
# Malignant Comment Classification

Submitted by:

Ali Asgar

# *ACKNOWLEDGMENT*


- https://www.ucanwest.ca/blog/media-communication/how-has-social-media-emerged-as-a-powerful-communication-medium
- https://www.socedo.com/4-reasons-read-followers-comments-social-media/
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
- https://arxiv.org/abs/1106.1813
- https://machinelearningmastery.com/logistic-regression-for-machine-learning/
- https://scikit-learn.org/stable/modules/svm.html
- https://scikit-learn.org/stable/modules/tree.html
- https://scikit-learn.org/stable/modules/neighbors.html
- https://scikit-learn.org/stable/modules/ensemble.html#forest
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
- https://eli5.readthedocs.io/en/latest/overview.html

# *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.

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. If they come across any kind of a malignant or toxic type of reply which can also be a threat or an insult or any kind of harassment that makes them uncomfortable, they might defer using the social media platform in the future

With data science and analysis, Our goal is to build a prototype of online hate and abuse comment classifier which can be used to classify hate and offensive comments so that they can be controlled and restricted from spreading hatred and cyberbullying. It will help social media houses and other agencies to identify and put a check on such comments and behavior.

# Conceptual Background of the Domain Problem

## Malignant or Toxic Comments!

Platforms that aggregate user content are the foundation of knowledge sharing on the Internet. Blogs, forums, discussion boards, and, of course, Wikipedia. But the catch is that not all people on the Internet are interested in participating nicely, and some see it as an avenue to vent their rage, insecurity, and prejudices.

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 behavior.

Online platforms when used by normal people can only be comfortably used by them only when they feel that they can express themselves freely and without any reluctance. If they come across any kind of a malignant or toxic type of a reply which can also be a threat or an insult or any kind of harassment which makes them uncomfortable, they might defer using the social media platform in the future. Thus, it becomes extremely essential for any organization or community to have an automated system that can efficiently identify and keep a track of all such comments and thus take any respective action for it, such as reporting or blocking the same to prevent any such kinds of issues in the future.

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 offensive.

# Review of Literature

## What is Commenting in Digital World?

Commenting has become a new form of written communication as it allows people to avoid being interrupted and express their ideas. People write many comments on social media (on private accounts and beyond them), and it can help brands a lot. Tracking social activity can help you better serve customers.

Every person wants to be heard, and it's in our nature to share things that we have on the mind. Social media platforms are public, so people know that they can share their views and ideas with others.

There are 7.7 billion people, and, out of these 7.7 billion, more than 3.5 billion people use some of the other forms of online social media. Which means that every one-in-three people uses social media platform. Online communication has brought information to people and audiences that previously could not be reached. It has increased awareness among people about what is happening in other parts of the world. A perfect example of social media's reach can be seen in the way the story about the Amazon Rainforest fire spread. It started with a single post and was soon present on everyone's newsfeed across different social media platforms.

**How has social media influenced the way we communicate?**

- **Essential business strategy:** Everything is just a click away and includes everything from news to buying your groceries. The ease of access that social media provides has taken over the traditional methods of shopping, reading the news, and even studying. Education too has incorporated forums and social media chat rooms to increase interactivity among students, conduct webinars and promote events and courses. Social media is a crucial section of digital marketing, helping businesses go beyond demographic and geographic boundaries.
- **Online payments:** While there was a time when online payments just meant online banking services, the dynamics have changed today. Social networking platforms like WhatsApp are incorporating options (WhatsApp Payments) within the application that allows you to transfer money to other people with minimal effort. The only challenge that these new payment systems present is the maintenance of security standards. While this is a convenient way to digitally send money, the security standards must be regulated to ensure customers are stepping into a safe zone.
- **Online healthcare:** Social media has changed the way healthcare services are carried out. Rather than physically visiting a doctor for your ailments, you can now speak to a virtual doctor who will suggest medications based on your symptoms. Some doctors even consult with patients over Skype calls to better understand their ailments. While this poses a great advantage for patients who do not want to pay a visit to the clinic, it also comes with its share of risks. The chances of non-qualified individuals posing as doctors are high. Moreover, it is difficult to confirm a doctor's credibility by the look

of their profile online. However, if used appropriately, this can be a beneficial resource for people worldwide.

- **Increased civic awareness:** Social media has changed how we are governed by making the process more transparent. Many leaders across the world have taken to social media to voice their opinions and priority issues, giving people a better understanding of the government they have elected. It has also limited the influence of political stakeholders over what information should reach people. Before social media entered the digital era, traditional media and the government were the only sources of information. However, this has now changed for the better. The downside of this is that some organizations are also misusing the power of social media to negatively influence people.
- **Disaster management:** Global warming has affected our planet to the extent that natural calamities make headlines every other day. In the face of this, social media has become a savior, enabling relief funds, information, and support can be sent and accessed more easily. For example, the safety check feature on Facebook allows you to mark yourself safe in disaster zones, helping your friends and family know that you are safe, in case there is no other medium of communication available.
- **Social justice:** Social media has brought people face-to-face with humanitarian issues. Many social work organizations such as animal welfare and fundraising organizations are also taking to social media to create awareness about the issues of society. It brings together activists, allows people to raise their voices against injustice (for example, the #metoo movement), and helps people come together for social causes.

With the increased spread of social media culture, our communication has increased. People around the globe often interact on social media over various issues. This creates a pleasant and knowledgeable environment.

## Motivation for the Problem Undertaken

As more and more people are moving towards the social media platform, and social media platforms becoming the new way of communication, it is our responsibility to keep the environment suitable for everyone. Some people often spread hate and toxicity in this environment which often prove inappropriate, disturbing, or maybe fatal for some individuals. Such actions and comments should be stopped or filtered. Considering the text data, we can use NLP processing and machine learning to filter out such comments or conversations. It is often a challenging task and one of the most required procedures nowadays to keep social media and digital platforms equally available for every individual in the world.

# *Analytical Problem Framing*

## Mathematical/ Analytical Modeling of the Problem

Our objective in the given problem is to analyze the comments collected from various social media platforms. Our task is to classify the comment into one of the categories.

In the dataset, we have 8 columns that contain Id, comment text, and six categories which are malignant, highly malignant, rude, threat, abuse and loathe. In the comment text column, we have a collection of comments that require NLP pre-processing to get a machine learning model trained and get a favorable result.

During our analysis, we observe that usually negative comments or comments connected to any of the six categories have more text. Comments that belong to these categories use abusive and insulting words. Comments with more positive words are often short. Categories like malignant, highly malignant, rude, abuse have highly abusive words. Malignant comments have high counts. Threat comments usually have words like die, kill. We also observe a good correlation between rude, abuse, and malignant categories. Threat comments usually have words like die, kill. We also observe a good correlation between rude, abuse, and malignant categories.

## Data Sources and their formats

We are provided with a dataset that has comments collected from various social media platforms and user interaction portals. We have a training dataset that has 159571 records and 8 columns and the test dataset has 153164 records and 2 columns.

Given in the dataset we have an Id column that points to a specific comment, a Comment text column that has comments, and six categories column which are Malignant, Highly Malignant, Rude, Threat, Abuse and Loathe. Each of the six categories contains a 0 or 1 for a comment.

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe |
|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | Explanation\nWhy the edits made under my usern... | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 000103f0d9cfb60f | D'aww! He matches this background colour I'm s... | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 000113f07ec002fd | Hey man, I'm really not trying to edit war. It... | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0001b41b1c6bb37e | "\nMore\nI can't make any real suggestions on ... | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0001d958c54c6e35 | You, sir, are my hero. Any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 00025465d4725e87 | "\n\nCongratulations from me as well, use the ... | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0002bcb3da6cb337 | COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK | 1 | 1 | 1 | 0 | 1 | 0 |
| 7 | 00031b1e95af7921 | Your vandalism to the Matt Shirvington article... | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 00037261f536c51d | Sorry if the word 'nonsense' was offensive to ... | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 00040093b2687caa | alignment on this subject and which are contra... | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0005300084f90edc | "\nFair use rationale for Image:Wonju.jpg\n\nT... | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 00054a5e18b50dd4 | bbq \n\nbe a man and lets discuss it-maybe ove... | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0005c987bdfc9d4b | Hey... what is it..\n@ \| talk .\nWhat is it..... | 1 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0006f16e4e9f292e | Before you start throwing accusations and warn... | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 00070ef96486d6f9 | Oh, and the girl above started her arguments w... | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 00078f8ce7eb276d | "\n\nJuelz Santanas Age\n\nIn 2002, Juelz Sant... | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0007e25b2121310b | Bye! \n\nDon't look, come or think of comming ... | 1 | 0 | 0 | 0 | 0 | 0 |
| 17 | 000897889268bc93 | REDIRECT Talk:Voydan Pop Georgiev- Chernodrinski | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 0009801bd85e5806 | The Mitsurugi point made no sense - why not ar... | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 0009eaea3325de8c | Don't mean to bother you \n\nI see that you're... | 0 | 0 | 0 | 0 | 0 | 0 |

We have a comment column as categorical and the other six categories as integer type. We don't have any null values in the dataset.

**Dataset 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
```

## Overviewing Test Dataset

```
df_test.head(20)
```

| | id | comment_text |
|---|---|---|
| 0 | 00001cee341fdb12 | Yo bitch Ja Rule is more succesful then you'll... |
| 1 | 0000247867823ef7 | == From RfC == \n\n The title is fine as it is... |
| 2 | 00013b17ad220c46 | " \n\n == Sources == \n\n * Zawe Ashton on Lap... |
| 3 | 00017563c3f7919a | :If you have a look back at the source, the in... |
| 4 | 00017695ad8997eb | I don't anonymously edit articles at all. |
| 5 | 0001ea8717f6de06 | Thank you for understanding. I think very high... |
| 6 | 00024115d4cbde0f | Please do not add nonsense to Wikipedia. Such ... |
| 7 | 000247e83dcc1211 | :Dear god this site is horrible. |
| 8 | 00025358d4737918 | " \n Only a fool can believe in such numbers. ... |
| 9 | 00026d1092fe71cc | == Double Redirects == \n\n When fixing double... |
| 10 | 0002eadc3b301559 | I think its crap that the link to roggenbier i... |
| 11 | 0002f87b16116a7f | "::: Somebody will invariably try to add Relig... |
| 12 | 0003806b11932181 | , 25 February 2010 (UTC) \n\n :::Looking it ov... |
| 13 | 0003e1cccfd5a40a | " \n\n It says it right there that it IS a typ... |
| 14 | 00059ace3e3e9a53 | " \n\n == Before adding a new product to the l... |
| 15 | 000634272d0d44eb | ==Current Position== \n Anyone have confirmati... |
| 16 | 000663aff0fffc80 | this other one from 1897 |
| 17 | 000689dd34e20979 | == Reason for banning throwing == \n\n This ar... |
| 18 | 000834769115370c | :: Wallamoose was changing the cited material ... |
| 19 | 000844b52dee5f3f | \|blocked]] from editing Wikipedia. \| |

```
df_test.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
```

## Data Preprocessing Done

As the data was collected from various websites, it contains garbage data that is not useful. We pre-process the data including Natural Language Processing to filter out the garbage data as much as we can.

### Copying

We start by creating a copy of the dataset

```
# Making a copy of dataset
df1=df_train.copy()
```

### Converting the comments into lowercase

```
# Converting text to lowercase
df1['comment_text']=df1['comment_text'].str.lower()
```

| | 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 |

### Checking the length of each review before cleaning

We check the length of each review and store it in the ‘uncleaned_Length’ column.

