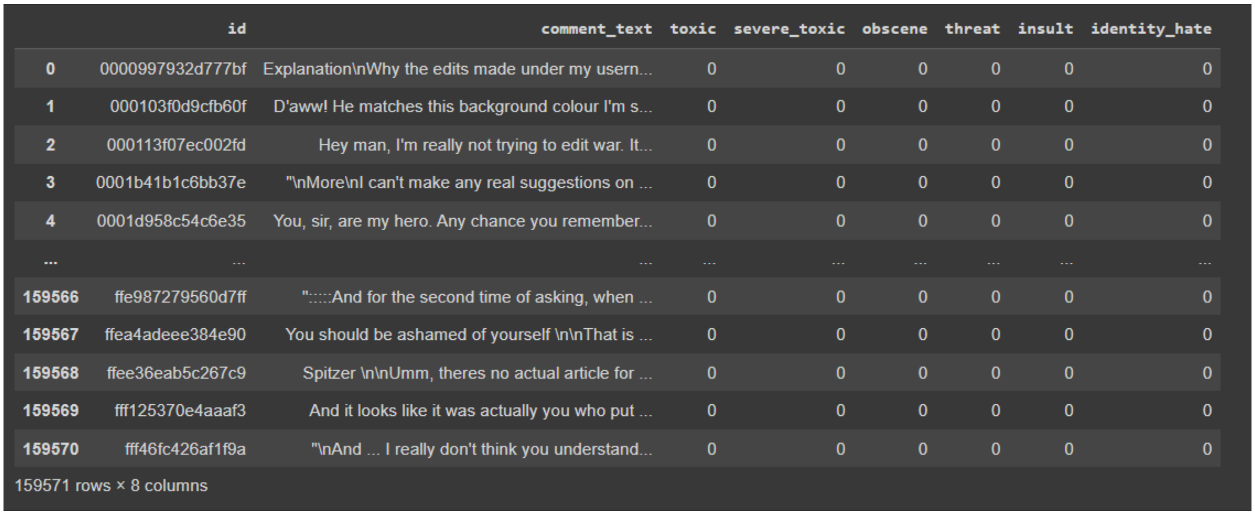
PRML MAJOR PROJECT

Toxic Comments Classification

Honey Solanki (B21EE068)
Jatin Kumar (B21EE027)
Divyanshu Singhal (B21CS025)

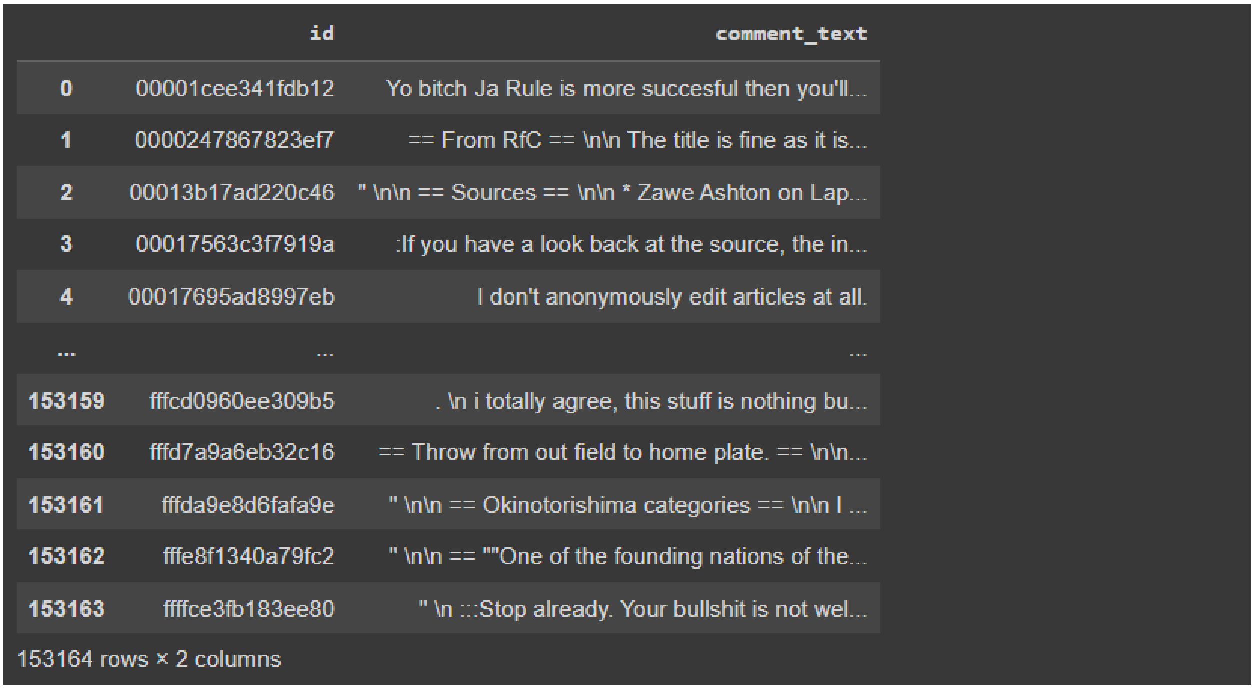
Data Preprocessing :

- ➔ Imported necessary libraries namely numpy, pandas, seaborn, matplotlib, sklearn etc.
- ➔ Read the dataset using read_csv
- ➔ Gather required information of the dataset like shape, columns, etc.



	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
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
...	...	...	...	...	...	...	...	...
159566	ffe987279560d7ff	":::::And for the second time of asking, when ...	0	0	0	0	0	0
159567	ffea4adeee384e90	You should be ashamed of yourself \n\nThat is ...	0	0	0	0	0	0
159568	ffee36eab5c267c9	Spitzer \n\nUmm, theres no actual article for ...	0	0	0	0	0	0
159569	fff125370e4aaaf3	And it looks like it was actually you who put ...	0	0	0	0	0	0
159570	fff46fc426af1f9a	"\nAnd ... I really don't think you understand...	0	0	0	0	0	0

159571 rows × 8 columns



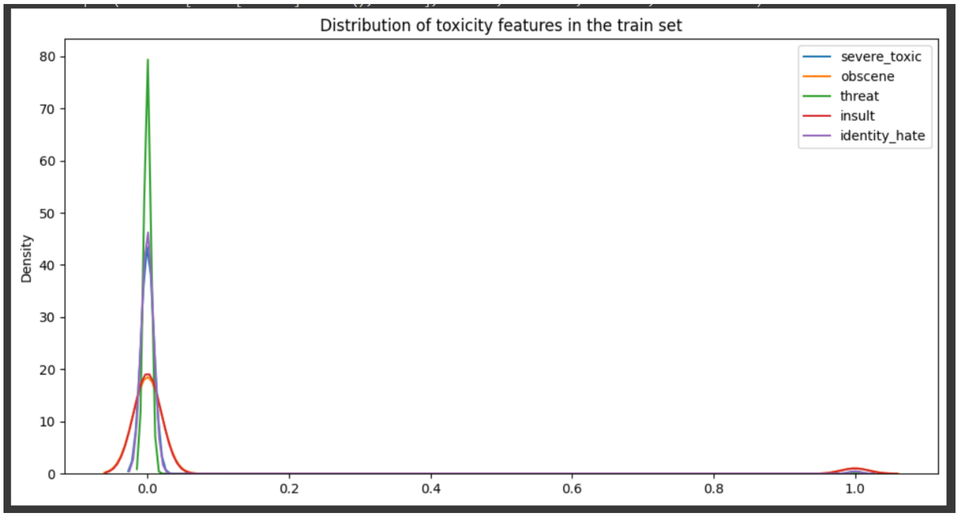
	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.
...	...	...
153159	fffcd0960ee309b5	. \n i totally agree, this stuff is nothing bu...
153160	fffd7a9a6eb32c16	== Throw from out field to home plate. == \n\n...
153161	fffda9e8d6fafa9e	" \n\n == Okinotorishima categories == \n\n I ...
153162	fffe8f1340a79fc2	" \n\n == ""One of the founding nations of the...
153163	ffffce3fb183ee80	" \n :::Stop already. Your bullshit is not wel...

