# MALIGNANT COMMENTS CLASSIFICATION

## Problem Statements

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.

## Data Set 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.

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

from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,rc
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,f1_s
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,GradientBoostin
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
test_df = pd.read_csv('test_df.csv')
train_df = pd.read_csv('train_df.csv')
```

In [3]:
```
test_df.head(10)
```

Out[3]:

| | id | comment_text |
|---|---|---|
| **0** | 00001cee341fdb12 | Yo bitch Ja Rule is more succesful then you'll... |
| **1** | 0000247867823ef7 | == From RfC == \n\n The title is fine as it is... |
| **2** | 00013b17ad220c46 | " \n\n == Sources == \n\n * Zawe Ashton on Lap... |
| **3** | 00017563c3f7919a | :If you have a look back at the source, the in... |
| **4** | 00017695ad8997eb | I don't anonymously edit articles at all. |
| **5** | 0001ea8717f6de06 | Thank you for understanding. I think very high... |
| **6** | 00024115d4cbde0f | Please do not add nonsense to Wikipedia. Such ... |
| **7** | 000247e83dcc1211 | :Dear god this site is horrible. |
| **8** | 00025358d4737918 | " \n Only a fool can believe in such numbers. ... |
| **9** | 00026d1092fe71cc | == Double Redirects == \n\n When fixing double... |

In [4]:
```
train_df.head(10)
```

Out[4]:

| | 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 |
| **5** | 00025465d4725e87 | "\n\nCongratulations from me as well, use the ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0002bcb3da6cb337 | COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK | 1 | 1 | 1 | 0 | 1 | 0 |
| **7** | 00031b1e95af7921 | Your vandalism to the Matt Shirvington article... | 0 | 0 | 0 | 0 | 0 | 0 |
| **8** | 00037261f536c51d | Sorry if the word 'nonsense' was offensive to ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 00040093b2687caa | alignment on this subject and which are contra... | 0 | 0 | 0 | 0 | 0 | 0 |

In [5]:
```
train_df.tail(10)
```

Out[5]:

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe |
|---|---|---|---|---|---|---|---|---|
| **159561** | ffd2e85b07b3c7e4 | "\nNo he did not, read it again (I would have ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159562** | ffd72e9766c09c97 | "\n Auto guides and the motoring press are not... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159563** | ffe029a7c79dc7fe | "\nplease identify what part of BLP applies be... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159564** | ffe897e7f7182c90 | Catalan independentism is the social movement ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159565** | ffe8b9316245be30 | The numbers in parentheses are the additional ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159566** | ffe987279560d7ff | ":::::And for the second time of asking, when ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159567** | ffea4adeee384e90 | You should be ashamed of yourself \n\nThat is ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159568** | ffee36eab5c267c9 | Spitzer \n\nUmm, theres no actual article for ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159569** | fff125370e4aaaf3 | And it looks like it was actually you who put ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **159570** | fff46fc426af1f9a | "\nAnd ... I really don't think you understand... | 0 | 0 | 0 | 0 | 0 | 0 |

In [6]:
```
#we can see all columns use to this fuction----> pd.set_option()
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

In [7]:
```
#Here the check how many row and columns
train_df.shape
```

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

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

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

In [9]:
```
#check the dataset information
train_df.info()
```

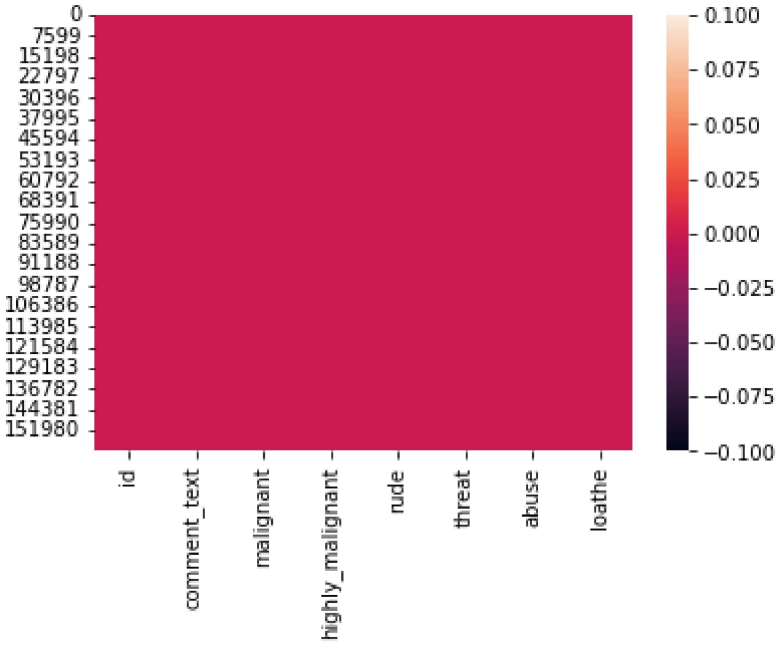
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159571 entries, 0 to 159570
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   id                159571 non-null  object
 1   comment_text      159571 non-null  object
 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
```

In [10]:
```
test_df.info()
```

```
<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
```

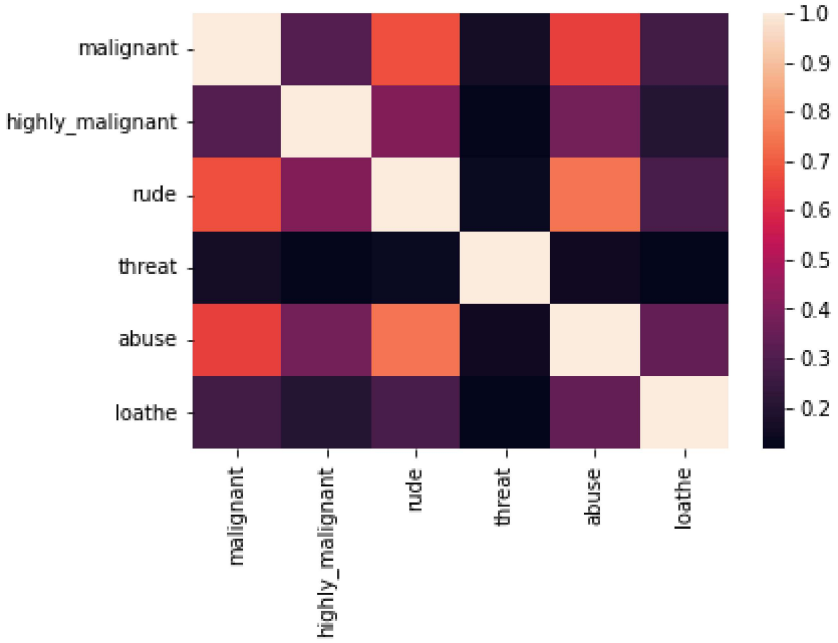
In [11]:
```
# checking null values
print(train_df.isnull().sum())
print(sns.heatmap(train_df.isnull()))
```

```
id                 0
comment_text       0
malignant          0
highly_malignant   0
rude               0
threat             0
abuse              0
loathe             0
dtype: int64
AxesSubplot(0.125,0.125;0.62x0.755)
```




In [12]:
```
## checking correlation in dataset
print(train_df.corr())
print(sns.heatmap(train_df.corr()))
```

```
                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

                 loathe
malignant      0.266009
highly_malignant  0.201600
rude           0.286867
threat         0.115128
abuse          0.337736
loathe         1.000000
AxesSubplot(0.125,0.125;0.62x0.755)
```

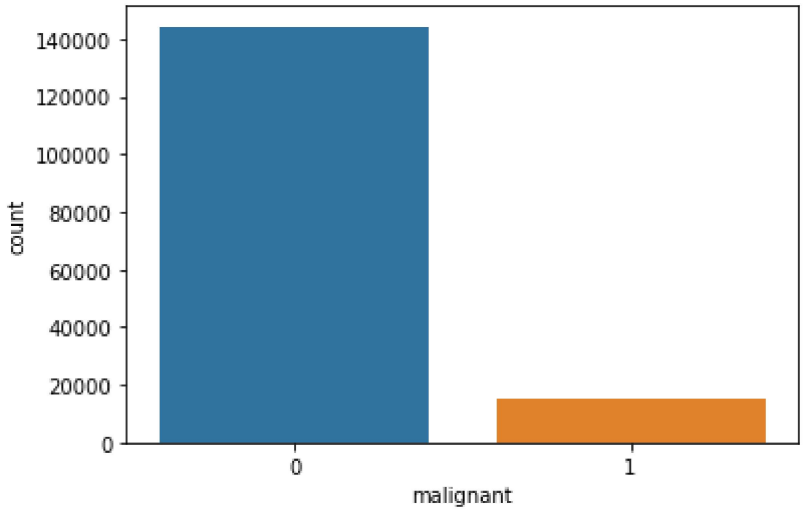



In [13]:
```
# checking the skewness for the features:
train_df.skew()
```

Out[13]:
```
malignant            2.745854
highly_malignant     9.851722
rude                 3.992817
threat              18.189001
abuse                4.160540
loathe              10.515923
dtype: float64
```

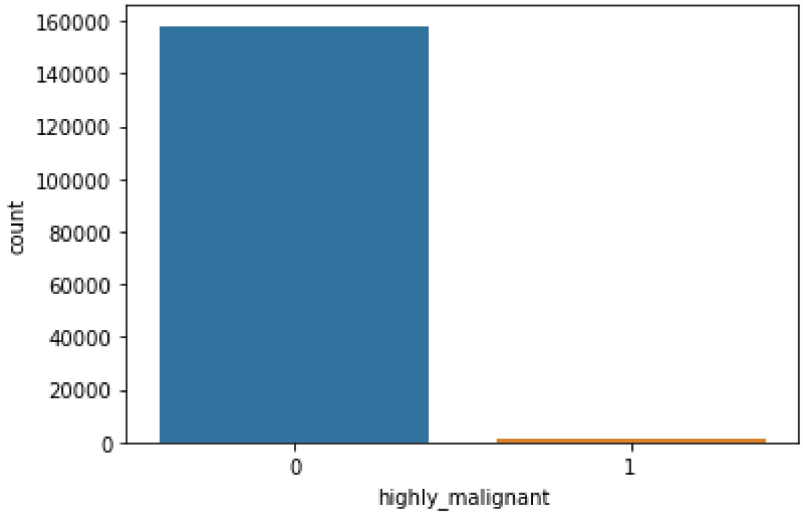
In [14]:
```
col=['malignant','highly_malignant','loathe','rude','abuse','threat']
for i in col:
    print(i)
    print("\n")
    print(train_df[i].value_counts())
    sns.countplot(train_df[i])
    plt.show()
```

```
malignant


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



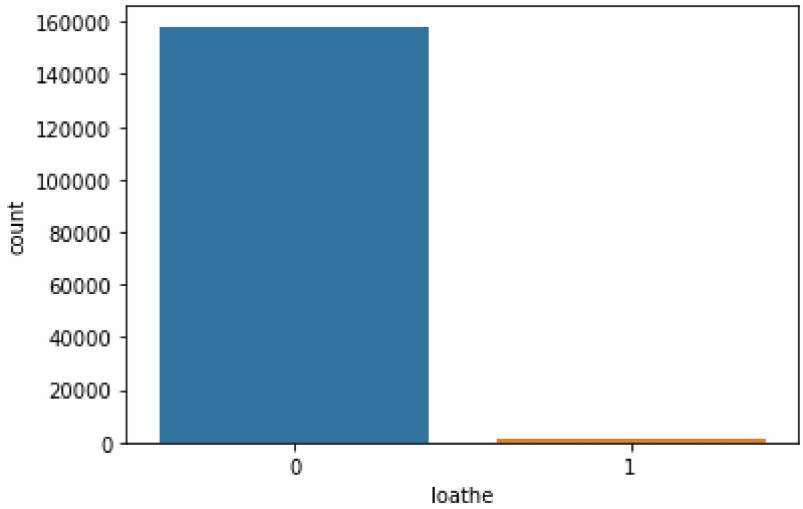

highly_malignant

```
0    157976
1      1595
Name: highly_malignant, dtype: int64
```



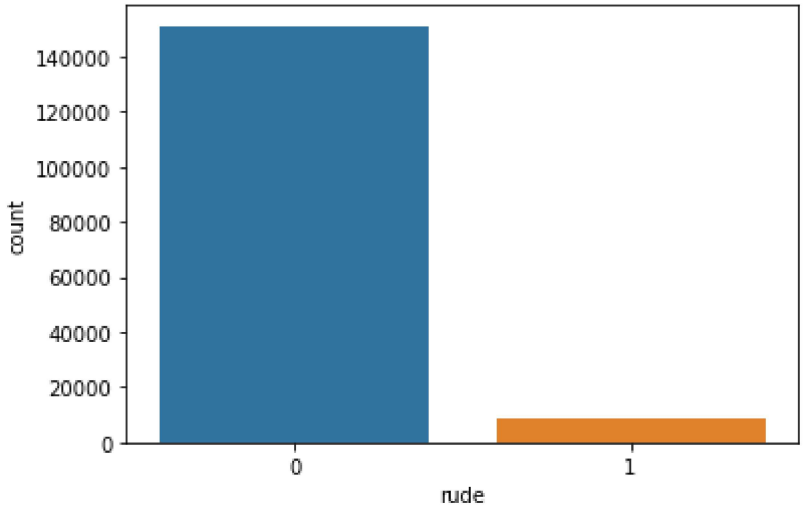

loathe

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



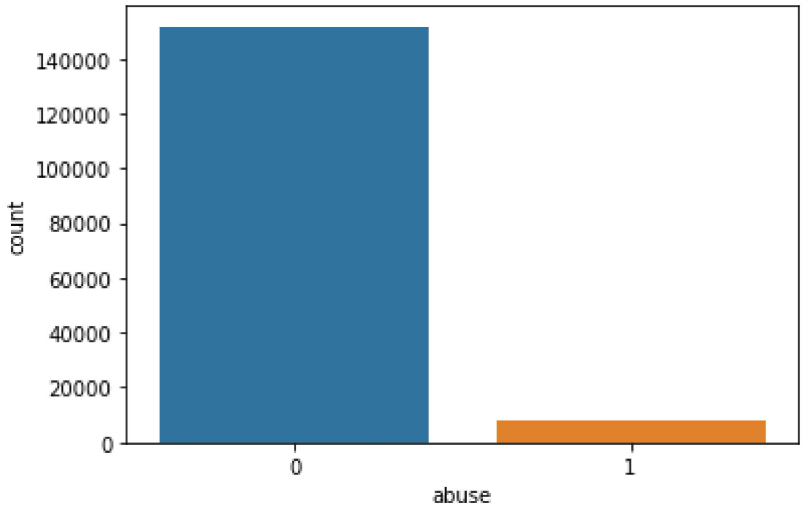

rude

```
0    151122
1      8449
Name: rude, dtype: int64
```

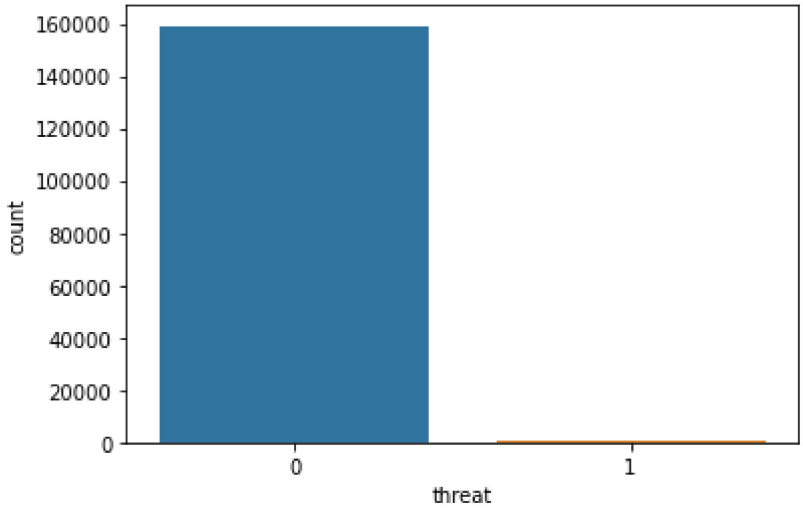



abuse

```
0    151694
1      7877
Name: abuse, dtype: int64
```




```
threat


0    159093
1       478
Name: threat, dtype: int64
```




In [15]:
```
from nltk.stem import WordNetLemmatizer
import nltk
from nltk.corpus import  stopwords
import string
```

In [16]:
```
train_df['length'] = train_df['comment_text'].str.len()
train_df.head(2)
```

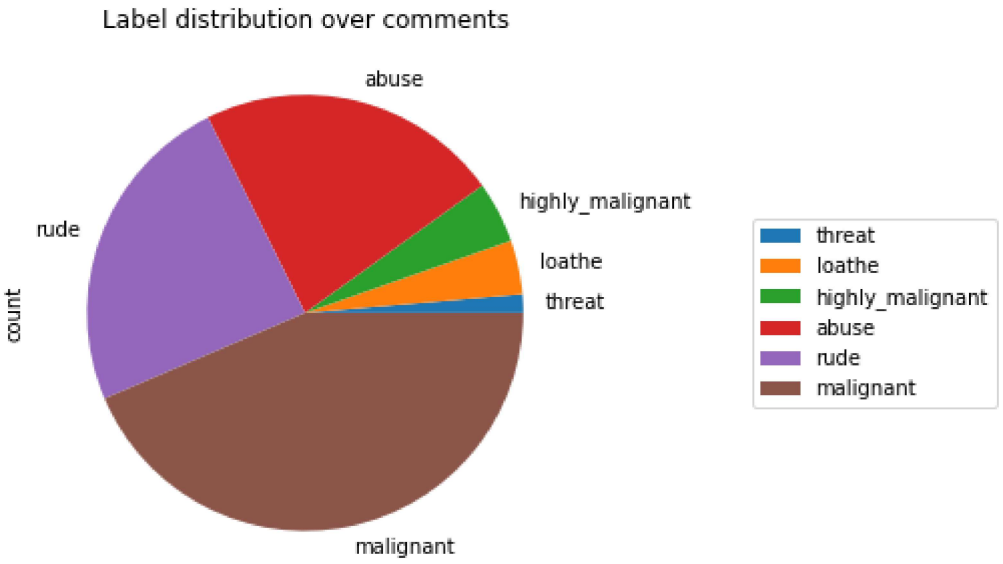
Out[16]:

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | le |
|---|---|---|---|---|---|---|---|---|---|
| **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 | |

In [17]:
```
# Convert all messages to lower case
train_df['comment_text'] = train_df['comment_text'].str.lower()

# Replace email addresses with 'email'
train_df['comment_text'] = train_df['comment_text'].str.replace(r'^.+@[^\.].*\.[a-z]{2
                                 'emailaddress')

# Replace URLs with 'webaddress'
train_df['comment_text'] = train_df['comment_text'].str.replace(r'^http\://[a-zA-Z0-9\
                                 'webaddress')

# Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
train_df['comment_text'] = train_df['comment_text'].str.replace(r'£|\$', 'dollers')

# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dash
train_df['comment_text'] = train_df['comment_text'].str.replace(r'^\(?[\d]{3}\)?[\s-]
                                 'phonenumber')
```

In [18]:
```
cols_target = ['malignant','highly_malignant','rude','threat','abuse','loathe']
df_distribution = train_df[cols_target].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))
```

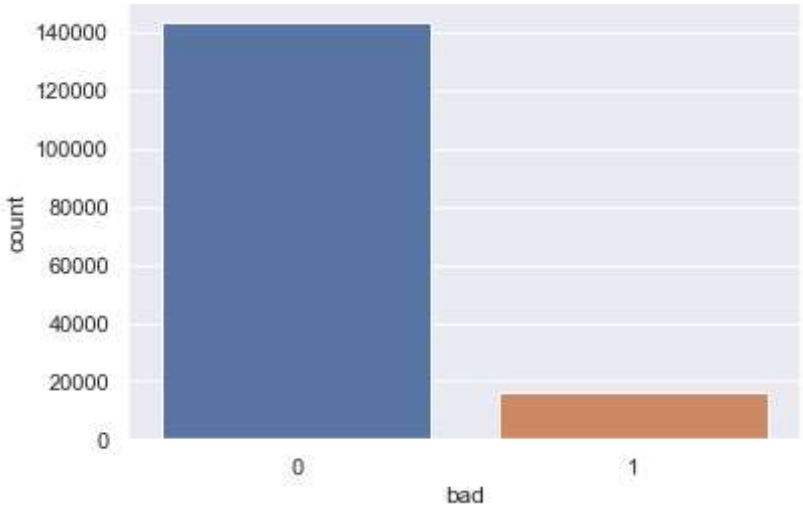
Out[18]:
```
<matplotlib.legend.Legend at 0x234c561f940>
```




In [19]:
```
target_data = train_df[cols_target]

train_df['bad'] =train_df[cols_target].sum(axis =1)
print(train_df['bad'].value_counts())
train_df['bad'] = train_df['bad'] > 0
train_df['bad'] = train_df['bad'].astype(int)
print(train_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 [20]:
```
sns.set()
sns.countplot(x="bad" , data = train_df)
plt.show()
```




In [21]:
```
#  Convert text into vectors using TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vec = TfidfVectorizer(max_features = 10000, stop_words='english')
features = tf_vec.fit_transform(train_df['comment_text'])
x = features
```

In [22]:
```
test_df.shape
```

Out[22]:
```
(153164, 2)
```

In [23]:
```
train_df.shape
```

Out[23]:
```
(159571, 10)
```

In [24]:
```
y=train_df['bad']
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=56,test_size=.30)
```

In [25]:
```
y_train.shape,y_test.shape
```

Out[25]:
```
((111699,), (47872,))
```

In [26]:
```
# LogisticRegression
LG = LogisticRegression(C=1, max_iter = 3000)
```

```
LG.fit(x_train, y_train)

y_pred_train = LG.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = LG.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 0.9595430576818056
Test accuracy is 0.9552347927807486
[[42731   219]
 [ 1924  2998]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.93      0.61      0.74      4922

    accuracy                           0.96     47872
   macro avg       0.94      0.80      0.86     47872
weighted avg       0.95      0.96      0.95     47872
```

In [27]:
```
# 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 0.999059973679263
Test accuracy is 0.9409675802139037
[[41627  1323]
 [ 1503  3419]]
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     42950
           1       0.72      0.69      0.71      4922

    accuracy                           0.94     47872
   macro avg       0.84      0.83      0.84     47872
weighted avg       0.94      0.94      0.94     47872
```

In [28]:
```
#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 0.9990331157843848
Test accuracy is 0.9564045788770054
[[42393   557]
 [ 1530  3392]]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     42950
           1       0.86      0.69      0.76      4922

    accuracy                           0.96     47872
   macro avg       0.91      0.84      0.87     47872
weighted avg       0.95      0.96      0.95     47872
```

In [29]:
```
# xgboost
import xgboost
xgb = xgboost.XGBClassifier()
xgb.fit(x_train, y_train)
y_pred_train = xgb.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = xgb.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 0.9610470997949847
Test accuracy is 0.9521014371657754
[[42692   258]
 [ 2035  2887]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     42950
           1       0.92      0.59      0.72      4922

    accuracy                           0.95     47872
   macro avg       0.94      0.79      0.84     47872
weighted avg       0.95      0.95      0.95     47872
```

In [30]:
```
#AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=100)
ada.fit(x_train, y_train)
y_pred_train = ada.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = ada.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 0.9507605260566343
Test accuracy is 0.9490307486631016
[[42557   393]
 [ 2047  2875]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     42950
           1       0.88      0.58      0.70      4922

    accuracy                           0.95     47872
   macro avg       0.92      0.79      0.84     47872
weighted avg       0.95      0.95      0.94     47872
```

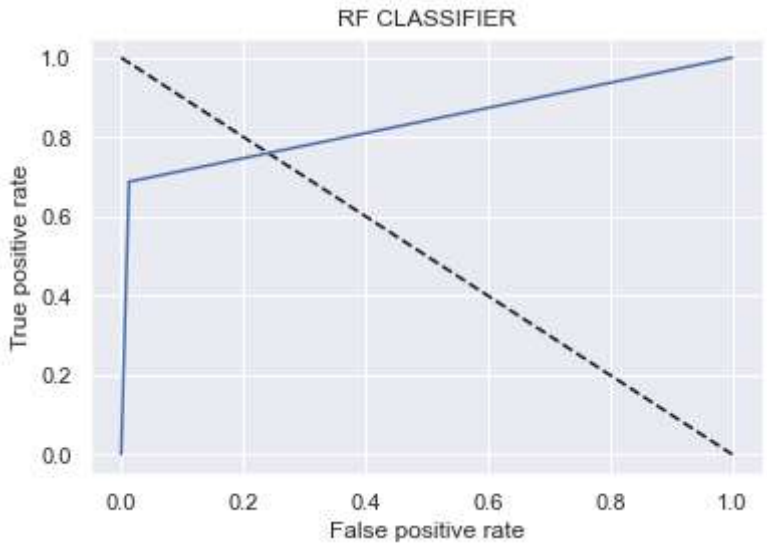
In [31]:
```
#KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
y_pred_train = knn.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = knn.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.919605367997923
Test accuracy is 0.915274064171123
[[42820   130]
 [ 3926   996]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.95     42950
           1       0.88      0.20      0.33      4922

    accuracy                           0.92     47872
   macro avg       0.90      0.60      0.64     47872
weighted avg       0.91      0.92      0.89     47872
```

In [35]:
```
# 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)))
cvs=cross_val_score(RF, x, y, cv=10, scoring='accuracy').mean()
print('cross validation score :',cvs*100)
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9990420684160108
Test accuracy is 0.9562165775401069
cross validation score : 95.68029214338827
[[42394   556]
 [ 1540  3382]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.86      0.69      0.76      4922

    accuracy                           0.96     47872
   macro avg       0.91      0.84      0.87     47872
weighted avg       0.95      0.96      0.95     47872
```

In [36]:
```
#Plotting the graph which tells us about the area under curve , more the area under cu
# 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()
```




In [41]:
```
import eli5
eli5.show_weights(RF,vec = tf_vec, top = 15)  #random forest
# will give you top 15 features or words  which makes a comment toxic
```

Out[41]:

| Weight | Feature |
|---|---|
| 0.0684 ± 0.0523 | fuck |
| 0.0400 ± 0.0440 | fucking |
| 0.0251 ± 0.0298 | shit |
| 0.0191 ± 0.0142 | stupid |
| 0.0186 ± 0.0111 | idiot |
| 0.0183 ± 0.0202 | ass |
| 0.0178 ± 0.0212 | bitch |
| 0.0173 ± 0.0178 | suck |
| 0.0152 ± 0.0141 | asshole |
| 0.0118 ± 0.0113 | dick |
| 0.0115 ± 0.0114 | cunt |
| 0.0102 ± 0.0093 | faggot |
| 0.0100 ± 0.0054 | gay |
| 0.0082 ± 0.0072 | hell |
| 0.0065 ± 0.0048 | bullshit |
| … 9985 more … | |

In [43]:
```
test_data =tf_vec.fit_transform(test_df['comment_text'])
test_data
```

Out[43]:
```
<153164x10000 sparse matrix of type '<class 'numpy.float64'>'
        with 2940344 stored elements in Compressed Sparse Row format>
```

In [45]:
```
prediction=RF.predict(test_data)
prediction
```

Out[45]:
```
array([0, 0, 0, ..., 0, 0, 1])
```

In [46]:
```
import joblib
joblib.dump(RF,"malig.pkl")
```

Out[46]:
```
['malig.pkl']
```

In [ ]:

In [ ]: