School of Engineering and Technology

PRACTICAL FILE

OF

Machine Learning and Pattern Recognition

ENSP202

Submitted to:	**Submitted by:**
Mr. Kunal Rai	**Omm Routray**
Samatrix	**2301730333**
	B.Tech CSE (AI/ML)

Index

Introduction

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

We aim to analyze the sentiment of the tweets provided from the **Sentiment140 dataset** by developing a Machine Learning model involving the use of three classifiers:

1. Logistic Regression – LR
2. Bernoulli Naive Bayes – BNB
3. Support Vector Machine – SVM

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.

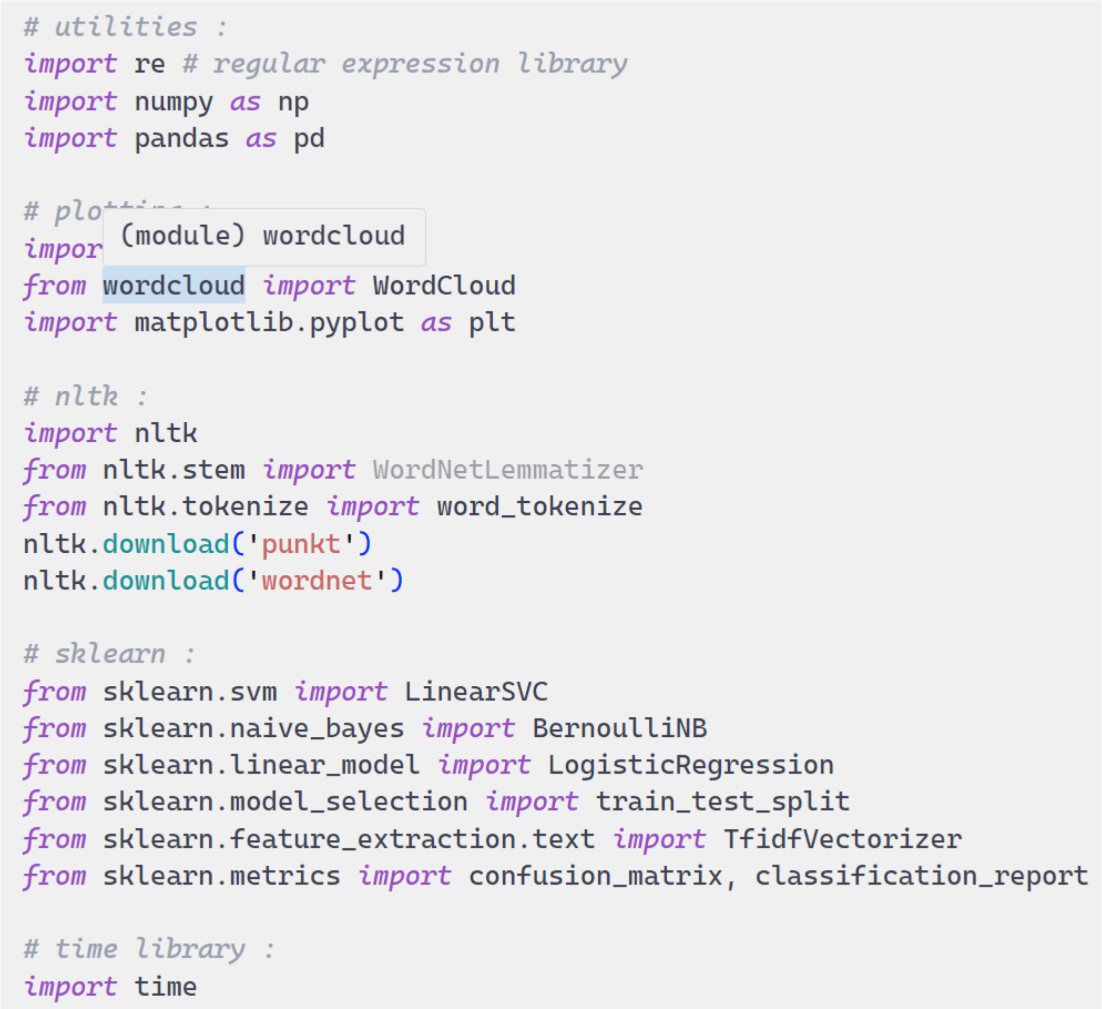
Project Pipeline

The various steps involved in the Machine Learning Pipeline are :

1. Import necessary dependencies.
2. Read and load the dataset.
3. Exploratory data analysis.
4. Data visualization of Target Variables.
5. Data preprocessing.
6. Splitting our data into Train and Test subset.
7. Transforming dataset using TF-IDF Vectorizer.
8. Function for model evaluation.
9. Model building.
10. Conclusion

Importing the necessary dependencies:

Here in this part, we import all the necessary libraries that we will use in our project. The choice of libraries depends on the approach we will follow.



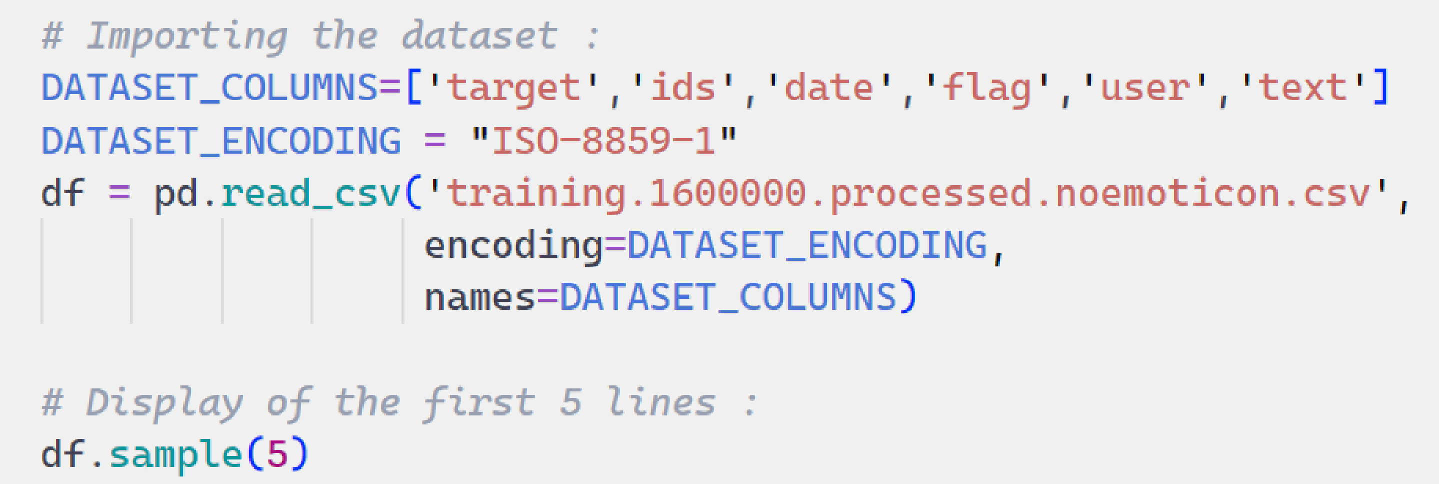
```
# utilities :
import re # regular expression library
import numpy as np
import pandas as pd

# plo
impor
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# nltk :
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('wordnet')

# 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

# time library :
import time
```

Reading and loading the dataset:

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 various columns present in the dataset are:

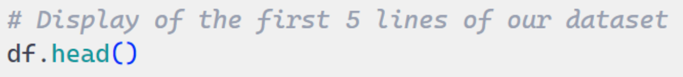
- `target`: the polarity of the tweet (positive or negative)
- `ids`: Unique id of the tweet
- `date`: the date of the tweet
- `flag`: It refers to the query. If no such query exists then it is NO QUERY.
- `user`: It refers to the name of the user that tweeted
- `text`: It refers to the text of the tweet



