MALIGNANT COMMENTS CLASSIFIER PROJECT

Prepared by:
Vikash Kumar Singh

SME Name:
Gulshana Chaudhary

ACKNOWLEDGMENT

I would like to convey my heartfelt gratitude to Flip Robo Technologies for providing me with this wonderful opportunity to work on a Machine Learning project "MALIGNANT COMMENTS CLASSIFICATION" and also want to Thank my SME, **Gulshana Chaudhary** for providing the dataset and guiding me to complete this project. This project would not have been accomplished without their help and insights.

I would also like to thank my academic "Data Trained Education" and their team who has helped me to learn Machine Learning.

I also references from some websites which are- https://www.youtube.com https://www.kaggle.com ,
https://www.github.com , https://stackoverflow.com

Working on this project was an incredible experience as I learnt more from this Project during completion.

INTRODUCTION

1. Business Problem Framing

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 cyber bullying.

2. Conceptual Background of the Domain Problem

There has been a remarkable increase in the cases of cyber bullying 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.

3. Review of Literature

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

Online hate, described as abusive language, aggression, cyber bullying, 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 behavior.

4. Motivation for the Problem Undertaken

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 user. This means that insults to third parties such as celebrities will be tagged as inoffensive, but “u are an idiot” is clearly offensive.

Analytical Problem Framing

1. Mathematical/ Analytical Modeling of the Problem

1) Used Panda’s Library to save data into csv file
2) Cleaned Data by removing irrelevant features
3) Descriptive Statistics
4) Analyzed correlation
5) Converted all messages to lower case
6) Replaced email addresses with 'email'
7) Replaced URLs with 'web address'
8) Replaced money symbols with 'moneysymb'
   (£ can by typed with ALT key + 156)
9) Replaced 10digit phone numbers (formats include parenthesis, spaces, no spaces, dashes) with 'phone number'
10) Replace Numbers with 'number'
11) Removed Punctuation
12) Replaced extra space
13) Replaced leading and trailing white space
14) Removed \n
15) Added and removed stop words
16) Words of Sentence
17) Calculated length of sentence
18) Made one Target Column
19) Removed Total length
20) Checked the word which are offensive using Word Cloud
21) Checked the word which are not offensive using Word Cloud
22) Converted text into vectors using TF-IDF

2. Data Sources and their formats

There are two data-set in csv format: **train and test dataset**. Features of this dataset are:

- 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 andhurtful.
- Rude: It denotes comments that are very rude and offensive.
- Threat: It contains indication of the comments that are giving any threatto someone.
- Abuse: It is for comments that are abusive in nature.
- Loathe: It describes the comments which are hateful and loathing innature.
- ID: It includes unique Ids associated with each comment text given.
- Comment text: This column contains the comments extracted fromvarious social media platforms.

3. Data Pre-processing:

a) Checked Top 5 Rows of both Dataset and Checked Total Numbers of Rows and Column

```
print('No. of Rows :',df.shape[0])
print('No. of Columns :',df.shape[1])
pd.set_option('display.max_columns',None)
df.head()
```

```
No. of Rows : 159571
No. of Columns : 8
```

	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

```
print('No. of Rows :',dft.shape[0])
print('No. of Columns :',dft.shape[1])
pd.set_option('display.max_columns',None)
dft.head()
```

```
No. of Rows : 153164
No. of Columns : 2
```

'8]:

	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.

b) Sorting out columns for datatypes

```
In [74]:  # Sorting out columns for datatypes
          df.columns.to_series().groupby(df.dtypes).groups

Out[74]: {int64: ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe'], object: ['id', 'comment_text']}
```

c) Checked Data Type of All Data

```
df.dtypes

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

```
dft.dtypes

0]:  id              object
     comment_text    object
     dtype: object
```

d) Checked for Null Values

```
df.isnull().sum()

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

```
dft.isnull().sum()

]:  id              0
    comment_text    0
    dtype: int64
```

There is no null value in the dataset.

e) Checked total number of unique values

```
df.nunique()
```

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

```
dft.nunique()
```

```
5]: id              153164
    comment_text    153164
    dtype: int64
```

f) Checking unique values present in the columns: ("malignant", "highly_malignant", "rude", "threat", "abuse", "loathe")

```
comment_columns= ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
for i in df[comment_columns]:
    print(i, df[i].unique(),"\n")
```

```
malignant [0 1]

highly_malignant [0 1]

rude [0 1]

threat [0 1]

abuse [0 1]

loathe [0 1]
```

g) Information about Data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159571 entries, 0 to 159570
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                159571 non-null  object
 1   comment_text      159571 non-null  object
 2   malignant         159571 non-null  int64
 3   highly_malignant  159571 non-null  int64
 4   rude              159571 non-null  int64
 5   threat            159571 non-null  int64
 6   abuse             159571 non-null  int64
 7   loathe            159571 non-null  int64
dtypes: int64(6), object(2)
memory usage: 9.7+ MB
```

```
dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153164 entries, 0 to 153163
Data columns (total 2 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            153164 non-null  object
 1   comment_text  153164 non-null  object
dtypes: object(2)
memory usage: 2.3+ MB
```

h) Data cleaning

- Dropped Column " id" as this column contains serial no.

```
#dropping column "id" as this column contains unique value which is not relevant for prediction
df.drop("id",axis=1,inplace=True)
```

