FLIP ROBO

MALIGNANT COMMENTS CLASSIFIER PROJECT

Submitted by:

Madhurima Srivastava

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Flip Robo Technologies, Bangalore for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I want to thank my SME Mr. Mohd Kashif for providing the Dataset and helping us to solve the problem and addressing out our Query in right time. I would like to express my gratitude towards my parents & members of Flip Robo for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to our institute DataTrained & others seen unseen hands which have given us direct & indirect help in completion of this project. With help of their brilliant guidance and encouragement, I was able to complete my tasks properly and were up to the mark in all the tasks assigned.

During the process, I got a chance to see the stronger side of my technical and non-technical aspects and also strengthen my concepts.

INTRODUCTION

Problem Statement

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.

Business Problem Framing

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.

Multilabel vs Multiclass classification

In multi-class classification, the data can belong to only one label out of all the labels we have. For example, a given picture of an animal may be a cat, dog or elephant only and not a combination of these.

In multi-label classification, data can belong to more than one label simultaneously. For example, in our case a comment may be malignant, threat or loathe at the same time. It may also happen that the comment is positive/neutral and hence does not belong to any of the six labels.

Conceptual Background of the Domain Problem

In the past few years, it's seen that the cases related to social media hatred have increased exponentially. Social media is turning into a dark venomous pit for people nowadays. Online hate is the result of difference in opinion, race, religion, occupation, nationality etc. In social media the people spreading or involved in such kind of activities use filthy languages, aggression, images etc. to offend and gravely hurt the person on the other side. This is one of the major concerns now.

Review of Literature

Sentiment classification regarding toxicity has been intensively researched in the past few years, largely in the context of social media data where researchers have applied various machine learning systems to try and tackle the problem of toxicity as well as the related, more well-known task of sentiment analysis. Comment abuse classification research begins with combining of TF-IDF with sentiment/contextual features. The motivation for our project is to build a model that can detect toxic comments and find the bias with respect to the mention of select identities.

Motivation for the Problem Undertaken

The upsurge in the volume of unwanted comments called malignant comments has created an intense need for the development of more dependable and robust malignant comments filters. Machine learning methods of recent are being used to successfully detect and filter malignant comments. Build a model which can be used to predict in terms of a probability for comments to be malignant. In this case, Label ‘1’ indicates that the comment is malignant, while, Label ‘0’ indicates that the comment is not malignant.

ANALYTICAL PROBLEM FRAMING

Model Building Phase

You need to build a machine learning model. Before model building do all data pre-processing steps involving NLP. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science.

Include all the steps like

1. Data Cleaning

2. Exploratory Data Analysis

3. Data Pre-processing

4. Model Building

5. Model Evaluation

6. Selecting the best model

Data Sources and their formats

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.

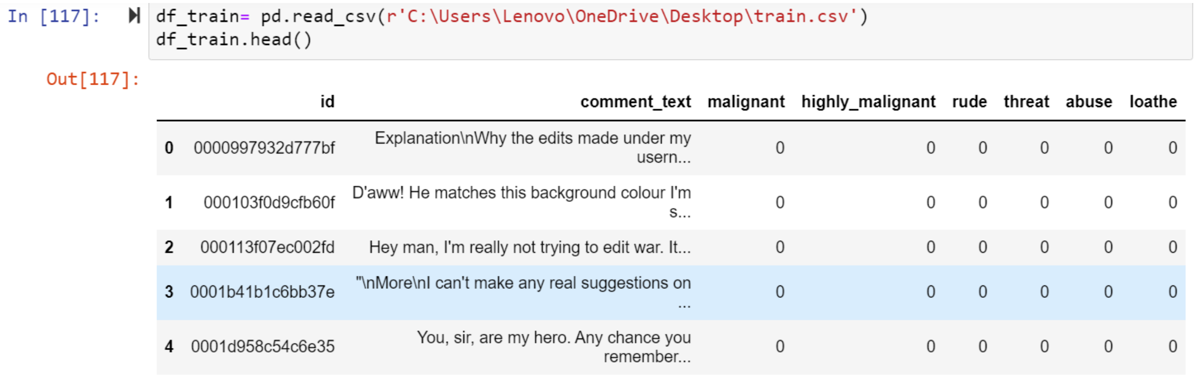
- ID: It includes unique Ids associated with each comment text given.

- Comment text: This column contains the comments extracted from various social media platforms.

Both train and test csv(s) are loaded respectively, where, in training dataset, the independent variable is Comment text which is of ‘object’type and rest 6 categories or labels are the dependent features whose values needs to be predicted, are of Boolean in nature being ‘int64’ type.

The sample datasets are given below:

Train dataset.

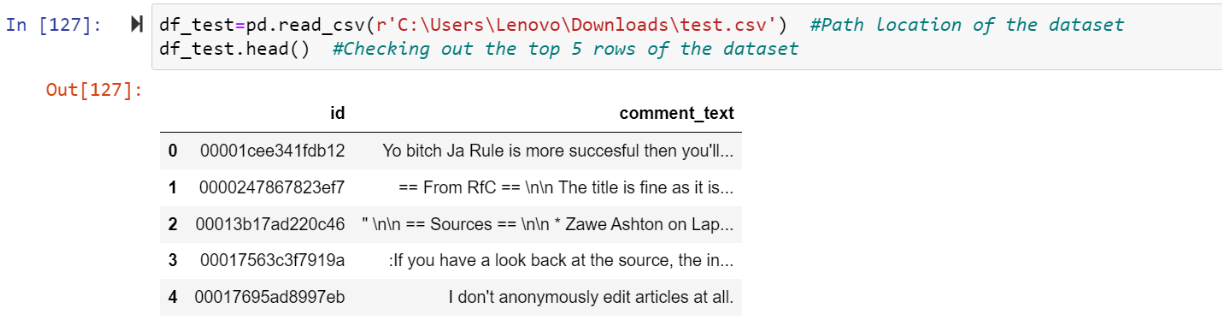


```
In [117]:   df_train= pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\train.csv')
            df_train.head()
```

Out[117]:

	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

Test dataset



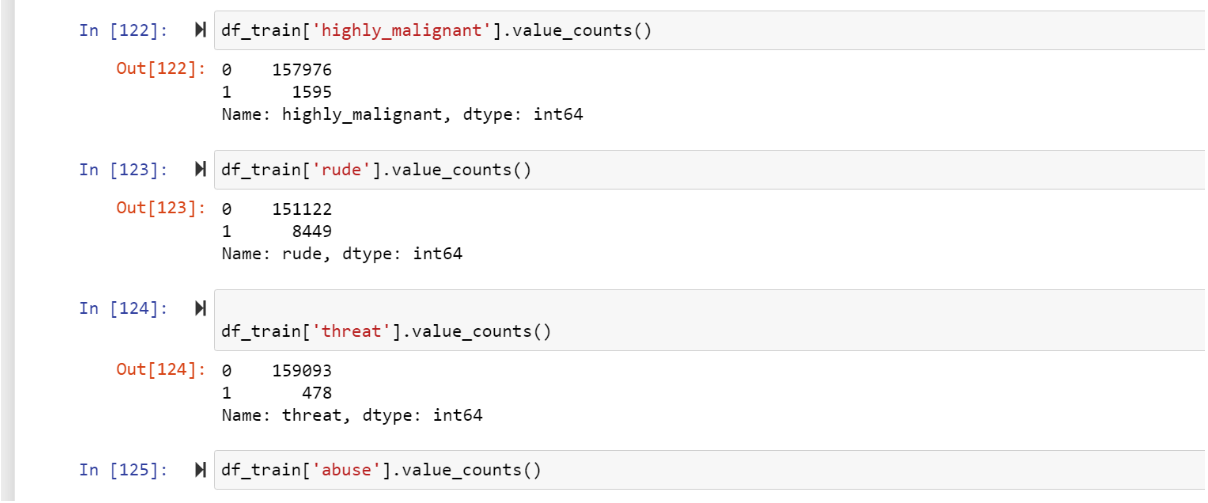
```
In [127]:   df_test=pd.read_csv(r'C:\Users\Lenovo\Downloads\test.csv')  #Path Location of the dataset
            df_test.head()  #Checking out the top 5 rows of the dataset
```

Out[127]:

	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.

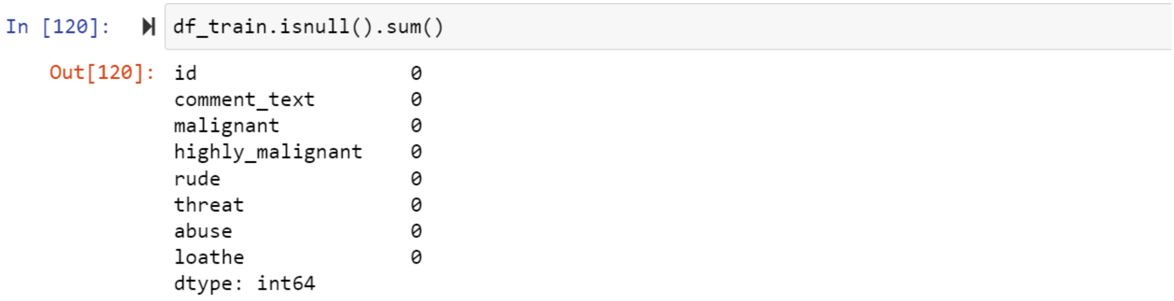
Data Pre-processing

Checking the value counts of the features



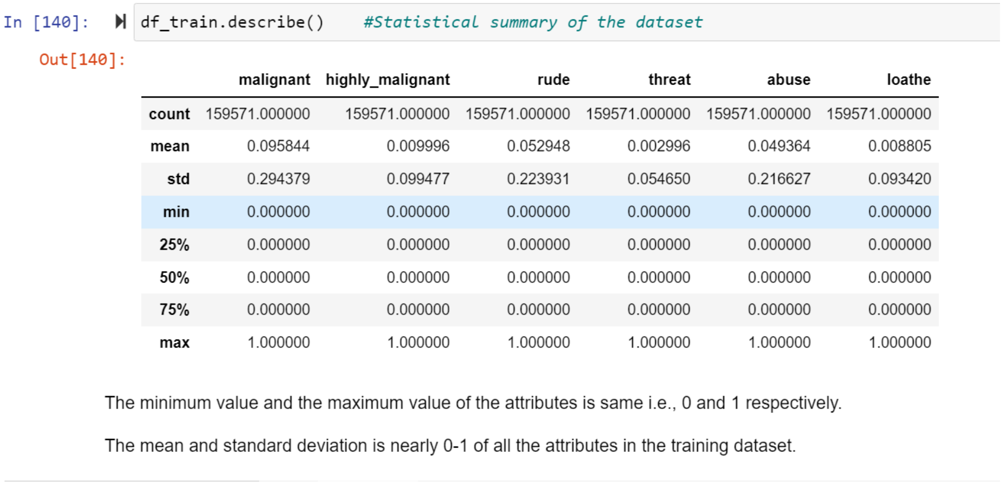
```
In [122]:  df_train['highly_malignant'].value_counts()

Out[122]:  0    157976
           1      1595
           Name: highly_malignant, dtype: int64

In [123]:  df_train['rude'].value_counts()

Out[123]:  0    151122
           1      8449
           Name: rude, dtype: int64

In [124]:
           df_train['threat'].value_counts()

Out[124]:  0    159093
           1       478
           Name: threat, dtype: int64

In [125]:  df_train['abuse'].value_counts()
```

Checking for null values



```
In [120]:  df_train.isnull().sum()

Out[120]:  id                  0
           comment_text        0
           malignant           0
           highly_malignant    0
           rude                0
           threat              0
           abuse               0
           loathe              0
           dtype: int64
```

Checking statistical summary of the dataset



```
In [140]:  df_train.describe()     #Statistical summary of the dataset

Out[140]:
```

	malignant	highly_malignant	rude	threat	abuse	loathe
count	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
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

The minimum value and the maximum value of the attributes is same i.e., 0 and 1 respectively.

The mean and standard deviation is nearly 0-1 of all the attributes in the training dataset.

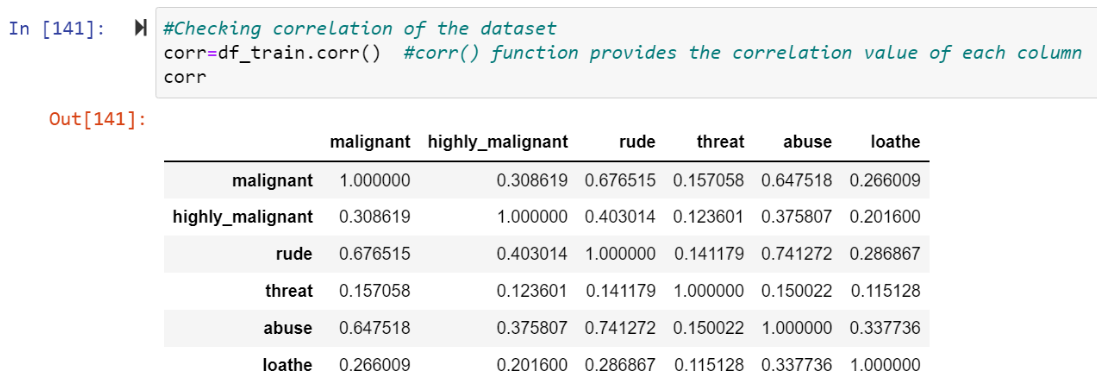
- The minimum value and the maximum value of the attributes is same i.e., 0 and 1 respectively.

- The mean and standard deviation is nearly 0-1 of all the attributes in the training dataset.

- Here, with this statistical analysis, it is interpreted that there are no outliers as well as skewness present in this training dataset.

- The count of each field is equal which shows that there are no missing values present.

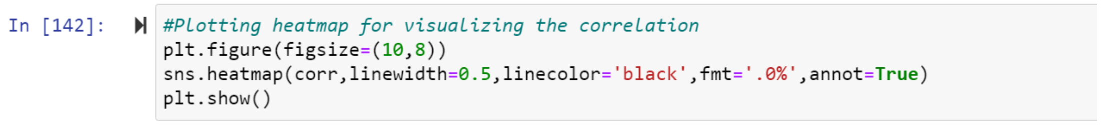
Finding relationship among data using correlation.



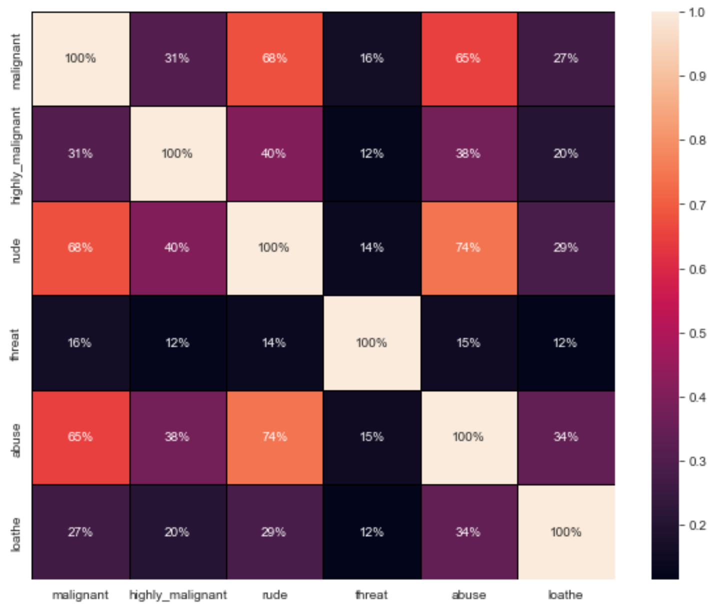
```
In [141]:  #Checking correlation of the dataset
           corr=df_train.corr()  #corr() function provides the correlation value of each column
           corr
```

Out[141]:

	malignant	highly_malignant	rude	threat	abuse	loathe
malignant	1.000000	0.308619	0.676515	0.157058	0.647518	0.266009
highly_malignant	0.308619	1.000000	0.403014	0.123601	0.375807	0.201600
rude	0.676515	0.403014	1.000000	0.141179	0.741272	0.286867
threat	0.157058	0.123601	0.141179	1.000000	0.150022	0.115128
abuse	0.647518	0.375807	0.741272	0.150022	1.000000	0.337736
loathe	0.266009	0.201600	0.286867	0.115128	0.337736	1.000000



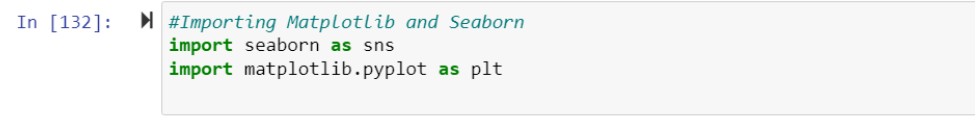
```
In [142]:  #Plotting heatmap for visualizing the correlation
           plt.figure(figsize=(10,8))
           sns.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',annot=True)
           plt.show()
```



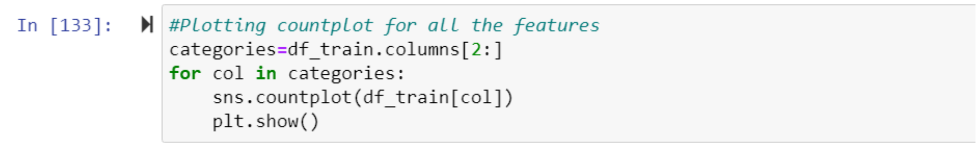
DATA VISUALIZATION

The use of tables, graphs, and charts play a vital role in presenting the data being used to draw these conclusions. Thus, data visualization is the best way to explore the data as it allows in-depth analysis.

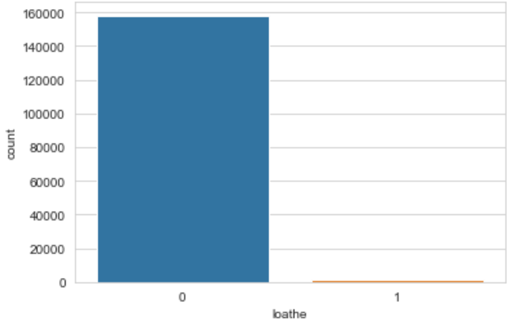
Plotting countplot for all features

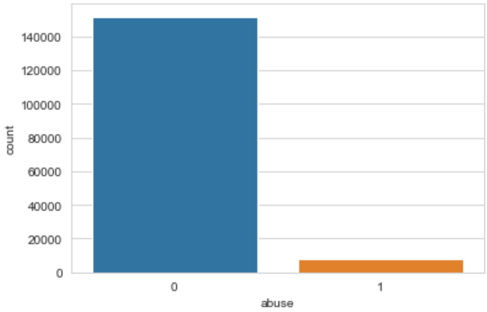


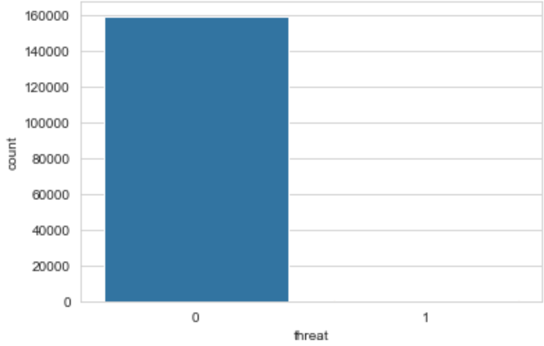
```
In [132]:  #Importing Matplotlib and Seaborn
            import seaborn as sns
            import matplotlib.pyplot as plt
```

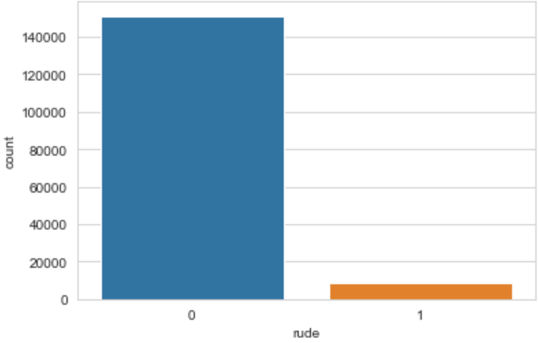


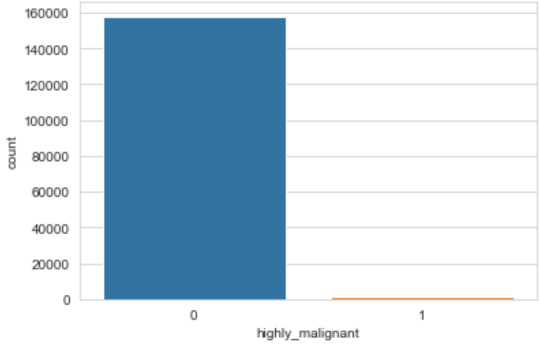
```
In [133]:  #Plotting countplot for all the features
            categories=df_train.columns[2:]
            for col in categories:
                sns.countplot(df_train[col])
                plt.show()
```

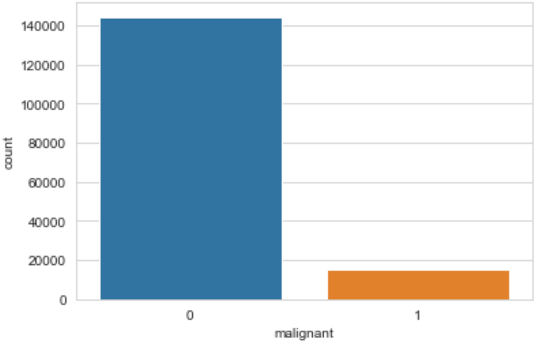




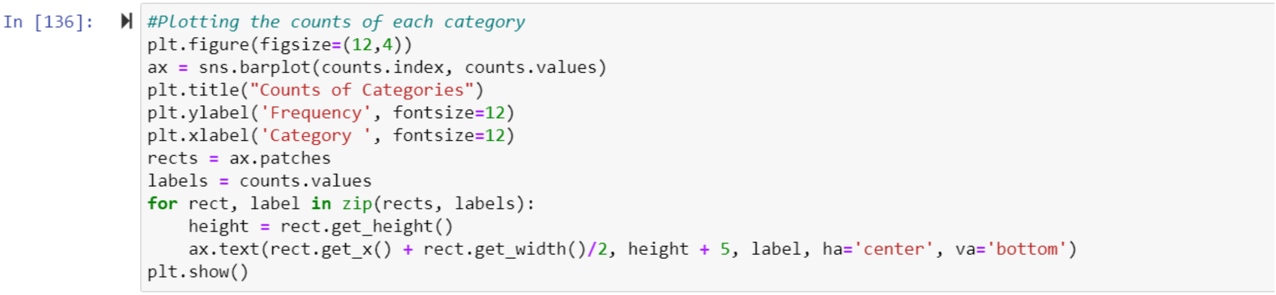






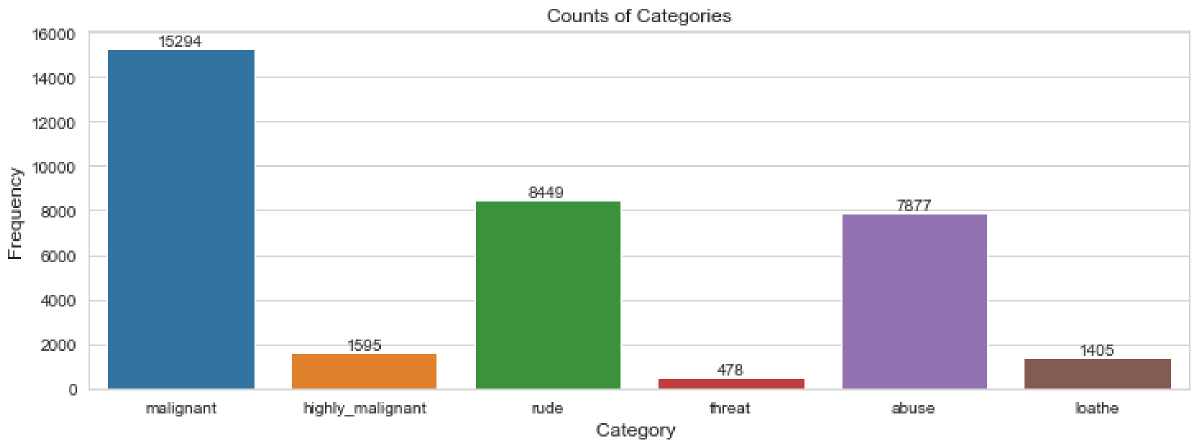


Plotting the counts for each category



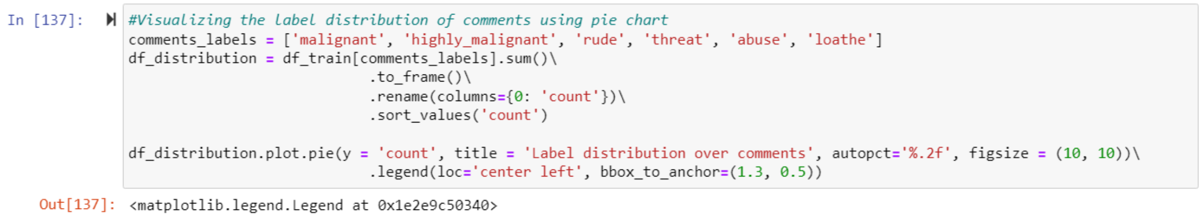
In [136]:

```
#Plotting the counts of each category
plt.figure(figsize=(12,4))
ax = sns.barplot(counts.index, counts.values)
plt.title("Counts of Categories")
plt.ylabel('Frequency', fontsize=12)
plt.xlabel('Category ', fontsize=12)
rects = ax.patches
labels = counts.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')
plt.show()
```



Malignant comments are the highest among all whereas threat comments are very less. Rude and abuse comments are also present more

Plotting pie chart plot

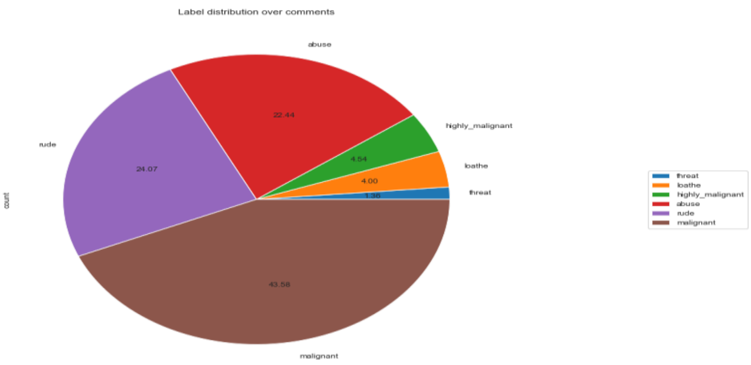


In [137]:

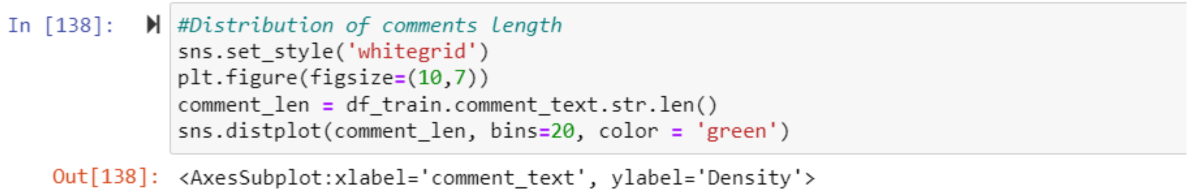
```
#Visualizing the label distribution of comments using pie chart
comments_labels = ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe']
df_distribution = df_train[comments_labels].sum()\
                            .to_frame()\
                            .rename(columns={0: 'count'})\
                            .sort_values('count')

df_distribution.plot.pie(y = 'count', title = 'Label distribution over comments', autopct='%.2f', figsize = (10, 10))\
                            .legend(loc='center left', bbox_to_anchor=(1.3, 0.5))
```

Out[137]:

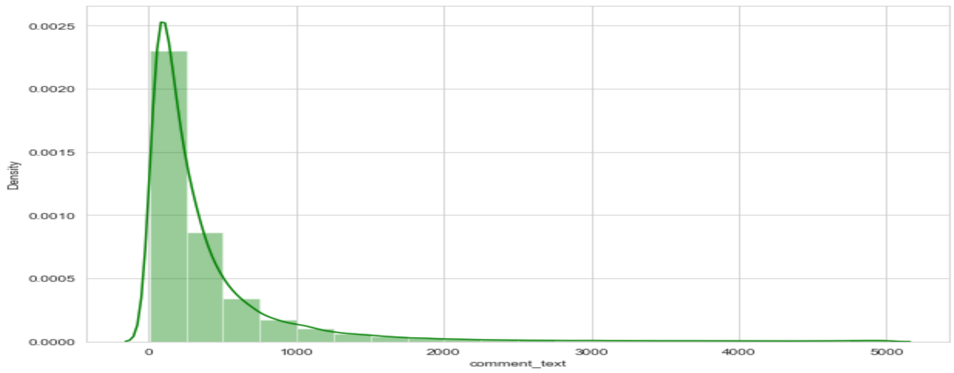
```
<matplotlib.legend.Legend at 0x1e2e9c50340>
```



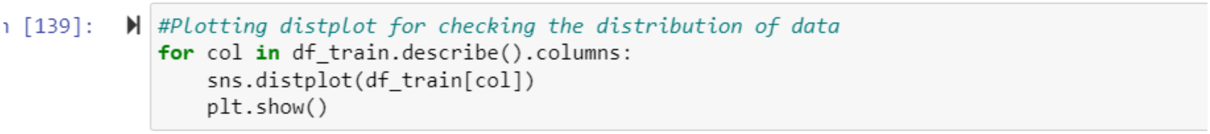
Distribution of comments length



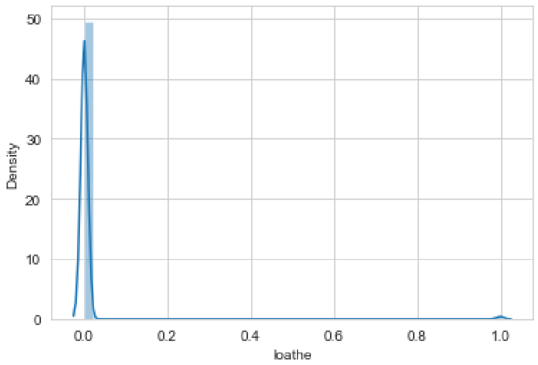
```
In [138]:  #Distribution of comments length
            sns.set_style('whitegrid')
            plt.figure(figsize=(10,7))
            comment_len = df_train.comment_text.str.len()
            sns.distplot(comment_len, bins=20, color = 'green')

Out[138]: <AxesSubplot:xlabel='comment_text', ylabel='Density'>
```

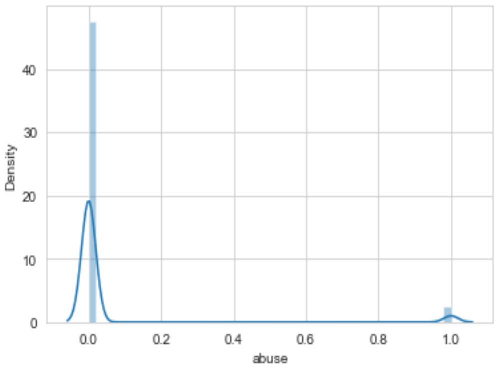


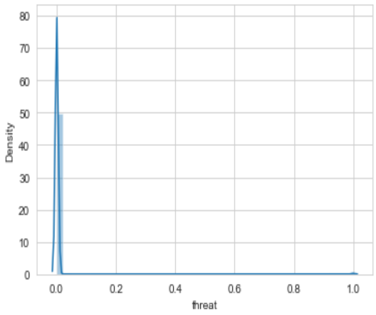
Plotting distribution plot for all features

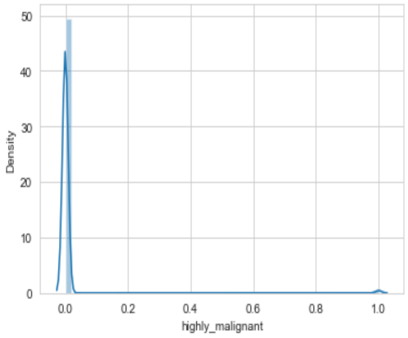


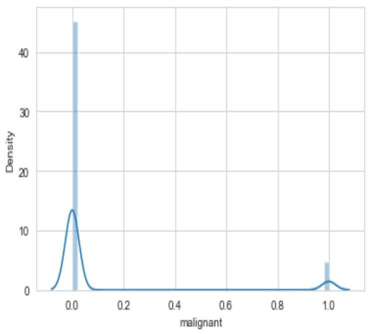
```
ı [139]:  #Plotting distplot for checking the distribution of data
          for col in df_train.describe().columns:
              sns.distplot(df_train[col])
              plt.show()
```











Dropping the Column

Due to the wide range of given data, it is extremely fruitful to clean, shape and set the data in the most suitable form. Dropping unnecessary columns declines the chances of producing errors. Thus, column ‘ID’ was dropped from the train dataset as every comment has its own unique id. After dropping the same, the dataset is now having 7 attributes in total including the target variables.

Cleaning the data using NLP

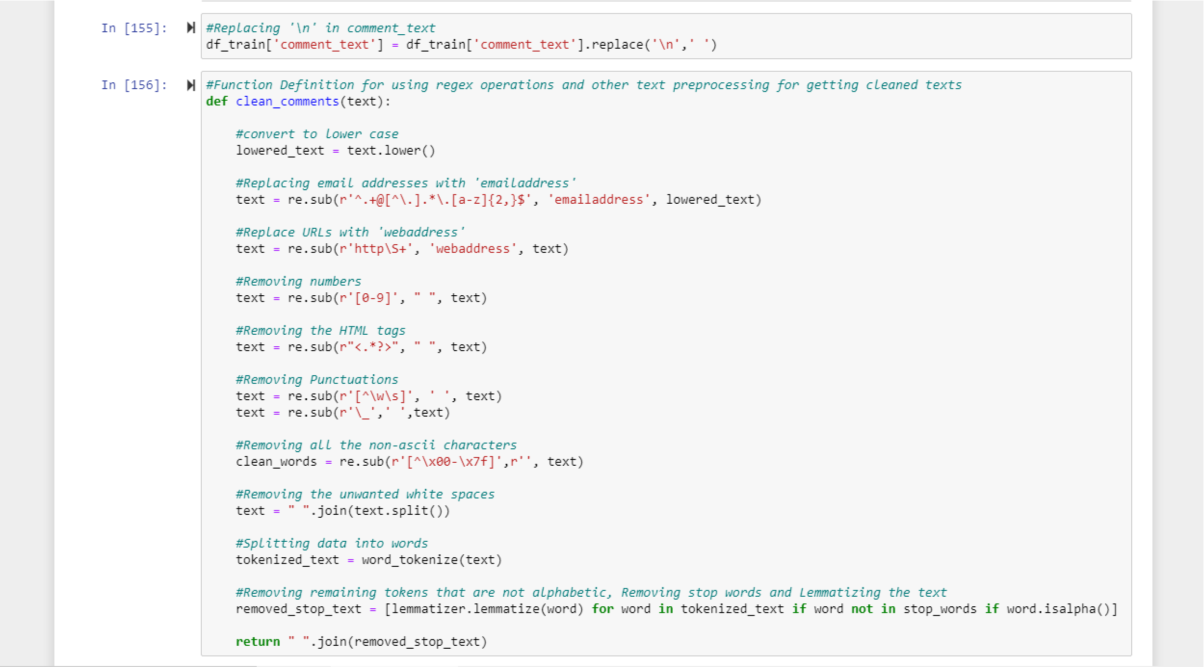
Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. This data is usually not necessary or helpful when it comes to analysing data because it may hinder the process or provide inaccurate results.

Before cleaning the data, a new column is created named ‘length_before_cleaning’ which shows the total length of the comments respectively before cleaning the text.

The following steps were taken in order to clean the text:

- Replaced the extra lines or ‘\n’ from the text.
- Transform the text into lower case.
- Replaced the email addresses with the text 'emailaddress'
- Replaced the URLs with the text 'webaddress'
- Removed the numbers
- Removed the HTML tags
- Removed the punctuations
- Removed all the non-ascii characters
- Removed the unwanted white spaces
- Removed the remaining tokens that are not alphabetic
- Removed the stop words

The code for the above-mentioned steps is mentioned below:



```
In [155]:  #Replacing '\n' in comment_text
            df_train['comment_text'] = df_train['comment_text'].replace('\n',' ')

In [156]:  #Function Definition for using regex operations and other text preprocessing for getting cleaned texts
            def clean_comments(text):

                #convert to lower case
                lowered_text = text.lower()

                #Replacing email addresses with 'emailaddress'
                text = re.sub(r'^.+@[^\.].*\.[a-z]{2,}$', 'emailaddress', lowered_text)

                #Replace URLs with 'webaddress'
                text = re.sub(r'http\S+', 'webaddress', text)

                #Removing numbers
                text = re.sub(r'[0-9]', " ", text)

                #Removing the HTML tags
                text = re.sub(r"<.*?>", " ", text)

                #Removing Punctuations
                text = re.sub(r'[^\w\s]', ' ', text)
                text = re.sub(r'\_',' ',text)

                #Removing all the non-ascii characters
                clean_words = re.sub(r'[^\x00-\x7f]',r'', text)

                #Removing the unwanted white spaces
                text = " ".join(text.split())

                #Splitting data into words
                tokenized_text = word_tokenize(text)

                #Removing remaining tokens that are not alphabetic, Removing stop words and Lemmatizing the text
                removed_stop_text = [lemmatizer.lemmatize(word) for word in tokenized_text if word not in stop_words if word.isalpha()]

                return " ".join(removed_stop_text)
```

All the text cleaning or the above steps are performed by defining a function and applying the same using apply() to the comment_text column of the train dataset.

Tokenization

Word tokenization is the process of splitting a large sample of text into words. This is a requirement in natural language processing tasks where each word needs to be captured and subjected to further analysis.

After cleaning the text, each comment i.e., the corpus is split into words. Thus, the text is tokenized into words using word_tokenize()

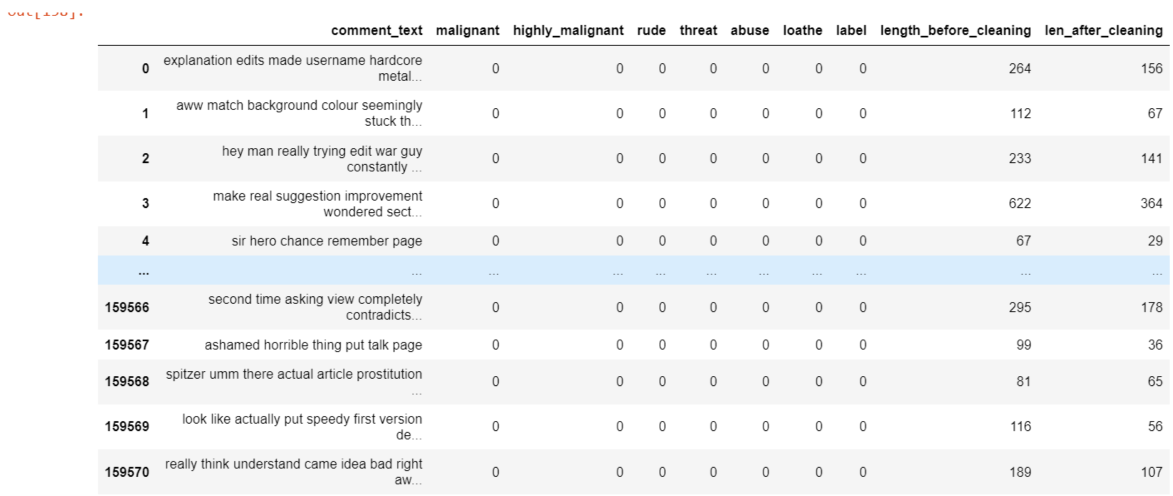
Lemmatization

Lemmatization in NLTK refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word known as the lemma.

The NLTK Lemmatization method is based on WorldNet's builtin morph function. Thus, the words are lemmatized using WordNetLemmatizer() after importing the necessary library to perform the same and then creating the instance for it.

¬ After cleaning the text, a new column is created named 'len_after_cleaning' representing the length of each comment respectively in a column 'comment_text' after doing the required cleaning of the text. With this, it’s come to know about how much data is cleaned.

Length of comment_text data, before and after:

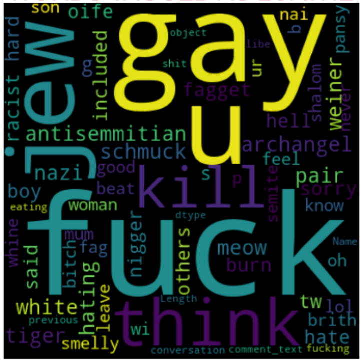


	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	label	length_before_cleaning	len_after_cleaning
0	explanation edits made username hardcore metal...	0	0	0	0	0	0	0	264	156
1	aww match background colour seemingly stuck th...	0	0	0	0	0	0	0	112	67
2	hey man really trying edit war guy constantly ...	0	0	0	0	0	0	0	233	141
3	make real suggestion improvement wondered sect...	0	0	0	0	0	0	0	622	364
4	sir hero chance remember page	0	0	0	0	0	0	0	67	29
...	...	...	...	...	...	...	...	...	...	...
159566	second time asking view completely contradicts...	0	0	0	0	0	0	0	295	178
159567	ashamed horrible thing put talk page	0	0	0	0	0	0	0	99	36
159568	spitzer umm there actual article prostitution ...	0	0	0	0	0	0	0	81	65
159569	look like actually put speedy first version de...	0	0	0	0	0	0	0	116	56
159570	really think understand came idea bad right aw...	0	0	0	0	0	0	0	189	107

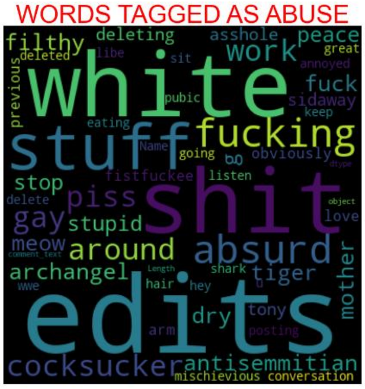
Similar step was used on test data. The dataset was processed using the function created.

Plotting wordcloud for each feature:

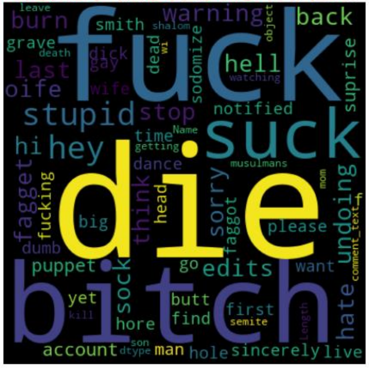
WORDS TAGGED AS LOATHE



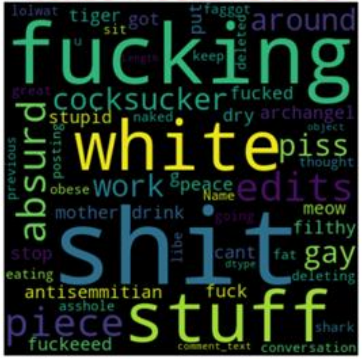
WORDS TAGGED AS ABUSE



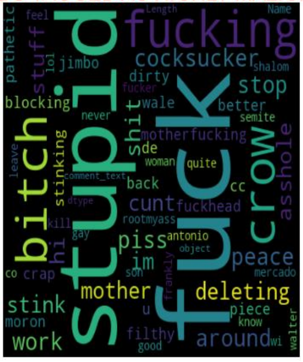
WORDS TAGGED AS THREAT



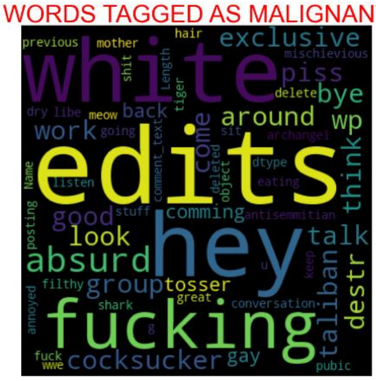
WORDS TAGGED AS RUDE



WORDS TAGGED AS HIGHLY MALIGNANT



WORDS TAGGED AS MALIGNANT

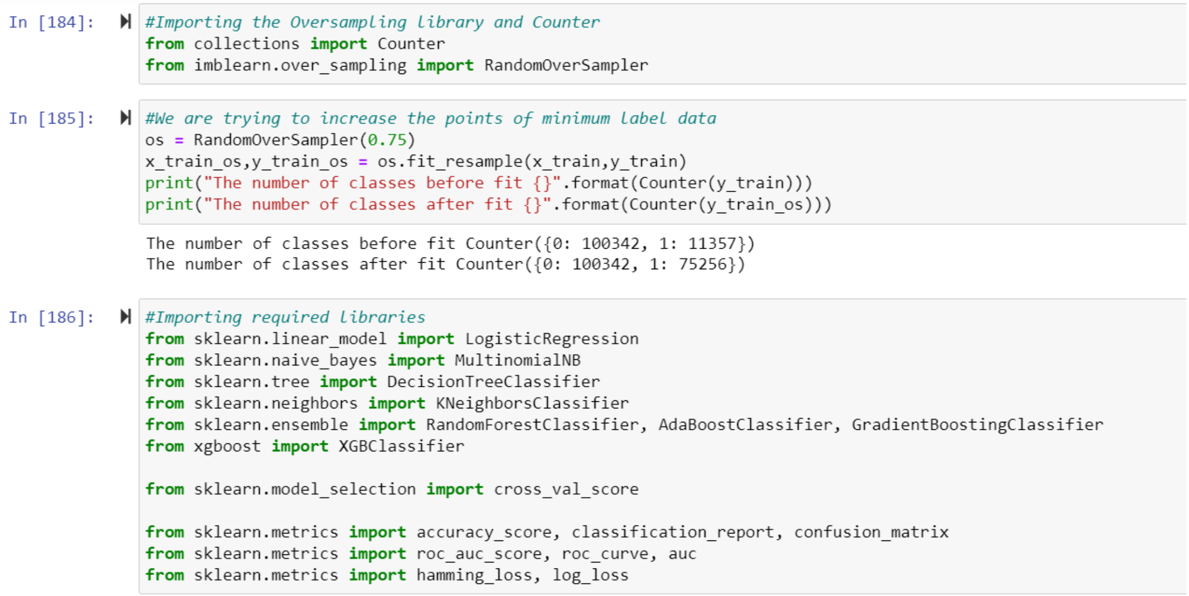


From the above wordclouds, we can see that the large texts have more weightage in their respective type of comments whereas small texts have the lesser weightages

HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

For doing this project, the hardware used is a laptop with high end specification and a stable internet connection. While coming to software part, I had used anaconda navigator and in that I have used Jupyter notebook to do my python programming and analysis.

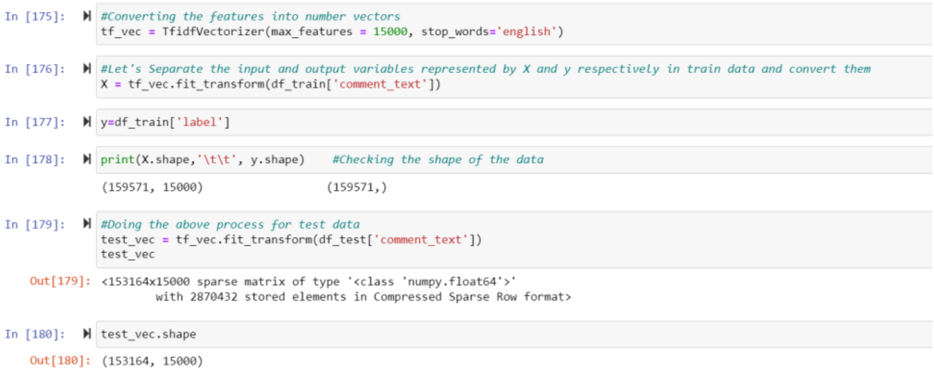
For using an CSV file, Microsoft excel is needed. In Jupyter notebook, I had used lots of python libraries to carry out this project and I have mentioned below with proper justification:



```
In [184]:  #Importing the Oversampling library and Counter
            from collections import Counter
            from imblearn.over_sampling import RandomOverSampler

In [185]:  #We are trying to increase the points of minimum label data
            os = RandomOverSampler(0.75)
            x_train_os,y_train_os = os.fit_resample(x_train,y_train)
            print("The number of classes before fit {}".format(Counter(y_train)))
            print("The number of classes after fit {}".format(Counter(y_train_os)))

            The number of classes before fit Counter({0: 100342, 1: 11357})
            The number of classes after fit Counter({0: 100342, 1: 75256})

In [186]:  #Importing required libraries
            from sklearn.linear_model import LogisticRegression
            from sklearn.naive_bayes import MultinomialNB
            from sklearn.tree import DecisionTreeClassifier
            from sklearn.neighbors import KNeighborsClassifier
            from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
            from xgboost import XGBClassifier

            from sklearn.model_selection import cross_val_score

            from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
            from sklearn.metrics import roc_auc_score, roc_curve, auc
            from sklearn.metrics import hamming_loss, log_loss
```

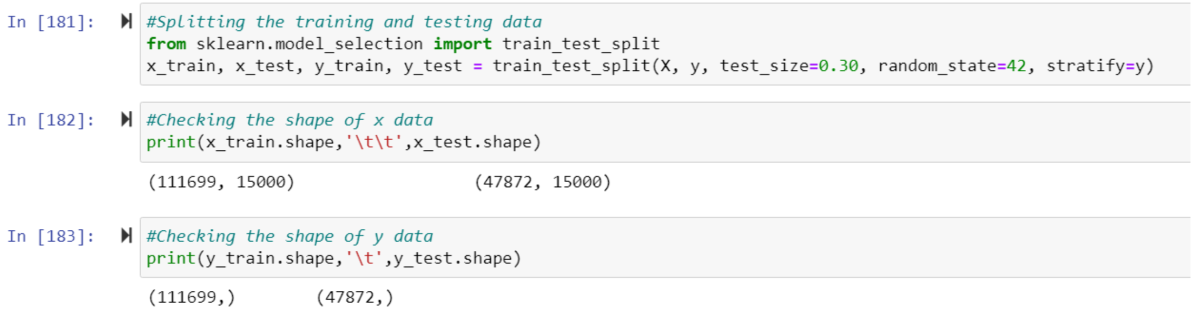
MODEL/S DEVELOPMENT AND EVALUATION

Separating independent and dependent features and converting the data into number features using Vectorizer



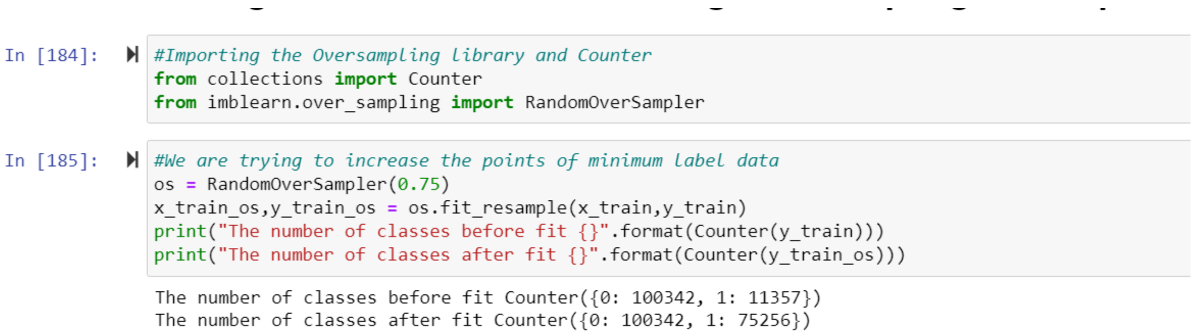
```
In [175]:  #Converting the features into number vectors
            tf_vec = TfidfVectorizer(max_features = 15000, stop_words='english')

In [176]:  #Let's Separate the input and output variables represented by X and y respectively in train data and convert them
            X = tf_vec.fit_transform(df_train['comment_text'])

In [177]:  y=df_train['label']

In [178]:  print(X.shape,'\t\t', y.shape)    #Checking the shape of the data

            (159571, 15000)                 (159571,)

In [179]:  #Doing the above process for test data
            test_vec = tf_vec.fit_transform(df_test['comment_text'])
            test_vec

Out[179]:  <153164x15000 sparse matrix of type '<class 'numpy.float64'>'
                    with 2870432 stored elements in Compressed Sparse Row format>

In [180]:  test_vec.shape

Out[180]:  (153164, 15000)
```

Splitting training and testing data, along with handling imbalanced dataset using RandomOverSampler



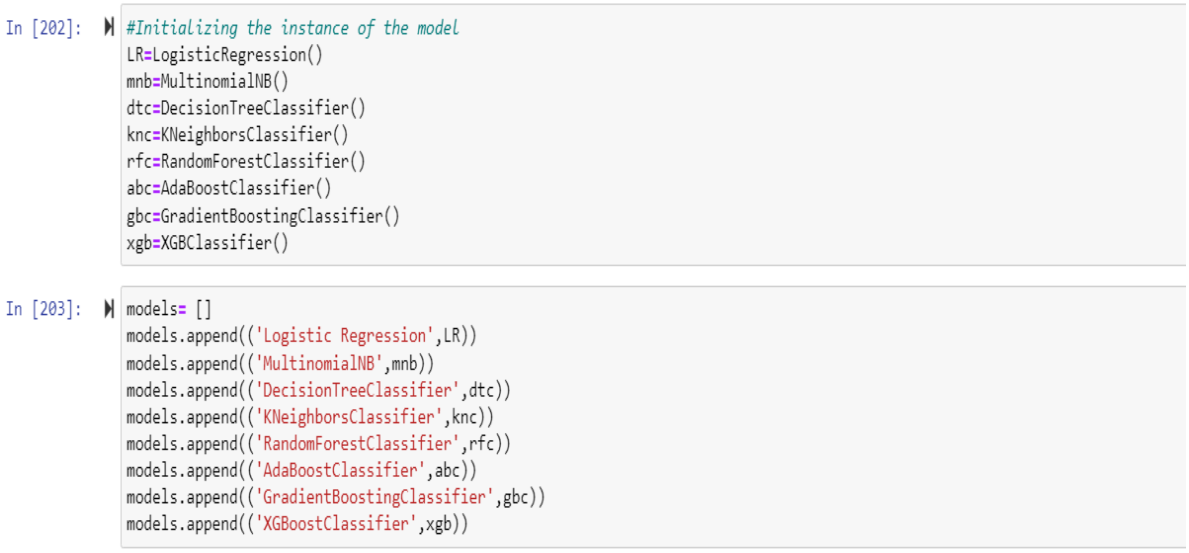
```
In [181]:  #Splitting the training and testing data
           from sklearn.model_selection import train_test_split
           x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42, stratify=y)

In [182]:  #Checking the shape of x data
           print(x_train.shape,'\t\t',x_test.shape)

           (111699, 15000)                  (47872, 15000)

In [183]:  #Checking the shape of y data
           print(y_train.shape,'\t',y_test.shape)

           (111699,)        (47872,)
```

Handling the imbalance data using oversampling technique

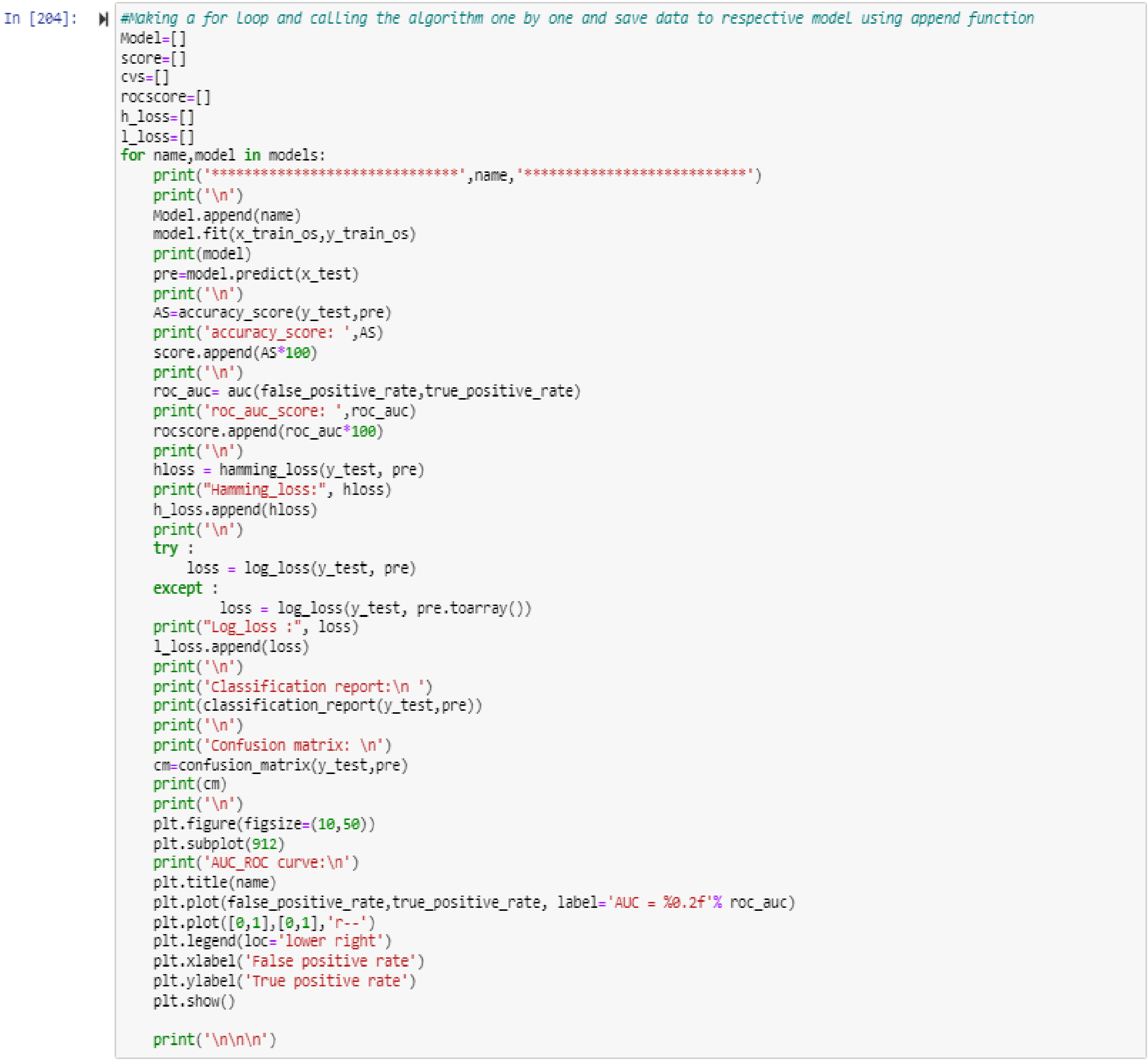


```
In [184]:  #Importing the Oversampling library and Counter
           from collections import Counter
           from imblearn.over_sampling import RandomOverSampler

In [185]:  #We are trying to increase the points of minimum label data
           os = RandomOverSampler(0.75)
           x_train_os,y_train_os = os.fit_resample(x_train,y_train)
           print("The number of classes before fit {}".format(Counter(y_train)))
           print("The number of classes after fit {}".format(Counter(y_train_os)))

           The number of classes before fit Counter({0: 100342, 1: 11357})
           The number of classes after fit Counter({0: 100342, 1: 75256})
```

After oversampling, we can see that the dataset is balanced, by increasing the weightage of the lowest kind with sampling 75% of the highest weightage data to it.



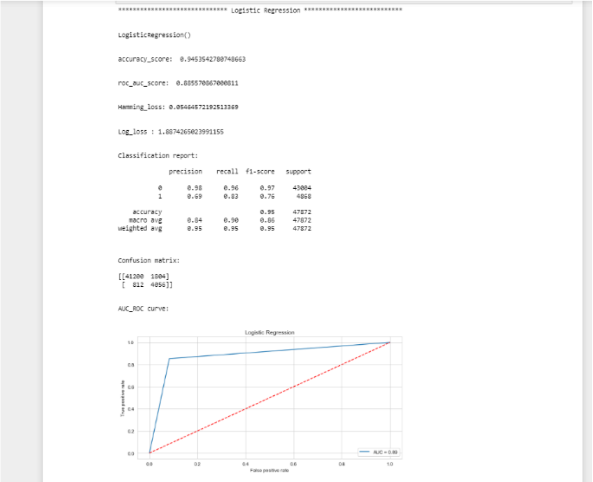
```
In [202]:  #Initializing the instance of the model
           LR=LogisticRegression()
           mnb=MultinomialNB()
           dtc=DecisionTreeClassifier()
           knc=KNeighborsClassifier()
           rfc=RandomForestClassifier()
           abc=AdaBoostClassifier()
           gbc=GradientBoostingClassifier()
           xgb=XGBClassifier()

In [203]:  models= []
           models.append(('Logistic Regression',LR))
           models.append(('MultinomialNB',mnb))
           models.append(('DecisionTreeClassifier',dtc))
           models.append(('KNeighborsClassifier',knc))
           models.append(('RandomForestClassifier',rfc))
           models.append(('AdaBoostClassifier',abc))
           models.append(('GradientBoostingClassifier',gbc))
           models.append(('XGBoostClassifier',xgb))
```



```
In [204]:  #Making a for loop and calling the algorithm one by one and save data to respective model using append function
Model=[]
score=[]
cvs=[]
rocscore=[]
h_loss=[]
l_loss=[]
for name,model in models:
    print('******************************',name,'***************************')
    print('\n')
    Model.append(name)
    model.fit(x_train_os,y_train_os)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=accuracy_score(y_test,pre)
    print('accuracy_score: ',AS)
    score.append(AS*100)
    print('\n')
    roc_auc= auc(false_positive_rate,true_positive_rate)
    print('roc_auc_score: ',roc_auc)
    rocscore.append(roc_auc*100)
    print('\n')
    hloss = hamming_loss(y_test, pre)
    print("Hamming_loss:", hloss)
    h_loss.append(hloss)
    print('\n')
    try :
        loss = log_loss(y_test, pre)
    except :
            loss = log_loss(y_test, pre.toarray())
    print("Log_loss :", loss)
    l_loss.append(loss)
    print('\n')
    print('Classification report:\n ')
    print(classification_report(y_test,pre))
    print('\n')
    print('Confusion matrix: \n')
    cm=confusion_matrix(y_test,pre)
    print(cm)
    print('\n')
    plt.figure(figsize=(10,50))
    plt.subplot(912)
    print('AUC_ROC curve:\n')
    plt.title(name)
    plt.plot(false_positive_rate,true_positive_rate, label='AUC = %0.2f'% roc_auc)
    plt.plot([0,1],[0,1],'r--')
    plt.legend(loc='lower right')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.show()

    print('\n\n\n')
```

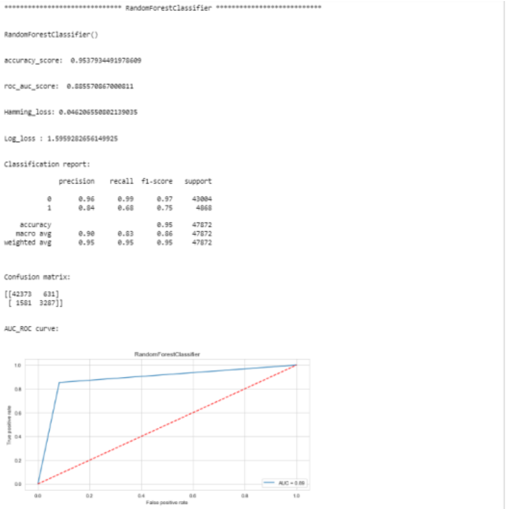
After running the above for loops for the algorithms, the output will be as follows:

Logistic Regression.



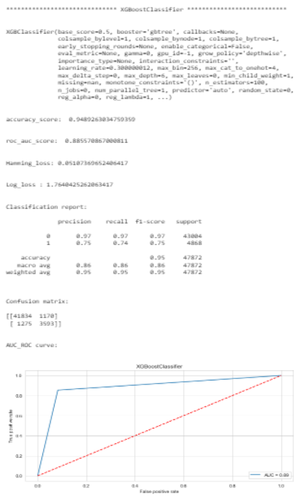
```
****************************** Logistic Regression ***************************


LogisticRegression()


accuracy_score:  0.9453542780748663


roc_auc_score:  0.885570067000811


Hamming_loss: 0.05464572192513369


Log_loss : 1.8874265023991155


Classification report:

              precision    recall  f1-score   support

           0       0.98      0.96      0.97     43004
           1       0.69      0.83      0.76      4868

    accuracy                           0.95     47872
   macro avg       0.84      0.90      0.86     47872
weighted avg       0.95      0.95      0.95     47872



Confusion matrix:

[[41200  1804]
 [  812  4056]]


AUC_ROC curve:
```

Random Classifier



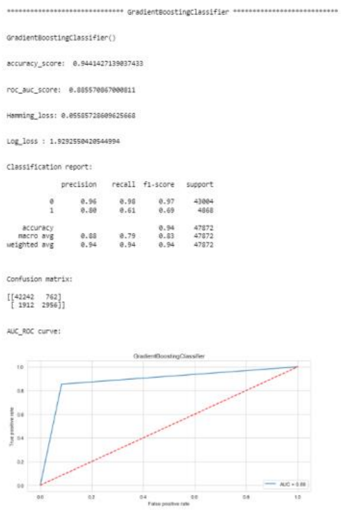
```
****************************** RandomForestClassifier ***************************

RandomForestClassifier()

accuracy_score:  0.9537934491978609

roc_auc_score:  0.885570867000811

Hamming_loss: 0.046206550802139035

Log_loss : 1.5959282656149925

Classification report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     43004
           1       0.84      0.68      0.75      4868

    accuracy                           0.95     47872
   macro avg       0.90      0.83      0.86     47872
weighted avg       0.95      0.95      0.95     47872


Confusion matrix:

[[42373   631]
 [ 1581  3287]]

AUC_ROC curve:
```

Xgboost



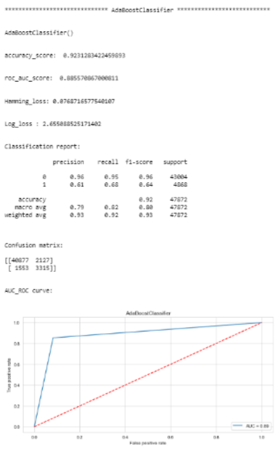
```
***************************** XGBoostClassifier **************************

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)

accuracy_score:  0.9489263034759359

roc_auc_score:  0.885570867000811

Hamming_loss: 0.05107369652406417

Log_loss : 1.7640425262063417

Classification report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     43004
           1       0.75      0.74      0.75      4868

    accuracy                           0.95     47872
   macro avg       0.86      0.86      0.86     47872
weighted avg       0.95      0.95      0.95     47872


Confusion matrix:

[[41834  1170]
 [ 1275  3593]]

AUC_ROC curve:
```

Gradient Classifier



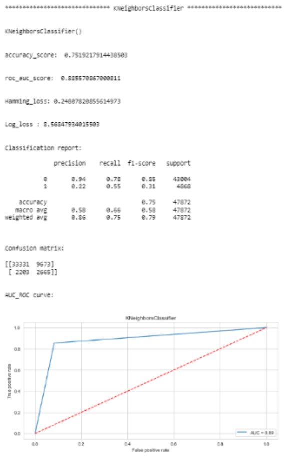
```
****************************** GradientBoostingClassifier ***************************

GradientBoostingClassifier()

accuracy_score:  0.9441427139037433

roc_auc_score:  0.885570867000811

Hamming_loss: 0.05585728609625668

Log_loss : 1.9292550420544994

Classification report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     43004
           1       0.80      0.61      0.69      4868

    accuracy                           0.94     47872
   macro avg       0.88      0.79      0.83     47872
weighted avg       0.94      0.94      0.94     47872


Confusion matrix:

[[42242   762]
 [ 1912  2956]]

AUC_ROC curve:
```

Adaboost



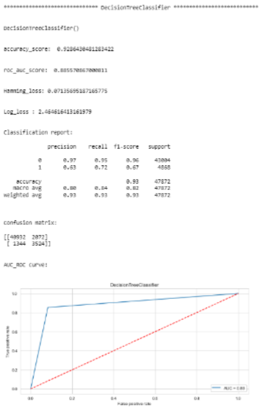
```
***************************** AdaBoostClassifier **************************

AdaBoostClassifier()

accuracy_score:  0.9231283422459893

roc_auc_score:  0.885570867000811

Hamming_loss: 0.07687165775401007

Log_loss : 2.655088525171402

Classification report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96     43004
           1       0.61      0.68      0.64      4868

    accuracy                           0.92     47872
   macro avg       0.79      0.82      0.80     47872
weighted avg       0.93      0.92      0.93     47872


Confusion matrix:

[[40877  2127]
 [ 1553  3315]]

AUC_ROC curve:
```

KNeighbors



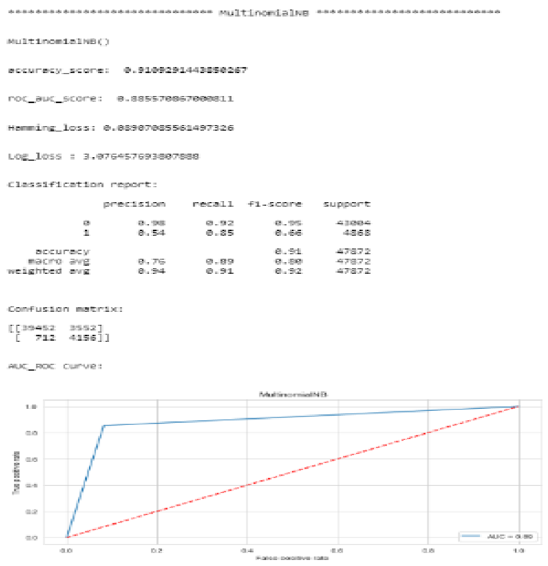
```
****************************** KNeighborsClassifier ***************************

KNeighborsClassifier()

accuracy_score:  0.7519217914438503

roc_auc_score:  0.885570867000811

Hamming_loss: 0.24807820855614973

Log_loss : 8.56847934015503

Classification report:
              precision    recall  f1-score   support

           0       0.94      0.78      0.85     43004
           1       0.22      0.55      0.31      4868

    accuracy                           0.75     47872
   macro avg       0.58      0.66      0.58     47872
weighted avg       0.86      0.75      0.79     47872


Confusion matrix:

[[33331  9673]
 [ 2203  2665]]

AUC_ROC curve:
```

Decision Tree



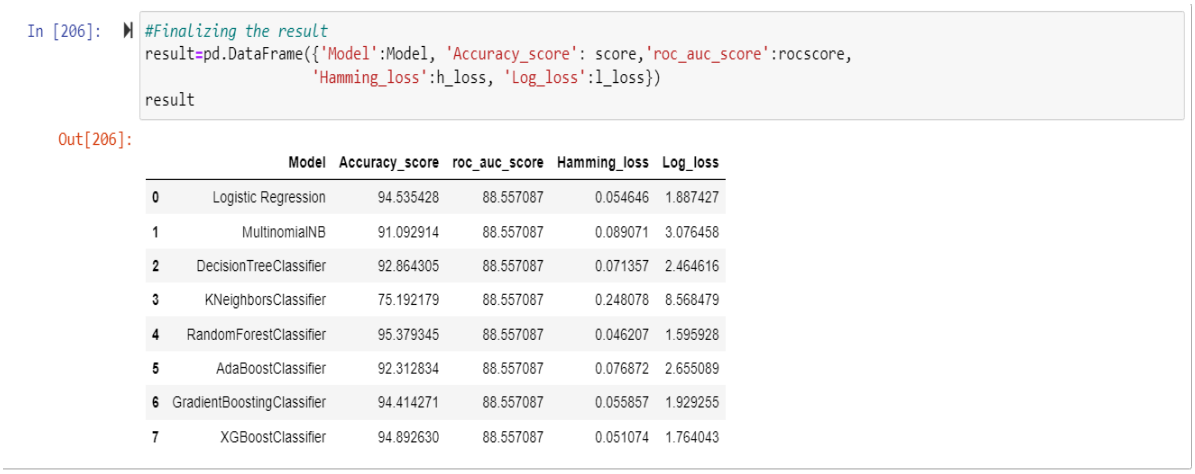
```
***************************** DecisionTreeClassifier **************************

DecisionTreeClassifier()

accuracy_score:  0.9286430481283422

roc_auc_score:  0.885570867000811

Hamming_loss: 0.07135695187165775

Log_loss : 2.464616413161979

Classification report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96     43004
           1       0.63      0.72      0.67      4868

    accuracy                           0.93     47872
   macro avg       0.80      0.84      0.82     47872
weighted avg       0.93      0.93      0.93     47872


confusion matrix:

[[40932  2072]
 [ 1344  3524]]

AUC_ROC curve:
```

MultinomialNB



```
****************************** MultinomialNB **************************

MultinomialNB()

accuracy_score:  0.9109291443850267

roc_auc_score:  0.8855708670008811

Hamming_loss: 0.08907085561497326

Log_loss : 3.076457693807888

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.92      0.95     43004
           1       0.54      0.85      0.66      4868

    accuracy                           0.91     47872
   macro avg       0.76      0.89      0.80     47872
weighted avg       0.94      0.91      0.92     47872


Confusion matrix:
[[39452  3552]
 [  712  4156]]

AUC_ROC Curve:
```

Results in dataset

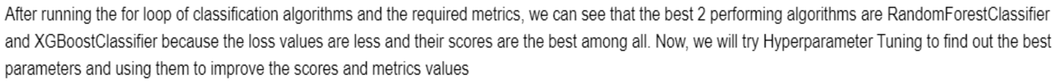


In [206]:

```
#Finalizing the result
result=pd.DataFrame({'Model':Model, 'Accuracy_score': score,'roc_auc_score':rocscore,
                     'Hamming_loss':h_loss, 'Log_loss':l_loss})
result
```

Out[206]:

	Model	Accuracy_score	roc_auc_score	Hamming_loss	Log_loss
0	Logistic Regression	94.535428	88.557087	0.054646	1.887427
1	MultinomialNB	91.092914	88.557087	0.089071	3.076458
2	DecisionTreeClassifier	92.864305	88.557087	0.071357	2.464616
3	KNeighborsClassifier	75.192179	88.557087	0.248078	8.568479
4	RandomForestClassifier	95.379345	88.557087	0.046207	1.595928
5	AdaBoostClassifier	92.312834	88.557087	0.076872	2.655089
6	GradientBoostingClassifier	94.414271	88.557087	0.055857	1.929255
7	XGBoostClassifier	94.892630	88.557087	0.051074	1.764043

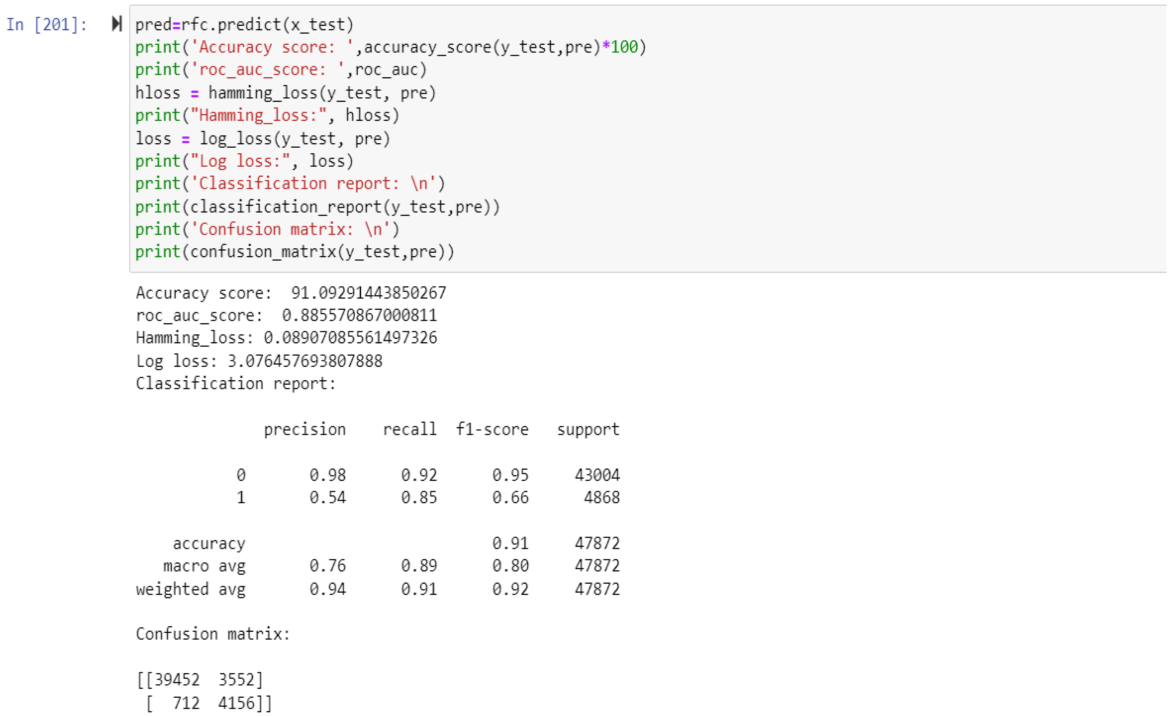


After running the for loop of classification algorithms and the required metrics, we can see that the best 2 performing algorithms are RandomForestClassifier and XGBoostClassifier because the loss values are less and their scores are the best among all. Now, we will try Hyperparameter Tuning to find out the best parameters and using them to improve the scores and metrics values

After running the for loop of classification algorithms and the required metrics, we can see that the best 2 performing algorithms are RandomForestClassifier and XGBoostClassifier because the loss values are less and their scores are the best among all. Now, we will try Hyperparameter Tuning to find out the best parameters and using them to improve the scores and metrics values.

Hyperparameter Tuning

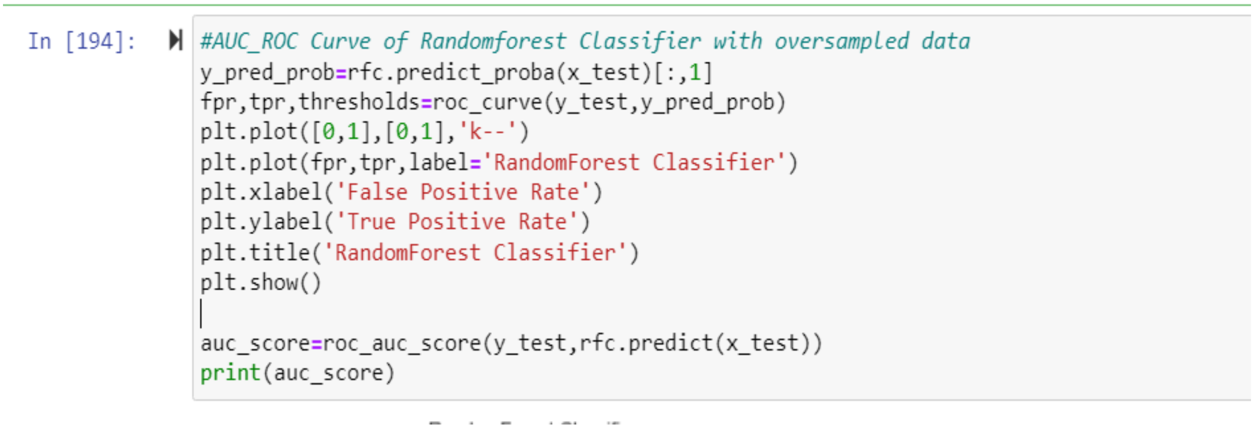
If we run GridSearchCV and RandomSearchCV, it takes more than 2 hours to run the code as the dataset is huge and the best params are not obtained from it due to more computational power requirement. The AUC Score, f1-score and recall value is high when we use randomforest with over sampling data. So, we choose RandomForestClassifier model with over sampled data as our best model among all models.



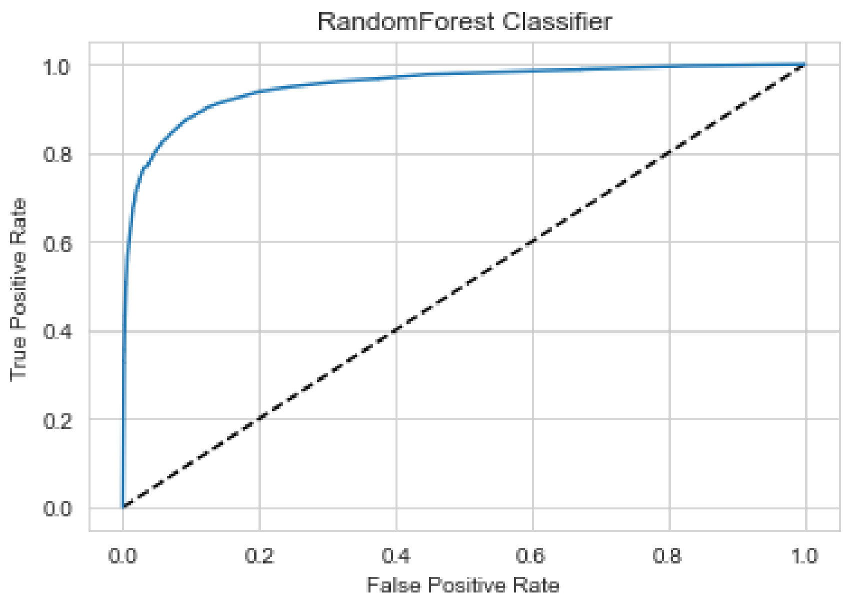
```
In [201]:  pred=rfc.predict(x_test)
            print('Accuracy score: ',accuracy_score(y_test,pre)*100)
            print('roc_auc_score: ',roc_auc)
            hloss = hamming_loss(y_test, pre)
            print("Hamming_loss:", hloss)
            loss = log_loss(y_test, pre)
            print("Log loss:", loss)
            print('Classification report: \n')
            print(classification_report(y_test,pre))
            print('Confusion matrix: \n')
            print(confusion_matrix(y_test,pre))
```

```
Accuracy score:  91.09291443850267
roc_auc_score:  0.885570867000811
Hamming_loss: 0.08907085561497326
Log loss: 3.076457693807888
Classification report: 

              precision    recall  f1-score   support

           0       0.98      0.92      0.95     43004
           1       0.54      0.85      0.66      4868

    accuracy                           0.91     47872
   macro avg       0.76      0.89      0.80     47872
weighted avg       0.94      0.91      0.92     47872

Confusion matrix: 

[[39452  3552]
 [  712  4156]]
```

After finding the metrics values and other required scores, we will plot the auc roc curve with the help of auc score obtained with the help of the fitted model above:

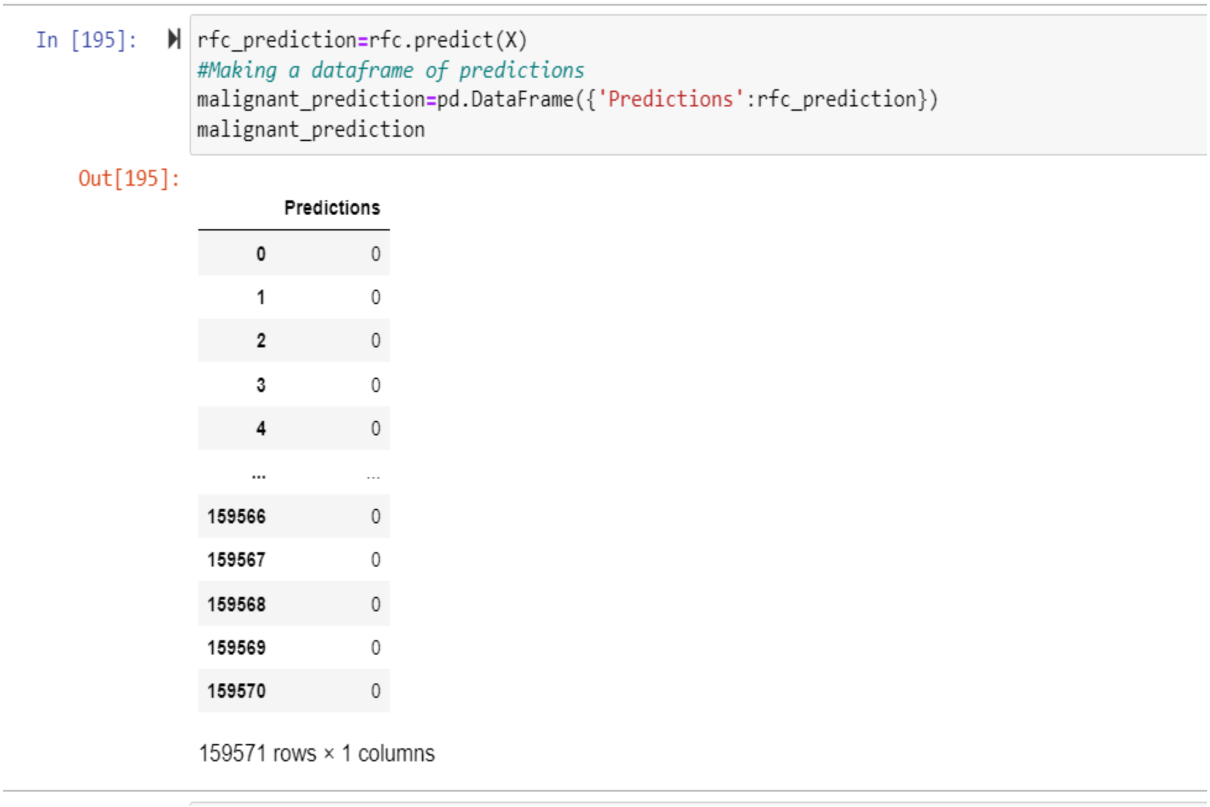


```
In [194]:  #AUC_ROC Curve of Randomforest Classifier with oversampled data
            y_pred_prob=rfc.predict_proba(x_test)[:,1]
            fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
            plt.plot([0,1],[0,1],'k--')
            plt.plot(fpr,tpr,label='RandomForest Classifier')
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
            plt.title('RandomForest Classifier')
            plt.show()

            auc_score=roc_auc_score(y_test,rfc.predict(x_test))
            print(auc_score)
```



Finalizing the model

We will final the model by predicting the values and saving the model in a pickle file, which will be used for prediction of test data



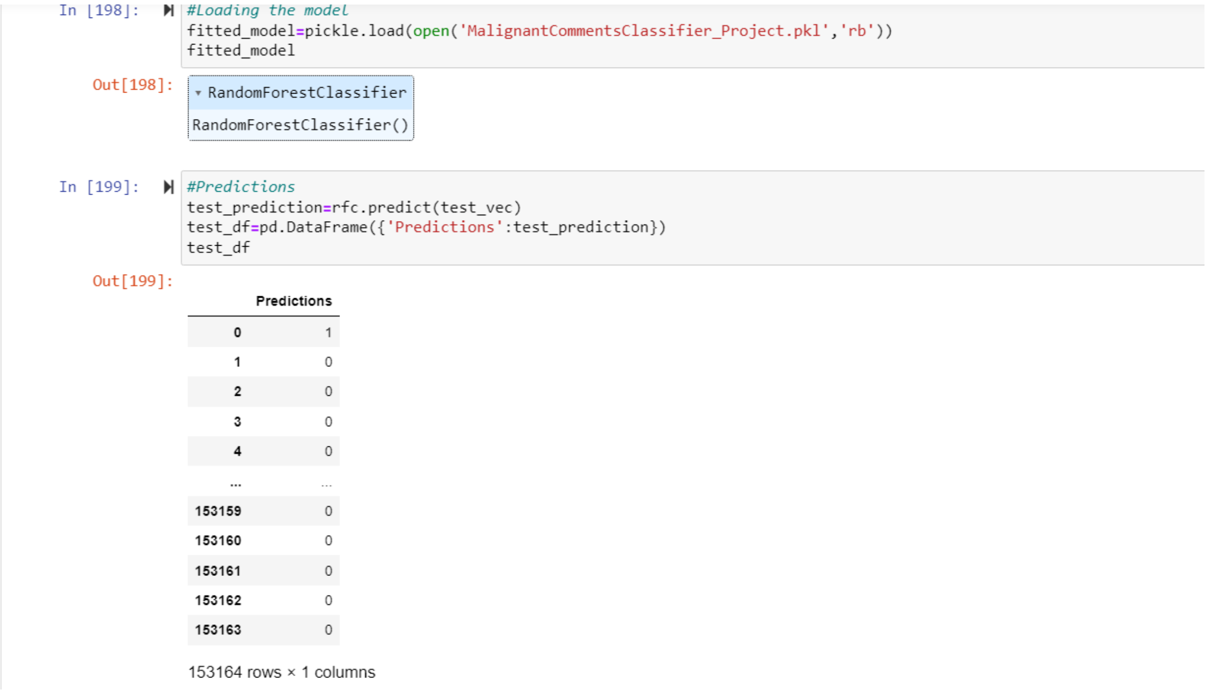
In [195]:
```
rfc_prediction=rfc.predict(X)
#Making a dataframe of predictions
malignant_prediction=pd.DataFrame({'Predictions':rfc_prediction})
malignant_prediction
```

Out[195]:

	Predictions
0	0
1	0
2	0
3	0
4	0
...	...
159566	0
159567	0
159568	0
159569	0
159570	0

159571 rows × 1 columns

Predicting using test data



```
In [198]:  #Loading the model
           fitted_model=pickle.load(open('MalignantCommentsClassifier_Project.pkl','rb'))
           fitted_model

Out[198]:  ▾ RandomForestClassifier
           RandomForestClassifier()

In [199]:  #Predictions
           test_prediction=rfc.predict(test_vec)
           test_df=pd.DataFrame({'Predictions':test_prediction})
           test_df

Out[199]:
                  Predictions
                0           1
                1           0
                2           0
                3           0
                4           0
              ...         ...
           153159           0
           153160           0
           153161           0
           153162           0
           153163           0

           153164 rows × 1 columns
```

After predicting using test data, we will store the results in a csv file.

CONCLUSION

Key Findings and Conclusions of the Study

After the completion of this project, we got an insight of how to preprocess the data, analysing the data and building a model.

-> First, we imported both training and testing data, which had nearly 150000+ records.

-> We did all the required pre-processing steps like checking null values, datatypes check, dropping unnecessary columns, etc.

-> We used the training data for doing Exploratory Data Analysis using various plots and recorded the observations.

-> While observing the results, we found that the dataset was in highly imbalanced side and we need to handle it, in order to avoid overfitting problem.

-> Using NLP, we pre-processed the comment text and did other steps.

-> As the problem was a multi-class classifier, we took a new feature known as label and combined the comment_labels output together using sum() and then stored in that feature. For a binary classification problem, we scaled the data accordingly.

-> After applying Tf-idf Vectoriser, we used an oversampling technique called RandomOverSampler for handling the imbalanced data. There, we took 75% of the high points data and sampled it to the low points data so that both weights could be balanced equally and we could get proper result.

-> Then, we split the data using train_test_split and then we started the model building process by running as many algorithms in a for loop, with difference metrics like cross_val_score, confusion matrix, auc_score, log loss, hamming loss, etc.

-> We found that RandomForestClassifier and XGBoostClassifier were performing well. The next step was to perform hyperparameter tuning technique to these models for finding out the best parameters and trying to improve our scores.

-> The major problem with this dataset occurred in this step. It took me nearly 2 hrs to run the code for finding out the best parameters itself as the dataset is large and more computational power was required. Even though we found the best algorithms, it took me 2 hrs to get the results.

-> Therefore, without hyperparameter tuning, we finalized RandomForest as the best performing algorithm by predicting the outputs, saving the model and storing the results in a csv file

-> Then, by using the model we got, another set of predictions were done by using the test data and the results were stored in a separate csv file.