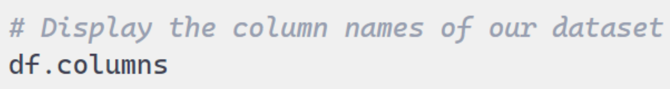
```
# Importing the dataset :
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('training.1600000.processed.noemoticon.csv',
                 encoding=DATASET_ENCODING,
                 names=DATASET_COLUMNS)

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

Exploratory data analysis:

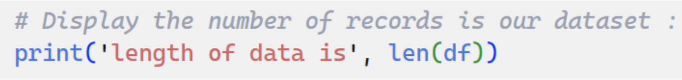
In this part, the objective is to know the imported data as much as possible, we analyze a sample, we look for the shape of the dataset, the column names, the data type information, we check if there are null values, in short, we process our data and above all we target the data (columns) that interests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



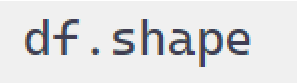
```
# Display of the first 5 lines of our dataset
df.head()
```



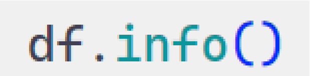
```
# Display the column names of our dataset
df.columns
```



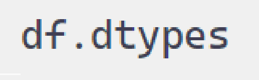
```
# Display the number of records is our dataset :
print('length of data is', len(df))
```



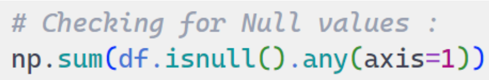
```
df.shape
```



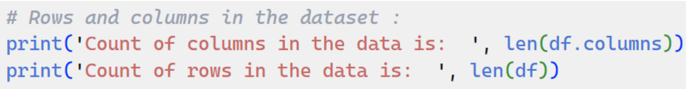
```
df.info()
```



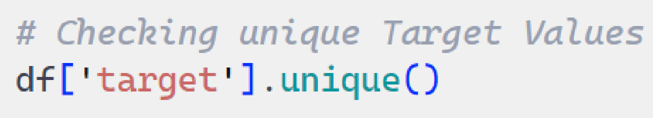
```
df.dtypes
```



```
# Checking for Null values :
np.sum(df.isnull().any(axis=1))
```



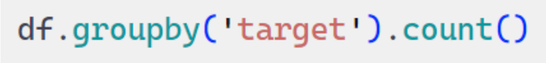
```
# Rows and columns in the dataset :
print('Count of columns in the data is:  ', len(df.columns))
print('Count of rows in the data is:  ', len(df))
```



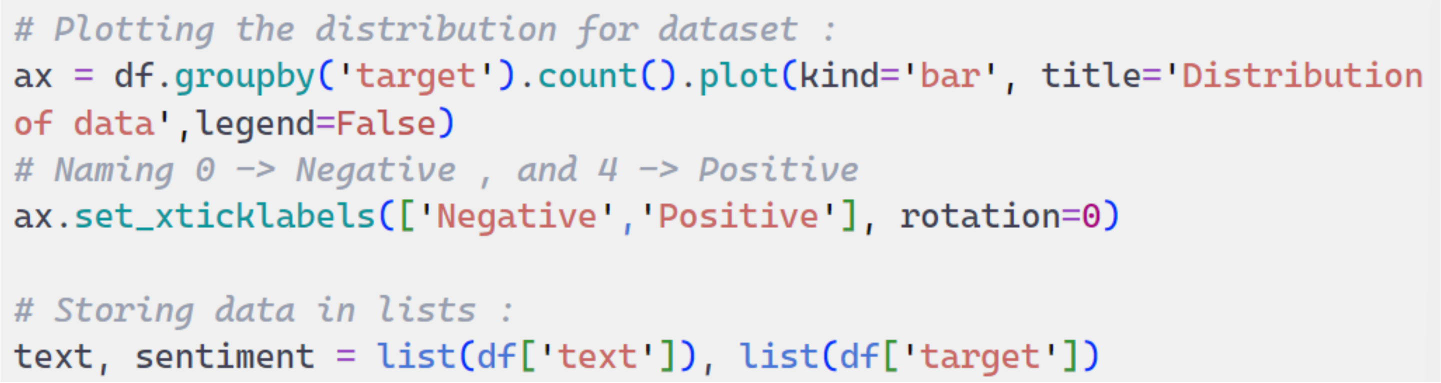
```
# Checking unique Target Values
df['target'].unique()
```

Data visualization of target variables:

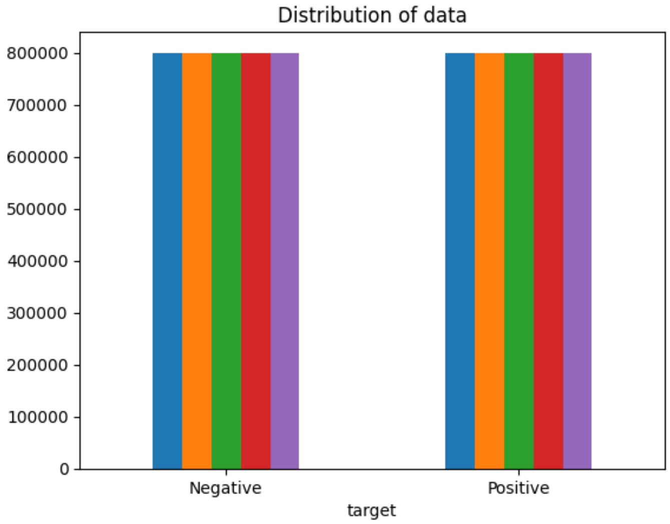
After processing our data and targeting the columns we are interested in, the next step is to have a visual on our data with mathematical plots, the reason for using plots is that a plots makes the data speak more, so it become more understandable.

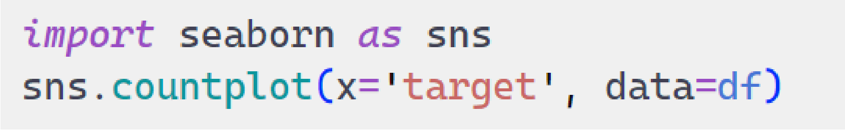


```
df.groupby('target').count()
```

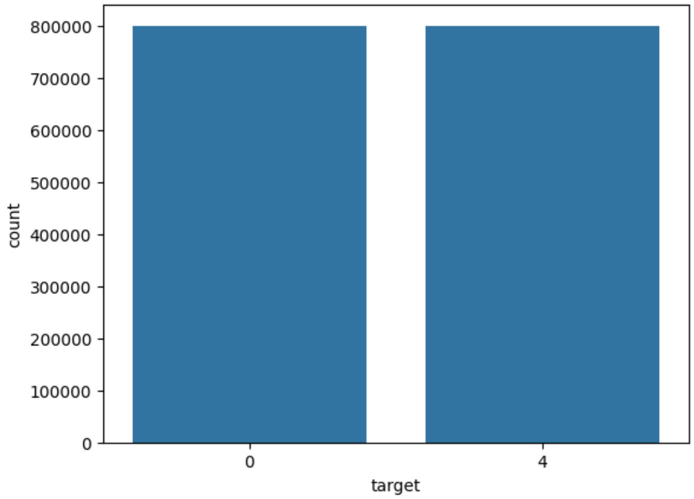


```
# Plotting the distribution for dataset :
ax = df.groupby('target').count().plot(kind='bar', title='Distribution of data',legend=False)
# Naming 0 -> Negative , and 4 -> Positive
ax.set_xticklabels(['Negative','Positive'], rotation=0)

# Storing data in lists :
text, sentiment = list(df['text']), list(df['target'])
```





```
import seaborn as sns
sns.countplot(x='target', data=df)
```



Data preprocessing:

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

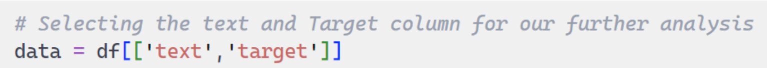
- Removing stop words.
- Removing emojis.
- Converting the text document to lowercase for better generalization.
- Cleaning the ponctuation (to reduce unnecessary noise from the dataset).
- Removing the repeating characters from the words along with removing the URLs as they do not have any significant importance.

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

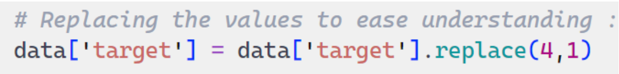
We will then performe

Stemming : reducing the words to their derived stems.

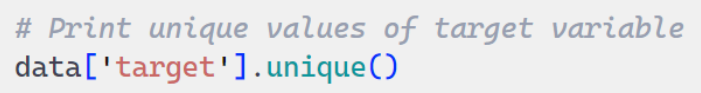
Lemmatization: reducing the derived words to their root form known as lemma for better results.



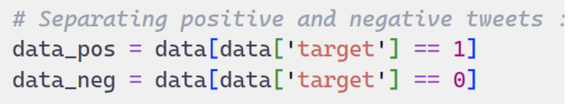
```
# Selecting the text and Target column for our further analysis
data = df[['text','target']]
```



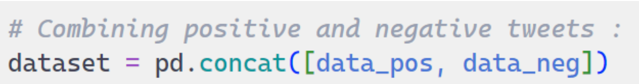
```
# Replacing the values to ease understanding :
data['target'] = data['target'].replace(4,1)
```



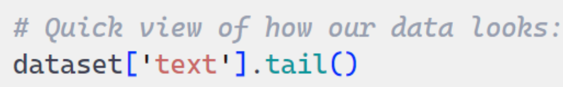
```
# Print unique values of target variable
data['target'].unique()
```



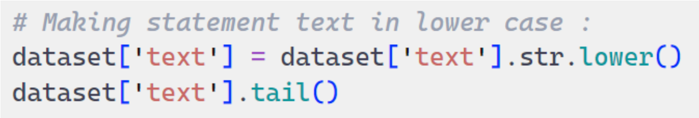
```
# Separating positive and negative tweets :
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
```



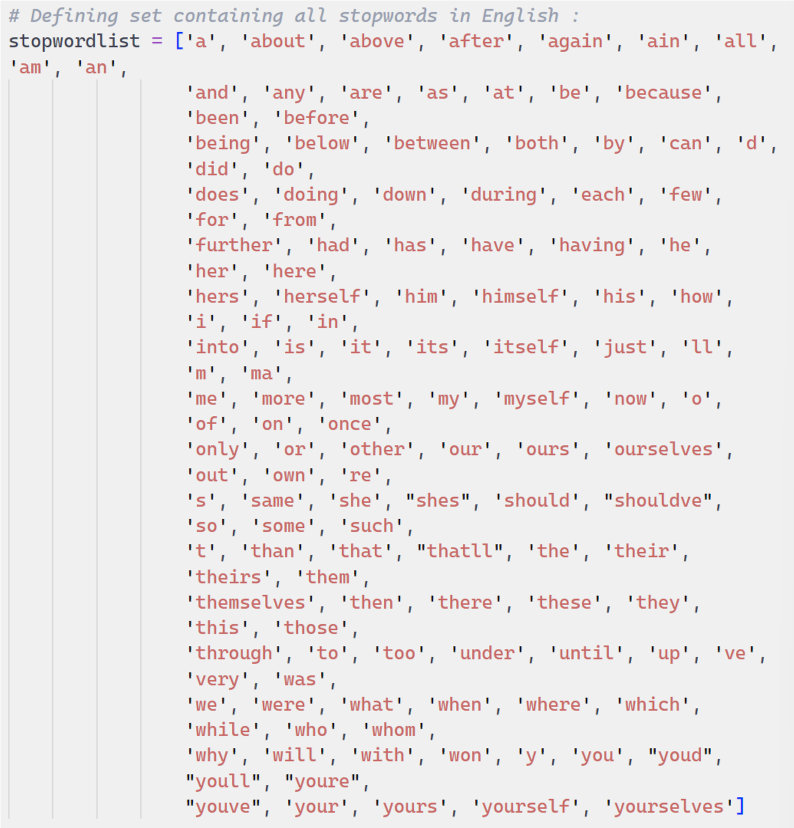
```
# Combining positive and negative tweets :
dataset = pd.concat([data_pos, data_neg])
```



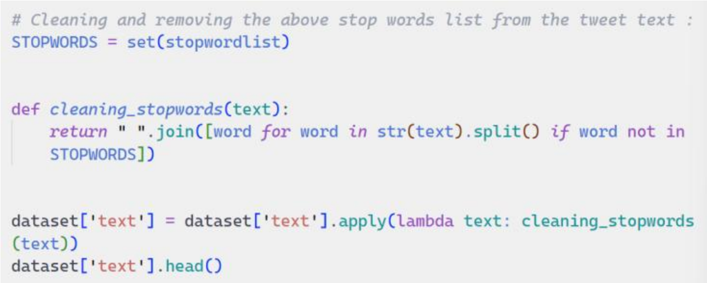
```
# Quick view of how our data looks:
dataset['text'].tail()
```



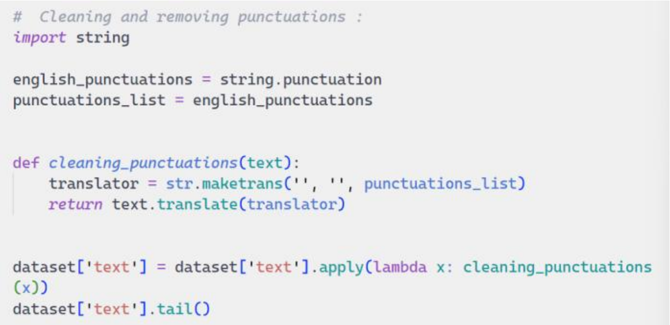
```python
# Making statement text in lower case :
dataset['text'] = dataset['text'].str.lower()
dataset['text'].tail()
```



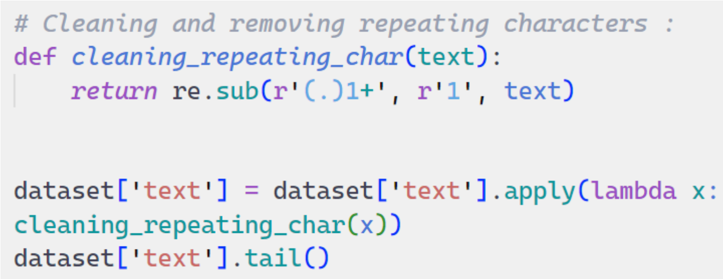
```python
# Defining set containing all stopwords in English :
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all',
'am', 'an',
                    'and', 'any', 'are', 'as', 'at', 'be', 'because',
                    'been', 'before',
                    'being', 'below', 'between', 'both', 'by', 'can', 'd',
                    'did', 'do',
                    'does', 'doing', 'down', 'during', 'each', 'few',
                    'for', 'from',
                    'further', 'had', 'has', 'have', 'having', 'he',
                    'her', 'here',
                    'hers', 'herself', 'him', 'himself', 'his', 'how',
                    'i', 'if', 'in',
                    'into', 'is', 'it', 'its', 'itself', 'just', 'll',
                    'm', 'ma',
                    'me', 'more', 'most', 'my', 'myself', 'now', 'o',
                    'of', 'on', 'once',
                    'only', 'or', 'other', 'our', 'ours', 'ourselves',
                    'out', 'own', 're',
                    's', 'same', 'she', "shes", 'should', "shouldve",
                    'so', 'some', 'such',
                    't', 'than', 'that', "thatll", 'the', 'their',
                    'theirs', 'them',
                    'themselves', 'then', 'there', 'these', 'they',
                    'this', 'those',
                    'through', 'to', 'too', 'under', 'until', 'up', 've',
                    'very', 'was',
                    'we', 'were', 'what', 'when', 'where', 'which',
                    'while', 'who', 'whom',
                    'why', 'will', 'with', 'won', 'y', 'you', "youd",
                    "youll", "youre",
                    "youve", 'your', 'yours', 'yourself', 'yourselves']
```



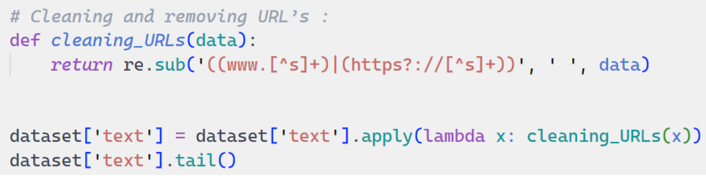
```python
# Cleaning and removing the above stop words list from the tweet text :
STOPWORDS = set(stopwordlist)


def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in
    STOPWORDS])


dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords
(text))
dataset['text'].head()
```



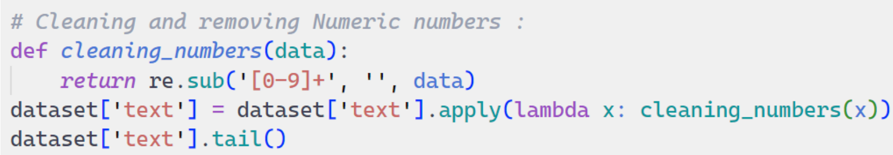
```python
#  Cleaning and removing punctuations :
import string

english_punctuations = string.punctuation
punctuations_list = english_punctuations


def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)


dataset['text'] = dataset['text'].apply(lambda x: cleaning_punctuations
(x))
dataset['text'].tail()
```



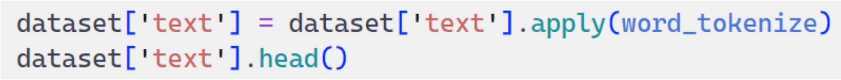
```python
# Cleaning and removing repeating characters :
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)


dataset['text'] = dataset['text'].apply(lambda x:
cleaning_repeating_char(x))
dataset['text'].tail()
```



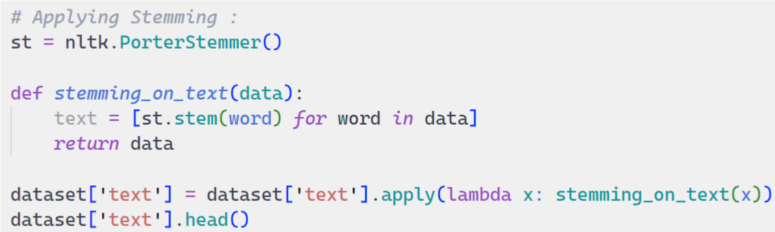
```python
# Cleaning and removing URL's :
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))', ' ', data)


dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()
```



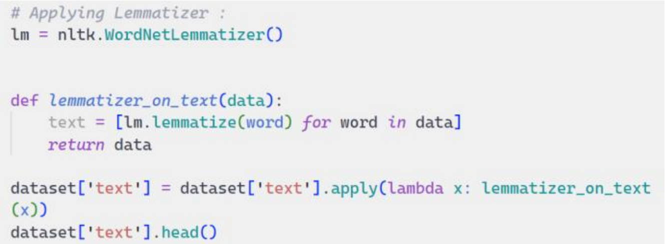
```python
# Cleaning and removing Numeric numbers :
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
```



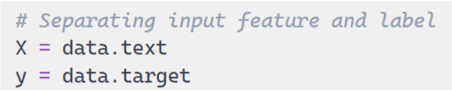
```python
dataset['text'] = dataset['text'].apply(word_tokenize)
dataset['text'].head()
```



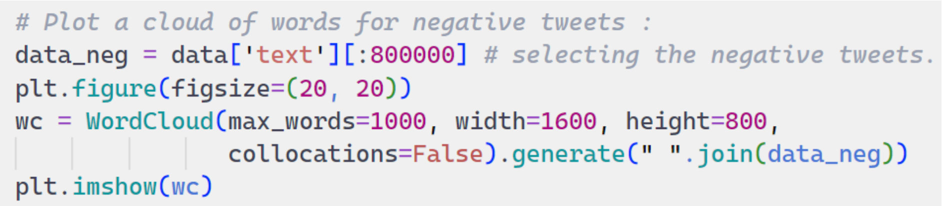
```
# Applying Stemming :
st = nltk.PorterStemmer()

def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data

dataset['text'] = dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'].head()
```



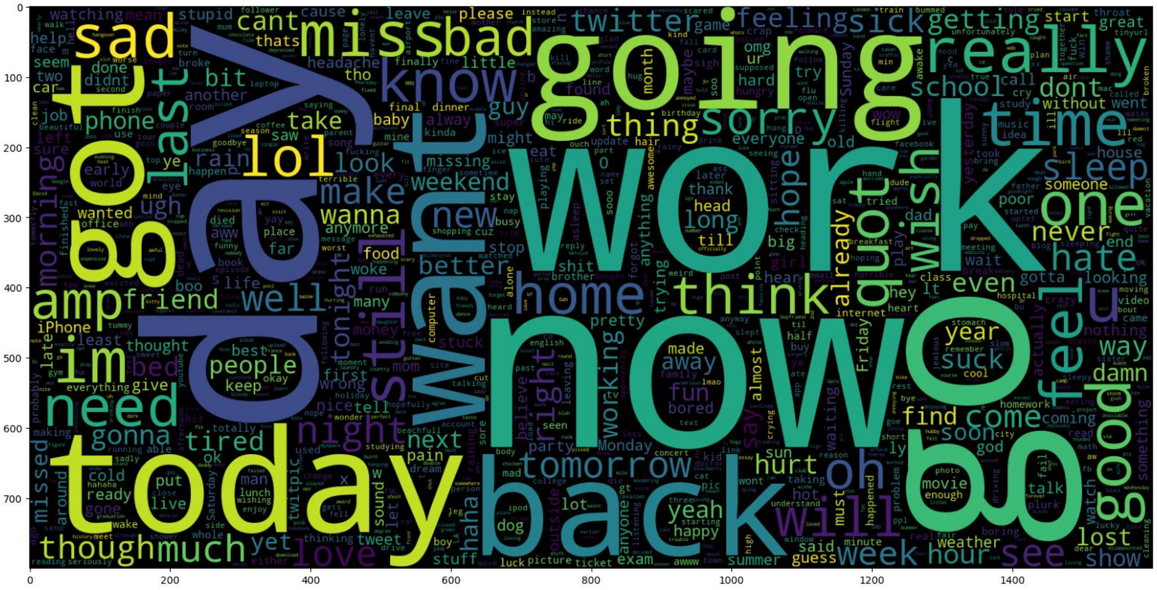
```
# Applying Lemmatizer :
lm = nltk.WordNetLemmatizer()


def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data

dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text
(x))
dataset['text'].head()
```

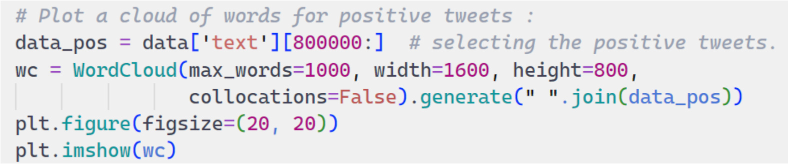


```
# Separating input feature and label
X = data.text
y = data.target
```

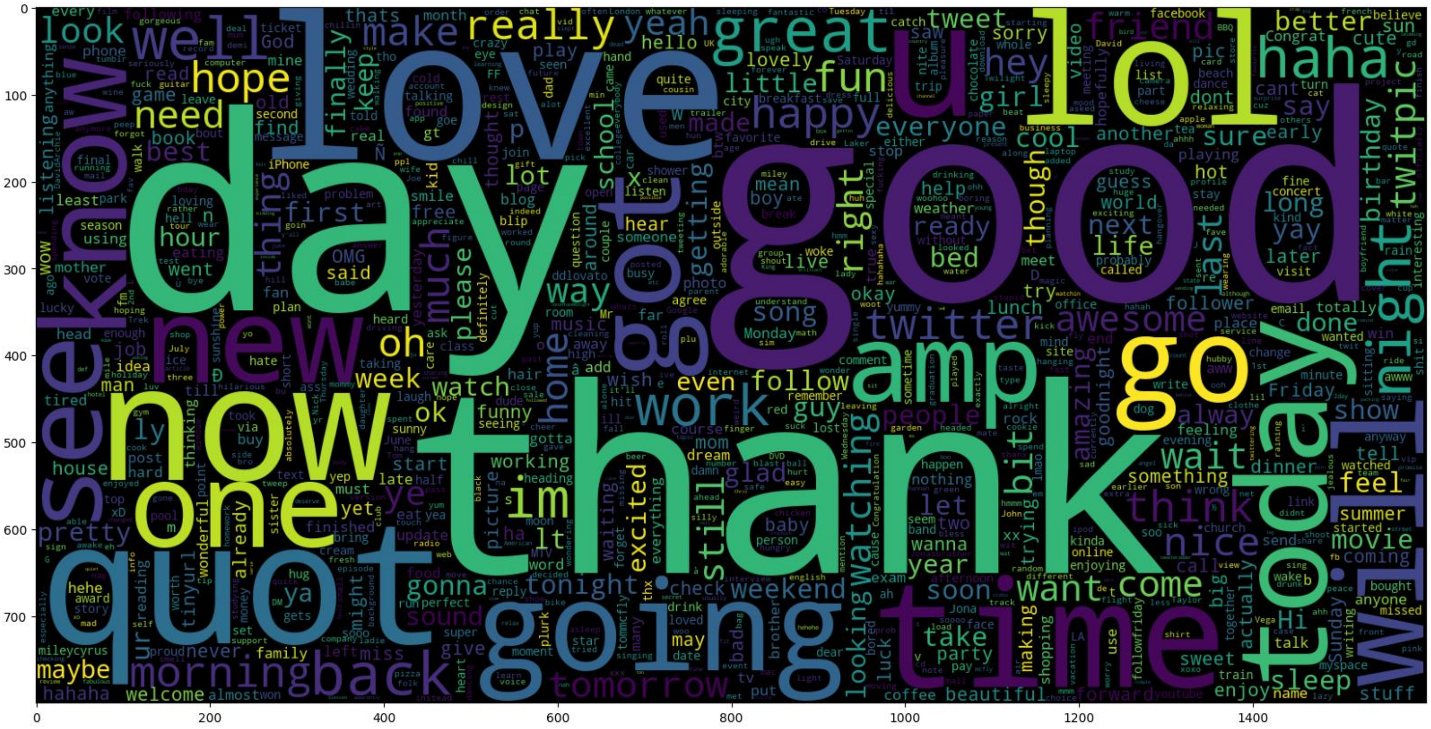


```
# Plot a cloud of words for negative tweets :
data_neg = data['text'][:800000] # selecting the negative tweets.
plt.figure(figsize=(20, 20))
wc = WordCloud(max_words=1000, width=1600, height=800,
               collocations=False).generate(" ".join(data_neg))
plt.imshow(wc)
```

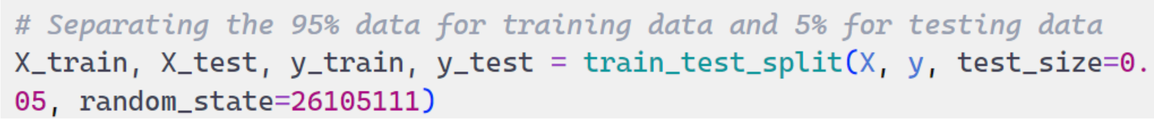




```
# Plot a cloud of words for positive tweets :
data_pos = data['text'][800000:]  # selecting the positive tweets.
wc = WordCloud(max_words=1000, width=1600, height=800,
               collocations=False).generate(" ".join(data_pos))
plt.figure(figsize=(20, 20))
plt.imshow(wc)
```



Splitting our data into Train and Test subset:



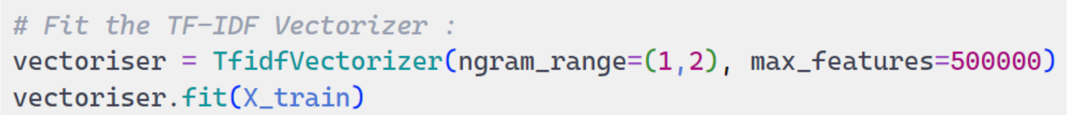
```
# Separating the 95% data for training data and 5% for testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
05, random_state=26105111)
```

- **`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.

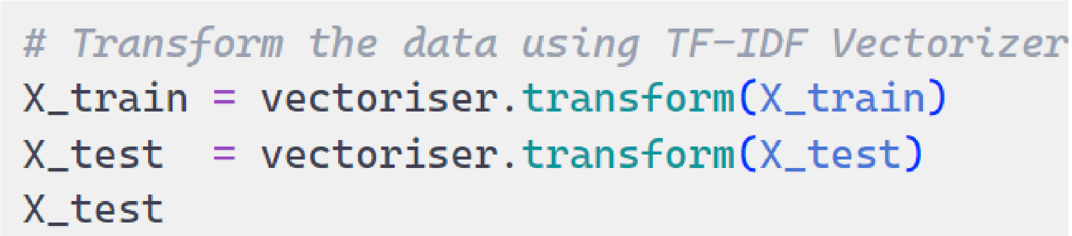
- X contains data.text
- y contains = data.target
- X_train contains 95% of data.text
- X_test contains 5% of data.text
- y_train contains 95% of data.target
- y_test contains 5% of data.target

Transforming dataset using TF-IDF Vectorizer:

Scikit-learn's Tfidftransformer and Tfidfvectorizer aim to do the same thing, which is to convert a collection of raw documents to a matrix of TF-IDF features.



```
# Fit the TF-IDF Vectorizer :
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
```



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

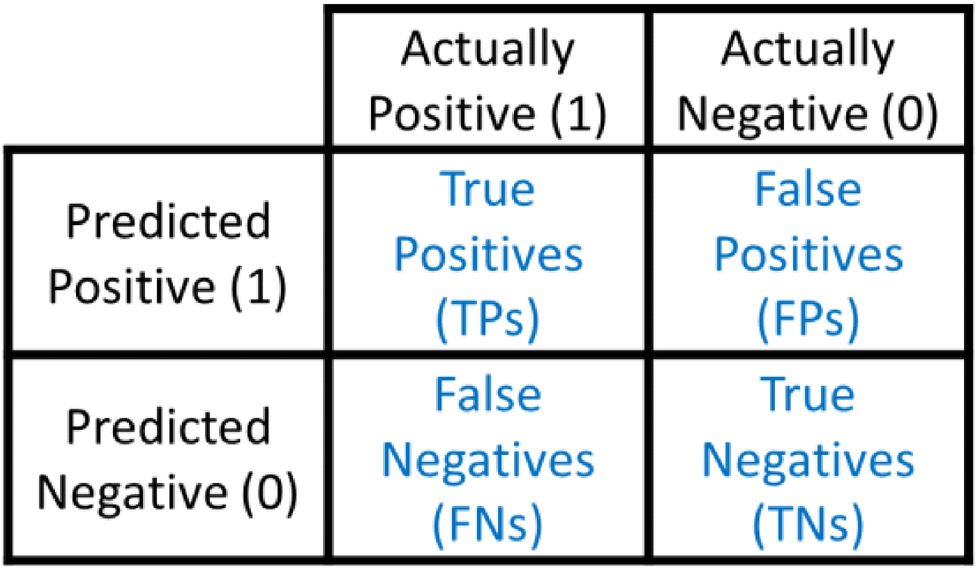
Function for model evaluation:

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 :

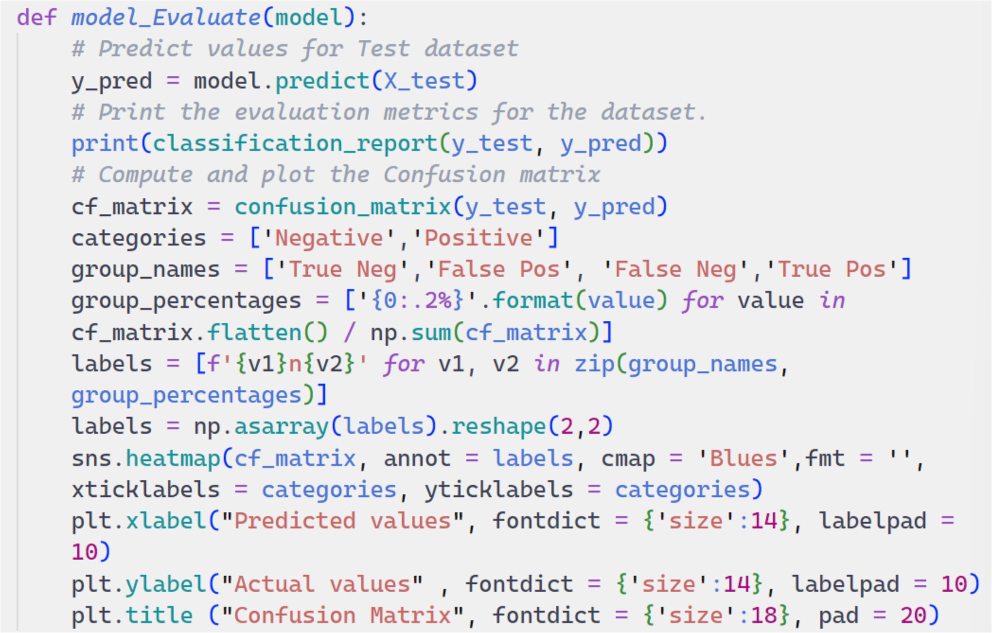
Accuracy Score: Typically, the accuracy of a predictive model is good (above 90% accuracy)

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.



	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)



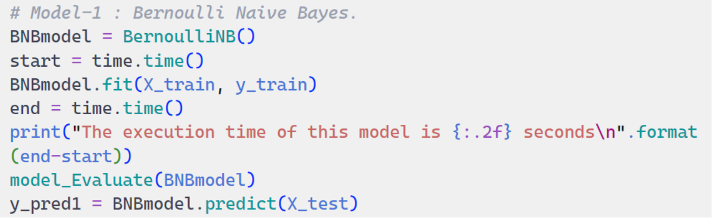
```
def model_Evaluate(model):
    # Predict values for Test dataset
    y_pred = model.predict(X_test)
    # Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))
    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)
    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)
```

Model building:

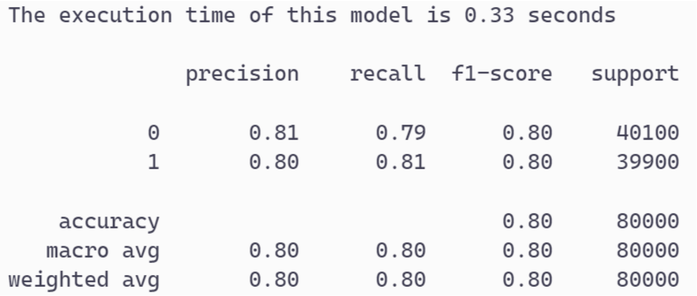
In the problem statement we have used three different models respectively :

- Bernoulli Naive Bayes
- SVM (Support Vector Machine)
- Logistic Regression

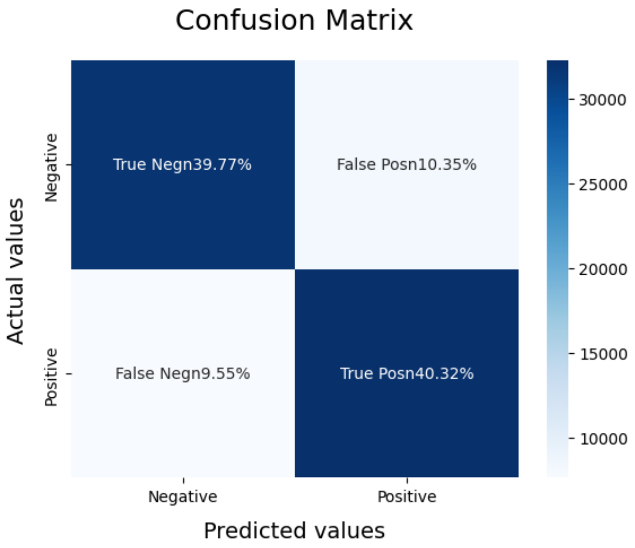
The idea behind choosing these models is that we want to try all the classifiers on the dataset ranging from simple models to complex models, and try to find the one that performs the best.

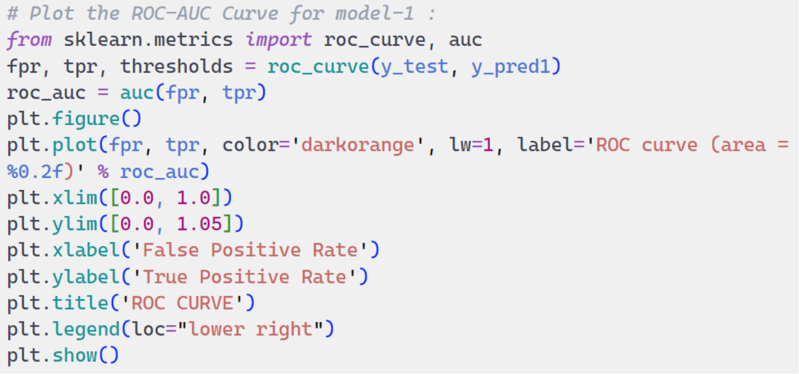


```
# Model-1 : Bernoulli Naive Bayes.
BNBmodel = BernoulliNB()
start = time.time()
BNBmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format
(end-start))
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test)
```

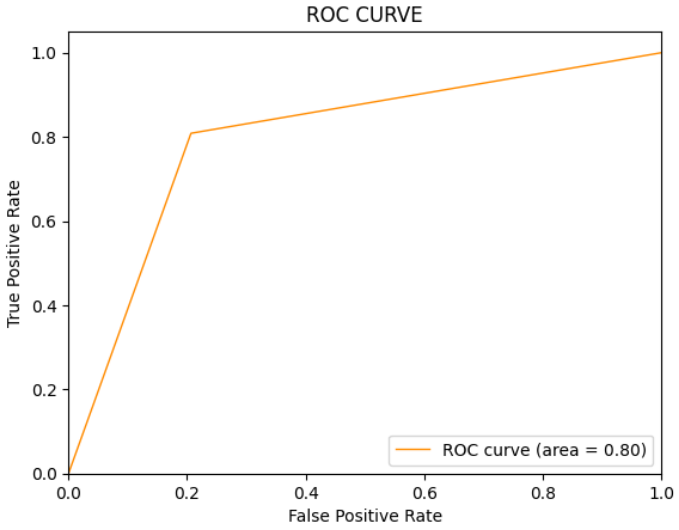


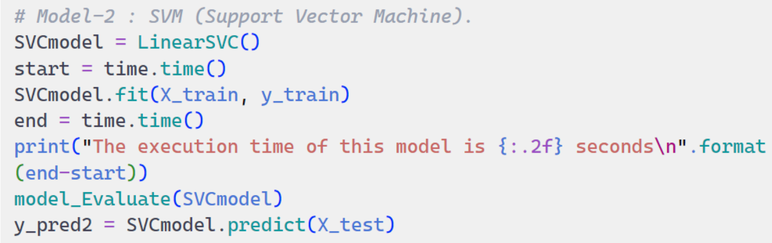
```
The execution time of this model is 0.33 seconds

              precision    recall  f1-score   support

           0       0.81      0.79      0.80     40100
           1       0.80      0.81      0.80     39900

    accuracy                           0.80     80000
   macro avg       0.80      0.80      0.80     80000
weighted avg       0.80      0.80      0.80     80000
```



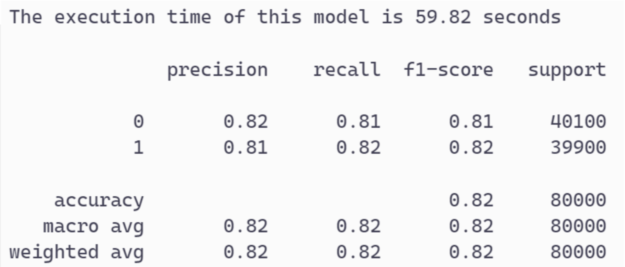


```
# Plot the ROC-AUC Curve for model-1 :
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred1)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

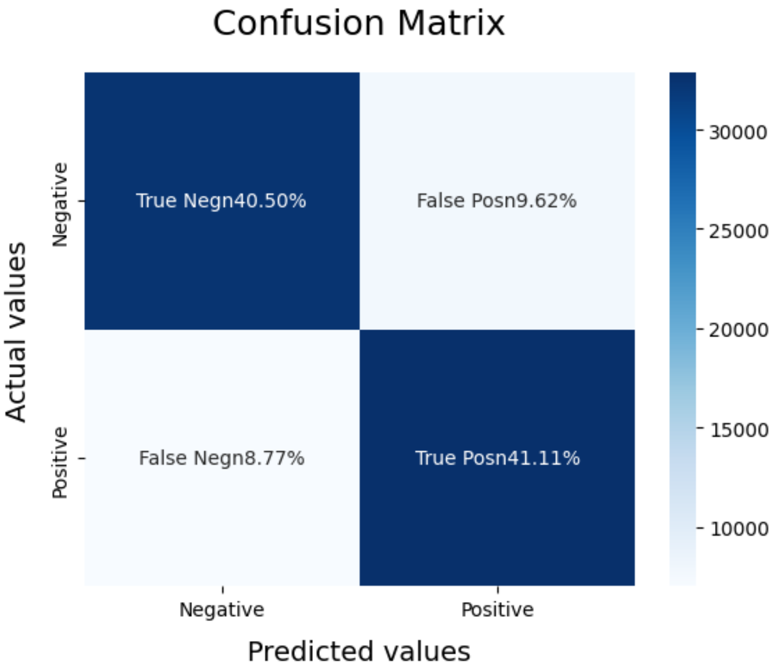


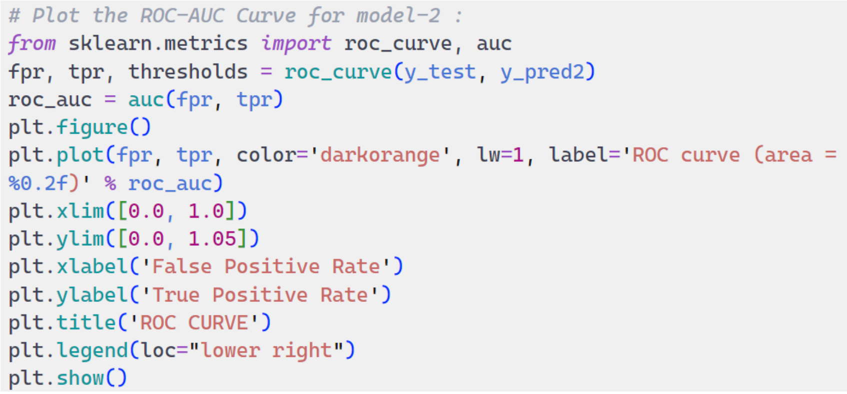


```
# Model-2 : SVM (Support Vector Machine).
SVCmodel = LinearSVC()
start = time.time()
SVCmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format
(end-start))
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test)
```

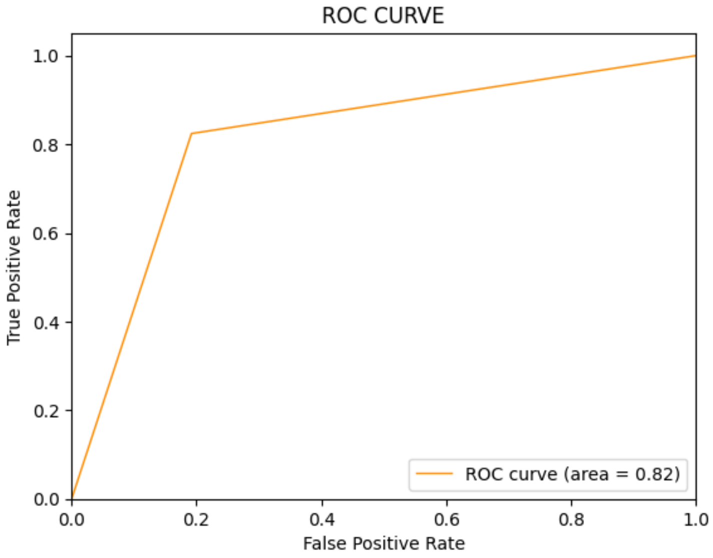


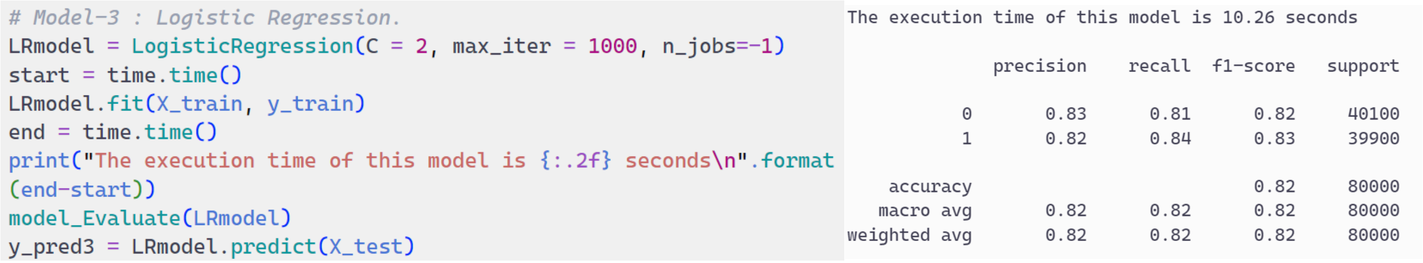
```
The execution time of this model is 59.82 seconds

              precision    recall  f1-score   support

           0       0.82      0.81      0.81     40100
           1       0.81      0.82      0.82     39900

    accuracy                           0.82     80000
   macro avg       0.82      0.82      0.82     80000
weighted avg       0.82      0.82      0.82     80000
```





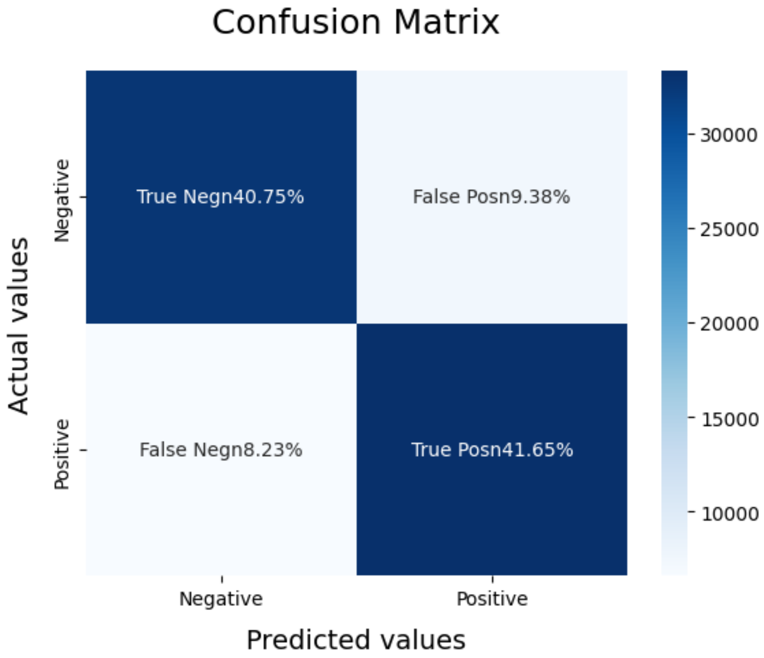
```
# Plot the ROC-AUC Curve for model-2 :
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

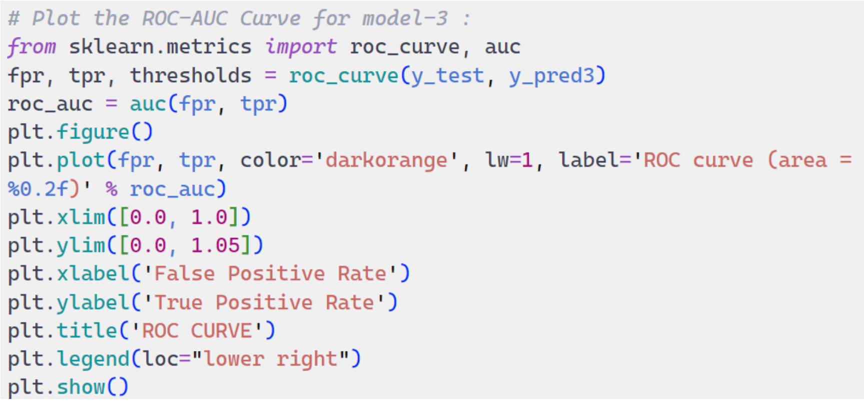




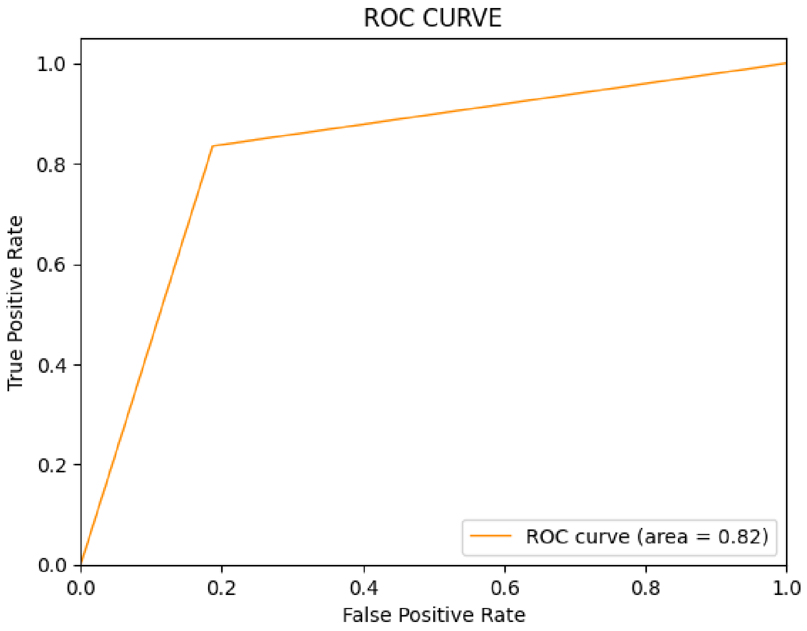
```
# Model-3 : Logistic Regression.
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
start = time.time()
LRmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format
(end-start))
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)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

```
The execution time of this model is 10.26 seconds

              precision    recall  f1-score   support

           0       0.83      0.81      0.82     40100
           1       0.82      0.84      0.83     39900

    accuracy                           0.82     80000
   macro avg       0.82      0.82      0.82     80000
weighted avg       0.82      0.82      0.82     80000
```

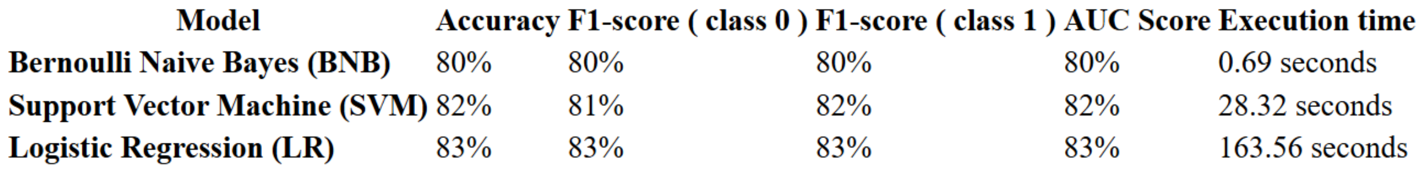




```
# Plot the ROC-AUC Curve for model-3 :
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred3)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])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



Conclusion



Model	Accuracy	F1-score (class 0)	F1-score (class 1)	AUC Score	Execution time
Bernoulli Naive Bayes (BNB)	80%	80%	80%	80%	0.69 seconds
Support Vector Machine (SVM)	82%	81%	82%	82%	28.32 seconds
Logistic Regression (LR)	83%	83%	83%	83%	163.56 seconds

Execution time: When it comes to comparing the running time of models, `Bernoulli Naive Bayes` performs faster than `SVM`, which in turn runs faster than `Logistic Regression`.

Accuracy: When it comes to model accuracy, `logistic regression` performs better than `SVM`, which in turn performs better than `Bernoulli Naive Bayes`.

F1-score: The F1 Scores for class 0 and class 1 are :

For class 0 (negative tweets) :

- accuracy : BNB (= 0.80) < SVM (=0.81) < LR (= 0.83)

For class 1 (positive tweets) :

- accuracy : BNB (= 0.80) < SVM (=0.82) < LR (= 0.83)

AUC Score: All three models have the same ROC-AUC score.

- AUC score : BNB (= 0.80) < SVM (=0.82) < LR (= 0.83)

We therefore conclude that logistic regression is the best model for the above dataset.(although it took much longer to run than other models).

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.