```
#dropping column "id" as this column contains unique value which is not relevant for prediction
dft.drop("id",axis=1,inplace=True)
```

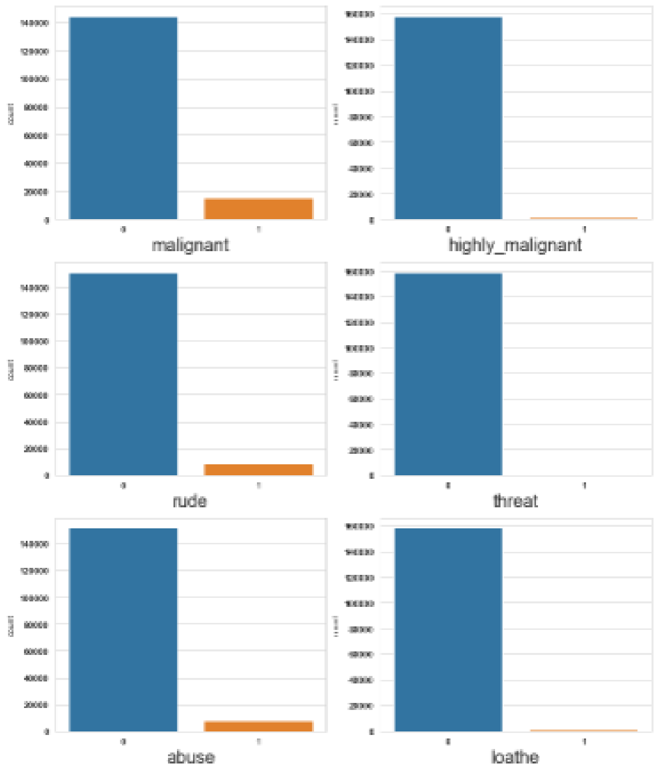
i) Checking the comments text of train dataset

```
## comments
df['comment_text'][0]
```

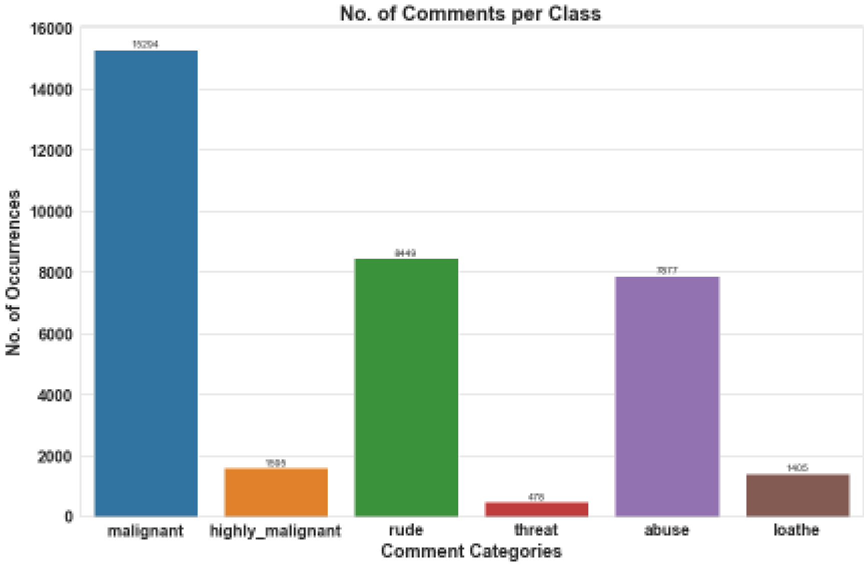
```
9]: "Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They weren't vandalisms, just closu
re on some GAs after I voted at New York Dolls FAC. And please don't remove the template from the talk page since I'm retire
d now.89.205.38.27"
```

j) Data Visualization

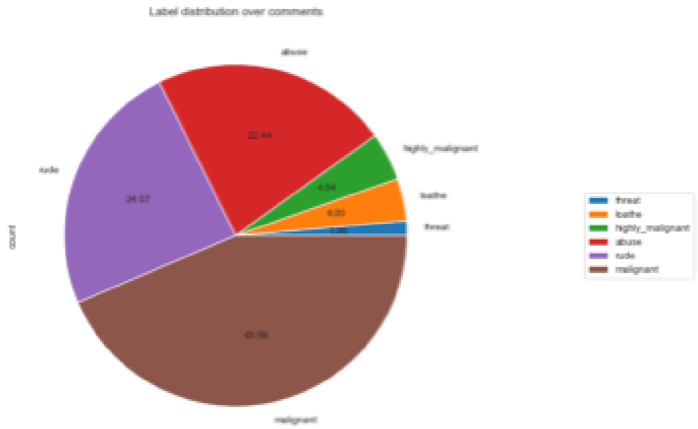
- Uni-Variate Analysis



Bivariate Analysis



Multivariate Analysis



4. Data Inputs- Logic- Output Relationships

i Descriptive Statistics

```
df.describe().T
```

1]:

	count	mean	std	min	25%	50%	75%	max
malignant	159571.0	0.095844	0.294379	0.0	0.0	0.0	0.0	1.0
highly_malignant	159571.0	0.009996	0.099477	0.0	0.0	0.0	0.0	1.0
rude	159571.0	0.052948	0.223931	0.0	0.0	0.0	0.0	1.0
threat	159571.0	0.002996	0.054650	0.0	0.0	0.0	0.0	1.0
abuse	159571.0	0.049364	0.216627	0.0	0.0	0.0	0.0	1.0
loathe	159571.0	0.008805	0.093420	0.0	0.0	0.0	0.0	1.0

ii Checking Correlation

```
#Checking correlation of the dataset
corr=df.corr()
corr
```

	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

iii Correlation with Heatmap

```
# Plotting heatmap for visualizing the correlation
plt.figure(figsize=(15, 10))
corr = df.corr()
sns.heatmap(corr, linewidth=0.5, linecolor='black', fmt='.0%', annot=True)
plt.show()
```

iv Handling "df" Dataset

```
#Defining the stop words
stop_words = stopwords.words('english')

#Defining the Lemmatizer
lemmatizer = WordNetLemmatizer()
```

```
#Replacing '\n' in comment_text
df['comment_text'] = df['comment_text'].replace('\n',' ')
```

```
#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)
```

```
# Calling the above function for the column comment_text in training dataset to replace original with cleaned text
df['comment_text'] = df['comment_text'].apply(clean_comments)
df['comment_text'].head()
```

08]:
```
0    explanation edits made username hardcore metal...
1    aww match background colour seemingly stuck th...
2    hey man really trying edit war guy constantly ...
3    make real suggestion improvement wondered sect...
4                       sir hero chance remember page
Name: comment_text, dtype: object
```

```
# Creating a column 'len_after_cleaning'
# Representing the length of the each comment respectively in a column 'comment_text' after cleaning the text.
df['length_after_cleaning'] = df['comment_text'].map(lambda comment_text: len(comment_text))
df.head()
```

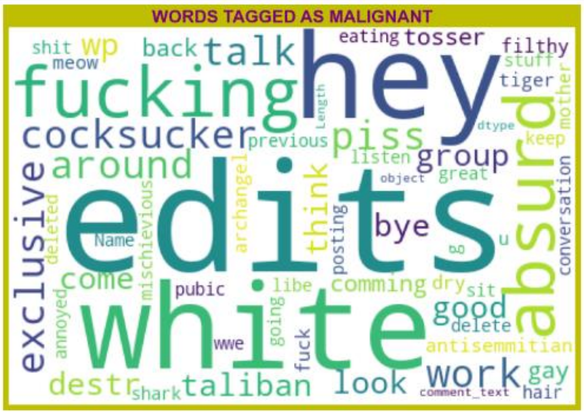
09]:

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length_before_cleaning	length_after_cleaning
0	explanation edits made username hardcore metal...	0	0	0	0	0	0	264	156
1	aww match background colour seemingly stuck th...	0	0	0	0	0	0	112	67
2	hey man really trying edit war guy constantly ...	0	0	0	0	0	0	233	141
3	make real suggestion improvement wondered sect...	0	0	0	0	0	0	622	364
4	sir hero chance remember page	0	0	0	0	0	0	67	29

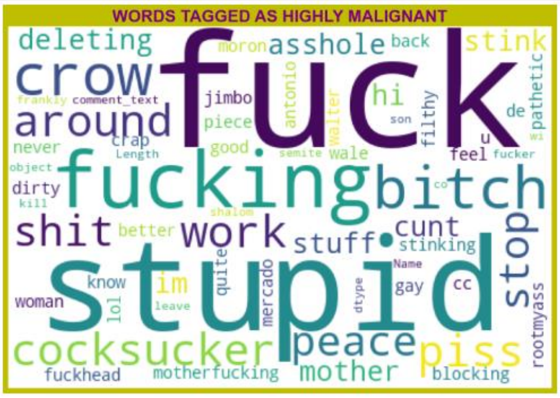
```
# Checking Total Length removal in train dataset
print("Original Length:", df.length_before_cleaning.sum())
print("Cleaned Length:", df.length_after_cleaning.sum())
print("Total Words Removed:", (df.length_before_cleaning.sum()) - (df.length_after_cleaning.sum()))
```

```
Original Length: 62893130
Cleaned Length: 38474840
Total Words Removed: 24418290
```

```
# Plotting for malignant
df_malignant=df[(df['malignant']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df_malign
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS MALIGNANT',fontdict={'fontsize':22, 'fontweight':'bold', 'color':'purp
plt.show()
```




```
# Plotting for highly_malignant
df_highlymalignant=df[(df['highly_malignant']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df_highl
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS HIGHLY MALIGNANT',fontdict={'fontsize':22, 'fontweight':'bold', 'colo
plt.show()
```




```
# Plotting for rude
df_rude=df[(df['rude']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df_r
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS RUDE',fontdict={'fontsize':22, 'fontweight':'bold', 'color':'purp
plt.show()
```

```
# Plotting for threat
df_threat=df[(df['threat']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df_
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS THREAT',fontdict={'fontsize':22, 'fontweight':'bold', 'color':'
plt.show()
```


```
# Plotting for abuse
df_abuse=df[(df['abuse']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS ABUSE',fontdict={'fontsize':22, 'fontweight':'bold', 'color':'p
plt.show()
```


```
# Plotting for Loathe
df_loathe=df[(df['loathe']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df_l
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS LOATHE',fontdict={'fontsize':22, 'fontweight':'bold', 'color':'pu
plt.show()
```

v Handling "dft" Dataset

```
# Calling the above function for the column comment_text in test dataset so that we can replace original with cleaned text
dft['comment_text'] = dft['comment_text'].apply(clean_comments)
dft['comment_text'].head()
```

```
0    yo bitch ja rule succesful ever whats hating s...
1                                 rfc title fine imo
2                         source zawe ashton lapland
3    look back source information updated correct f...
4                           anonymously edit article
Name: comment_text, dtype: object
```

```
#Creating a column 'Len_after_cleaning'
#It represents the length of the each comment respectively in a column 'comment_text' after cleaning the text
dft['length_after_cleaning'] = dft['comment_text'].map(lambda comment_text: len(comment_text))
dft.head()
```

	comment_text	length_before_cleaning	length_after_cleaning
0	yo bitch ja rule succesful ever whats hating s...	367	235
1	rfc title fine imo	50	18
2	source zawe ashton lapland	54	26
3	look back source information updated correct f...	205	109
4	anonymously edit article	41	24

```
# Total length removal in test dataset
print('Original Length:',dft.length_before_cleaning.sum())
print('Clean Length:',dft.length_after_cleaning.sum())
print("Total Words Removed:", (dft.length_before_cleaning.sum()) - (dft.length_after_cleaning.sum()))
```

```
Original Length: 55885733
Clean Length: 34282033
Total Words Removed: 21603700
```

5. State the set of assumptions (if any) related to theproblem under consideration

- It was observed that there is one column "id" which is irrelevant column as it contains serial no, so, have to drop this column.
- It was observed that in columns there are irrelevant values present in comment_text. So, we need to drop, replace and remove those values.
- Also have to convert comment_text into vectors using TF-IDF
- Have to create on Target column also.

6. Hardware and Software Requirements and Tools Used

- **Hardware tools:**

1. Windows laptop
2. i5 processor

3. 4GB ram 4. 250 GB SSD card

- **Software tools:**

1. windows 10
2. Anaconda Navigator
3. Jupyter Notebook
4. Python

- **Libraries and packages:**

1. Pandas
2. NumPy
3. SciPy
4. Seaborn
5. Mat plot
6. Sklearn

And

```python
# Importing Required Libraries
import nltk
import re
import string
from nltk.corpus import stopwords
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

Model/s Development and Evaluation

1. Identification of possible problem-solving approaches(methods)

In this project, we want to differentiate between comments and itscategories and for this we have used these approaches:

- Checked Total Numbers of Rows and Column
- Checked All Column Name
- Checked Data Type of All Data
- Checked for Null Values
- Checked for special character present in dataset or not
- Checked total number of unique values
- Information about Data
- Checked Description of Data and Dataset
- Dropped irrelevant Columns
- Replaced special characters and irrelevant data
- Checked all features through visualization.
- Checked correlation of features
- Converted all messages to lower case
- Replaced email addresses with 'email'
- Replaced URLs with 'web address'
- Replaced money symbols with 'moneysymb'(£ can by typed with ALT key + 156)
- Replaced 10digit phone numbers (formats include parenthesis, spaces,no spaces, dashes) with 'phone number'
- Replace Numbers with 'number'
- Removed Punctuation
- Replaced extra space
- Replaced leading and trailing white space
- Removed \n
- Added and removed stop words
- Words of Sentence
- Calculated length of sentence
- Made one Target Column
- Removed Total length
- Checked the word which are offensive using Word Cloud
- Checked the word which are not offensive using Word Cloud
- Converted text into vectors using TF-IDF

Testing of Identified Approaches (Algorithms)

1. Logistic Regression
2. AdaBoost Classifier
3. Decision Tree Classifier
4. KNN Classifier
5. Gradient Boosting Classifier
6. XGB Classifier
7. MultinomialNB

2. Run and evaluate selected models

```
#Importing Machine Learning Model Library
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
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.svm import SVC, LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import hamming_loss, log_loss
from sklearn.model_selection import RepeatedStratifiedKFold
```

Creating Model

We are using Classification Algorithm

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=70)
```

```
x.shape, y.shape
```

```
((159571, 10000), (159571,))
```

```
x_train.shape,y_train.shape, x_test.shape,y_test.shape
```

```
((111699, 10000), (111699,), (47872, 10000), (47872,))
```

We can see the x.shape value is divided into x_train.shape and x_test.shape and like this y.shape is also divided. We will understand this by Classification problem.

1. Logistic Regression

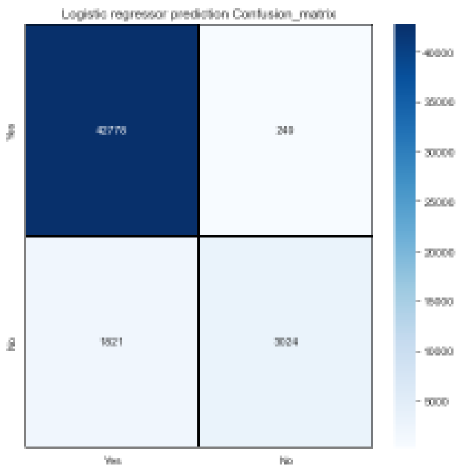
```
lr=LogisticRegression()
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)

print("accuracy_score: ", accuracy_score(y_test, pred_lr))
print("confusion_matrix: \n", confusion_matrix(y_test, pred_lr))
print("classification_report: \n", classification_report(y_test,pred_lr))
```

```
accuracy_score:  0.956759692513369
confusion_matrix:
 [[42778   249]
 [ 1821  3024]]
classification_report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.98     43027
           1       0.92      0.62      0.75      4845

    accuracy                           0.96     47872
   macro avg       0.94      0.81      0.86     47872
weighted avg       0.96      0.96      0.95     47872
```

Confusion Matrix for Logistic Regression

```
cm = confusion_matrix(y_test,pred_lr)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("Logistic regressor prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'Logistic regressor prediction Confusion_matrix')
```



Cross Validation Score for Logistic Regression

```
#CV Score for Logistic Regression
print('CV score for Logistic Regression: ',cross_val_score(lr,x,y,cv=5).mean())
```

```
CV score for Logistic Regression:  0.9562898458849187
```

2. AdaBoost Classifier

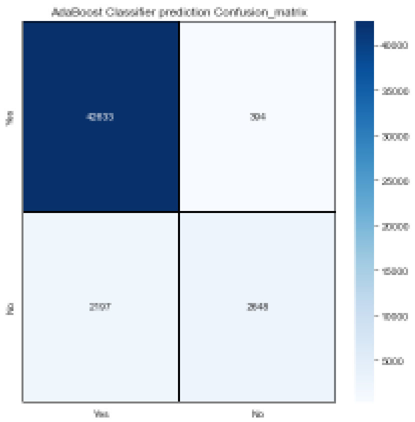
```
abc = AdaBoostClassifier()
abc.fit(x_train,y_train)
pred_abc = abc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_abc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_abc))
print("classification_report: \n",classification_report(y_test,pred_abc))
```

```
accuracy_score:  0.9458765848186952
confusion_matrix:
 [[42633   394]
 [ 2197  2648]]
classification_report:
               precision    recall  f1-score   support

           0       0.95      0.99      0.97     43027
           1       0.87      0.55      0.67      4845

    accuracy                           0.95     47872
   macro avg       0.91      0.77      0.82     47872
weighted avg       0.94      0.95      0.94     47872
```

Confusion Matrix for AdaBoost Classifier

```
cm = confusion_matrix(y_test,pred_abc)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues"
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("AdaBoost Classifier prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'AdaBoost Classifier prediction Confusion_matrix')
```



Cross Validation Score for AdaBoost Classifier

```
print('CV score for AdaBoost Classifier: ',cross_val_score(abc,x,y,cv=5).mean())
```

```
CV score for AdaBoost Classifier:  0.9458924205583014
```

3. Decision Tree Classifier

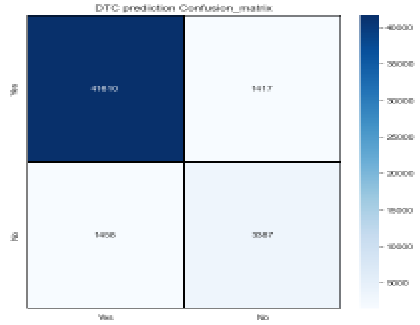
```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
pred_dtc = dtc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_dtc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_dtc))
print("classification_report: \n",classification_report(y_test,pred_dtc))
```

```
accuracy_score:  0.9399448173796791
confusion_matrix:
 [[41610  1417]
 [ 1458  3387]]
classification_report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97     43027
           1       0.71      0.70      0.70      4845

