PRML Lab - 5

1. Given the tweets dataset, we have to perform the required operations on the data

   a. Data Preparation:

   i. Load the data :

```
data = pd.read_csv('/content/train.csv')
```

The csv file can be read in data dataframe using the above code .
The dataframe consists of 7613 data samples and 4 features which are Id , keyword , location and text respectively . Each data sample is classified into two categories i.e 1 or 0

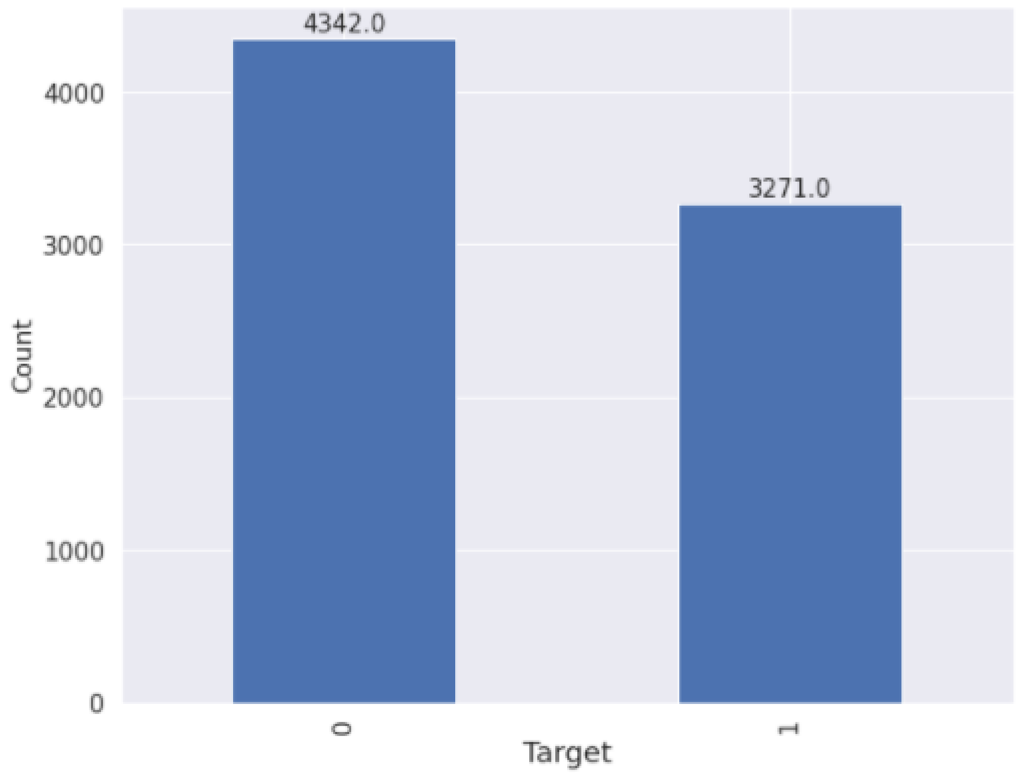
   ii Plot the count for each target:

```
data["target"].value_counts()
```

The above code will give counts of samples classified as 0 and 1 respectively .

We can use a bar plot to plot the information

```
splot = data['target'].value_counts().plot.bar()
```



The plot shows that 4342 samples are classified as 0 and 3271 samples are classified as 1

iii Print the unique keywords :

```
unique_keywords = data.keyword.unique()
```

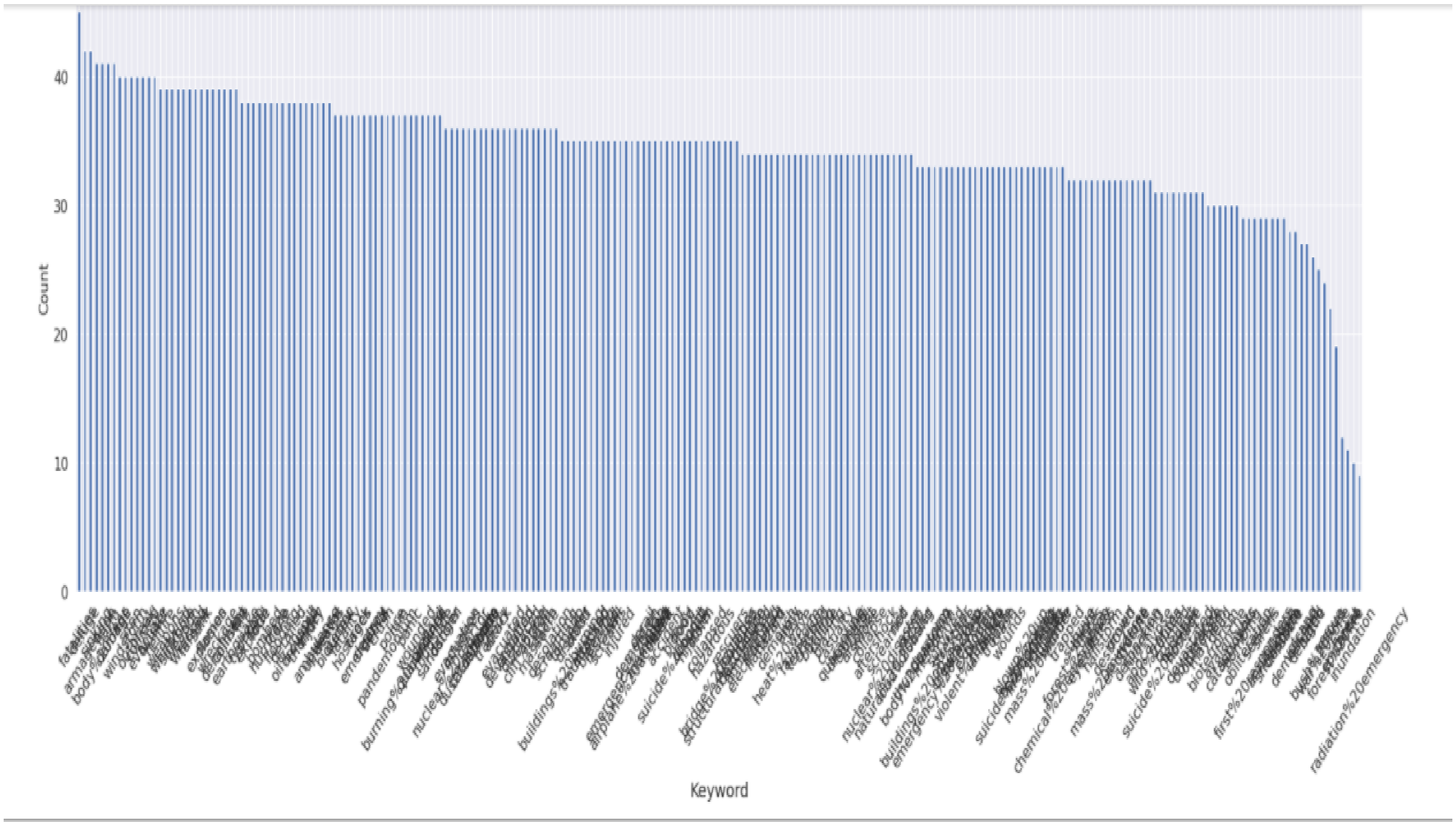
The above code snippet will store unique keywords from keyword column in unique_keywords (a numpy array)

Then we simply used print statement to print the unique keywords

iv. Plot the count of each keyword:

```
pd.value_counts(data['keyword']).plot.bar()
```

The above snippet is used to plot a bar plot depiction count of unique keywords

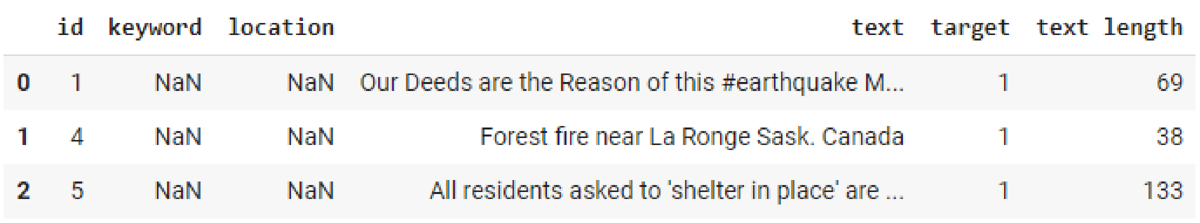


As the number of unique keywords are 222 , the bar plot seems messy.

v. Visualize the correlation of the length of a tweet with its target:

To visualise the correlation of length of a tweet with its target, we first added a column “text length” containing length of text for each data sample .

```
data["text length"]= data["text"].str.len()
```



	id	keyword	location	text	target	text length
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	69
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	38
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	133

Dataframe looks something like this .

