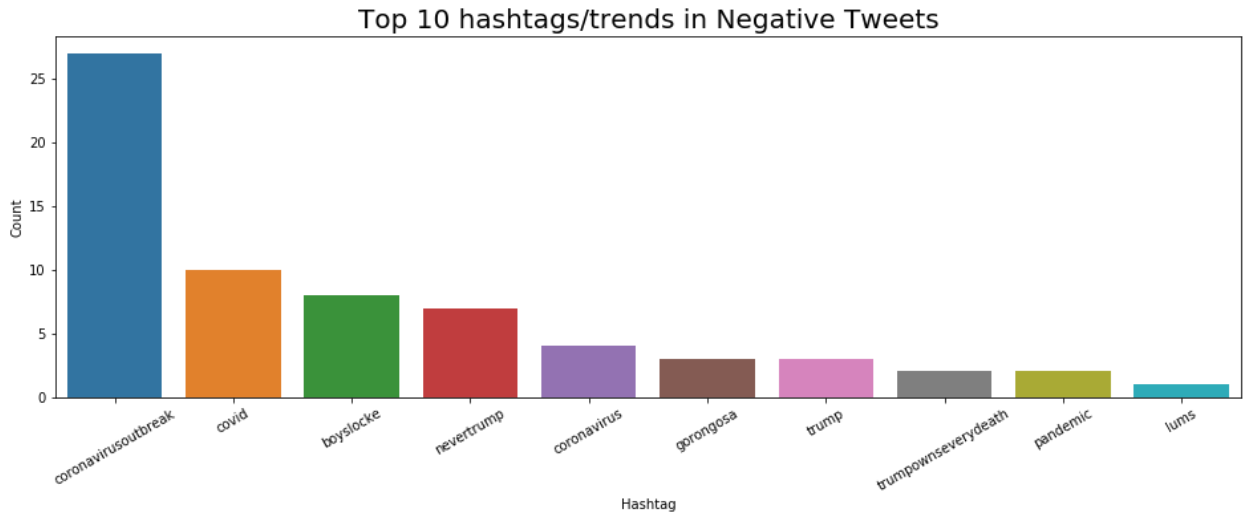
Looking into the terms shown more closely, we can see obvious examples of positive and negative words. Positive words include great, good, best, and save lives; while negative includes hard truth, slow preparation, horrifying, and worst pandemic. What stuck out here were the presence of government related words in the negative tweets such as Chinese, Communist, intel, and states. From these results, we would venture to guess that more negative tweets emphasized a poor response from the government while positive tweets focused on general good news.

D) Understanding the impact of Hashtags on tweets sentiment

Hashtags in twitter are synonymous with the ongoing trends on twitter at any particular point in time. We should try to check whether these hashtags add any value to our sentiment analysis task, i.e., they help in distinguishing tweets into the different sentiments.

We stored all the trend terms in two separate lists — one for positive tweets and the other for negative tweets. Now that we have prepared our lists of hashtags for both the sentiments, we can plot the top 'n' hashtags.





As expected, negative tweets use words with a more negative connotation. For example, positive tweets more frequently referred to the outbreak as covid, while negative tweets used the term coronavirus outbreak. Virus and outbreak seem to carry more weight than just covid, thus adding to the negative connotation of the tweet.

Sentiment Analysis using Vader Sentiment Intensity Analyser

Before moving on to extracting features, we need to take into consideration the emoticons, emojis and punctuations we removed earlier. Punctuations and emojis are a way of expression. TextBlob gets confused with punctuations and doesn't work well. TextBlob is a decent sentiment analyser to get a good idea of what is going on social media. To dig in further, we used Vader Sentiment Intensity Analyser. It takes into consideration the intensity of expression as well. For example:

```
#!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

sent_analyser = SentimentIntensityAnalyzer()
```

```
def calculate_sentiment_analyser(text):
    return sent_analyser.polarity_scores(text)
```

```
c = (" I love Mango so much that I am starting to hate it")
calculate_sentiment_analyser(c)
```

```
]: {'neg': 0.207, 'neu': 0.559, 'pos': 0.235, 'compound': 0.128}
```

```
c = (" I love Mango so much that I am starting to hate it!!!")
calculate_sentiment_analyser(c)
```

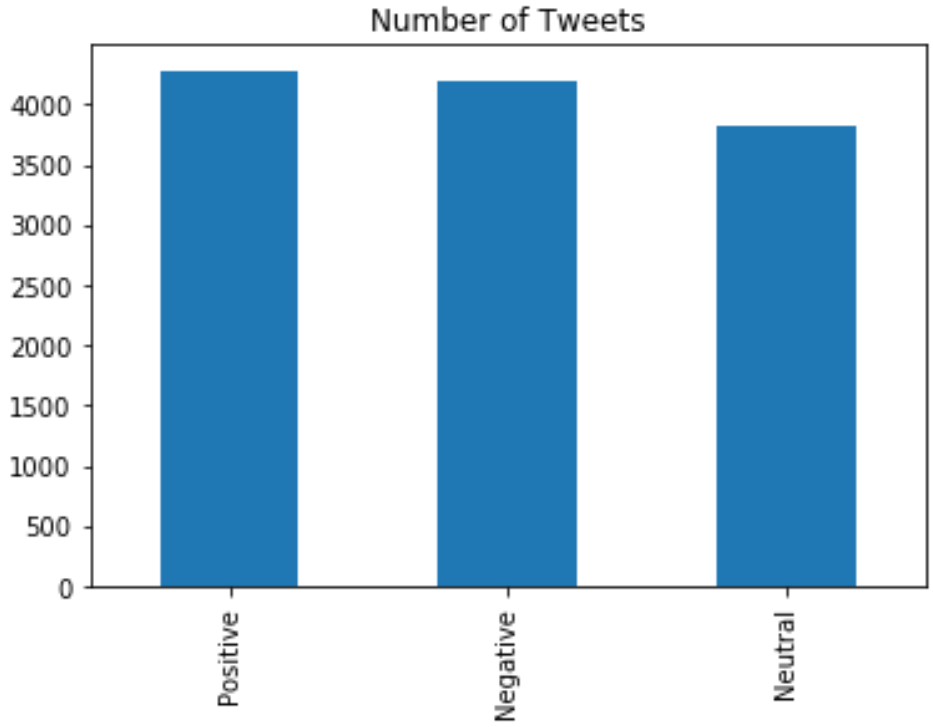
```
]: {'neg': 0.197, 'neu': 0.533, 'pos': 0.27, 'compound': 0.3348}
```

```
c = (" I love Mango so much that I am starting to hate it!! <3")
calculate_sentiment_analyser(c)
```

```
]: {'neg': 0.173, 'neu': 0.468, 'pos': 0.359, 'compound': 0.6103}
```

We went ahead and cleaned our 'text' column with the preprocessing techniques mentioned before. We removed user handles, short words, special characters, numbers, rare words, stopwords and URLs. We left the punctuations, emojis and emoticons in their original form.We then assigned class Positive, Negative, or Neutral based on the compound_score returned by sentiment intensity analyser

Tweets in each class:



These results were interesting as we assumed there would be more negative results during this time of crisis. Instead, there seems to be a very even spread between the three categories. This supports our rejection of the hypothesis, negative tweets are slightly less in number (TextBlob showed greater difference). It would be interesting to see if this trend continues in more "normal" times.

Feature Engineering:

To analyse a preprocessed data, it needs to be converted into features. Depending upon the usage, text features can be constructed using assorted techniques such as: Bag of Words, TF-IDF, and Word Embeddings.

1. Bag of Words Features:

Consider a Corpus C of D documents {d1,d2…..dD} and N unique tokens extracted out of the corpus C. The N tokens (words) will form a dictionary and the size of the bag-of-words matrix M will be given by D X N. Each row in the matrix M contains the frequency of tokens in document D(i). Example:

D1: He is a lazy boy. She is also lazy.

D2: Smith is a lazy person.

The dictionary created would be a list of unique tokens in the corpus
=[‘He’,’She’,’lazy’,’boy’,’Smith’,’person’]

Here, D=2, N=6

The matrix M of size 2 X 6 will be represented as following (explanation credits)

	He	She	lazy	boy	Smith	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

Now the columns in the above matrix can be used as features to build a classification model.

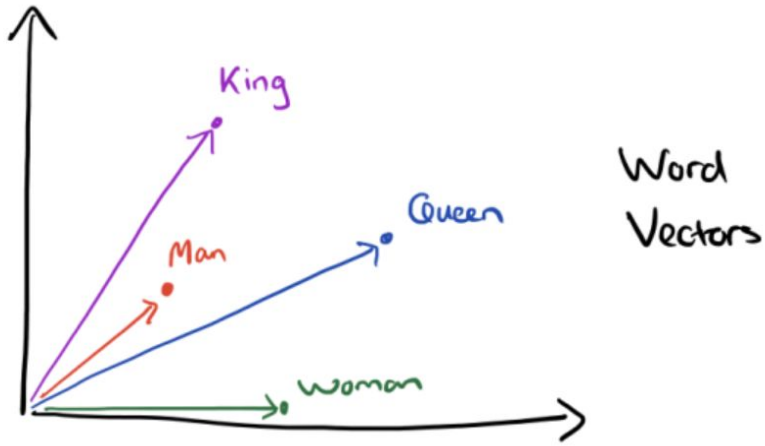
2. TF-IDF Features:

This method is based on the frequency method but it is different to the bag-of-words approach in the sense that it takes into account not just the occurrence of a word in a single document (or tweet) but in the entire corpus.

TF-IDF works by penalising the common words by assigning them lower weights while giving importance to words which are rare in the entire corpus but appear in good numbers in few documents. Example:

- TF = (Number of times term t appears in a document)/(Number of terms in the document)
- IDF = log(N/n), where, N is the number of documents and n is the number of documents a term t has appeared in.
- TF-IDF = TF*IDF

3. Word2Vec Features:



Word embeddings are the modern way of representing words as vectors. The objective of word embeddings is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. They are able to achieve tasks like **King -man +woman = Queen**, which is mind-blowing.

The advantages of using word embeddings over BOW or TF-IDF are:

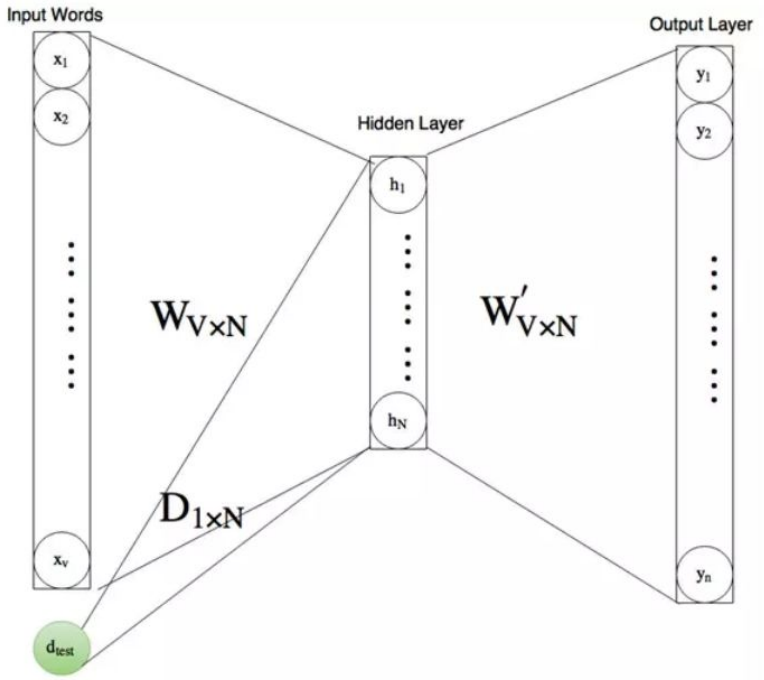
- Dimensionality reduction - significant reduction in the no. of features required to build a model.
- It captures meanings of the words, semantic relationships and the different types of contexts they are used in.

Word2Vec Embeddings:

Word2Vec is not a single algorithm but a combination of two techniques – **CBOW (Continuous bag of words)** and **Skip-gram model**. Both of these are shallow neural networks which map word(s) to the target variable which is also a word(s). Both of these techniques learn weights which act as word vector representations.

CBOW tends to predict the probability of a word given a context. A context may be a single adjacent word or a group of surrounding words. The Skip-gram model works in the reverse manner, it tries to predict the context for a given word. Below is a diagrammatic representation of a 1-word context window Word2Vec model.

There are three layers: - an input layer, - a hidden layer, and - an output layer.



The input layer and the output, both are one- hot encoded of size [1 X V], where V is the size of the vocabulary (no. of unique words in the corpus). The output layer is a softmax layer which is used to sum the probabilities obtained in the output layer to 1. The weights learned by the model are then used as the word-vectors.

We will go ahead with the Skip-gram model as it has the following advantages:

- It can capture two semantics for a single word. i.e it will have two vector representations of ‘apple’. One for the company Apple and the other for the fruit.
- Skip-gram with negative sub-sampling outperforms CBOW generally.

We will train a Word2Vec model on our data to obtain vector representations for all the unique words present in our corpus. There is one more option of using **pre-trained word vectors** instead of training our own model. Some of the freely available pre-trained vectors are:

- Google News Word Vectors
- Freebase names
- DBPedia vectors (wiki2vec)

We have trained our own word vectors since the size of the pre-trained word vectors is generally huge.

```
#Example:

#We will specify a word
#and the model will pull out the
#most similar words from the corpus.

model_w2v.wv.most_similar(positive="trump")
```

```
[('donald', 0.6122864484786987),
 ('administration', 0.5912898778915405),
 ('retire', 0.5111641883850098),
 ('hide', 0.5052764415740967),
 ('iraq', 0.5042929649353027),
 ('vkozcq', 0.5014142990112305),
 ('staged', 0.49990102648735046),
 ('gaze', 0.4993215799331665),
 ('erased', 0.49691057205200195),
 ('customarily', 0.49473753571510315)]
```

```
model_w2v.wv.most_similar(positive="pandemic")
```

```
[('bitches', 0.401022732257843),
 ('curve', 0.3848109841346741),
 ('kdyeaj', 0.38240426778793335),
 ('cebm', 0.38196104764938354),
 ('ferrari', 0.38168415427207947),
 ('getting', 0.3783207833766937),
 ('rock', 0.37499311566352844),
 ('shore', 0.37347519397735596),
 ('bloomberg', 0.37278324365615845),
 ('nrea', 0.3705732226371765)]
```

From the above two examples, we can see that our word2vec model does a good job of finding the most similar words for a given word. But how is it able to do so? The reason is it has learned vectors for every unique word in our data and it uses cosine similarity to find out the most similar vectors (words).

Preparing Vectors for Tweets:

Since our data contains tweets and not just words, we'll have to figure out a way to use the word vectors from the word2vec model to create vector representation for an entire tweet. There is a simple solution to this problem, we can simply take the mean of all the word vectors present in the tweet. The length of the resultant vector will be the same, i.e. 200. We will repeat the same

process for all the tweets in our data and obtain their vectors. Now we have 200 word2vec features for our data.

Classification Modelling:

After all the pre-modeling stages required to get the data in the proper form and shape, we built models on the datasets with different feature sets prepared in the earlier sections — Bag-of-Words, TF-IDF, word2vec vectors, and doc2vec vectors. We used the following algorithms to build models:

- RandomForest
- Logistic Regression
- MultiNomialNaiveBayes
- Support Vector Machine

Evaluation Metric

F1 score is being used as the evaluation metric. It is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is suitable for uneven class distribution problems. (Why F1 explanation?)