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

Confusion Matrix for Decision Tree Classifier

6]:
```
cm = confusion_matrix(y_test,pred_dtc)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("DTC prediction Confusion_matrix")
```

t[136]:
```
Text(0.5, 1.0, 'DTC prediction Confusion_matrix')
```



Cross Validation Score for Decision Tree Classifier

9]:
```
print('CV score for Decision Tree Classifier: ',cross_val_score(dtc,x,y,cv=5).mean())
```

```
CV score for Decision Tree Classifier:  0.9408037798905567
```

4. KNN Classifier

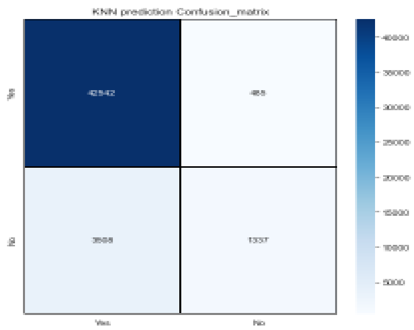
137]:
```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
pred_knn = knn.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_knn))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_knn))
print("classification_report: \n",classification_report(y_test,pred_knn))
```

```
accuracy_score:  0.9165900735294118
confusion_matrix:
 [[42542   485]
 [ 3588  1337]]
classification_report:
               precision    recall  f1-score   support

           0       0.92      0.99      0.96     43027
           1       0.73      0.28      0.40      4845

    accuracy                           0.92     47872
   macro avg       0.83      0.63      0.68     47872
weighted avg       0.90      0.92      0.90     47872
```

Confusion Matrix for KNN

]:
```
cm = confusion_matrix(y_test,pred_knn)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("KNN prediction Confusion_matrix")
```

[138]:
```
Text(0.5, 1.0, 'KNN prediction Confusion_matrix')
```



Cross Validation Score for KNN Classifier

]:
```
print('CV score for KNN Classifier: ',cross_val_score(knn,x,y,cv=5).mean())
```

```
CV score for KNN Classifier:  0.918876233559345
```

5. Gradient Boosting Classifier

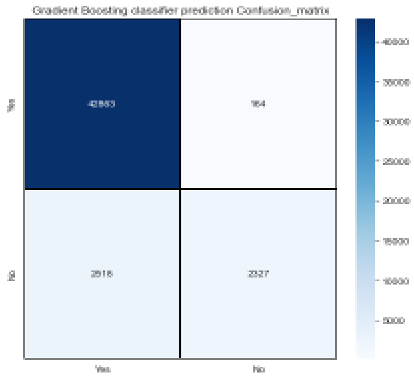
```
gb = GradientBoostingClassifier(n_estimators =100,learning_rate=0.1, max_depth=4)
gb.fit(x_train,y_train)
pred_gb = gb.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_gb))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_gb))
print("classification_report: \n",classification_report(y_test,pred_gb))
```

```
accuracy_score:  0.9439756016042781
confusion_matrix:
 [[42863   164]
 [ 2518  2327]]
classification_report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97     43027
           1       0.93      0.48      0.63      4845

    accuracy                           0.94     47872
   macro avg       0.94      0.74      0.80     47872
weighted avg       0.94      0.94      0.94     47872
```

Confusion Matrix for Gradient Boosting classifier

```
cm = confusion_matrix(y_test,pred_gb)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("Gradient Boosting classifier prediction Confusion_matrix")
```

```
Text(0.5, 1.0, 'Gradient Boosting classifier prediction Confusion_matrix')
```



Cross Validation Score for Gradient Boosting Classifier

```
print('CV score for Gradient Boosting Classifier: ',cross_val_score(gb,x,y,cv=5).mean())
```

```
CV score for Gradient Boosting Classifier:  0.9436882409923324
```

6. XGB Classifier

```
XGBC= XGBClassifier()
XGBC.fit(x_train,y_train)
pred_XGBC = XGBC.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_XGBC))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_XGBC))
print("classification_report: \n",classification_report(y_test,pred_XGBC))
```

```
accuracy_score:  0.9538561163101604
confusion_matrix:
 [[42737   290]
 [ 1919  2926]]
classification_report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.97     43027
           1       0.91      0.60      0.73      4845

    accuracy                           0.95     47872
   macro avg       0.93      0.80      0.85     47872
weighted avg       0.95      0.95      0.95     47872
```

Confusion Matrix for XGB Classifier

```
cm = confusion_matrix(y_test,pred_XGBC)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("XGB Classifier prediction Confusion_matrix")
```

[142]: Text(0.5, 1.0, 'XGB Classifier prediction Confusion_matrix')

Cross Validation Score for XGB Classifier

```
print('CV score for XGB Classifier: ',cross_val_score(XGBC,x,y,cv=5).mean())
```

```
CV score for XGB Classifier:  0.9537760580926742
```

7. MultinomialNB

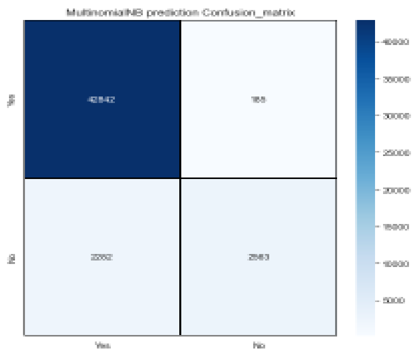
```
MNB= MultinomialNB()
MNB.fit(x_train,y_train)
pred_MNB = MNB.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_MNB))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_MNB))
print("classification_report: \n",classification_report(y_test,pred_MNB))
```

```
accuracy_score:  0.9484667446524064
confusion_matrix:
 [[42842   185]
 [ 2282  2563]]
classification_report:
               precision    recall  f1-score   support

           0       0.95      1.00      0.97     43027
           1       0.93      0.53      0.68      4845

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

Confusion Matrix for MultinomialNB

```
cm = confusion_matrix(y_test,pred_MNB)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title("MultinomialNB prediction Confusion_matrix")
```

[144]: Text(0.5, 1.0, 'MultinomialNB prediction Confusion_matrix')



Cross Validation Score for XGB Classifier

