FLIP ROBO

MALIGNANT COMMENT CLASSIFICATION

Submitted by:

Asheem Siwach

ACKNOWLEDGMENT

I would like to thank Flip Robo Technologies for providing me with the opportunity to work on this project from which I have learned a lot. I am also grateful to Mr. Shubham Yadav for his constant guidance and support.

Reference sources are :-

- Google
- Stackoverflow.com
- Analytics vidhya
- Notes and repository from DataTrained

Research papers –

https://www.nltk.org/book/ch05.html

https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

INTRODUCTION

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.

Conceptual Background of the Domain Problem

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

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

Analytical Problem Framing

- ### Mathematical/ Analytical Modeling of the Problem

This was a NLP Project and in this project we deals with the textual data and for understanding the data we used some methods like removing punctuations ,numbers, stopwords and using the lemmatization process convert the complex words into their simpler forms. These processes helped in cleaning the unwanted words form the comments and we were left with only those words which were going to help in our model building. After cleaning the data we used TF-IDF Vectorizer technique to convert textual data into vector form. This technique works on the basis of the frequency of words present in the document. After training with train dataset we use this technique into test dataset.

- ### Data Sources and their formats

This data was provided to me by FlipRobo Technologies into a csv file format. This file contains training and testing dataset. On training dataset we build a model and using this model we have to predict the outcomes from testing dataset.

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.

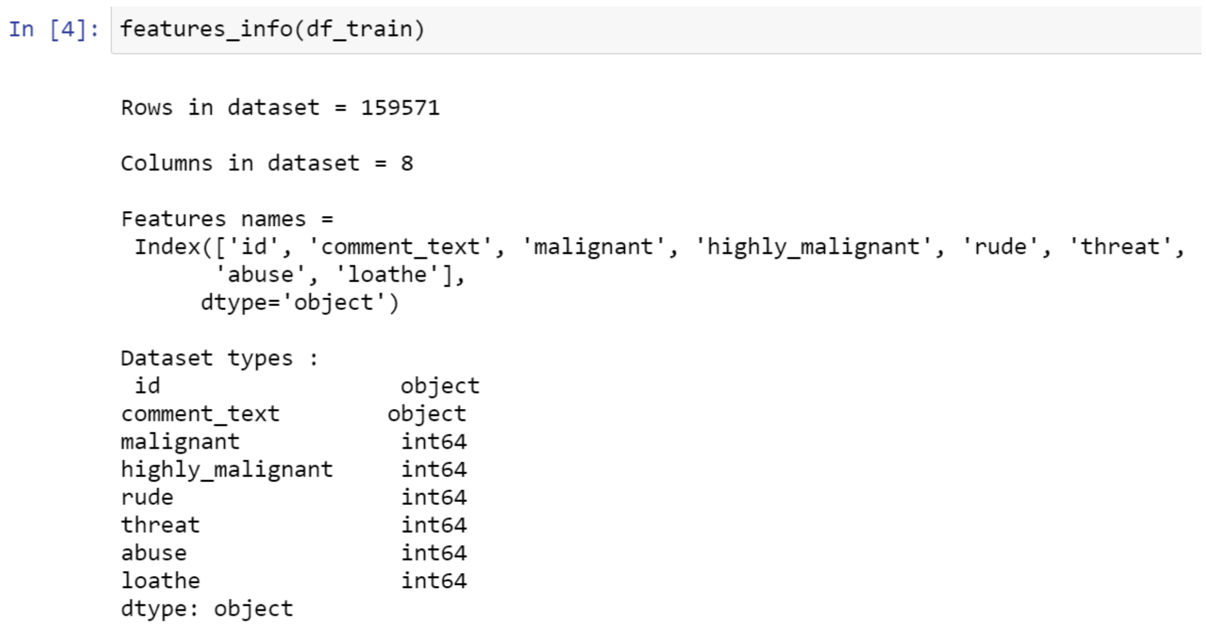
-**Abuse**: It is for comments that are abusive in nature.

-**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.

First of all we upload the training dataset into df_train dataframe and check the datatypes present in it.

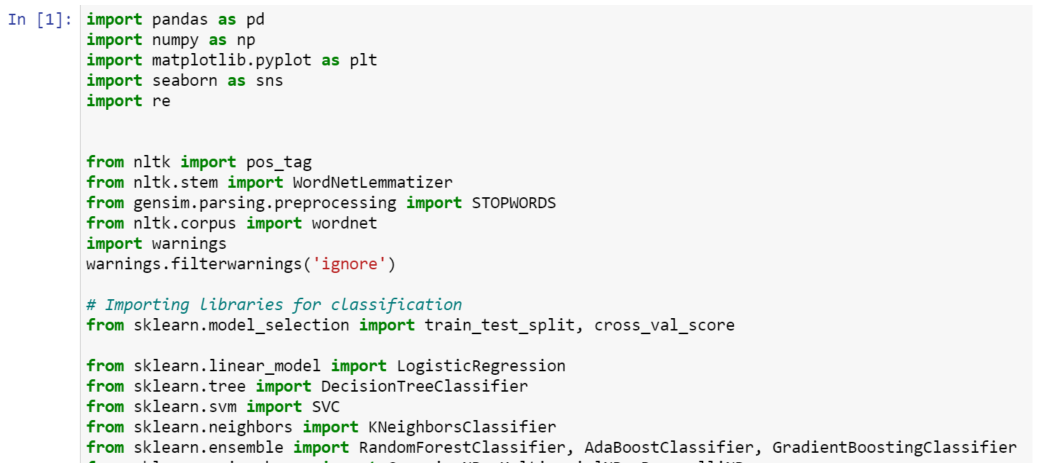


```
In [4]: features_info(df_train)

        Rows in dataset = 159571

        Columns in dataset = 8

        Features names =
         Index(['id', 'comment_text', 'malignant', 'highly_malignant', 'rude', 'threat',
               'abuse', 'loathe'],
              dtype='object')

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

So the data present in training data is both object and integer in nature. 2 columns are object dtypes and 6 columns are integer in dtypes.

• Data Preprocessing Done

Data preprocessing is the data mining technique that involves transforming raw data into an understandable data format. So in data preprocessing technique first step is import data and the libraries to be used in model building.



```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import re


        from nltk import pos_tag
        from nltk.stem import WordNetLemmatizer
        from gensim.parsing.preprocessing import STOPWORDS
        from nltk.corpus import wordnet
        import warnings
        warnings.filterwarnings('ignore')

        # Importing libraries for classification
        from sklearn.model_selection import train_test_split, cross_val_score

        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.svm import SVC
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
```

Loading Dataset

```
In [2]: # loading training dataset
        df_train = pd.read_csv(r"D:\Flip robo inter\data\Malignant-Comments-Classifier-Project--1-\Malignant Co
        df_train.head(5)
```

Out[2]:

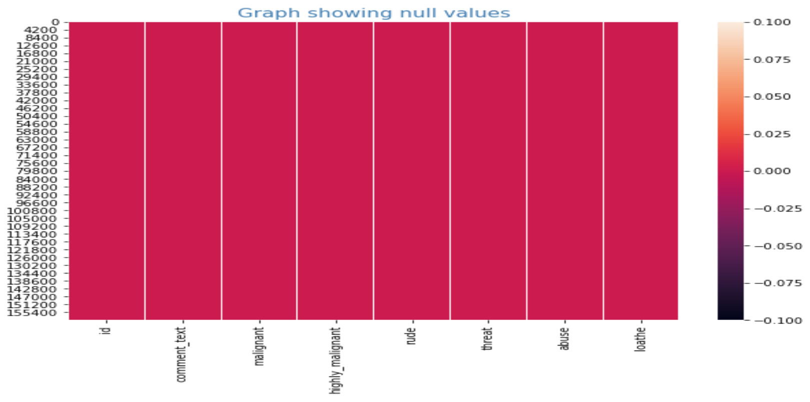
	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	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

Now after loading the dataset we check the missing values –

```
In [3]: # Creating function to see information of dataset
        def features_info(dataset):

            print("\nRows in dataset =",dataset.shape[0])
            print("\nColumns in dataset =",dataset.shape[1])
            print("\nFeatures names =\n",dataset.columns)
            print("\nDataset types : \n",dataset.dtypes)
            print("\nNull values in dataset :\n",dataset.isnull().sum())
            print("\nAny duplicated values :",dataset.duplicated().values.any(),"\n")
            for i in dataset.columns:
                print("Total unique values in {} = {}".format(i,dataset[i].nunique()))
            plt.figure(figsize=(10,7))
            sns.heatmap(dataset.isnull())
            plt.title("Graph showing null values",color='steelblue',fontsize=15)
            plt.show()
```

```
In [4]: features_info(df_train)
```

```
Null values in dataset :
 id                 0
comment_text       0
malignant          0
highly_malignant   0
rude               0
threat             0
abuse              0
loathe             0
dtype: int64
```



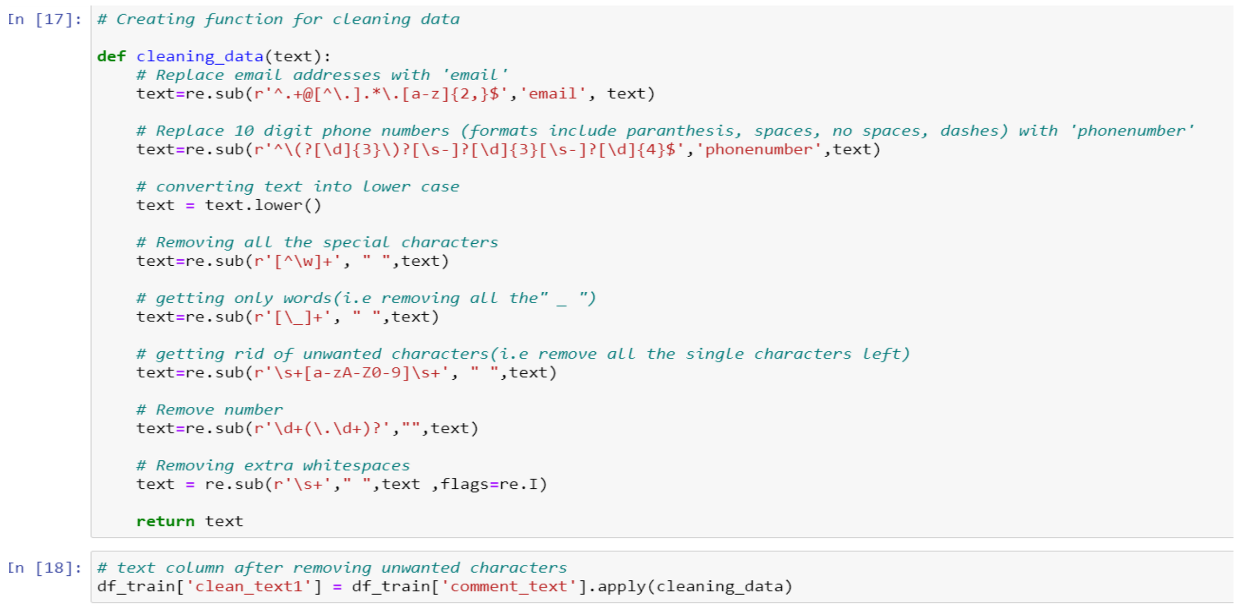
So there were **no null or missing values** in our dataset we move to next step of data cleaning –

In data cleaning we dop the ID column as it gives no information. After dropping it we created another feature "**negative_cmnts**" which shows the labelled data of positive and negative comments.

```
In [5]: # Dropping id feature as there is no use of it
        df_train.drop(columns=['id'],inplace=True)
```

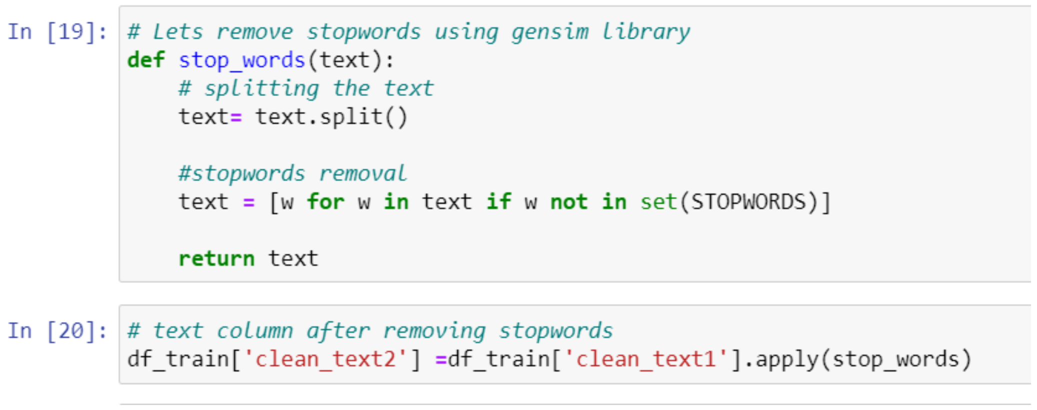
```
In [13]: # Let's create another feature whihc contains overall classification of all positive & negative comments
         df_train['negative_cmnts'] = df_train.iloc[:,1:].max(axis=1)
```

Now in cleaning the textual data we created a function to remove **unwanted space, punctuation , numbers, emails , phone numbers etc** and converted **upper case** letters into **lower case** and append the result into a new column.



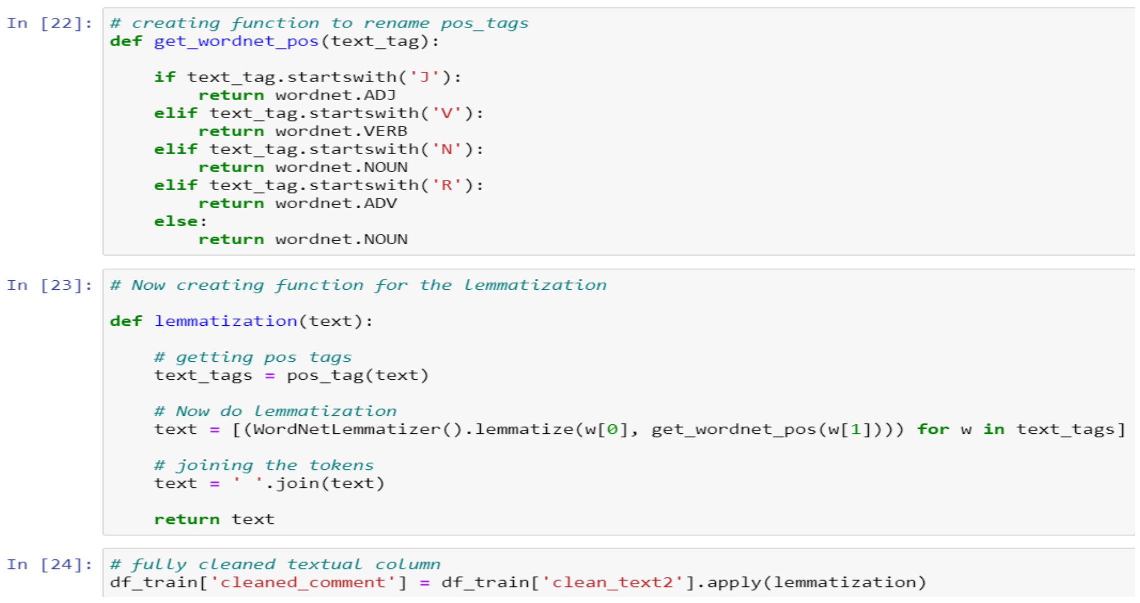
```
In [17]: # Creating function for cleaning data

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

             # Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
             text=re.sub(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$','phonenumber',text)

             # converting text into lower case
             text = text.lower()

             # Removing all the special characters
             text=re.sub(r'[^\w]+', " ",text)

             # getting only words(i.e removing all the" _ ")
             text=re.sub(r'[\_]+', " ",text)

             # getting rid of unwanted characters(i.e remove all the single characters left)
             text=re.sub(r'\s+[a-zA-Z0-9]\s+', " ",text)

             # Remove number
             text=re.sub(r'\d+(\.\d+)?',"",text)

             # Removing extra whitespaces
             text = re.sub(r'\s+'," ",text ,flags=re.I)

             return text

In [18]: # text column after removing unwanted characters
         df_train['clean_text1'] = df_train['comment_text'].apply(cleaning_data)
```

After the removal of unwanted notations we moved to remove **stopwords** from our dataset. For removing them we created another function named stop_words and append the text into another new column.



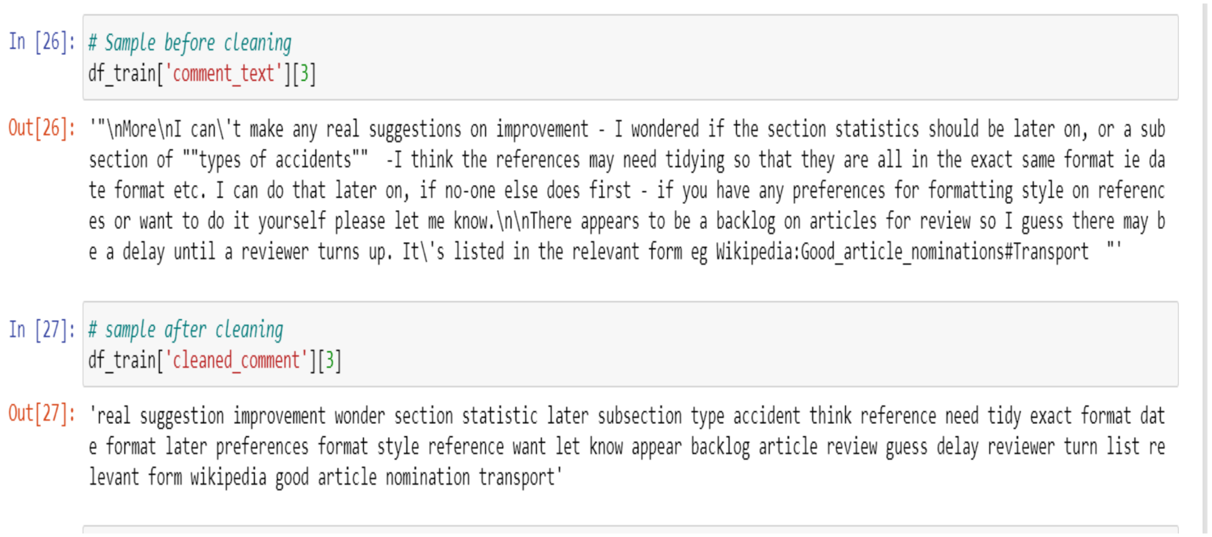
```
In [19]: # Lets remove stopwords using gensim library
         def stop_words(text):
             # splitting the text
             text= text.split()

             #stopwords removal
             text = [w for w in text if w not in set(STOPWORDS)]

             return text

In [20]: # text column after removing stopwords
         df_train['clean_text2'] =df_train['clean_text1'].apply(stop_words)
```

After removing all the unnecessary words or numbers we converted the words into their simpler form using the **Lemmatization process.** In this process we defined two function , first one will tag the words into proper format and the other function will convert them into simpler form using the tagged alphabet.



```
In [22]: # creating function to rename pos_tags
         def get_wordnet_pos(text_tag):

             if text_tag.startswith('J'):
                 return wordnet.ADJ
             elif text_tag.startswith('V'):
                 return wordnet.VERB
             elif text_tag.startswith('N'):
                 return wordnet.NOUN
             elif text_tag.startswith('R'):
                 return wordnet.ADV
             else:
                 return wordnet.NOUN

In [23]: # Now creating function for the lemmatization

         def lemmatization(text):

             # getting pos tags
             text_tags = pos_tag(text)

             # Now do lemmatization
             text = [(WordNetLemmatizer().lemmatize(w[0], get_wordnet_pos(w[1]))) for w in text_tags]

             # joining the tokens
             text = ' '.join(text)

             return text

In [24]: # fully cleaned textual column
         df_train['cleaned_comment'] = df_train['clean_text2'].apply(lemmatization)
```

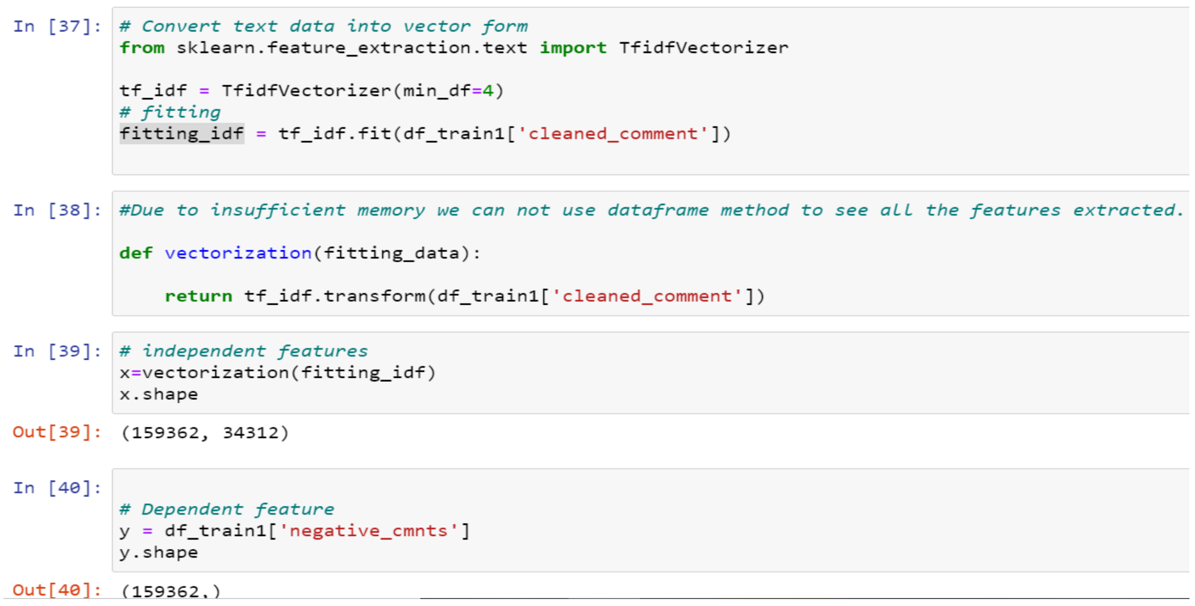
Now we took a random sample from our dataset and compared the text of before and after the treatment.



```
In [26]: # Sample before cleaning
         df_train['comment_text'][3]

Out[26]: '"\nMore\nI can\'t make any real suggestions on improvement - I wondered if the section statistics should be later on, or a sub
         section of ""types of accidents""  -I think the references may need tidying so that they are all in the exact same format ie da
         te format etc. I can do that later on, if no-one else does first - if you have any preferences for formatting style on referenc
         es or want to do it yourself please let me know.\n\nThere appears to be a backlog on articles for review so I guess there may b
         e a delay until a reviewer turns up. It\'s listed in the relevant form eg Wikipedia:Good_article_nominations#Transport  "'

In [27]: # sample after cleaning
         df_train['cleaned_comment'][3]

Out[27]: 'real suggestion improvement wonder section statistic later subsection type accident think reference need tidy exact format dat
         e format later preferences format style reference want let know appear backlog article review guess delay reviewer turn list re
         levant form wikipedia good article nomination transport'
```

Encoding the categorical data (Feature Extraction and scaling)

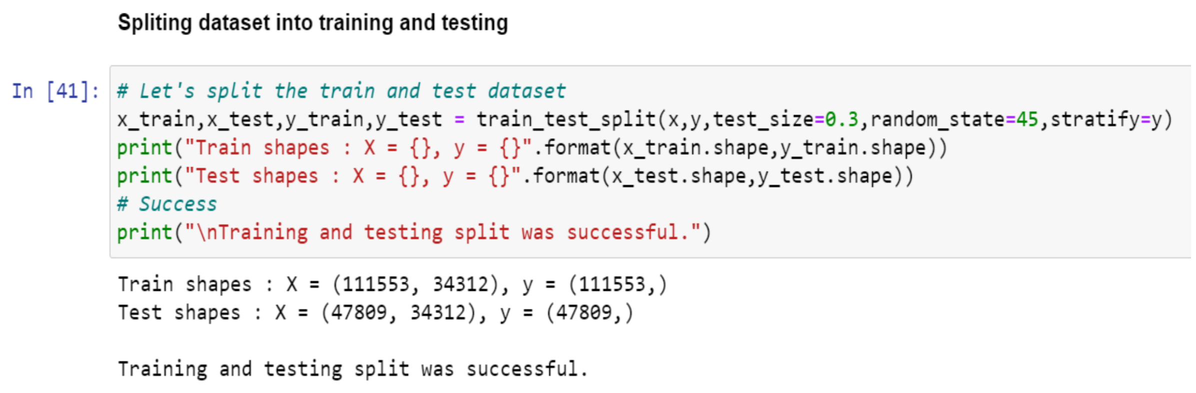
As of now we had cleaned the data , now another major step towards building a model is to convert the textual data into numerical form because our algorithms understand only the numerical data. So for converting into numerical or vector form we used Tf-Idf Vectorizer technique which converted the textual data into the vectors using the terms frequency method.



```
In [37]: # Convert text data into vector form
         from sklearn.feature_extraction.text import TfidfVectorizer

         tf_idf = TfidfVectorizer(min_df=4)
         # fitting
         fitting_idf = tf_idf.fit(df_train1['cleaned_comment'])

In [38]: #Due to insufficient memory we can not use dataframe method to see all the features extracted.

         def vectorization(fitting_data):

             return tf_idf.transform(df_train1['cleaned_comment'])

In [39]: # independent features
         x=vectorization(fitting_idf)
         x.shape

Out[39]: (159362, 34312)

In [40]:
         # Dependent feature
         y = df_train1['negative_cmnts']
         y.shape

Out[40]: (159362,)
```

Splitting the dataset –

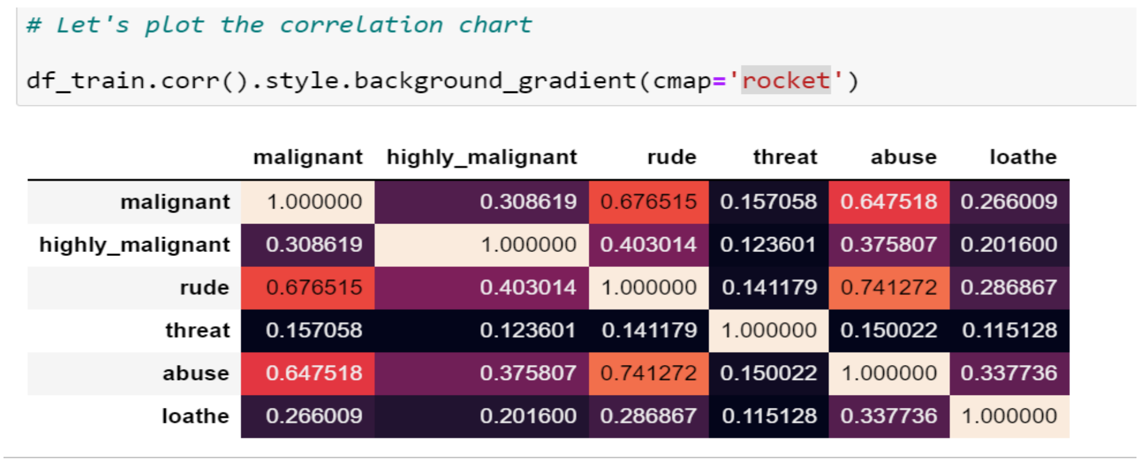
The last step for data-preprocessing is splitting the dataset into training and testing dataset Using the train_test_split model selection method we converted the dataset into training and testing.



Spliting dataset into training and testing

```
In [41]: # Let's split the train and test dataset
         x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=45,stratify=y)
         print("Train shapes : X = {}, y = {}".format(x_train.shape,y_train.shape))
         print("Test shapes : X = {}, y = {}".format(x_test.shape,y_test.shape))
         # Success
         print("\nTraining and testing split was successful.")

         Train shapes : X = (111553, 34312), y = (111553,)
         Test shapes : X = (47809, 34312), y = (47809,)

         Training and testing split was successful.
```

- ## Data Inputs- Logic- Output Relationships