The important components of F1 score are:

True Positives (TP) - These are the correctly predicted positive values which means that the value of the actual class is yes and the value of the predicted class is also yes.

True Negatives (TN) - These are the correctly predicted negative values which means that the value of the actual class is no and the value of the predicted class is also no.

False Positives (FP) – When actual class is no and predicted class is yes.

False Negatives (FN) – When actual class is yes but predicted class in no.

Precision = TP/TP+FP

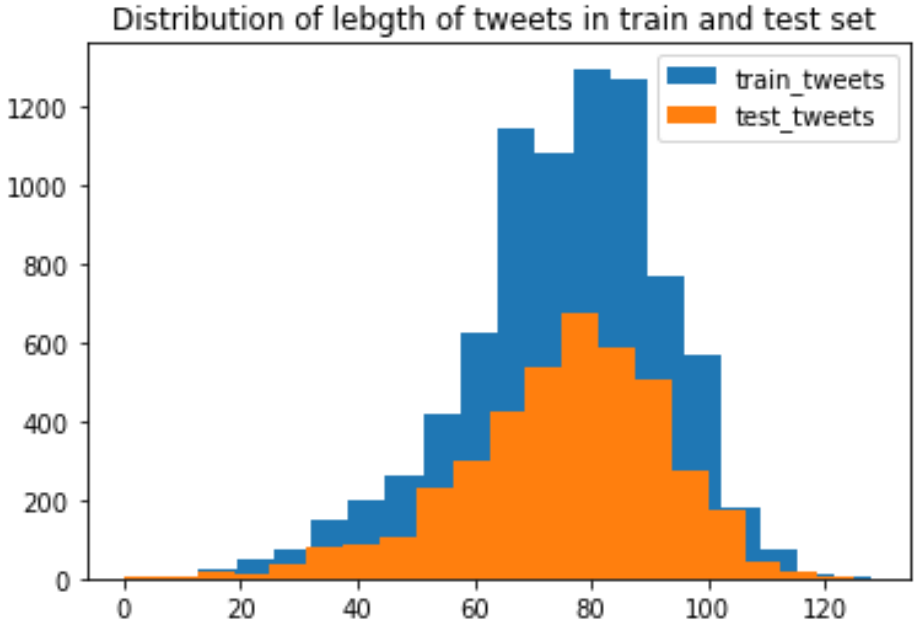
Recall = TP/TP+FN

F1 Score = 2(Recall * Precision) / (Recall + Precision)

Train Test Split:

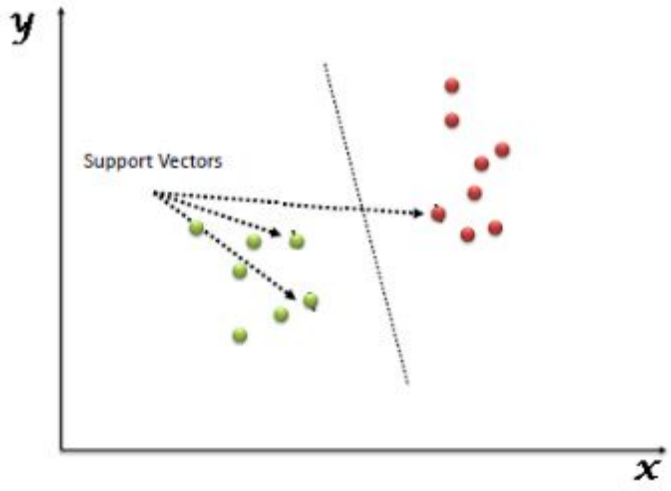
Some training algorithms will perform poorly if they get many similar instances (labels) in a row. So we shuffled the data before splitting it into train and test sets. It was observed that running train_test_split on the heavy preprocessed data frame sometimes resulted in the system going out of memory. Hence to avoid such cases, one extra line of code was added. We used df.reindex code will shuffle the indices initially, so that later splitting the dataset into training and testing will give fairer results. We used numpy's random permutation as well.

After the split, we noticed that the data in the train and test set had almost even spread out of class labels. We then checked the distribution of length of tweets in the train and test set.Clearly, most of the tweets have the length between 65 and 95. It seems to be skewed for both train and test set, with minimum 0, maximum 120 and with mean around 80. This would help us during another hypothesis of word count.



Support Vector Machine:

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes as shown in the plot below:



This article has more information about SVM.

Logistic Regression:

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as the dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

The following equation is used in Logistic Regression:

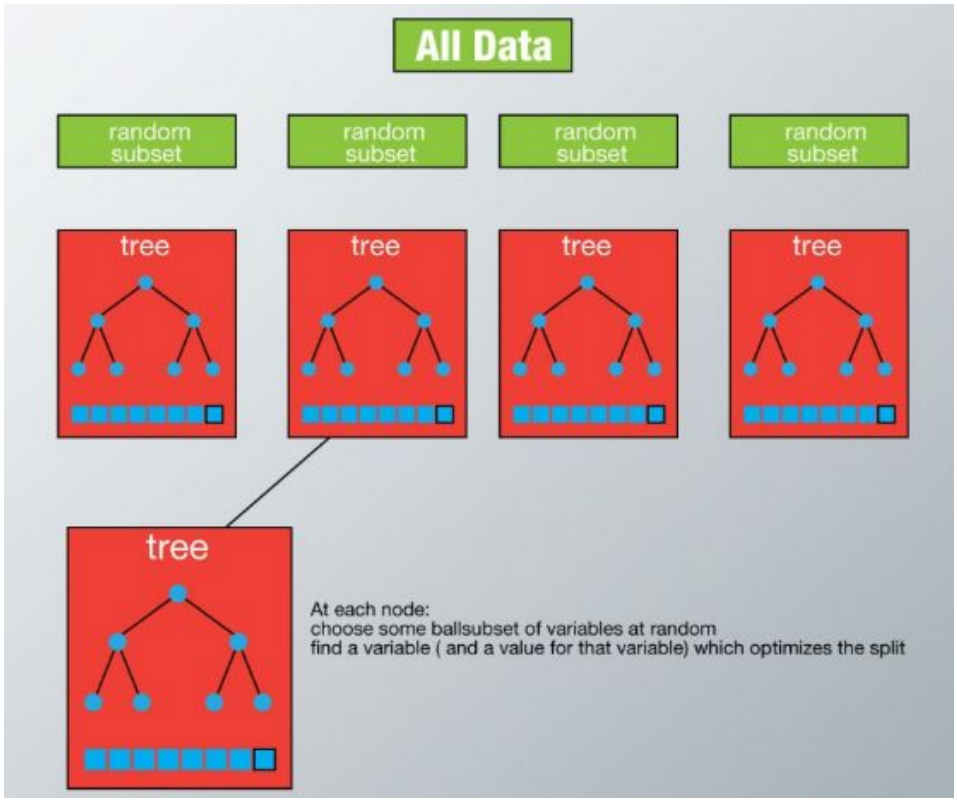
$$\log\left(\frac{p}{1-p}\right) = \beta_o + \beta(\text{Age})$$

This article includes more about Logistic Regression.

Random Forest Classifier:

Random Forest is a versatile machine learning algorithm capable of performing both regression and classification tasks. It is a kind of ensemble learning method, where a few weak models combine to form a powerful model. In Random Forest, we grow multiple trees as opposed to a decision single tree. To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).It works in the following manner. Each tree is planted & grown as follows:

- Assume the number of cases in the training set is N. Then, a sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
- If there are M input variables, a number m (m<M) is specified such that at each node, m variables are selected at random out of the M. The best split on these m variables is used to split the node. The value of m is held constant while we grow the forest.
- Each tree is grown to the largest extent possible and there is no pruning.
- Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).



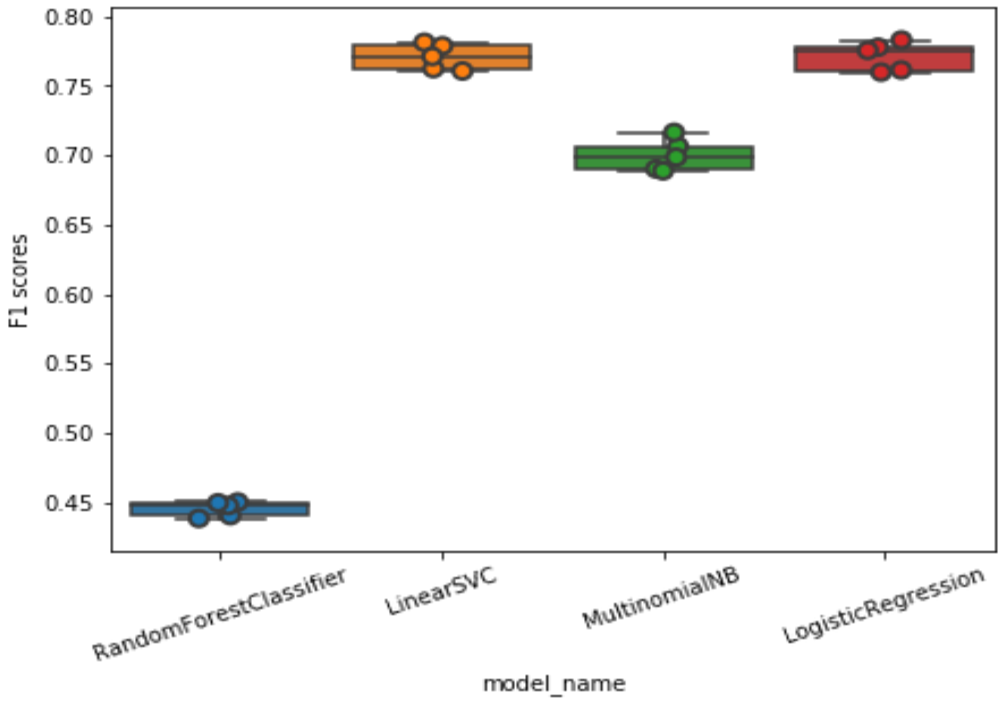
More on this article

MultiNomial-NaiveBayes Classifier:

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

Modelling:

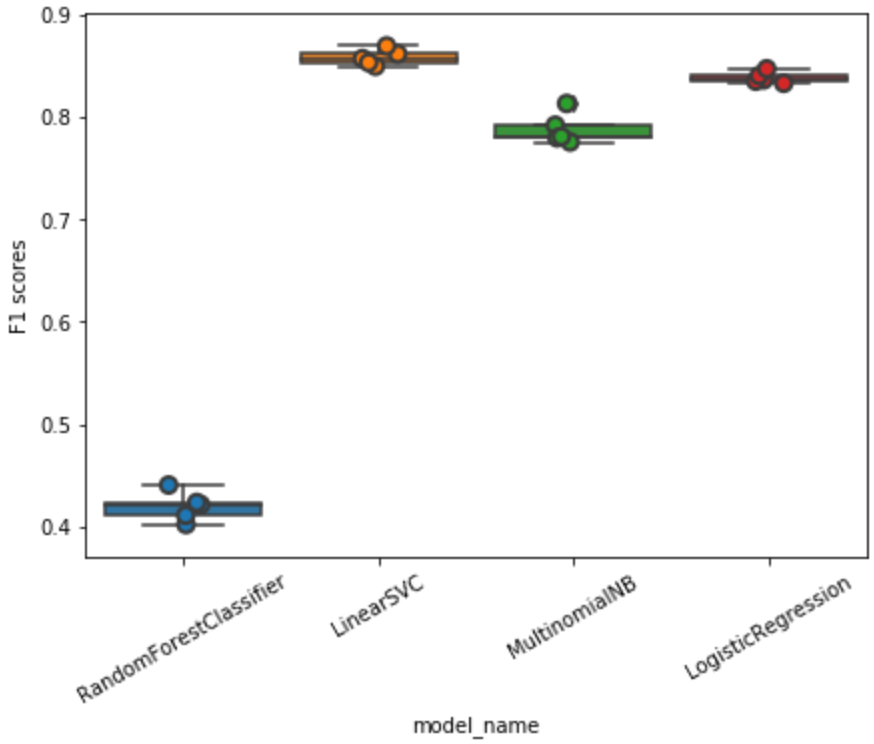
Modelling using Bag of Words Features: It is clear at first glance that all the models gave F1 score of less than 80% on Bag of Words Features. Random Forest Classifier did not perform well at all.




```
F1 scores of different models on Bag of Words (BoW) Features

model_name
LinearSVC                0.771260
LogisticRegression       0.775601
MultinomialNB            0.698686
RandomForestClassifier   0.447707
Name: f1, dtype: float64
```

LogisticRegression gave the highest F-1 score of 77.5%. LinearSVC gave the second highest score of 77.1%

Modelling using TF-IDF Features: Again, Random Forest Classifier did not perform well at all. It is clear that other models have now performed well with F-1 score more than 80% now.

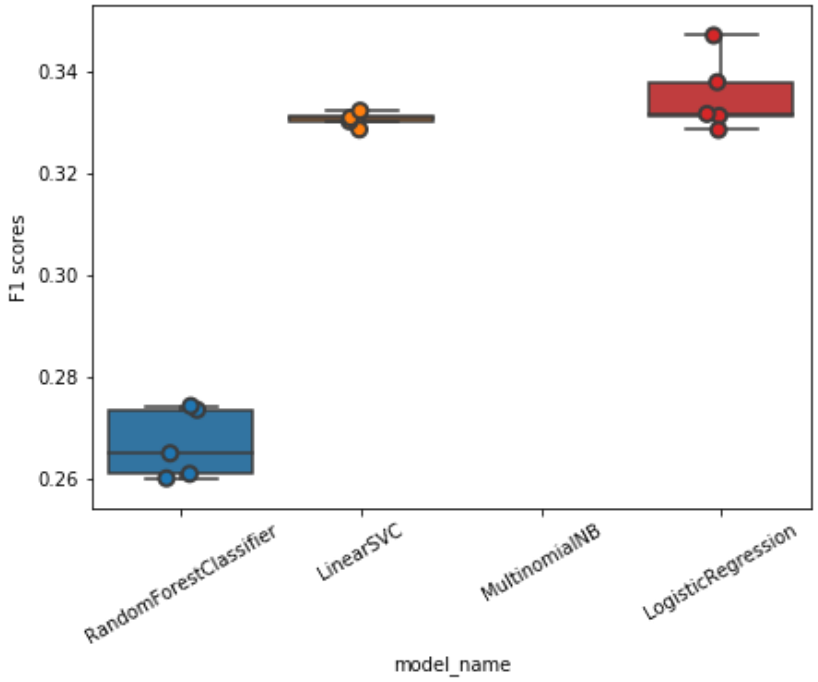



```
F1 scores of different models on TF-IDF Features

model_name
LinearSVC                0.857309
LogisticRegression       0.836482
MultinomialNB            0.781035
RandomForestClassifier   0.421456
Name: f1, dtype: float64
```

Linear SVC model gave us the highest F-1 score of 85.75%. It performed 8% higher on TF-IDF features than that on bag of words features.

Uptil now, LinearSVC on TF-IDF has performed the best.

