Problem Statement

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.

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.

Objective:

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

The steps we will go through are:

Data Pre-processing: pre-process the data to suit them with the analysis method.

The pre-processing may involve cleaning up the data, transforming the data, or creating new variables that may bring useful information for the analysis steps.

Exploratory Data Analysis (EDA): this step creates textual and visual summaries of the dataset that highlight some characteristics of the data.

Model Selection and Training, Test and Evaluate the Model: evaluate the performance of the proposed model

Data pre-processing

Getting the system ready and loading the data

We will be using Python for this course along with the below-listed libraries.

In [1]:
```
##Importing all necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
train=pd.read_csv('../input/malignant-comment-classification/train.csv')
train.head()
```

Out[2]:

	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

In [3]:

```
test=pd.read_csv('../input/malignant-comment-classification/test.csv')
test.head()
```

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.

In [4]:

```
print('train shape is ',train.shape)
print('test shape is ',test.shape)
print('test info',test.info)


print('train info',train.info)
```

```
train shape is  (159571, 8)
test shape is  (153164, 2)
test info <bound method DataFrame.info of                       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.
...                  ...                                                ...
153159  fffcd0960ee309b5  . \n i totally agree, this stuff is nothing bu...
153160  fffd7a9a6eb32c16  == Throw from out field to home plate. == \n\n...
153161  fffda9e8d6fafa9e  " \n\n == Okinotorishima categories == \n\n I ...
153162  fffe8f1340a79fc2  " \n\n == ""One of the founding nations of the...
153163  ffffce3fb183ee80  " \n :::Stop already. Your bullshit is not wel...

[153164 rows x 2 columns]>
train info <bound method DataFrame.info of                       id                                       comment_text  \
0       0000997932d777bf  Explanation\nWhy the edits made under my usern...
1       000103f0d9cfb60f  D'aww! He matches this background colour I'm s...
2       000113f07ec002fd  Hey man, I'm really not trying to edit war. It...
3       0001b41b1c6bb37e  "\nMore\nI can't make any real suggestions on ...
4       0001d958c54c6e35  You, sir, are my hero. Any chance you remember...
...                  ...                                                ...
159566  ffe987279560d7ff  ":::::And for the second time of asking, when ...
```

```
159567  ffea4adeee384e90  You should be ashamed of yourself \n\nThat is ...
159568  ffee36eab5c267c9  Spitzer \n\nUmm, theres no actual article for ...
159569  fff125370e4aaaf3  And it looks like it was actually you who put ...
159570  fff46fc426af1f9a  "\nAnd ... I really don't think you understand...

        malignant  highly_malignant  rude  threat  abuse  loathe
0               0                 0     0       0      0       0
1               0                 0     0       0      0       0
2               0                 0     0       0      0       0
3               0                 0     0       0      0       0
4               0                 0     0       0      0       0
...           ...               ...   ...     ...    ...     ...
159566          0                 0     0       0      0       0
159567          0                 0     0       0      0       0
159568          0                 0     0       0      0       0
159569          0                 0     0       0      0       0
159570          0                 0     0       0      0       0

[159571 rows x 8 columns]>
```

In [5]:

```python
print('train data Set descriptin',train.describe())
print('test data Set descriptin',test.describe())
```

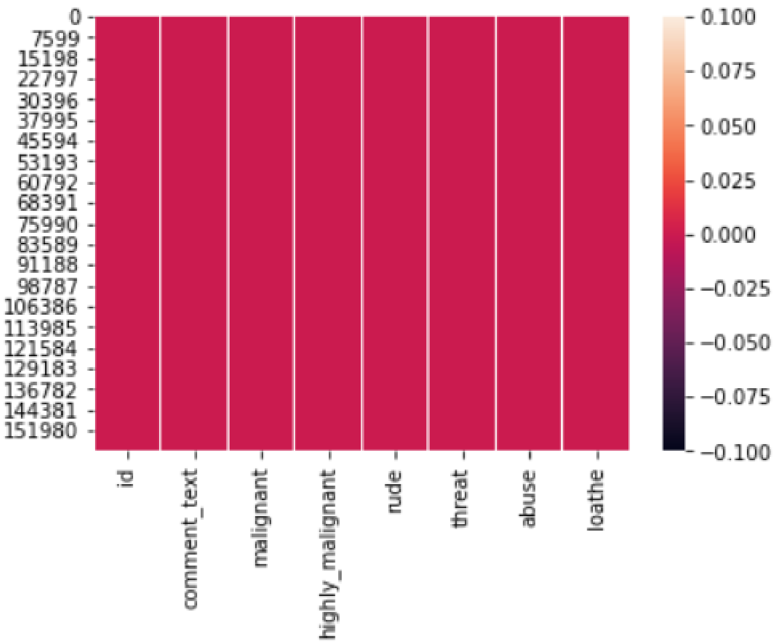
```
train data Set descriptin            malignant  highly_malignant          rude          thre
at  \
count  159571.000000     159571.000000  159571.000000  159571.000000
mean        0.095844          0.009996       0.052948       0.002996
std         0.294379          0.099477       0.223931       0.054650
min         0.000000          0.000000       0.000000       0.000000
25%         0.000000          0.000000       0.000000       0.000000
50%         0.000000          0.000000       0.000000       0.000000
75%         0.000000          0.000000       0.000000       0.000000
max         1.000000          1.000000       1.000000       1.000000

               abuse         loathe
count  159571.000000  159571.000000
mean        0.049364       0.008805
std         0.216627       0.093420
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.000000       0.000000
max         1.000000       1.000000
test data Set descriptin                       id                                       comm
ent_text
count                153164                                            153164
unique               153164                                            153164
top      821fdc095707837b  Sockpuppet== \n See Wikipedia:Requests for com...
freq                      1                                                 1
```

Data Visualization:

In [6]:

```python
# checking null values
print(train.isnull().sum())
print(sns.heatmap(train.isnull()))
```

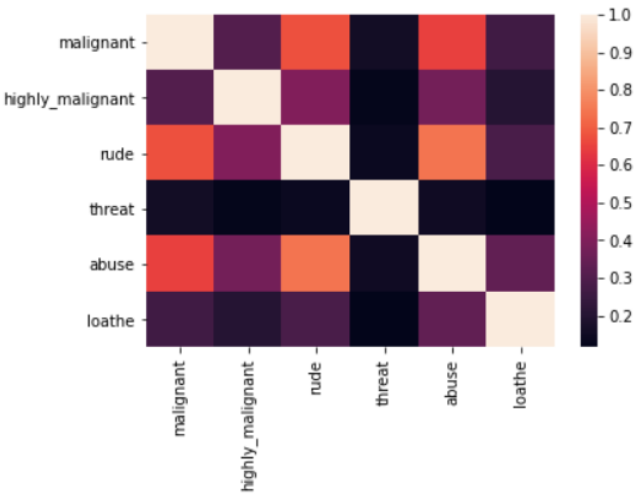
```
id                  0
comment_text        0
malignant           0
highly_malignant    0
rude                0
threat              0
abuse               0
loathe              0
dtype: int64
AxesSubplot(0.125,0.125;0.62x0.755)
```



In [7]:

```
## checking correlation in dataset
print(train.corr())
print(sns.heatmap(train.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 [8]:

```
# checking the skewness for the features:
train.skew()
```

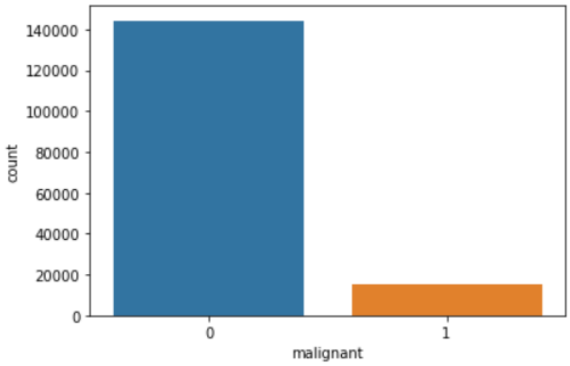
Out[8]:

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

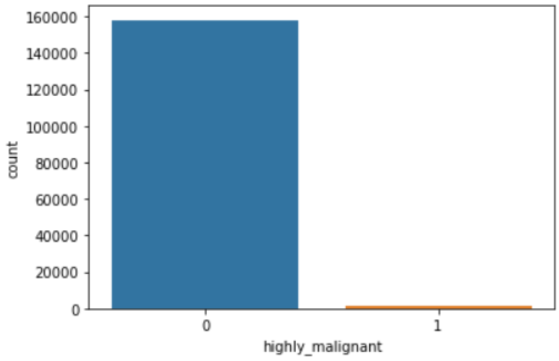
In [9]:

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

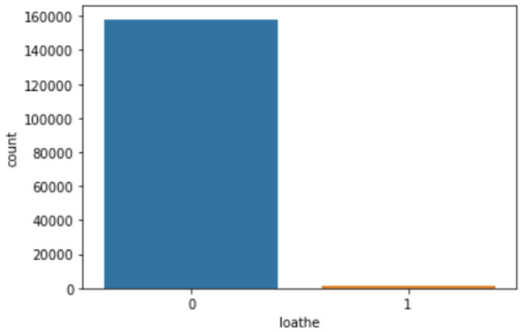
```
malignant