153164 rows × 2 columns

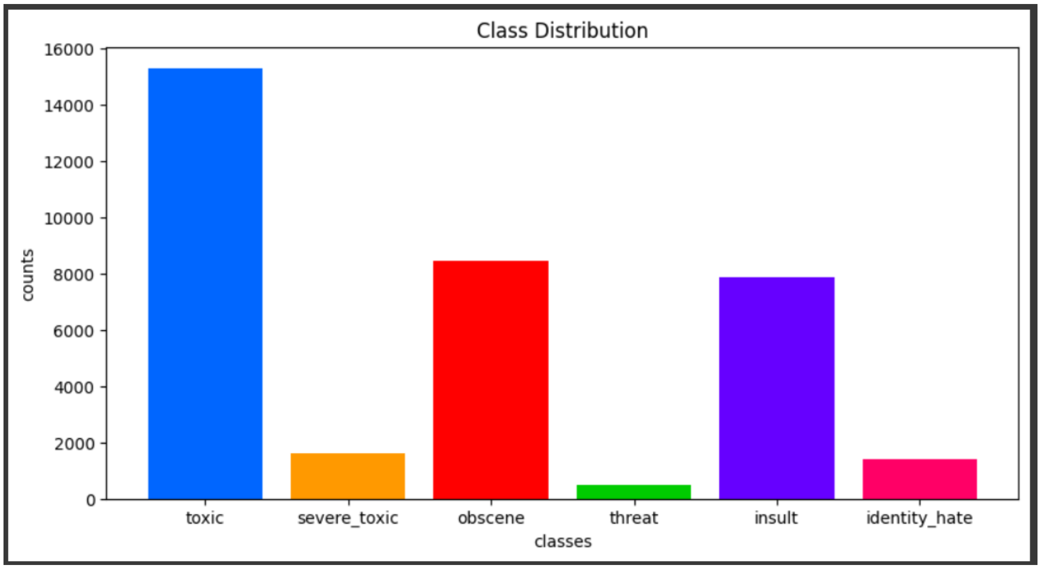
- ➔ Checked if any null value is present in the dataset and found out no null values are present.
- ➔ Then dropped I'd and comment text column and For each column, the code prints the percent of rows that have a value of 1 (i.e., the percentage of comments that belong to that class).

Data Visualization :

- ➔ Made a function that uses the Seaborn library to plot the distribution of one or more features in a dataset.
- ➔ Plotted Distribution of toxicity features in the train set.