Modelling on Word2Vec Features: Surprisingly, all the models performed the worst on Word2Vec Features. All F-1 scores are less than 35%



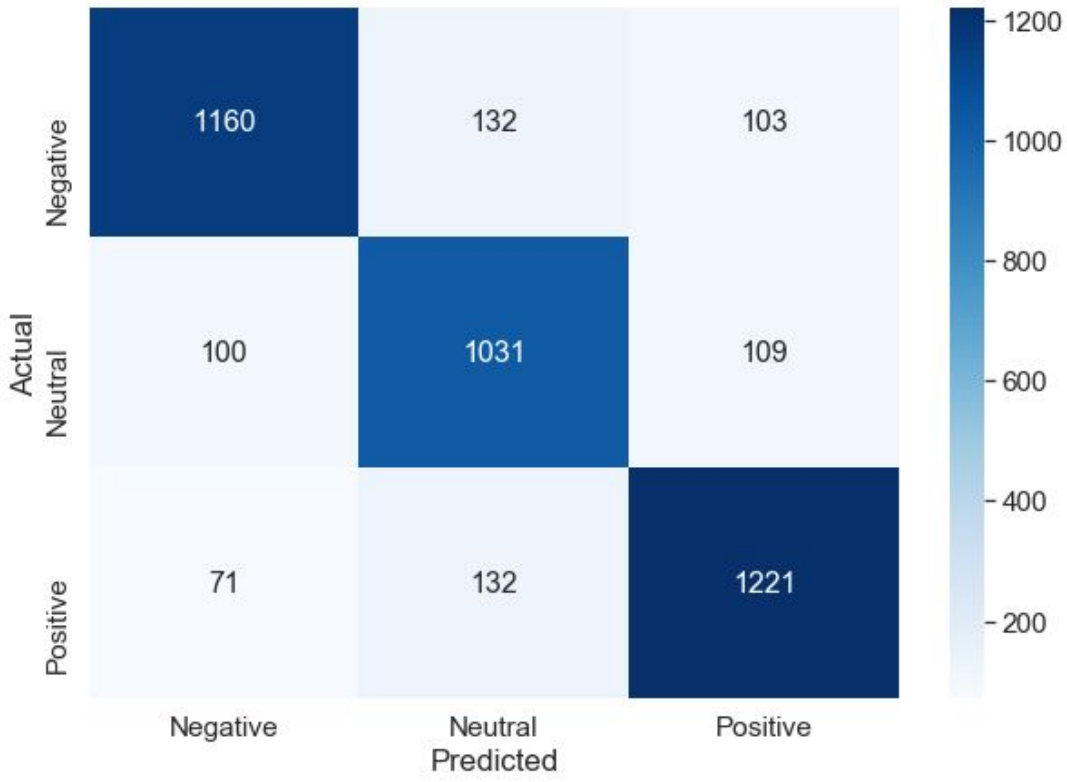

```
F1 scores of different models on Word2Vec Features

model_name
LinearSVC                0.330869
LogisticRegression       0.331635
MultinomialNB                 NaN
RandomForestClassifier   0.264882
Name: f1, dtype: float64
```

Comparison

model_name	bow	tfidf	w2v
LinearSVC	0.771260	0.857309	0.330869
LogisticRegression	0.775601	0.836482	0.331635
MultinomialNB	0.698686	0.781035	NaN
RandomForestClassifier	0.447707	0.421456	0.264882

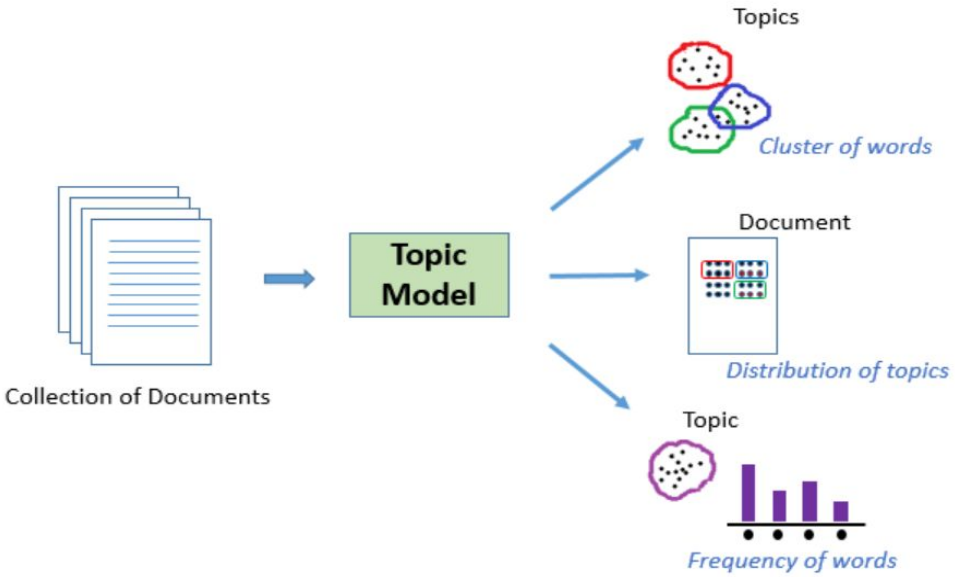
Linear SVC on TF-IDF features gave us the highest F1 score of 85.73%. Amongst, the algorithms we modelled, LinearSVC proved to classify the sentiment of the tweets better than the rest. Hence, we can use the Linear SVC on TF-IDF Features of tweets to classify them into positive, negative or neutral classes, and it will do so with an F-1 score of 85.73% which is not bad!

```
Accuracy :  0.8107932287827527
F1 :  0.8408521468204391
Recall :  0.8406011332840602
Precision :  0.8416657892256071
Classification Report :
              precision    recall  f1-score   support

    Negative       0.87      0.83      0.85      1395
     Neutral       0.80      0.83      0.81      1240
    Positive       0.85      0.86      0.85      1424

    accuracy                           0.84      4059
   macro avg       0.84      0.84      0.84      4059
weighted avg       0.84      0.84      0.84      4059
```

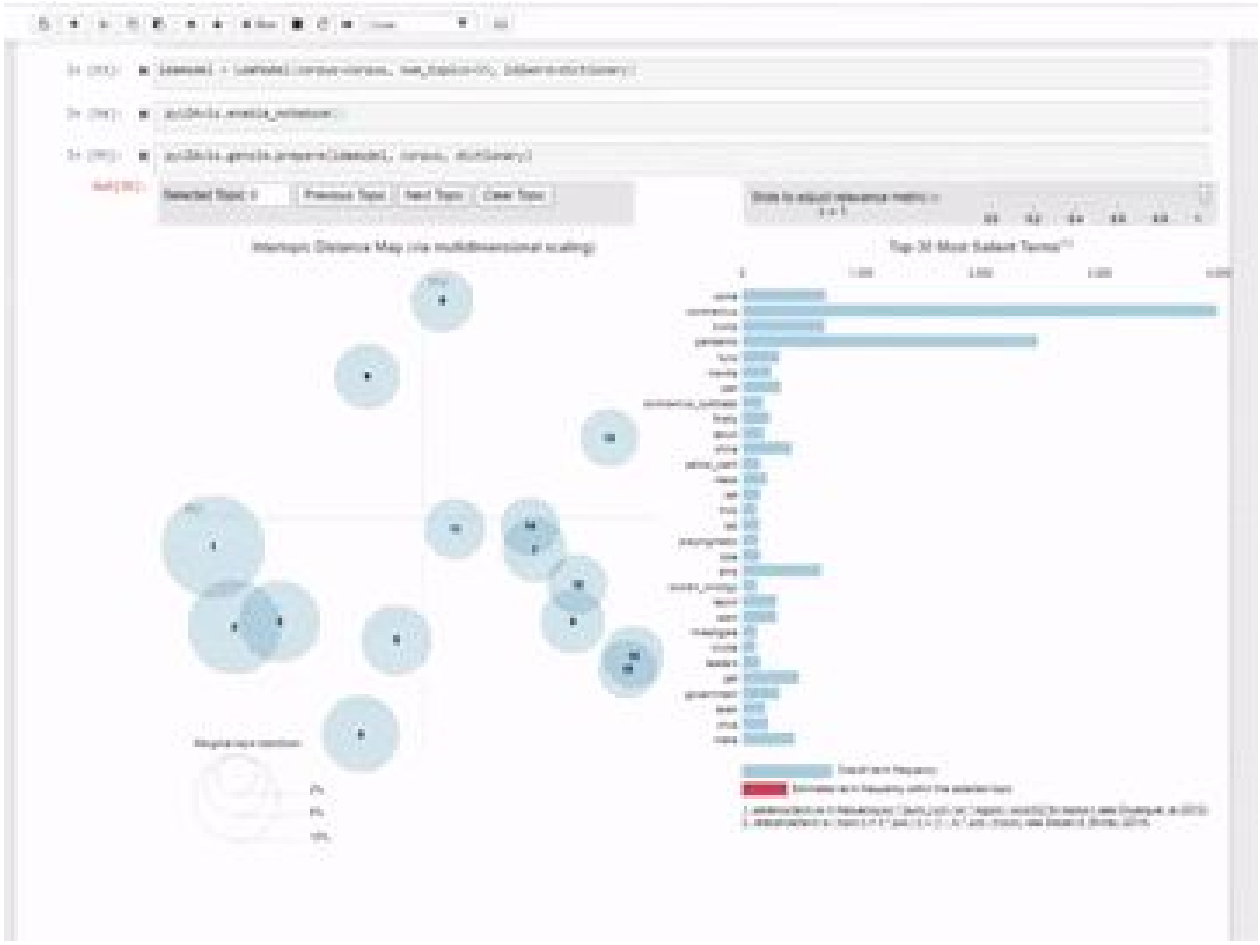


Topic Modelling - LDA

LDA stands for Latent Dirichlet Allocation. As the name suggests, Topic Modeling is a process to automatically identify topics present in a text object and to derive hidden patterns exhibited by a text corpus. Credits



To visualize our topics in a 2-dimensional space we will use the **pyLDAvis library**. This visualization is interactive in nature and displays topics along with the most relevant words.



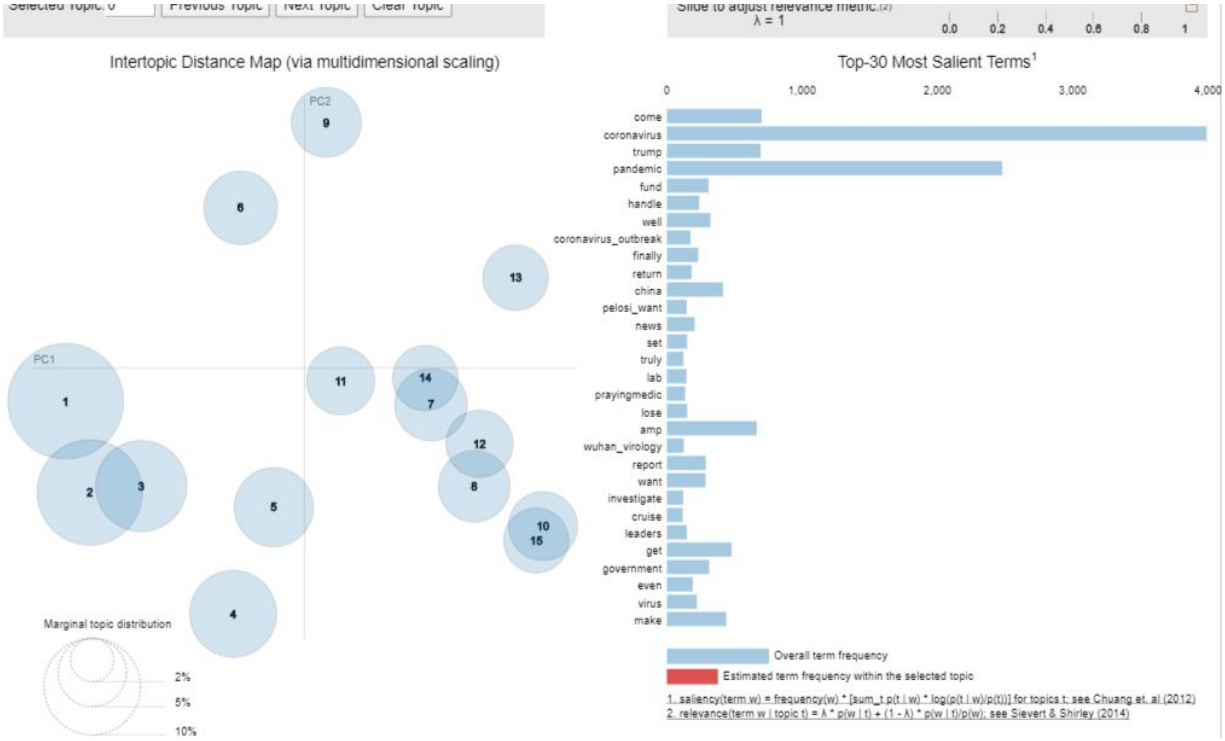
For clearer view: https://www.loom.com/share/d6456ef28c2c43c889341891584aefb4

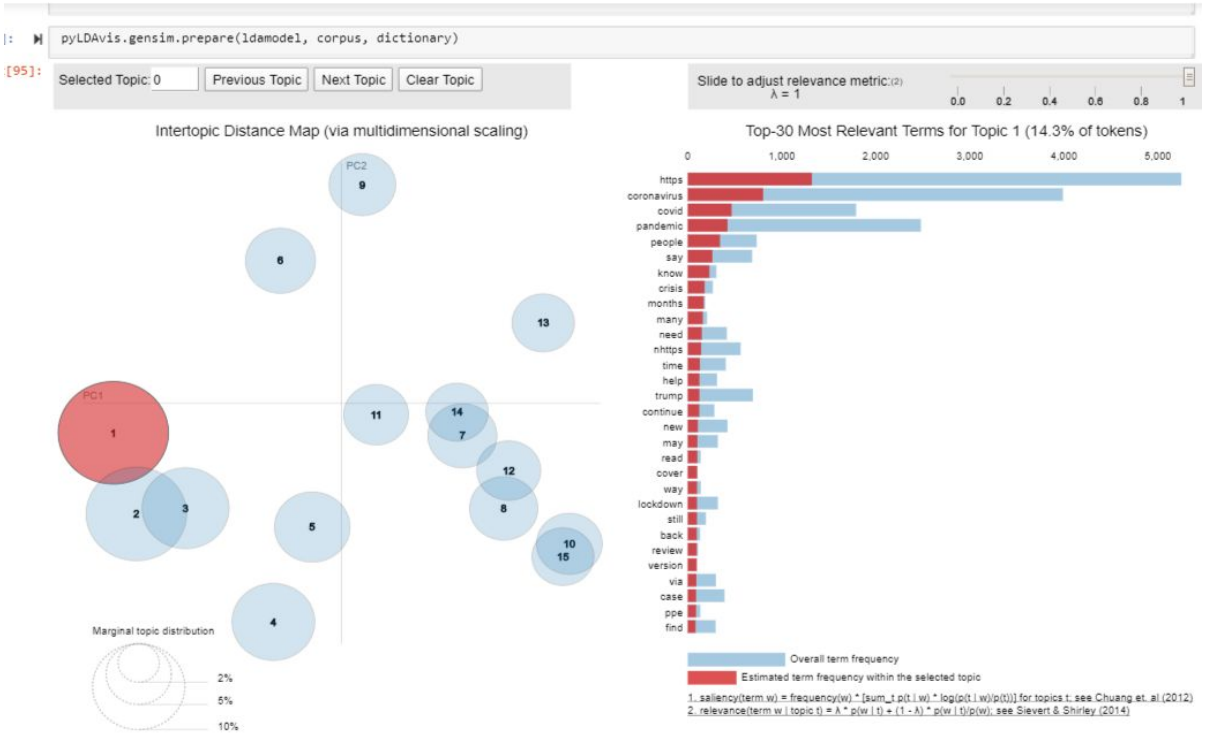
Interpreting;

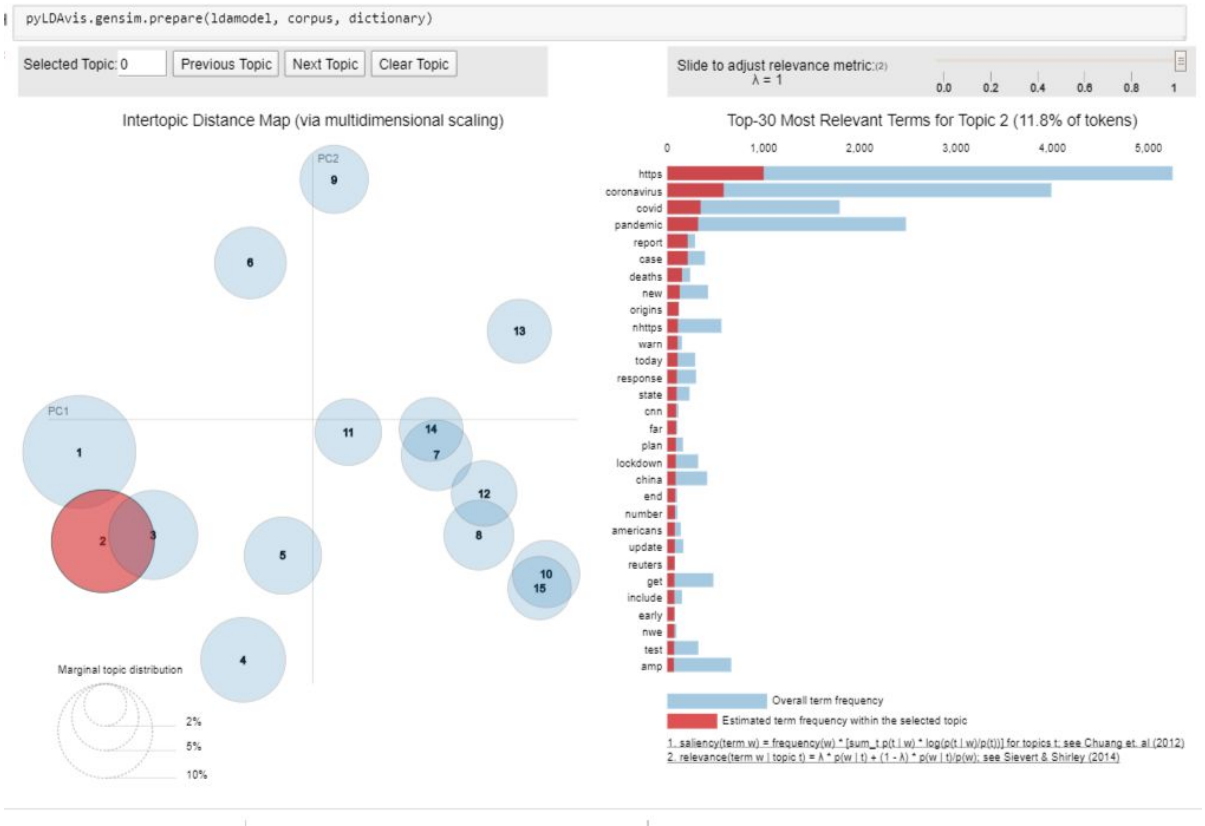
- The Left side shows the multi-dimensional "word-space" superimposed on two "Principal components" and the relative positions of all the topics.
- The size of the circle represents what % of the corpus it contains.
- The right side shows the word frequencies within the topic and in the whole corpus.

PC stands for ‘Principal Components’. It is a statistical technique. It extracts low dimensional set of features by taking a projection of irrelevant dimensions from a high dimensional data set with a motive to capture as much information as possible. With fewer variables obtained, while minimising the loss of information, visualization also becomes much more meaningful. And for a human mind, it is difficult to interpret numbers at first glance but can easily interpret colors and

shapes. This is exactly what LDA did for us. It took a bunch of words and found hidden patterns and we visualized it in a 2-D space. The topics from the LDA model are stored in a form of matrix. Further, these topics/components can serve as an input to machine learning algorithms for better results. It obviously saves a ton of space and processing to work with fewer numbers of components.




pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)
[95]:
Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0 0.2 0.4 0.6 0.8 1
Intertopic Distance Map (via multidimensional scaling)
PC1
PC2
Marginal topic distribution
2%
5%
10%
Top-30 Most Relevant Terms for Topic 1 (14.3% of tokens)
0 1,000 2,000 3,000 4,000 5,000
https
coronavirus
covid
pandemic
people
say
know
crisis
months
many
need
nhttps
time
help
trump
continue
new
may
read
cover
way
lockdown
still
back
review
version
via
case
ppe
find
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)


pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)
Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0 0.2 0.4 0.6 0.8 1
Intertopic Distance Map (via multidimensional scaling)
PC1
PC2
Marginal topic distribution
2%
5%
10%
Top-30 Most Relevant Terms for Topic 2 (11.8% of tokens)
0 1,000 2,000 3,000 4,000 5,000
https
coronavirus
covid
pandemic
report
case
deaths
new
origins
nhttps
warn
today
response
state
cnn
far
plan
lockdown
china
end
number
americans
update
reuters
get
include
early
nwe
test
amp
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Results:

- We believed that the global quarantine efforts would lead to a higher percentage of negative tweets due to the associated struggles, but overall sentiment on twitter suggests otherwise.
- Frequent words in positive tweets are not focused on the coronavirus situation per se. However, negative tweets are focused on the pandemic situation, in fact, some negative tweets have words like government, intel, chinese etc.
- Linear SVC with TF-IDF features had the highest F1 score of 85.73% for multiclass classification of tweets as positive, negative or neutral.
- The LDA model gave us the topics/components to focus on. These topics can be further fed into advanced ML algorithms.

Account Creation Date and Word Count

Hypothesis

This is the first dataset we analyzed for this project. To begin with, we wanted to start with a fairly simple analysis of twitter variables that we had not previously used in class. After looking through what the API offered, we settled on Account Creation Date and Word Count. In relation to account creation date, we believed that newer accounts would have more sentimental tweets. Our reasoning was that newer accounts were created by younger users, thus leading to more argumentative and less evidence based tweets. In regards to word count, we believed that longer tweets would lead to higher polarity as there would be more information to dispute in these tweets. To begin, we first focused on collecting enough tweets to support our analysis.

Data Collection

Once we had settled upon our first research topic, we needed to decide how many tweets to analyze. For this particular data set, we chose to pull 2000 tweets from all around the world. The two terms we used were “Covid” and “Corona” within the body of the tweet. We included all usual variables in the search results, but added the user since date (the account creation date), word count, and unique word count. To do this, we needed to separate the account creation data from the time as Twitter includes them both in one field.

Analysis and Results

To first analyze account creation date, we needed to convert it to a measurable value. We began this process by importing the datetime library. We first used the astype method from the dataframe library to convert the date to an integer. After that, we conducted some research into the best way to convert dates into a single value. We discovered ordinal time, and used the code provided here as a basis to convert our integer values into this. Below is our screenshot of our process and some of the results.

```
In [9]: #converting account creation date to datetime

        df['usersincedate'] = df['usersincedate'].astype('datetime64[ns]')
```

```
In [15]: #converting datetime to ordinal
         #https://stackoverflow.com/questions/53893323/how-to-convert-pandas-data-frame-datetime-column-to-int

         df['usersincedate'] = df['usersincedate'].apply(lambda x:x.toordinal())
```

```
In [16]: df
```

Out[16]:

text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	count_sent	count_word	count_unique_word	usersincedate	usersincetime
lDonaldTrump elieves killing 70,0...	0	None	7	10	Boca Raton, FL	0.000000	-0.100000	1	16	16	736436	00-41-42
@bessbell: My an ICU doctor treati...	48841	None	800	1822	Singapore	0.375000	0.125000	1	26	26	732826	13-12-36
b'RT as1774Paine: eaked dossier conclu...	29	None	8907	8524	Austin	1.000000	0.000000	1	17	17	734874	17-14-21
ee COVID-19 is available to ALL Lo...	0	None	5458	673	South Los Angeles, Los Angeles	0.600000	0.400000	1	18	16	735228	22-33-20
b'RT evelandPolice: N SCAMS TO ATCH OUT ...	1	None	2703	3117	NaN	0.454545	0.136364	1	21	21	733821	15-15-04

Once we had converted the values, we moved onto finding relationships between the variables. We first started by trying to answer our initial research question about whether Account Creation Date would affect Polarity. As we received a correlation coefficient of around -0.01, it was determined that no such relationship exists. We moved onto Subjectivity just to try it out, but we received an even lower correlation of -0.002. These results indicated that Account Creation Date is not a predictor of Sentimality.

```
In [26]: print("Polarity vs Account Creation Date Correlation:")
         np.corrcoef(df.usersincedate, df.Polarity)
```

```
Polarity vs Account Creation Date Correlation:
```

```
Out[26]: array([[ 1.        , -0.01019287],
                [-0.01019287,  1.        ]])
```

```
In [18]: print("Subjectivity vs Account Creation Date Correlation:")
         np.corrcoef(df.usersincedate, df.Subjectivity)
```

```
Subjectivity vs Account Creation Date Correlation:
```

```
Out[18]: array([[ 1.        , -0.00243186],
                [-0.00243186,  1.        ]])
```

Next, we moved onto analyzing Word Count. While the relationships here were still not very strong, they were more related than Account Creation Date. Word Count and Polarity had a low correlation of 0.02, but the correlation jumped up to 0.17 for Word Count and Subjectivity. Even though it is relatively higher, this coefficient is not enough to say that a relationship exists as it falls far below the standard of 0.7. These results indicate that both of our original

hypotheses were wrong, and that no relationship exists between either Word Count or Account Creation Date and Sentimality. Included below is a visualization of the strongest relationship we found, Word Count vs Subjectivity. Due to the low correlation though, it is not a good predictor at all.

In [20]:
```
print("Subjectivity vs Word Count Correlation:")
np.corrcoef(df.count_word, df.Subjectivity)
```

```
Subjectivity vs Word Count Correlation:

array([[1.        , 0.17368737],
       [0.17368737, 1.        ]])
```

Out[20]:

In [23]:
```
print("Followers vs Word Count Correlation:")
np.corrcoef(df.followers, df.count_word)
```

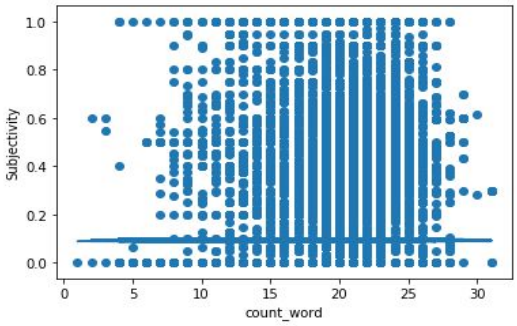
```
Followers vs Word Count Correlation:

array([[ 1.        , -0.01918361],
       [-0.01918361,  1.        ]])
```

Out[23]:

In [30]:
```
#Visualizing the strongest relationship, Word Count vs Subjectivity

y = df.Subjectivity #outcome variable
x = df.count_word #predictor
x = sm.add_constant(x)
lr_model = sm.OLS(y,x).fit()
print(lr_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Subjectivity   R-squared:                       0.030
Model:                            OLS   Adj. R-squared:                  0.030
Method:                 Least Squares   F-statistic:                     382.5
Date:                Wed, 06 May 2020   Prob (F-statistic):          6.53e-84
Time:                        13:57:10   Log-Likelihood:                -2871.5
No. Observations:               12300   AIC:                             5747.
Df Residuals:                   12298   BIC:                             5762.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0886      0.013      7.042      0.000       0.064       0.113
count_word     0.0123      0.001     19.559      0.000       0.011       0.014
==============================================================================
Omnibus:                      998.920   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              935.283
Skew:                           0.612   Prob(JB):                    8.05e-204
Kurtosis:                       2.430   Cond. No.                         91.6
==============================================================================
```

```
In [31]: plt.scatter(df.count_word, df.Subjectivity)
         plt.plot(df.count_word, 0.0123 * df.Subjectivity + 0.0886)
         plt.xlabel('count_word')
         plt.ylabel('Subjectivity')
         np.corrcoef(df.count_word, df.Subjectivity)[0,1]

Out[31]: 0.17368736699233153
```



Confirmed Cases by State

Hypothesis

For our next data set, we wanted to see if states with more confirmed cases would have higher Polarity among tweets. Our reasoning was that as states increased in cases, the issues surrounding Covid would become more widely discussed, and thus more controversial. We believed that finding the average polarity of tweets in a state and comparing it to the number of confirmed cases would support our hypothesis.

Data Collection

Out of all our data sets, this required the most extensive collection phase. We first started on the most challenging aspect, collecting tweets by state. Unfortunately, we were unable to figure out how to do this through the Twitter API. To get around this, we had to resort to some creativity. First, we added the state's name to the search terms (along with the original search terms of Covid and Corona). Using this method both had its downsides and upsides. On a positive note, including the state name led to tweets that were discussing issues relating to Covid and the state, which actually fit this topic. Unfortunately, it did not guarantee that the tweet was created by an account in the state as people sometimes discussed states that they did not live in.

```
In [32]: !python ./twitter_search.py "Covid Corona Kansas"
```

To resolve this issue, we added the Location variable to our Twitter results. After doing this, we realized that Location is user defined. This means that we could not simply filter out the

state's name as people would write the abbreviation or just their city. What we ended up doing was putting all of the tweets in an Excel sheet and manually filtering out related location terms. Due to the work intensive nature of this methodology, we limited each state to 100 tweets each and focused on 10 states, spread out as evenly as possible across the country. We then separated each state's tweets into its own sheet, and then used Excel to find the average Polarity. While this is not the wide dataset we wanted, we did discover correlation (which will be discussed later on) so we were satisfied by the results.

For the number of confirmed cases, we pull the data from John Hopkins University Hospital. While we had to manually type it into an Excel sheet, they had confirmed cases for each state up to the current day. As shown below, we combined this with the Polarity data we had
found before.

	A	D	J
1	location	text	Polarity
2	Utah	b'Utah federal judge enters injunction ha	0
3	Farmington, Utah	b'RT @UtahCoronavirus: Testing For COV	0.45
4	Salt Lake City, UT	b"CDC announced three new COVID-19 s	0.136363636
5	Utah, USA	b'RT @Utah_DVMA: Veterans and their fa	0
6	Salt Lake City, UT	b'RT @utahalumni: Join the SLC Alumni C	0.357142857
7	Utah	b'Dr. Dunn on going forward with #COVID	0
8	Salt Lake City, Utah USA	b'Funding is also available through @Arts	0.2
9	Salt Lake City, UT	b'More info on motor vehicle services in	0.5
10	Sandy, Utah	b'Hill Air Force Base is doing a statewide	-0.8
11	Salt Lake City, Utah	b'RT @UtahCoronavirus: We have four st	0.8
12	Salt Lake City, Utah	b'RT @UtahCoronavirus: Veterans need t	0.125
13	Utah, USA	b'Important info. The state of Utah is nov	0.4
14	Salt Lake City, Utah	b'RT @edcutahorg: We\xe2\x80\x99re gl	0.4
15	Salt Lake City, UT	b'We\xe2\x80\x99re glad to see the COV	0.5
16	Salt Lake City, Utah	b'RT @RobertGehrke: Today a federal juc	0
17	Utah, USA	b'RT @UtahUnclaimProp: We encourage	0
18	Salt Lake City, Utah	b'RT @AlisonT23674586: Learn how one h	0.16
19	Salt Lake City, Utah	b"RT @abc4utah: UTAH'S DAILY CORONAV	0

	A	B	C
1	State	Polarity	Cases
2	Maine	0.070479	1000.00
3	Kansas	0.136036	3700.00
4	Utah	0.112762	4000.00
5	Virginia	0.071925	14300.00
5	California	0.016864	46200.00

Analysis and Results

While it is important to keep in mind that this is an extremely small dataset that may not represent a general trend, from the data we gathered there is a strong relationship between polarity and confirmed cases in a state. With the exception of Maine, the polarity goes down as cases go up. When performing the analysis, we found a strong correlation of -0.82. Additionally, we were able to calculate a regression line with equation: Polarity = -1.994e-06 * Cases + 0.1092.

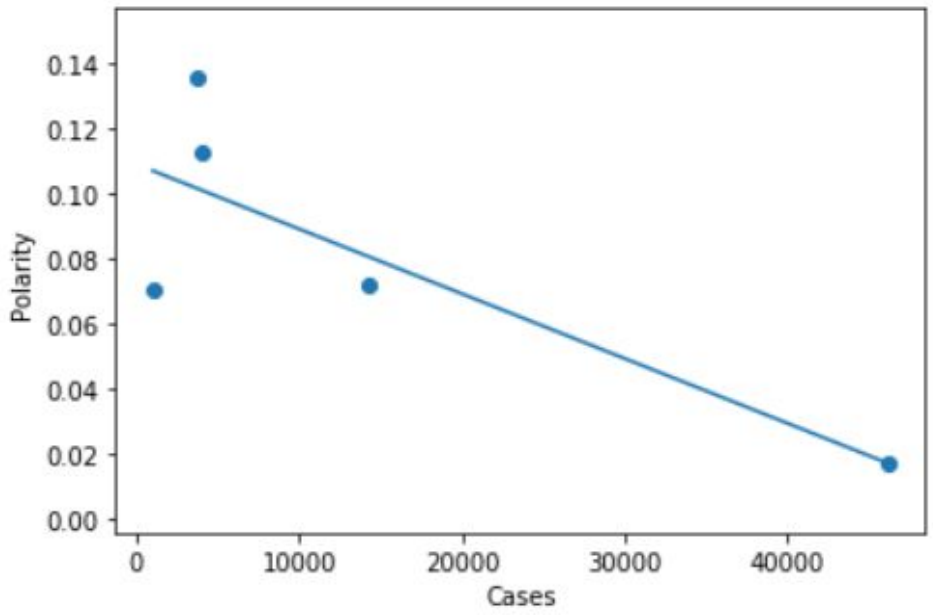
This data that we gathered actually contradicts our hypothesis, which was that there would be a positive correlation between Polarity and Confirmed Cases. It appears that as cases increase, there is less disagreement within tweets. A possible explanation for this is that the rising number of cases actually unifies the state around the issue. Another viable reasoning is that as cases increase, so does visibility. This leads to more consistent information and viewpoints being shared among the members of the state. Regardless of the reasoning, it was interesting to find such a strong correlation after our previous datasets had very weak relationships.

```
#finding regression data for Case Count vs Polarity

y = df2.Polarity #outcome variable
x = df2.Cases #predictor
x = sm.add_constant(x)
lr_model = sm.OLS(y,x).fit()
print(lr_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Polarity   R-squared:                       0.673
Model:                            OLS   Adj. R-squared:                  0.564
Method:                 Least Squares   F-statistic:                     6.169
Date:                Wed, 06 May 2020   Prob (F-statistic):             0.0890
Time:                        14:57:31   Log-Likelihood:                 11.688
No. Observations:                   5   AIC:                            -19.38
Df Residuals:                       3   BIC:                            -20.16
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1092      0.017      6.249      0.008       0.054       0.165
Cases      -1.994e-06   8.03e-07     -2.484      0.089   -4.55e-06    5.61e-07
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   2.210
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.164
Skew:                        -0.136   Prob(JB):                        0.921
Kurtosis:                     2.157   Cond. No.                     2.82e+04
==============================================================================
```

```
#Plotting regression and calculating correlation coefficient

plt.scatter(df2.Cases, df2.Polarity)
plt.plot(df2.Cases, -1.994e-06 * df2.Cases + 0.1092)
plt.xlabel('Cases')
plt.ylabel('Polarity')
np.corrcoef(df2.Cases, df2.Polarity)[0,1]
```

```
-0.8202599838525004
```



Appendix

```
#!pip install gensim

#!pip install plotly

#!pip install GetOldTweets3

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
%matplotlib inline
import seaborn as sns
import pyLDAvis
import pyLDAvis.gensim
import gc
import time
import datetime
import warnings
import re
import plotly.express as px
import plotly.graph_objects as go

import GetOldTweets3 as got
from datetime import datetime, timedelta

from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.corpus import wordnet

import gensim
from gensim.models import CoherenceModel, LdaModel, LsiModel, HdpModel
from gensim.models.wrappers import LdaMallet
from gensim.corpora import Dictionary

from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.validation import check_X_y, check_is_fitted
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import log_loss
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from scipy import sparse

#settings
warnings.filterwarnings("ignore")
lem = WordNetLemmatizer()
tokenizer=ToktokTokenizer()

from textblob import TextBlob
```

Data Collection:

```
import os
os.system("python ./twitter_search2.py covid-19 -c 5000")
```

```
64]: 0
```

```
os.system("python ./twitter_search2.py coronavirus -c 5000")
```

```
65]: 0
```

```
os.system("python ./twitter_search2.py pandemic -c 5000")
```

```
29]: 0
```

```
df.shape
```

```
2]: (12300, 14)
```

Data PreProcessing:

Removing unwanted patterns

```
#function to remove unwanted text patterns from the tweets.

def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)
    return input_txt
```

Removing user handles

```
#  Removing Twitter Handles (@user)

df['tidy_tweet'] = np.vectorize(remove_pattern)(df['text'], "@[\w]*")
```

Removing Punctuations, Special characters and numbers

```
#Removing Punctuations, Numbers and Special Characters

#Here we will replace everything except characters and hashtags with spaces.
#The regular expression "[^a-zA-Z#1-9]" means anything except alphabets and '#'.

df['tidy_tweet'] = df['tidy_tweet'].str.replace("[^a-zA-Z#]", " ")
```

Removing Short Words

```
#Removing Short Words: less than 3 alphabets. Eg: hi, oh, hmm

df['tidy_tweet'] = df['tidy_tweet'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))
```

Convert to Lower Case

```
# Lower Casing

df['tidy_tweet']  = df['tidy_tweet'].str.lower()
```

Remove StopWords

```
#Importing stopwords from nltk library

from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))

# Function to remove the stopwords
def stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

# Applying the stopwords to 'text_punct' and store into 'text_stop'
df["tidy_tweet"] = df["tidy_tweet"].apply(stopwords)
```

Remove Common words

```
#We can also remove commonly occurring words from our text data First
#Let's check the 15 most frequently occurring words in our text data.


# Checking the first 10 most frequent words
from collections import Counter
cnt = Counter()
for text in df["tidy_tweet"].values:
    for word in text.split():
        cnt[word] += 1

cnt.most_common(15)
```

```
50]: [('https', 4649),
 ('covid', 3326),
 ('pandemic', 3144),
 ('coronavirus', 1571),
 ('people', 822),
 ('#covid', 743),
 ('trump', 654),
 ('president', 480),
 ('nhttps', 414),
 ('cases', 367),
 ('time', 344),
 ('government', 340),
 ('world', 335),
 ('want', 334),
 ('health', 324)]
```

Remove Rare Words

```
# Removal of 15 rare words

freq_less = pd.Series(' '.join(df['tidy_tweet']).split()).value_counts()[-15:] # 15 rare words

freq_less
```

```
#teamindiainventors    1
voucher                1
cautions               1
doysqgv                1
libe                   1
thewrightstuff         1
insular                1
vajfsaq                1
rhmw                   1
gxsndby                1
ngejmpfvyz             1
cremation              1
ereofda                1
vxnnfbnnnx             1
lecturers              1
dtype: int64
```

```
df['tidy_tweet'] = df['tidy_tweet'].apply(lambda x: " ".join(x for x in x.split() if x not in freq_less))
```

Convert Emojis to Text

```
from emot.emo_unicode import UNICODE_EMO, EMOTICONS
#!pip install emoji
import emoji

# Converting emojis to words
def convert_emojis(text):
    for emot in UNICODE_EMO:
        text = emoji.demojize(text)
        #text = text.replace(emot, "_".join(UNICODE_EMO[emot].replace(",","").replace(":","").split()))
        return text
```

```
#example

text1 = "Hilarious 😂"
print(convert_emojis(text1))
```

```
Hilarious :face_with_tears_of_joy:
```

```
#Applying the remove emoji function

df['tidy_tweet'] = df['tidy_tweet'].apply(convert_emojis)
```

Remove URLs

```
# Function for url's

def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

# Examples

text = "This is my website, https://www.abc.com"
remove_urls(text)
```

```
9]: 'This is my website, '
```

```
#Passing the function to 'text_rare'
df['tidy_tweet'] = df['tidy_tweet'].apply(remove_urls)
```

Tokenization

```
#Creating function for tokenization
def tokenization(text):
    text = re.split('\W+', text)
    return text

df['token_tweet'] = df['tidy_tweet'].apply(lambda x: tokenization(x)
```

Stemming

```
#Stemming

from nltk.stem.porter import *
stemmer = PorterStemmer()

stem_tweet = df['token_tweet'].apply(lambda x: [stemmer.stem(i) for i in x])

#Now let's stitch these tokens back together. It can easily be done using nltk's MosesDetokenizer function.

for i in range(len(stem_tweet)):
    stem_tweet[i] = ' '.join(stem_tweet[i])

df['stem_tweet'] = stem_tweet
```

Lemmatization

```
#Lemmatizing

import nltk
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# Pos tag, used Noun, Verb, Adjective and Adverb
wordnet_map = {"N":wordnet.NOUN, "V":wordnet.VERB, "J":wordnet.ADJ, "R":wordnet.ADV}

# Function for lemmatization using POS tag
def lemmatize_words(text):
    pos_tagged_text = nltk.pos_tag(text.split())
    return " ".join([lemmatizer.lemmatize(word, wordnet_map.get(pos[0], wordnet.NOUN)) for word, pos in pos_tagged_text])

# Passing the function to 'text_rare' and store in 'text_lemma'
df["lemm_tweet"] = df["tidy_tweet"].apply(lemmatize_words)
```

Sentiment Analysis using TextBlob

The tweetCriteria in the screenshot should not be on the next line.That was done to fit the width.
Code:

tweetCriteria = got.manager.TweetCriteria().setQuerySearch("corona virus pandemic covid").setSince(start_date.strftime('%Y-%m-%d')).setUntil((start_date + timedelta(days=1)).strftime('%Y-%m-%d')).setMaxTweets(100)
tweets = got.manager.TweetManager.getTweets(tweetCriteria)

```
#Collecting 100 tweets for each day from Feb 11, 2020 to May 08, 2020 (excluded)

posPercent = []
negPercent = []
numTweets = []
dates = []

#set start date, strip time because the 'created_at' is datetime object. We only need date.


start_date = datetime.strptime('2020-02-11', '%Y-%m-%d')

while start_date != datetime.strptime('2020-05-08', '%Y-%m-%d'): #runs till date May 08, 2020

    dates.append(start_date) #append dates (used to plot)

    #search for keywords. strip time. Max tweets 100.
    #tried for 150-500 tweets, was taking a lot of time and had to interrupt kernel so stuck with 100.
    #using Kafka mightbe a work around, but I am still learning it. Hence, stuck with 100

    tweetCriteria = got.manager.TweetCriteria().setQuerySearch("corona virus pandemic covid")
        .setSince(start_date.strftime('%Y-%m-%d')).
        setUntil((start_date + timedelta(days=1)).strftime('%Y-%m-%d')).setMaxTweets(100)
    tweets = got.manager.TweetManager.getTweets(tweetCriteria)

    #incremented days by 1. SO for sure, we get tweets for each day related to our keywords.

    ptweet = 0
    ntweet = 0
    for tweet in tweets:

        #remove unnecessary patterns

        text = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t]) |(\w+:\/\/\S+)", " ", tweet.text).split())

        #count number of positive and negative tweets.

        ptweet += (TextBlob(text).sentiment.polarity > 0.2)
        ntweet += (TextBlob(text).sentiment.polarity < -0.2)

    #calculate percent of positive and negative tweets

    posPercent.append(100*ptweet/len(tweets))
    negPercent.append(100*ntweet/len(tweets))
    numTweets.append(len(tweets))

    start_date += timedelta(days=1)
```

```
data = pd.DataFrame(list(zip(dates, numTweets, posPercent, negPercent)), columns=['Date','Num Tweets', '% Pos', '% Neg'])
fig = go.Figure()

fig.add_trace(go.Scatter(x=data.Date, y=data['% Pos'],
                    mode='lines',
                    name='% Pos'))
fig.add_trace(go.Scatter(x=data.Date, y=data['% Neg'],
                    mode='lines',
                    name='% Neg'))
fig.update_layout(title='Twitter Sentiment Analysis Coronavirus Response',
                  xaxis_title='Date',
                  yaxis_title='Percentage')
fig.show()
```

For story generation and visualization of tweets, we added the Subjectivity and Polarity in our dataframe

```
subj_list= []
polarity_list = []

for index, row in df.iterrows():
    tweet = row["text"]
    text = TextBlob(tweet) #convert tweet to TextBlob object

    #sentiment analysis
    subjectivity = text.sentiment.subjectivity # 0 to 1
    polarity = text.sentiment.polarity # -1 to 1

    #print(tweet, subjectivity, polarity)

    subj_list.append(subjectivity)
    polarity_list.append(polarity)

df['Subjectivity'] = subj_list
df['Polarity'] = polarity_list
```

```
df.sample(4)
```

]:

created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	count_sent	count_word	count_uniqu
5/4/2020 15:09	8/2/2010 9:49	b'TOGETHER UNITED\nCoronavirus means we really...	0	None	591	1336	Guelph, ONT. Canada	0.60	0.10	1	19	
5/4/2020 15:06	1/8/2019 17:05	b'We will continue to provide current informat...	2	None	696	133	NaN	0.40	0.00	1	20	
5/4/2020 15:06	6/23/2018 17:13	b'RT @DrTedros: Thank you @_AfricanUnion Ambas...	62	COVID19	2633	4999	Modjadjiskloof, South Africa	0.15	0.05	1	21	
5/4/2020 15:05	4/20/2008 19:17	b"RT @cnni: Most of Vietnam's 22 million stude...	23	None	155	88	Porto Velho-Rondônia, Brasil	0.50	0.50	1	22	

Word Cloud for Tweets

```
from wordcloud import WordCloud

all_words = ''.join([text for text in df['tidy_tweet']])

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=200).generate(all_words)

plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Frequent Words in Tweets", fontsize=20)
plt.axis('off')
plt.show()
```


Word Cloud for positive tweets

```
pos_words =' '.join([text for text in df['tidy_tweet'][df['Polarity'] > 0.4]])
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=200).generate(pos_words)
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Frequent Words in Positive Tweets", fontsize=20)
plt.axis('off')
plt.show()
```

Word Cloud for negative tweets

```
neg_words =' '.join([text for text in df['tidy_tweet'][df['Polarity'] < -0.4]])
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=200).generate(neg_words)
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Frequent Words in Negative Tweets", fontsize=20)
plt.axis('off')
plt.show()
```

Impact of Hashtags on sentiment of tweets

```
# function to collect hashtags

def hashtag_extract(x):
    hashtags = []

    # Loop over the words in the tweet
    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)
    return hashtags

# extracting hashtags from positive tweets
tw_pos = hashtag_extract(df['tidy_tweet'][df['Polarity'] > 0.4])
ht_pos = hashtag_extract(df['hashtag'][df['Polarity'] > 0.4])

# extracting hashtags from negative tweets
tw_neg = hashtag_extract(df['tidy_tweet'][df['Polarity'] < -0.4])
ht_neg = hashtag_extract(df['hashtag'][df['Polarity'] < -0.4])

# combining trends list
pos = sum(ht_pos,tw_pos)
neg = sum(ht_neg,tw_neg)

#unnesting list
pos = sum(pos, [])
neg = sum(neg, [])
```

```
print(pos[0:5])
print(neg[0:5])
```

```
['syria', 'coronavirus', 'covid', 'covid', 'narayanganj']
['coronavirusoutbreak', 'gorongosa', 'boyslocke', 'nevertrump', 'lums']
```

Bar Plot for positive tweets

```
import seaborn as sns

# plot positive hashtags

a = nltk.FreqDist(pos)
d = pd.DataFrame({'Hashtag': list(a.keys()),'Count': list(a.values())})

# selecting top 10 most frequent hashtags

d = d.nlargest(columns="Count", n = 10)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.title("Top 10 hashtags/trends in Positive Tweets", fontsize=20)
plt.xticks(rotation=30)
plt.show()
plt.tight_layout()
```

Bar Plot for negative tweets

```
#plot negative hashtags

a = nltk.FreqDist(neg)
d = pd.DataFrame({'Hashtag': list(a.keys()),'Count': list(a.values())})

# selecting top 10 most frequent hashtags

d = d.nlargest(columns="Count", n = 10)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.title("Top 10 hashtags/trends in Negative Tweets", fontsize=20)
plt.xticks(rotation=30)
plt.show()
plt.tight_layout()
```

Sentiment Analysis using Vader Sentiment Intensity Analyser

Cleaned the tweets, but did not remove the punctuations and emojis

```
#remove user handles
df['vader_tweet'] = np.vectorize(remove_pattern)(df['text'], "@[\w]*")
df.head(2)
```

80]:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_twee
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	trum believe killin american destroyin am.
1	5/4/2020 15:07	5/30/2007 13:12	b'RT @bessbell: My dad is an ICU doctor treati...	48841	None	800	1822	Singapore	0.375	0.125	...	2020-05-04	15-07-00	docto treatin covi patient past wee never.

2 rows × 27 columns

```
# remove special characters, kept punctuation like , and ! intact
df['vader_tweet'] = df['vader_tweet'].str.replace("[^a-zA-Z#!,$]", " ")
df.head(2)
```

81]:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_twee
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	trum believe killin american destroyin am.
1	5/4/2020 15:07	5/30/2007 13:12	b'RT @bessbell: My dad is an ICU doctor treati...	48841	None	800	1822	Singapore	0.375	0.125	...	2020-05-04	15-07-00	docto treatin covi patient past wee never.

2 rows × 27 columns

```
#remove short words
df['vader_tweet'] = df['vader_tweet'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))
df.head(2)
```

82]:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_twee
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	trum believe killin american destroyin am.

```
#Lower case
df['vader_tweet'] = df['vader_tweet'].str.lower()
df.head(2)
```

]:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_twee
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	trum believe killin american destroyin am.
1	5/4/2020 15:07	5/30/2007 13:12	b'RT @bessbell: My dad is an ICU doctor treati...	48841	None	800	1822	Singapore	0.375	0.125	...	2020-05-04	15-07-00	docto treatin covi patient past wee never.

2 rows × 27 columns

```
#remove stopwords
df["vader_tweet"] = df["vader_tweet"].apply(stopwords)
df.head(2)
```

]:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_twee
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	trum believe killin american destroyin am.
1	5/4/2020 15:07	5/30/2007 13:12	b'RT @bessbell: My dad is an ICU doctor treati...	48841	None	800	1822	Singapore	0.375	0.125	...	2020-05-04	15-07-00	docto treatin covi patient past wee never.

2 rows × 27 columns

```
#remove rare words
df['vader_tweet'] = df['vader_tweet'].apply(lambda x: " ".join(x for x in x.split() if x not in freq_less))
df.head(2)
```

]:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_twee
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	trum believe killin american destroyin

```
#remove URLs
df['vader_tweet'] = df['vader_tweet'].apply(remove_urls)
df.head(2)
```

:

	created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	...	created_date	created_time	tidy_tw
0	5/4/2020 15:07	4/17/2017 0:41	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000	-0.100	...	2020-05-04	15-07-00	tr beli ki ameri destro a
1	5/4/2020 15:07	5/30/2007 13:12	b'RT @bessbell: My dad is an ICU doctor treati...	48841	None	800	1822	Singapore	0.375	0.125	...	2020-05-04	15-07-00	dc trea c pati past w nev

2 rows × 27 columns

```
#!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

sent_analyser = SentimentIntensityAnalyzer()
```

```
def calculate_sentiment_analyser(text):
    return sent_analyser.polarity_scores(text)
```

```
df['sentiment_analyser']=df['vader_tweet'].apply(calculate_sentiment_analyser)

s = pd.DataFrame(index = range(0,len(df)),columns= ['compound_score','compound_score_sentiment'])

for i in range(0,len(df)):

  s['compound_score'][i] = df['sentiment_analyser'][i]['compound']

  if (df['sentiment_analyser'][i]['compound'] <= -0.05):
    s['compound_score_sentiment'][i] = 'Negative'

  if (df['sentiment_analyser'][i]['compound'] >= 0.05):
    s['compound_score_sentiment'][i] = 'Positive'

  if ((df['sentiment_analyser'][i]['compound'] >= -0.05) & (df['sentiment_analyser'][i]['compound'] <= 0.05)):
    s['compound_score_sentiment'][i] = 'Neutral'

df['compound_score'] = s['compound_score']
df['compound_score_sentiment'] = s['compound_score_sentiment']
```

```
df[['vader_tweet','Polarity', 'compound_score', 'compound_score_sentiment']].sample(10)
```

]:

	vader_tweet	Polarity	compound_score	compound_score_sentiment
3672	covid nuttar pradesh nmigrant registration inc...	0.000000	0.296	Positive
9726	massive twitter uproar arrests arrest civil so...	-0.025000	-0.6486	Negative
4150	#caualumni! join president george french, #hbc...	0.000000	0.3595	Positive
3101	abacha sending covid palliatives grave	0.000000	-0.3818	Negative
5834	chinese communist party continues block access...	0.333333	0.3182	Positive
7523	invite lovely women #prayforhumanity pray save...	0.500000	0.872	Positive
2709	mars moves aquarius, extra careful #covid #loc...	-0.050000	0.1531	Positive
9044	endures worst death toll coronavirus pandemic ...	-1.000000	-0.8402	Negative
1412	abacha sending covid palliatives grave	0.000000	-0.3818	Negative
5179	mitch need baseball right need testing need ne...	0.285714	0.25	Positive

Feature Engineering

Bag-of-Words Features

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim


bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')
bow = bow_vectorizer.fit_transform(df['tidy_tweet'])
bow.shape
```

```
]: (12300, 1000)
```

TF-IDF Features

```
#TF-IDF vectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(df['tidy_tweet'])
tfidf.shape
```

```
]: (12300, 1000)
```

Word2Vec Embeddings

```
#Word2Vec features

tokenized_tweet = df['token_tweet']


model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)

model_w2v.train(tokenized_tweet, total_examples= len(df['token_tweet']), epochs=20)
```

```
WARNING:gensim.models.base_any2vec:Effective 'alpha' higher than previous training cycles
```

```
: (2041508, 2472400)
```

Preparing Vectors for Tweets

```
def word_vector(tokens, size):
    '''
    This function will create a vector for each tweet
    by taking the average of the vectors of the words present in the tweet.
    '''
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model_w2v[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary
            continue

    if count != 0:
        vec /= count
    return vec
```

```
#Preparing word2vec feature set…

wordvec_arrays = np.zeros((len(tokenized_tweet), 200))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 200)
    wordvec_df = pd.DataFrame(wordvec_arrays)

wordvec_df.shape

#Now we have 200 new features, whereas in Bag of Words and TF-IDF we had 1000 features.
```

```
(12300, 200)
```

Train Test Split

Splitting dataset into training and testing

```
# below line causes shuffling of indices, to avoid using train_test_split later
df = df.reindex(np.random.permutation(df.index))
```

```
df_copy = df.copy()
```

```
df_new = df[['tidy_tweet', 'compound_score_sentiment']]
```

```
def shuffle(df, test_proportion):
    ratio = int(len(df)/test_proportion)
    train = df[ratio:][:]
    test =  df[:ratio][:]

    return train,test

train,test= shuffle(df_new,3)
```

```
print(train['compound_score_sentiment'].value_counts())
print(test['compound_score_sentiment'].value_counts())
```

```
 1    2841
-1    2803
 0    2556
Name: compound_score_sentiment, dtype: int64
 1    1437
-1    1398
 0    1265
Name: compound_score_sentiment, dtype: int64
```

Modelling on Bag of Words

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim


bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features = 1000, stop_words='english')

bow_features = bow_vectorizer.fit_transform(df['tidy_tweet']).toarray()
labels = df_new.compound_score_sentiment
bow_features.shape
```

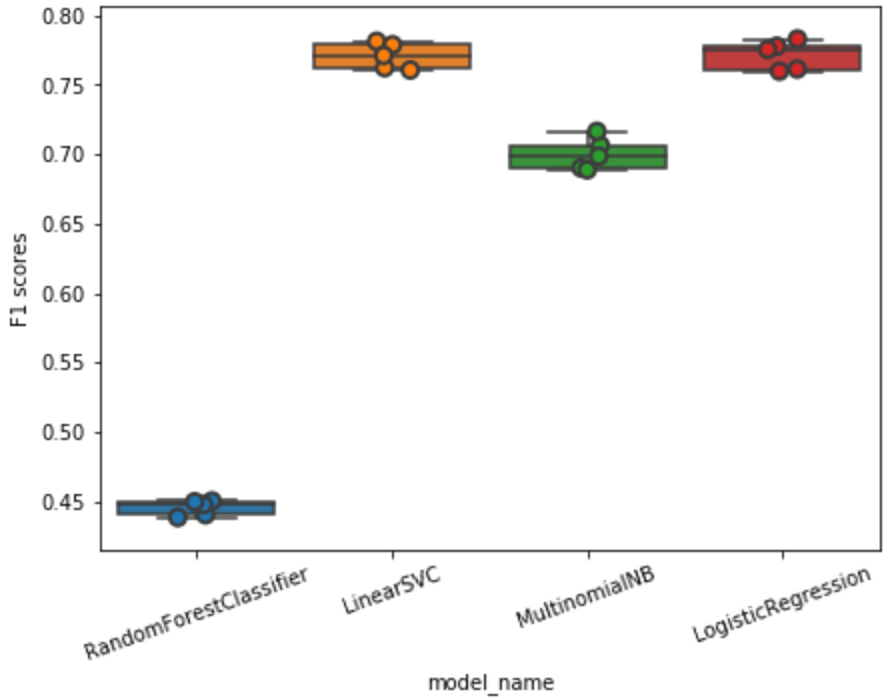
```
]: (12300, 1000)
```

```


models = [
    RandomForestClassifier(n_estimators=200, max_depth=3, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0)
]

CV = 5

cv_bow = pd.DataFrame(index=range(CV * len(models)))
entries = []

for model in models:

    model_name = model.__class__.__name__
    f1 = cross_val_score(model, bow_features, labels, scoring='f1_weighted', cv=CV)

    for fold_idx, f1 in enumerate(f1):
        entries.append((model_name, fold_idx, f1))

cv_bow = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'f1'])
```

```
cv_bow.sample(5)
```

	model_name	fold_idx	f1
7	LinearSVC	2	0.762468
9	LinearSVC	4	0.771260
0	RandomForestClassifier	0	0.438404
5	LinearSVC	0	0.781463
8	LinearSVC	3	0.778955

```
plt.figure(figsize=(7,5))
sns.boxplot(x='model_name', y='f1', data=cv_bow)
sns.stripplot(x='model_name', y='f1', data=cv_bow, size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.xticks(rotation = 20)
plt.ylabel("F1 scores")
plt.show()
```




```
print("F1 scores of different models on Bag of Words (BoW) Features")
bow_df = cv_bow.groupby('model_name').f1.median()
bow_df
```

```
F1 scores of different models on Bag of Words (BoW) Features

model_name
LinearSVC                 0.771260
LogisticRegression        0.775601
MultinomialNB             0.698686
RandomForestClassifier    0.447707
Name: f1, dtype: float64
```

Modelling on TF-IDF Features

```
#for modelling on TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='12',
                        encoding='latin-1', ngram_range=(1, 2), stop_words='english')

features = tfidf.fit_transform(df_new.tidy_tweet).toarray()
labels = df_new.compound_score_sentiment
features.shape
```

```
(12300, 6518)
```

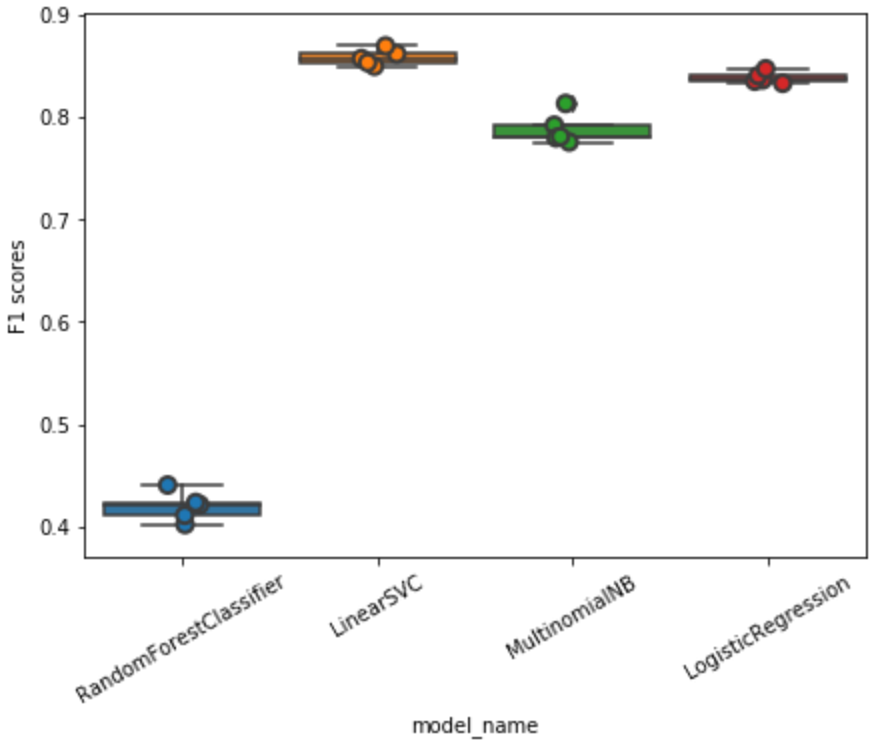
```
# on TF-IDF
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC

from sklearn.model_selection import cross_val_score


models = [
    RandomForestClassifier(n_estimators=200, max_depth=3, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0)
]

CV = 5

cv_tfidf = pd.DataFrame(index=range(CV * len(models)))
entries = []

for model in models:

    model_name = model.__class__.__name__
    f1 = cross_val_score(model, features, labels, scoring='f1_weighted', cv=CV)
    #accuracies = cross_val_score(model, features, labels, scoring='accuracy', cv=CV)

    for fold_idx, f1 in enumerate(f1):
        entries.append((model_name, fold_idx, f1))

cv_tfidf = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'f1'])
```

```
cv_tfidf.sample(5)
```

	model_name	fold_idx	f1
4	RandomForestClassifier	4	0.440608
18	LogisticRegression	3	0.840711
2	RandomForestClassifier	2	0.411034
6	LinearSVC	1	0.849812
17	LogisticRegression	2	0.833173

```
plt.figure(figsize=(7,5))
sns.boxplot(x='model_name', y='f1', data=cv_tfidf)
sns.stripplot(x='model_name', y='f1', data=cv_tfidf, size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.xticks(rotation = 30)
plt.ylabel("F1 scores")
plt.show()
```



```
print("F1 scores of different models on TF-1DF Features")
tfidf_df = cv_tfidf.groupby('model_name').f1.median()
tfidf_df
```

```
F1 scores of different models on TF-1DF Features

model_name
LinearSVC                  0.857309
LogisticRegression         0.836482
MultinomialNB              0.781035
RandomForestClassifier     0.421456
Name: f1, dtype: float64
```

Modelling on Word2Vec Features

```
#Preparing word2vec feature set for modelling

wordvec_arrays = np.zeros((len(tokenized_tweet), 200))

for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 200)
    wordvec_features = wordvec_arrays

labels = df_new.compound_score_sentiment
wordvec_features.shape
```

```
(12300, 200)
```

```
#on Word2Vec

models = [
    RandomForestClassifier(n_estimators=200, max_depth=3, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0),

]

CV = 5

cv_w2v = pd.DataFrame(index=range(CV * len(models)))
entries = []

for model in models:

    model_name = model.__class__.__name__
    f1 = cross_val_score(model, wordvec_features, labels, scoring='f1_weighted', cv=CV)

    for fold_idx, f1 in enumerate(f1):
        entries.append((model_name, fold_idx, f1))

cv_w2v = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'f1'])
```

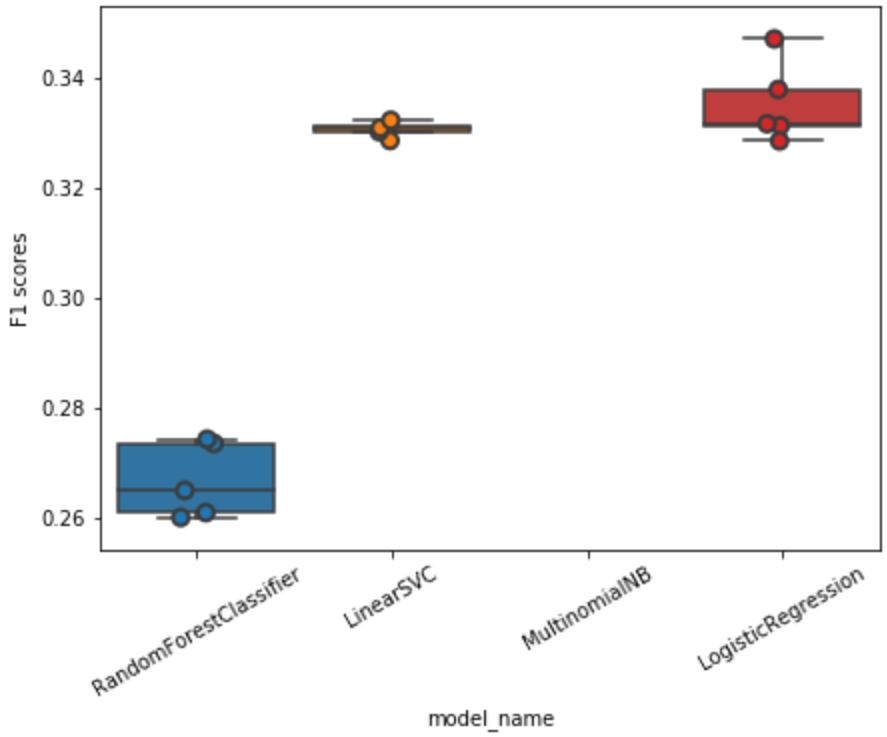
```
cv_w2v.sample(3)
```

	model_name	fold_idx	f1
10	MultinomialNB	0	NaN
18	LogisticRegression	3	0.34708
2	RandomForestClassifier	2	0.25996

```
cv_w2v.dropna()
```

	model_name	fold_idx	f1
0	RandomForestClassifier	0	0.273446
1	RandomForestClassifier	1	0.274214
2	RandomForestClassifier	2	0.259960
3	RandomForestClassifier	3	0.264882
4	RandomForestClassifier	4	0.260847
5	LinearSVC	0	0.331120
6	LinearSVC	1	0.328596
7	LinearSVC	2	0.330183
8	LinearSVC	3	0.330869
9	LinearSVC	4	0.332274
15	LogisticRegression	0	0.331274
16	LogisticRegression	1	0.331635
17	LogisticRegression	2	0.328506
18	LogisticRegression	3	0.347080
19	LogisticRegression	4	0.337892

```
plt.figure(figsize=(7,5))
sns.boxplot(x='model_name', y='f1', data=cv_w2v)
sns.stripplot(x='model_name', y='f1', data=cv_w2v, size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.xticks(rotation = 30)
plt.ylabel("F1 scores")
plt.show()
```




```
print("F1 scores of different models on Word2Vec Features")
w2v_df = cv_w2v.groupby('model_name').f1.median()
w2v_df
```

```
F1 scores of different models on Word2Vec Features

model_name
LinearSVC                0.330869
LogisticRegression       0.331635
MultinomialNB                 NaN
RandomForestClassifier   0.264882
Name: f1, dtype: float64
```

Comparing models

```
result = pd.merge(bow_df, tfidf_df, on='model_name')
result = pd.merge(result, w2v_df, on = 'model_name')
result.columns = ['bow', 'tfidf', 'w2v']
result
```

model_name	bow	tfidf	w2v
LinearSVC	0.771280	0.857309	0.330869
LogisticRegression	0.775801	0.836482	0.331635
MultinomialNB	0.698686	0.781035	NaN
RandomForestClassifier	0.447707	0.421456	0.264882

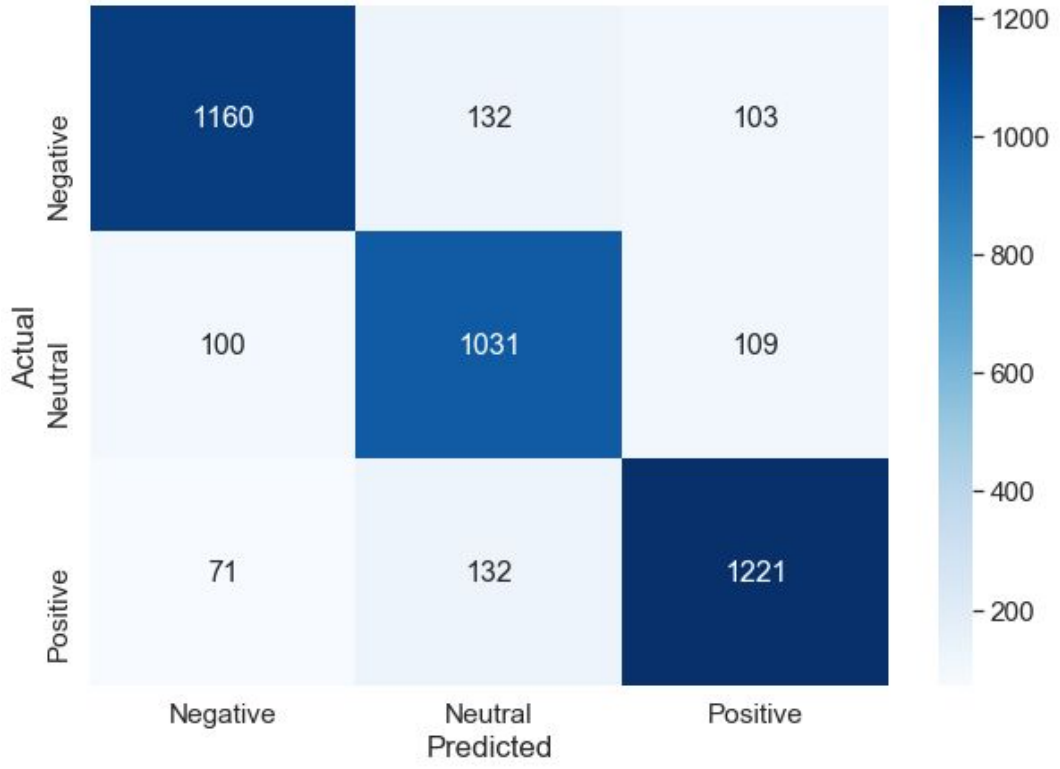
Linear SVC

```
#Model SVC on TF-IDF prediction

model = LinearSVC()

X_train, X_test, y_train, y_test, indices_train, indices_test = train_test_split(features,
                                                                                 labels,
                                                                                 df.index,
                                                                                 test_size=0.33,
                                                                                 random_state=0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score

acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average = 'weighted')
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average = 'weighted')

cfr = classification_report(y_test, y_pred)

print("Accuracy : ", accuracy)
print("F1 : ", f1)
print("Recall : ", recall)
print("Precision : ", precision)
print("Classification Report : ", cfr)
```

```
Accuracy :  0.8107932287827527
F1 :  0.8408521468204391
Recall :  0.8406011332840602
Precision :  0.8416657892256071
Classification Report :
              precision    recall  f1-score   support

    Negative       0.87      0.83      0.85      1395
     Neutral       0.80      0.83      0.81      1240
    Positive       0.85      0.86      0.85      1424

    accuracy                           0.84      4059
   macro avg       0.84      0.84      0.84      4059
weighted avg       0.84      0.84      0.84      4059
```

```
#confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", fmt = 'd', annot=True,annot_kws={"size": 16})# font size
```



Topic Modelling - LDA

```
#PLDA Topic Modelling

#BIGRAMS

bigram = gensim.models.Phrases(text)

def clean(word_list):
    """
    Function to clean the pre-processed word lists
    #ADDED BIGRAMS for LDA

    Following transformations will be done
    1) Stop words removal from the nltk stopword list
    2) Bigram collation (Finding common bigrams and grouping them together using gensim.models.phrases)
    3) Lemmatization (Converting word to its root form : babies --> baby ; children --> child)
    """
    #remove stop words
    clean_words = [w for w in word_list if not w in eng_stopwords]

    #collect bigrams
    clean_words = bigram[clean_words]

    #Lemmatize
    clean_words=[lem.lemmatize(word, "v") for word in clean_words]
    return(clean_words)
```

```
all_text = text.apply(lambda x:clean(x))
```

```
all_text = text.apply(lambda x:clean(x))
```

```
dictionary = Dictionary(all_text)
print("There are",len(dictionary),"number of words in the final dictionary")
```

```
There are 23180 number of words in the final dictionary
```

```
print(dictionary.doc2bow(all_text.iloc[1]))
print("Wordlist from the sentence:",all_text.iloc[1])

#to check
print("Wordlist from the dictionary lookup:",
      dictionary[21],dictionary[22],dictionary[23],dictionary[24],dictionary[25],dictionary[26],dictionary[27])
```

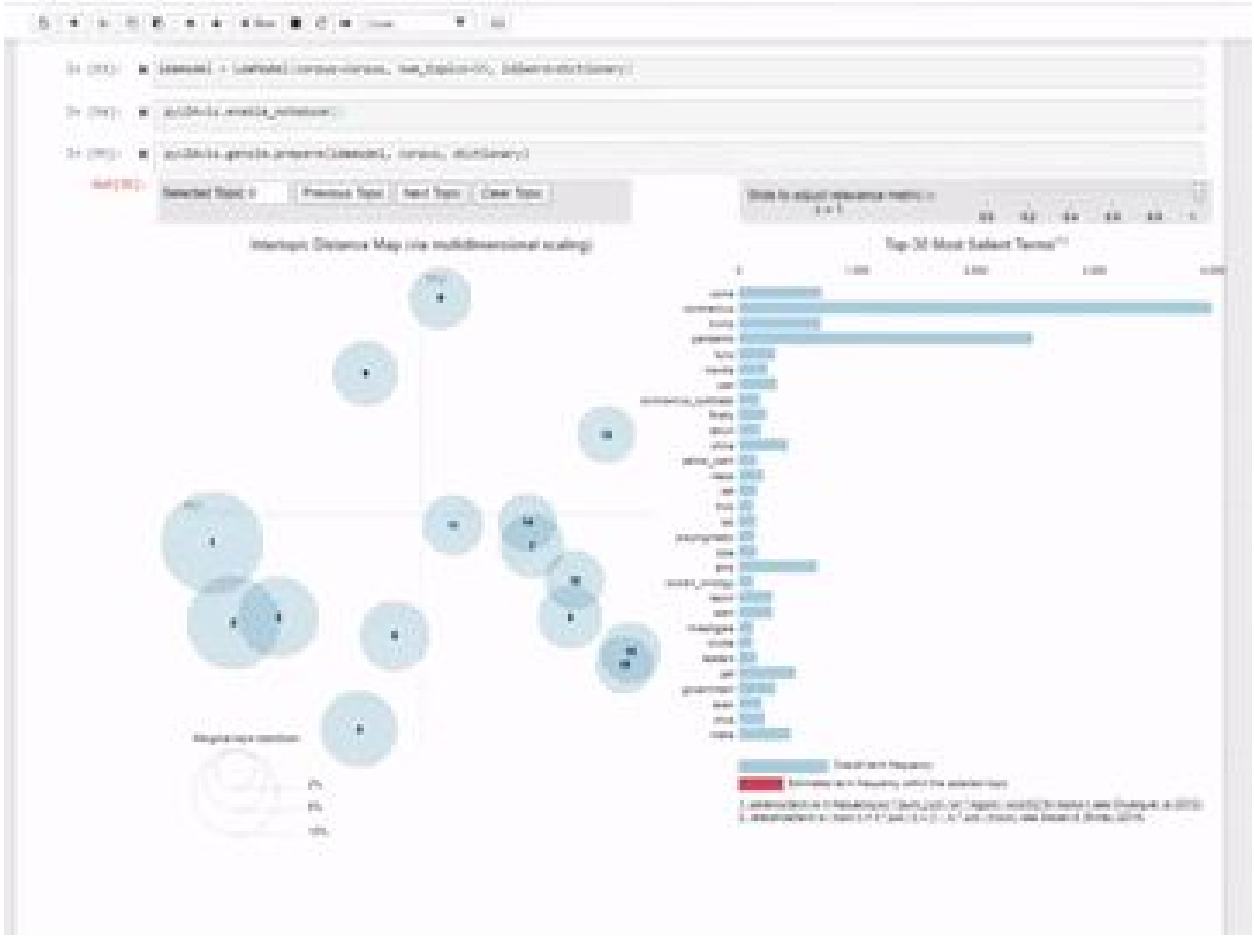
```
[(12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 1), (20, 1)]
Wordlist from the sentence: ['bessbell_dad', 'icu_doctor', 'treating_covid', 'patients', 'past_week', 'set', 'never_seen', 'heart_rate', 'rbc_count']
Wordlist from the dictionary lookup: china_lied conclude crucial_early deliberately_suppressed destroyed_evidence leaked_dossier thomas_paine
```

```
corpus = [dictionary.doc2bow(text) for text in all_text]
```

```
ldamodel = LdaModel(corpus=corpus, num_topics=15, id2word=dictionary)
```

```
pyLDAvis.enable_notebook()
```

```
pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)
```

