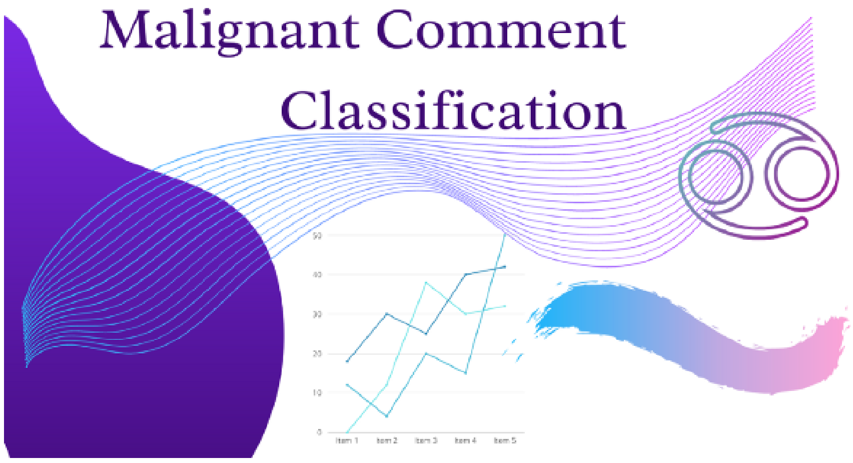
MALIGNANT COMMENTS CLASSIFICATION PROJECT



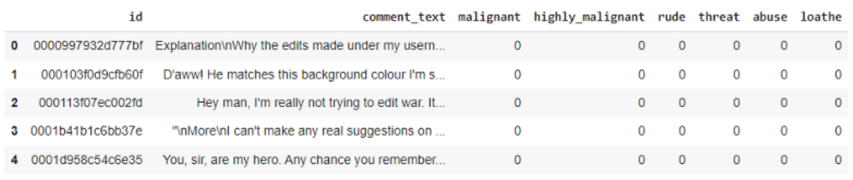
Online forums and social media platforms have provided individuals with the means to put forward their thoughts and freely express their opinion on various issues and incidents. In some cases, these online comments contain explicit language which may hurt the readers. Comments containing explicit language can be classified into myriad categories such as Toxic, Severe Toxic, Obscene, Threat, Insult, and Identity Hate. The threat of abuse and harassment means that many people stop expressing themselves and give up on seeking different opinions.

Step 1:Importing Libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
import xgboost
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
confusion_matrix,
classification_report,roc_curve,roc_auc_score,auc
from sklearn.model_selection import train_test_split
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix,f1_sco
re
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import
cross_val_score,GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import
RandomForestClassifier,AdaBoostClassifier,GradientBoostingCl
assifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

import joblib

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

Step 2: Importing Dataset :

```
df_train =
pd.read_csv("/content/drive/MyDrive/train.csv",encoding='utf
-8')

df_test =
pd.read_csv("/content/drive/MyDrive/test.csv",encoding='utf-
8')


df_train.head()
```



	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

```
print(df_train.shape)
print(df_test.shape)


(159571, 8)
(153164, 2)


print(df_train.info())
print(df_test.info())


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159571 entries, 0 to 159570
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                159571 non-null  object
 1   comment_text      159571 non-null  object
 2   malignant         159571 non-null  int64
 3   highly_malignant  159571 non-null  int64
 4   rude              159571 non-null  int64
 5   threat            159571 non-null  int64
 6   abuse             159571 non-null  int64
 7   loathe            159571 non-null  int64
dtypes: int64(6), object(2)
memory usage: 9.7+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153164 entries, 0 to 153163
Data columns (total 2 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            153164 non-null  object
 1   comment_text  153164 non-null  object
dtypes: object(2)
memory usage: 2.3+ MB
None
```

Step 3: Problem Statement and Background

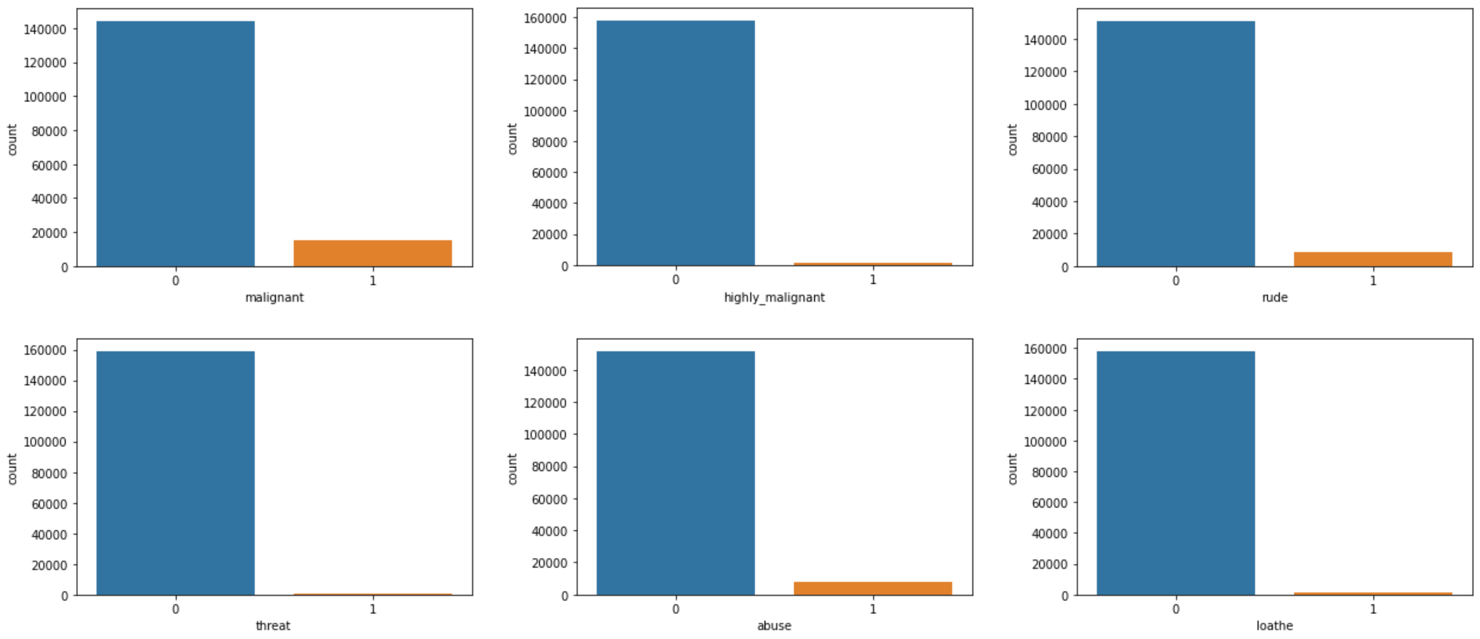
The background for the problem originates from the multitude of online forums, where-in people participate actively and make comments. As the comments some times may be abusive, insulting or even hate-based, it becomes the responsibility of the hosting organizations to ensure that these conversations are not of negative type. The task was thus to build a model which could make prediction to classify the comments into various categories. Through the head().method we can observed 8 rows. As per problem statement this is classification problem and here need to predict whether comment is malignant or not.

Step 4: Studying data & identifying hidden patterns

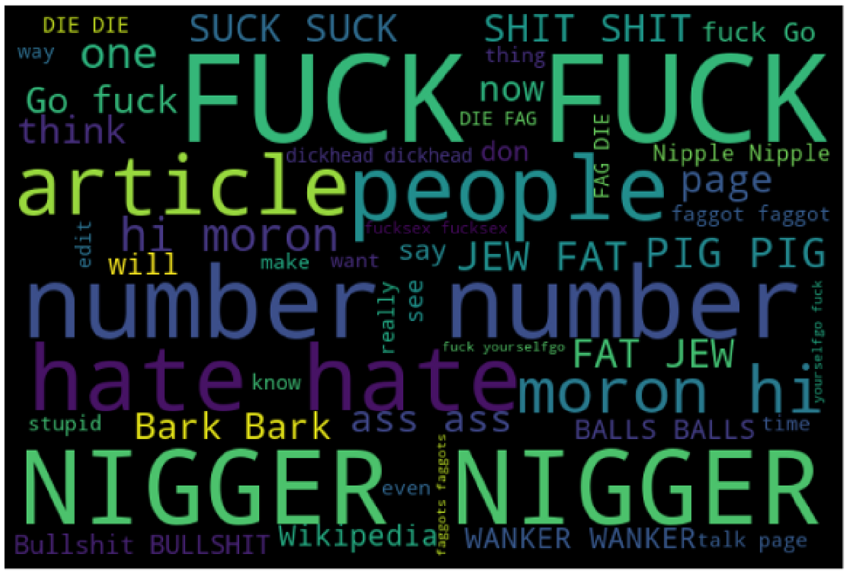
I had a dataset of 159571 samples of comments along with their labels. I observed that every 1 in 10 samples was toxic, every 1 in 50 samples was obscene and insulting, but the occurrences of sample being severe-toxic, threat and identity hate was extremely rare. The first 5 rows appeared as shown above.

For the next visualisation, We have checked the null values also and I had length of comments on the independent axis again similar to the previous plot. But instead of counting number of comments, I counted comments belonging to each of the different categories.

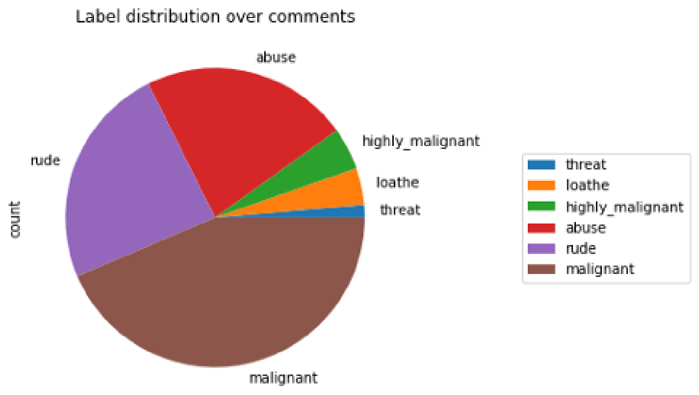
```
#checking the values counts
columns = [ 'malignant', 'highly_malignant', 'rude',
'threat',
       'abuse', 'loathe']

for col in columns:
  print(col)
  print(df_train[col].value_counts())
  sns.countplot(df_train[col])
  plt.show()
```



```
# checking the most offencive words
hams = df_train['comment_text'][df_train['malignant']==1]
spam_cloud = WordCloud(width=600, height=400,
background_color = 'black', max_words=50).generate(''.
join(hams))
plt.figure(figsize=(10,8), facecolor='k')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



The above ***visualisation*** *plots the number of comments belonging to various categories. Toxic comments were highest in number, followed by obscene, insult, severe-toxic, identity-hate and threat using count plot.*



```
# checking the total distribution of commants
df_distribution = df_train[columns].sum()\
                            .to_frame()\
                            .rename(columns={0: 'count'})\
                            .sort_values('count')
df_distribution.plot.pie(y = 'count',
                          title = 'Label distribution over 
comments',
                          figsize=(5,5))\
.legend(loc = 'center left', bbox_to_anchor = (1.3, 0.5))
```

STEP 5: Data Preprocessing

1. removal of special character ***marks:*** There are many numeric and special characters which we have to remove from dataset.

```
#replacing email address with email


df_train['comment_text']
=df_train['comment_text'].str.replace(r'^.+@[^\.].*\.[a-z]
{2,}$','emailaddress')


# Replace URLs with 'webaddress'


df_train['comment_text'] =
df_train['comment_text'].str.replace(r'^http\://[a-zA-Z0-9\-
\.]+\.[a-zA-Z]{2,3}(/\S*)?$','webaddress')


# Replace money symbols with 'moneysymb' (£ can by typed 
with ALT key + 156)
```

```
df_train['comment_text'] =
df_train['comment_text'].str.replace(r'£|\$', 'dollers')


# Replace 10 digit phone numbers (formats include
paranthesis, spaces, no spaces, dashes) with 'phonenumber'


df_train['comment_text'] =
df_train['comment_text'].str.replace(r'^\(?[\d]{3}\)?[\s-]?
[\d]{3}[\s-]?[\d]{4}$','phonenumber')


# Replace numbers with 'number'


df_train['comment_text'] =
df_train['comment_text'].str.replace(r'\d+(\.\d+)?',
'number')


# Remove punctuation


df_train['comment_text'] =
df_train['comment_text'].str.replace(r'[^\w\d\s]', ' ')


# Replace whitespace between terms with a single space


df_train['comment_text'] =
df_train['comment_text'].str.replace(r'\s+', ' ')


# Remove leading and trailing whitespace


df_train['comment_text'] =
df_train['comment_text'].str.replace(r'^\s+|\s+?$', '')
```

2. Preparation for removal of punctuation marks & Updating the list of stop words:

```
df_train['comment_text'] =
df_train['comment_text'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in
string.punctuation))

stop_words = set(stopwords.words('english') + ['u', 'ü',
'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])
df_train['comment_text'] =
df_train['comment_text'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in stop_words))

lem=WordNetLemmatizer()
df_train['comment_text'] =
df_train['comment_text'].apply(lambda x: ' '.join(
 lem.lemmatize(t) for t in x.split()))
```

3. Stemming and Lemmatising :

I imported the string library comprising all punctuation characters and appended the numeric digits to it, as those were required to be removed too.

Stemming is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word. E.g. words like “stems”, “stemmer”, “stemming”, “stemmed” are based on “stem”. This helps in achieving the training process with a better accuracy.

Lemmatising is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but not exactly same. Lemmatising depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. I used the ***word-net library*** in nltk for this purpose. Stemmer and Lemmatizer were imported from nltk.

4. Applying Tf-id Vectorizer :

***Tf-id** Vectorizer is used for converting a string of words into a matrix of words. Column headers have the words themselves and the cell values signify the frequency of occurrence of the word.*

```
#  Convert text into vectors using TF-IDF
tf_vec = TfidfVectorizer(max_features = 10000,
stop_words='english')
X_features = tf_vec.fit_transform(df_train['comment_text'])
X = X_features
```

STEP 6: Model Building :

```
y = df_train['malignant']


# spliting into train and test


x_train,x_test,y_train,y_test=train_test_split(
X,y,random_state=56,test_size=.30)


#RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is
{}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is
{}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))


Training accuracy is 0.9988451105202374
Test accuracy is 0.9555899064171123
[[42695   552]
 [ 1574  3051]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     43247
           1       0.85      0.66      0.74      4625

    accuracy                           0.96     47872
   macro avg       0.91      0.82      0.86     47872
weighted avg       0.95      0.96      0.95     47872
```

```
# DecisionTreeClassifier
DT = DecisionTreeClassifier()

DT.fit(x_train, y_train)
y_pred_train = DT.predict(x_train)
print('Training accuracy is
{}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = DT.predict(x_test)
print('Test accuracy is
{}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))


Training accuracy is 0.9988809210467416
Test accuracy is 0.9396933489304813
```

```
[[41867  1380]
 [ 1507  3118]]
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     43247
           1       0.69      0.67      0.68      4625

    accuracy                           0.94     47872
   macro avg       0.83      0.82      0.83     47872
weighted avg       0.94      0.94      0.94     47872
```

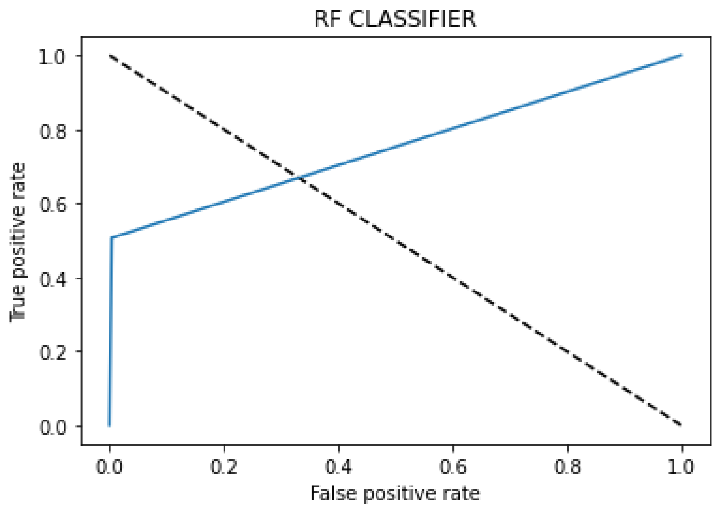
```
# MultinomialNBClassifiear
MN = MultinomialNB()

MN.fit(x_train, y_train)
y_pred_train = MN.predict(x_train)
print('Training accuracy is
{}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = MN.predict(x_test)
print('Test accuracy is
{}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))


Training accuracy is 0.9514588313234675
Test accuracy is 0.9488218582887701
[[43079   168]
 [ 2282  2343]]
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     43247
           1       0.93      0.51      0.66      4625

    accuracy                           0.95     47872
   macro avg       0.94      0.75      0.81     47872
weighted avg       0.95      0.95      0.94     47872
```

```
# GradientBoostingClassifier
GBC = GradientBoostingClassifier()

GBC.fit(x_train, y_train)
y_pred_train = GBC.predict(x_train)
print('Training accuracy is
{}'.format(accuracy_score(y_train, y_pred_train)))
```

```
y_pred_test = GBC.predict(x_test)
print('Test accuracy is
{}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))


Training accuracy is 0.9988809210467416
Test accuracy is 0.9396933489304813
[[41867  1380]
 [ 1507  3118]]
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     43247
           1       0.69      0.67      0.68      4625

    accuracy                           0.94     47872
   macro avg       0.83      0.82      0.83     47872
weighted avg       0.94      0.94      0.94     47872
```

Plotting the graph which tells us about the area under curve , more the area under curve more will be the better prediction
# model is performing good :

```
fpr,tpr,thresholds=roc_curve(y_test,y_pred_test)
roc_auc=auc(fpr,tpr)
plt.plot([0,1],[1,0],'k--')
plt.plot(fpr,tpr,label = 'RF Classifier')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('RF CLASSIFIER')
plt.show()
```



as peri found above models scores random forest is giving better accuracy hence considering as final model and saving it using joblib library.

Predicting the test data:

```
df_test =tf_vec.fit_transform(df_test['comment_text'])
df_test

prediction=RF.predict(df_test)
prediction
```

STEP 7: Saving Model :

```
import joblib

joblib.dump(RF,"maligant_commant.pkl")

['maligant_commant.pkl']
```

STEP 7: Conclution :

To conclude, In this project we have performed various techniques like Data analysis which has proven way for any organizations and enterprises to gain the information they need to make better decisions, serve their customers, and increase productivity and revenue. The benefits of data analysis are almost too numerous to count, and some of the most rewarding benefits include getting the right information for business, getting more value out of it. And it creates more effective marketing campaigns, gaining a better understanding of customers, and so on. In addition, Stemming is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word. E.g. words like “stems”, “stemmer”, “stemming”, “stemmed” are based on “stem”. This helps in achieving the training process with a better accuracy.

Lemmatising is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but not exactly same. Lemmatising depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. I used the word-net library in nltk for this purpose. Stemmer and Lemmatizer were imported from nltk.

Tfidf Vectorizer is used for converting a string of words into a matrix of words. Column headers have the words themselves and the cell values signify the frequency of occurrence of the word.Finally I reached the core part of the project, where I could start building the classifier. I had use 4 major algorithm . Where, our random forest model giving the good accuracy hence selected it and saved it using joblib library.