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



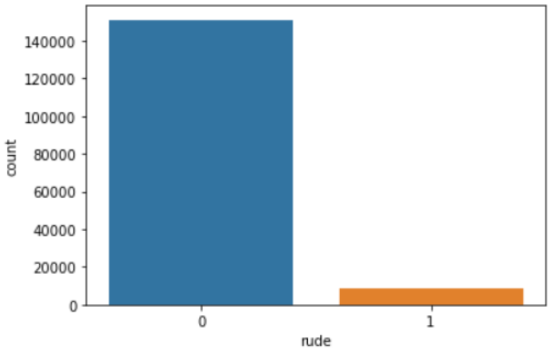
```
highly_malignant


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



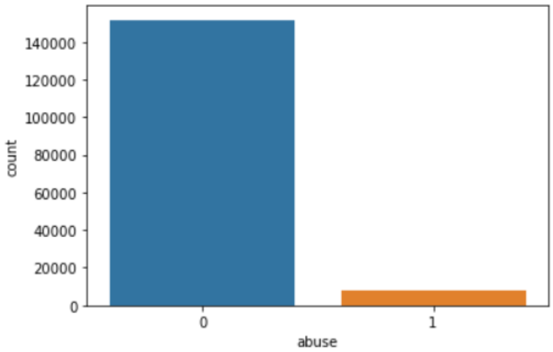

```
loathe


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



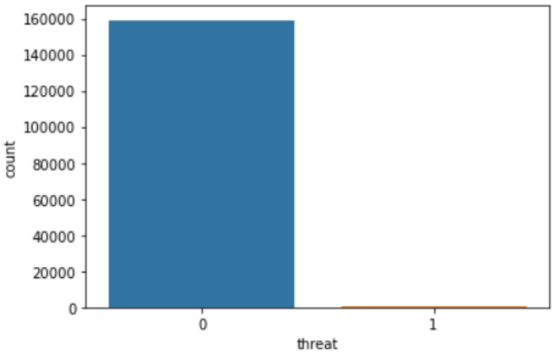
```
rude


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 [10]:

```
from nltk.stem import WordNetLemmatizer
import nltk
from nltk.corpus import  stopwords
import string
```

In [11]:

```
train['length'] = train['comment_text'].str.len()
train.head(2)
```

Out[11]:

	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

```python
# Convert all messages to lower case
train['comment_text'] = train['comment_text'].str.lower()

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

# Replace URLs with 'webaddress'
train['comment_text'] = 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)
train['comment_text'] = train['comment_text'].str.replace(r'£|\$', 'dollers')

# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'p
honenumber'
train['comment_text'] = train['comment_text'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?
[\d]{4}$',
                                  'phonenumber')


# Replace numbers with 'numbr'
train['comment_text'] = train['comment_text'].str.replace(r'\d+(\.\d+)?', 'numbr')


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

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

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

In [ ]:

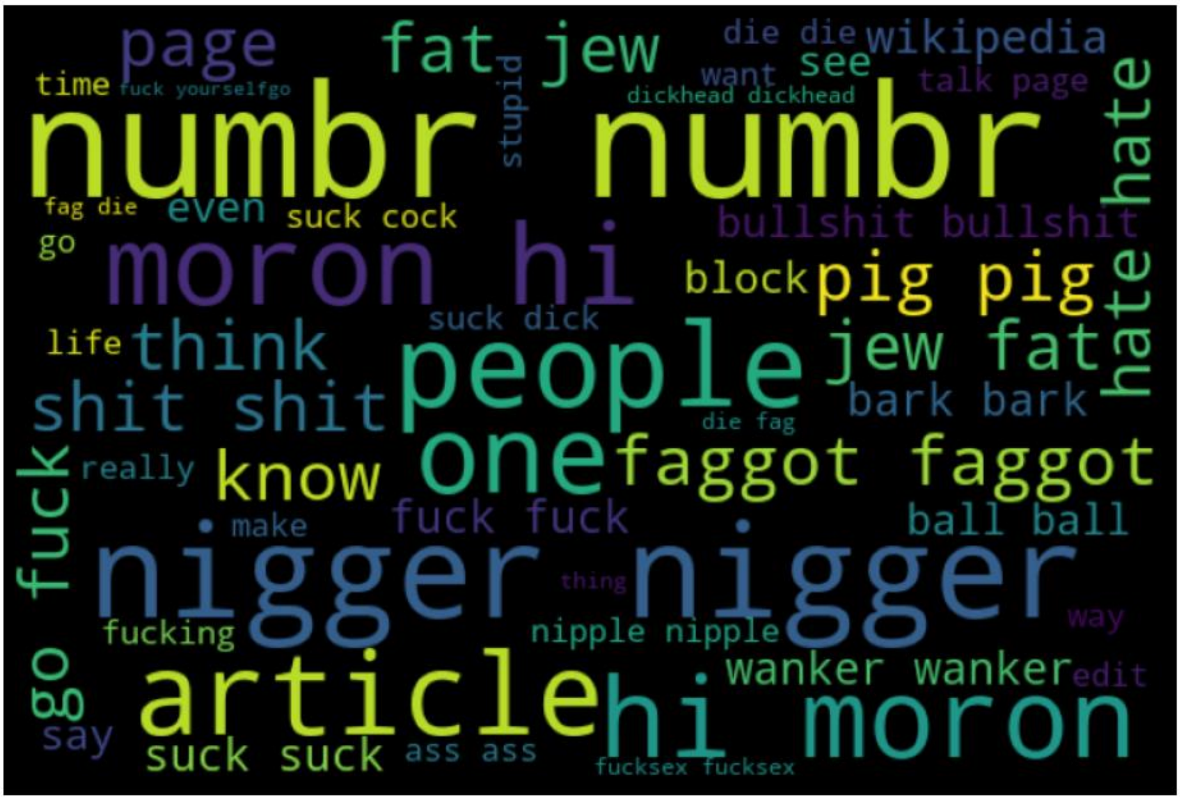
```
train['clean_length'] = train.comment_text.str.len()
train.head()
```

In [ ]:

```
# Total length removal
print ('Origian Length', train.length.sum())
print ('Clean Length', train.clean_length.sum())
```

In [15]:

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



Model Building

The modelling process consists in selecting models that are based on various machine learning techniques used in the experimentation. In this case various predictive models were used such as those based on decision tree, Random forest, logistic regression and AdaBoostClassifier. The goal is to identify the best classifier for the analysed problem. Each classifier must therefore be trained on the featured set and the classifier with the best classification results is used for prediction. The classification algorithms taken into consideration are: • Logistic Regression classifier, • Decision tree classifier, • Random forest classifier, • AdaBoostClassifier.

In this step, we will start modifying model parameters, perform feature engineering and balancing data strategies to improve the performance of the models. Try with more trees in the Random Forest model, include new variables, penalize wrong predictions from the minority class until you beat the performance of our current best model.

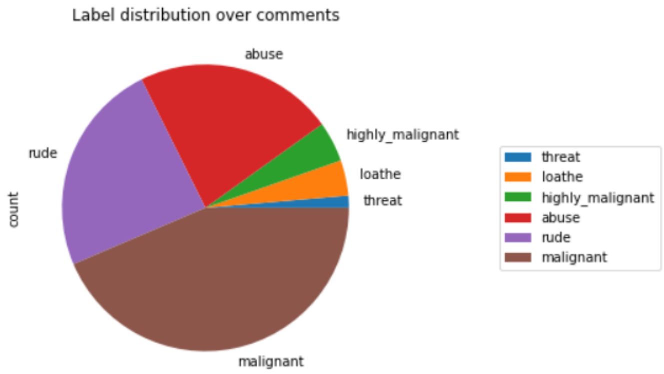
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,roc_curve,r
oc_auc_score,auc
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,f1_score
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,GradientBoostingClassifi
er
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

In [17]:

```
cols_target = ['malignant','highly_malignant','rude','threat','abuse','loathe']
df_distribution = train[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[17]:

```
<matplotlib.legend.Legend at 0x7fd6b3783a50>
```



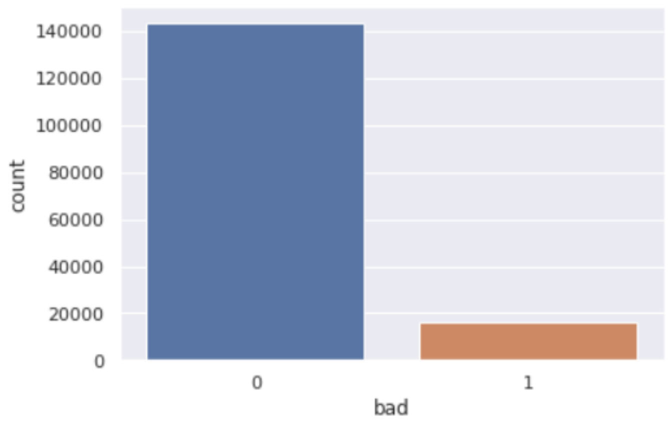
In [18]:

```
target_data = train[cols_target]

train['bad'] =train[cols_target].sum(axis =1)
print(train['bad'].value_counts())
train['bad'] = train['bad'] > 0
train['bad'] = train['bad'].astype(int)
print(train['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 [19]:

```
sns.set()
sns.countplot(x="bad" , data = train)
plt.show()
```



To begin, let's split the dataset into training and test sets using 70/30 split; 70% of data will be used to train the model and the rest 30% to test the accuracy of the model. Then we can up sample the minority class, in this case the positive class.

```
In [26]: # 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['comment_text'])
         x = features

In [27]: train.shape

Out[27]: (159571, 11)

In [28]: test.shape

Out[28]: (153164, 4)

In [29]: y=train['bad']
         x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=56,test_size=.30)

In [30]: y_train.shape,y_test.shape

Out[30]: ((111699,), (47872,))
```

In [31]:
```
# 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 0.9595072471553013
Test accuracy is 0.9552556818181818
[[42729   221]
 [ 1921  3001]]
              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 [32]:
```
# 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 0.9988988263099938
Test accuracy is 0.9395053475935828
[[41587  1363]
 [ 1533  3389]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     42950
           1       0.71      0.69      0.70      4922

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

[33]:
```
#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))
```

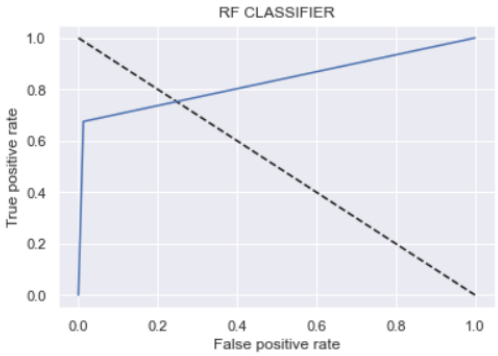
```
Training accuracy is 0.9513603523755808
Test accuracy is 0.9494694184491979
[[42551   399]
 [ 2020  2902]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     42950
           1       0.88      0.59      0.71      4922

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

```
In [34]: #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 0.9988809210467416
         Test accuracy is 0.9553392379679144
         [[42412   538]
          [ 1600  3322]]
                       precision    recall  f1-score   support

                    0       0.96      0.99      0.98     42950
                    1       0.86      0.67      0.76      4922

             accuracy                           0.96     47872
            macro avg       0.91      0.83      0.87     47872
         weighted avg       0.95      0.96      0.95     47872
```

```
In [35]: #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()
```

Model validation

Finally, after testing our models with the test set, we concluded that best model was the Random Forest (RF). Now we will Hyper tune our model with the help of GridSearchCV to increase our model accuracy.




```
In [36]: auc_score=roc_auc_score(y_test,RF.predict(x_test))
```

```
In [37]: print(auc_score)

         0.8312013487234384
```

```
In [38]: from sklearn.model_selection import GridSearchCV
```

```
In [39]: #Creating parameter list to pass in GridSearch

         parameters={'max_depth':np.arange(2,15),'criterion':['gini','entropy']}
```

```
In [40]: GCV=GridSearchCV(RandomForestClassifier(),parameters,cv=5)
```

In [41]:
```
GCV.fit(x_train,y_train)
```
Out[41]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])})
```
In [42]:
```
GCV.best_params_ #Printing the best parameter found by GridSearch
```
Out[42]:
```
{'criterion': 'gini', 'max_depth': 14}
```
In [43]:
```
GCV_pred=GCV.best_estimator_.predict(x_test)
```
In [44]:
```
accuracy_score(y_test,GCV_pred)
```
Out[44]:
```
0.9000250668449198
```
In [45]:
```
!pip install eli5
```
```
Requirement already satisfied: eli5 in c:\programdata\anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from eli5) (1.5.0)
Requirement already satisfied: jinja2 in c:\programdata\anaconda3\lib\site-packages (from eli5) (2.11.2)
Requirement already satisfied: numpy>=1.9.0 in c:\programdata\anaconda3\lib\site-packages (from eli5) (1.18.5)
Requirement already satisfied: graphviz in c:\programdata\anaconda3\lib\site-packages (from eli5) (0.17)
Requirement already satisfied: scikit-learn>=0.20 in c:\programdata\anaconda3\lib\site-packages (from eli5) (0.23.1)
Requirement already satisfied: tabulate>=0.7.7 in c:\programdata\anaconda3\lib\site-packages (from eli5) (0.8.9)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from eli5) (1.15.0)
Requirement already satisfied: attrs>16.0.0 in c:\programdata\anaconda3\lib\site-packages (from eli5) (19.3.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\programdata\anaconda3\lib\site-packages (from jinja2->eli5) (1.1.1)
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.20->eli5) (0.1
6.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.20->eli
5) (2.1.0)
```
In [46]:
```
import eli5
eli5.show_weights(RF,vec = tf_vec, top = 20)  #random forest
# will give you top 15 features or words  which makes a comment toxic
```
Out[46]:

Weight	Feature
0.0777 ± 0.0571	fuck
0.0436 ± 0.0461	fucking
0.0267 ± 0.0270	shit
0.0206 ± 0.0186	suck
0.0199 ± 0.0118	idiot
0.0194 ± 0.0196	bitch
0.0186 ± 0.0160	stupid
0.0171 ± 0.0140	asshole
0.0111 ± 0.0121	cunt
0.0110 ± 0.0099	dick
0.0108 ± 0.0102	faggot
0.0101 ± 0.0060	gay
0.0081 ± 0.0074	hell
0.0070 ± 0.0080	ass
0.0067 ± 0.0051	bullshit
0.0063 ± 0.0067	bastard
0.0061 ± 0.0087	cock
0.0058 ± 0.0060	loser
0.0058 ± 0.0044	moron
0.0056 ± 0.0037	hate
… 9980 more …	

In [47]:
```
test_data =tf_vec.fit_transform(test['comment_text'])
test_data
```
Out[47]:
```
<153164x10000 sparse matrix of type '<class 'numpy.float64'>'
	with 2839239 stored elements in Compressed Sparse Row format>
```
In [52]:
```
import joblib
joblib.dump(RF,"Malignant_Comment2.csv.obj")
```
Out[52]:
```
['Malignant_Comment2.csv.obj']
```
In [53]:
```
p=joblib.load("Malignant_Comment2.csv.obj")
```

In [55]:

```
import numpy as np
a=np.array(y_test)
predicted=np.array(RF.predict(x_test))
test_data=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
```

In [56]:

```
test_data
```

Out[56]:

	original	predicted
0	0	0
1	1	1
2	0	0
3	1	1
4	0	0
...	...	...
47867	0	0
47868	0	0
47869	0	0
47870	0	0
47871	0	0

47872 rows × 2 columns