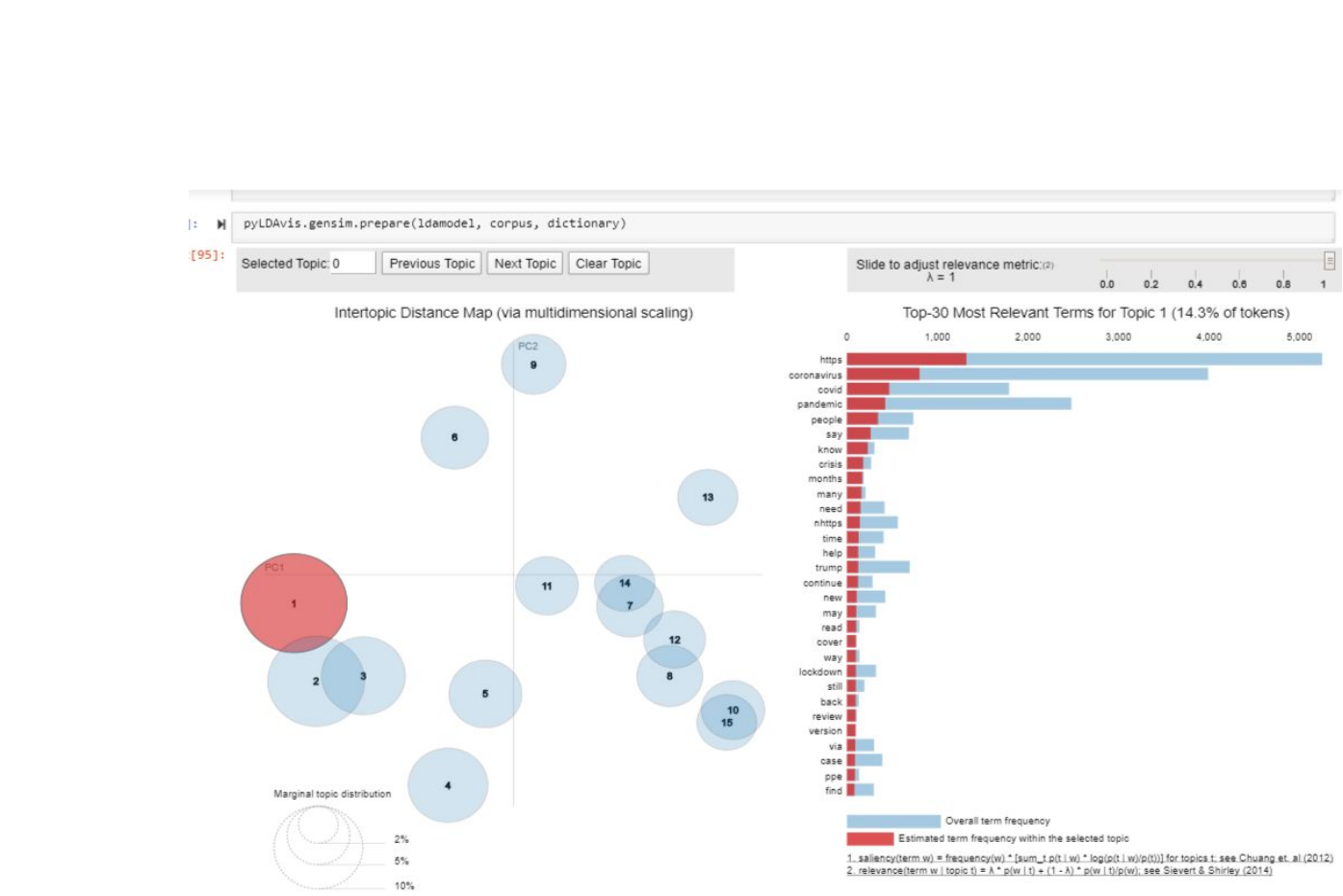

Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:
Intertopic Distance Map (via multidimensional scaling)
Top 30 Most Salient Terms
Overall Term Frequency
Estimated term frequency within the selected topic

```
pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)
```

[95]:

Selected Topic: 0 | Previous Topic | Next Topic | Clear Topic

Slide to adjust relevance metric:(2)
λ = 1




```
pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)
```

Selected Topic: 0 | Previous Topic | Next Topic | Clear Topic

Slide to adjust relevance metric:(2)
λ = 1

In [3]:
```
#importing packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import os
from textblob import TextBlob
from datetime import datetime
import time


%matplotlib inline
```

In [4]:
```
df = pd.read_csv('tweets1.csv', index_col = 0)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12300 entries, 2020-05-04 15:07:36 to 2020-05-04 15:08:50
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   usersince          12300 non-null  object
 1   text               12300 non-null  object
 2   retwc              12300 non-null  int64
 3   hashtag            12300 non-null  object
 4   followers          12300 non-null  int64
 5   friends            12300 non-null  int64
 6   authorloc          8807 non-null   object
 7   Subjectivity       12300 non-null  float64
 8   Polarity           12300 non-null  float64
 9   count_sent         12300 non-null  int64
 10  count_word         12300 non-null  int64
 11  count_unique_word  12300 non-null  int64
 12  usersincedate      12300 non-null  object
 13  usersincetime      12300 non-null  object
dtypes: float64(2), int64(6), object(6)
memory usage: 1.7+ MB
```

In [9]:
```
#converting account creation date to datetime

df['usersincedate'] = df['usersincedate'].astype('datetime64[ns]')
```

In [15]:
```
#converting datetime to ordinal
#https://stackoverflow.com/questions/53893323/how-to-convert-pandas-data-frame-datetime-column-to-int

df['usersincedate'] = df['usersincedate'].apply(lambda x:x.toordinal())
```

In [16]:
```
df
```

Out[16]:

created	usersince	text	retwc	hashtag	followers	friends	authorloc	Subjectivity	Polarity	count_sent	count_word	count_unique_word
2020-05-04 15:07:36	2017-04-17 00:41:42	b'@realDonaldTrump Trump believes killing 70,0...	0	None	7	10	Boca Raton, FL	0.000000	-0.100000	1	16	16
2020-05-04	2007-05-30	b'RT @bessbell: My dad is an ICU doctor	48841	None	800	1822	Singapore	0.375000	0.125000	1	26	26

```
# devide timestamp into date and time

df['Date'] = pd.to_datetime(df['usersince'])
df['usersince_date'] = df['Date'].apply( lambda x: x.strftime("%Y-%m-%d"))
df['usersince_time'] = df['Date'].apply( lambda x: x.strftime("%H-%M-%S"))
df.drop(['Date'],axis = 1, inplace =True)

df['Date2'] = pd.to_datetime(df['created'])
df['created_date'] = df['Date2'].apply( lambda x: x.strftime("%Y-%m-%d"))
df['created_time'] = df['Date2'].apply( lambda x: x.strftime("%H-%M-%S"))
df.drop(['Date2'],axis = 1, inplace =True)
```

In [26]:
```
print("Polarity vs Account Creation Date Correlation:")
np.corrcoef(df.usersincedate, df.Polarity)
```

```
Polarity vs Account Creation Date Correlation:
```

Out[26]:
```
array([[ 1.        , -0.01019287],
       [-0.01019287,  1.        ]])
```

In [18]:
```
print("Subjectivity vs Account Creation Date Correlation:")
np.corrcoef(df.usersincedate, df.Subjectivity)
```

```
Subjectivity vs Account Creation Date Correlation:
```

Out[18]:
```
array([[ 1.        , -0.00243186],
       [-0.00243186,  1.        ]])
```

In [19]:
```
print("Polarity vs Word Count Correlation:")
np.corrcoef(df.count_word, df.Polarity)
```

```
Polarity vs Word Count Correlation:
```

Out[19]:
```
array([[1.        , 0.02116283],
       [0.02116283, 1.        ]])
```

In [20]:
```
print("Subjectivity vs Word Count Correlation:")
np.corrcoef(df.count_word, df.Subjectivity)
```

```
Subjectivity vs Word Count Correlation:
```

Out[20]:
```
array([[1.        , 0.17368737],
       [0.17368737, 1.        ]])
```

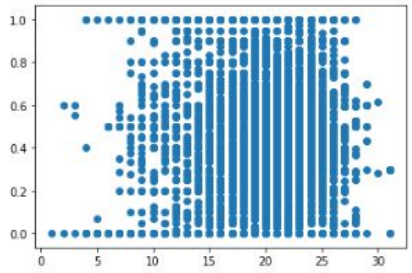
In [23]:
```
print("Followers vs Word Count Correlation:")
np.corrcoef(df.followers, df.count_word)
```

```
Followers vs Word Count Correlation:
```

Out[23]:
```
array([[ 1.        , -0.01918361],
       [-0.01918361,  1.        ]])
```

In [28]:
```
plt.scatter(df.count_word, df.Subjectivity)
```

Out[28]:
```
<matplotlib.collections.PathCollection at 0x23a5c532488>
```

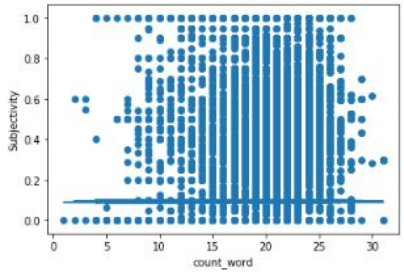


In [30]:
```
#Visualizing the strongest relationship, Word Count vs Subjectivity

y = df.Subjectivity #outcome variable
x = df.count_word #predictor
x = sm.add_constant(x)
lr_model = sm.OLS(y,x).fit()
print(lr_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Subjectivity   R-squared:                       0.030
Model:                            OLS   Adj. R-squared:                  0.030
Method:                 Least Squares   F-statistic:                     382.5
Date:                Wed, 06 May 2020   Prob (F-statistic):          6.53e-84
Time:                        13:57:10   Log-Likelihood:                -2871.5
No. Observations:               12300   AIC:                             5747.
Df Residuals:                   12298   BIC:                             5762.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0886      0.013      7.042      0.000       0.064       0.113
count_word     0.0123      0.001     19.559      0.000       0.011       0.014
==============================================================================
Omnibus:                      998.920   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              935.283
Skew:                           0.612   Prob(JB):                    8.05e-204
Kurtosis:                       2.430   Cond. No.                         91.6
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [31]:
```
plt.scatter(df.count_word, df.Subjectivity)
plt.plot(df.count_word, 0.0123 * df.Subjectivity + 0.0886)
plt.xlabel('count_word')
plt.ylabel('Subjectivity')
np.corrcoef(df.count_word, df.Subjectivity)[0,1]
```

Out[31]:
```
0.17368736699233153
```

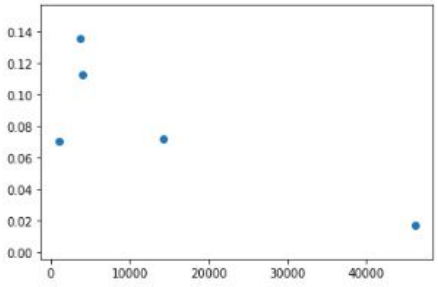


```
In [50]: !python ./twitter_search.py "Covid Kansas"

         Query: Covid Kansas, Location: , Language: en, Search type: mixed, Count: 2000

In [37]: df2 = pd.read_csv('cases.csv', index_col = 0)

In [38]: df2
```

Out[38]:

State	Polarity	Cases
Maine	0.070479	1000.0
Kansas	0.136036	3700.0
Utah	0.112762	4000.0
Virginia	0.071925	14300.0
California	0.016864	46200.0

```
In [42]: print("Case Count vs Polarity Correlation:")
         np.corrcoef(df2.Cases, df2.Polarity)

         Case Count vs Polarity Correlation:

Out[42]: array([[ 1.        , -0.82025998],
                [-0.82025998,  1.        ]])

In [44]:
         plt.scatter(df2.Cases, df2.Polarity)

Out[44]: <matplotlib.collections.PathCollection at 0x23a5ba3ee48>
```



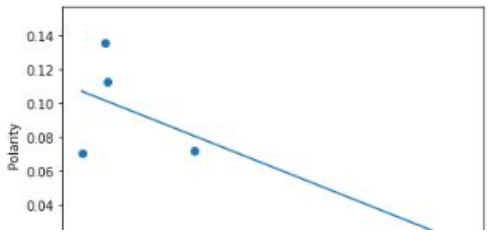
In [46]:
```
#finding regression data for Case Count vs Polarity

y = df2.Polarity #outcome variable
x = df2.Cases #predictor
x = sm.add_constant(x)
lr_model = sm.OLS(y,x).fit()
print(lr_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Polarity   R-squared:                       0.673
Model:                            OLS   Adj. R-squared:                  0.564
Method:                 Least Squares   F-statistic:                     6.169
Date:                Wed, 06 May 2020   Prob (F-statistic):             0.0890
Time:                        14:57:31   Log-Likelihood:                 11.688
No. Observations:                   5   AIC:                            -19.38
Df Residuals:                       3   BIC:                            -20.16
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1092      0.017      6.249      0.008       0.054       0.165
Cases      -1.994e-06   8.03e-07     -2.484      0.089   -4.55e-06    5.61e-07
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   2.210
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.164
Skew:                        -0.136   Prob(JB):                        0.921
Kurtosis:                     2.157   Cond. No.                     2.82e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.82e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
C:\Users\abedi\Anaconda3\lib\site-packages\statsmodels\stats\stattools.py:71: ValueWarning: omni_normtest is not valid with les
s than 8 observations; 5 samples were given.
  "samples were given." % int(n), ValueWarning)
```

In [49]:
```
#Plotting regression and calculating correlation coefficient

plt.scatter(df2.Cases, df2.Polarity)
plt.plot(df2.Cases, -1.994e-06 * df2.Cases + 0.1092)
plt.xlabel('Cases')
plt.ylabel('Polarity')
np.corrcoef(df2.Cases, df2.Polarity)[0,1]
```

Out[49]:
```
-0.8202599838525004
```



References:

https://medium.com/swlh/coronavirus-python-tutorial-1-520cc960aac1
https://github.com/niks92/Twitter-sentiment-analysis/blob/master/Sentiment_vaderSentiment.ipynb
https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/
https://towardsdatascience.com/fine-grained-sentiment-analysis-in-python-part-2-2a92fdc0160d
https://www.analyticsvidhya.com/blog/2018/10/mining-online-reviews-topic-modeling-lda/