```
# Creating a column to get length of each comment
df1['uncl_length']=df1['comment_text'].str.len()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation\nwhy the edits made under my usern... | 0 | 0 | 0 | 0 | 0 | 0 | 264 |
| 1 | 000103f0d9cfb60f | d'aww! he matches this background colour i'm s... | 0 | 0 | 0 | 0 | 0 | 0 | 112 |
| 2 | 000113f07ec002fd | hey man, i'm really not trying to edit war. it... | 0 | 0 | 0 | 0 | 0 | 0 | 233 |
| 3 | 0001b41b1c6bb37e | "\nmore\ni can't make any real suggestions on ... | 0 | 0 | 0 | 0 | 0 | 0 | 622 |
| 4 | 0001d958c54c6e35 | you, sir, are my hero. any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 | 67 |

## Expanding Contracted words

We have lots of contracted words in the dataset. We use 'Contractions' library to expand them.

```
import contractions # for handling Contrated words
```

```
# Expanding Contracted words using Contractions Library

df1['comment_text'] = df1['comment_text'].apply(lambda x: [contractions.fix(word) for word in x.split()])
```

```
# Rejoining tokenized words to form string

df1['comment_text'] = [' '.join(map(str, text)) for text in df1['comment_text']]
```

```
df1.head()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation why the edits made under my userna... | 0 | 0 | 0 | 0 | 0 | 0 | 264 |
| 1 | 000103f0d9cfb60f | d'aww! he matches this background colour I am ... | 0 | 0 | 0 | 0 | 0 | 0 | 112 |
| 2 | 000113f07ec002fd | hey man, I am really not trying to edit war. i... | 0 | 0 | 0 | 0 | 0 | 0 | 233 |
| 3 | 0001b41b1c6bb37e | " more i cannot make any real suggestions on i... | 0 | 0 | 0 | 0 | 0 | 0 | 622 |
| 4 | 0001d958c54c6e35 | you, sir, are my hero. any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 | 67 |

## Replacing email addresses, web addresses, any money symbols, phone numbers or any digit in the comment section

```
# Replace email addresses with 'email'
df1['comment_text'] = df1['comment_text'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$','emailaddress')

# Replace URLs with 'webaddress'
df1['comment_text'] = df1['comment_text'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                'webaddress')

# Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
df1['comment_text'] = df1['comment_text'].str.replace(r'£|\$', 'currencysymb')

# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
df1['comment_text'] = df1['comment_text'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$','phonenumber')

# Replace numbers with 'numbr'
df1['comment_text'] = df1['comment_text'].str.replace(r'\d+(\.\d+)?', 'numbr')
```

## Removing Punctuation and Extra characters

```
# Removing Punctuation and other non-required characters if any

df1['comment_text']=df1['comment_text'].str.replace(r'[^\w\d\s]',' ')
```

```
df1.head()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation why the edits made under my userna... | 0 | 0 | 0 | 0 | 0 | 0 | 264 |
| 1 | 000103f0d9cfb60f | d aww he matches this background colour I am ... | 0 | 0 | 0 | 0 | 0 | 0 | 112 |
| 2 | 000113f07ec002fd | hey man I am really not trying to edit war i... | 0 | 0 | 0 | 0 | 0 | 0 | 233 |
| 3 | 0001b41b1c6bb37e | more i cannot make any real suggestions on i... | 0 | 0 | 0 | 0 | 0 | 0 | 622 |
| 4 | 0001d958c54c6e35 | you sir are my hero any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 | 67 |

## Removing Stopwords

```
# Removing Stopwords
stop_words = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])

df1['comment_text'] = df1['comment_text'].apply(lambda x: ' '.join(term for term in x.split() if term not in stop_words))
```

```
df1.head()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation edits made username hardcore metal... | 0 | 0 | 0 | 0 | 0 | 0 | 264 |
| 1 | 000103f0d9cfb60f | aww matches background colour I seemingly stuc... | 0 | 0 | 0 | 0 | 0 | 0 | 112 |
| 2 | 000113f07ec002fd | hey man I really trying edit war guy constantl... | 0 | 0 | 0 | 0 | 0 | 0 | 233 |
| 3 | 0001b41b1c6bb37e | cannot make real suggestions improvement wonde... | 0 | 0 | 0 | 0 | 0 | 0 | 622 |
| 4 | 0001d958c54c6e35 | sir hero chance remember page | 0 | 0 | 0 | 0 | 0 | 0 | 67 |

## Lemmatization

We will lemmatize each word in the reviews to get short yet meaningful words.

### Lemmatization

```
from nltk.stem import WordNetLemmatizer

lemma=WordNetLemmatizer()
```

```
# lemmatizing words
df1['comment_text'] = df1['comment_text'].apply(lambda x: ' '.join(lemma.lemmatize(word) for word in x.split()))
```

```
df1.head()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation edits made username hardcore metal... | 0 | 0 | 0 | 0 | 0 | 0 | 264 |
| 1 | 000103f0d9cfb60f | aww matches background colour I seemingly stuc... | 0 | 0 | 0 | 0 | 0 | 0 | 112 |
| 2 | 000113f07ec002fd | hey man I really trying edit war guy constantl... | 0 | 0 | 0 | 0 | 0 | 0 | 233 |
| 3 | 0001b41b1c6bb37e | cannot make real suggestions improvement wonde... | 0 | 0 | 0 | 0 | 0 | 0 | 622 |
| 4 | 0001d958c54c6e35 | sir hero chance remember page | 0 | 0 | 0 | 0 | 0 | 0 | 67 |

## Checking length of the comments after processing

```
# Creating a column to get comment length after cleaning

df1['cleaned_length']=df1['comment_text'].str.len()
```

```
df1.head()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length | cleaned_length |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation edits made username hardcore metal... | 0 | 0 | 0 | 0 | 0 | 0 | 264 | 172 |
| 1 | 000103f0d9cfb60f | aww matches background colour I seemingly stuc... | 0 | 0 | 0 | 0 | 0 | 0 | 112 | 95 |
| 2 | 000113f07ec002fd | hey man I really trying edit war guy constantl... | 0 | 0 | 0 | 0 | 0 | 0 | 233 | 143 |
| 3 | 0001b41b1c6bb37e | cannot make real suggestions improvement wonde... | 0 | 0 | 0 | 0 | 0 | 0 | 622 | 381 |
| 4 | 0001d958c54c6e35 | sir hero chance remember page | 0 | 0 | 0 | 0 | 0 | 0 | 67 | 29 |

We checked the total sum of uncleaned data length and cleaned data length to find out the number of garbage words.

```
# Checking cleaned garbage text quantity

print(df1.uncl_length.sum())
print(df1.cleaned_length.sum())

print('************ \n')
print('Total Garbage Cleaned : ',df1.uncl_length.sum()-df1.cleaned_length.sum())
print('************ \n')
print('Total Garbage % : ',((df1.uncl_length.sum()-df1.cleaned_length.sum())/df1.uncl_length.sum())*100)
```

```
62893157
40081108
************

Total Garbage Cleaned :  22812049
************

Total Garbage % :  36.2711145188657
```

## Vectorization

To pass the text data to the machine learning model we need to vectorize the text data. We will use TF-IDF vectorizer for the task.

```
from sklearn.feature_extraction.text import TfidfVectorizer   # for vectorizing text
```

Similar preprocessing is performed on the Test Dataset before passing for prediction.

## Balancing Dataset

In the given dataset we have to identify the comment category. Our target variables which are malignant, highly malignant, rude, threat, abuse, and loathe have two classes which are namely '1' – Show the comment belongs to the category and '0' – doesn't belong to the category. We observe an imbalance in all the target variables.  One way the imbalance may affect our Machine Learning algorithm is when our algorithm completely ignores the minority class. The reason this is an issue is that the minority class is often the class that we are most interested in.

For balancing our target variable we used SMOTE method from  Imbalanced-Learn Library. One approach to addressing imbalanced datasets is to oversample the minority class. The simplest approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the **Synthetic Minority Oversampling Technique**, or **SMOTE** for short.

We oversampled the minority class by generating synthetic data using the KNeighbors technique.

We balanced the data for each category before passing it to the machine learning algorithm.

# Balancing Dataset

```
# Using SMOTE to oversample minorities and balance our dataset

from imblearn.over_sampling import SMOTE
```

```
smote=SMOTE(sampling_strategy='minority')
```

```
x_malig_bal,y_malig_bal=smote.fit_resample(X,Y_malig)
```

```
y_malig_bal.value_counts()
```

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

```
x_malig=x_malig_bal
y_malig=y_malig_bal
```

# Balancing Dataset

```
x_hmalig_bal,y_hmalig_bal=smote.fit_resample(X,Y_hmalig)
```

```
y_hmalig_bal.value_counts()
```

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

```
x_hmalig = x_hmalig_bal
y_hmalig = y_hmalig_bal
```

# Balancing Dataset

```
x_rude_bal,y_rude_bal=smote.fit_resample(X,Y_rude)
```

```
y_rude_bal.value_counts()
```

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

```
x_rude = x_rude_bal
y_rude = y_rude_bal
```

# Balancing Dataset

```
x_threat_bal,y_threat_bal=smote.fit_resample(X,Y_threat)
```

```
y_threat_bal.value_counts()
```

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

```
x_threat=x_threat_bal
y_threat=y_threat_bal
```

# Balancing Dataset

```
x_abuse_bal,y_abuse_bal=smote.fit_resample(X,Y_abuse)
```

```
y_abuse_bal.value_counts()
```

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

```
x_abuse = x_abuse_bal
y_abuse = y_abuse_bal
```

# Balancing Dataset

```
x_loathe_bal,y_loathe_bal=smote.fit_resample(X,Y_loathe)
```

```
y_loathe_bal.value_counts()
```

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

```

```

```
x_loathe = x_loathe_bal
y_loathe = y_loathe_bal
```

## Hardware and Software Requirements and Tools Used

Software Requirement:

1. Anaconda Navigator software.

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.

2. Jupyter Notebook - as an IDE for writing Python codes for analyzing data and machine learning purposes.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

**A web application**: a browser-based tool for interactive authoring of documents that combine explanatory text, mathematics, computations, and their rich media output.

**Notebook documents**: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects

3. Python Programming

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are

available in source or binary form without charge for all major platforms and can be freely distributed

## 4. Libraries Used

### Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms basic linear algebra, basic statistical operations, random simulation and much more.

### Pandas

Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data-wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState's ActivePython.

### Matplotlib

Matplotlib is a cross-platform, data visualization, and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

- The pyplot API is a hierarchy of Python code objects topped by *matplotlib.pyplot*
- An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

## Selenium

Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. It provides extensions to emulate user interaction with browsers, a distribution server for scaling browser allocation, and the infrastructure for implementations of the W3C WebDriver specification that lets you write interchangeable code for all major web browsers.

This project is made possible by volunteer contributors who have put in thousands of hours of their own time and made the source code freely available for anyone to use, enjoy, and improve. Selenium brings together browser vendors, engineers, and enthusiasts to further an open discussion around the automation of the web platform. The project organizes an annual conference to teach and nurture the community. At the core of Selenium is WebDriver, an interface to write instruction sets that can be run interchangeably in many browsers.

## Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

## Scikit-Learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Some popular groups of models provided by scikit-learn include:

- Clustering: for grouping unlabeled data such as KMeans.
- Cross-Validation: for estimating the performance of supervised models on unseen data.

- Datasets: for test datasets and for generating datasets with specific properties for investigating model behavior.
- Dimensionality Reduction: for reducing the number of attributes in data for summarization, visualization, and feature selection such as Principal component analysis.
- Ensemble methods: for combining the predictions of multiple supervised models.
- Feature extraction: for defining attributes in image and text data.
- Feature selection: for identifying meaningful attributes from which to create supervised models.
- Parameter Tuning: for getting the most out of supervised models.
- Manifold Learning: For summarizing and depicting complex multi-dimensional data.

Supervised Models: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines, and decision trees.

# *Model/s Development and Evaluation*

## Identification of possible problem-solving approaches (methods)

The given scenario challenges us to predict the category for comment. This kind of problem is very challenging in the field of data science and machine learning as it is based on Natural Language Processing. Our task in the dataset is to predict the category of the comment. Thus, we have six categories as our target variables where we have two classes which are from 0 to 1.

As we have six target variables we have to train six models. Also, the target variables are binary, thus, we will move ahead with the categorical approach. We have processed the data and balanced the data count for each class, hence, there is no imbalance in the dataset.

## Testing of Identified Approaches (Algorithms)

After performing various preprocessing tasks on our dataset like punctuation and other non-relevant character removals, expanding contracted words, removing extra spaces, replace digits with 'numbr' to indicate, and vectorization in the dataset, we made our data suitable for passing through the algorithms to train the best predicting model.

After preprocessing we created a train and test split using the input and target variable as 'x_malig', 'x_hmalig', 'x_rude' etc and 'y_malig', 'y_hmalig', 'y_rude' etc respectively for each model. We divided the input features in the ratio where 33.33% of data goes for testing and 66.66% of data is used for training. Using one of the algorithms we tried to find the best random state to get the best results possible.

## Finding Best Random State

```
maxAccu=0
maxRs=0
for i in range(1,500):
    x_train,x_test,y_train,y_test=train_test_split(x_malig,y_malig,test_size=.33,random_state=i)
    mnb=MultinomialNB()
    mnb.fit(x_train,y_train)
    mnb_pred=mnb.predict(x_test)
    acc=accuracy_score(y_test,mnb_pred)*100
    if acc>maxAccu:
        maxAccu=acc
        maxRs=i
print('Best Accuracy Score is : ', maxAccu, ' when Random state is : ',maxRs)
```

```
Best Accuracy Score is :  90.72703023429214  when Random state is :  122
```

Using the above-stated random state we created our train and test data.

```
x_train_malig,x_test_malig,y_train_malig,y_test_malig=train_test_split(x_malig,y_malig,test_size=.33,random_state=122)
```

With the train-test split done we move ahead and test our dataset for accuracy score with various algorithms. Algorithms used for training are :

1. Logistic Regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables.

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data. The actual representation of the model that you would store in memory or a file is the coefficients in the equation (the beta value or b's).

## 2. Decision Tree Classifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white-box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black-box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
-

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node, or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated

by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

- Some concepts are hard to learn because decision trees do not express them easily, such as XOR, parity, or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset before fitting with the decision tree

## 3. KNeighbors Classifier

Sklearn.neighbors provide functionality for unsupervised and supervised neighbors-based learning methods. Unsupervised nearest neighbors are the foundation of many other learning methods, notably manifold learning and spectral clustering. Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels and regression for data with continuous labels.

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning) or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods since they simply "remember" all of their training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree).

Despite its simplicity, nearest neighbors have been successful in a large number of classification and regression problems, including handwritten digits and satellite image scenes. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

## 4. Ensemble Methods

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm to improve generalizability/robustness over a single estimator. Two families of ensemble methods are usually distinguished:

- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimators because its variance is reduced.

  Examples: Bagging methods, Forests of randomized trees,

- By contrast, in boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

  Examples: AdaBoost, Gradient Tree Boosting, …

### ❖ Random Forest Classifier

In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max_features. (See the parameter tuning guidelines for more details).

The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yields decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice, the variance reduction is often significant hence yielding an overall better model.

### ❖ Gradient Boosting Classifier

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions. GBDT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems in a variety of areas including Web search ranking and ecology.

### ❖ Ada Boost Classifier

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessings, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights, $w_1$ , $w_2$ …$w_N$, to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence [HTF].

## 5. Extreme Gradient Boosting Classifier (XGB)

XGBoost stands for eXtreme Gradient Boosting. The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms.

It is an implementation of gradient boosting machines created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library.

XGBoost Features

The library is laser-focused on computational speed and model performance, as such there are few frills. Nevertheless, it does offer several advanced features.

Model Features

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

- Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
- Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
- Regularized Gradient Boosting with both L1 and L2 regularization.

System Features

The library provides a system for use in a range of computing environments, not least:

- Parallelization of tree construction using all of your CPU cores during training.
- Distributed Computing for training very large models using a cluster of machines.
- Out-of-Core Computing for very large datasets that don't fit into memory.
- Cache Optimization of data structures and algorithm to make the best use of hardware.

Algorithm Features

The implementation of the algorithm was engineered for the efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model. Some key algorithm implementation features include:

- Sparse Aware implementation with automatic handling of missing data values.
- Block Structure to support the parallelization of tree construction.

- Continued Training so that you can further boost an already fitted model on new data.

## 6. MultinomialNB Classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable $y$ and dependent feature vector $x_1$ through $x_n$ :

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Figure: Importing required algorithms and evaluation metrics

```
# Importing Classification Algorithms
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

We created a For Loop to test the accuracy of all the algorithms for all the models.

```
# Defining Model List
model_list=[LogisticRegression(),SVC(),DecisionTreeClassifier(),KNeighborsClassifier(),RandomForestClassifier(),GradientBoosting
```

For Malignant Comments

```
# Creating For loop to print Training and Test accuracy score
for m in model_list:
    model=m
    model.fit(x_train_malig,y_train_malig)
    model_pred_train=model.predict(x_train_malig)
    model_pred=model.predict(x_test_malig)
    print('Training Accuracy for the model ',m,' is: ',accuracy_score(y_train_malig,model_pred_train))
    print('Testing Accuracy for the model ',m,' is: ',accuracy_score(y_test_malig,model_pred))
    print('\n')
```

```
Training Accuracy for the model  LogisticRegression()  is:  0.9473752269423942
Testing Accuracy for the model  LogisticRegression()  is:  0.9345326234208122


Training Accuracy for the model  DecisionTreeClassifier()  is:  0.9997258587603643
Testing Accuracy for the model  DecisionTreeClassifier()  is:  0.94344853659305


Training Accuracy for the model  KNeighborsClassifier()  is:  0.5657706213695682
Testing Accuracy for the model  KNeighborsClassifier()  is:  0.5465276246285036


Training Accuracy for the model  RandomForestClassifier()  is:  0.9997258587603643
Testing Accuracy for the model  RandomForestClassifier()  is:  0.9842159982357204


Training Accuracy for the model  GradientBoostingClassifier()  is:  0.8395394427174121
Testing Accuracy for the model  GradientBoostingClassifier()  is:  0.8395660712222888


Training Accuracy for the model  AdaBoostClassifier()  is:  0.8209081833746269
Testing Accuracy for the model  AdaBoostClassifier()  is:  0.8208731083876795


Training Accuracy for the model  MultinomialNB()  is:  0.9172403804873507
Testing Accuracy for the model  MultinomialNB()  is:  0.9072703023429214

Training Accuracy for the model  XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)  is:  0.9241197738593396
Testing Accuracy for the model  XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)  is:  0.9170683553343205
```

## For Highly Malignant Comments

```
# Creating For loop to print Training and Test accuracy score
for m in model_list:
    model=m
    model.fit(x_train_hmalig,y_train_hmalig)
    model_pred_train=model.predict(x_train_hmalig)
    model_pred=model.predict(x_test_hmalig)
    print('Training Accuracy for the model ',m,' is: ',accuracy_score(y_train_hmalig,model_pred_train))
    print('Testing Accuracy for the model ',m,' is: ',accuracy_score(y_test_hmalig,model_pred))
    print('\n')
```

```
Training Accuracy for the model  LogisticRegression()  is:  0.9769470964206588
Testing Accuracy for the model  LogisticRegression()  is:  0.9765405457248357


Training Accuracy for the model  DecisionTreeClassifier()  is:  0.9996457033261372
Testing Accuracy for the model  DecisionTreeClassifier()  is:  0.9889991847695775


Training Accuracy for the model  KNeighborsClassifier()  is:  0.5786940152205851
Testing Accuracy for the model  KNeighborsClassifier()  is:  0.5622404450199012


Training Accuracy for the model  RandomForestClassifier()  is:  0.9996457033261372
Testing Accuracy for the model  RandomForestClassifier()  is:  0.9959622116721815


Training Accuracy for the model  GradientBoostingClassifier()  is:  0.9512440537208237
Testing Accuracy for the model  GradientBoostingClassifier()  is:  0.9508368100513116


Training Accuracy for the model  AdaBoostClassifier()  is:  0.940204169363258
Testing Accuracy for the model  AdaBoostClassifier()  is:  0.9395770392749244


Training Accuracy for the model  MultinomialNB()  is:  0.9758983782660249
Testing Accuracy for the model  MultinomialNB()  is:  0.9748813120414329
```

```
Training Accuracy for the model  XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)  is:  0.9867162367079698
Testing Accuracy for the model  XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)  is:  0.9850477149570805
```

We did this for all the categories and then did cross-validation of the top 5 accurate models for each category.

## Cross-Validation

We then Cross validated each algorithm to check for any overfitting or underfitting over various CV values.

```
from sklearn.model_selection import cross_val_score
```

### For Malignant Comments

```
# Logistic Regression
lr_malig=LogisticRegression()
lr_malig.fit(x_train_malig,y_train_malig)
lr_malig_pred=lr_malig.predict(x_test_malig)
testing_accu=accuracy_score(y_test_malig,lr_malig_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(lr_malig,x_malig,y_malig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 92.93407819680198 and the accuracy for testing is 93.58138264914989


At crossfold 3 the CV score is 93.34474857903948 and the accuracy for testing is 93.58138264914989


At crossfold 4 the CV score is 93.54055372331034 and the accuracy for testing is 93.58138264914989


At crossfold 5 the CV score is 93.64729126509617 and the accuracy for testing is 93.58138264914989


At crossfold 6 the CV score is 93.69789193739835 and the accuracy for testing is 93.58138264914989
```

```
# DecisionTreeClassifier
dtc_malig=DecisionTreeClassifier()
dtc_malig.fit(x_train_malig,y_train_malig)
dtc_malig_pred=dtc_malig.predict(x_test_malig)
testing_accu=accuracy_score(y_test_malig,dtc_malig_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(dtc_malig,x_malig,y_malig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 93.39707645709295 and the accuracy for testing is 94.36165632252712


At crossfold 3 the CV score is 94.31649009925502 and the accuracy for testing is 94.36165632252712


At crossfold 4 the CV score is 94.64398656366204 and the accuracy for testing is 94.36165632252712


At crossfold 5 the CV score is 94.93439814704679 and the accuracy for testing is 94.36165632252712


At crossfold 6 the CV score is 95.01029793672427 and the accuracy for testing is 94.36165632252712
```

## Ensemble Methods

```
# RandomForestClassifier
rfc_malig=RandomForestClassifier()
rfc_malig.fit(x_train_malig,y_train_malig)
rfc_malig_pred=rfc_malig.predict(x_test_malig)
testing_accu=accuracy_score(y_test_malig,rfc_malig_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(rfc_malig,x_malig,y_malig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 97.78620292908779 and the accuracy for testing is 98.34703800552388


At crossfold 3 the CV score is 98.33306834086879 and the accuracy for testing is 98.34703800552388


At crossfold 4 the CV score is 98.55243847111446 and the accuracy for testing is 98.34703800552388


At crossfold 5 the CV score is 98.60650053812762 and the accuracy for testing is 98.34703800552388


At crossfold 6 the CV score is 98.64635551712534 and the accuracy for testing is 98.34703800552388
```

## Naive Bayes

```
# MultinomialNB

mnb_malig=MultinomialNB()
mnb_malig.fit(x_train_malig,y_train_malig)
mnb_malig_pred=mnb_malig.predict(x_test_malig)
testing_accu=accuracy_score(y_test_malig,mnb_malig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(mnb_malig,x_malig,y_malig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 90.32867416359522 and the accuracy for testing is 90.63986641882738


At crossfold 4 the CV score is 90.5993343795721 and the accuracy for testing is 90.63986641882738


At crossfold 5 the CV score is 90.71438977539543 and the accuracy for testing is 90.63986641882738


At crossfold 6 the CV score is 90.80414857077388 and the accuracy for testing is 90.63986641882738
```

## Extreme Gradient Boosting

```
# XGBClassifier

xgb_malig=XGBClassifier(verbosity=0)
xgb_malig.fit(x_train_malig,y_train_malig)
xgb_malig_pred=xgb_malig.predict(x_test_malig)
testing_accu=accuracy_score(y_test_malig,xgb_malig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(xgb_malig,x_malig,y_malig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 91.28551630981617 and the accuracy for testing is 91.55141089862742


At crossfold 4 the CV score is 91.36591943897423 and the accuracy for testing is 91.55141089862742


At crossfold 5 the CV score is 91.38601723804497 and the accuracy for testing is 91.55141089862742


At crossfold 6 the CV score is 91.38463767238541 and the accuracy for testing is 91.55141089862742
```

### For Highly Malignant Comments

```
# Logistic Regression

lr_hmalig=LogisticRegression()
lr_hmalig.fit(x_train_hmalig,y_train_hmalig)
lr_hmalig_pred=lr_hmalig.predict(x_test_hmalig)
testing_accu=accuracy_score(y_test_hmalig,lr_hmalig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(lr_hmalig,x_hmalig,y_hmalig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 97.56355417088565 and the accuracy for testing is 97.65021819402484


At crossfold 4 the CV score is 97.66451866106244 and the accuracy for testing is 97.65021819402484


At crossfold 5 the CV score is 97.71610904935525 and the accuracy for testing is 97.65021819402484


At crossfold 6 the CV score is 97.74965836808165 and the accuracy for testing is 97.65021819402484
```

```
# DecisionTreeClassifier

dtc_hmalig=DecisionTreeClassifier()
dtc_hmalig.fit(x_train_hmalig,y_train_hmalig)
dtc_hmalig_pred=dtc_hmalig.predict(x_test_hmalig)
testing_accu=accuracy_score(y_test_hmalig,dtc_hmalig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(dtc_hmalig,x_hmalig,y_hmalig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 98.84286223794837 and the accuracy for testing is 98.89128662542561


At crossfold 4 the CV score is 98.89160378791715 and the accuracy for testing is 98.89128662542561


At crossfold 5 the CV score is 98.94034540291238 and the accuracy for testing is 98.89128662542561


At crossfold 6 the CV score is 98.9473086061023 and the accuracy for testing is 98.89128662542561
```

## Ensemble Method

```
# RandomForestClassifier

rfc_hmalig=RandomForestClassifier()
rfc_hmalig.fit(x_train_hmalig,y_train_hmalig)
rfc_hmalig_pred=rfc_hmalig.predict(x_test_hmalig)
testing_accu=accuracy_score(y_test_hmalig,rfc_hmalig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(rfc_hmalig,x_hmalig,y_hmalig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 99.57746755656581 and the accuracy for testing is 99.59909845106219


At crossfold 4 the CV score is 99.61038385577557 and the accuracy for testing is 99.59909845106219


At crossfold 5 the CV score is 99.61766359679369 and the accuracy for testing is 99.59909845106219


At crossfold 6 the CV score is 99.6325392400472 and the accuracy for testing is 99.59909845106219
```

## Naive Bayes

```
# MultinomialNB

mnb_hmalig=MultinomialNB()
mnb_hmalig.fit(x_train_hmalig,y_train_hmalig)
mnb_hmalig_pred=mnb_hmalig.predict(x_test_hmalig)
testing_accu=accuracy_score(y_test_hmalig,mnb_hmalig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(mnb_hmalig,x_hmalig,y_hmalig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 97.37175271030716 and the accuracy for testing is 97.48621301491393


At crossfold 4 the CV score is 97.38472932597357 and the accuracy for testing is 97.48621301491393


At crossfold 5 the CV score is 97.39137596615033 and the accuracy for testing is 97.48621301491393


At crossfold 6 the CV score is 97.39295869232522 and the accuracy for testing is 97.48621301491393
```

## Extreme Gradient Boosting

```
# XGBClassifier

xgb_hmalig=XGBClassifier(verbosity=0)
xgb_hmalig.fit(x_train_hmalig,y_train_hmalig)
xgb_hmalig_pred=xgb_hmalig.predict(x_test_hmalig)
testing_accu=accuracy_score(y_test_hmalig,xgb_hmalig_pred)*100
for k in range(3,7):
    cv_score=cross_val_score(xgb_hmalig,x_hmalig,y_hmalig,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 3 the CV score is 98.41684838624599 and the accuracy for testing is 98.46832590035007


At crossfold 4 the CV score is 98.4345723401023 and the accuracy for testing is 98.46832590035007


At crossfold 5 the CV score is 98.45704457368362 and the accuracy for testing is 98.46832590035007


At crossfold 6 the CV score is 98.46907151166234 and the accuracy for testing is 98.46832590035007
```

Similarly, we cross-validated the top 5 models for each category.

## Run and Evaluate selected models

Based on the accuracy score and cross-validation score, we selected top-performing algorithms. Random Forest showed the highest accuracy with minimum overfitting. We tested the selected models for Precision, Recall, and F1 scores.

Checking Classification Reports of Top 5 models

```
print('Classification Report for LR is : \n',classification_report(y_test_malig,lr_malig_pred),'\n')
print('----------------------------')
print('Classification Report for DTC is : \n',classification_report(y_test_malig,dtc_malig_pred),'\n')
print('----------------------------')
print('Classification Report for RFC is : \n',classification_report(y_test_malig,rfc_malig_pred),'\n')
print('----------------------------')
print('Classification Report for MNB is : \n',classification_report(y_test_malig,mnb_malig_pred),'\n')
print('----------------------------')
print('Classification Report for XGB is : \n',classification_report(y_test_malig,xgb_malig_pred),'\n')
print('----------------------------')
```

```
Classification Report for LR is :
               precision    recall  f1-score   support

           0       0.95      0.91      0.93     47987
           1       0.92      0.95      0.94     47236

    accuracy                           0.93     95223
   macro avg       0.93      0.93      0.93     95223
weighted avg       0.94      0.93      0.93     95223


----------------------------
Classification Report for DTC is :
               precision    recall  f1-score   support

           0       0.97      0.92      0.95     47987
           1       0.92      0.97      0.95     47236

    accuracy                           0.95     95223
   macro avg       0.95      0.95      0.95     95223
weighted avg       0.95      0.95      0.95     95223


----------------------------
Classification Report for RFC is :
               precision    recall  f1-score   support

           0       0.99      0.98      0.98     47987
           1       0.98      0.99      0.98     47236

    accuracy                           0.98     95223
   macro avg       0.98      0.98      0.98     95223
weighted avg       0.98      0.98      0.98     95223


----------------------------
```

```
Classification Report for MNB is :
              precision    recall  f1-score   support

           0       0.88      0.94      0.91     47987
           1       0.93      0.87      0.90     47236

    accuracy                           0.91     95223
   macro avg       0.91      0.91      0.91     95223
weighted avg       0.91      0.91      0.91     95223


----------------------------
Classification Report for XGB is :
              precision    recall  f1-score   support

           0       0.88      0.97      0.92     47987
           1       0.96      0.86      0.91     47236

    accuracy                           0.92     95223
   macro avg       0.92      0.92      0.92     95223
weighted avg       0.92      0.92      0.92     95223


----------------------------
```

We checked the confusion matrix for False-positive and False-negative count.

Checking Confusion Matrix of Top 5 models

```
print('Confusion Matrix for LR is : \n',confusion_matrix(y_test_malig,lr_malig_pred),'\n')
print('----------------------------')
print('Confusion Matrix for DTC is : \n',confusion_matrix(y_test_malig,dtc_malig_pred),'\n')
print('----------------------------')
print('Confusion Matrix for RFC is : \n',confusion_matrix(y_test_malig,rfc_malig_pred),'\n')
print('----------------------------')
print('Confusion Matrix for MNB is : \n',confusion_matrix(y_test_malig,mnb_malig_pred),'\n')
print('----------------------------')
print('Confusion Matrix for XGB is : \n',confusion_matrix(y_test_malig,xgb_malig_pred),'\n')
print('----------------------------')
```

```
Confusion Matrix for LR is :
 [[43868  4119]
 [ 2137 45099]]

----------------------------
Confusion Matrix for DTC is :
 [[44171  3816]
 [ 1310 45926]]

----------------------------
Confusion Matrix for RFC is :
 [[46873  1114]
 [  350 46886]]

----------------------------
Confusion Matrix for MNB is :
 [[45024  2963]
 [ 5950 41286]]

----------------------------
Confusion Matrix for XGB is :
 [[46369  1618]
 [ 6427 40809]]

----------------------------
```

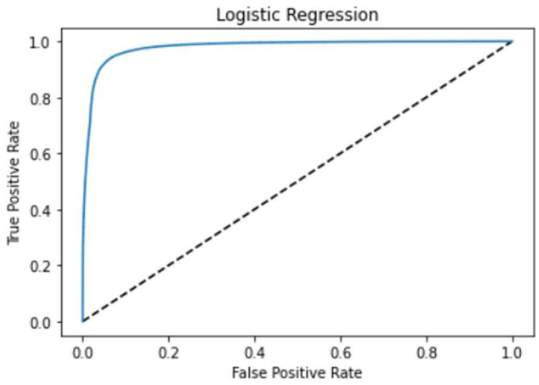
We observe that the best Recall, Precision, and F1 score for all the classes are given by Radom Forest Algorithm.

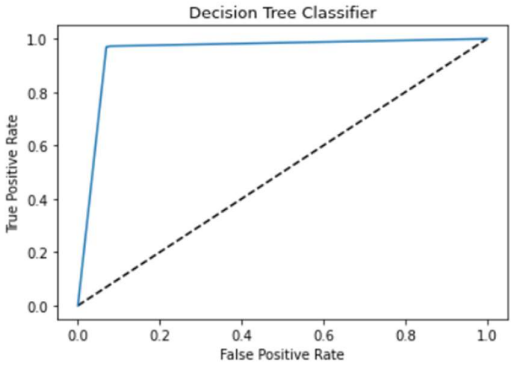
We plotted the AUC ROC curve and checked the AUC score for top 5 models for each category.

```
# Importing Auc Roc
from sklearn.metrics import roc_curve,roc_auc_score
```

```
algo_name=['Logistic Regression','Decision Tree Classifier','Random Forest Classifier','Multinomial NB','XGB Classifier']
model_name_malig=[lr_malig,dtc_malig,rfc_malig,mnb_malig,xgb_malig]
```

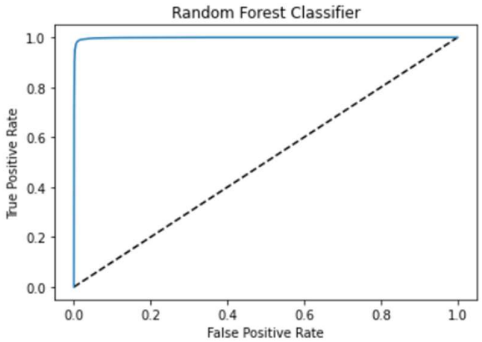
```
for j, k in zip(model_name_malig,algo_name):

    y_pred_prob=j.predict_proba(x_test_malig)[:,1]
    fpr,tpr,thresholds=roc_curve(y_test_malig,y_pred_prob)

    plt.plot([0,1],[0,1],'k--')
    plt.plot(fpr,tpr,label=k)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(k)
    plt.show()

    # Auc Score
    print('\n---------------------\n')
    print(f'Auc Score of {k}:\n')
    auc_score=roc_auc_score(y_test_malig,y_pred_prob)*100
    print(auc_score)
    print('\n-------------------')
```

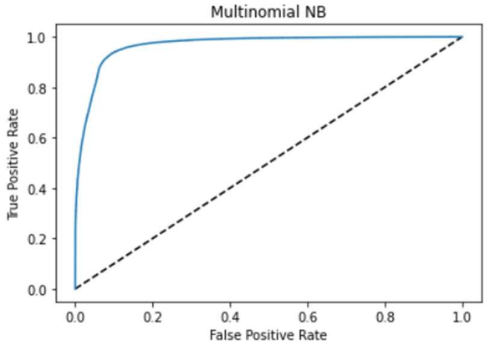



```
---------------------

Auc Score of Logistic Regression:

98.00020766361091

-------------------
```

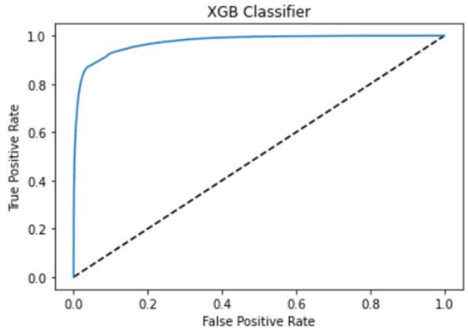
```
---------------------

Auc Score of Decision Tree Classifier:

95.12706116371106

-------------------
```




```
---------------------

Auc Score of Random Forest Classifier:

99.87378038932881

-------------------
```

```
---------------------

Auc Score of Multinomial NB:

96.77685077642165

-------------------
```




```
--------------------

Auc Score of XGB Classifier:

97.42290204002681

--------------------
```

Similarly, we plotted AUC ROC curve for all the models for each category. RFC has the best performance in all the categories.

## Hypertuning

We Hypertuned the algorithm to check for any betterment.

```
# Importing Random Search CV

from sklearn.model_selection import RandomizedSearchCV
```

```
# defining parameters for Random Forest


rfc_param={'n_estimators':list(range(50,400,50)),'max_depth' : np.arange(2,15),'criterion':['gini','entropy'],'max_features':['au
rcv_rfc= RandomizedSearchCV(estimator=rfc_malig,param_distributions=rfc_param,scoring='accuracy',cv=5)
```

```
# Getting Best Parameters

rcv_rfc.fit(x_train_malig,y_train_malig)
rcv_rfc.best_params_
```

```
{'n_estimators': 100,
 'max_features': 'sqrt',
 'max_depth': 14,
 'criterion': 'entropy'}
```

```
rfc1_malig=RandomForestClassifier(criterion='entropy',max_depth=14,max_features='sqrt',n_estimators=100)
rfc1_malig.fit(x_train_malig,y_train_malig)
rfc1_malig_pred=rfc1_malig.predict(x_test_malig)

print('Accuracy Score of the model is : \n',accuracy_score(y_test_malig,rfc1_malig_pred)*100,'\n')
print('Confusion Matrix is : \n',confusion_matrix(y_test_malig,rfc1_malig_pred),'\n')
print('Classification Report is : \n',classification_report(y_test_malig,rfc1_malig_pred),'\n')
```

```
Accuracy Score of the model is :
 81.1694653602596

Confusion Matrix is :
 [[33664 14323]
 [ 3608 43628]]

Classification Report is :
              precision    recall  f1-score   support

           0       0.90      0.70      0.79     47987
           1       0.75      0.92      0.83     47236

    accuracy                           0.81     95223
   macro avg       0.83      0.81      0.81     95223
weighted avg       0.83      0.81      0.81     95223
```

Hypertuning didn't help much with the performance. Thus, We move ahead with the Random Forest With Default Parameters

We hyper-tuned the model in all the categories. After hyper-tuning the model we didn't find any improvement in the accuracy. Thus, we move ahead with Random Forest Algorithm as our final model with default parameters.

## Key Metrics for success in solving the problem under consideration

Metrics used for solving the current scenario:

Accuracy: The accuracy of any machine learning model is defined as the closeness of a measured value to a true value. That means the closer the measured value to the true value, the better is the accuracy.

We get the accuracy in our model as

1. Malignant - RFC – 98.34%.
2. Highly Malignant – RFC – 99.59%
3. Rude – RFC – 99.12%
4. Threat – RFC – 99.94%
5. Abuse – RFC – 99.19%
6. Loathe – RFC – 99.50%

Thus, we can say that our models are these much accurate while predicting the true values.

Precision: Precision refers to the closeness of two or more measurements to each other. Accuracy and Precision are independent of each other. A model can be highly precise but not necessarily accurate.

In our model, we get the precision accuracy at 98 to 100%. High precision value enables our model to reduce False Positive prediction.

Recall: We are taking recall as the evaluation metric in our case as there is a high value for false-negative prediction. If in our case a 0(actual positive) is predicted as 1(predicted negative), this can cause a problem.

F1 Score: F1 score is the harmonic mean of Precision and Recall. It calculates a balance between precision and recall. F1 score in our model is 98 to 100% which ensures that the false-positive and false-negative predictions are less and thus making our model more efficient.

# Exploratory Data Analysis and Visualization

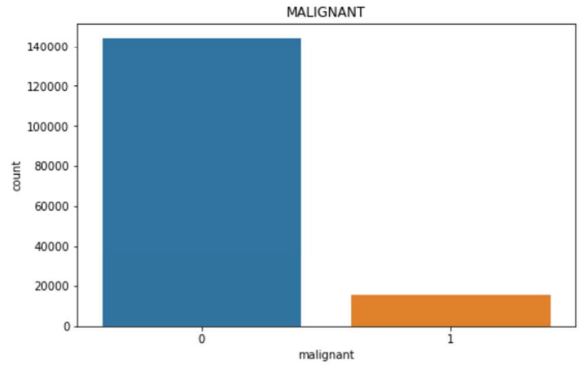
We performed analysis on the dataset and inferred some info regarding the comment pattern of the users.

We created a list of columns to iterate through it.

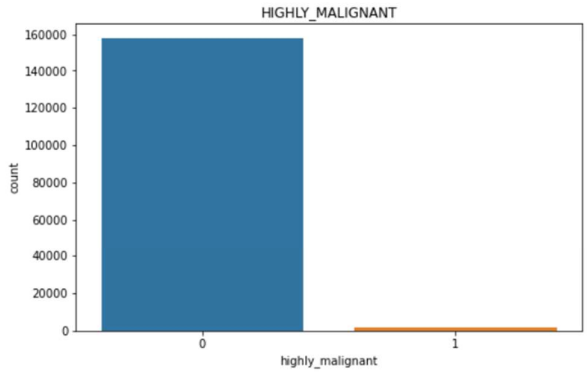
```
columns=['malignant','highly_malignant','rude','threat','abuse','loathe']
```

## Checking the count of each category

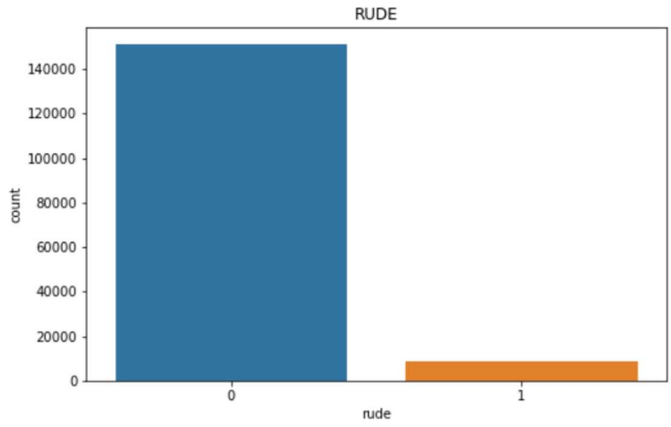
```
for x in columns:
    plt.figure(figsize=(8,5))
    sns.countplot(df1[x])
    plt.title(str.upper(x))
    plt.show()

    print(df1[x].value_counts(normalize=True)*100)
```



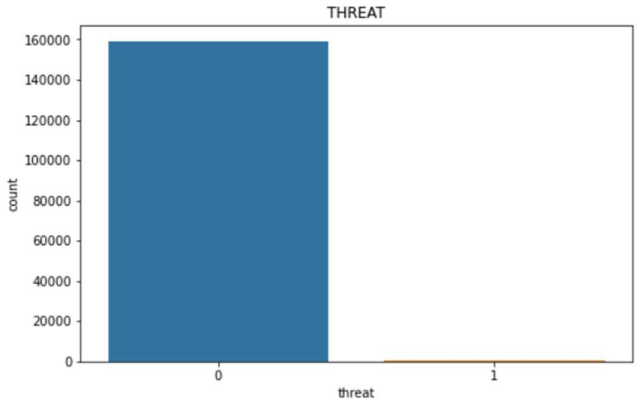

```
0    90.415552
1     9.584448
Name: malignant, dtype: float64
```



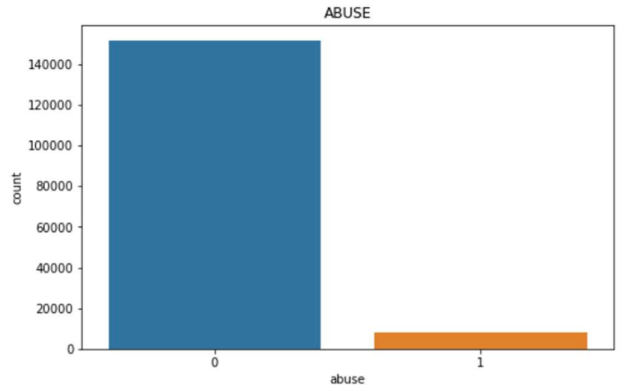

```
0    99.000445
1     0.999555
Name: highly_malignant, dtype: float64
```



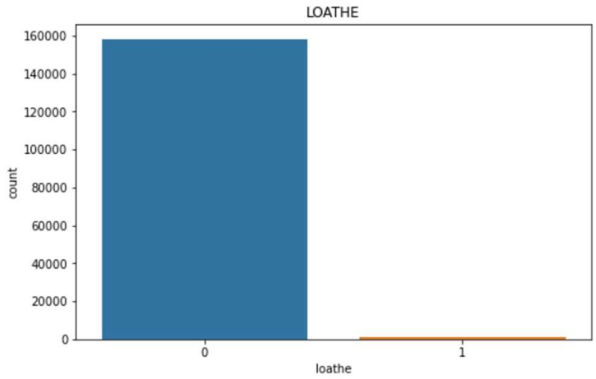

```
0    94.705178
1     5.294822
Name: rude, dtype: float64
```




```
0    99.700447
1     0.299553
Name: threat, dtype: float64
```




```
0    95.063639
1     4.936361
Name: abuse, dtype: float64
```



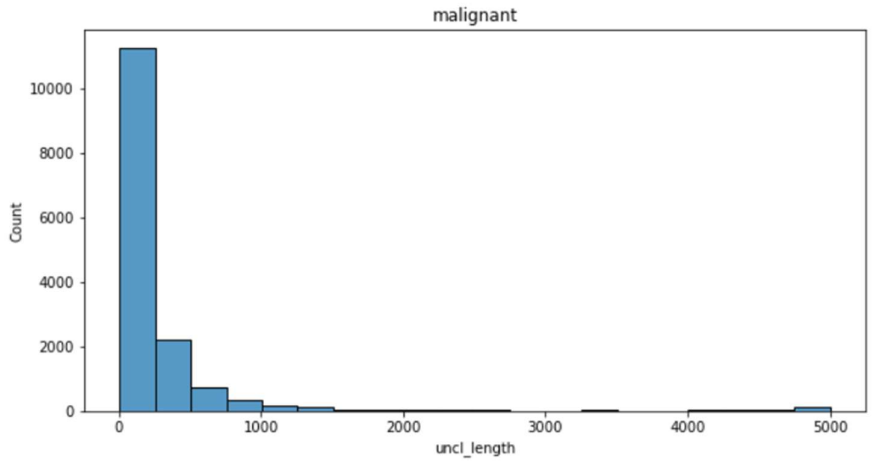

```
0    99.119514
1     0.880486
Name: loathe, dtype: float64
```

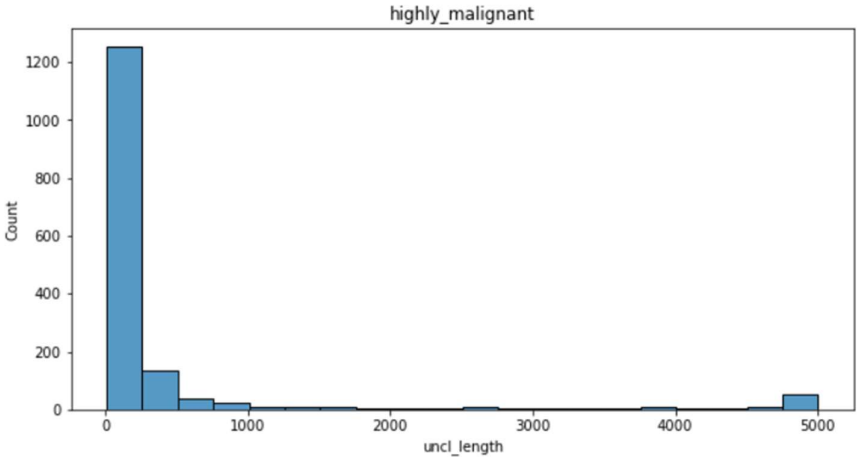
The highest Count we have is for Malignant comments which are 9.58% while the least is for threat comments which are 0.22% only.

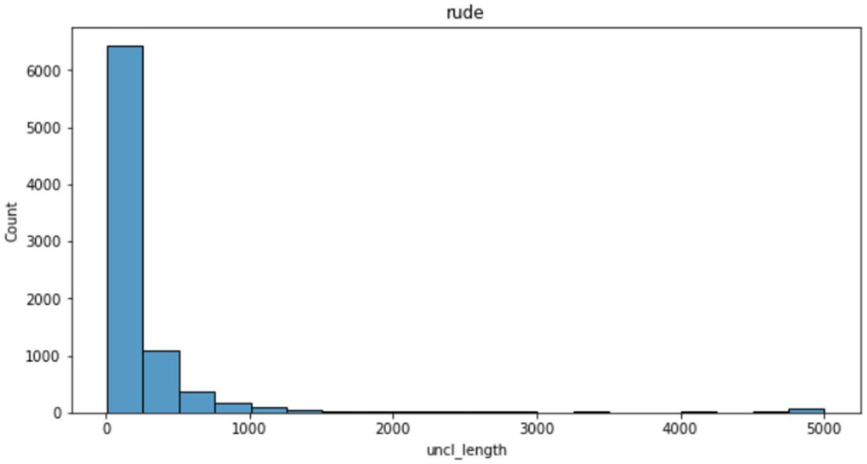
## Checking the string length of comments

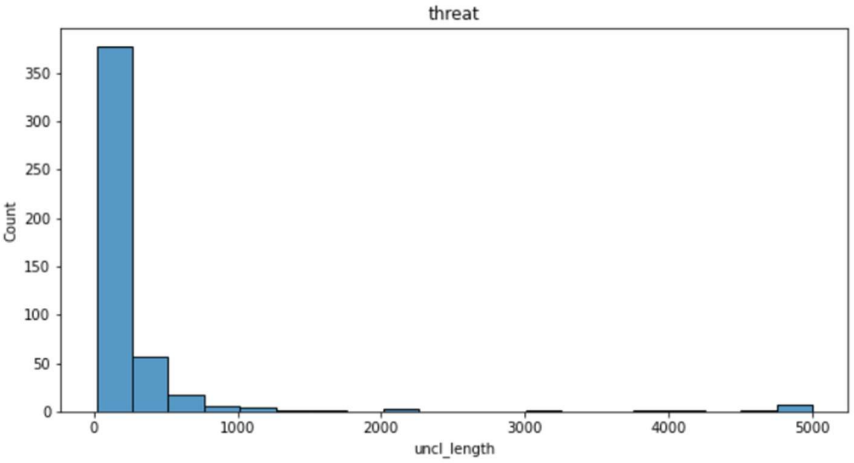
```
columns=['malignant','highly_malignant','rude','threat','abuse','loathe']
```

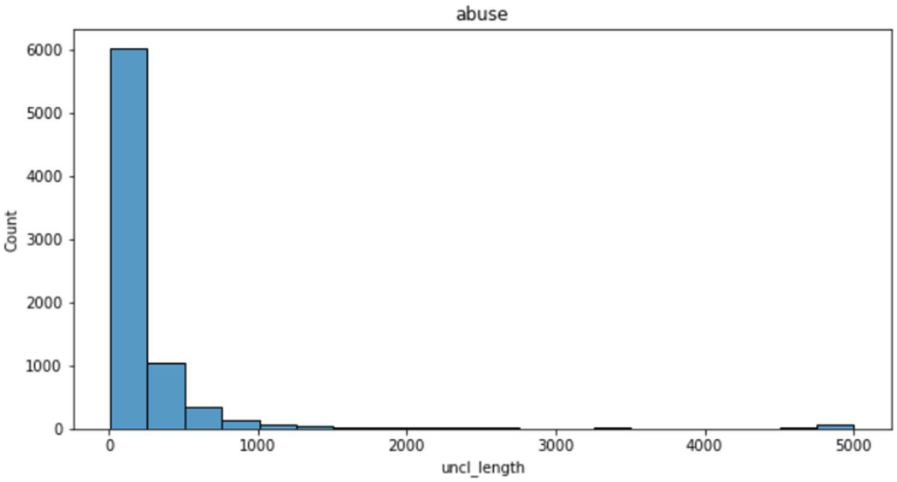
```
for column in columns:
    plt.figure(figsize=(10,5))
    sns.histplot(df1[df1[column]==1]['uncl_length'],bins=20)
    plt.title(column)
    plt.show()
```

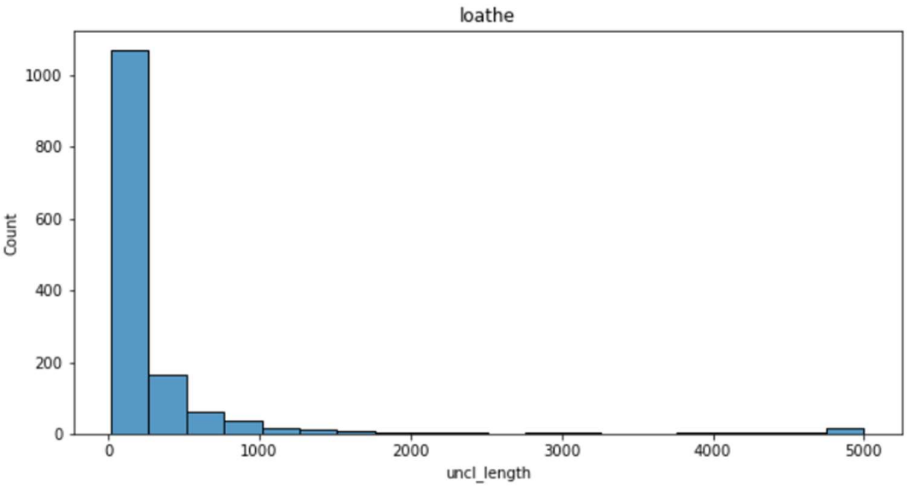



rude
Count
uncl_length
0
1000
2000
3000
4000
5000
6000


threat
Count
uncl_length
0
50
100
150
200
250
300
350


abuse
Count
uncl_length
0
1000
2000
3000
4000
5000
6000


loathe
Count
uncl_length
0
200
400
600
800
1000

We observe that comments majorly range to 200 characters and extend to 1000 characters. Some comments have 5000 characters in them. We see that comments which have 5000 characters are more in the Highly malignant category.

## Checking string length of normal comments

We created a column ‘bad’ which stores 1 if the comment falls under any category and a 0 if falls under none.

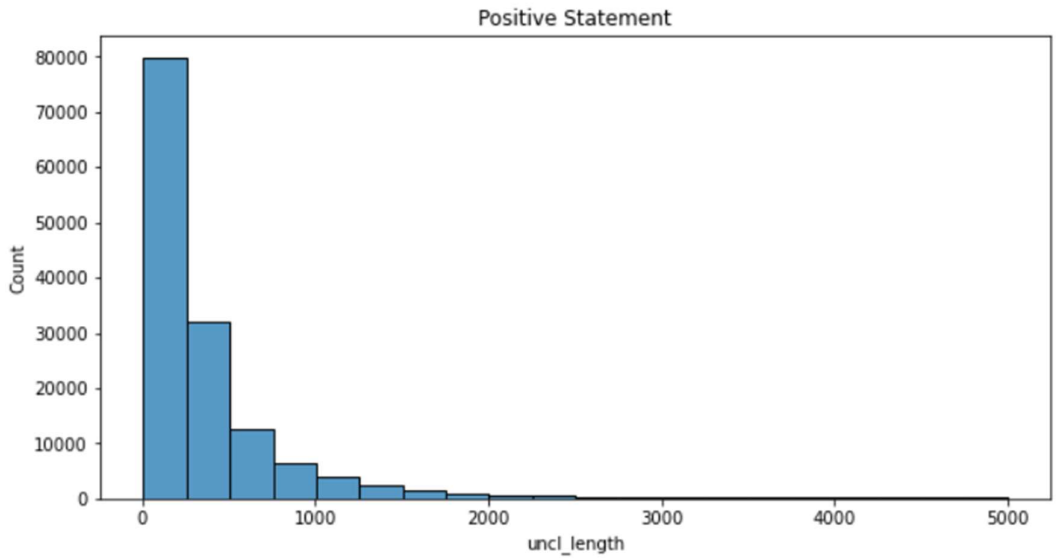
```
# Creating a new column Bad which will mark toxic statement as 1 and positive statement as 0
df1['bad']=df1.iloc[:,2:8].sum(axis=1)

for i in range(len(df1.bad)):
    if df1['bad'][i]>0:
        df1['bad'][i]=1
    else:
        pass
```

```
df1.head()
```

| | id | comment_text | malignant | highly_malignant | rude | threat | abuse | loathe | uncl_length | cleaned_length | bad |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation edits made username hardcore metal... | 0 | 0 | 0 | 0 | 0 | 0 | 264 | 170 | 0 |
| 1 | 000103f0d9cfb60f | aww match background colour I seemingly stuck ... | 0 | 0 | 0 | 0 | 0 | 0 | 112 | 93 | 0 |
| 2 | 000113f07ec002fd | hey man I really trying edit war guy constantl... | 0 | 0 | 0 | 0 | 0 | 0 | 233 | 143 | 0 |
| 3 | 0001b41b1c6bb37e | cannot make real suggestion improvement wonder... | 0 | 0 | 0 | 0 | 0 | 0 | 622 | 372 | 0 |
| 4 | 0001d958c54c6e35 | sir hero chance remember page | 0 | 0 | 0 | 0 | 0 | 0 | 67 | 29 | 0 |

Then checked the string count for comment which has 0 in the ‘bad’ column

```
plt.figure(figsize=(10,5))
sns.histplot(df1[df1['bad']==0]['uncl_length'],bins=20)
plt.title('Positive Statement')
plt.show()
```




We observe that the string count for normal comments is major up to 200 characters and also has a significant number for up to 500 characters. Although we see a spread to 200 characters it doesn’t go any further much.

## Correlation Matrix

We plotted a correlation matrix to check the correlation between various categories

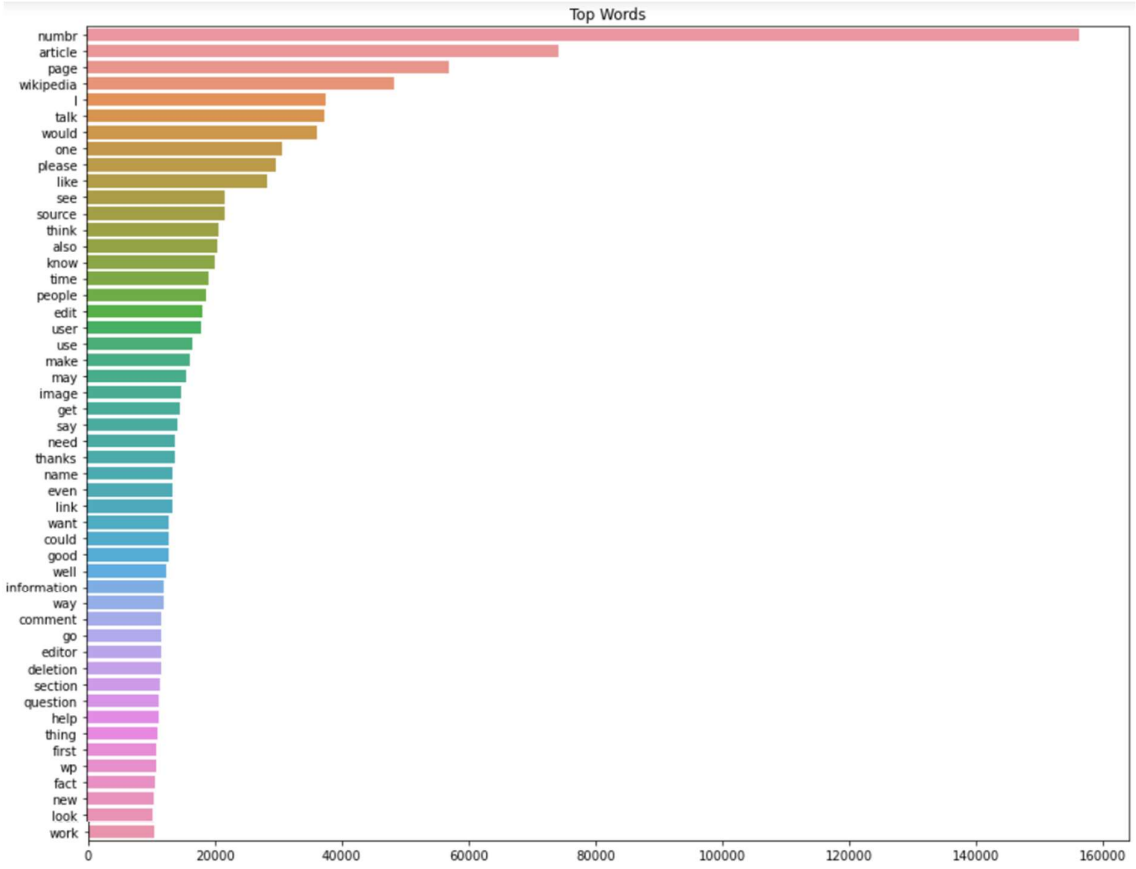
```
df1_corr=df_train.corr()

# Plotting a heatmap of Correlation to visualize

plt.figure(figsize=(12,8))
sns.heatmap(df1_corr,annot=True,fmt='.2f')
plt.show()
```


We see a high correlation between rude and abusive comments. There is also a good correlation between rude-malignant and malignant-abuse pair.

## Checking Top Words

```
from collections import  Counter

def top_words(text):
    stop=set(stopwords.words('english'))
    new= text.str.split()
    new=new.values.tolist()
    corpus=[word for i in new for word in i]

    counter=Counter(corpus)
    most=counter.most_common()
    x, y=[], []
    for word,count in most[:50]:
        if (word not in stop):
            x.append(word)
            y.append(count)

    plt.figure(figsize=(15,12))
    sns.barplot(x=y,y=x)
    plt.title('Top Words')
    plt.show()
```

```
top_words(df1['comment_text'])
```



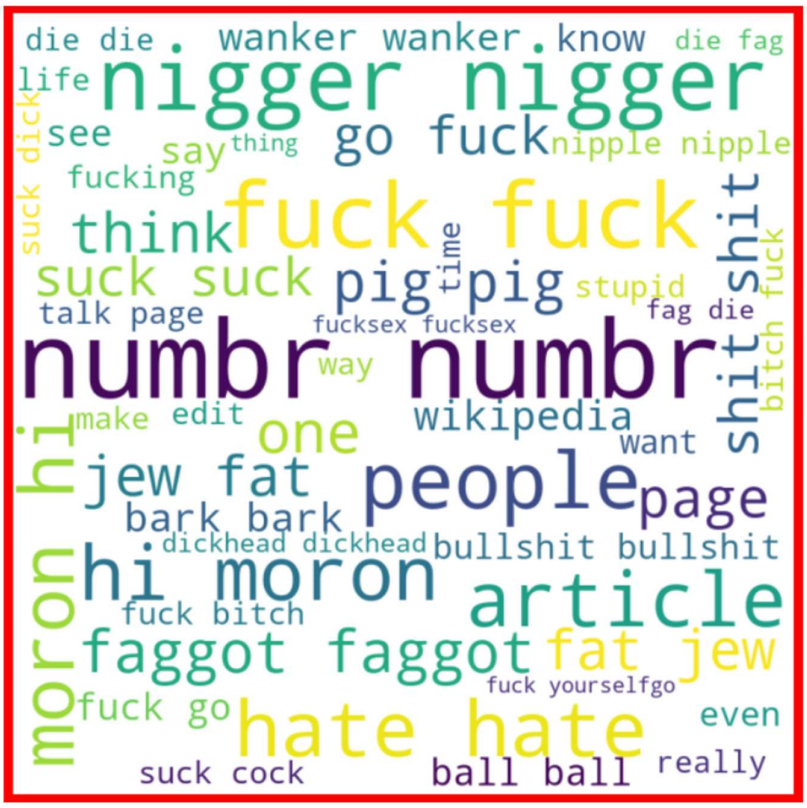

We replaced digits with 'numbr'. We see a high count of digits in the text. Article, page, Wikipedia also has a high count. This refers to a conversation related to articles on Wikipedia or related to Wikipedia article pages.

## WordCloud

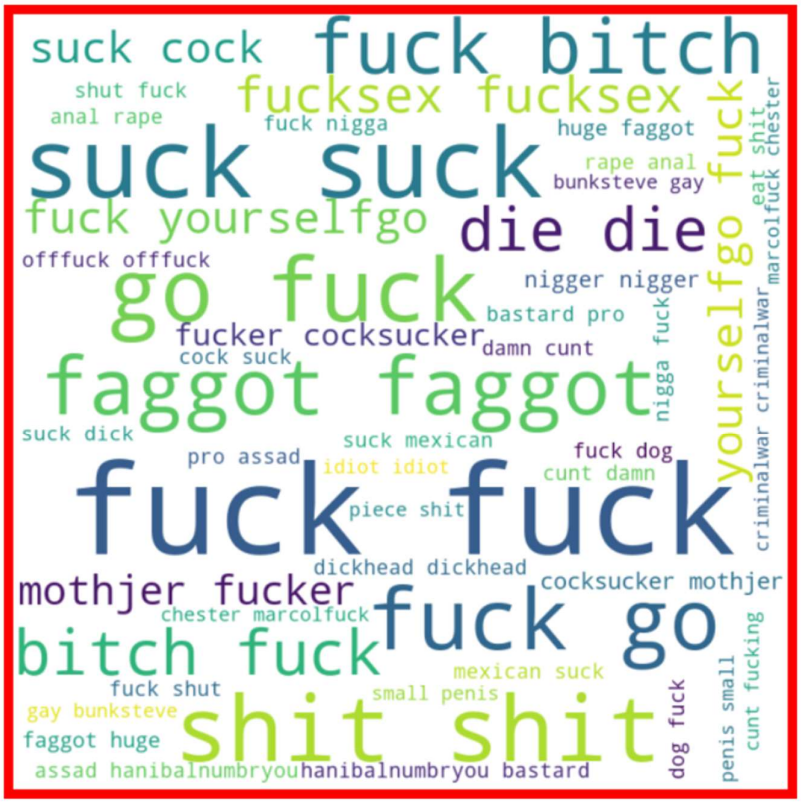
We plotted Word Cloud to see the most occurring word for each category.

```
from wordcloud import WordCloud, STOPWORDS
```

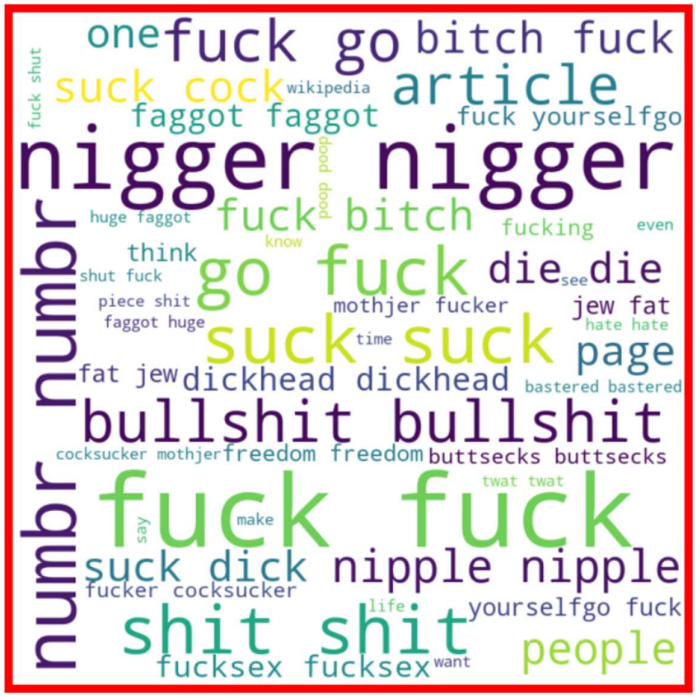
```
com_malig=df1['comment_text'][df1['malignant']==1]

wc_1= WordCloud(width = 800, height = 800,
                background_color ='white',
               max_words=50,
                min_font_size = 10).generate(' '.join(com_malig))
plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(wc_1)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```




```
com_hmalig=df1['comment_text'][df1['highly_malignant']==1]

wc_2= WordCloud(width = 800, height = 800,
                background_color ='white',
               max_words=50,
                min_font_size = 10).generate(' '.join(com_hmalig))
plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(wc_2)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



```
com_rude=df1['comment_text'][df1['rude']==1]

wc_3= WordCloud(width = 800, height = 800,
                background_color ='white',
               max_words=50,
                min_font_size = 10).generate(' '.join(com_rude))
plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(wc_3)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```




```
com_threat=df1['comment_text'][df1['threat']==1]

wc_4= WordCloud(width = 800, height = 800,
                background_color ='white',
               max_words=50,
                min_font_size = 10).generate(' '.join(com_threat))
plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(wc_4)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

```
com_abuse=df1['comment_text'][df1['abuse']==1]

wc_5= WordCloud(width = 800, height = 800,
                background_color ='white',
               max_words=50,
                min_font_size = 10).generate(' '.join(com_abuse))
plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(wc_5)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```


```
com_loathe=df1['comment_text'][df1['loathe']==1]

wc_6= WordCloud(width = 800, height = 800,
                background_color ='white',
               max_words=50,
                min_font_size = 10).generate(' '.join(com_loathe))
plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(wc_6)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Observation:

1. All the comments falling under any of the categories usually have abusive, racist, or discriminating, provoking words.
2. Malignant comments have nigger, fuck, faggot, pig as most used words. There we see a high number of 'numbr' which denotes digits.
3. Highly malignant comment again has fuck, faggot as common with malignant. There are words like shit, bitch, mother fu**er, cocksucker usually seen in this category.
4. Rude comments have extensive use of words like nigger, bullshit, fuck, suck, nipple, bitch, faggot. As we see a correlation between malignant and rude category we also see common words in them. Numbr again appears several times.
5. Threat column has die, kill as most common words. Must die, going kill, murder is the most common word. We also see lifetime ban, fucking, Jim wale as most used.
6. The abuse column has abusive words. Nigger, fuck, bitch dickhead, faggot are most common. These are similar to rude and malignant comments. We see the use of words moron, fat, jew to be used as abusive words.
7. Loathe comments have common words like nigger, die, faggot, etc. Similar to abuse it also contains fat, jew. We see bunksteve, gay, Mexican, Jewish, licker as different words denoting a loathe comment.

## Interpretation of the Results

We observe that the negative comment falling under any of the categories is usually long in terms of string length. Most of the columns contain abusive words like nigger, fuck, suck, etc. Threat column has die, kill must kill, going die as most common.

# *CONCLUSION*

## Key Findings and Conclusions of the Study

One of the major points is the sentiment of the user and the user's point of view. As we saw, people use intended inappropriate words. This use of words is intentional which creates an environment of discomfort for other users. Users having toxic behavior or have frustration usually use social media platforms to release it. Inappropriate words like fuck, nigger, bitch, suck, etc are used for such types of comments. Comment length for these is between 0 to 300 characters. These comments can range up to 5000 characters where neutral comments are rarely that long.

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

Comment classification requires working on the text data which represents a person's sentiment and thoughts. It is a challenging task to work through the language data and gets the useful insights and outcome of it. Natural language processing is a complex task that requires many different ways to be tested upon. In our case, we process the raw data obtained from various websites. Using Natural Language Processing techniques we tried to clean the data and only keep the text which can prove to be helpful in the machine learning process. We often opt for different methods and techniques to keep making our model better and better.

## Limitations of this work and Scope for Future Work

Working on the language data can be tricky. Here, if we can get the data about the user's info like age, country, etc. It can help in identifying the age group of the users usually doing these kinds of comments. Country can also give us a peep about region-specific classification. Models can be incorporated into the platforms to identify words as toxic or inappropriate at the time they are posted. Some platforms are already working in this direction.