SENTIMENT ANALYSIS ON IPHONE TWEETS

Priyabrata Thatoi | Trishla Mishra | Shweta Parida | Harshitha Pamyshetty

EXECUTIVE SUMMARY

Sentiment analysis has been a significant approach to business to identify positive, negative and neutral reviews based on text format. The text form is used to get insights from social media reviews, responses and product comments, and helps in solving business problems by making decisions. With the enormous data available in the present day, sentiment analysis has become a tool for making sense of the data.

This project is to mine customer's sentiments in the form of comments and reviews from social media platforms such as Twitter to capture and compare their reactions and online conversations posted after the launch of iPhones. With this project, we aim to assist a potential buyer in giving them an honest idea on customer's review using sentiment analysis, and help the company understand if they have successfully fulfilled their customers' demands and managed to set a benchmark again, like they always did in the past.

STATEMENT OF SCOPE

The objective of this project is to:

- Analyze the customer reviews and comments about the Apple products i.e. iPhone 7, iPhone 7 Plus, iPhone 8, iPhone 8 Plus, iPhone X, iPhone XS, iPhone XS Max, iPhone XR, iPhone 11, iPhone 11 Pro, iPhone 11 Pro Max.
- Scrapping the twitter data using the package **TwitterScraper** in Python followed by in-depth sentiment analysis of the tweets and understand how people perceive the new release of iPhone 11 with respect to the other iPhones released previously.

Tweeter Variables:

Variable name	Variable description
fullname	Full name of the user
html	html code of specific twitter page
is_retweet	
likes	Number of likes on tweets
replies	Number of replies for tweets
retweet_id	ID of retweet posted
retweeter_userid	UserID of retweet posted
retweeter_username	Username of retweet posted
retweets	Tweets which have been re-posted on twitter wall
text	Tweets posted by the users
timestamp	Datetime of the tweets posted
timestamp_epochs	Same as timestamp (above)
tweet_id	ID of tweet
tweet_url	url of tweet posted
user_id	ID of user
username	Username provided for twitter account by the user

DATA PREPARATION

In this project we have used social media platform such as twitter as the source of our data files. Twitter is a gold mine of data. Almost every tweet of the users is completely public and can be pulled, which is not very likely in other social media platforms. This is an advantage as we will be pulling a large amount of data to run analytics on. Also, the twitter data is quite specific as the

API’s of Twitter allows us to do complex queries such as pulling every tweet regarding a specific topic for specific time interval. In this project, we are using the package called ‘TwitterScraper’ to pull the data from twitter. TwitterScraper is not limited by the number of requests sent, but by the internet speed/Bandwidth and the number of instances of TwitterScraper we are willing to start.

Using TwitterScraper we scraped the data for iPhone 7, 7 Plus, 8, 8 Plus, X, XS, XS Max, XR, 11, 11 Pro, 11 Pro Max. We obtained the json files of all these scraped data and using the function “concat()”, we combined all the data into a single dataframe.

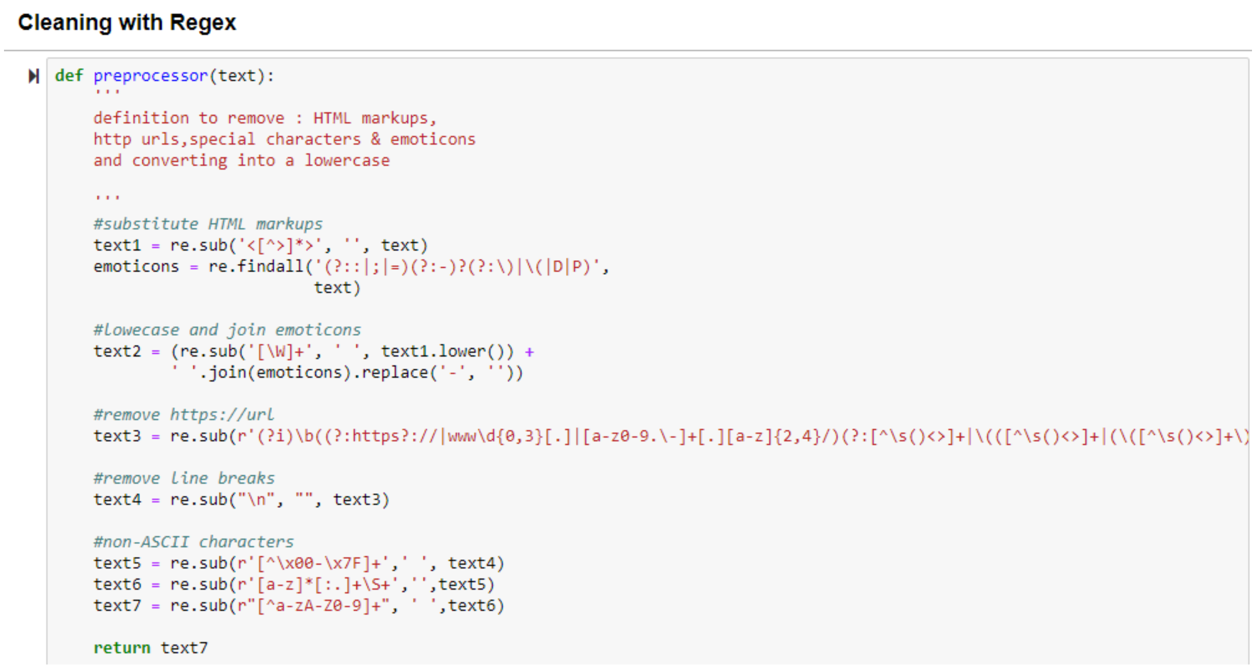
DATA CONSOLIDATION

As we have a lot of data sets, there is a need to consolidate all these data sets into one single data file. We do this by using the function “concat()”. This function performs the concatenation operation along an axis, while performing the intersection of the indexes on the other axes. There are two ways in which we can join the dataframes, i.e. inner and outer. Outer is used for union whereas inner is used for intersection.

In this project, we have intersectioned all the dataframes into a single data frame using concat(). After combining all the data into a single dataframe, we were able to get a dataset of size around 40MB compromising of 33115 rows and 17 columns. We then converted this dataset into a csv file.

DATA CLEANING

After the data has been consolidated, it has to be cleaned to make it useful. We use “Regex” to clean the data. The whole data is described by 17 attributes which are ‘unnamed’, ‘fullname’, ‘html’, ‘is_retweets’, ‘likes’, ‘replies’, ‘retweet_id’, ‘retweeter_userid’, ‘retweeter_username’, ‘retweets’, ‘text’, ‘timestamp’, ‘timestamp_epochs’, ‘tweet_id’, ‘tweet_url’, ‘userid’ and ‘username’. The consolidated dataset shows the items which are needed to be cleaned before the analysis: Below is the snap shot of the regex routine designed in Python to clean the data such as html, url, special characters and others.



Cleaning with Regex

```
def preprocessor(text):
    '''
    definition to remove : HTML markups,
    http urls,special characters & emoticons
    and converting into a lowercase

    '''
    #substitute HTML markups
    text1 = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',
                           text)

    #lowecase and join emoticons
    text2 = (re.sub('[\W]+', ' ', text1.lower()) +
            ' '.join(emoticons).replace('-', ''))

    #remove https://url
    text3 = re.sub(r'(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)

    #remove line breaks
    text4 = re.sub("\n", "", text3)

    #non-ASCII characters
    text5 = re.sub(r'[^\x00-\x7F]+',' ', text4)
    text6 = re.sub(r'[a-z]*[:.]+\S+','',text5)
    text7 = re.sub(r"[^a-zA-Z0-9]+", ' ',text6)

    return text7
```

- Irrelevant Attributes

There are few attributes which we do not need for our analysis such as ‘Unnamed’, ‘fullname’, ‘html’, ‘is_retweets’, ‘retweet_id’, ‘retweeter_userid’, ‘retweeter_username’, ‘timestamp_epochs’, ‘tweet_url’ and ‘username’. So we delete all these attributes and retain only the following attributes ‘likes’, ‘replies’, ‘retweets’, ‘text’, ‘user_id’, ‘timestamp’, and ‘tweet_id’.

- Duplicated records

The consolidated dataset has duplicated records which can affect our analysis results. We do not require tweets which are posted multiple times on twitter, so we need to eliminate the duplicated records under the variable “text”. For this, we have used the function ‘drop()’ to remove duplicate data of text, which resulted into 14242 records.

DATA TRANSFORMATION

Here we will take the cleansed data and enhance it further for analysis. In this step, we define a function called ‘preprocessor()’ in order to transform words into machine learning algorithm readable format in which we remove HTML Markups, http urls, special characters & emoticons, line breaks and non – ASCII characters as these do not contribute any meaning for the text and it will not be used for sentiment analysis.

We will also change the words into lower case format in order to reduce the size of the text data vocabulary and remove unwanted words such as http, twitter, com, Also, we will be manipulating the date and time of the timestamp column. Previously, before manipulating, timestamp was of the type ‘Object’. But, after manipulation, it is of the type ‘datetime’. We do this because, we cannot perform any time series-based operations on the dates if they are not in the right format.

DATA REDUCTION

After data cleaning and transformation of the original dataset, we have now obtained a dataset which is in a reduced state. The number of records is now reduced to 14242 with 7 attributes. We have removed all the irrelevant attributes and the duplicate records which gave us “iPhone1” dataset.

DATA DICTIONARY

The following table captures all the selected attributes of the data for our project with Data Typ e, Example and Descriptions.

Variable Name	Data Type	Example	Description
likes	int	82	Number of likes each tweet has obtained
replies	int	3	Number of replies/comments , tweet ha s got.
retweets	int	13	Tweets posted on a tweet
text	object	'NYC + iPhone 11 P ro Max: Dope! pic.t witter.com/zwVBw 5NRMD'	Tweets posted on twitter
user_id	int	40538510	Unique ID of User
timestamp	datetime	2019-10-30 23:59:16	Date & time of the tweet posted
tweet_id	int	118969328208495 0016	Unique ID of the tweet
Unnamed	int	5	Index

TEXT MINING

A textual data has to undergo pre-processing procedure to transform words into the format which is supported for machine learning algorithms. We have used a library called **NLTK(Natural Language Toolkit)** for text pre-processing in Python.

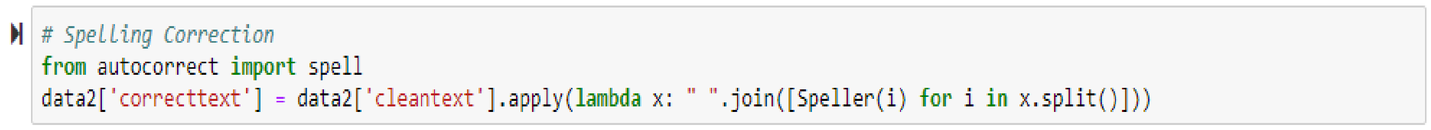
Following are the text pre-processing steps which the textual data has undergone:

Spelling correction:

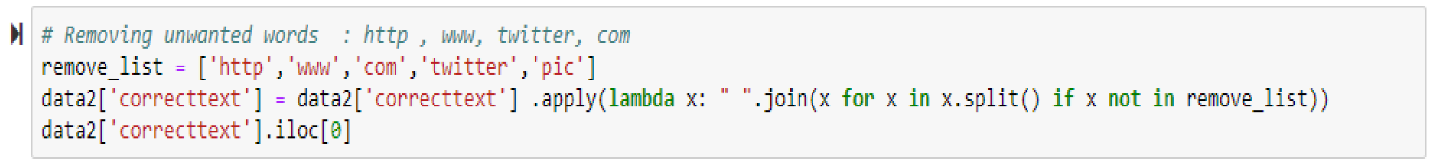
We have used a library called “autocorrect” to spell check in Python. From autocorrect we have imported spell which works for words. Using the autocorrect library, we iterate through the rows of the dataframe and then iterate through the words within the specific rows to apply the spell method. This process is for correcting a words’ spelling for example, “lisr” instead of ‘list”.

Removing unwanted words:

There are few words in the text which are not required such as http, com, twitter, www, pic etc. Hence, we remove all those words from the text so that they don’t add unnecessary weightage to the text and so that it becomes easier for sentiment analysis.



```
# Spelling Correction
from autocorrect import spell
data2['correcttext'] = data2['cleantext'].apply(lambda x: " ".join([Speller(i) for i in x.split()]))
```



```
# Removing unwanted words  : http , www, twitter, com
remove_list = ['http','www','com','twitter','pic']
data2['correcttext'] = data2['correcttext'] .apply(lambda x: " ".join(x for x in x.split() if x not in remove_list))
data2['correcttext'].iloc[0]
```

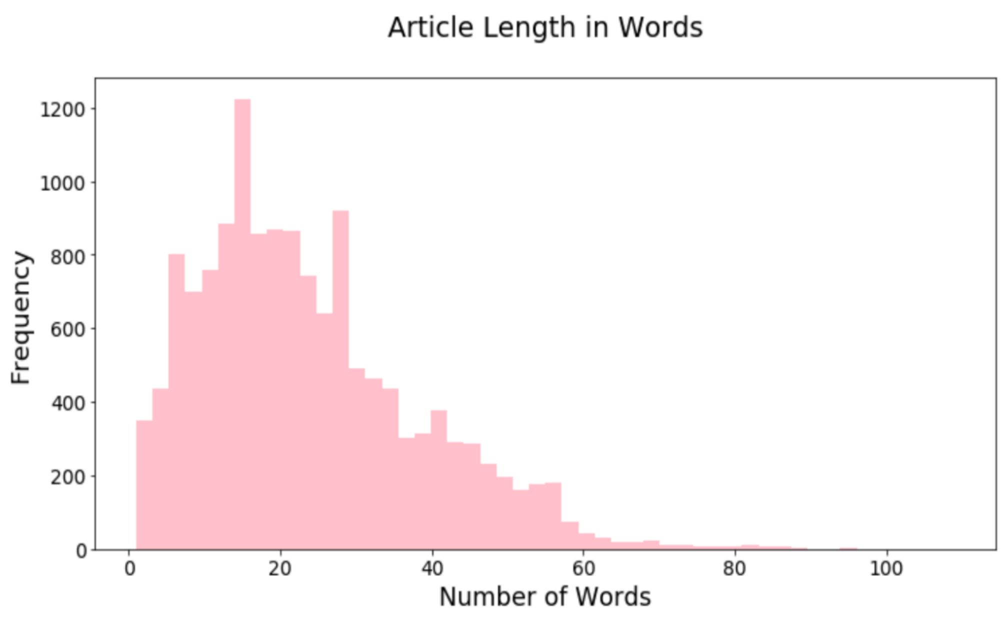
ANALYSIS ON WORDS

After removing unwanted words from the text, we need to analyze the words. We can observe that the mean number of words are 23.8, the minimum number of words are 1 and the maximum number of words are 109.

Data with more than 0 words:

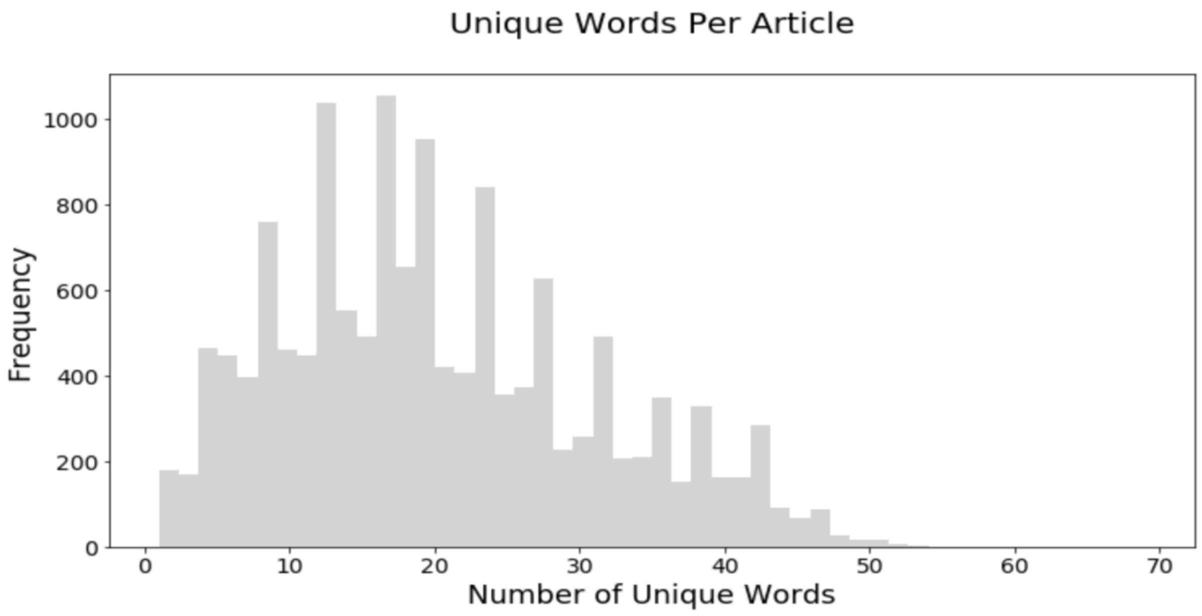
- We created a dataframe named articles and checked all the data having number of words num_wds greater than 0.
- The dataframe captured the number of words which is greater than 0.
- To have a visual inference, we have created a histogram to display the frequency at which each count of words occurs in each row.

- The maximum frequency (approx.1200) lies in the interval of 0 to 20 as observed in the histogram of article length (In words) which implies most of the rows in the dataframe are having wordcount between 0 to 20.
- The graph, which shows the article length, is skewed right. On the right side, the frequency of word count > 20 is decreasing.



Number of unique words:

- We introduced a column ***uniq_wds*** in the dataframe ***articles***, which captures all the distinct words in the dataset by using a function ***set()***.
- Now, we have the count of unique words for each row of the dataframe and the number of times, the count occurs.
- To have a visual inference, we have created a histogram to display the frequency at which each count of unique words occurs in each row.
- The maximum frequency (approx. 800) lies in the interval of 10 to 20 as observed in the histogram of unique words (per article) which implies most of the rows in the dataframe are having wordcount between 10 to 20 unique words.
- The graph, which shows the unique words per article, is skewed right, which is implying, the frequency of the wordcount of unique words is decreasing with the wordcount except it has shown a rise at around 30.



Stop word removal:

Stop words are the words which do not provide any meaning to the context or the meaning of the sentence. These words must be removed because of following reasons:

- No meaning to the content
- Takes Unnecessary space in the database
- Increases processing time

So, it is safe to remove the stop words without any impact to the meaning of the content.

- We have taken a set of stopwords in English language from NLTK library and have stored in a list *stop* .
- Then, we have performed an operation on texts under the column *correcttext* of dataframe *data2* by selecting all the words from the column which are not in the list *stop* and joining those as elements in a list by a function *join()*.
- So, the column *correcttext* of dataframe *data2* has no stop words.

Stemming:

The next step after removing stopwords is stemming, for which we import PorterStemmer from nltk package. PorterStemmer is basically a process for removing the suffix from words in English. This process of stemming removes the last few characters of the words which leads to incorrect spellings and meaningless words. This the main reason why we do not use stemming and use lemmatization instead as it gives meaningful words.

Lemmatization:

Lemmatization is the process of conversion of words to its root form (valid words), to have a morphological analysis of the data. Lemmatization gives us meaningful words as it converts the words into its root words by taking the context into consideration.

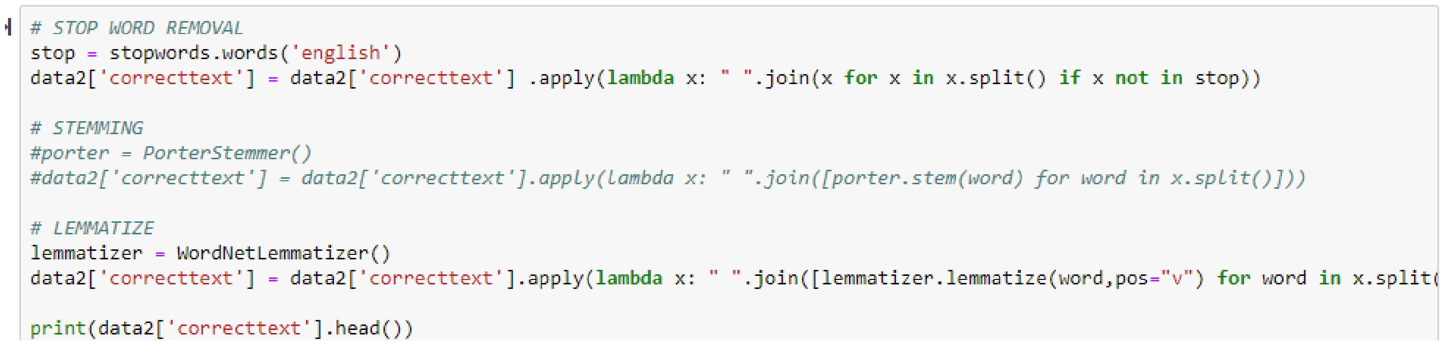
- We have used a function *WordNetLemmatizer()* to get the lemmas of words from the the library WordNetLemmatizer from imported package *nltk.stem* and stored in *lemmatizer.*
- Each elements (verb words) in the list of all words from the column *correcttext* of dataframe *data2* have been converted to its root form by *lemmatizer* and joined by *join()* to form the sentence.
- We have also added verb for the part of speech pos as a parameter in order to convert to verb root words

Count Vectorization:

Text data is required to be converted into numerical data as machine learning language understands numerical attributes to perform various mathematical operations.For this, we have used the process of Count Vectorization & featurization.

Count vectorization is the process of transforming the string feature of text data into vectors & Featurization is converting to numeric feature. To have this operation done on the dataset, we followed following procedure:

- We imported library CountVectorizer from the package sklearn.feature_extraction.text to perform the function and declared the function in a variable vectorizer.
- Then, we applied fit transform method on the text data correcttext of dataset data2, where we are using a function fit_transform() which combines 2 functions : fit() and transform().
- First, the function fit() vectorizers each token by providing indices to the tokens.
- Second, the function transform(), we get the count of each token appeared in the dataset.
- Then, we used function array() on the transformed data to have an array representation of the output in a matrix format.
- We also mapped the integer indices to the feature name as column by using the function get_feature_names().
- The array of transformed data has been captured in a dataframe tokens_data.



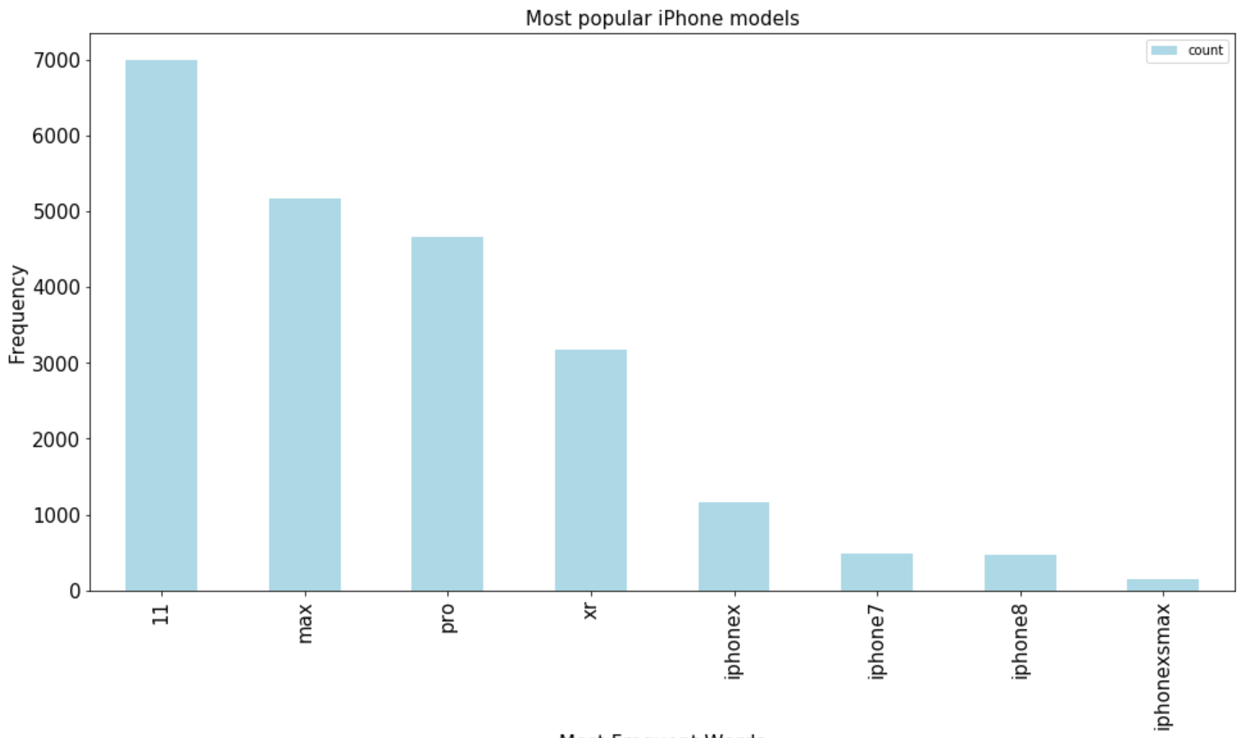
```
# STOP WORD REMOVAL
stop = stopwords.words('english')
data2['correcttext'] = data2['correcttext'] .apply(lambda x: " ".join(x for x in x.split() if x not in stop))

# STEMMING
#porter = PorterStemmer()
#data2['correcttext'] = data2['correcttext'].apply(lambda x: " ".join([porter.stem(word) for word in x.split()]))

# LEMMATIZE
lemmatizer = WordNetLemmatizer()
data2['correcttext'] = data2['correcttext'].apply(lambda x: " ".join([lemmatizer.lemmatize(word,pos="v") for word in x.split(

print(data2['correcttext'].head())
```

Most Popular iPhone Models:

- To have total number of count of tokens, we used a function sum() without taking elements which are displaying NA.
- The values of sum have been stored in a dataframe a.

As the project is based on the analysis of tweets related to the iPhone products, we need to find out the number of times, each product has appeared on twitter wall to have a clear view of its popularity among the users.

- First, we created a list where we included the exact feature names of iPhone products: ['11','xs','pro','max','iphonex','xr','x','iphone7','iphone8','iphone8plus','iphonexsmax'] displayed in token_data.
- Second, we filtered the row word of dataframe top30 with all feature names by using a function isin(), in order to have sum values for plotting.
- For a visual representation, we created a histogram between list of feature names against its frequency in token_data.
- From the graph, we can observe that feature name with "11" have occurred the highest number of times in the dataset, which implies iPhone 11 is the most talked product of iPhone. We are relating frequency with the popularity of the products.
- The frequency is decreased with iPhone max which shows that it has less popularity than iPhone 11 , but it has maintained its popularity than the rest of the products.
- We can also infer from the graph that iPhonexsmax has lowest popularity, hence, the tweets related to the product have been talked less among the twitter users.
- This interpretation is based on the frequent words related to the iPhone products appeared in the dataset.



SENTIMENT ANALYSIS

In today's world, we are overloaded with data. The companies might have clusters of feedback and opinions collected but this data is not easily understood or analyzed by the people.

Sentiment analysis is an important process into identifying the issues from the customers perspective. It allows businesses to understand how customers feel about their products and services and pull out valuable insights for better decision making. Sentiment analysis can be automated, and the decisions can be made on huge data.

Sentiment analysis is also known as "opinion mining" and uses natural language processing, text analysis, computational linguistics and biometrics to identify, analyze and study affective states and subjective information.

After mining the data, processing it and cleaning the text, we need to identify if the processed data is positive, negative or neutral. Sentiment analysis is an automatic process for identifying such emotions within the text.

In our project, sentiment analysis helps us make sense of responses and tweets and helps in answering questions such as:

- How many negative responses were received about the tweets?
- What aspects of the product do the customers love?

- What aspects of the product do the customers hate?
- Has a product feature improved?

Sentiment analysis basically consists of two parts: **Subjectivity and Polarity.**

Polarity – It is basically the emotions expressed in a sentence which can range from positive to negative.

Subjectivity – It is basically the sentences which express some personal feelings, beliefs or views.

Text Blob:

Text blob is a Python library for processing the texual data. It provides a simple API for getting into the natural language processing (NLP) task such as parts of speech tagging, noun phrase extraction, sentiment analysis etc. Text blob aims in providing access to common text processing operations through a familiar interface wherein we can treat text blob objects as if they were Python strings that learnt natural language processing.

Text blob has various features, few of which are noun phrase extraction, sentiment analysis, POS tagging, language translation, tokenization, parsing, ngrams, spelling correction, wordnet integration etc.

When calculating the sentiment of a single word, textblob uses a technique called "averaging". Textblob also handles modifier words. It ignores one letter words in its sentiment phrases and also ignores words which it doesn't know anything about

```
data2['sentiment'] = data2['correcttext'].apply(lambda tweet: TextBlob(tweet).sentiment.polarity)
data2[['correcttext','sentiment']].head()
```

	correcttext	sentiment
0	drop news give away free phone 11 february ent...	0.60
1	enter skint phone 11 giveaway chance win brand...	0.47
2	enter chance win phone 11 256gb airposts speci...	0.58
3	okay new phone 11 pro really good video shit b...	0.40
4	know brood phone11	0.00

Sentiment Polarity Distribution:

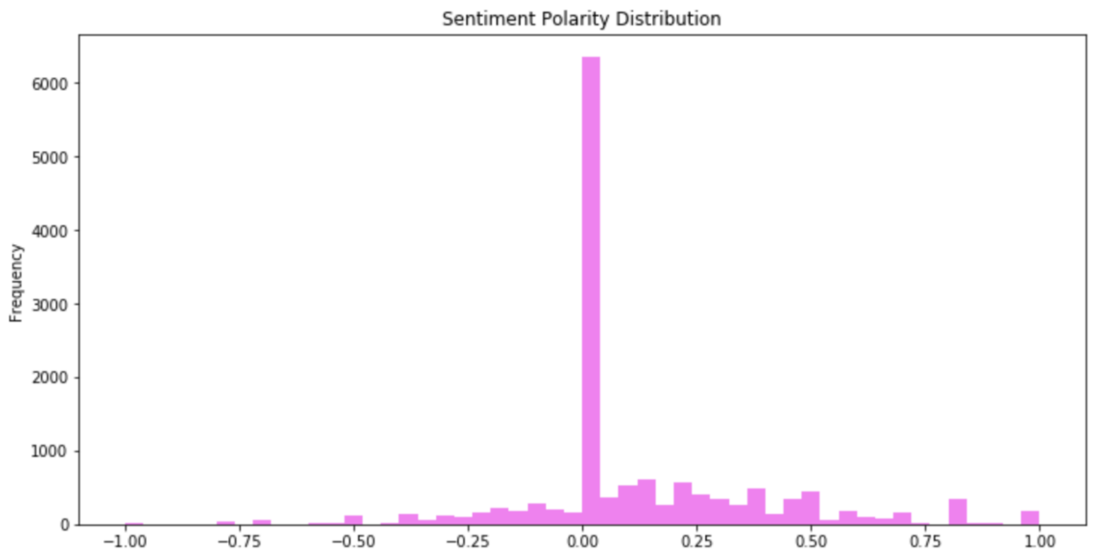
A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature level whether the expressed opinion in a document, a sentence or an entity feature is positive, negative, or neutral.

Polarity, which is also known as orientation is the emotion expressed in the sentence. Subjectivity is when the text is an explanatory article which must be analyzed in context.

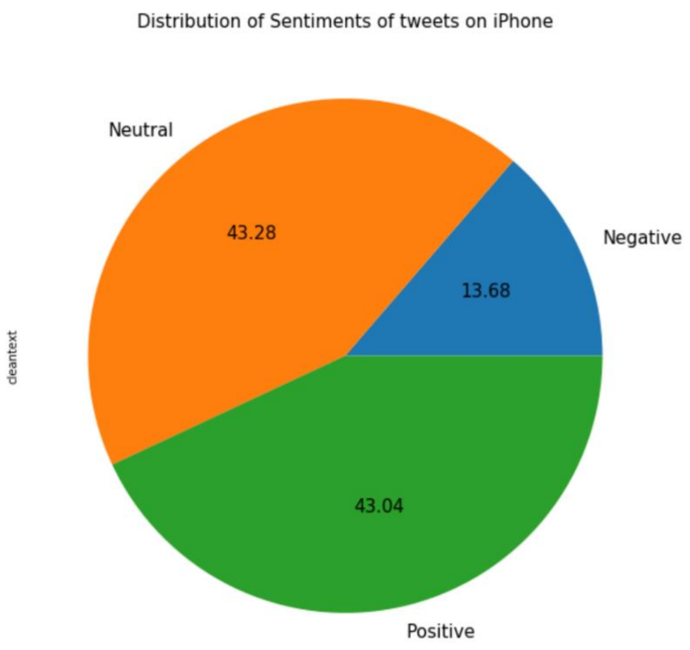
In this project, we performed sentiment analysis using **Textblob** and found the polarity for each correcttext(tweet). From this we found that majority of the sentiments were above 0, which says that they are neutral to positive. If the sentiment value is > 0 , it means that the statement is positive and if the sentiment value is == 0, then it means that the statement is neutral and if the sentiment value is < 0 then it means that the statement is negative.

With this sentiment value, we plot a histogram which explains the polarity distribution of the sentiment analysis. From the histogram it is very clear that most of the tweets have values equal to or greater than 0, which means that the overall mean polarity is neutral to positive. Comparatively, there are lesser negative valued data than positive valued data.

Below are the graphs that explains the distribution of the polarity of the tweets collected on iphone. It is apparent that most of tweets have polarity close to 0 which means then tend to be neutral.



After this, we group the sentiment value and cleantext with the function groupyby() and count all the sentiments and plot a pie graph. From this pie graph we can see that the value for neutral is 43.28% which is the highest. The value for positive is 43.04% which is almost close to neutral and the value of negative is 13.68% which is the least.
From this we can say that the iPhone models have got a neutral to positive review rather than a negative review.



TOKENIZATION AND POS-TAGGING

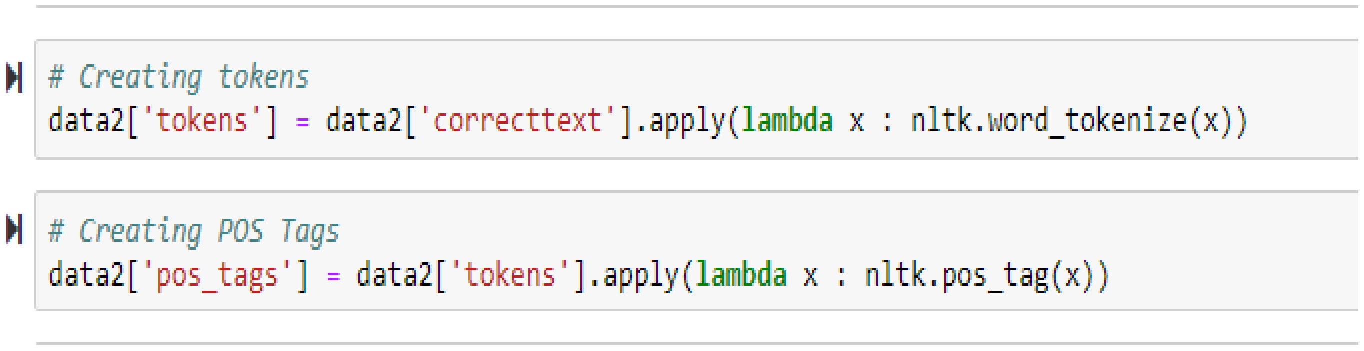
Tokenization is the process of replacing sensitive data with unique identification symbols that retain all the essential information about the data without compromising its security. Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens* , perhaps at the same time throwing away certain characters, such as punctuation.

For example, let us consider this sentence: On Tuesday, the tram was late.

This sentence is broken down into tokens such as "On" "Tuesday" "the" "tram" "was" "late".

In the above example, it throws away the punctuation i.e., comma and retains the sentence splitting each word as individual tokens.

From the natural language toolkit package, we import word_tokenize, which is used to tokenize the text into individual tokens which helps for sentiment analysis.



```
# Creating tokens
data2['tokens'] = data2['correcttext'].apply(lambda x : nltk.word_tokenize(x))
```

```
# Creating POS Tags
data2['pos_tags'] = data2['tokens'].apply(lambda x : nltk.pos_tag(x))
```

POS – Tagging:

Parts of speech tagging is also known as grammatical tagging or word category disambiguation. It is the process of marking up a word in a text as corresponding to a part of speech based on both its definition and context, its relationship with adjacent and related words in a phrase, sentence or paragraph.

In simple terms, it's the identification of words as nouns, verbs, adjectives etc.

POS can be complex, because some words can represent more than one parts of speech at different times. Hence, we use POS tagging in sentiment analysis, as it can solve problems from the root. For example, a word can, be used in two different ways in two different sentences which can have a positive or a negative impact. So, in such cases, if we segregate the words into verbs or nouns, then we can avoid the negative impacts of the sentences.

From the natural language toolkit, we import pos_tag and pos_tag_sent which are basically used for tagging the text. Pos_tag_sent is used to tag a given list of sentences each consisting of a list of tokens.

Another part of POS is chunking(ne_chunk). NLTK already has a standard Named Entity(NE) annotater. Ne_chunk needs parts-of-speech annotations to add named entity labels to the sentence. The output of ne_chunk is an ntlk.tree. The ne.chunk function acts as a chunker producing two level trees:

- Nodes on level 1: Outside any chunk
- Nodes on level 2: Inside a chunk

Using this, we can tag the words in the sentences as nouns, verbs, adverbs etc.

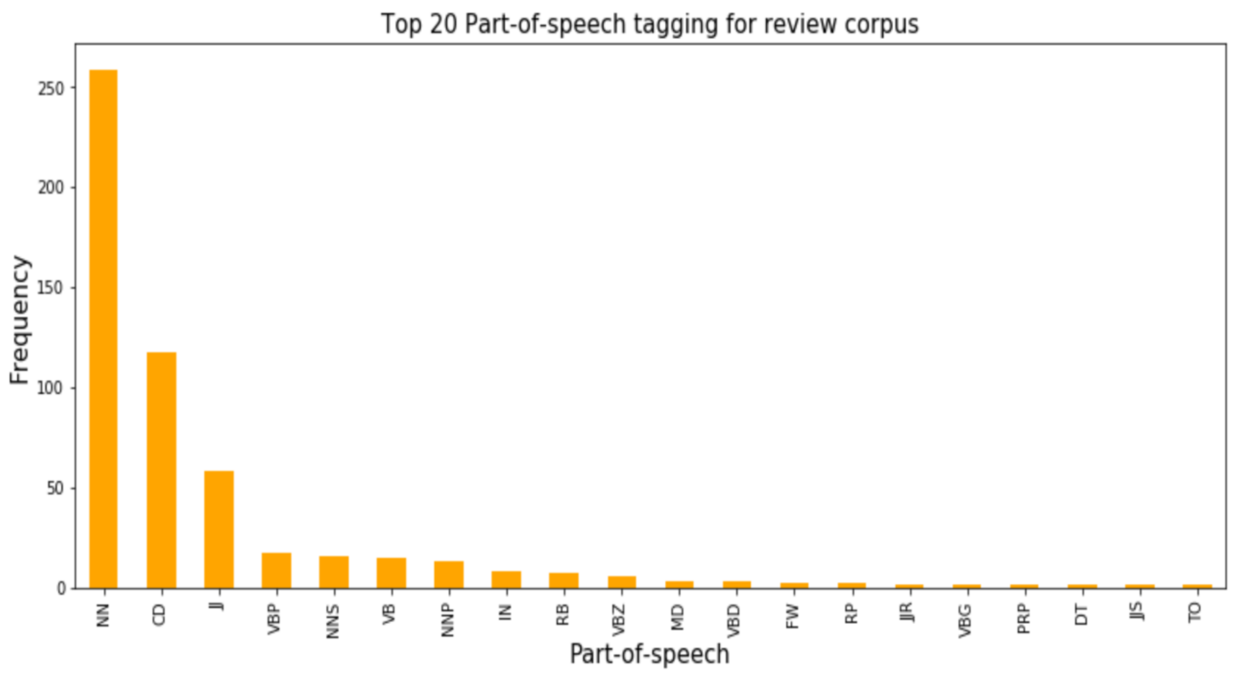
Parts of speech tagging is performed on the text, wherein it segregates each tokens in a particular string and identifies if they are nouns, verbs or adverbs and then the chunker iterates through each string separating all the nouns, verbs, adverbs and geo tags of that particular string. This is done by defining tweet_ner(chunks).

- NN -> Nouns
- JJ -> Adverbs
- VB -> verbs
- GEO -> Geo tags

ANALYSING NOUNS, VERBS AND ADJECTIVES

Top 20 POS tagging for review corpus:

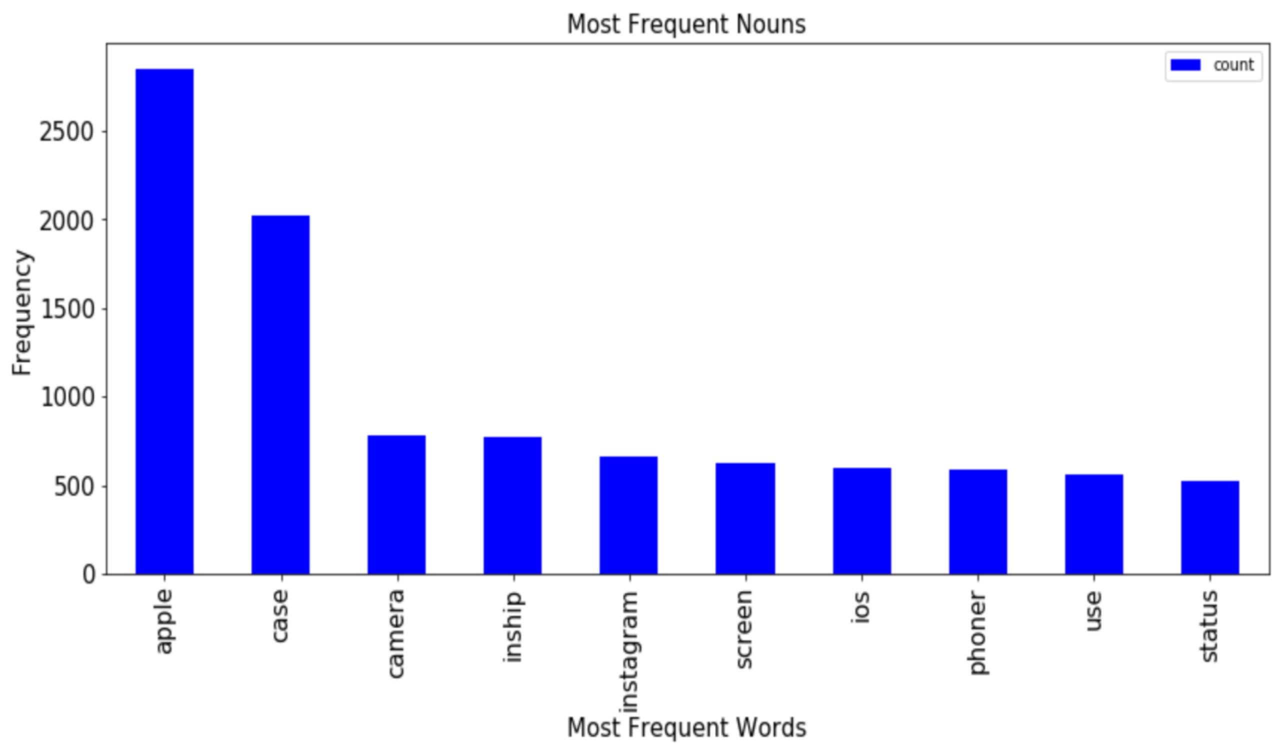
- We extracted all the string data (review corpus or tokenized user reviews) from the column correcttext of dataframe data2 and stored in blob.
- Then, a dataframe pos_df is created, which is capturing all words word tagged with its part of speech pos.
- To showcase the top 20 part of speeches used in the textual data, we used a function value_counts() which provides count of unique pos.
- To have a visual inference, we have created a bar chart to display the frequency at which Top 20 Part-of-speech tagging has occurred for.
- According to the chart, words tagged NN (noun, singular) has high frequency i.e. singular noun words have appeared more in tweets posted by the twitter users.
- CD (cardinal digit) , JJ (adjective) & VBP (verb, singular present) has shown second, third & fourth highest frequency respectively after nouns.
- As we know important features of words are nouns, verbs & adjectives which give a sense to the content, so, our focus is on these 3 features only for the sentiment analysis.



Most Frequent Nouns:

One of the most important features of textual data is Noun. Noun is a word which can denote a class of people, place or thing. According to the TextBlob library, it is denoted as “NN”. To have a clear picture of sentiments or opinion of the textual data, we need to find the frequently occurring nouns. To accomplish this, we followed the following procedure :

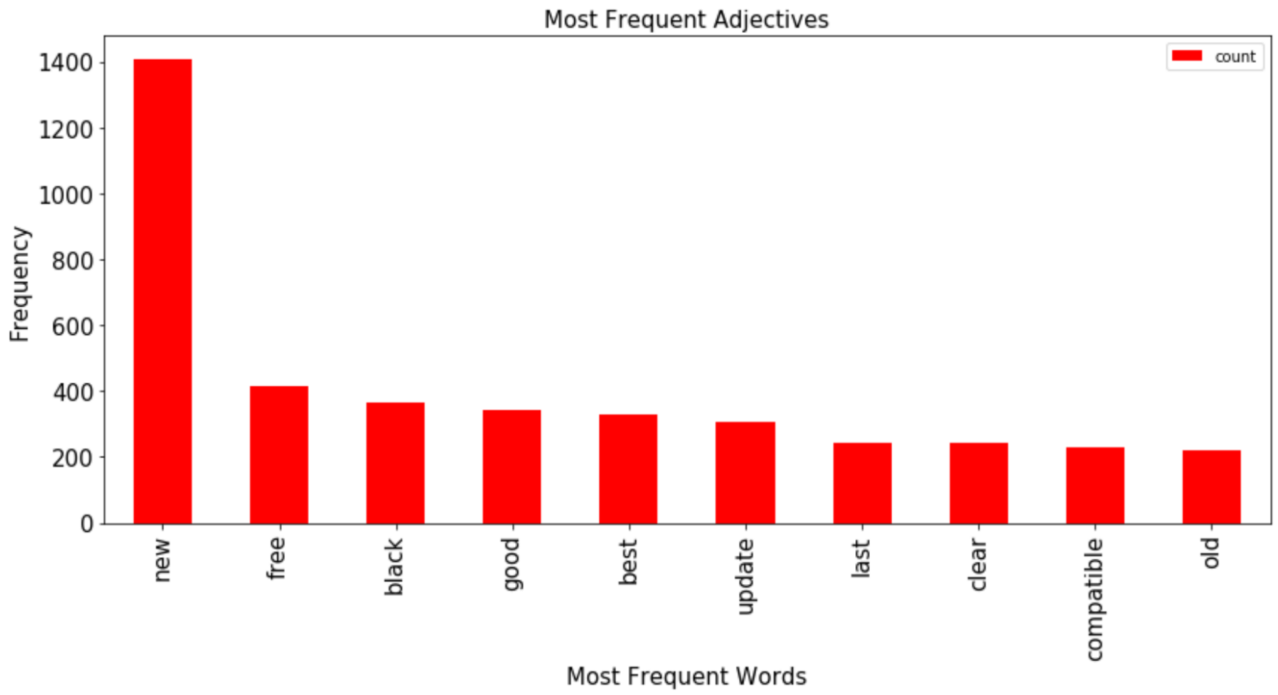
- We imported library CountVectorizer from the package from sklearn.feature_extraction.text to perform the function and declared the function in a variable vectorizer.
- Then, we applied fit transform method on the text data NN of dataset data2, where we are using a function fit_transform() which combines 2 functions : fit() and transform().
- First, the function fit() vectorizers each token by providing indices to the tokens.
- Second, the function transform(), we get the count of each token appeared in the dataset.
- Then, we used function array() on the transformed data to have an array representation of the output in a matrix format.
- We also mapped the integer indices to the feature name as column by using the function get_feature_names().
- The array of transformed data has been captured in a dataframe tokens_data_NN.
- Then, a dataframe a is created which captures the average sum of count by a function sum() , without taking values showing NA with the help of a parameter skipna declared as TRUE.
- Rename reset (could not understand)
- We created a list *list_iphone* where we included the words which we do not want to take into account as they do no contribute any meaning to the sentiment of the data: ['11','xs','pro','max','iphonex','xr','x','iphone7','iphone8','iphone8plus','iphonexsmax','iphon','com','http',' twitter','phone','appl','www','jj','cd','plu','igshid','iphonexr'].
- Then, we filtered the row *word* of dataframe *x* with all nouns with the tokens captured in the declared list, by using a function *isin().* This resulted into top 10 high frequency nouns.
- For a visual representation, we created a bar chart to represent the top 10 high frequency nouns , which are in the transformed tweet data.
- From the graph, we can observe that nouns “apple” has occurred most of the times, beating the occurrences of the noun “case”.
- The noun “status” shows the lowest frequency among all top 10 noun words.



Most Frequent Adjectives:

- Another important feature of textual data is Adjective. Adjective is a word which modifies or describes a noun. According to the TextBlob library, it is denoted as "JJ". To have a feeling of sentiments or opinion of the textual data, we need to find the frequently occurring adjectives. To accomplish this, we followed the same procedure as in case of Nouns :
- We imported library CountVectorizer from the package from sklearn.feature_extraction.text to perform the function and declared the function in a variable vectorizer.
- Then, we applied fit transform method on the text data JJ of dataset data2, where we are using a function fit_transform() which combines 2 functions : fit() and transform().
- Then, we used function array() on the transformed data to have an array representation of the output in a matrix format.
- We also mapped the integer indices to the feature name as column by using the function get_feature_names().
- The array of transformed data has been captured in a dataframe tokens_data_JJ.
- Then, a dataframe b is created which captures the average sum of count by a function sum() , without taking values showing NA with the help of a parameter skipna declared as TRUE.
- We created a list list_iphone where we included the words which we do not want to consider as they do no contribute any meaning to the sentiment of the data: ['http','appl','iphon','pro','iphone11pro','utm','iphone11','ly','gray']
- Then, we filtered the row word of dataframe x1 with all adjectives with the tokens captured in the declared list, by using a function isin(). This resulted into top 10 high frequency adjectives.

- For a visual representation, we created a bar chart to represent the top 10 high frequency adjectives , which are in the transformed tweet data.
- From the graph, we can observe that adjective "new" has occurred highest number of times. This may imply or give an idea that people are more excited with the new iPhone.
- The adjective "soft" shows the lowest frequency among all top 10 adjective words.



Most frequent Verbs:

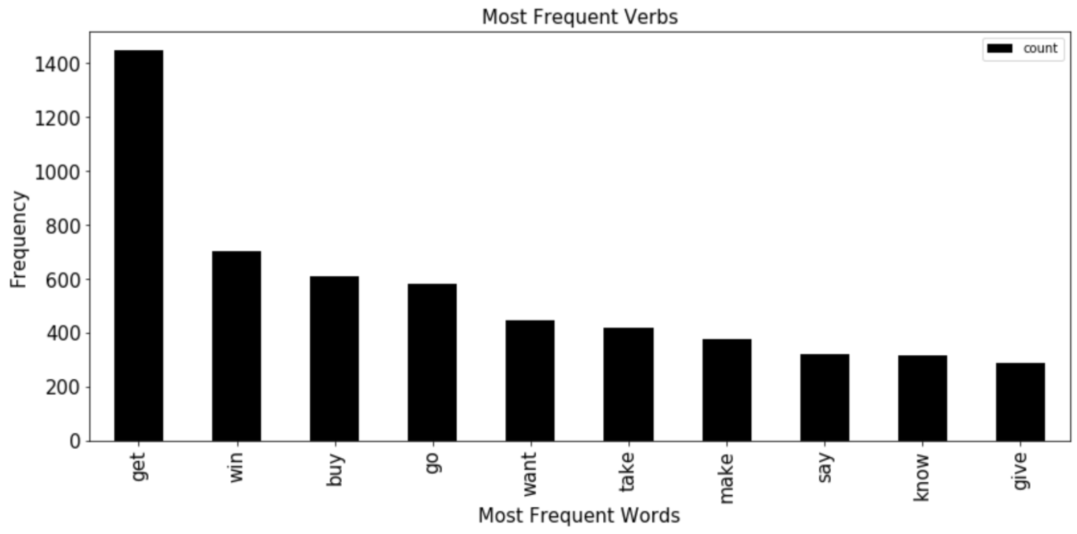
Another obvious & important feature of textual data is Verb. Verb is a word which describes action, state or occurrences of the sentence. According to the TextBlob library, it is denoted as "VB". To know the impact of the sentiments or opinion of the textual data, we need to find the frequently occurring adjectives. To accomplish this, we followed the same procedure as in case of Nouns & Adjectives :

- We imported library CountVectorizer from the package from sklearn.feature_extraction.text to perform the function and declared the function in a variable vectorizer.
- Then, we applied fit transform method on the text data VB of dataset data2, where we are using a function fit_transform() which combines 2 functions : fit() and transform().
- Then, we used function array() on the transformed data to have an array representation of the output in a matrix format.
- We also mapped the integer indices to the feature name as column by using the function get_feature_names().
- The array of transformed data has been captured in a dataframe tokens_data_VB.
- Then, a dataframe c is created which captures the average sum of count by a function sum() , without taking values showing NA with the help of a parameter skipna declared as TRUE.

- We created a list list_iphone where we included the words which we do not want to consider as they do no contribute any meaning to the sentiment of the data:

  ['11','xs','pro','max','iphonex','xr','x','iphone7','iphone8','iphone8plus','iphonexsmax','iphon','com' ,'http','twitter','phone','appl','www','jj','cd','plu','igshid','iphonexr','nn','nnp','io']

- Then, we filtered the row word of dataframe y with all adjectives with the tokens captured in the declared list, by using a function isin(). This resulted into top 10 high frequency verbs used in the tweet data.
- For a visual representation, we created a bar chart to represent the top 10 high frequency verbs , which are in the transformed tweet data.
- From the graph, we can observe that verb "get" shows highest frequency.
- The verb "give" shows the lowest frequency among all top 10 verb words.



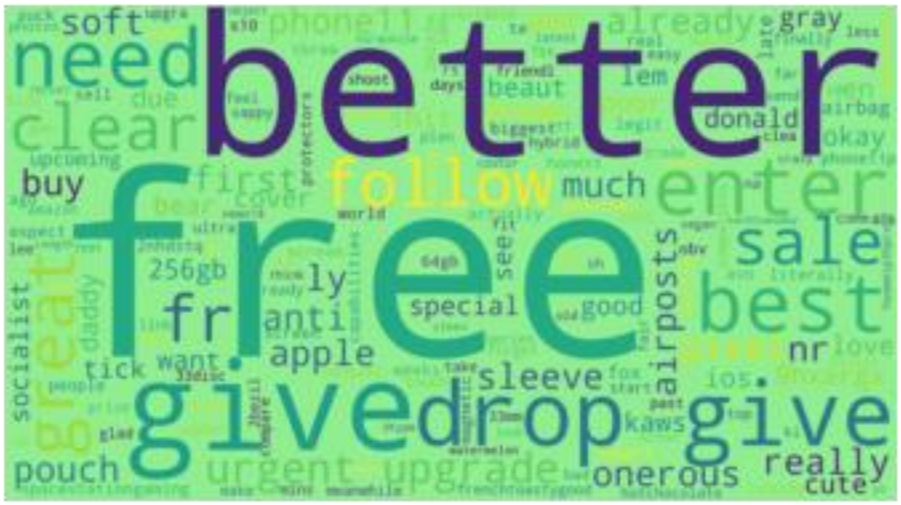
WORD CLOUD

In today's world, there is a lot of unstructured data in social media platforms especially twitter, therefore there is a need to analyze these texts which are generated. Word cloud is a very effective way to do this as it can visually interpret the text and it helps in gaining insights easily. It visualizes the word frequency in the text as weighted list.

Word cloud is a visual representation of the text data which is in the form of tags. These are basically single words. Their importance is viewed by the size of the word and the color.

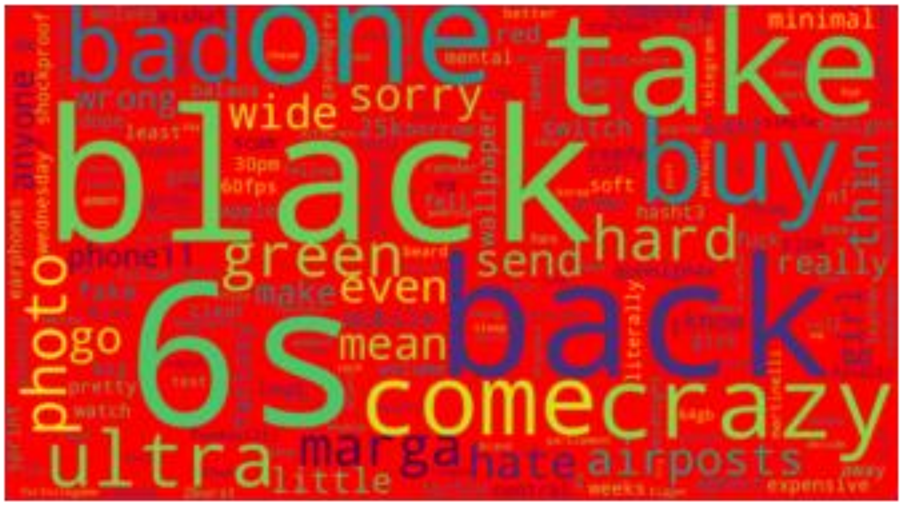
We import the package "word clouds" and update stopwords into it to generate the image representing a particular sentiment.

- The following image shows the tags that represented positivity from the text.



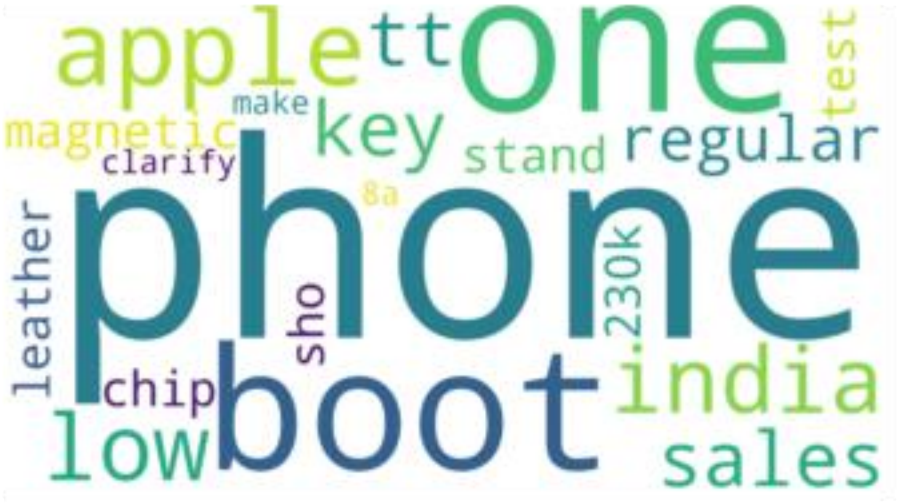
From the above image, it is evident that the words “free”, “better”, “give”, “drop”, “need”, “enter”, “clear”, “best”, “sale” etc added impact to the positive reviews.

The following image shows the tags that represented negativity from the text.



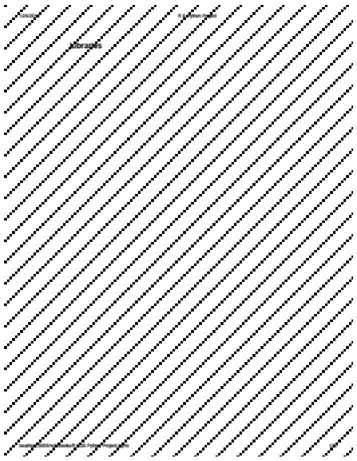
From the above image, it is evident that the words “black”, “back”, “bad”, “one”, “take”, “buy”, “ultra”, “come”, “crazy” etc added impact to the negative reviews.

- The following image shows the tags that represented neutral sentiment from the text.



From the above image, it is evident that the words “phone”, “boot”, “apple”, “one”, “india”, “sales”, “regular”, “low” etc added impact to the neutral reviews.

Python Script:



R & Python Project.ipynb