```
print('CV score for MultinomialNB: ',cross_val_score(MNB,x,y,cv=5).mean())
```

```
CV score for MultinomialNB:  0.947941672439417
```

8. Random Forest Classifier

45]:

```
RFC= RandomForestClassifier()
RFC.fit(x_train,y_train)
pred_RFC = RFC.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_RFC))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_RFC))
print("classification_report: \n",classification_report(y_test,pred_RFC))
```

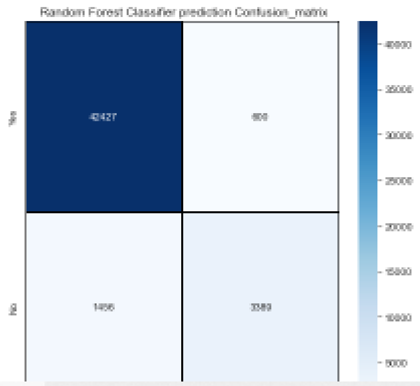
```
accuracy_score:  0.9570521390374331
confusion_matrix:
 [[42427   600]
 [ 1456  3389]]
classification_report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98     43027
           1       0.85      0.70      0.77      4845

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

Confusion Matrix for Random Forest Classifier

47]:

```
cm = confusion_matrix(y_test,pred_RFC)
x_axis_labels = ["Yes","No"]
y_axis_labels = ["Yes","No"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
xticklabels=x_axis_labels,
yticklabels=y_axis_labels)
plt.title(" Random Forest Classifier prediction Confusion_matrix")
```

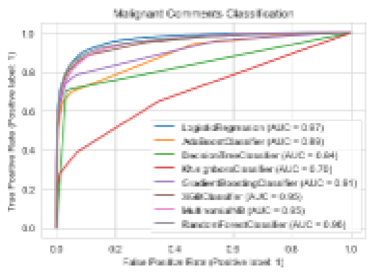
ut[147]: `Text(0.5, 1.0, ' Random Forest Classifier prediction Confusion_matrix')`



ROC & AUC Curve for all model

[156]:

```
#Lets plot roc curve and check auc and performance of all algorithms
from sklearn.metrics import roc_curve, auc, roc_auc_score, accuracy_score,classification_report, confusion_matrix, plot_roc_c
disp = plot_roc_curve(lr, x_test, y_test)
plot_roc_curve(abc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(dtc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(knn, x_test, y_test, ax = disp.ax_)
plot_roc_curve(gb, x_test, y_test, ax = disp.ax_)
plot_roc_curve(XGBC, x_test, y_test, ax = disp.ax_)
plot_roc_curve(MNB, x_test, y_test, ax = disp.ax_)
plot_roc_curve(RFC, x_test, y_test, ax = disp.ax_)
plt.title("Malignant Comments Classification")
plt.legend(prop={"size" :10} ,loc = 'lower right')
plt.show()
```



From the observation of accuracy and cross validation score and their difference we can predict that Logistic Regression is the best model.

Hyper parameter tuning for best model

The Logistic Regression with GridsearchCV

```
from sklearn.model_selection import  GridSearchCV
```

```
solver_options = ['newton-cg', 'lbfgs', 'liblinear', 'sag']
multi_class_options = ['ovr', 'multinomial']
class_weight_options = ['None', 'balanced']
```

```
param_grid = dict(solver = solver_options,
                  multi_class = multi_class_options,
                  class_weight = class_weight_options)
```

```
clf = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring = 'accuracy', )
```

```
clf.fit(x,y)
```

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
clf.best_estimator_
```

```
LogisticRegression
LogisticRegression(class_weight='None', multi_class='ovr')
```

```
print (f'Accuracy - : {clf.score(x,y)}')
```

```
Accuracy - : 0.9608074148811501
```

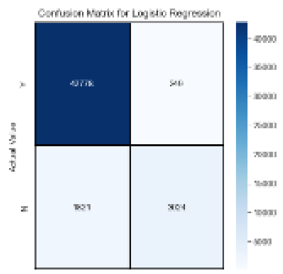
```
malignant= LogisticRegression(class_weight='None',multi_class='ovr')
malignant.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression(class_weight='None', multi_class='ovr')
```

```
pred = malignant.predict(x_test)
print("accuracy score: ",accuracy_score(y_test,pred))
print("Cross_validation_Score :", cross_val_score(lr,x,y,cv=5).mean())
print("confusion_matrix: \n",confusion_matrix(y_test,pred))
print("classification_report: \n",classification_report(y_test,pred))
```

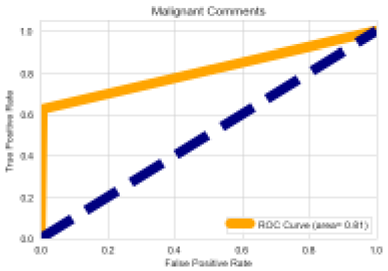
```
accuracy score:  0.956759692513369
Cross_validation_Score : 0.9562890458849187
confusion_matrix:
 [[42778   249]
 [ 1821  3024]]
classification_report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.98     43027
           1       0.92      0.62      0.75      4845

    accuracy                           0.96     47872
   macro avg       0.94      0.81      0.86     47872
weighted avg       0.96      0.96      0.95     47872
```

```
cm = confusion_matrix(y_test, pred)

x_axis_labels = ["Y","N"]
y_axis_labels = ["Y","N"]

f, ax = plt.subplots(figsize =(5,5))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues", xticklabels=x_axis_labels
plt.xlabel("Predicted Value")
plt.ylabel("Actual Value ")
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```



```
fpr, tpr, threshold = roc_curve(y_test,pred)
auc = roc_auc_score(y_test,pred)
```

```
plt.figure()
plt.plot(fpr,tpr,color="orange",lw=10,label="ROC Curve (area= %0.2f)" % auc)
plt.plot([0,1],[0,1],color="navy",lw=10,linestyle="--")
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Malignant Comments")
plt.legend(loc="lower right")
plt.show()
```



This is the AUC-ROC curve for the models which is plotted False positive rate against True positive rate. So the best model has the area under curve as 0.81.

The Logistic Regression with RandomizedSearchCV

```
from sklearn.model_selection import RandomizedSearchCV
param =      {'warm_start':[True,False],
             'dual':[True,False],
              'random_state':[50,70,100]}
```

```
rand_search = RandomizedSearchCV(lr,param_distributions=param,cv=2)
```

```
rand_search.fit(x_train,y_train)
```

```
171]:
RandomizedSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
rand_search.best_params_
```

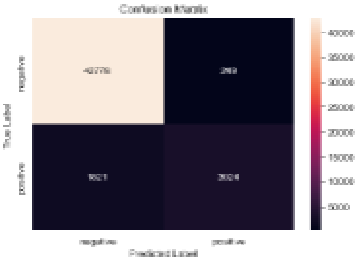
```
172]: {'warm_start': False, 'random_state': 100, 'dual': False}
```

```
lr= LogisticRegression(warm_start=False,random_state=100,dual=False)
lr.fit(x_train,y_train)

y_pred1= lr.predict(x_test)
```

```
print(" Accuracy score :",accuracy_score(y_test,y_pred1),
      "\n","="*80,
      "\n Cross_validation_Score :", cross_val_score(lr,x,y,cv=5).mean(),
      "\n","="*80,
      "\n Classification report :\n",classification_report(y_test,y_pred1),
     "="*80,
      "\n Confusion matrix :\n",confusion_matrix(y_test,y_pred1))
```

```
 Accuracy score : 0.956759692513369
 ================================================================================
 Cross_validation_Score : 0.9562898458849187
 ================================================================================
 Classification report :
               precision    recall  f1-score   support

           0       0.96      0.99      0.98     43027
           1       0.92      0.62      0.75      4845

    accuracy                           0.96     47872
   macro avg       0.94      0.81      0.86     47872
weighted avg       0.96      0.96      0.95     47872
 ================================================================================
 Confusion matrix :
 [[42778   249]
 [ 1821  3024]]
```

```
[175]:  conf_mat = confusion_matrix(y_test, y_pred1)
class_label = ["negative", "positive"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

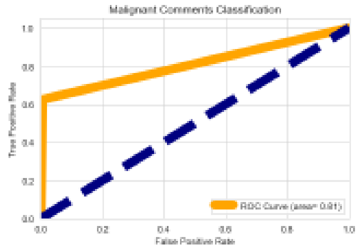


Here the final model gives 95% accuracy after tuning.

ROC-AUC Curve

```
fpr, tpr, threshold = roc_curve(y_test,y_pred1)
auc = roc_auc_score(y_test,y_pred1)
```

```
plt.figure()
plt.plot(fpr,tpr,color="orange",lw=10,label="ROC Curve (area= %0.2f)" % auc)
plt.plot([0,1],[0,1],color="navy",lw=10,linestyle="--")
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Malignant Comments Classification")
plt.legend(loc="lower right")
plt.show()
```



This is the AUC-ROC curve for the models which is plotted False positive rate against True positive rate. So the best model has the area under curve as 0.81.

We can see both method of hypertunning is giving same result. So, we can proceed with any one and here proceeding with The Logistic Regression with RandomizedSearchCV.

Saving the Model

```
import pickle
```

```
filename='Malignant_Comments_Classification.pickle'
pickle.dump(lr,open(filename,'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
```

Checking predicted and original values

```
a =np.array(y_test)
predicted=np.array(loaded_model.predict(x_test))
Malignant_Comments_Classification=pd.DataFrame({'Orginal':a,'Predicted':predicted}, index=range(len(a)))
Malignant_Comments_Classification
```

[181]:

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

47872 rows × 2 columns

Verifying Model on Testing Data

```
3]: #test data (comments) converted to vectors
    testing_data = tf_vec.fit_transform(dft["comment_text"])
```

```
4]: prediction=lr.predict(testing_data)
    prediction
```

```
t[184]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

```
5]: dft['label'] = prediction
    dft.head()
```

t[185]:

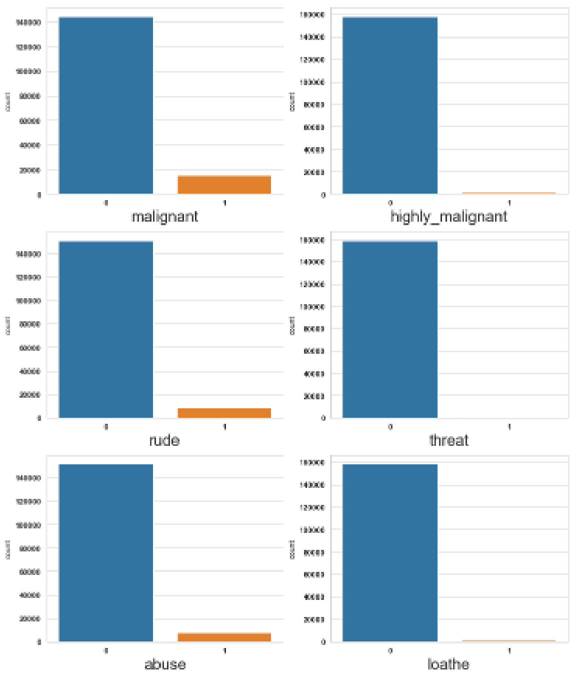
	comment_text	length_before_cleaning	length_after_cleaning	label
0	yo bitch ja rule succesful ever whats hating s...	367	235	0
1	rfc title fine imo	50	18	0
2	source zawe ashton lapland	54	28	0
3	look back source information updated correct f...	205	109	0
4	anonymously edit article	41	24	0