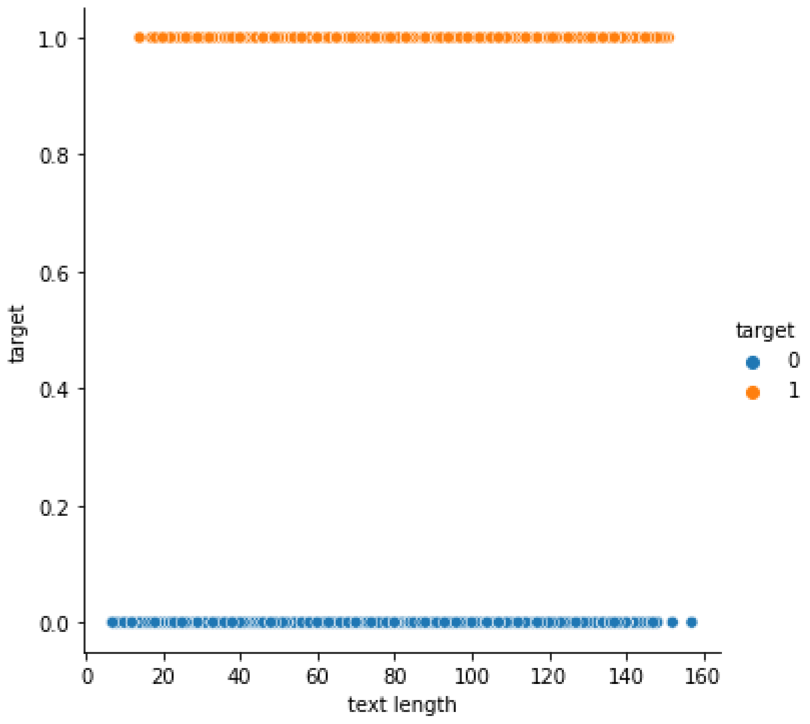
We found the correlation using corr function,

```
data['text length'].corr(data['target'])
```

Which gives output as `0.18181684254460748` which shows that there is
Little to no correlation between length of a tweet with its target.

```
sns.relplot(x=data["text length"], y="target",
hue="target", data=data)
```

Plotting can be done using above code



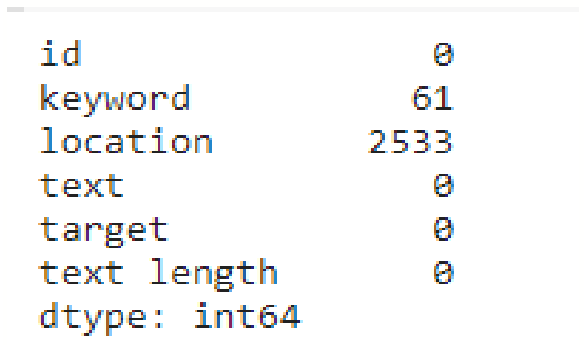
From the graph also we can see that for most of the text lengths , the target can be both 1 and 0 i.e there is no correlation . However , a little bit part before text length = 20 is classified as 0 .

vi. Print the null values in a column :

```
data.isna().sum()
```

This will give number of NaN values for each column

The output of the snippet looks like this



```
id                0
keyword          61
location       2533
text              0
target            0
text length       0
dtype: int64
```

vii. Removing null values :

```
data = data.dropna(axis = 0)
```

The above code drops every row consisting null values . The number of rows remaining after dropping is `5080`

viii. Removing Double Spaces, Hyphens and arrows, Emojis, URL, another Non-English or special symbol :

Firstly the text is lowercased using

```
data["text_lower"] = data["text"].str.lower()
```

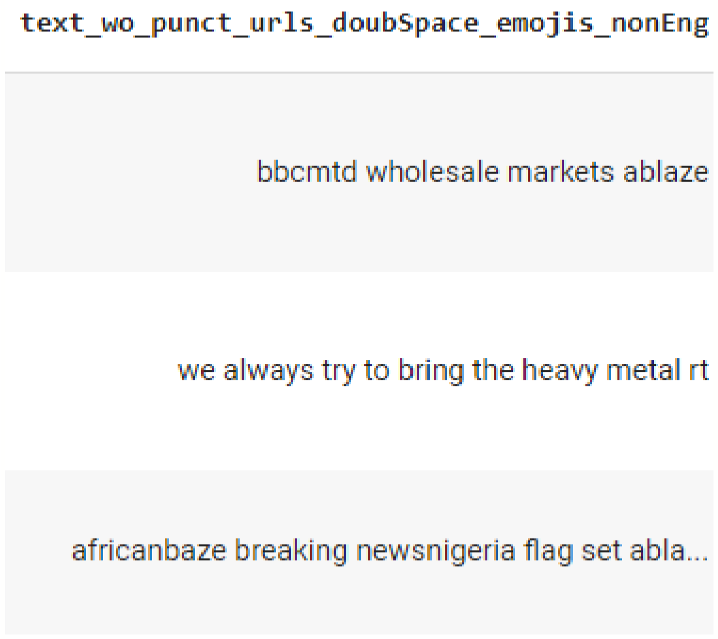
The code creates a new column ‘text_lower’ with all lowercase text

The functions used to remove all the redundant things can be seen in the notebook . The code uses Regular expression i.e re. During each cleaning step , new column is created as follows .

```
data["text_wo_urls"]=data["text_lower"].apply(lambda
text: remove_urls(text))

data["text_wo_punct_urls"]=data["text_wo_urls"].apply(
lambda text: remove_punctuation(text))

data["text_wo_punct_urls_doubSpace"]=data["text_wo_pun
ct_urls"].apply(lambda text:
remove_double_space(text))

data["text_wo_punct_urls_doubSpace_emojis"]=data["text
_wo_punct_urls_doubSpace"].apply(lambda text:
remove_emoji(text))
```

```
data["text_wo_punct_urls_doubSpace_emojis_nonEng"]=dat
a["text_wo_punct_urls_doubSpace_emojis"].apply(lambda
text: remove_non_english(text))
```

The final cleaned data is stored in column named `"text_wo_punct_urls_doubSpace_emojis_nonEng"`



text_wo_punct_urls_doubSpace_emojis_nonEng
bbcmtd wholesale markets ablaze
we always try to bring the heavy metal rt
africanbaze breaking newsnigeria flag set abla...

ix. Replace wrong spellings with correct ones :

Using `pyspellchecker library` , wrong spellings are replaced with corrected ones.

```
data["text_wo_punct_urls_doubSpace_emojis_nonEng"]=
data["text_wo_punct_urls_doubSpace_emojis_nonEng"].app
ly(lambda text: correct_spellings(text))
```

x. Plot a word cloud of the real and fake target :

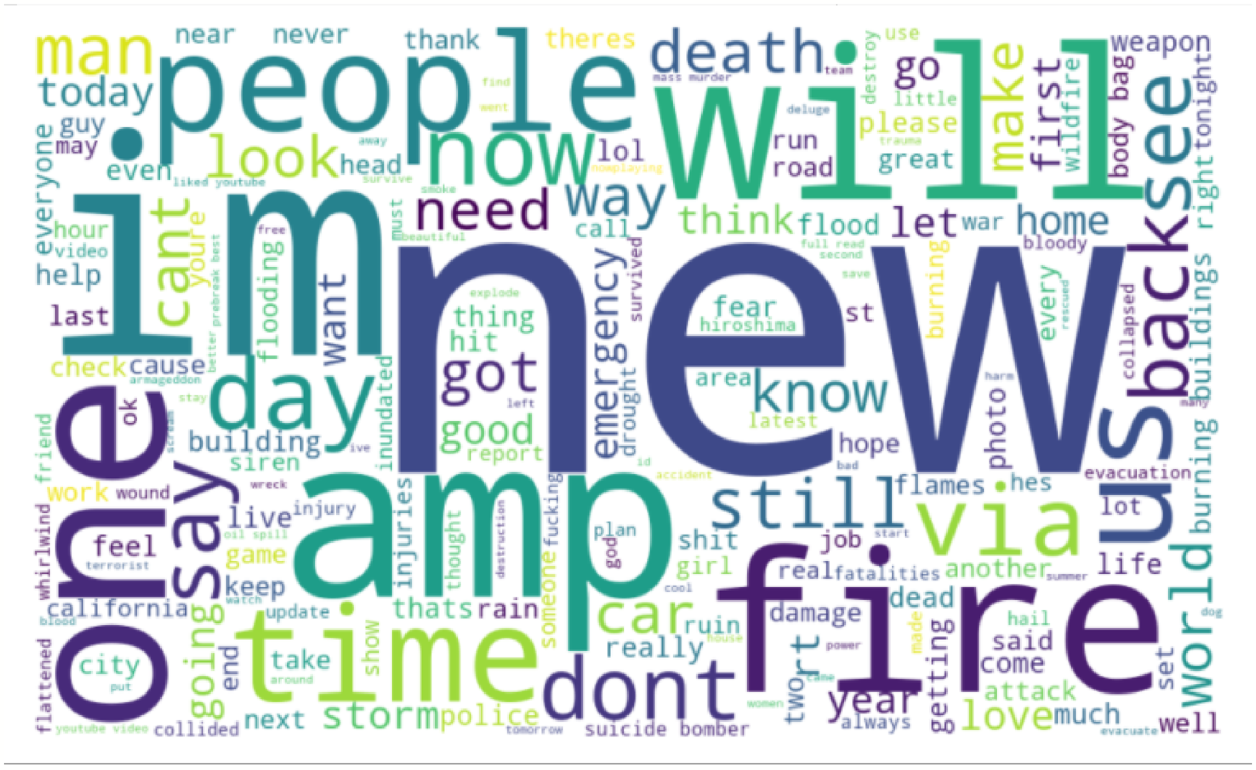
To generate the word cloud , we need to read the data in ‘text’ which can be done using .

```
text = " ".join(word for word in
data.text_wo_punct_urls_doubSpace_emojis_nonEng)
```

After that wordcloud can be formed using :

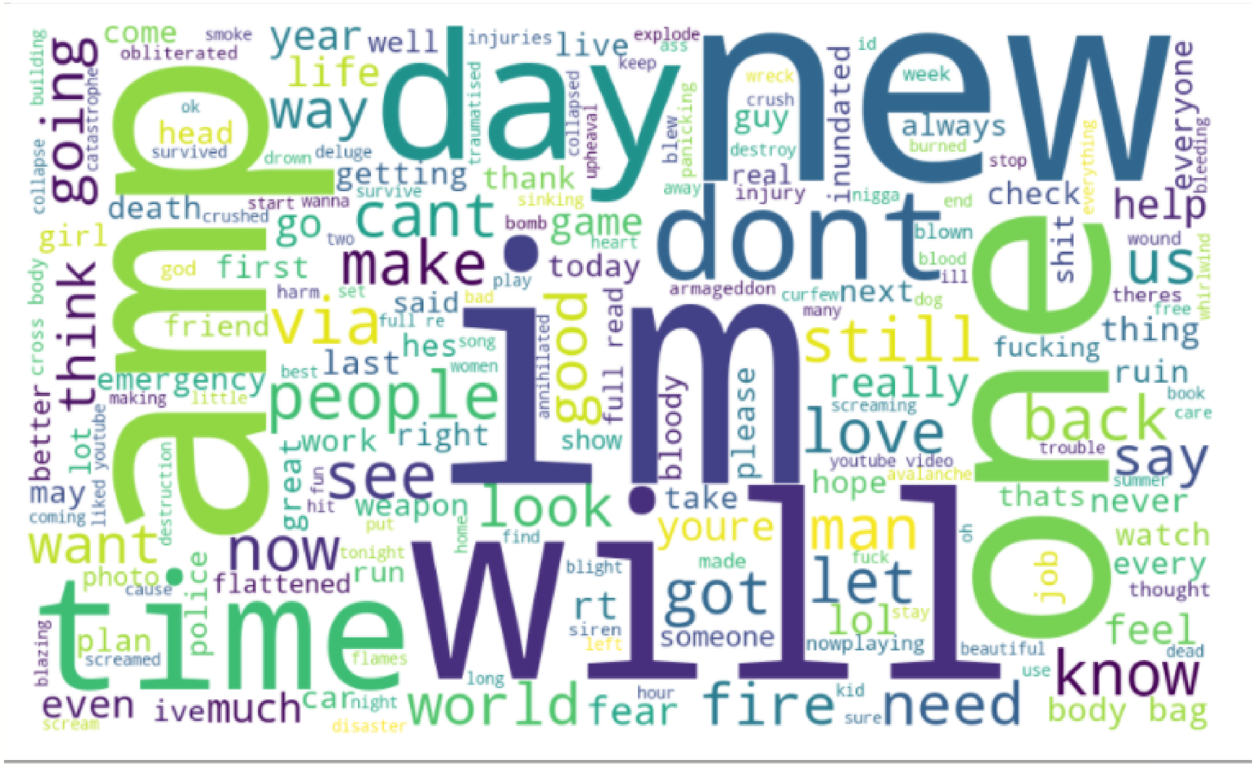
```
wordcloud = WordCloud(stopwords=stopwords,
background_color="white",width=1000,
height=600).generate(text)
```

The wordcloud for the whole data is :



The largest word depicts that the occurrence of the word is maximum in the data .

The wordcloud for data for which target = 0 is :



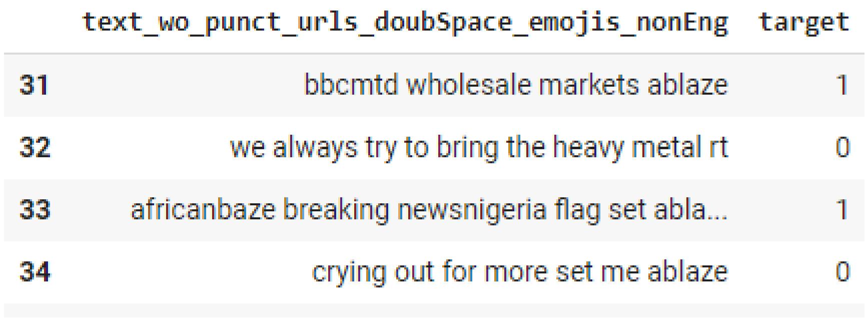
The word cloud for data for which target = 1 is

new
amp
fire
via
will
us
now
death
people
flood
home
time
look
storm
california
hiroshima
car
emergency
building
wildfire
say
one
police
attack
damage
today
back
killed
burning
area
day
suicide bomber
two
dont
city
drought
oil spill
severe thunderstorm
confirmed mh
evacuation
year
near
flooding
mass murder
burning buildings

xi. Remove all columns except text and target :

```
data=data[['text_wo_punct_urls_doubSpace_emojis_nonEng
','target']]
```

Now the data looks like this :



	text_wo_punct_urls_doubSpace_emojis_nonEng	target
31	bbcmtd wholesale markets ablaze	1
32	we always try to bring the heavy metal rt	0
33	africanbaze breaking newsnigeria flag set abla...	1
34	crying out for more set me ablaze	0

xii. Split data into train and validation :

The 20% of data will be used for testing and 80% will be used for training

After splitting the X_train will contain `4064` samples and X_test will contain `1016` samples.

b. Compute the Term Document matrix for the whole train dataset as well as for the two classes. :

```
Counter(text.split())
```

The above code will create a dictionary with words as key and frequency as value for the complete data

To find total number of unique words in the data :

```
unique_in_doc= len(Counter(text.split()))
```

Unique_in_doc will store the number of unique words in the whole data which is `13151`

To compute tdm , we will use sklearn's `CountVectorizer`

```
countvec = CountVectorizer()
countvec.fit_transform(train_df.text_wo_punct_urls_dou
bSpace_emojis_nonEng)
```

The above code will give a sparse matrix with `4064` number of rows
We can convert it into the dataframe using

```
tdm_train=pd.DataFrame(countvec.fit_transform(train_df
.text_wo_punct_urls_doubSpace_emojis_nonEng).toarray()
, columns=countvec.get_feature_names())
```

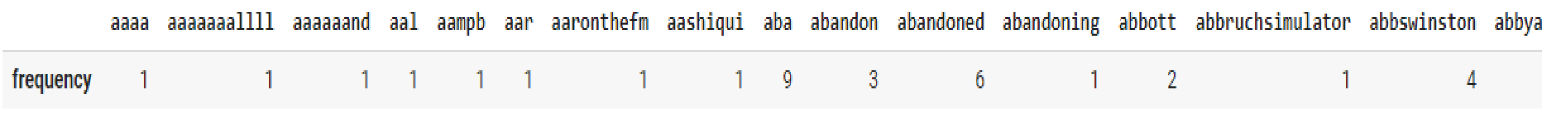
To find the overall frequency for each unique word in training data , we will sum over rows and find `frequency` and drop all the rows by using

```
frequency = tdm_train.sum()
tdm_train.drop(tdm_train.index[[i  for  i  in
range (rows_train)]],inplace=True)
```

Now the frequency will be appended to the dataframe by using :

```
tdm_train = tdm_train.append(frequency.transpose())
```

The tdm for training set will look something like this :



	aaaa	aaaaaaallll	aaaaaand	aal	aampb	aar	aaronthefm	aashiqui	aba	abandon	abandoned	abandoning	abbott	abbruchsimulator	abbswinston	abbya
frequency	1	1	1	1	1	1	1	1	9	3	6	1	2	1	4	

The similar process is done to find tdm for target = 0 and target = 1 in train data
`data_target_zero , data_target_one`
are dataframes corresponding to target= 0 and target=1 in the train data respectively . Similarly `train_df` is dataframe corresponding to whole train data.

c. Find the frequency of words in class 0 and 1. :

For getting frequency of words in class 0, we will deal with `data_target_zero` dataframe . Firstly we join text for every column and store in `text_target_zero`.

```
text_target_zero = " ".join(review for review in
data_target_zero.text_wo_punct_urls_doubSpace_emojis_n
onEng)
```

Now Counter can be used to get frequency of each word in class

```
cnt_zero = Counter(text_target_zero.split())
print(Counter(text_target_zero.split()))
```

The same procedure can be done for getting frequency of each word in class 1.

We have created dict_zero and dict_one dictionaries which stores frequency of unique words for class 0 and class 1 respectively in a key-value pair way.

d. Does the sum of the unique words in target 0 and 1 sum to the total number of unique words in the whole document? Why or why not? Explain in the report. :

We can find unique words in target 0 using ;

```
unique_in_target_zero=len(Counter(text_target_zero.spl
it()))
```

The value of unique_in_target_zero comes out to be `7909`

Similarily using :

```
unique_in_target_one=len(Counter(text_target_one.split
()))
```

The value of unique_in_target_one comes out to be `6095`
The sum is `14004` which is more than `11452 (unique_in_train)`

The reason behind can be understood by taking analogy of sets in which
A U B = A + B - A intersection B

Suppose there is a word ‘normal’ which is present 10 times in train data . 6 times, the target is 0 and 4 times, the target is 1 . When finding number of unique words in the train _data the number of unique words will be 1 . The number of unique words for target = 0 and target= 1 will both be 1 . Summing the unique words for target=0 and target=1, we found that the sum is 2 which is greater than the total number of unique words.

e. **Calculate the probability for each word in a given class.**

We have dict_zero and dict_one which consists of count of word in the respective class in key-value pair manner.

Probability of word in class zero = frequency of word in class zero / total number of words in class zero

```
word_list_zero = []
prob_zero = []
for word in dict_zero :
 prob =  dict_zero[word] / words_in_class_zero
 word_list_zero.append(word)
 prob_zero.append(prob)

prob_word_zero = dict(zip(word_list_zero, prob_zero))
```

The prob_word_zero will consist of probability of word in class zero in a key-value pair manner

The same process can be repeated for class one too.
We get prob_word_one which will contain probability of word in class one in a key-value pair manner

f. **Use Bayes with Laplace smoothing to predict the probability for sentences in the validation set.**

Laplace smoothing is used to find likelihood of a word which was not present in training data i.e likelihood for the word is zero leading to the overall zero posterior probability.

$$P(w'|positive) = \frac{\text{number of reviews with w' and y = positive } + \ \alpha}{\text{N } + \alpha * K}$$

For our case :

Alpha =1

N = number of words corresponding to the class

K =2

```
prob_class_one = []
for column in
test_df['text_wo_punct_urls_doubSpace_emojis_nonEng']:
  text_col = column.split()
  likelihood = 1
  for word in text_col :
    if word in prob_word_one :
      likelihood = likelihood * ((prob_word_one[word]
+ alpha)/ (N_target_one + alpha*k))
    else :
      likelihood = likelihood * (alpha/ N_target_one +
alpha*k)
  numerator = likelihood * prior_one
  prob_class_one.append(numerator)
```

The above code is used to find posterior probabilities for class one

The same procedure can be done for class zero

After we get posterior probabilities for both classes, to find the target value , we need to compare the probabilities which is done using

```
y_pred= []
```

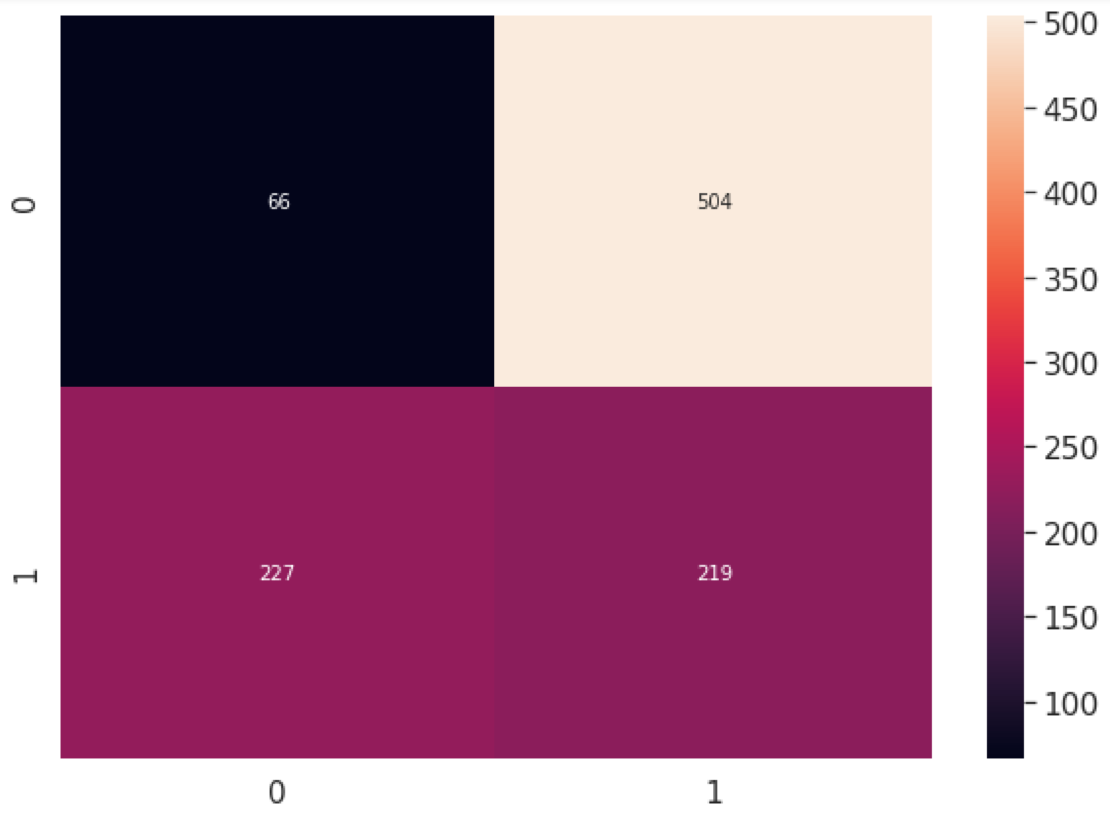
```
for i in range (test_df.shape[0]):
  if(prob_class_zero[i] > prob_class_one[i]):
   y_pred.append(0)
  else:
   y_pred.append(1)
```

y_pred will contain target labels for the test data

g. Print the confusion matrix with precision, recall and f1 score

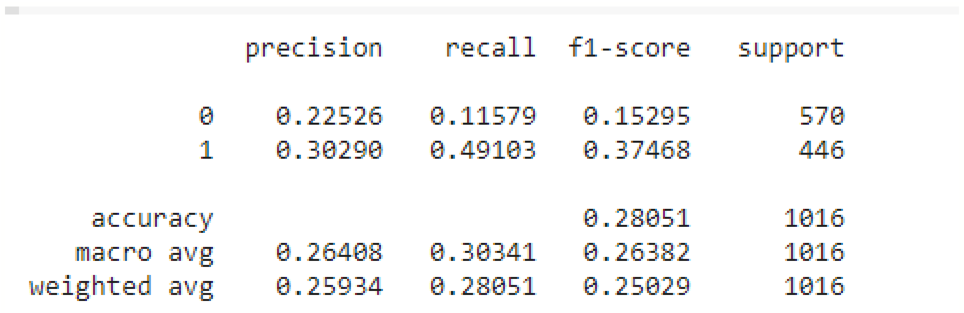
```
def plot_confusion_matrix(y_test, y_pred):
      cm = confusion_matrix(y_test, y_pred)

      df_cm = pd.DataFrame(cm, range(cm.shape[0]),
                   range(cm.shape[1]))
    plt.figure(figsize = (10,7))
    # Plot the confusion matrix
    sn.set(font_scale=1.4) #for label size
    sn.heatmap(df_cm,
annot=True,fmt='.0f',annot_kws={"size": 10})# font
size
    plt.show()
plot_confusion_matrix(y_test, y_pred)
```

The code above prints confusion matrix for test data .
Confusion matrix looks like this



To get precision, recall, f1 scores :

```
print(metrics.classification_report(y_test,    y_pred,
digits=5))
```



```
              precision    recall  f1-score   support

           0    0.22526   0.11579   0.15295       570
           1    0.30290   0.49103   0.37468       446

    accuracy                        0.28051      1016
   macro avg    0.26408   0.30341   0.26382      1016
weighted avg    0.25934   0.28051   0.25029      1016
```