```
# Let's plot the correlation chart

df_train.corr().style.background_gradient(cmap='rocket')
```

	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

All the harmful comments present in dataset are having positive correlation with each others. Rude and abuse are highly correlated.

- <u>Hardware and Software Requirements and Tools Used</u>

There is no such requirement for hardware ,but I have used intel i5 8th generation processor with 6 GB Ram.

Software: Jupyter Notebook (Anaconda 3)

Language : Python 3.9

Libraries used in project:

a. Pandas

b. Numpy

c. Matplotlib

d. Seaborn

e. Sklearn

Model/s Development and Evaluation

- <u>Identification of possible problem-solving approaches (methods)</u>

In this project there were 5-6 features which defines the type of comment like malignant , hate , abuse, threat, loathe but we created another feature named as “negative_cmnts” which is combined of all the above features and contains the labelled data into the format of 0 and 1 where 0 represents “NO” and 1 represents “Yes”.

As we have labelled data into our target feature which is the case of classification method. So we are going to use algorithms based on classification.

- <u>Testing of Identified Approaches (Algorithms)</u>

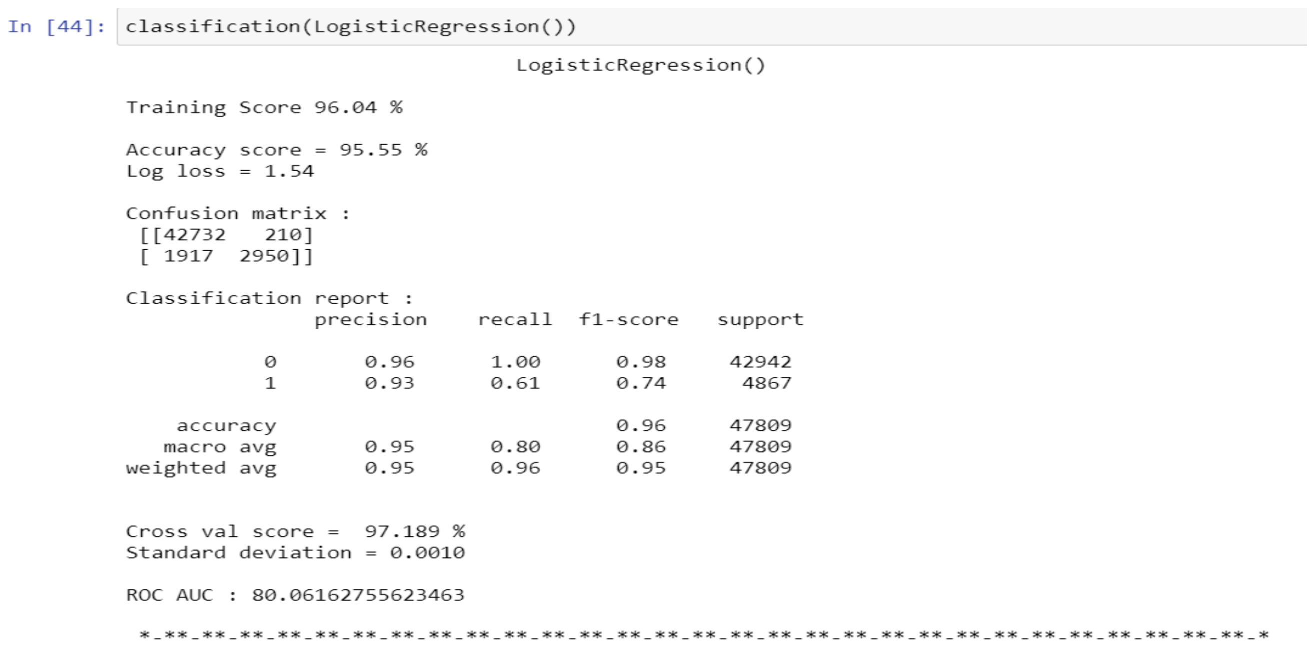
Based on the classification approach we are going to use following approaches :

I. Logistic Regression
II. Decision Tree Classifier
III. RandomForest Classifier
IV. AdaBoost Classifier
V. GradientBoosting Classifier

In NLP we use Naïve Bayes algorithms mostly but due to our systems we faced memory error to deal with them.

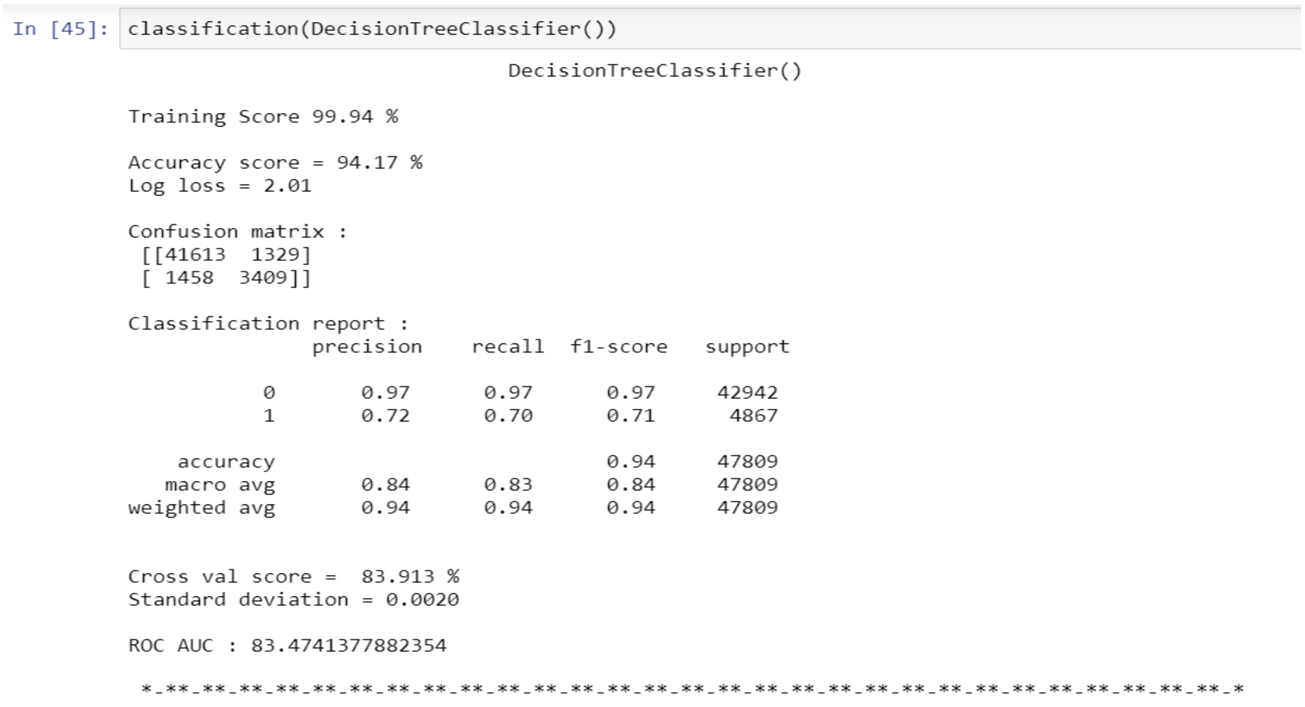
• Run and Evaluate selected models

I. Logistic Regresison -



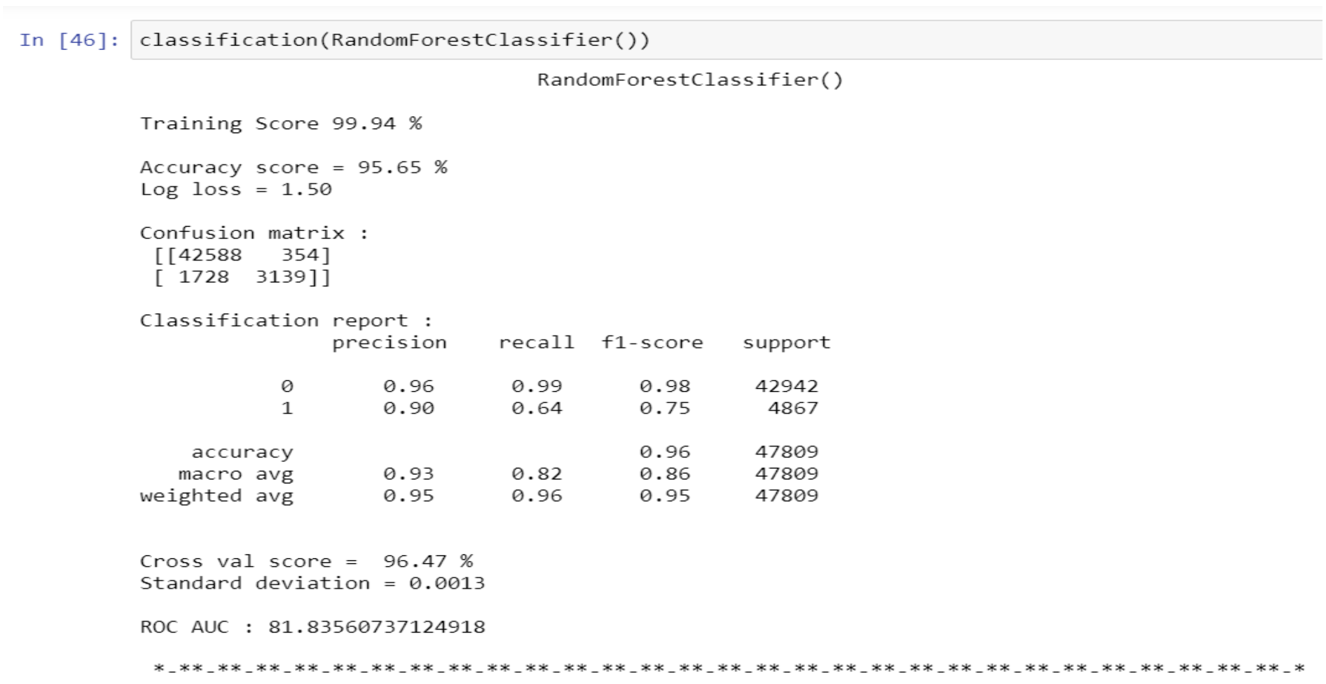
```
In [44]: classification(LogisticRegression())
                                    LogisticRegression()

         Training Score 96.04 %

         Accuracy score = 95.55 %
         Log loss = 1.54

         Confusion matrix :
          [[42732   210]
          [ 1917  2950]]

         Classification report :
                       precision    recall  f1-score   support

                    0       0.96      1.00      0.98     42942
                    1       0.93      0.61      0.74      4867

             accuracy                           0.96     47809
            macro avg       0.95      0.80      0.86     47809
         weighted avg       0.95      0.96      0.95     47809


         Cross val score =  97.189 %
         Standard deviation = 0.0010

         ROC AUC : 80.06162755623463

          *_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_*
```

II. DecisonTree Classifier –



```
In [45]: classification(DecisionTreeClassifier())
                                          DecisionTreeClassifier()

         Training Score 99.94 %

         Accuracy score = 94.17 %
         Log loss = 2.01

         Confusion matrix :
          [[41613  1329]
          [ 1458  3409]]

         Classification report :
                        precision    recall  f1-score   support

                   0       0.97      0.97      0.97     42942
                   1       0.72      0.70      0.71      4867

             accuracy                           0.94     47809
            macro avg       0.84      0.83      0.84     47809
         weighted avg       0.94      0.94      0.94     47809


         Cross val score =  83.913 %
         Standard deviation = 0.0020

         ROC AUC : 83.4741377882354

          *_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_*
```

III. RandomForest Classifier –



```
In [46]: classification(RandomForestClassifier())
                                          RandomForestClassifier()

         Training Score 99.94 %

         Accuracy score = 95.65 %
         Log loss = 1.50

         Confusion matrix :
          [[42588   354]
          [ 1728  3139]]

         Classification report :
                        precision    recall  f1-score   support

                   0       0.96      0.99      0.98     42942
                   1       0.90      0.64      0.75      4867

             accuracy                           0.96     47809
            macro avg       0.93      0.82      0.86     47809
         weighted avg       0.95      0.96      0.95     47809


         Cross val score =  96.47 %
         Standard deviation = 0.0013

         ROC AUC : 81.83560737124918

          *_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_*
```

IV. AdaBoost Classifier –

```
In [47]: classification(AdaBoostClassifier())
```

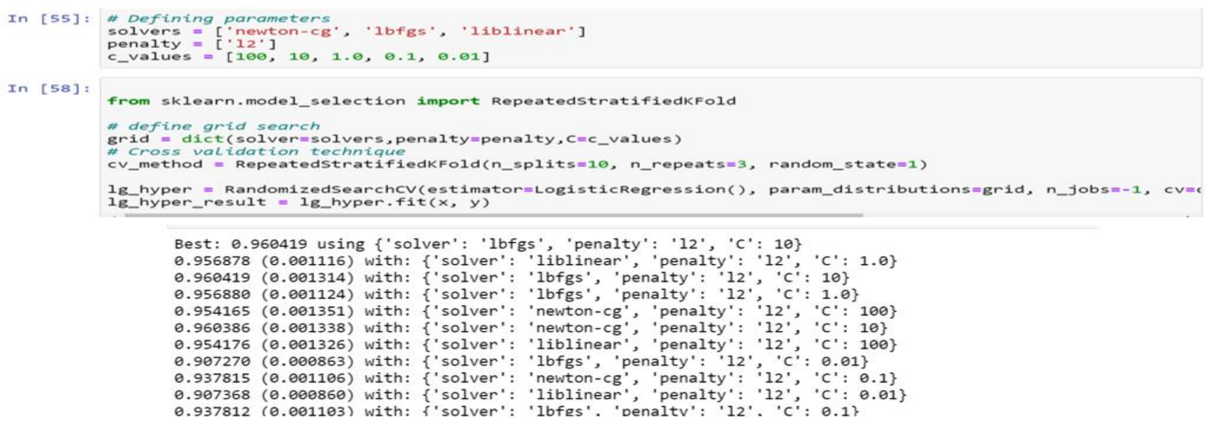
```
                                      AdaBoostClassifier()

Training Score 94.78 %

Accuracy score = 94.62 %
Log loss = 1.86

Confusion matrix :
 [[42507   435]
 [ 2138  2729]]

Classification report :
               precision    recall  f1-score   support

           0       0.95      0.99      0.97     42942
           1       0.86      0.56      0.68      4867

    accuracy                           0.95     47809
   macro avg       0.91      0.78      0.83     47809
weighted avg       0.94      0.95      0.94     47809


Cross val score =  89.769 %
Standard deviation = 0.0031

ROC AUC : 77.52925384028919

 *_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_*
```

V. GradientBoosting Classifier –

```
In [48]: classification(GradientBoostingClassifier())
```

```
                                      GradientBoostingClassifier()

Training Score 94.37 %

Accuracy score = 94.15 %
Log loss = 2.02

Confusion matrix :
 [[42815   127]
 [ 2670  2197]]

Classification report :
               precision    recall  f1-score   support

           0       0.94      1.00      0.97     42942
           1       0.95      0.45      0.61      4867

    accuracy                           0.94     47809
   macro avg       0.94      0.72      0.79     47809
weighted avg       0.94      0.94      0.93     47809


Cross val score =  90.066 %
Standard deviation = 0.0026

ROC AUC : 72.42249801594473

 *_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_*
```

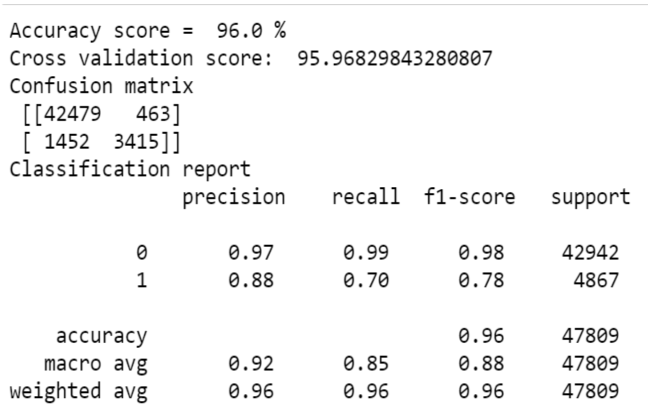
Logistic Regression Hypertunning



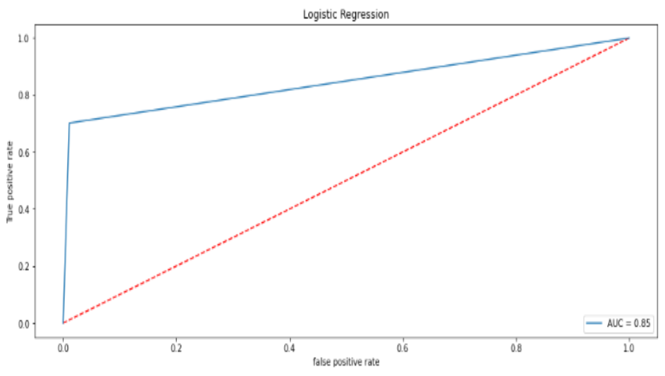
```
In [55]: # Defining parameters
         solvers = ['newton-cg', 'lbfgs', 'liblinear']
         penalty = ['l2']
         c_values = [100, 10, 1.0, 0.1, 0.01]
```

```
In [58]:
         from sklearn.model_selection import RepeatedStratifiedKFold

         # define grid search
         grid = dict(solver=solvers,penalty=penalty,C=c_values)
         # Cross validation technique
         cv_method = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

         lg_hyper = RandomizedSearchCV(estimator=LogisticRegression(), param_distributions=grid, n_jobs=-1, cv=c
         lg_hyper_result = lg_hyper.fit(x, y)
```

```
Best: 0.960419 using {'solver': 'lbfgs', 'penalty': 'l2', 'C': 10}
0.956878 (0.001116) with: {'solver': 'liblinear', 'penalty': 'l2', 'C': 1.0}
0.960419 (0.001314) with: {'solver': 'lbfgs', 'penalty': 'l2', 'C': 10}
0.956880 (0.001124) with: {'solver': 'lbfgs', 'penalty': 'l2', 'C': 1.0}
0.954165 (0.001351) with: {'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}
0.960386 (0.001338) with: {'solver': 'newton-cg', 'penalty': 'l2', 'C': 10}
0.954176 (0.001326) with: {'solver': 'liblinear', 'penalty': 'l2', 'C': 100}
0.907270 (0.000863) with: {'solver': 'lbfgs', 'penalty': 'l2', 'C': 0.01}
0.937815 (0.001106) with: {'solver': 'newton-cg', 'penalty': 'l2', 'C': 0.1}
0.907368 (0.000860) with: {'solver': 'liblinear', 'penalty': 'l2', 'C': 0.01}
0.937812 (0.001103) with: {'solver': 'lbfgs', 'penalty': 'l2', 'C': 0.1}
```



```
Accuracy score =  96.0 %
Cross validation score:  95.96829843280807
Confusion matrix
 [[42479   463]
 [ 1452  3415]]
Classification report
               precision    recall  f1-score   support

           0       0.97      0.99      0.98     42942
           1       0.88      0.70      0.78      4867

    accuracy                           0.96     47809
   macro avg       0.92      0.85      0.88     47809
weighted avg       0.96      0.96      0.96     47809
```



• Key Metrics for success in solving problem under consideration

For solving the problems and understanding the result of each algorithms we have used a lot of metrics :

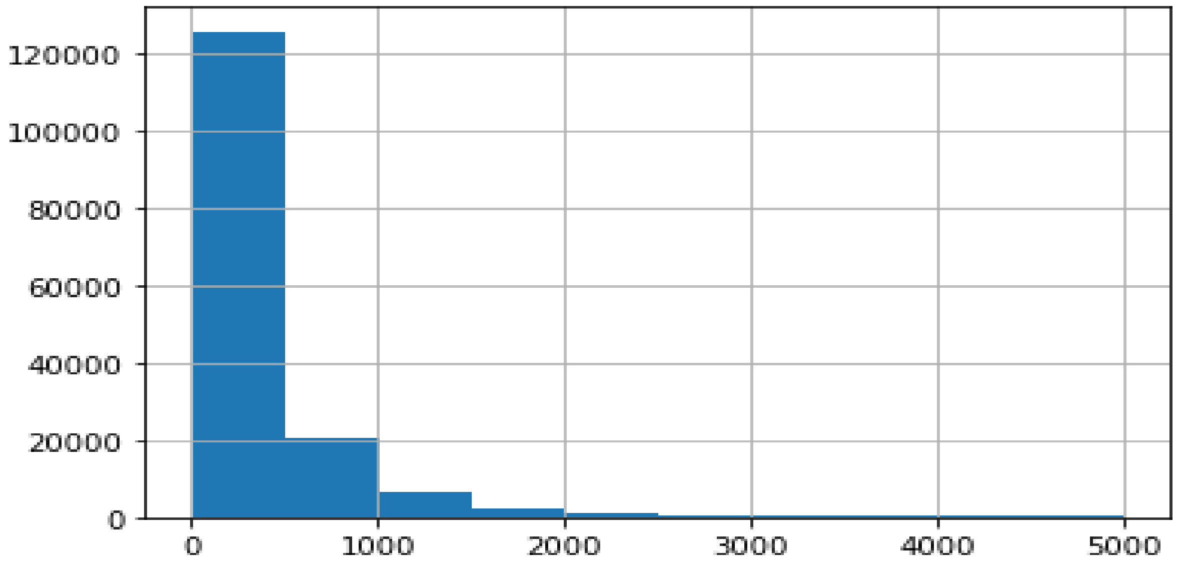
a) Accuracy Score
b) Classification Report
c) Confusion Matrix
d) Log Loss
e) Roc-Auc

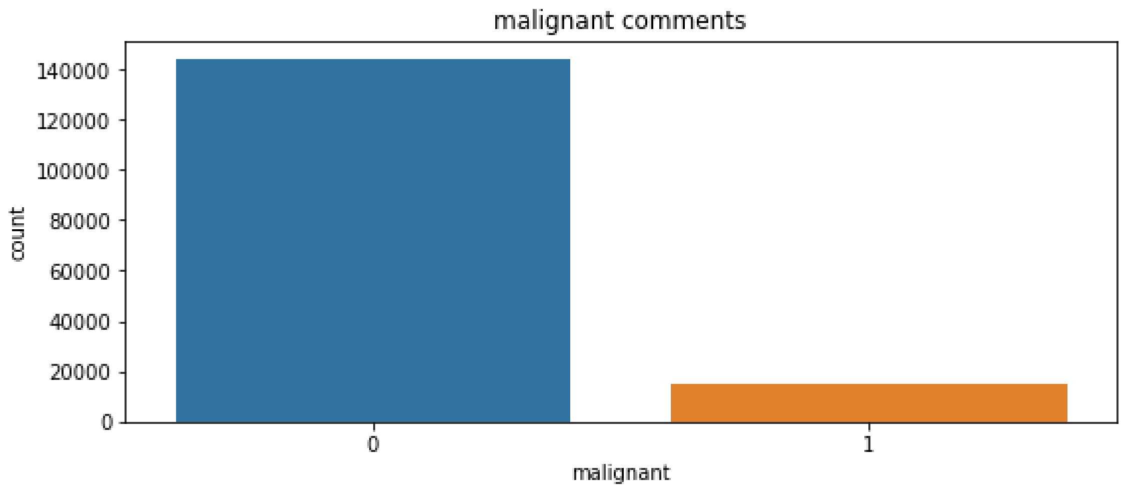
• Visualizations

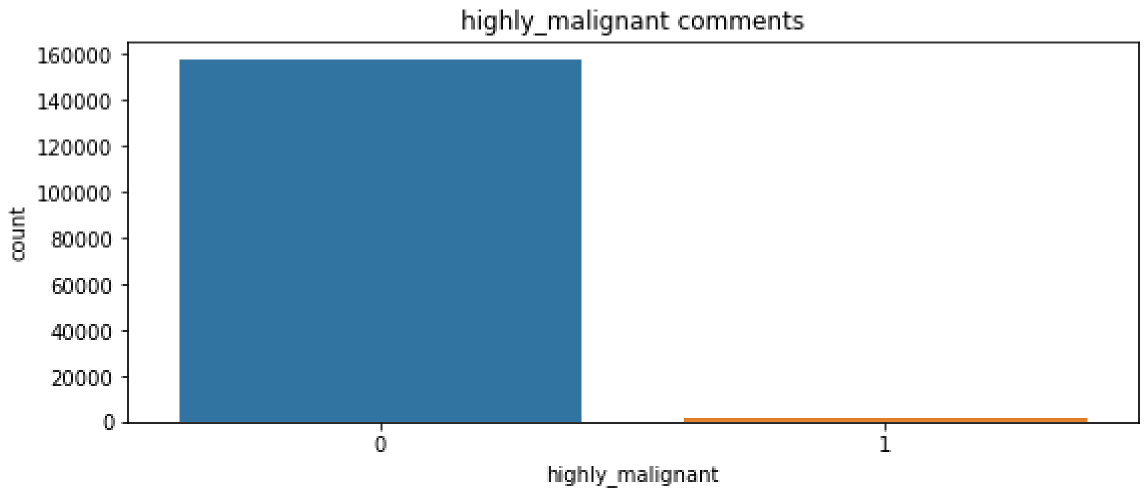
For the Visualization we have used Matplotlib and Seaborn library to plot the numerical data into graphs –

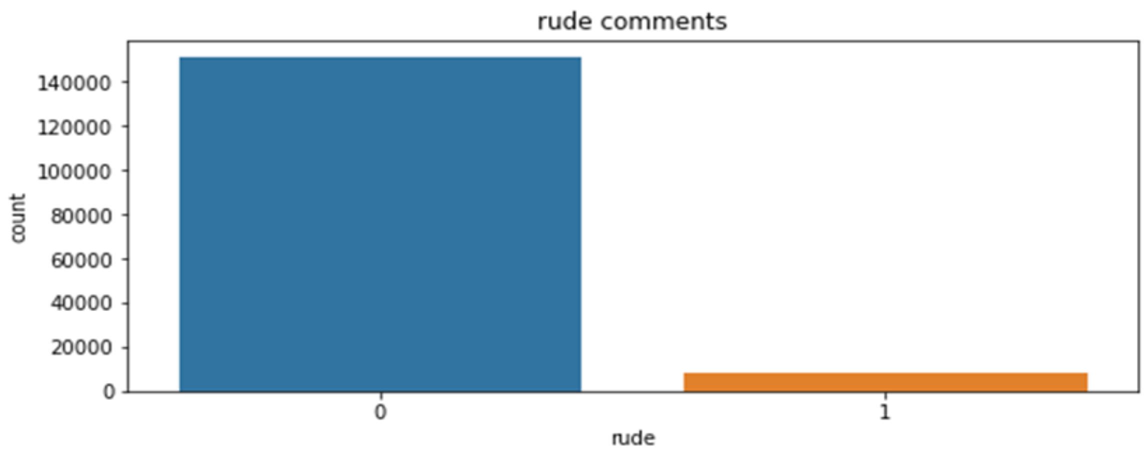
```
In [8]: # Distribution plot of comment length before cleaning
        cmnt_len_train.hist()
```

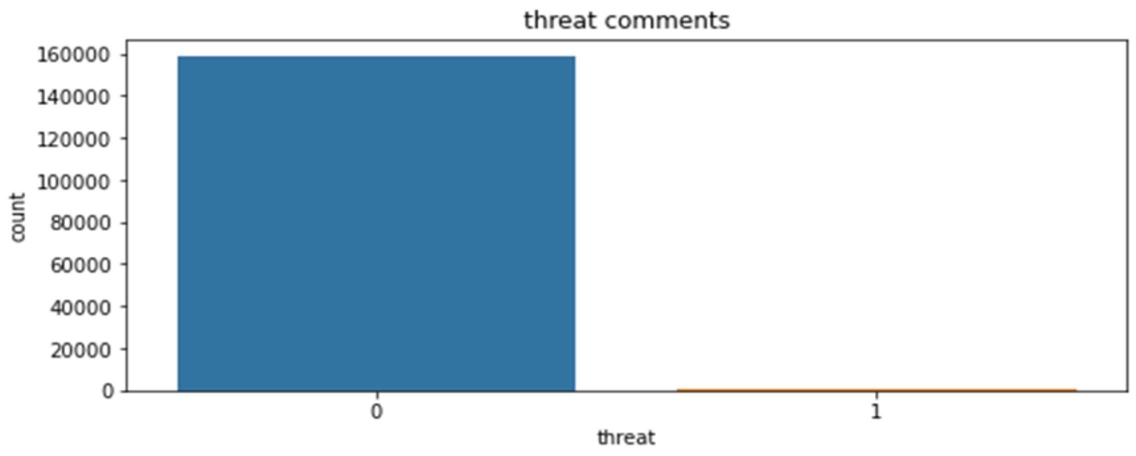
```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x23115cb82b0>
```

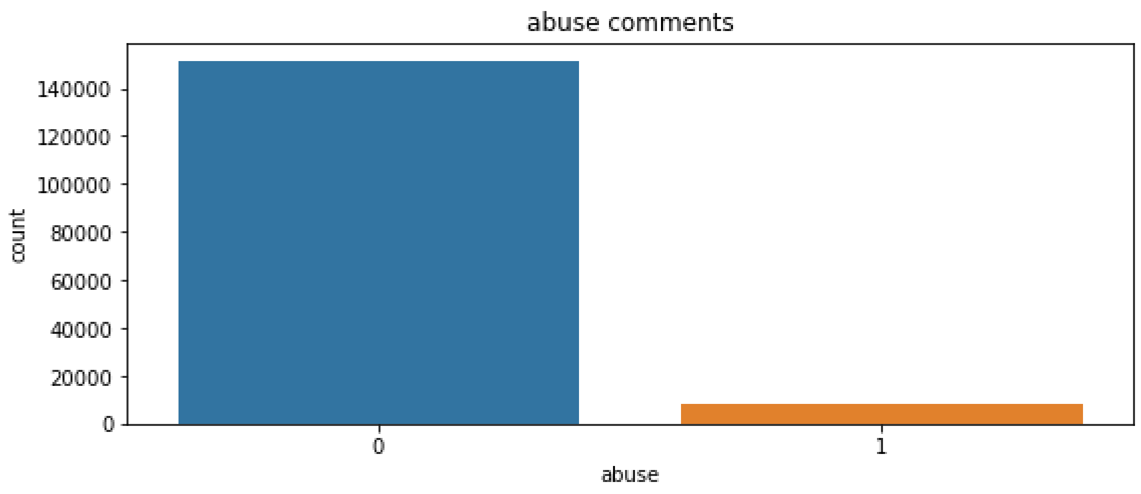


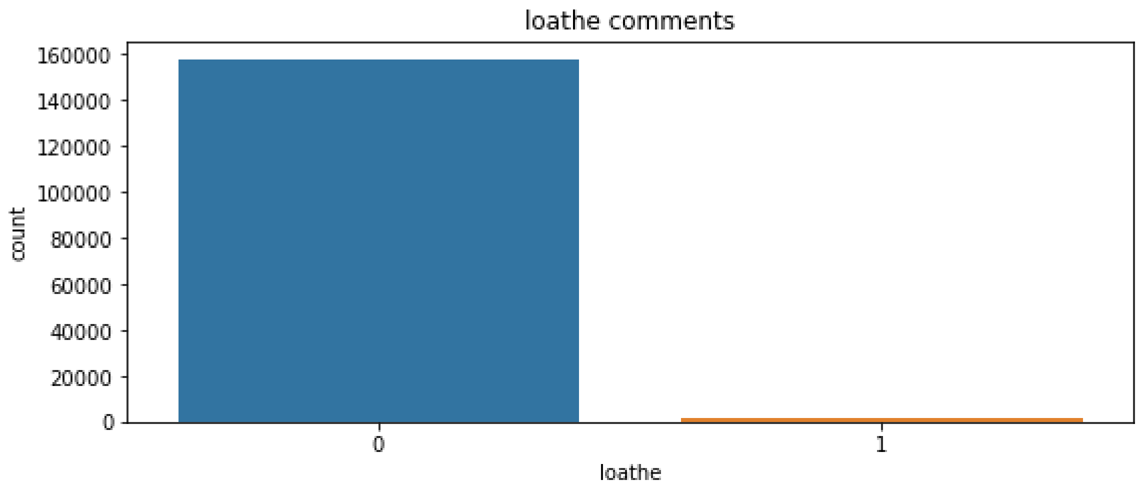


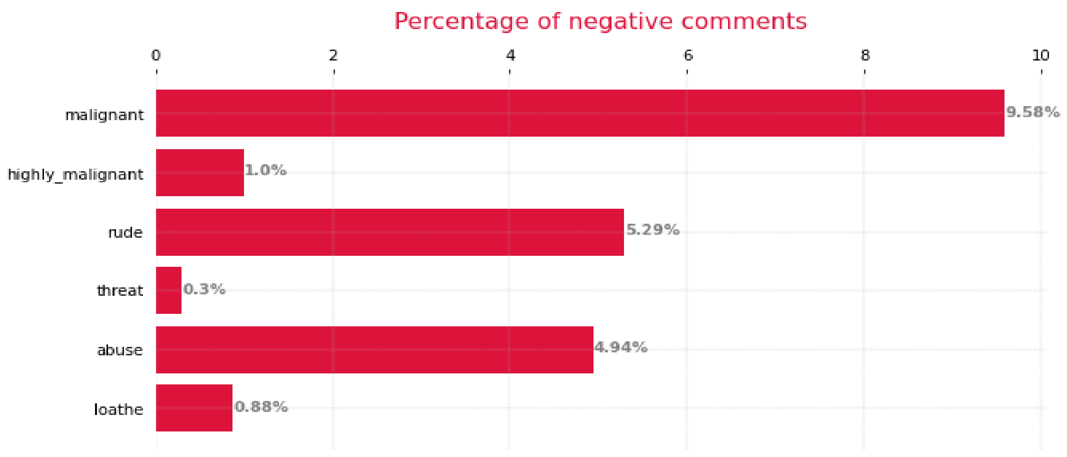


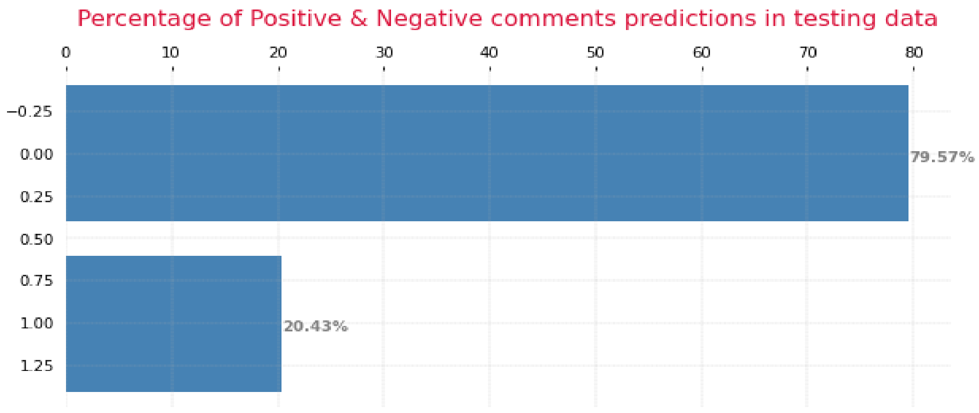

rude comments
count
0
20000
40000
60000
80000
100000
120000
140000
0
1
rude

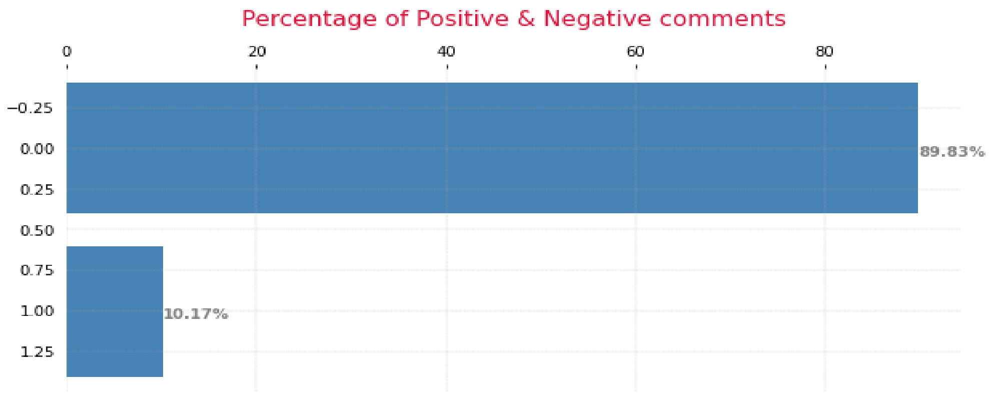

threat comments
count
0
20000
40000
60000
80000
100000
120000
140000
160000
0
1
threat


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

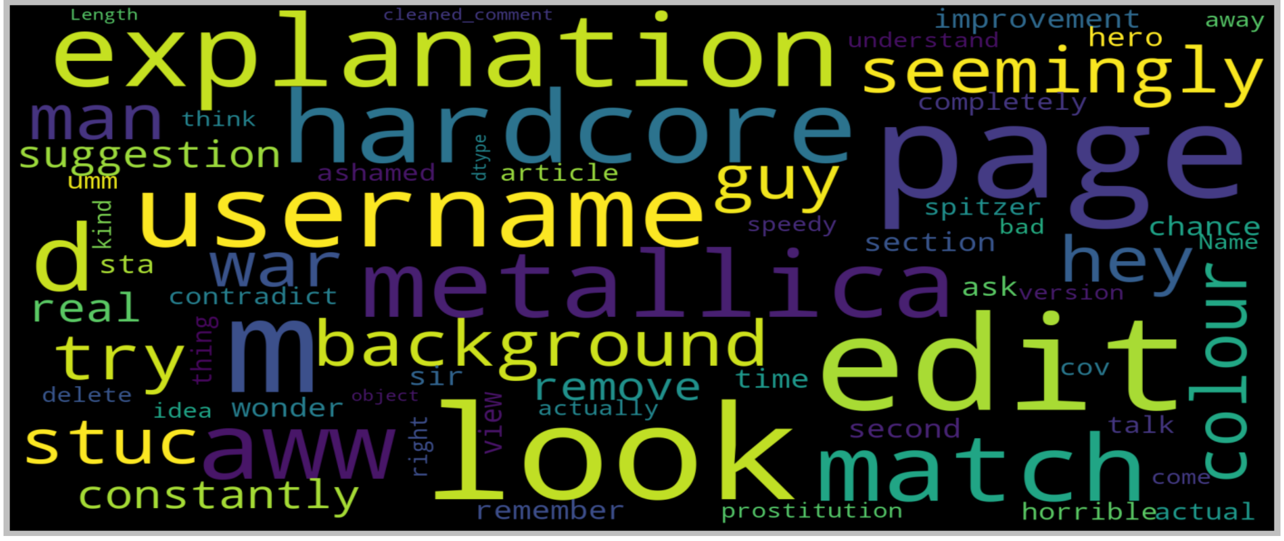

loathe comments
count
0
20000
40000
60000
80000
100000
120000
140000
160000
0
1
loathe


Percentage of negative comments
0
2
4
6
8
10
malignant
highly_malignant
rude
threat
abuse
loathe
9.58%
1.0%
5.29%
0.3%
4.94%
0.88%

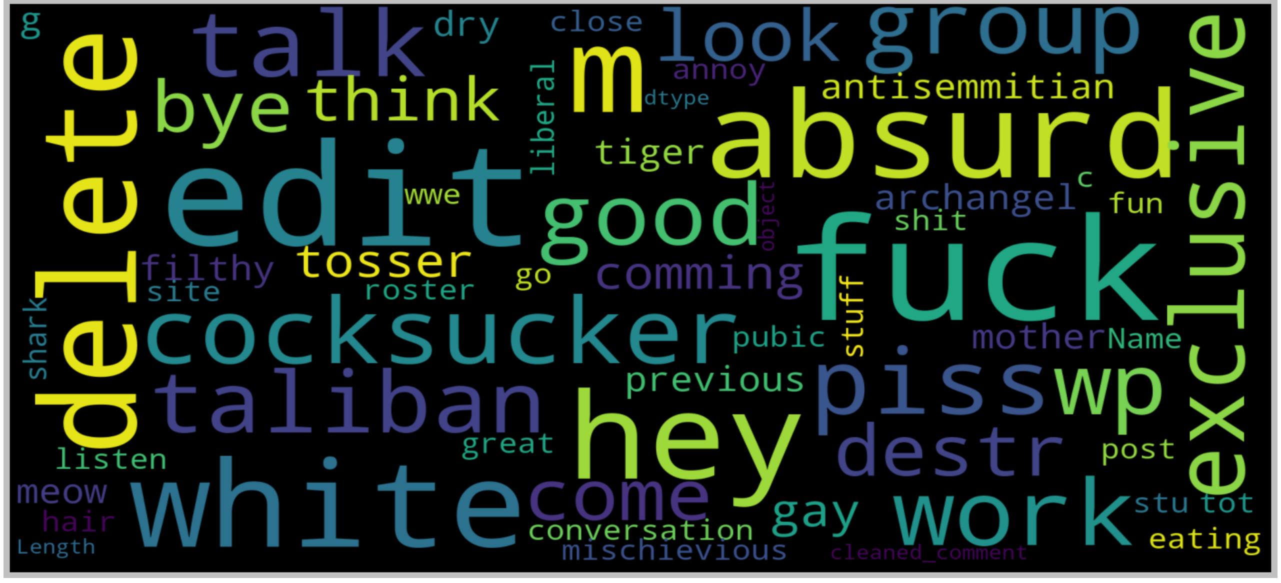

Percentage of Positive & Negative comments predictions in testing data
0
10
20
30
40
50
60
70
80
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
79.57%
20.43%


Percentage of Positive & Negative comments
0
20
40
60
80
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
89.83%
10.17%

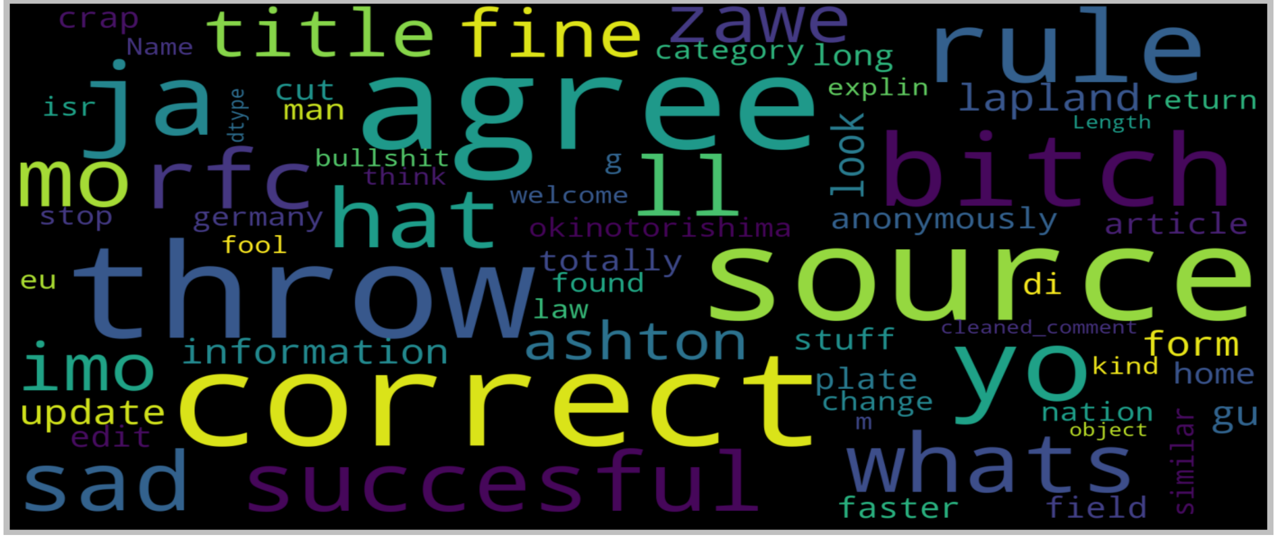
Words Present in Postive Comments



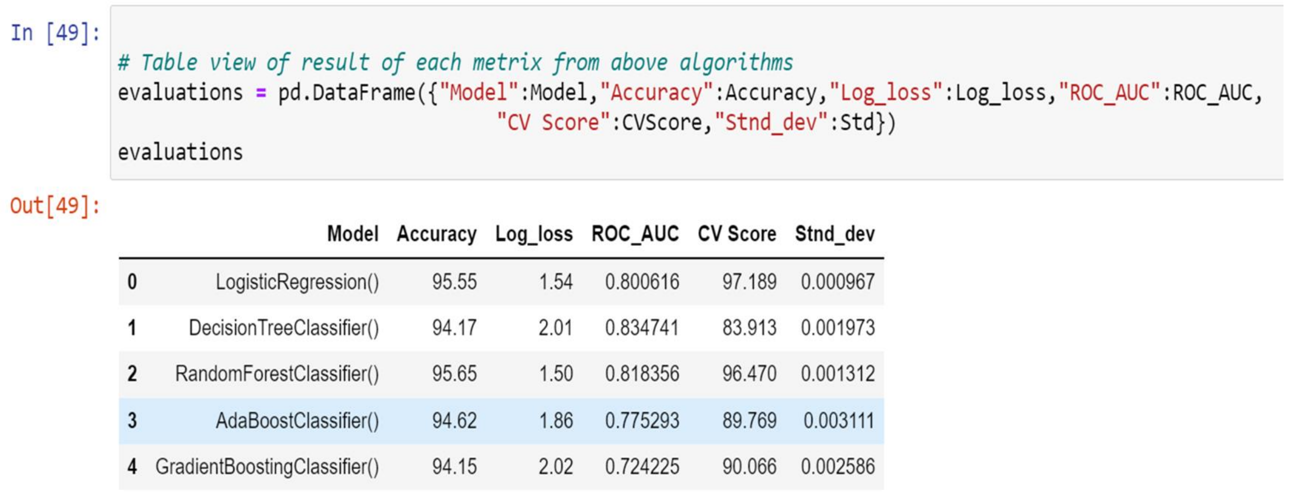
Words present in Negative Comments



Words Present in Testing dataset comments



• Interpretation of the Results



In [49]:

```
# Table view of result of each metrix from above algorithms
evaluations = pd.DataFrame({"Model":Model,"Accuracy":Accuracy,"Log_loss":Log_loss,"ROC_AUC":ROC_AUC,
                            "CV Score":CVScore,"Stnd_dev":Std})
evaluations
```

Out[49]:

	Model	Accuracy	Log_loss	ROC_AUC	CV Score	Stnd_dev
0	LogisticRegression()	95.55	1.54	0.800616	97.189	0.000967
1	DecisionTreeClassifier()	94.17	2.01	0.834741	83.913	0.001973
2	RandomForestClassifier()	95.65	1.50	0.818356	96.470	0.001312
3	AdaBoostClassifier()	94.62	1.86	0.775293	89.769	0.003111
4	GradientBoostingClassifier()	94.15	2.02	0.724225	90.066	0.002586

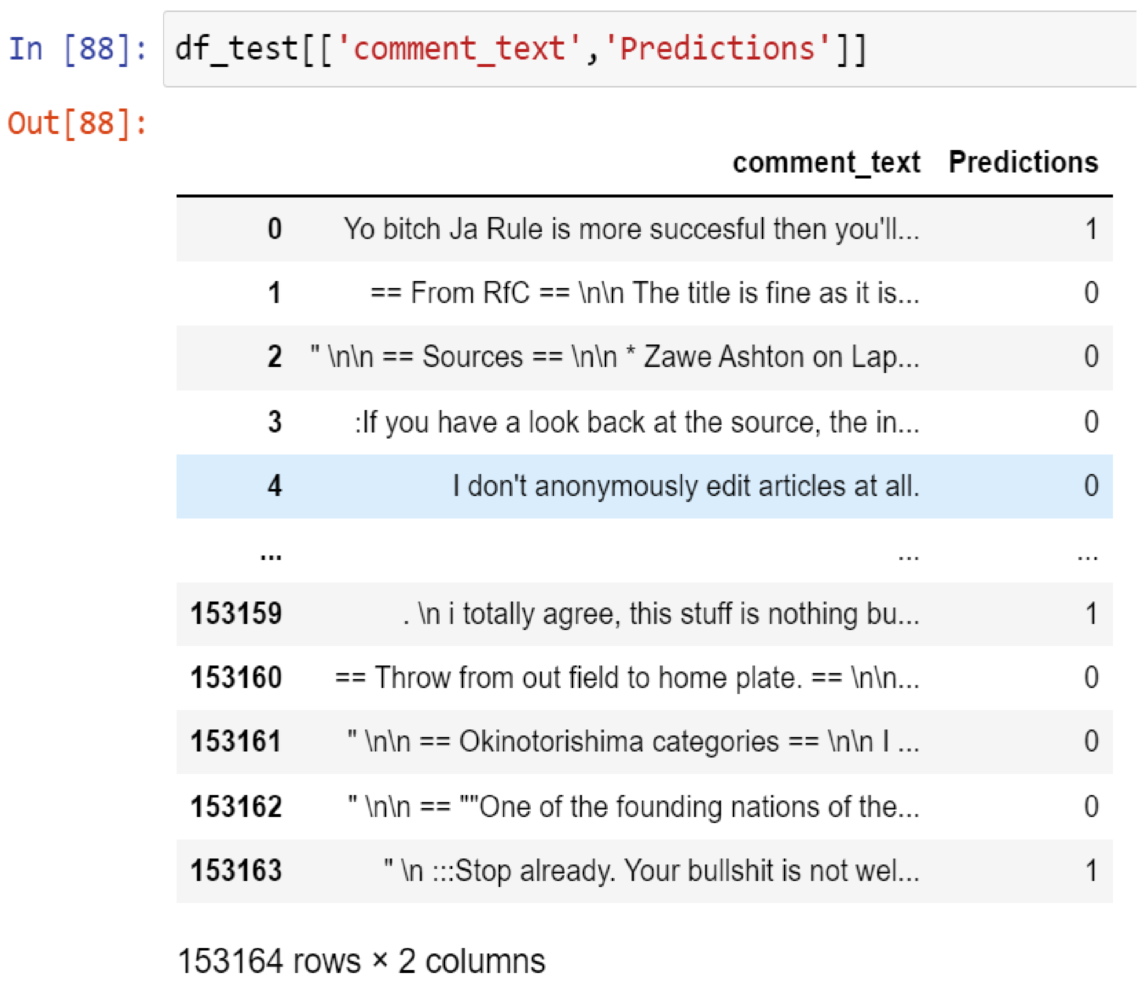
As our target feature is imbalanced which means that alone accuracy score will not give best results but instead accuracy we are using the classification report log loss score , Roc-Auc score and confusion matrix to find the best algorthim which is above all.

So we selected Logistic Regression and RandomForest Classifier based on above condition and use RandomizedSearchCV for hyper tunning the parameters of these 2 selected algorithms . After hyper tunning the parameters we selected the Logistic Regression algorithm as it give upto 85% score of Roc-Auc and losgg loss of 1.38 far better than RandomForest Classifier whose log loss was above 2.0 and again the classification report also tells that it is better than RandomForest Classifier.

Finalising the Model

- So after tunning the parameters of both the selected algorithms we are finalised Logistic Regression for our model. As the ROC AUC curve and the score of Logg loss is far better than Random Forest Classifier.
- Logistic Regression gives us Log loss of 1.38 and the Area under the curve is 85% with an Accuracy of 96%.

• Predictions

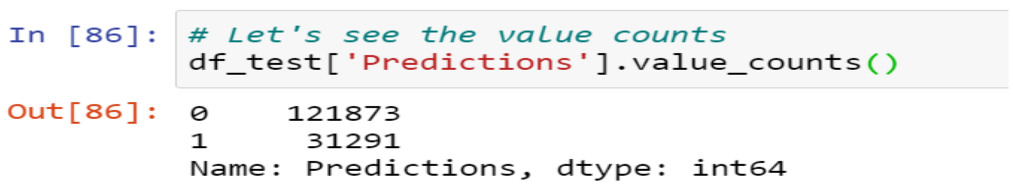


```
In [88]: df_test[['comment_text','Predictions']]
Out[88]:
```

	comment_text	Predictions
0	Yo bitch Ja Rule is more succesful then you'll...	1
1	== From RfC == \n\n The title is fine as it is...	0
2	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	0
3	:If you have a look back at the source, the in...	0
4	I don't anonymously edit articles at all.	0
...	...	...
153159	. \n i totally agree, this stuff is nothing bu...	1
153160	== Throw from out field to home plate. == \n\n...	0
153161	" \n\n == Okinotorishima categories == \n\n I ...	0
153162	" \n\n == ""One of the founding nations of the...	0
153163	" \n :::Stop already. Your bullshit is not wel...	1

153164 rows × 2 columns

These are the predictions rom our testing dataset.



```
In [86]: # Let's see the value counts
         df_test['Predictions'].value_counts()

Out[86]: 0     121873
         1      31291
         Name: Predictions, dtype: int64
```

CONCLUSION

• Key Findings and Conclusions of the Study

The conclusion for our study :-

a) In training dataset we have only 10% of data which is spreading hate on social media.
b) In this 10% data most of the comments are malignant ,rude or abuse.
c) After using the wordcloud we find that there are so many abusive words present in the negative comments. While in positive comments there is no use of such comments.
d) Some of the comments are very long while some are very short.

- ## Learning Outcomes of the Study in respect of Data Science

From this project we learned a lot. Gains new techniques and ways to deal with uncleaned data. Find a solution to deal with multiple target features. Tools used for visualizations gives a better understanding of dataset. We have used a lot of algorithms and find that in the classification problem where we have only two labels , Logistic Regression gives better results compared to others.

But due to our system we could not use algorithms which gives much better results in NLP project like GaussinaNB, MultinomailNM. We also used googlecolab and some pipelines techniques but none of them worked here and also it was too much time consuming.

- ## Limitations of this work and Scope for Future Work

This project was amazing to work on , it creates new ideas to think about but there were some limitations in this project like unbalanced dataset, multiple target features. To overcome these limitations we have to use balanced datset so that during the training of dataset our algorithm will not give biased result.

Thankyou