Saving Testing Data

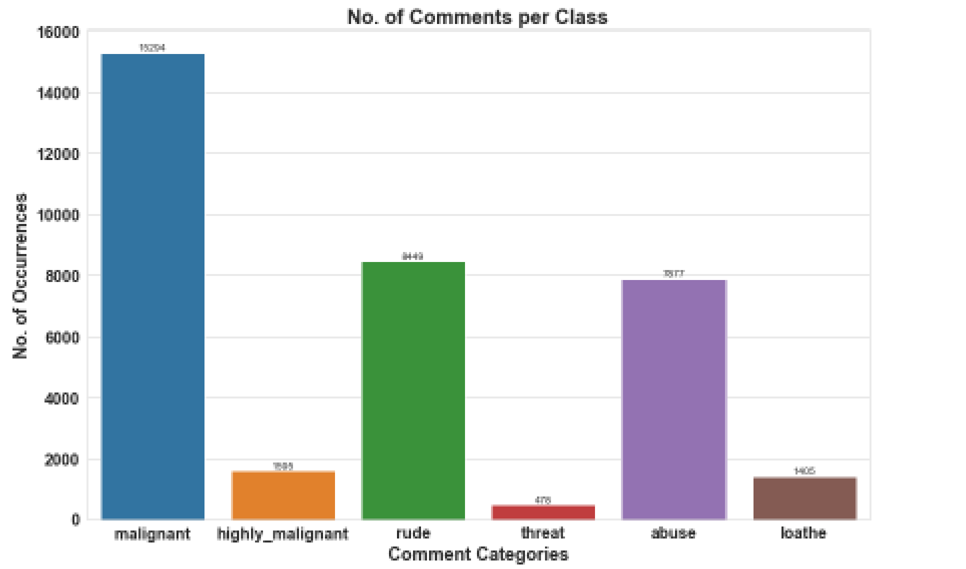
```
7]: dft.to_csv('Malignant_Test.csv',index=False)
```

- **Key Metrics for success in solving problem under consideration**
  - Accuracy Score, Precision Score, Recall Score, F1-Score and CV score are used for success. Also, confusion matrix and AUC-ROCCurve is used for success.

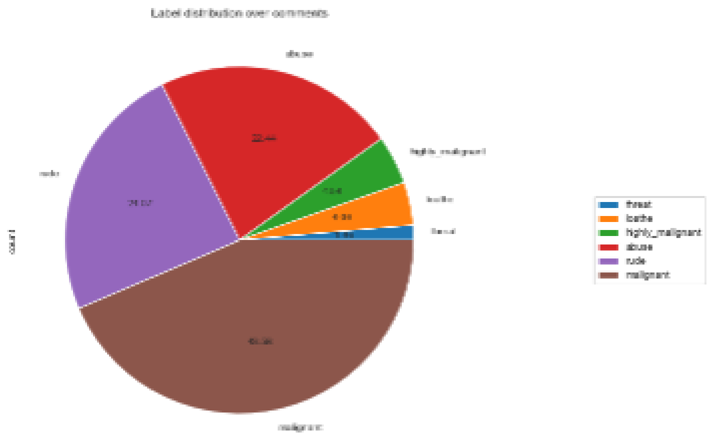
- **Visualizations**
  - Uni-Variate Analysis



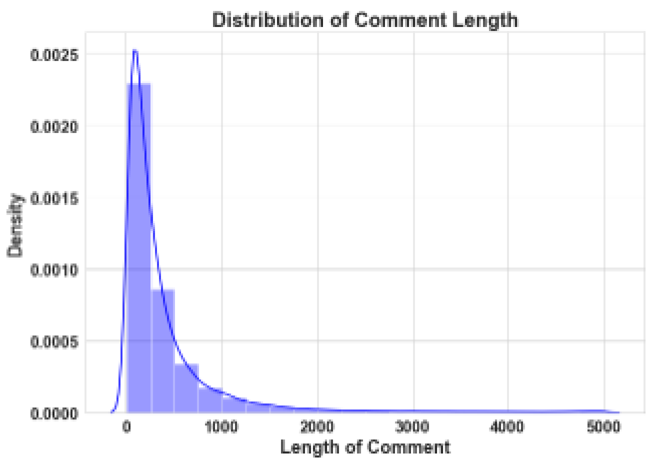
- Bi-Variate Analysis



- Pie chart



- Scatterplot



- **Interpretation of the Results**

    - Through Pre-processing it is interpretive Converted all messages to lower case, Replaced email addresses with 'email', Replaced URLs with 'web address', Replaced money symbols with 'moneysymb', (£ can by typed with ALT key + 156), Replaced 10digit phone numbers (formats include parenthesis, spaces, no spaces, dashes) with 'phone number', Replace Numbers with 'number', Removed Punctuation, Replaced extra space, Replaced leading and trailing white space, Removed \n, Added and removed stop words, Calculated length of sentence, Made one Target Column, Removed Total length, Converted text into vectors using TF-IDF
    - By creating/building model we get best model: Logistic Regression.

CONCLUSION

1. Key Findings and Conclusions of the Study

Here we have made a MALIGNANT COMMENTS CLASSIFICATION. We have done EDA, cleaned data and Visualized Data. While cleaning the data it is analyzed that:

- One column “id” is irrelevant so dropped this column.

After that we have done prediction on basis of Data using Data Pre- processing, Checked Correlation, removed email addresses, URLs, money symbols, 10digit phone numbers, Punctuation, extra space, leading and trailing white space, \n, stop words, converted text into vectors using TF-IDF and at last train our data by splitting our data through train-test split process.

Built our model using 7 models and finally selected best model which was giving best accuracy that is Logistic Regression. Then tuned our model through Hyper Tuning using GridSearchCV and RandomizedSearchCV, in which proceeded with RandomizedSearchCV.And at last compared our predicted and Actual test data. Thus, our project is completed.

2. Learning Outcomes of the Study in respect of Data Science

- This project has demonstrated the importance of NLP.
- Through different powerful tools of visualization, we were able to analyze and interpret the huge data and with the help of pie plot, count plot & word cloud, I am able to see the distribution ofthreat comments.
- Through data cleaning we were able to remove unnecessary columns, values, special characters, symbols, stop-words and punctuation from our dataset due to which our model would havesuffered from over fitting or under fitting.

The few challenges while working on this project were: -

- To find punctuations & stop words, which took time to run using NLP.
- The data set is huge it took time to run some algorithms & to check thecross-validation score.

3. Limitations of this work and Scope for Future Work

While we couldn't reach out goal of 100% accuracy but created a system that made data get very close to that goal. This project allows multiple algorithms to be integrated together tocombine modules and their results to increase the accuracy ofthe final result.

For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset.