MALIGANAT COMMENT PROJECT

Submitted by:

KHUSHBOO GUPTA

ACKNOWLEDGMENT

First and foremost, I would like to thank Flip Robo Technologies to provide me a chance to work on this project. It was a great experience to work on this project under your guidance.

I would like to present my gratitude to the following websites:

- Zendesk
- Kaggle
- Datatrained Notes
- Sklearn.org
- Crazyegg
- Towards data science

These websites were of great help and due to this, I was able to complete my project effectively and efficiently.

INTRODUCTION

• Business Problem Framing

This project is more about exploration, feature engineering and classification that can be done on this data. Since the data set is huge and includes many categories of comments, we can do good amount of data exploration and derive some interesting features using the comments text column available. You need to build a model that can differentiate between comments and its categories.

• Conceptual Background of the Domain Problem

Basic EDA concepts and classification algorithms must be known to work on this project. One should know what is a malignant comment and what type of words make it a malignant one? How the comment can be differentiating between different categories like threat, loathe, rude etc.

• Review of Literature

The proliferation of social media enables people to express their opinions widely online. However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users. Although researchers have found that hate is a problem across multiple platforms, there is a lack of models for online hate detection.

Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour.

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

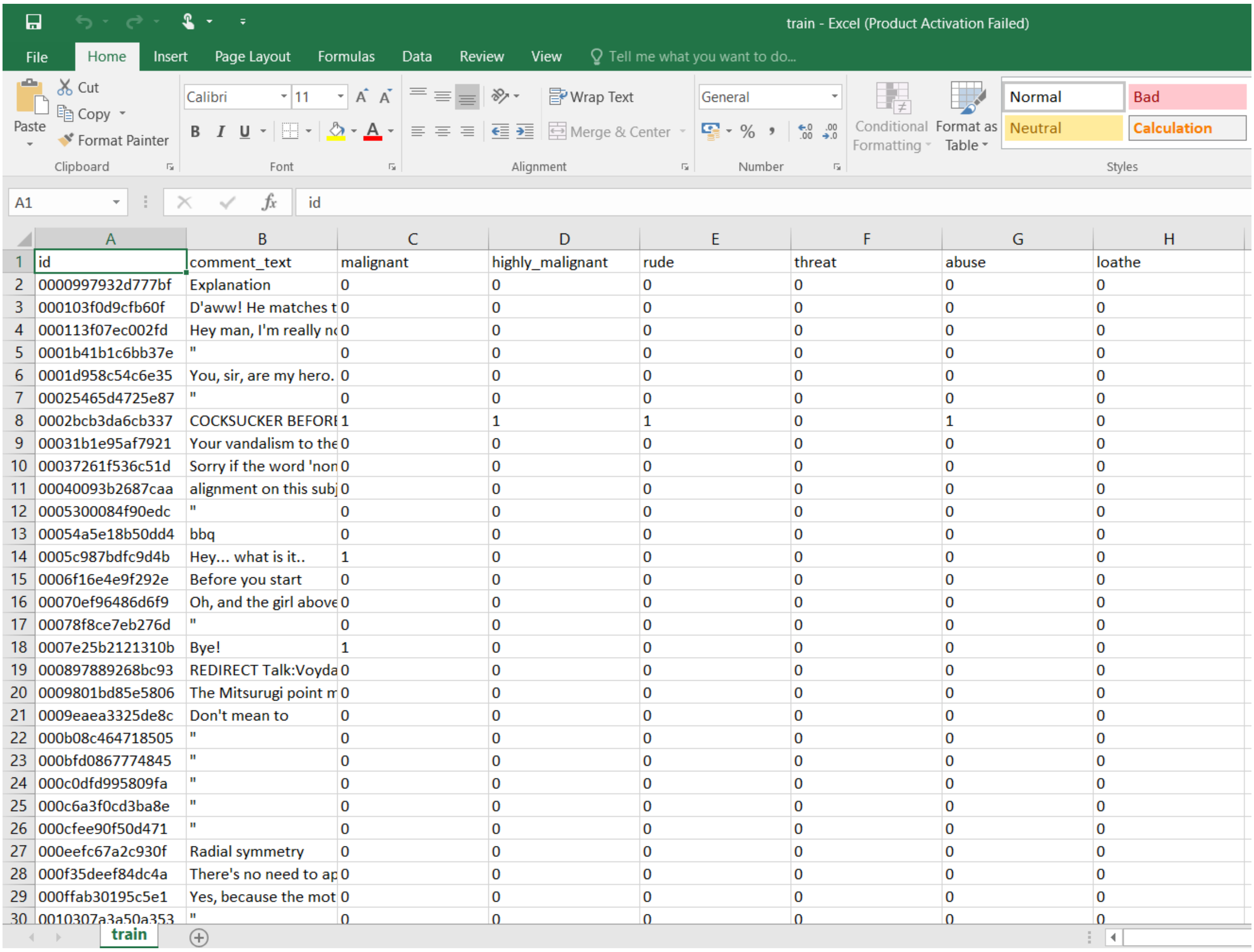
Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate speech, machine learning can be used to fight it. The problem we sought to solve was the tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as unoffensive, but “u are an idiot” is clearly offensive.

Our goal is to build a prototype of online hate and abuse comment classifier which can used to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyberbullying.

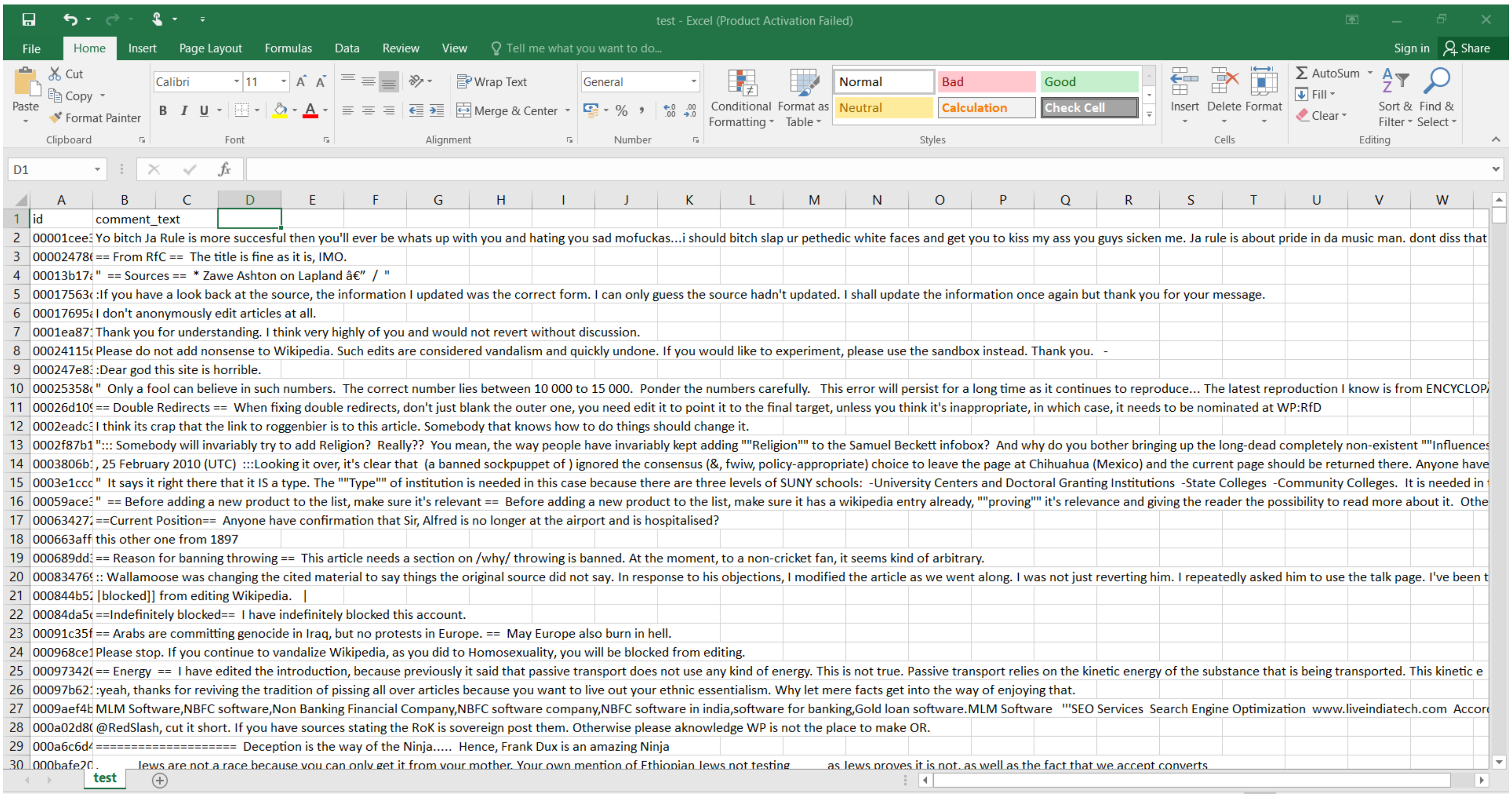
Analytical Problem Framing

- Data Sources and their formats

The dataset is provided by the internship organization in an csv format which contains the data in code sheet. Train dataset contains 8 columns and 159571 rows while test dataset contains 2 columns and 153164 rows. There are words which make a comment make it fall in any of these categories. Every comment falls in at least one of the category and even more than one.



	A	B	C	D	E	F	G	H
1	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
2	0000997932d777bf	Explanation	0	0	0	0	0	0
3	000103f0d9cfb60f	D'aww! He matches t	0	0	0	0	0	0
4	000113f07ec002fd	Hey man, I'm really nc	0	0	0	0	0	0
5	0001b41b1c6bb37e	"	0	0	0	0	0	0
6	0001d958c54c6e35	You, sir, are my hero.	0	0	0	0	0	0
7	00025465d4725e87	"	0	0	0	0	0	0
8	0002bcb3da6cb337	COCKSUCKER BEFORE	1	1	1	0	1	0
9	00031b1e95af7921	Your vandalism to the	0	0	0	0	0	0
10	00037261f536c51d	Sorry if the word 'non	0	0	0	0	0	0
11	00040093b2687caa	alignment on this subj	0	0	0	0	0	0
12	0005300084f90edc	"	0	0	0	0	0	0
13	00054a5e18b50dd4	bbq	0	0	0	0	0	0
14	0005c987bdfc9d4b	Hey... what is it..	1	0	0	0	0	0
15	0006f16e4e9f292e	Before you start	0	0	0	0	0	0
16	00070ef96486d6f9	Oh, and the girl above	0	0	0	0	0	0
17	00078f8ce7eb276d	"	0	0	0	0	0	0
18	0007e25b2121310b	Bye!	1	0	0	0	0	0
19	000897889268bc93	REDIRECT Talk:Voyda	0	0	0	0	0	0
20	0009801bd85e5806	The Mitsurugi point m	0	0	0	0	0	0
21	0009eaea3325de8c	Don't mean to	0	0	0	0	0	0
22	000b08c464718505	"	0	0	0	0	0	0
23	000bfd0867774845	"	0	0	0	0	0	0
24	000c0dfd995809fa	"	0	0	0	0	0	0
25	000c6a3f0cd3ba8e	"	0	0	0	0	0	0
26	000cfee90f50d471	"	0	0	0	0	0	0
27	000eefc67a2c930f	Radial symmetry	0	0	0	0	0	0
28	000f35deef84dc4a	There's no need to ap	0	0	0	0	0	0
29	000ffab30195c5e1	Yes, because the mot	0	0	0	0	0	0
30	0010307a3a50a353	"	0	0	0	0	0	0



- # Dataset Description

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes ‘Id’, ‘Comments’, ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.

