SENTIMENT ANALYSIS FOR MARKETING USING MACHINE LEARNING

TEAM MEMBERS

310821104003-ABINAYA TS

310821104034-HARINI M

310821104042-JOTHIKA K

310821104043-JULIYA A

Phase-5 SUBMISSION DOCUMENT

Project:Sentiment Analysis for Marketing:

Topic:In this section we will documented the complete project and prepare it for submission.

🐦 Tweeter Sentiment Analysis 📈

🔰 INTRODUCTION:

- In today's digital age, businesses are inundated with an unprecedented volume of customer feedback and opinions across various online platforms, including social media, review websites, and forums. Understanding and leveraging this wealth of information is crucial for informed decision-making, product improvement, and effective marketing strategies. Sentiment analysis, a subfield of natural language processing (NLP), plays a pivotal role in helping businesses gain insights from unstructured text data and gauge public sentiment toward their products, services, or brand. In this introduction, we'll explore the fundamentals of sentiment analysis for marketing, with a focus on how machine learning techniques can be employed to extract valuable insights.

- 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.

1).Data Collection:

Gather customer feedback data from various sources such as online reviews, social media, forums, or surveys. Ensure that the data includes feedback related to your competitors' product.

2).Data Preprocessing:

Preprocess the customer feedback data to clean and prepare it for sentiment analysis. Common preprocessing steps include:

- Lowercasing the text.
- Removing special characters and punctuation
- Tokenization (splitting text into words or phrases).
- Removing stop words.
- Lemmatization or stemming.

3).Sentiment Insights:

Analyze the sentiment results to gain insights into competitor products. Look for patterns and common sentiments in the feedback, and consider the following aspects:

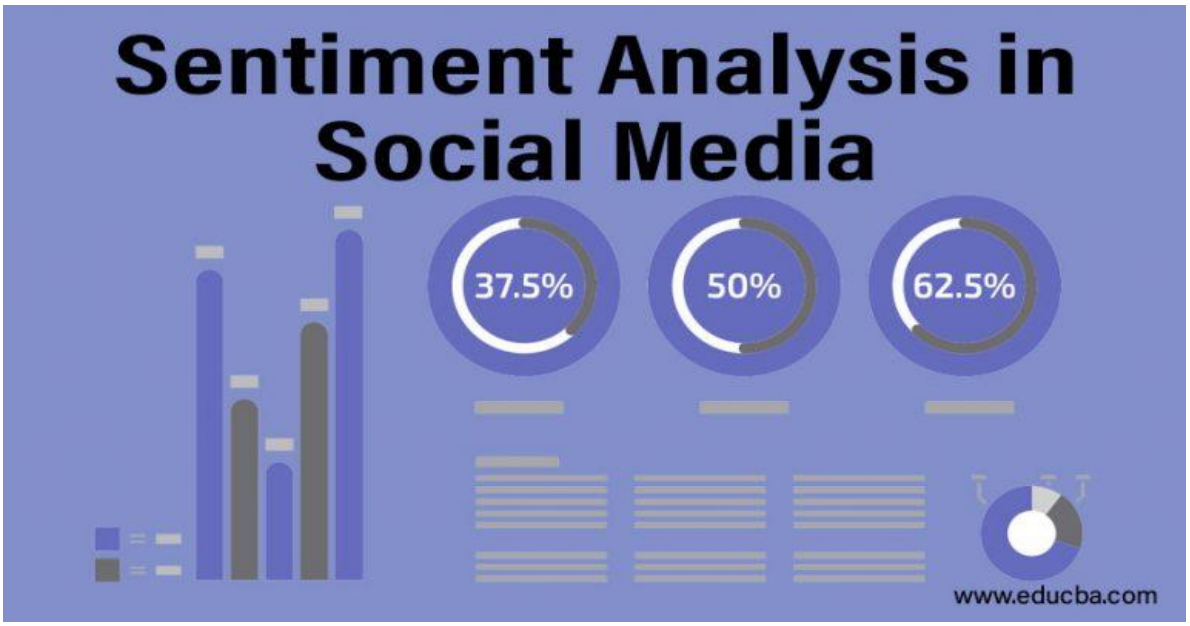
- Overall sentiment distribution (positive, negative, neutral) across feedback.
- Specific positive aspects or strengths of competitors' products mentioned by customers.
- Common pain points or weaknesses that customers are dissatisfied with.
- Trends and changes in sentiment over time.

4).Competitive Analysis:

Compare the sentiment analysis results for competitor products with your own products to identify areas where you can improve or differentiate your offerings.

5).Visualisation:

Create visualisations (e.g., charts, word clouds) to present the sentiment insights in a clear and digestible format for decision-makers. Actionable Recommendations: Based on the sentiment insights, formulate actionable recommendations for your marketing and product development strategies. Consider addressing customer pain points, emphasising your product's strengths, or adjusting pricing and marketing message acoordingly.



PRE-TRAINED MODELS:

Fine-tuning involves taking a pre-trained model and training it further on a smaller dataset of labeled data specific to your task. This allows the model to learn the nuances of your dataset and improve its performance on your task.

❖ Here are some advanced techniques for fine-tuning pre-trained sentiment analysis models:

- **Use a domain-specific dataset**
- **Use transfer learning**
- **Freeze the lower layers**
- **Use a smaller learning rate**
- **Use a regularization technique**

❖ **Here are the key steps:**

1. **Install Required Libraries:**

First, make sure you have the necessary libraries installed, including `transformers`, `torch`, and `scikit-learn`.

2. **Data Preparation:**

You'll need a labeled dataset for sentiment analysis. Ensure your dataset is in a format that includes text samples and corresponding sentiment labels (e.g., positive, negative, neutral).

3. **Load Pre-trained Model:**

You can choose from various pre-trained BERT-based models like BERT, RoBERTa, or others. Load the model and tokenizer from the Hugging Face Transformers library.

4. **Data Preprocessing:**

Tokenize and preprocess your dataset. This includes converting text to input features compatible with the model.

5. **Fine-tuning the Model:**

Define your training loop, loss function, and optimizer. Fine-tune the model on your sentiment dataset.
(device)

6. **Evaluate the Model:**

After training, evaluate the model on a separate validation dataset to assess its performance

7. Inference:

Use the fine-tuned model for sentiment analysis on new text samples.

8. Hyperparameter Tuning:

You may need to experiment with hyperparameters such as batch size, learning rate, and the number of training epochs to optimize model performance.

9. Save and Load the Model:

You can save your fine-tuned model for future use and load it when needed.

📣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.

Building and Developing The Project :

1️⃣Importing 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
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]:

```
# Importing the dataset :

DATASET_COLUMNS=['target','ids','date','flag','user','t

ext'] DATASET_ENCODING = "ISO-8859-1"

df
=pd.read_csv('../input/tweets/training.1600000.processed.noemoticon.cs
v' , 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

3Exploratory 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.

4Data 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**

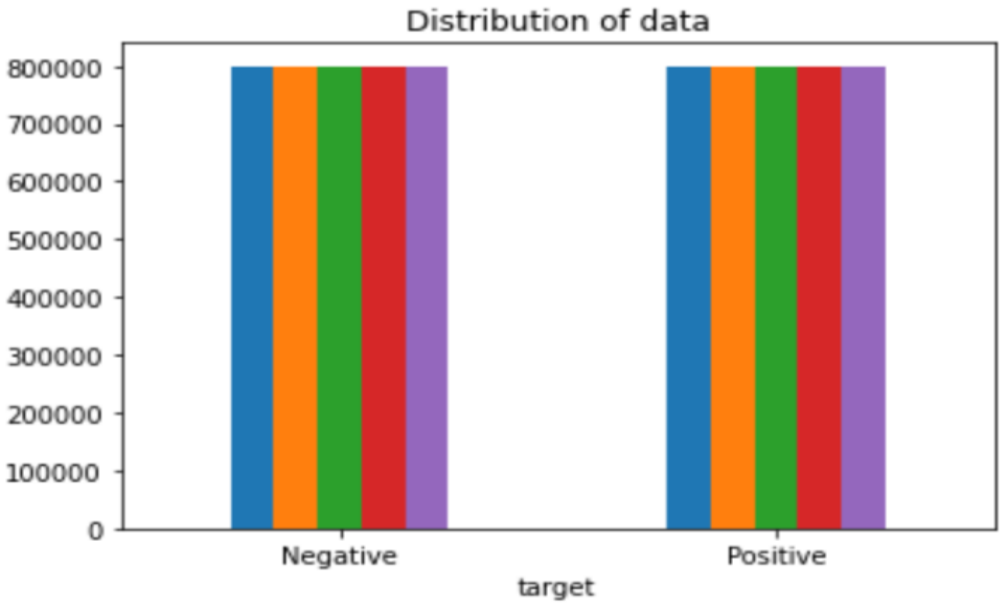
In[17]:

```
# 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]:



- Each color represents one of the columns : **`ids`**, **`date`**, **`flag`**, **`user`** and **`text`**

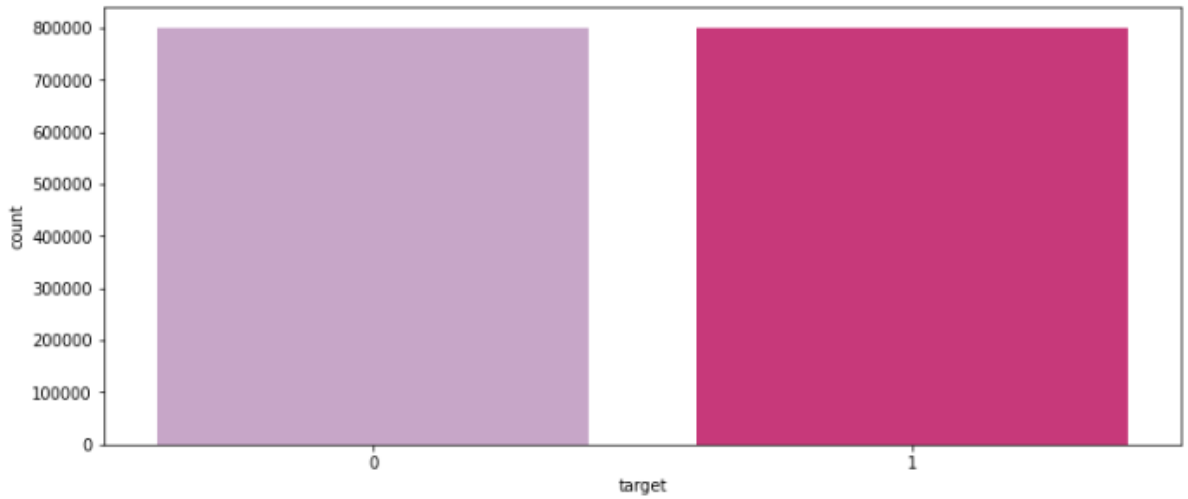
- **`text`** variable contains the **`text`** column.
- **`sentiment`** variable contains the **`target`** column.

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`.

5Data 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.

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]:

```
# 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]:

```
# 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]:

```
## 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 manwas ironing fave wear meeting burnt	[oh, manwas, ironing, fave, wear, meeting, burnt]	oh manwa iron fave wear meet burnt	oh manwa 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.

6Data Visualization after preprocessing:

Before performing the machine learning, **let's have a general idea of the accuracy of our data**, to do this we will use a **word cloud** which is a collection, or group, of words represented in different sizes. The bigger and bolder the word appears, the more often it is mentioned in a given text and the more important it is.

Which means that in our case we expect a **word cloud** to contain a sample of words representing the category we are plotting

we'll generate a **word cloud** for **positive tweets** and another for **negative tweets** to see which are the most commonly used words for each tweet category.

In[43]:

```
df.head(2)
```

Out[43]:

	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 ...

In[44]:

```
# Let's create a function which creates a wordcloud of a given pandas Series object :
def wordCloud(data_pos, max_words):
    # call the wordcloud function to show the most top 1000 used words:
    cloud = WordCloud(max_words=max_words, background_color="white", width=1600, height=800,
                      collocations=False).generate(" ".join(data_pos))
    plt.figure(figsize=(20, 20))
    plt.imshow(cloud)
```

Generating a `word cloud` for positive tweets :

In[45]:

```
wordCloud(df.loc[df["target"] == 1, "text"],2000)
```

Out[45]:

- As the picture shows, a lot of **positive words** appear: love, thank , haha, new, lol, great, nice, excited, happy, ready…

Generating a `word cloud` for negative tweets :

In[46]:

```
wordCloud(df.loc[df["target"] == 0, "text"], 2000)
```

Out[47]:

- As the picture shows, a lot of **negative words** appear: bad, sad, wish, need, sorry...

`seeking to gather more information about our data`

Now, let's compare the length of tweets from each sentiment category and see if there is a relationship between tweet sentiment and tweet length.

In[48]:

```
# Calculating tweet's lenght :
```

```
df["text_length"] = df["text"].apply(len)
```

```
# let's show the mean word count of each sentiment :
```

```
round(pd.DataFrame(df.groupby("target").text_length.mean()),2)
```

Out[48]:

	text_length
target	
0	41.25
1	40.92

We can see that positive and negative sentiment have the same average text length, which means the **sentiment** and tweet **length** are **independent** variables.

7Splitting our data into Train & Test Subset:

⚠⚠⚠ For performance reasons, for **some models** we will use the **full dataset**, for other **computationally heavy models** we will use **10% of the original dataset**.⚠⚠⚠.Creating a new variable **`df_reduced`** that contains a shuffled sample of the dataset :

In[49]:

```
# Generating one row :
```

```
df_reduced = df.sample(frac =.10)
```

```
  # Displaying the reduced dataset :
```

```
df_reduced
```

Out[49]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized	text_length
839773	1	bathroom series ellen tooo funny day	[bathroom, series, ellen, tooo, funny, day]	bathroom seri ellen tooo funni day	bathroom seri ellen tooo funni day	36
643717	0	feeling tired anxious today upsets home younge...	[feeling, tired, anxious, today, upsets, home,...	feel tire anxiou today upset home youngest son...	feel tire anxiou today upset home youngest son...	72
256786	0	online soon girl ill diie dont want dying love x	[online, soon, girl, ill, diie, dont, want, dy...	onlin soon girl ill diie dont want die love x	onlin soon girl ill diie dont want die love x	48
208926	0	lost send game	[lost, send, game]	lost send game	lost send game	14
1216154	1	ecstasy key treating ptsd like ptsd ecstasy tr...	[ecstasy, key, treating, ptsd, like, ptsd, ecs...	ecstasi key treat ptsd like ptsd ecstasi treat...	ecstasi key treat ptsd like ptsd ecstasi treat...	73
...	...	...	...	...	...	...
583215	0	rainy day catching episodes ncis	[rainy, day, catching, episodes, ncis]	raini day catch episod nci	raini day catch episod nci	32
900488	1	eighty min sec new record	[eighty, min, sec, new, record]	eighti min sec new record	eighti min sec new record	25
369744	0	till sale spent money like	[till, sale, spent, money, like]	till sale spent money like	till sale spent money like	26
1248856	1	got ma pink bikini b like twins	[got, ma, pink, bikini, b, like, twins]	got ma pink bikini b like twin	got ma pink bikini b like twin	31
1589701	1	working fun today loads hot guys	[working, fun, today, loads, hot, guys]	work fun today load hot guy	work fun today load hot guy	32

146148 rows × 6 columns

In[50]:

```
print( "The shape of the original dataset: " + str(df.shape))
print( "The shape of the reduced dataset: " + str(df_reduced.shape))
```

Out[50]:

```
The shape of the original dataset: (1461480, 6)
The shape of the reduced dataset: (146148, 6)
```

🙌 You can see here **reduced dataset** equals 10% of the **original dataset**

In[51]:

```
# Separating input feature and label :
X = df["tokenized_tweets_stemmed_lemmatized"]
y = df["target"]

X_reduced = df_reduced["tokenized_tweets_stemmed_lemmatized"]
y_reduced = df_reduced["target"]
```

In[52]:

```
# Separating the 85% data for training data and 15% for testing data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.15, random_state=100)
X_train_reduced, X_test_reduced, y_train_reduced, y_test_reduced =
train_test_split(X_reduced, y_reduced,
test_size=0.15, random_state=100)
```

- `random_state` is basically used for reproducing your problem the same every time it is run. If we do not use a **random_state** in **train_test_split**, every time you make the split we might get a different set of train and test data points and will not help in debugging in case we get an issue.
- `X` contains `df["tokenized_tweets_stemmed_lemmatized"]`
- `y` contains = `df["target"]`
- `X_train` contains **85%** of `df["tokenized_tweets_stemmed_lemmatized"]`
- `X_test` contains **15%** of `df["tokenized_tweets_stemmed_lemmatized"]`
- `y_train` contains **85%** of `df["target"]`
- `y_test` contains **15%** of `df["target"]`

⚠️ The same goes for the reduced variables !

8️⃣Word Embedding and Transforming Dataset using TF-IDF Vectorizer :

NLP experts developed a technique called **word embeddings** that convert words into their numerical representations. Once converted, **NLP algorithms can easily digest these learned representations to process textual information.** Word embeddings map the words as real-valued numerical vectors. It does so by tokenizing each word in a sequence (or sentence) and converting them into a vector space. Word embeddings aim to capture the semantic meaning of words in a sequence of text. It assigns similar numerical representations to words that have similar meanings.

Simply, these words **need to be made meaningful for machine learning or deep learning algorithms**. Therefore, **they must be expressed numerically**. Algorithms such as **One Hot Encoding**, **TF-IDF**, **Word2Vec**, **FastText** enable words to be expressed mathematically as word embedding techniques used to solve such problems

`One-hot encoding` is an important step for preparing our dataset for use in machine learning.

One-hot encoding `turns your categorical data into a binary vector representation`. Pandas get dummies makes this very easy!

- This means that for each unique value in a column, a new column is created. The values in this column are represented as 1s and 0s, depending on whether the value matches the column header.
- For example, with the help of the `get_dummies` function, we turn this table below :

Gender
Male
Female
Male
Male

  - To this :

Gender	Male	Female
Male	1	0
Female	0	1
Male	1	0
Male	1	0

- **`Bag Of Words`**:

The **bag-of-words** model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

Bag of Words (BOW) is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set

In[53]:

```
# Quick overview of our dataset:
df.head()
```

Out[53]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized	text_length
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	46
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 ...	54
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	41
3	0	body feels itchy like	[body, feels, itchy, like]	bodi feel itchi like	bodi feel itchi like	21
4	0	behaving im mad	[behaving, im, mad]	behav im mad	behav im mad	15

In[54]:

```
# Quick overview of our reduced dataset:
df_reduced.head()
```

Out[54]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized	text_length
839773	1	bathroom series ellen tooo funny day	[bathroom, series, ellen, tooo, funny, day]	bathroom seri ellen tooo funni day	bathroom seri ellen tooo funni day	36
643717	0	feeling tired anxious today upsets home younge...	[feeling, tired, anxious, today, upsets, home,...	feel tire anxiou today upset home youngest son...	feel tire anxiou today upset home youngest son...	72
256786	0	online soon girl ill diie dont want dying love x	[online, soon, girl, ill, diie, dont, want, dy...	onlin soon girl ill diie dont want die love x	onlin soon girl ill diie dont want die love x	48
208926	0	lost send game	[lost, send, game]	lost send game	lost send game	14
1216154	1	ecstasy key treating ptsd like ptsd ecstasy tr...	[ecstasy, key, treating, ptsd, like, ptsd, ecs...	ecstasi key treat ptsd like ptsd ecstasi treat...	ecstasi key treat ptsd like ptsd ecstasi treat...	73

In[55]:

```
# Fit the TF-IDF Vectorizer :
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=10000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

Out[55]:

```
No. of feature_words:  10000
```

In[56]:

```
# Fit the TF-IDF Vectorizer :
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=1000)
vectoriser.fit(X_train_reduced)
```

```
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

Out[56]:

```
No. of feature_words:  1000
```

In[57]:

```
# Transform the data using TF-IDF Vectorizer :
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)

X_train_reduced = vectoriser.transform(X_train_reduced)
X_test_reduced  = vectoriser.transform(X_test_reduced)
```

9Function for Model Evaluation :

After training the model we then apply the evaluation measures to check how the model is performing. Accordingly, we use the following evaluation parameters to check the performance of the models respectively :

- **Accuracy Score** : Typically, the accuracy of a predictive model is good (above 90% accuracy)
- **ROC-AUC Curve** : The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.
- **Confusion Matrix with Plot** : A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.
  - **Actual values** are the columns.
  - **Predicted values** are the lines.

	Positive	Negative
Positive	TP	TN
Negative	FP	TN

In[58]:

```
def model_Evaluate(model):

# Predict values for Test dataset

    y_pred = model.predict(X_test)

    # Print the evaluation metrics for the dataset.

    print(classification_report(y_test, y_pred))

    # Compute and plot the Confusion matrix

    cf_matrix = confusion_matrix(y_test, y_pred)

    categories = ['Negative','Positive']

    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten() / np.sum(cf_matrix)]

    labels = [f'{v1}n{v2}' for v1, v2 in
zip(group_names,group_percentages)]

    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',

   xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad =
10)

    plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)
```

```
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

- To avoid each time and for each model, drawing the confusion matrix, printing the precision, the f1-score... we just define the **`model Evaluate()`** function which will do the job each time.

🔟 Model Building:

In the problem statement we have used three different models respectively :

- **Model 1:** **`Bernoulli Naive Bayes`**.
- **Model 2:** **`SVM (Support Vector Machine)`**.
- **Model 3:** **`Logistic Regression`**.
- **Model 4:** **`Decision Tree`**.
- **Model 5:** **`K-nearest neighbors`**.

The idea behind choosing these models is that **we want to try all the classifiers on the dataset** ranging from simple models to complex models, and try to **find the one that performs the best**.

In[59]:

```
# Model-1 : Bernoulli Naive Bayes.

BNBmodel = BernoulliNB()

start1 = time.time()

BNBmodel.fit(X_train, y_train)

end1 = time.time()

print("\t\t⚠️⚠️⚠️ The training execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end1-start1))

start2 = time.time()
```

```
model_Evaluate(BNBmodel)
```

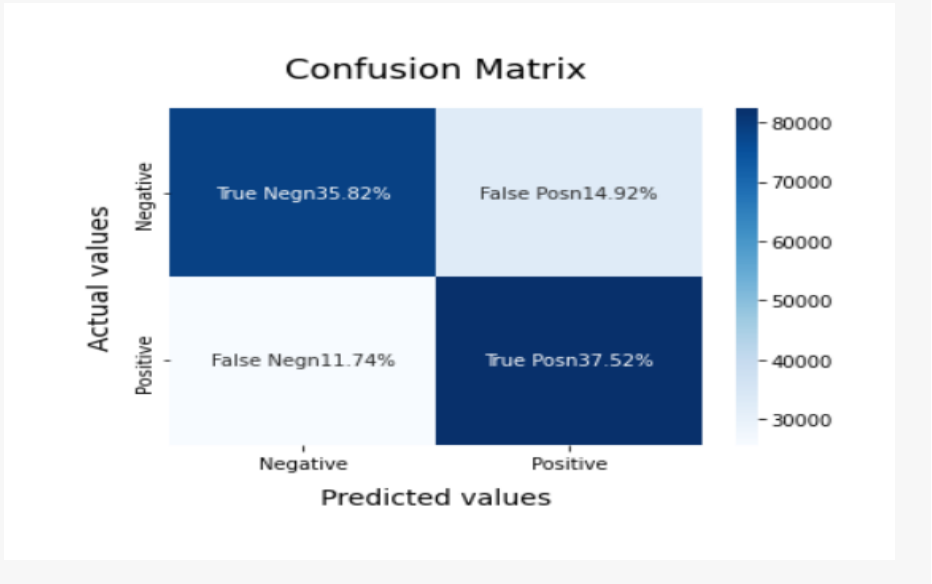
```
y_pred1 = BNBmodel.predict(X_test)
```

```
end2 = time.time()
```

```
print("\t\t⚠️⚠️⚠️ The test execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

Out[59]:

```
⚠️⚠️⚠️ The training execution time of this model is 0.39 seconds
⚠️⚠️⚠️

              precision    recall  f1-score   support

           0       0.75      0.71      0.73    111228

           1       0.72      0.76      0.74    107994

    accuracy                           0.73    219222

   macro avg       0.73      0.73      0.73    219222

weighted avg       0.73      0.73      0.73    219222

⚠️⚠️⚠️ The test execution time of this model is 0.61 seconds ⚠️⚠️⚠️
```

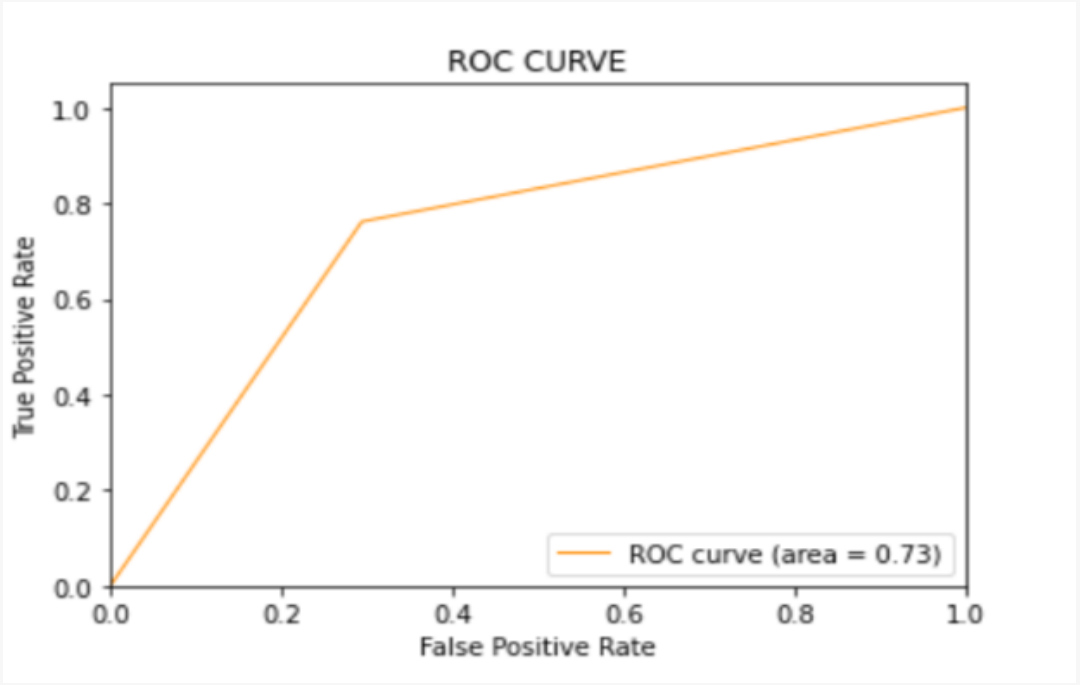


In[60]:

```
# Plot the ROC-AUC Curve for model-1 :

fpr, tpr, thresholds = roc_curve(y_test, y_pred1)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC CURVE')
```

```
plt.legend(loc="lower right")

plt.show()
```

Out[60]:



In[61]:

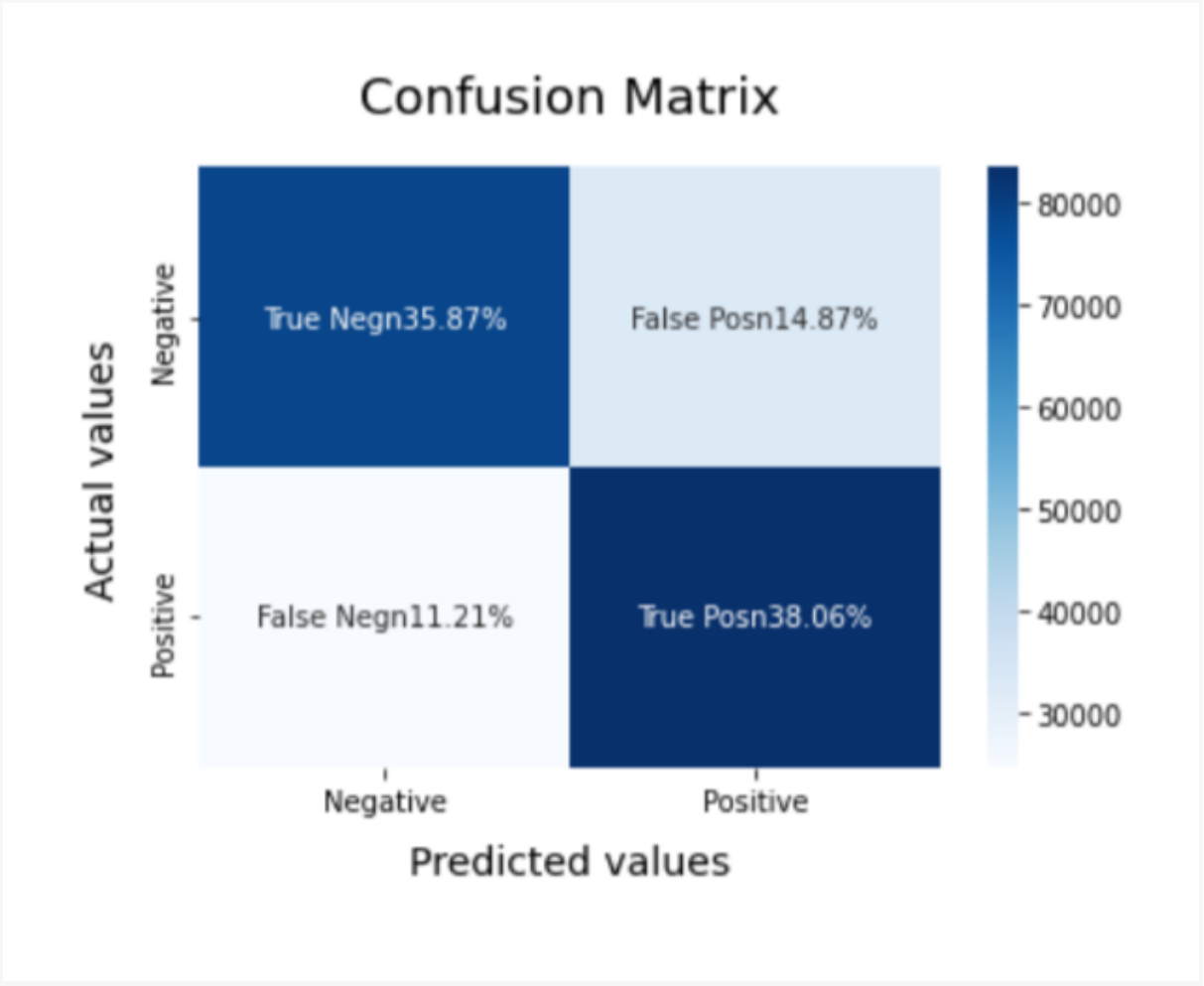
```
# Model-2 : SVM (Support Vector Machine).
SVCmodel = LinearSVC()
start1 = time.time()
SVCmodel.fit(X_train, y_train)
end1 = time.time()
print("\t\t⚠️⚠️⚠️ The training execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end1-start1))
start2 = time.time()
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test)
end2 = time.time()
```

```
print("\t\t⚠️⚠️⚠️ The test execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

Out[61]:

```
⚠️⚠️⚠️ The training execution time of this model is 24.07 seconds
⚠️⚠️⚠️
```

```
              precision    recall  f1-score   support

           0       0.76      0.71      0.73    111228
           1       0.72      0.77      0.74    107994

    accuracy                           0.74    219222
   macro avg       0.74      0.74      0.74    219222
weighted avg       0.74      0.74      0.74    219222

           ⚠️⚠️⚠️ The test execution time of this model is 0.53
seconds ⚠️⚠️⚠️
```

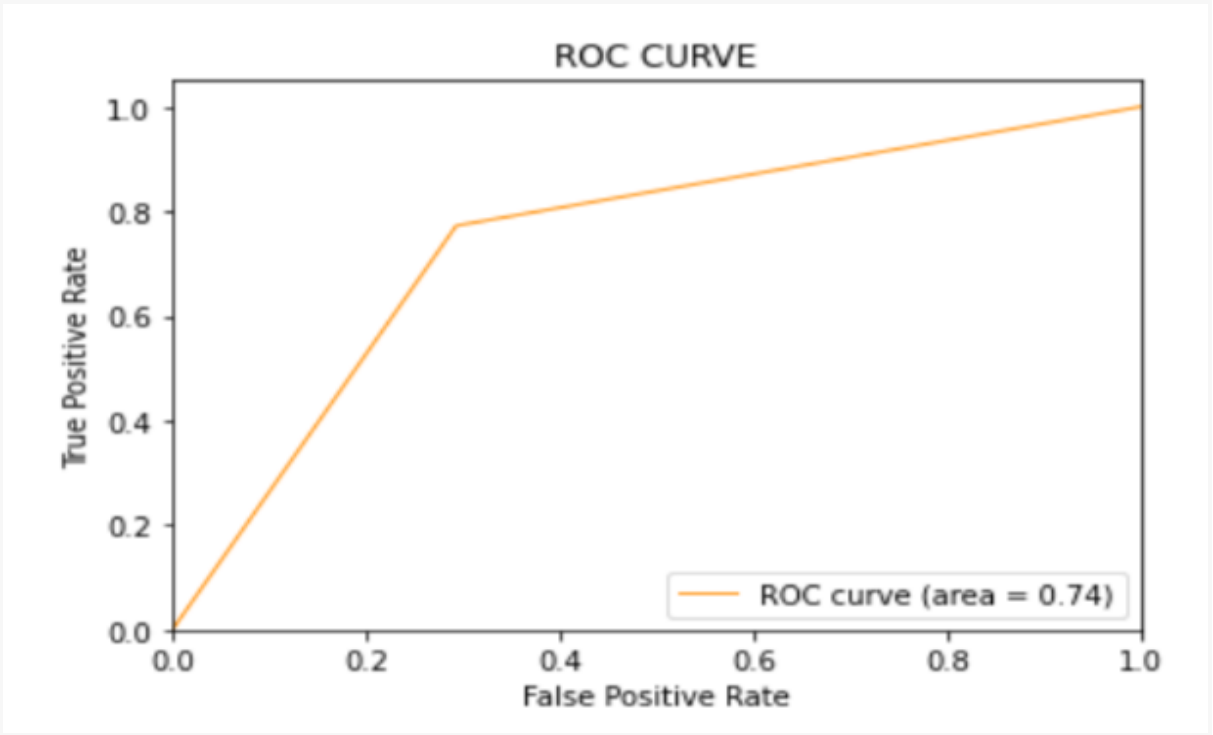


In[62]:

```
# Plot the ROC-AUC Curve for model-2 :
fpr, tpr, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

Out[62]:



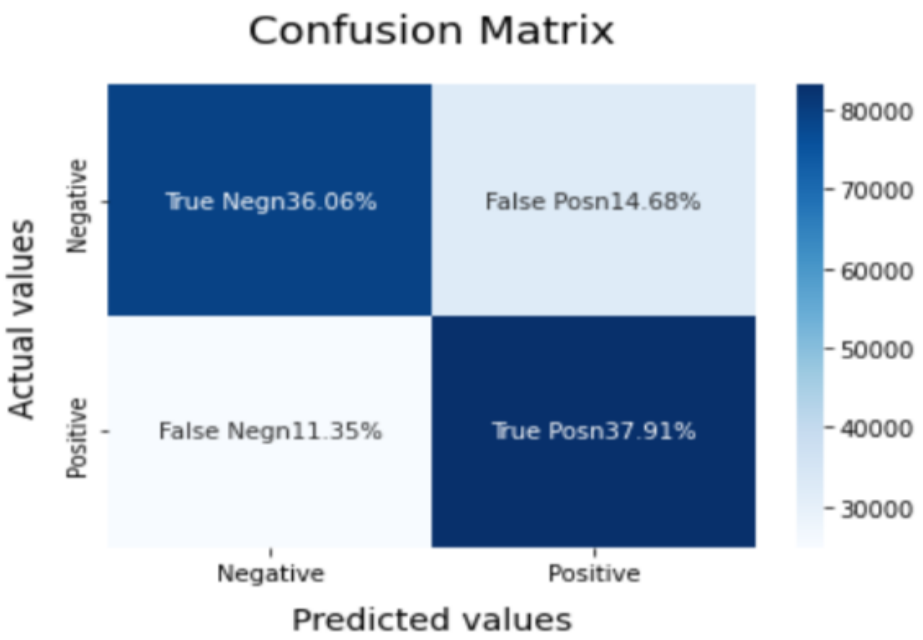
In[63]:

```
# Model-3 : Logistic Regression.
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
start1 = time.time()
LRmodel.fit(X_train, y_train)
end1 = time.time()
print("\t\t⚠️⚠️⚠️ The training execution time of this model is
{:.2f} seconds ⚠️⚠️⚠️\n".format(end1-start1))
start2 = time.time()
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)
end2 = time.time()
```

```
print("\t\t⚠⚠⚠ The test execution time of this model is {:.2f}
seconds ⚠⚠⚠\n".format(end2-start2))
```

Out[63]:

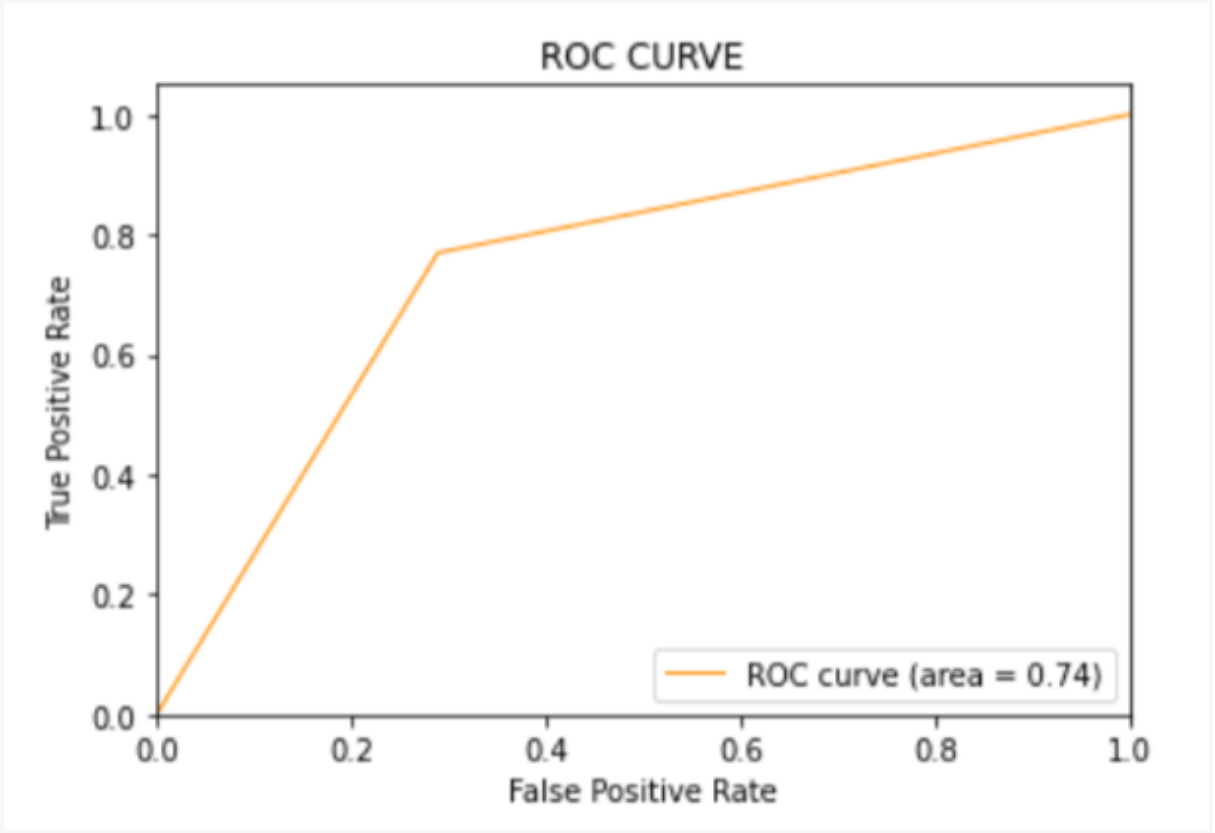
```
⚠⚠⚠ The training execution time of this model is 29.52 seconds
⚠⚠⚠

              precision    recall  f1-score   support

           0       0.76      0.71      0.73    111228
           1       0.72      0.77      0.74    107994

    accuracy                           0.74    219222
   macro avg       0.74      0.74      0.74    219222
weighted avg       0.74      0.74      0.74    219222

           ⚠⚠⚠ The test execution time of this model is 0.54
seconds ⚠⚠⚠
```



In[64]:

```
# Plot the ROC-AUC Curve for model-3 :
fpr, tpr, thresholds = roc_curve(y_test, y_pred3)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

Out[64]:



How do you choose the `value of k ( n_neighbors )` in KNN algorithm ?

In **KNN**, finding the value of k is not easy. A small value of k means that noise will have a higher influence on the result and a large value make it computationally expensive. Data scientists usually choose as an odd number if the number of classes is 2 and another simple approach to select k is set **K=sqrt(n).**

In[65]:

```
int(sqrt(len(df))) # k = sqrt(len(df) = sqrt(n) = sqrt(len(df)
```

Out[65]:

1208

In[66]:

```
# Model-4 : k-nearest neighbors.
knn = KNeighborsClassifier(n_neighbors=int(sqrt(len(df)))) #
sqrt(len(df)) = 1208
start1 = time.time()
knn.fit(X_train, y_train)
end1 = time.time()
print("⚠⚠⚠ The training execution time of this model is {:.2f}
seconds ⚠⚠⚠\n".format(end1-start1))
start2 = time.time()
y_pred4 = knn.predict(X_test_reduced)
print("The accuracy of the model is : " +
str(knn.score(X_test_reduced, y_test_reduced))) # Calculate the
accuracy of the model
end_2 = time.time()
print("⚠⚠⚠ The test execution time of this model is {:.2f}
seconds ⚠⚠⚠\n".format(end2-start2))
```

Out[67]:

```
⚠⚠⚠ The training execution time of this model is 0.12 seconds ⚠⚠⚠

The accuracy of the model is : 0.6163390046982621
⚠⚠⚠ The test execution time of this model is -2.93 seconds ⚠⚠⚠
```

1️⃣1️⃣ Conclusion :

After evaluating all models, we can conclude the following details :

Model Id	Model Name	Accuracy	F1-score (class 0)	F1-score (class 1)	AUC Score	Trainig execution time in seconds	Testing execution time in seconds	Nature of dataset used for training	Nature of dataset used for testing
1	Bernoulli Naive Bayes (BNB)	73%	73%	74%	73%	0.44	0.77	full dataset	full dataset
2	Support Vector Machine (SVM)	74%	73%	75%	74%	34.38	0.63	full dataset	full dataset
3	Logistic Regression (LR)	74%	74%	75%	74%	36.13	0.67	full dataset	full dataset
4	K-nearest neighbors (KNN)	60%	NaN	NaN	61%	0.16	NaN	10% of the dataset	10% of the dataset
5	Decision Tree (DT)	69%	69%	69%	69%	31.84	0.76	10% of the dataset	full dataset

- **`Execution time`** : When it comes to comparing the running time of models, `Bernoulli Naive Bayes` performs faster with a good accuracy score.
- **`Accuracy`** : When it comes to model accuracy, `logistic regression` & `Suppoort Vector Machine` performs better than most of the other models, with an accuracy of **74%**.
- **`F1-score`** : The F1 Scores for **class 0** and **class 1** are :
  - For **class 0** (negative tweets) :

    `accuracy : DT (=0.69) < BNB (=0.73) = SVM (=0.73) < LR (=0.74)`
  - For **class 1** (positive tweets) :

    `accuracy : DT (=0.69) < BNB (=0.74) < SVM (=0.73) = LR (=0.75)`

- **`AUC Score`** :

  `AUC score : KNN (=0.61) < BNB (=0.63) < DT (=0.69) < SVM (=0.74) = LR (=0.74)`

- We therefore conclude that **`logistic regression`** & **`Bernoulli Naive Bayes`** & **`Suppoort Vector Machine`** are the **best model** for the above dataset.

- In our problem statement, **`logistic regression`** follows **`Occam's razor principle`** which defines that for a particular problem statement, if the data has no assumptions, then the simplest model works best. Since our **dataset has no assumptions** and **logistic regression is a simple model**, so the concept holds true for the dataset mentioned above.(although it took much longer to run than the fastest model).
- In conclusion, sentiment analysis using machine learning is a vital tool for marketing, allowing businesses to tap into the voice of their customers and make data-driven decisions. When employed effectively, it can lead to enhanced brand reputation, improved products, and more successful marketing campaigns. To reap the full benefits of sentiment analysis, businesses must continually refine their models, monitor sentiment trends, and adapt to the ever-evolving digital landscape.