SENTIMENT ANALYSIS FOR MARKETING USING MACHINE LEARNING

TEAM MEMBERS

310821104003-ABINAYA TS
310821104034-HARINI M
310821104042-JOTHIKA K
310821104043-JULIYA A

Phase-3 DEVELOPMENT PART-1

Project:Sentiment Analysis for Marketing

INTRODUCTION:

- In Our days, people use social media networks with a unbelievable frequency, writing posts, sharing photos and videos and sending private or public messages. One of th most used social network is Tweeter. Twitter is one of the most popular social media platforms in the world, with 330 million monthly active users and 500 million tweets sent each day. That's why analyzing tweets is very important to understand how people deal with a given subject.Understanding the sentiment of tweets is important for a variety of reasons: business marketing, politics, public behavior analysis, and information gathering are just a few examples. Sentiment analysis of Twitter data can help marketers understand the customer response to product launches and marketing campaigns, and it can also help political parties understand the public response to policy changes or announcements. Since Tweeter generate a huge amount of data (6000 tweets per second).

- Sentiment analysis refers to identifying as well as classifying the sentiments that are expressed in the text source. Tweets are often useful in generating a vast amount of sentiment data upon analysis. These data are useful in understanding the opinion of the people about a variety of topics.Therefore we need to develop an Automated Machine Learning Sentiment Analysis Model in order to compute the customer perception. Due to the presence of non-useful characters (collectively termed as the noise) along with useful data, it becomes difficult to implement models on them.

📣OBJECTIVE :

In this project, we are trying to implement a Twitter sentiment analysis model that helps to overcome the challenges of identifying the sentiments of the tweets.We aim to analyze the sentiment of the tweets provided from the Sentiment140 dataset by developing a machine learning pipeline involving the use of three classifiers:

- Logistic Regression.
- Bernoulli Naive Bayes.
- Decision Tree.
- K-nearest neighbors.
- Support Vector Machine.

Along with using Term Frequency- Inverse Document Frequency (TF-IDF). The performance of these classifiers is then evaluated using accuracy, ROC-AUC Curve and F1 Scores.

1️⃣ Importing the necessary dependencies:

```
import warnings
warnings.filterwarnings('ignore')

# Importing necessary libraries and functions :
import pandas as pd
import numpy as np
from math import sqrt
import time

# Text processing libraries :
!pip install gensim
import gensim
import re # Regular Expression library
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
from gensim.parsing.preprocessing import remove_stopwords
from nltk.tokenize import word_tokenize # Tokenizaion
from spacy.lang.en import English
from spacy.lang.en.stop_words import STOP_WORDS

# Plotting libraries :
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# sklearn :
import sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.metrics import confusion_matrix, classification_report from
sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier
from sklearn.model_selection import train_test_split # Import
train_test_split function
```

```python
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
```

2Reading and loading the dataset:

In order to build our classifier model, we need a dataset which contains a huge number of tweets and the corresponding feeling being expressed at.

In any project related to the manipulation and analysis of data, we always start by collecting the data on which we are going to work. In our case, we will import our data from a `.csv` file. The dataset provided is the Sentiment140 Dataset which consists of 1,600,000 tweets that have been extracted using the Twitter API.

The various columns present in the dataset are:

- **`target`**: the polarity of the tweet (positive or negative)
- **`ids`**: Unique id of the tweet
- **`date`**: the date of the tweet
- **`flag`**: It refers to the query. If no such query exists then it is NO QUERY.
- **`user`**: It refers to the name of the user that tweeted
- **`text`**: It refers to the text of the tweet.

In[1]:

```python
# Importing the dataset :

DATASET_COLUMNS=['target','ids','date','flag','user','text']

DATASET_ENCODING = "ISO-8859-1"

df =pd.read_csv('../input/tweets/training.1600000.processed.noemoticon.csv'
, encoding=DATASET_ENCODING, names=DATASET_COLUMNS)

# Display of the first 5 lines :
df.sample(5)
```

Out[1]:

	target	ids	date	flag	user	text
652380	0	2238197273	Fri Jun 19 06:57:29 PDT 2009	NO_QUERY	TheMiss47	Agh! I made myself bleed again when I gave mys...
1220313	4	1990040256	Mon Jun 01 03:40:48 PDT 2009	NO_QUERY	Duenan	@JennaMadison ty for the follow!
1171965	4	1980566016	Sun May 31 07:05:48 PDT 2009	NO_QUERY	Mikeallnight	Ihop delievers who knew ? Haha Free breakfas...
467749	0	2175848474	Mon Jun 15 02:10:26 PDT 2009	NO_QUERY	LeahJKelly	@Trevieness no
831805	4	1557497279	Sun Apr 19 04:28:29 PDT 2009	NO_QUERY	ourellie	finished french course work;gonna paint my nails

3 Exploratory Data Analysis:

In this part, the objective is to know the imported data as much as possible, we analyze a sample, we look for the shape of the dataset, the column names, the data type information, we check if there are null values, in short, we process our data and above all we target the data (columns) that interests us, to do that we use multiple libraries such as **seaborn**, **matplotlib**, **pandas** and **numpy.**