The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

The data set includes:

- **Malignant:** It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.
- **Highly Malignant:** It denotes comments that are highly malignant and hurtful.
- **Rude:** It denotes comments that are very rude and offensive.
- **Threat:** It contains indication of the comments that are giving any threat to someone.
- **Abuse:** It is for comments that are abusive in nature.
- **Loathe:** It describes the comments which are hateful and loathing in nature.
- **ID:** It includes unique Ids associated with each comment text given.
- **Comment text:** This column contains the comments extracted from various social media platforms.

- Libraries Used

I am using different libraries to explore the datatset.

1. Pandas – It is used to load and store the dataset. We can discuss the dataset with the pandas different attributes like .info, .columns, .shape
2. Seaborn – It is used to plot the different types of plots like catplot, lineplot, countplot and more to have a better visualization of the dataset.
3. Matplotlib.pyplot – It helps to give a proper description to the plotted graph by seaborn and make our graph more informative.
4. Numpy – It is the library to perform the numerical analysis to the dataset

Load the Dataset

Importing the libraries

```
In [1]:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

Importing the dataset

```
In [2]:
df=pd.read_csv(r'F:\Internship - Data Science\Malignant Comments Classifier Project\train.csv')  #train dataset
df1=pd.read_csv(r'F:\Internship - Data Science\Malignant Comments Classifier Project\test.csv')  #test dataset
```

```
In [3]:
df.head()  #first 5 rows of the train dataset
```

Out[3]:

	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

We have successfully load our dataset for our further processes.

Checking the Attributes

- First & last five rows the dataset
- Shape of the dataset
- Columns present in the dataset
- Brief info about the dataset
- Datatype of each column
- Null values present in the dataset
- Number of unique values present in each column

```
In [7]: df.shape
```

```
Out[7]: (159571, 8)
```

It contains 159571 rows and 8 columns

```
In [8]: df1.shape
```

```
Out[8]: (153164, 2)
```

It contains 153164 rows and 2 columns

```
In [9]: print(df.info())  #a brief info for both the dataset
        print('\n')
        print(df1.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
```

In [10]:
```
print(df.dtypes)  #datatypes of each column in each dataset
print('\n')
print(df1.dtypes)
```

```
id                  object
comment_text        object
malignant            int64
highly_malignant     int64
rude                 int64
threat               int64
abuse                int64
loathe               int64
dtype: object


id              object
comment_text    object
dtype: object
```

In [11]:
```
print(df.nunique())   #unique values in each column
print('\n')
print(df1.nunique())
```

```
id                  159571
comment_text        159570
malignant                2
highly_malignant         2
rude                     2
threat                   2
abuse                    2
loathe                   2
dtype: int64
```

In [12]:

```python
print('Null Values in training set')
print('\n')
print(df.isnull().sum())
print('Null Values in test set')
print('\n')
print(df1.isnull().sum())
```

```
Null Values in training set


id                  0
comment_text        0
malignant           0
highly_malignant    0
rude                0
threat              0
abuse               0
loathe              0
dtype: int64
Null Values in test set


id              0
comment_text    0
dtype: int64
```

No null values present in the dataset

```
In [15]: cols=['malignant','highly_malignant','rude','threat','abuse','loathe']
```

```
In [16]: for i in cols:  #printing the value count in each column
             print(df[i].value_counts())
             print('\n')
```

```
0    144277
1     15294
Name: malignant, dtype: int64


0    157976
1      1595
Name: highly_malignant, dtype: int64


0    151122
1      8449
Name: rude, dtype: int64


0    159093
1       478
Name: threat, dtype: int64


0    151694
1      7877
Name: abuse, dtype: int64


0    158166
1      1405
Name: loathe, dtype: int64
```

Now we have checked the attributes for the dataset and get a rough idea about the dataset like the no of rows & columns, datatype & null values in the dataset.
We don't have any null value in the dataset i.e. we don't have to deal with them.

Now, we see that the dataset is not balanced. The target has column has a large difference between both the labels. So, we have to make the dataset balanced for the proper ML model.
We will do that by using the SMOTE which will make some extra rows whose percentage is less in the dataset & make the counting of both the labels equal.

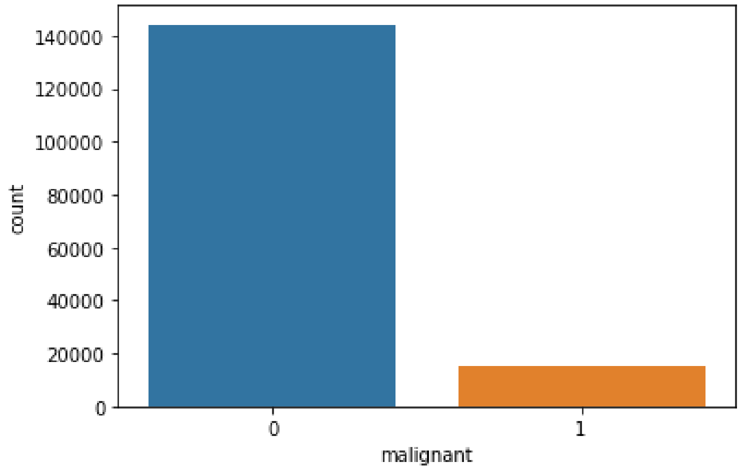
Now the dataset is balance & we can proceed further.

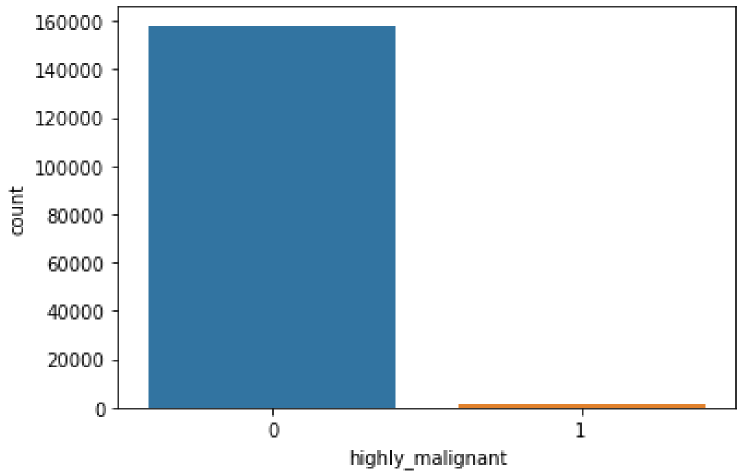
EXPLORATORY DATA ANALYSIS

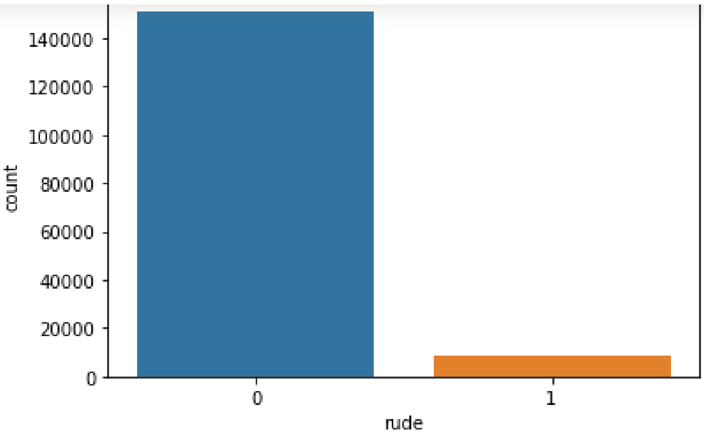
Plotting each target variable to understand the two labels in each column. Each column has a very low percentage of ‘1’ that i.e. very low frequency of yes.

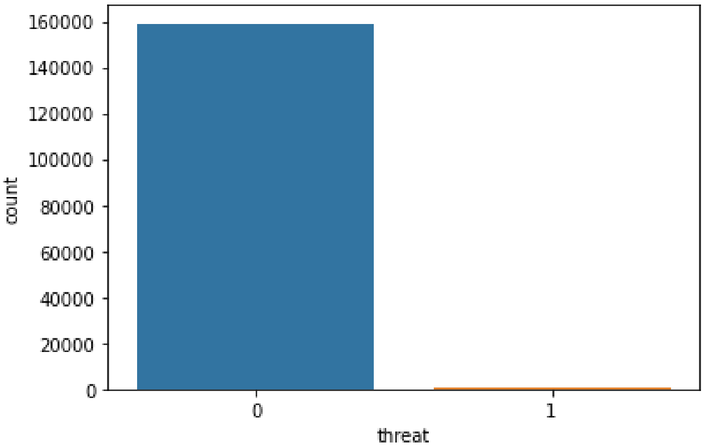
In [17]:

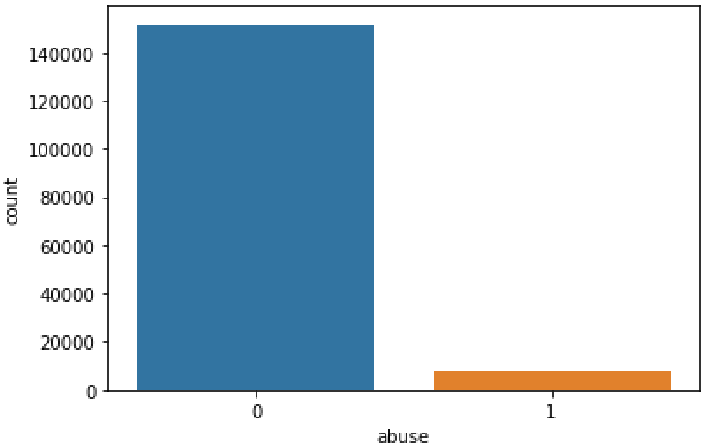
```
#plotting the diiferent columns value countsb

for i in cols:
    sns.countplot(df[i])
    plt.show()
```





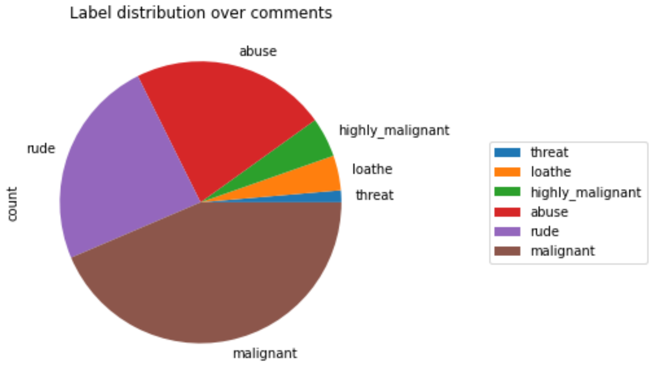

140000
120000
100000
80000
60000
40000
20000
0
count
0
1
rude


160000
140000
120000
100000
80000
60000
40000
20000
0
count
0
1
threat


140000
120000
100000
80000
60000
40000
20000
0
count
0
1
abuse

Plotting the percentage of each target variable. Most of the comment falls in the category of malignant followed by rude & abuse.

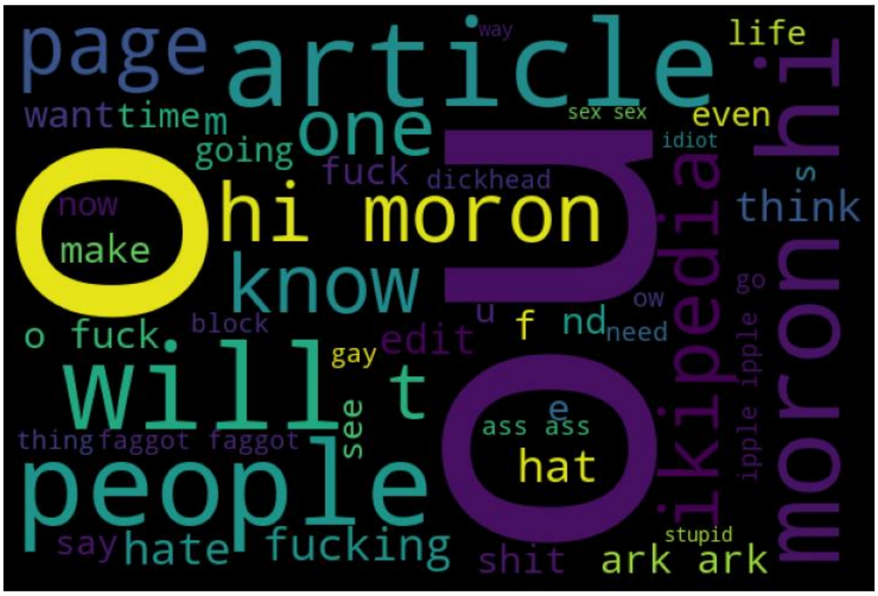
```
In [30]:  df_distribution = df[cols].sum()\
                                .to_frame()\
                                .rename(columns={0: 'count'})\
                                .sort_values('count')

          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))

Out[30]:  <matplotlib.legend.Legend at 0x241352e8040>
```



We have a very high ratio of malignant comments followed by rude and abuse

Getting ideas of those word which make a comment bad.

```
In [23]:  #Getting sense of loud words which are offensive
          from wordcloud import WordCloud
          hams = df['comment_text'][df['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()
```



Cleaning the comment column

In the comment_text column we have different types of comments but it contains some special characters and other things which needs to be removed to have a better perception. We are going to remove the extra spacing and keep only the alphabets using regexp_tokenize module.

In [18]:
```
import string
```

In [19]:
```
df['length'] = df['comment_text'].str.len()  #chcecking the length of the comment text
df.head(2)
```

Out[19]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	264
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	112

In [20]:
```
# Replacing '\n' with ' '

df.comment_text = df.comment_text.str.replace('\n',' ')

from nltk.tokenize import regexp_tokenize

# Keeping only text with letters a to z, 0 to 9 and words like can't, don't, couldn't etc

df.comment_text = df.comment_text.apply(lambda x: ' '.join(regexp_tokenize(x,"[a-z']+")))
```

In [21]:
```
df['clean_length'] = df.comment_text.str.len()  #Checking the length of the comment after clearing
df.head()
```

Out[21]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length	clean_length
0	0000997932d777bf	xplanation hy the edits made under my username...	0	0	0	0	0	0	264	229
1	000103f0d9cfb60f	'aww e matches this background colour 'm seemi...	0	0	0	0	0	0	112	79
2	000113f07ec002fd	ey man 'm really not trying to edit war t's ju...	0	0	0	0	0	0	233	225
3	0001b41b1c6bb37e	ore can't make any real suggestions on improve...	0	0	0	0	0	0	622	585
4	0001d958c54c6e35	ou sir are my hero ny chance you remember what...	0	0	0	0	0	0	67	61

In [22]:
```
# Total length removal
print ('Origian Length', df.length.sum())
print ('Clean Length', df.clean_length.sum())
```

```
Origian Length 62797479
Clean Length 56139334
```

Statistical Summary & Correlation

We will describe the statistical summary of the dataset and find the correlation of each column.

Statistical Summary

In [24]:
```
df.describe()
```

Out[24]:

	malignant	highly_malignant	rude	threat	abuse	loathe	length	clean_length
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805	393.539421	351.814139
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420	589.804780	530.032028
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	96.000000	81.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	205.000000	182.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	434.500000	393.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5000.000000	5000.000000

Correlation

In [25]:
```
corr=df.corr()
print(corr)
print(sns.heatmap(corr, cmap='twilight',annot=True))
```

```
                  malignant  highly_malignant      rude    threat     abuse  \
malignant          1.000000          0.308619  0.676515  0.157058  0.647518
highly_malignant   0.308619          1.000000  0.403014  0.123601  0.375807
rude               0.676515          0.403014  1.000000  0.141179  0.741272
threat             0.157058          0.123601  0.141179  1.000000  0.150022
abuse              0.647518          0.375807  0.741272  0.150022  1.000000
loathe             0.266009          0.201600  0.286867  0.115128  0.337736
length            -0.054649          0.009747 -0.043097 -0.007909 -0.045239
clean_length      -0.078878         -0.022518 -0.063586 -0.017325 -0.064501

                    loathe    length  clean_length
malignant         0.266009 -0.054649     -0.078878
highly_malignant  0.201600  0.009747     -0.022518
rude              0.286867 -0.043097     -0.063586
threat            0.115128 -0.007909     -0.017325
abuse             0.337736 -0.045239     -0.064501
loathe            1.000000 -0.014119     -0.025646
length           -0.014119  1.000000      0.976183
clean_length     -0.025646  0.976183      1.000000
AxesSubplot(0.125,0.125;0.62x0.755)
```

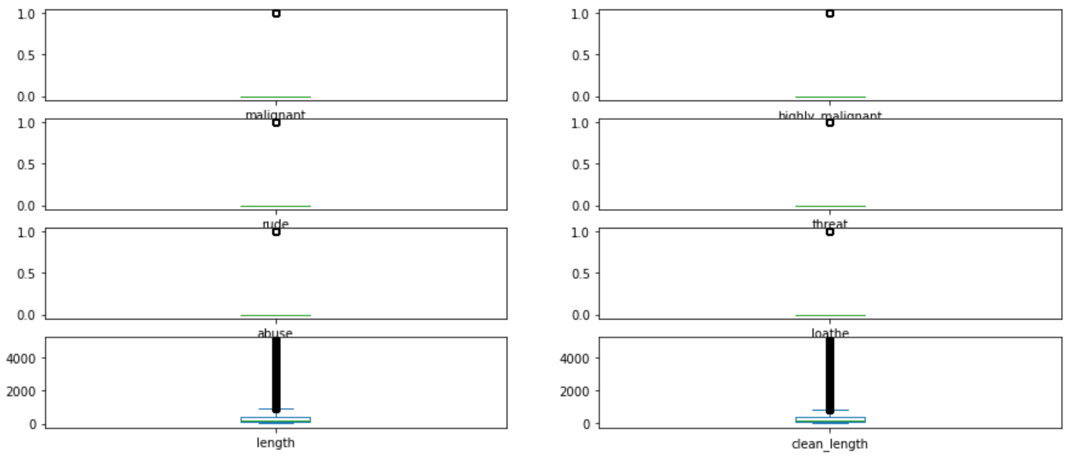
- We have 159571 rows in the dataset
- Being only two variables in the columns there will be no outliers.
- Also very little skewness will be present in the target variables.
- Minimum of each target variable is 0 and maximum is 1

Plotting the Outliers

Using the boxplot, we plot each column and interpret whether the outliers are present or not in that column. In our case each column contains only two variables either zero or one so there will be no outliers and no need to remove any rows. We can proceed further towards feature engineering.

Plotting the outliers

```
In [27]: df.plot(kind='box',subplots=True,layout=(5,2),figsize=(15,8))

Out[27]: malignant              AxesSubplot(0.125,0.749828;0.352273x0.130172)
         highly_malignant    AxesSubplot(0.547727,0.749828;0.352273x0.130172)
         rude                   AxesSubplot(0.125,0.593621;0.352273x0.130172)
         threat              AxesSubplot(0.547727,0.593621;0.352273x0.130172)
         abuse                  AxesSubplot(0.125,0.437414;0.352273x0.130172)
         loathe              AxesSubplot(0.547727,0.437414;0.352273x0.130172)
         length                 AxesSubplot(0.125,0.281207;0.352273x0.130172)
         clean_length        AxesSubplot(0.547727,0.281207;0.352273x0.130172)
         dtype: object
```



Now, our data cleaning & visualization part is done and we proceed with the model building.

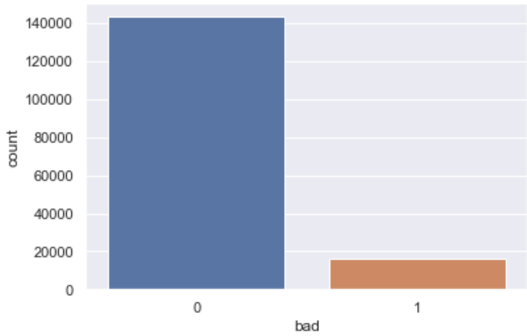
MODEL BUILDING

Before moving forward towards the model training we have a challenge that we have 6 target variables i.e. malignant, highly_malignant, rude, threat, abuse, loathe which needs to tackle down. So, we will create a separate which will contain the sum of different row entries. If the sum of all the target variable is 1 or more than 1, then it is malignant otherwise if sum is zero, then it is not malignant.

In [31]:
```
#combining all the target columns into one

df['bad'] =df[cols].sum(axis =1)
print(df['bad'].value_counts())
df['bad'] = df['bad'] > 0
df['bad'] = df['bad'].astype(int)
print(df['bad'].value_counts())
```

```
0    143346
1      6360
3      4209
2      3480
4      1760
5       385
6        31
Name: bad, dtype: int64
0    143346
1     16225
Name: bad, dtype: int64
```

In [32]:
```
sns.set()
sns.countplot(x="bad" , data = df)
plt.show()
```



Most of the comments are not fall in the bad category

We will import important libraries for the building the ML model and defining the different models for our easiness.

Finding the best random state for the train test split.

In [35]:
```
#defining the models

lg=LogisticRegression()
rdc=RandomForestClassifier()
dtc=DecisionTreeClassifier()
knc=KNeighborsClassifier()
ad=AdaBoostClassifier()
gb=GradientBoostingClassifier()
```

Finding the best random state

In [35]:
```
model=[lg,rdc,svc,dtc,knc,ad,gb]
maxAccu=0
bestRS=0
for i in range(40,60):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=i,test_size=.30)
    lg.fit(x_train,y_train)
    pred=lg.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        bestRS=i
print('Best Accuracy score is', maxAccu , 'on random state', bestRS)
```

```
Best Accuracy score is 0.9485503008021391 on random state 47
```

In [36]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=47,test_size=.30)
```

Classification Algorithms

We have use seven different regression algorithms to find the best model for our problem.

- **Logistic Regression**
  - from sklearn.linear_model import LogisticRegression
- **Decision Tree Classifier**
  - from sklearn.tree import DecisionTreeClassifier
- **KNN Classifier**
  - from sklearn.neighbors import KNeighborsClassifier
- **Random Forest Classifier**
  - from sklearn.ensemble import RandomForestClassifier
- **Multinomial NB**
  - from sklearn.naive_bayes import MultinomialNB
- **AdaBoost Classifier**
  - from sklearn.ensemble import AdaBoostClassifier
- **GradientBoosting Classifier**
  - from sklearn.ensemble import GradientBoostingClassifier

Let's see the different models accuracy at once.

MODEL	ACCURACY
Logistic Regression	0.9485503008021391
Decision Tree Classifier	0.9198696524064172
Random Forest Classifier	0.9454169451871658
KNN Classifier	0.5836397058823529
Multinomial NB	0.9416986965240641
Adaboost Classifier	0.93829378342246
GradientBoost Classifer	0.93413686497326

Logistic Regression

In [49]:
```
lg.fit(x_train,y_train)
pred1=lg.predict(x_test)
acc=accuracy_score(y_test,pred1)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred1))
print('Classification Report: ','\n',classification_report(y_test,pred1))
```

```
Accuracy Score:  0.9485503008021391
Confusion Matrix: 
 [[42844   213]
 [ 2250  2565]]
Classification Report: 
               precision    recall  f1-score   support

           0       0.95      1.00      0.97     43057
           1       0.92      0.53      0.68      4815

    accuracy                           0.95     47872
   macro avg       0.94      0.76      0.82     47872
weighted avg       0.95      0.95      0.94     47872
```

Decision Tree Calssifier

In [38]:
```
dtc.fit(x_train,y_train)
pred2=dtc.predict(x_test)
acc=accuracy_score(y_test,pred2)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred2))
print('Classification Report: ','\n',classification_report(y_test,pred2))
```

```
Accuracy Score:  0.9198696524064172
Confusion Matrix: 
 [[40706  2351]
 [ 1485  3330]]
Classification Report: 
               precision    recall  f1-score   support

           0       0.96      0.95      0.96     43057
           1       0.59      0.69      0.63      4815

    accuracy                           0.92     47872
   macro avg       0.78      0.82      0.79     47872
weighted avg       0.93      0.92      0.92     47872
```

Random Forest Classifier

In [40]:

```
rdc.fit(x_train,y_train)
pred4=rdc.predict(x_test)
acc=accuracy_score(y_test,pred4)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred4))
print('Classification Report: ','\n',classification_report(y_test,pred4))
```

```
Accuracy Score:  0.9454169451871658
Confusion Matrix: 
 [[42045  1012]
 [ 1601  3214]]
Classification Report: 
               precision    recall  f1-score   support

           0       0.96      0.98      0.97     43057
           1       0.76      0.67      0.71      4815

    accuracy                           0.95     47872
   macro avg       0.86      0.82      0.84     47872
weighted avg       0.94      0.95      0.94     47872
```

KNN Classifier ¶

In [36]:

```
knc.fit(x_train,y_train)
pred5=knc.predict(x_test)
acc=accuracy_score(y_test,pred5)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred5))
print('Classification Report: ','\n',classification_report(y_test,pred5))
```

```
Accuracy Score:  0.5836397058823529
Confusion Matrix: 
 [[24851 18206]
 [ 1726  3089]]
Classification Report: 
               precision    recall  f1-score   support

           0       0.94      0.58      0.71     43057
           1       0.15      0.64      0.24      4815

    accuracy                           0.58     47872
   macro avg       0.54      0.61      0.48     47872
weighted avg       0.86      0.58      0.67     47872
```

AdaBoost Classifier

In [37]:

```
ad.fit(x_train,y_train)
pred3=ad.predict(x_test)
acc=accuracy_score(y_test,pred3)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred3))
print('Classification Report: ','\n',classification_report(y_test,pred3))
```

```
Accuracy Score:  0.93829378342246
Confusion Matrix:
 [[42725   332]
 [ 2622  2193]]
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.99      0.97     43057
           1       0.87      0.46      0.60      4815

    accuracy                           0.94     47872
   macro avg       0.91      0.72      0.78     47872
weighted avg       0.93      0.94      0.93     47872
```

Gradient Boost Classifier

In [38]:

```
gb.fit(x_train,y_train)
pred6=gb.predict(x_test)
acc=accuracy_score(y_test,pred6)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred6))
print('Classification Report: ','\n',classification_report(y_test,pred6))
```

```
Accuracy Score:  0.934136864973262
Confusion Matrix:
 [[42943   114]
 [ 3039  1776]]
Classification Report:
               precision    recall  f1-score   support

           0       0.93      1.00      0.96     43057
           1       0.94      0.37      0.53      4815

    accuracy                           0.93     47872
   macro avg       0.94      0.68      0.75     47872
weighted avg       0.93      0.93      0.92     47872
```

Multinomial NB

In [41]:
```
mnb = MultinomialNB()
mnb.fit(x_train,y_train)
pred7=mnb.predict(x_test)
acc=accuracy_score(y_test,pred7)
print('Accuracy Score: ',acc)
print('Confusion Matrix: ' ,'\n',confusion_matrix(y_test,pred7))
print('Classification Report: ','\n',classification_report(y_test,pred7))
```

```
Accuracy Score:  0.9416986965240641
Confusion Matrix:
 [[42917   140]
 [ 2651  2164]]
Classification Report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97     43057
           1       0.94      0.45      0.61      4815

    accuracy                           0.94     47872
   macro avg       0.94      0.72      0.79     47872
weighted avg       0.94      0.94      0.93     47872
```

Hence, we are getting the best accuracy score through the Logistic Regression Model. We will go ahead with this to find the cross val score and hypermeter tuning.

Cross Val Score & Hypermeter Tuning

Cross-validation provides information about how well a classifier generalizes, specifically the range of expected errors of the classifier. Cross Val Score tells how the model is generalized at a particular cross validation.

At CV=6 we get the best results i.e. the Random Forest Classifier more generalized at cv=6, so we calculate the hyper parameters at this value.

We will find which parameters of random forest classifier are the best foe our model. We will do this using Grid Search CV method & also calculate the accuracy score at those best parameters.

Cross Val Score

In [44]:
```
from sklearn.model_selection import cross_val_score
for i in range(3,7):
    cr=cross_val_score(lg,x,y,cv=i)
    cr_mean=cr.mean()
    print("at cv= ", i)
    print('cross val score = ',cr_mean*100)
```

```
at cv=  3
cross val score =  94.641883514503
at cv=  4
cross val score =  94.70392461131532
at cv=  5
cross val score =  94.73901922948554
at cv=  6
cross val score =  94.74653894109105
```

Hypermeter Tuning

In [37]:
```
from sklearn.model_selection import GridSearchCV
# creating parameters
param={'penalty':['l1', 'l2', 'elasticnet', 'none'],
      'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}

GCV=GridSearchCV(lg,param,cv=6,scoring='accuracy')
GCV.fit(x_train,y_train)
GCV.best_params_
```

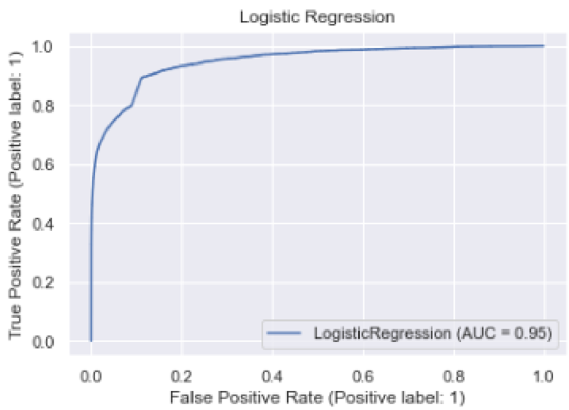
Out[37]:
```
{'penalty': 'l1', 'solver': 'liblinear'}
```

AUC ROC Curve

In our model AUC>0.5, so there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values.

AUC ROC Curve

In [39]:
```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(GCV.best_estimator_,x_test,y_test)
plt.title('Logistic Regression')
plt.show()
```



Our model accuracy is 95% which seems very good

Saving the Model

Saving the best model – Logistic Regression in this case for future predictions. Let's see what are the actual test data and what our model predicts.

Saving the Model

In [51]:
```
import pickle
filename='malignant.pkl'
pickle.dump(lg, open(filename,'wb'))
```

Conclusion

In [52]:
```
a=np.array(y_test)
pred=np.array(GCV_pred)
malignant=pd.DataFrame({'Actual':a,'Predicted':pred})
malignant
```

Out[52]:

	Actual	Predicted
0	0	1
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
47867	0	0
47868	0	0
47869	0	0
47870	0	0
47871	0	0

47872 rows × 2 columns

Hence up to some good extensions our model predicted so well.

Prediction

We save our model with 95% accuracy and now we can use it to predict whether the comment falls in the category of a malignant comment or not. If it comes to '0' that means it is not a malignant comment and if it is '1' then it is a malignant comment. But before that we have to clean the comment_text as we did in training part and convert it into vector for proper prediction.

Predicting for test dataset

In [43]:
```
# Replacing '\n' with ' '

df1.comment_text = df1.comment_text.str.replace('\n',' ')

from nltk.tokenize import regexp_tokenize

# Keeping only text with letters a to z, 0 to 9 and words like can't, don't, couldn't etc

df1.comment_text = df1.comment_text.apply(lambda x: ' '.join(regexp_tokenize(x,"[a-z']+")))
```

In [46]:
```
df1.head()
```

Out[46]:

	id	comment_text
0	00001cee341fdb12	o bitch a ule is more succesful then you'll ev...
1	0000247867823ef7	rom f he title is fine as it is
2	00013b17ad220c46	ources awe shton on apland
3	00017563c3f7919a	f you have a look back at the source the infor...
4	00017695ad8997eb	don't anonymously edit articles at all

In [47]:
```
test_data =tf_vec.fit_transform(df1['comment_text'])
test_data
```

Out[47]:
```
<153164x10000 sparse matrix of type '<class 'numpy.float64'>'
	with 2924094 stored elements in Compressed Sparse Row format>
```

In [53]:
```
#predicting using the saved model

loaded_model = pickle.load(open(filename, 'rb'))
pred=loaded_model.predict(test_data)
pred
```

Out[53]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [54]:
```
malignant_comment_prediction=pd.DataFrame(data=df1)
malignant_comment_prediction['Malignant or not']=pred
```

Final Output

In [55]:
```
malignant_comment_prediction
```

Out[55]:

	id	comment_text	Malignant or not
0	00001cee341fdb12	o bitch a ule is more succesful then you'll ev...	0
1	0000247867823ef7	rom f he title is fine as it is	0
2	00013b17ad220c46	ources awe shton on apland	0
3	00017563c3f7919a	f you have a look back at the source the infor...	0
4	00017695ad8997eb	don't anonymously edit articles at all	0
...	...	...	...
153159	fffcd0960ee309b5	i totally agree this stuff is nothing but too ...	0
153160	fffd7a9a6eb32c16	hrow from out field to home plate oes it get t...	0
153161	fffda9e8d6fafa9e	kinotorishima categories see your changes and ...	0
153162	fffe8f1340a79fc2	ne of the founding nations of the ermany has a...	0
153163	ffffce3fb183ee80	top already our bullshit is not welcome here '...	0

153164 rows × 3 columns

CONCLUSION

Conclusion of the Study

The results of this study suggest following outputs which might be useful to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyberbullying:

- Choice of word make a comment offensive, so model should be able to classify those words in the comment to recognize it to be offensive or not.
- Using such models, we can remove those offensive comments from online platforms before spreading them.

- ## Learning Outcomes of the Study in respect of Data Science
  - This projects teaches so many new things to me. I get to know new modules, new techniques to handle the dataset.
  - Data cleaning with new method.
  - New modules like wordcloud through which we get a better understanding of bad words.
  - How to tackle the six target variables and combine them into one for our model training and prediction?
  - Converting the comment into vector for proper training and machine understanding.