In[2]:

```
# Display the column names of our dataset :
Df.columns
```

Out[2]:

```
Index(['target', 'ids', 'date', 'flag', 'user', 'text'],
dtype='object')
```

In[3]:

```
# Display the number of records is our dataset :
print('length of our data is {} tweets'.format(len(df)))
```

Out[3]:

```
length of our data is 1600000 tweets
```

In[4]:

```
# The shape of our data :
print("The shape of our dataset is {}".format(df. shape))
```

Out[4]:

```
The shape of our dataset is (1600000, 6)
```

In[5]:

```
# Getting info about our dataset :
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
```

Out[5]:

```
Data columns (total 6 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----
 0 target 1600000 non-null int64
 1 ids 1600000 non-null int64
 2 date 1600000 non-null object
 3 flag 1600000 non-null object
 4 user 1600000 non-null object
 5 text 1600000 non-null object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

The range index of the records starts from **0** to **1599999**

In[6]:

```
print(df.dtypes)
```

Out[6]:

```
target int64

ids int64

date object

flag object

user object

text object

dtype: object
```

- The data type of some columns in our dataset is `object`, which means we still have to process our data before getting into machine learning stuff.

In[7]:

```
# Checking for Null values :

print("number of missing values in the dataframe is
{}".format(np.sum(df.isnull().any(axis=1))))
```

Out[7]:

```
number of missing values in the dataframe is 0
```

In[8]:

```
# Rows and columns in the dataset :

print('Count of columns in the data is: ', len(df.columns))
print('Count of rows in the data is: ', len(df))
```

Out[8]:

```
Count of columns in the data is: 6

Count of rows in the data is: 1600000
```

- **1600000** is the number of records in our dataset.
- **6** is the number of columns.

In[9]:

```
# Checking unique Target Values :

df['target'].unique()
```

Out[9]:

```
array([0, 4])
```

In[10]:

```
df['target'].nunique()
```

Out[10]:

2

In[11]:

```
# Let's explore our target variable 'target'

print("the number of unique values of the target variable is
{}".format(df['target'].nunique()))
print("unique values of target variable are {0} and

{1}".format(df['target'].unique()[0],df['target'].unique()[1]))
```

Out[11]:

```
the number of unique values of the target variable is 2
unique values of target variable are 0 and 4
```

The **`target`** column is composed of just **0** and **4**

- **0** stands for `negative` sentiment.
- **4** stands for `positive` sentiment.

In[12]:

```
# Replacing the values to ease understanding :

df['target'] = df['target'].replace(4,1)
```

Out[12]:

The **`target`** column is composed of just **0** and **1**

- **0** stands for **`negative`** sentiment.
- **1** stands for **`positive`** sentiment.

👉Since the number of unique values of the Ids is less than the length of our dataset, it means that the Ids have to be repeated. in other words, **there might be tweets that have the same ID or repeat each other**

In[13]:

```
# Exploring our date feature :
print("The number of unique values of the date feature is

{}".format(df['date'].nunique()))
```

Out[13]:

```
The number of unique values of the date feature is 774363
```

In[14]:

```
# Exploring the flag feature :

print("The number of unique values of the ids feature is
{}".format(df['flag'].nunique()))

print("Unique values of ids feature are

{}".format(df['flag'].unique()[0]))
```

Out[14]:

```
The number of unique values of the ids feature is 1

Unique values of ids feature are NO_QUERY
```

👉The feature **`flag`** has the same value for all rows, which makes it insignificant for our model

In[15]:

```
# Reviewing duplicates in tweet feature :

print("The number of unique values of the text feature is
{}".format(df['text'].nunique()))
```

Out[15]:

```
The number of unique values of the text feature is 1581466
```

👉Since the number of records in our dataset is 1600000, that means there are duplicates in the tweet records.

4️⃣Data Visualization of target Variables:

After processing our data and targeting the columns we are interested in, the next step is to have a visual on our data with mathematical plots, the reason for using plots is that a plots makes the data speak more, so it become more understandable.

In[16]:

```
df.groupby('target').count()
```

Out[16]:

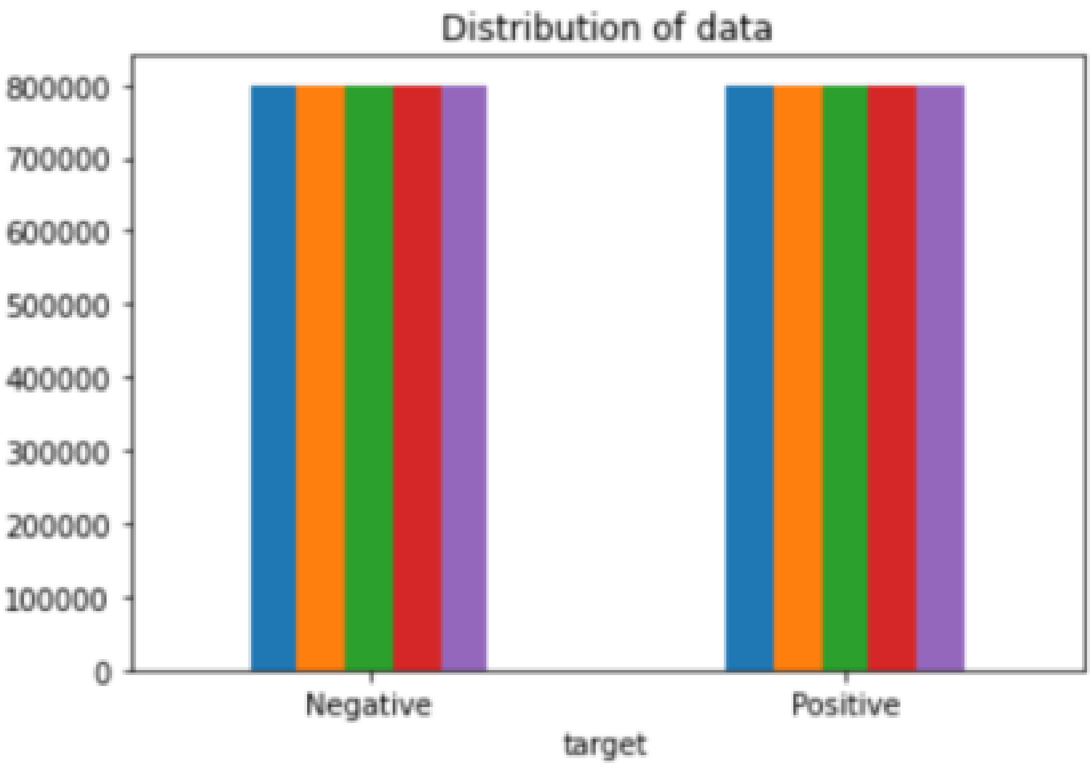
	ids	date	flag	user	text
target					
0	800000	800000	800000	800000	800000
1	800000	800000	800000	800000	800000

Since the `target` column only contains **0** or **4**, using the `.groupby()` function will result in two categories: **0** and **4**

In[17]:

```
# Plotting the distribution for dataset :

ax = df.groupby('target').count().plot(kind='bar', title='Distribution of data',legend=False)

# Naming 0 -> Negative , and 4 -> Positive

ax.set_xticklabels(['Negative','Positive'], rotation=0)

# Storing data in lists : text, sentiment =

list(df['text']), list(df['target'])
```

Out[17]:



We can see that we have an equal number of tweets with positive sentiments and negative sentiments.

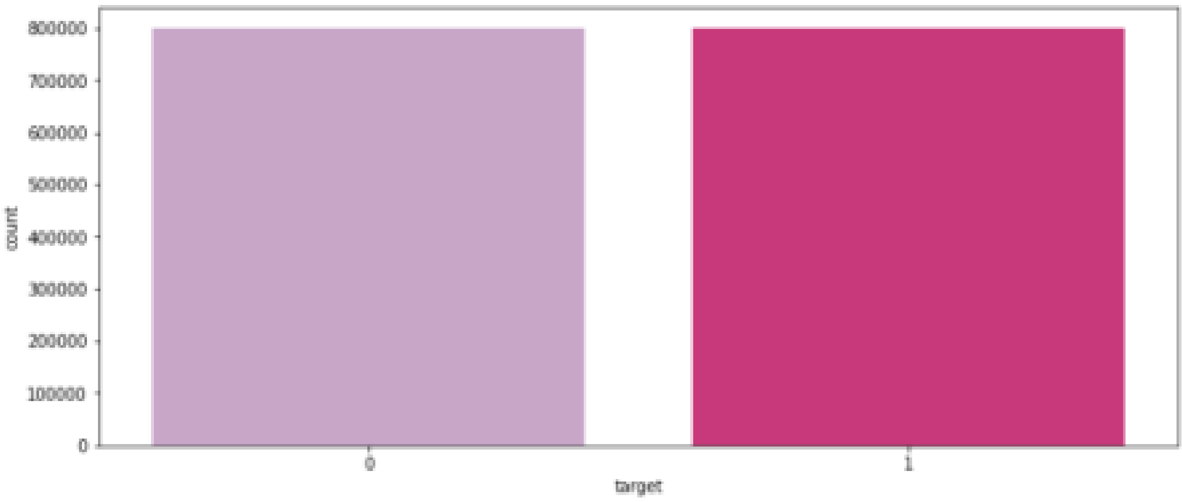
- Each color represents one of the columns : **`ids`**, **`date`**, **`flag`**, **`user`** and **`text`**.
- **`text`** variable contains the **`text`** column.
- **`sentiment`** variable contains the **`target`** column.

In[18]:

```
fig_dims = (12, 5)

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

sns.countplot(data=df, x="target", palette="PuRd")
```

Out[18]:

```
<AxesSubplot:xlabel='target', ylabel='count'>
```



- We did the same as before, we just used the **`.countplot()`** function from **`seaborn`**.

5️⃣ Data Preprocessing:

Our data generally comes from a variety of different sources and is often in a variety of different formats. For this reason, cleaning our raw data is an essential part of preparing our dataset. However, cleaning is not a simple process, as textual data often contains redundant and/or repetitive words.

Before training the model, we will perform various pre-processing steps on the dataset such as:

- Removing stop words.
- Removing emojis.
- Removing of mentions.
- Removal of numbers.
- Removal of whitespaces.
- Removal of duplicated rows.
- Removal of unuseful columns.
- Converting the text document to lowercase for better generalization.
- Cleaning the ponctuation (to reduce unnecessary noise from the dataset). ● Removing the repeating characters from the words along with removing the URLs/hyperlinks as they do not have any significant importance.

and much more, we will see this in detail later...

We will then performe:

- **`Stemming`** : reducing the words to their derived stems.
- **`Lemmatization`** : reducing the derived words to their root form known as lemma for better results.
- **`Lowering Case`**:

Lowering case is very imprtant since it allows us to make words with same value equal. This will be very useful to reduce the dimensions of our vocabulary.

In[19]:

```python
# Lowering Case :

print("=========== Before Lowering case ============\n")

print("\t" + df.loc[10, "text"])

print("\n=========== After Lowering case ===========\n")

df['text'] = df['text'].str.lower()

print("\t" + df.loc[10, "text"])
```

Out[19]:

```
=========== Before Lowering case ============

      spring break in plain city... it's snowing

=========== After Lowering case ===========

      spring break in plain city... it's snowing
```

Lower case was successfully applied to our data

- **Removal of Mentions**:

In social media, Mentions are used to call/mention another user into our post.

Generally, mentions don't have an added value to our model. So we will remove them.

A mention has a special pattern: **@UserName**, So we will remove all string which starts with @

In[20]:

```python
# Removal of Mentions:

## Creating a fucntion that will be applied to our datset :

def RemoveMentions(text):

    text_ = re.sub(r"@\S+", "", text)

    return text_

## Applying the function to each row of the data

print("=========== Before Removing Mentions ============\n")

print("\t" + df.loc[5, "text"])

print("\n=========== After Removing Mentions ===========\n")

df["text"] = df["text"].apply(RemoveMentions)

print("\t" + df.loc[5, "text"])
```

Out[20]:

```
=========== Before Removing Mentions ============

     @kwesidei not the whole crew

=========== After Removing Mentions ===========

      not the whole crew
```

Removal of Mentions was successfully applied to our data

- **Removal of Special Characters**:

Special characters are every where, since we have punctuation marks in our tweets. In order to treat, for example, **hello!** and **hello** in the same way. we have to remove the punctuation mark **!**

In[21]:

```
# Defining a list containing punctuation signs of english :

punctuations_list = string.punctuation

## Defining that will be applied to our datset :

def RemovePunctuations(text):

    transformator = str.maketrans('', '', punctuations_list)

    return text.translate(transformator)

## Applying the fucntion to all rows :

print("=========== Before Removing Punctuations =============\n")

print("\t" + df.loc[10, "text"])

print("\n=========== After Removing Punctuations \===========\n")

df["text"] = df["text"].apply(RemovePunctuations)

print("\t" + df.loc[10, "text"])
```

Out[21]:

```
=========== Before Removing Punctuations =============

      spring break in plain city... it's snowing

=========== After Removing Punctuations \===========

      spring break in plain city its snowing
```

`Removal of of Special Characters was successfully applied to our data`

- **`Removal of Stop words`**:

Stopwords are the most common words in any natural language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document.

Generally, the most common words used in a text are “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc.

Consider this text string – “There is a pen on the table”. Now, the words “is”, “a”, “on”, and “the” add no meaning to the statement while parsing it. Whereas words like “there”, “book”, and “table” are the keywords and tell us what the statement is all about.

  - **`Stopword Removal using NLTK`**:

NLTK, or the Natural Language Toolkit, is a treasure trove of a library for text preprocessing. It’s one of my favorite Python libraries. NLTK has a list of stopwords stored in 16 different languages.

In[22]:

```
df.loc[12]
```

Out[22]:

```
target                                               0
ids                                         1467812723
date                      Mon Apr 06 22:20:19 PDT 2009
flag                                          NO_QUERY
user                                              TLeC
text         i couldnt bear to watch it  and i thought the...
Name: 12, dtype: object
```

In[23]:

```
# Getting the pre defined stop words from nltk library :

stopwords = stopwords.words('english')

## Copying the df to use other libraries (spacy and gensim)

df_copy1 = df.loc[:100].copy(deep=True)

df_copy2 = df.copy(deep=True) # deep copy to create another df

## Applying the fucntion to all rows

print("=========== Before Removing Stop words ============\n")

print("\t" + df_copy2.loc[12, "text"])

print("\n=========== After Removing Stop words ===========\n")

## Exclude stopwords with Python's list comprehension and
pandas.DataFrame.apply.

df_copy2['text'] = df_copy2['text'].apply(lambda x: ' '.join([word for
word in x.split() if word not in (stopwords)]))

print("\t" + df_copy2.loc[12, "text"])
```

Out[23]:

```
=========== Before Removing Stop words ============

       i couldnt bear to watch it and i thought the ua loss was
embarrassing

=========== After Removing Stop words ===========

      couldnt bear watch thought ua loss embarrassing
```

Removal of Stop words using NLTK was successfully applied to our data

- **Stopword Removal using spaCy**:

spaCy is one of the most versatile and widely used libraries in NLP. We can quickly and efficiently remove stopwords from the given text using SpaCy. It has a list of its own stopwords that can be imported as STOP_WORDS from the spacy.lang.en.stop_words class

In[24]:

```
df.loc[12]
```

Out[24]:

```
target                                                   0
ids                                             1467812723
date                          Mon Apr 06 22:20:19 PDT 2009
flag                                              NO_QUERY
user                                                  TLeC
text         i couldnt bear to watch it  and i thought the...
Name: 12, dtype: object
```

In[25]:

```
## Creating a fucntion that will be applied to our datset :

def RemoveStopsSpacy(text):

    # Load English tokenizer, tagger, parser, NER and word vectors

    nlp = English()

    # "nlp" Object is used to create documents with linguistic
annotations.

    my_doc = nlp(text)

    # Create list of word tokens

    token_list = []

    for token in my_doc:

        token_list.append(token.text)

    # Create list of word tokens after removing stopwords

    filtered_sentence = []

    for word in token_list:

        lexeme = nlp.vocab[word]

        if lexeme.is_stop == False:

            filtered_sentence.append(word)

    return filtered_sentence

## Applying the fucntion to all rows

print("=========== Before Removing Stop words with spaCy
===========\n")

print("\t" + df_copy1.loc[12, "text"])
```

```
print("\n=========== After Removing Stop words with spaCy
===========\n")

## Exclude stopwords with Python's list comprehension and
pandas.DataFrame.apply.

df_copy1['text'] = df_copy1['text'].apply(lambda x: '
'.join(RemoveStopsSpacy(x)))

print("\t" + df_copy1.loc[12, "text"])
```

Out[26]:

```
=========== Before Removing Stop words with spaCy ===========

	i couldnt bear to watch it and i thought the ua loss was
embarrassing

=========== After Removing Stop words with spaCy ===========

	nt bear watch thought ua loss embarrassing
```

Removal of Stop words using spaCy was successfully applied to our data

- **Stopword Removal using Gensim**:

Gensim is a pretty handy library to work with on NLP tasks. While pre-processing, gensim provides methods to remove stopwords as well. We can easily import the remove_stopwords method from the class gensim.parsing.preprocessing.

In[27]:

```
df.loc[12]
```

Out[27]:

```
target                                                          0
ids                                                    1467812723
date                                 Mon Apr 06 22:20:19 PDT 2009
flag                                                     NO_QUERY
user                                                         TLeC
text          i couldnt bear to watch it  and i thought the...
Name: 12, dtype: object
```

In[28]:

```
## Applying the fucntion to all rows

print("=========== Before Removing Stop words with Gensim =======\n")

print("\t" + df.loc[12, "text"])

print("\n=========== After Removing Stop words with Gensim =======\n")

df['text'] = df['text'].apply(lambda x:
gensim.parsing.preprocessing.remove_stopwords(x))

print("\t" + df.loc[12, "text"])
```

Out[28]:

```
=========== Before Removing Stop words with Gensim =======

	i couldnt bear to watch it and i thought the ua loss was
embarrassing

=========== After Removing Stop words with Gensim =======

	bear watch thought ua loss embarrassing
```

Removal of Stop words using Gensim was successfully applied to our data

☝ We will use **Gensim** to `remove stopwords` in our case, because when we use Gensim to remove stopwords, we can use it directly on raw text. There is no need to perform tokenization before removing stop words. **It can save us a lot of time**.

- `Removal of Links/URLs`:

Tweets may contain URLs, which are not significant for our model. That's why we will remove them

In[29]:

```
## Creating a fucntion that will be applied to our datset :

def RemoveLinks(text):

    return re.sub(r"http\S+", "", text)

## Applying the fucntion to all rows of our dataset :

print("=========== Before Removing Hyperlinks =======\n")

print("\t" + df.loc[0, "text"]) # let's see for example the first row, which contains an hyperlink.

print("\n=========== After Removing Hyperlinks =======\n")

df['text'] = df['text'].apply(RemoveLinks)

print("\t" + df.loc[0, "text"])
```

Out[29]:

```
=========== Before Removing Hyperlinks =======

      httptwitpiccom2y1zl//

      awww thats bummer shoulda got david carr day

=========== After Removing Hyperlinks =======

       awww thats bummer shoulda got david carr day
```

Removal of Links/URLs was successfully applied to our data

- **Removal of numbers:**

In[30]:

```python
## Creating a fucntion that will be applied to our datset :

def RemoveNumbers(text):

    return re.sub(r"[0-9]+", "", text)

## Applying the fucntion to all rows

print("=========== Before Removing Numbers =======\n")

print("\t" + df.loc[2,"text"]) #let's see for example the thirs row,
which contains an number 50

print("\n=========== After Removing Numbers ========\n")

df['text'] = df['text'].apply(RemoveNumbers)

print("\t" + df.loc[2,"text"])
```

Out[30]:

```
=========== Before Removing Numbers =======

      dived times ball managed save 50 rest bounds

=========== After Removing Numbers ========

      dived times ball managed save rest bounds
```

Removal of numbers was successfully applied to our data

- **Removal of white spaces**:

In[31]:

```
## Creating a fucntion that will be applied to our datset :

def RemoveWhitespaces(text):

    text=text.strip() # Leading and trailing whitespaces are removed

    return re.sub(r" +"," ",text)

## Applying the fucntion to all rows :

df['text'] = df['text'].apply(lambda x: RemoveWhitespaces(x))
```

- **Removal of duplicated rows**:

As we have seen before, we may have some duplicated rows. let's check again

In[32]:

```
# And now, let's see our tweet content feature:

print("The number of unique values of the text feature is
{}".format(df['text'].nunique()))

print("The total number of rows in our dataframe is :
{}".format(len(df)))

print("The number of duplicated rows in our dataframe is :

{}".format(len(df)-df['text'].nunique()))
```

Out[32]:

```
The number of unique values of the text feature is 1461480

The total number of rows in our dataframe is : 1600000 The

number of duplicated rows in our dataframe is : 138520
```

In[33]:

```
# Removing duplicate row records but keeping original text : ( we only
keep the first duplicate )

df = df.drop_duplicates(subset='text', keep='first')
```

In[34]:

```
# Checking if duplicates have been removed:

print("The number of unique values of the text feature is
{}".format(df['text'].nunique()))

print("The total number of rows in our dataframe is :
{}".format(len(df)))

print("The number of duplicated rows in our dataframe is :

{}".format(len(df)-df['text'].nunique()))
```

Out[34]:

```
The number of unique values of the text feature is 1461480
The total number of rows in our dataframe is : 1461480
The number of duplicated rows in our dataframe is : 0
```

Removal of duplicated rows was successfully applied to our data

- **Removal of unuseful features**: We have already explained that the **ids**, **date**, **flag** and **user** features are not useful for our model. So we will drop them

In[35]:

```
# Viewing the initial dataframe columns :
df.columns
```

Out[35]:

```
Index(['target', 'ids', 'date', 'flag', 'user', 'text'],
dtype='object')
```

In[36]:

```
df=df.drop(['ids', 'date', 'flag', 'user'], axis = 1)
```

In[37]:

```
# Viewing the initial dataframe columns after dropping the unnecessary
ones :
Df.columns
```

Out[37]:

```
Index(['target', 'text'], dtype='object')
```

- `Tokenizing the text feature`:

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.

Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed. **Here is an example of tokenization** :

Input: Friends, Romans, Countrymen, lend me your ears; Output: [Friends Romans Countrymen lend me your ears]

What is `word_tokenize() ?`

- **Tokenization** is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens.
- `word_tokenize()` method. It actually returns the syllables from a single word. A single word can contain one or two syllables. **Return** : Return the list of syllables of words.

In[38]:

```
# NLTK (Natural Language Toolkit) provides a utility function for tokenizing data.

df['tokenized_tweets'] = df['text'].apply(word_tokenize)

df.head()
```

Out[38]:

	target	text	tokenized_tweets
0	0	awww thats bummer shoulda got david carr day d	[awww, thats, bummer, shoulda, got, david, car...
1	0	upset update facebook texting result school to...	[upset, update, facebook, texting, result, sch...
2	0	dived times ball managed save rest bounds	[dived, times, ball, managed, save, rest, bounds]
3	0	body feels itchy like	[body, feels, itchy, like]
4	0	behaving im mad	[behaving, im, mad]

`Tokenizer was successfully applied to our data`

What is `Stemming and lemmatization ?`

- The goal of both `stemming` and `lemmatization` is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. ● For instance:
    - `am`, `are`, `is` ⇒`be`
    - `car`, `cars`, `car's`, `cars'` ⇒ `car` The result of this mapping of text will be something like:
    - `the boy's cars are different colors` ⇒`the boy car be differ color`

- `Stemming the text feature`:

Stemming is the process of removing a part of a word, or reducing a word to its stem or root. This might not necessarily mean we're reducing a word to its dictionary root. We use a few algorithms to decide how to chop a word off. This is, for the most part, how stemming differs from lemmatization, which is reducing a word to its dictionary root, which is more complex and needs a very high degree of knowledge of a language. We'll later talk about lemmatization.

Let's assume we have a set of words — send, sent and sending. All three words are different tenses of the same root word send. So after we stem the words, we'll have just the one word — send. Similarly, if we have the words — ask, asking and asked — we can apply stemming algorithms to get the root word — ask. Stemming is as simple as that. But, unfortunately, it's not as simple as that. We will some times have complications. And these complications are called over stemming and under stemming. Let's see more about them in the next sections.

- **Over stemming** : For example, university and universe. Some stemming algorithms may reduce both the words to the stem univers, which would imply both the words mean the same thing, and that is clearly wrong.
- **Under stemming**: For example, consider the words "data" and "datum." Some algorithms may reduce these words to dat and datu respectively, which is obviously wrong.
- **Porter stemmer is a widely used stemming technique**. nltk.stem provides the utility function to stem 'PorterStemmer'

In[39]:

```
# Creating an instance of the stemmer :

stemmer = PorterStemmer()

## Creating a fucntion that will be applied to our datset :

def Stemmer(text):

    return " ".join([stemmer.stem(word) for word in

text]) ## Applying the fucntion to all rows :

df['tokenized_tweets_stemmed'] = df['tokenized_tweets'].apply(lambda

text: Stemmer(text))
```

In[40]:

```
# Checking the results :

df.head(10)
```

Out[40]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed
0	0	awww thats bummer shoulda got david carr day d	[awww, thats, bummer, shoulda, got, david, car...	awww that bummer shoulda got david carr day d
1	0	upset update facebook texting result school to...	[upset, update, facebook, texting, result, sch...	upset updat facebook text result school today ...
2	0	dived times ball managed save rest bounds	[dived, times, ball, managed, save, rest, bounds]	dive time ball manag save rest bound
3	0	body feels itchy like	[body, feels, itchy, like]	bodi feel itchi like
4	0	behaving im mad	[behaving, im, mad]	behav im mad
5	0	crew	[crew]	crew
6	0	need hug	[need, hug]	need hug
7	0	hey long time yes rains bit bit lol im fine th...	[hey, long, time, yes, rains, bit, bit, lol, i...	hey long time ye rain bit bit lol im fine than...
8	0	nope didnt	[nope, didnt]	nope didnt
9	0	que muera	[que, muera]	que muera

- **Stemming** has now been applied to the `text` column.
- **`Lemmatizing the text feature`**:

Lemmatization, unlike **Stemming**, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.

In[41]:

```
# Creating an instance of the limmatizer :

wordnet_lemmatizer = WordNetLemmatizer()

# Applying the limmatizer to all rows:

df['tokenized_tweets_stemmed_lemmatized'] =
df['tokenized_tweets_stemmed'].apply( lambda text:
wordnet_lemmatizer.lemmatize(text, pos="v"))
```

In[42]:

```
df.head(50)
```

Out[42]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized
0	0	awww thats bummer shoulda got david carr day d	[awww, thats, bummer, shoulda, got, david, car...	awww that bummer shoulda got david carr day d	awww that bummer shoulda got david carr day d
1	0	upset update facebook texting result school to...	[upset, update, facebook, texting, result, sch...	upset updat facebook text result school today ...	upset updat facebook text result school today ...
2	0	dived times ball managed save rest bounds	[dived, times, ball, managed, save, rest, bounds]	dive time ball manag save rest bound	dive time ball manag save rest bound
3	0	body feels itchy like	[body, feels, itchy, like]	bodi feel itchi like	bodi feel itchi like
4	0	behaving im mad	[behaving, im, mad]	behav im mad	behav im mad
5	0	crew	[crew]	crew	crew
6	0	need hug	[need, hug]	need hug	need hug
7	0	hey long time yes rains bit bit lol im fine th...	[hey, long, time, yes, rains, bit, bit, lol, i...	hey long time ye rain bit bit lol im fine than...	hey long time ye rain bit bit lol im fine than...
8	0	nope didnt	[nope, didnt]	nope didnt	nope didnt
9	0	que muera	[que, muera]	que muera	que muera
10	0	spring break plain city snowing	[spring, break, plain, city, snowing]	spring break plain citi snow	spring break plain citi snow
11	0	repierced ears	[repierced, ears]	repierc ear	repierc ear
12	0	bear watch thought ua loss embarrassing	[bear, watch, thought, ua, loss, embarrassing]	bear watch thought ua loss embarrass	bear watch thought ua loss embarrass
13	0	counts idk talk anymore	[counts, idk, talk, anymore]	count idk talk anymor	count idk talk anymor
14	0	wouldve didnt gun zac snyders doucheclown	[wouldve, didnt, gun, zac, snyders, doucheclown]	wouldv didnt gun zac snyder doucheclown	wouldv didnt gun zac snyder doucheclown
15	0	wish got watch miss premiere	[wish, got, watch, miss, premiere]	wish got watch miss premier	wish got watch miss premier
16	0	hollis death scene hurt severely watch film wr...	[hollis, death, scene, hurt, severely, watch, ...	holli death scene hurt sever watch film wri di...	holli death scene hurt sever watch film wri di...
17	0	file taxes	[file, taxes]	file tax	file tax
18	0	ahh ive wanted rent love soundtrack	[ahh, ive, wanted, rent, love, soundtrack]	ahh ive want rent love soundtrack	ahh ive want rent love soundtrack
19	0	oh dear drinking forgotten table drinks	[oh, dear, drinking, forgotten, table, drinks]	oh dear drink forgotten tabl drink	oh dear drink forgotten tabl drink
20	0	day didnt	[day, didnt]	day didnt	day didnt
21	0	friend called asked meet mid valley todaybut i...	[friend, called, asked, meet, mid, valley, tod...	friend call ask meet mid valley todaybut ive t...	friend call ask meet mid valley todaybut ive t...
22	0	baked cake ated	[baked, cake, ated]	bake cake ate	bake cake ate
23	0	week going hoped	[week, going, hoped]	week go hope	week go hope
24	0	blagh class tomorrow	[blagh, class, tomorrow]	blagh class tomorrow	blagh class tomorrow
25	0	hate wake people	[hate, wake, people]	hate wake peopl	hate wake peopl
26	0	going sleep watching marley	[going, sleep, watching, marley]	go sleep watch marley	go sleep watch marley
27	0	im sad misslilly	[im, sad, misslilly]	im sad misslilli	im sad misslilli
28	0	ooooh lol leslie ok wont leslie wont mad	[ooooh, lol, leslie, ok, wont, leslie, wont, mad]	ooooh lol lesli ok wont lesli wont mad	ooooh lol lesli ok wont lesli wont mad
29	0	meh lover exception track gets depressed time	[meh, lover, exception, track, gets, depressed...	meh lover except track get depress time	meh lover except track get depress time
30	0	some hacked account aim new	[some, hacked, account, aim, new]	some hack account aim new	some hack account aim new
31	0	want promote gear groove unfornately ride b go...	[want, promote, gear, groove, unfornately, rid...	want promot gear groov unforn ride b go anaheim	want promot gear groov unforn ride b go anaheim
32	0	thought sleeping option tomorrow realizing eva...	[thought, sleeping, option, tomorrow, realizin...	thought sleep option tomorrow realiz evalu mor...	thought sleep option tomorrow realiz evalu mor...
33	0	awe love miss	[awe, love, miss]	awe love miss	awe love miss
34	0	asian eyes sleep night	[asian, eyes, sleep, night]	asian eye sleep night	asian eye sleep night
35	0	ok im sick spent hour sitting shower cause sic...	[ok, im, sick, spent, hour, sitting, shower, c...	ok im sick spent hour sit shower caus sick sta...	ok im sick spent hour sit shower caus sick sta...
36	0	ill tell ya story later good day ill workin li...	[ill, tell, ya, story, later, good, day, ill, ...	ill tell ya stori later good day ill workin li...	ill tell ya stori later good day ill workin li...
37	0	sorry bed time came gmt	[sorry, bed, time, came, gmt]	sorri bed time came gmt	sorri bed time came gmt

38	0	dont depressing dont think want know kids suit...	[dont, depressing, dont, think, want, know, ki...	dont depress dont think want know kid suitcas	dont depress dont think want know kid suitcas
39	0	bed class work gym class day thats gonna fly m...	[bed, class, work, gym, class, day, thats, gon...	bed class work gym class day that gon na fli m...	bed class work gym class day that gon na fli m...
40	0	dont feel like getting today got study tomorro...	[dont, feel, like, getting, today, got, study,...	dont feel like get today got studi tomorrow pr...	dont feel like get today got studi tomorrow pr...
41	0	hes reason teardrops guitar break heart	[hes, reason, teardrops, guitar, break, heart]	he reason teardrop guitar break heart	he reason teardrop guitar break heart
42	0	sad sad sad dont know hate feeling wanna sleep	[sad, sad, sad, dont, know, hate, feeling, wan...	sad sad sad dont know hate feel wan na sleep	sad sad sad dont know hate feel wan na sleep
43	0	awww soo wish finally comfortable im sad missed	[awww, soo, wish, finally, comfortable, im, sa...	awww soo wish final comfort im sad miss	awww soo wish final comfort im sad miss
44	0	falling asleep heard tracy girls body sad hear...	[falling, asleep, heard, tracy, girls, body, s...	fall asleep heard traci girl bodi sad heart br...	fall asleep heard traci girl bodi sad heart br...
45	0	yay im happy job means time	[yay, im, happy, job, means, time]	yay im happi job mean time	yay im happi job mean time
46	0	checked user timeline blackberry looks like tw...	[checked, user, timeline, blackberry, looks, l...	check user timelin blackberri look like twank ...	check user timelin blackberri look like twank ...
47	0	oh maneas ironing fave wear meeting burnt	[oh, maneas, ironing, fave, wear, meeting, burnt]	oh manea iron fave wear meet burnt	oh manea iron fave wear meet burnt
48	0	strangely sad lilo samro breaking	[strangely, sad, lilo, samro, breaking]	strang sad lilo samro break	strang sad lilo samro break
49	0	oh im sorry didnt think retweeting	[oh, im, sorry, didnt, think, retweeting]	oh im sorri didnt think retweet	oh im sorri didnt think retweet

- **Lemmatizer** has now been applied to the `text` column.

☝ After preprocessing our data, we are a step away from using our dataframe for our machine/deep learning models, but before we'll save the new preprocessed dataframe as a **csv** file that we will use later.