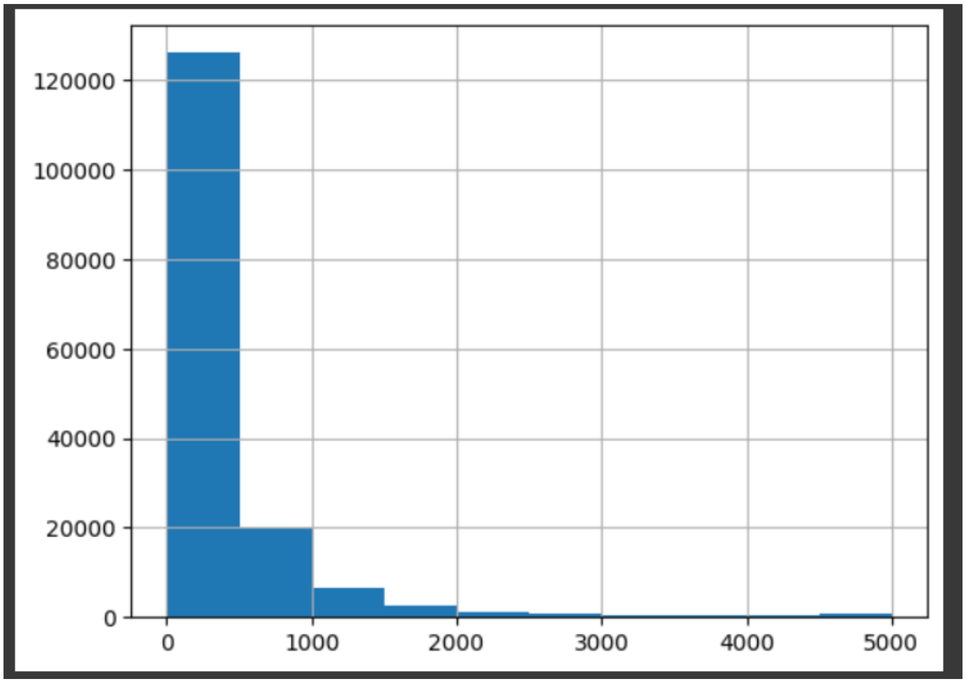


- ➔ Plotted a bar graph to visualize each column and its distribution.



Data Preprocessing :

- ➔ Calculated the count of unique values in the different columns of a DataFrame called 'train'.
- ➔ Then by using a for loop, calculated the total number of unwanted words in every column of the dataset.
- ➔ Calculated the length of each comment in the 'comment_text' column of the 'train' DataFrame and created a histogram of the 'lens' variable using ‘hist()’. The resulting plot shows the distribution of comment lengths in the 'comment_text' column of the 'train' DataFrame.

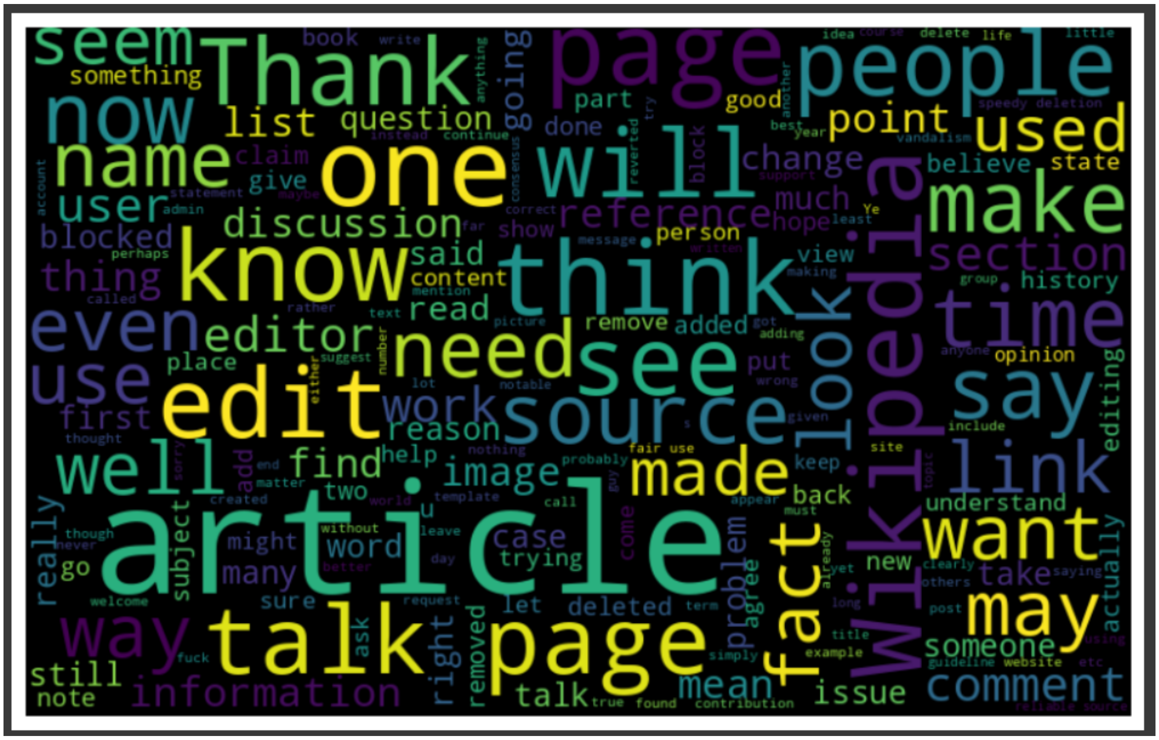


- ➔ Created a new column in the 'train' DataFrame named “text_clean” that contains the lowercase version of the values in the "comment_text" column.
- ➔ Dropped the “I’d” column in the train set.

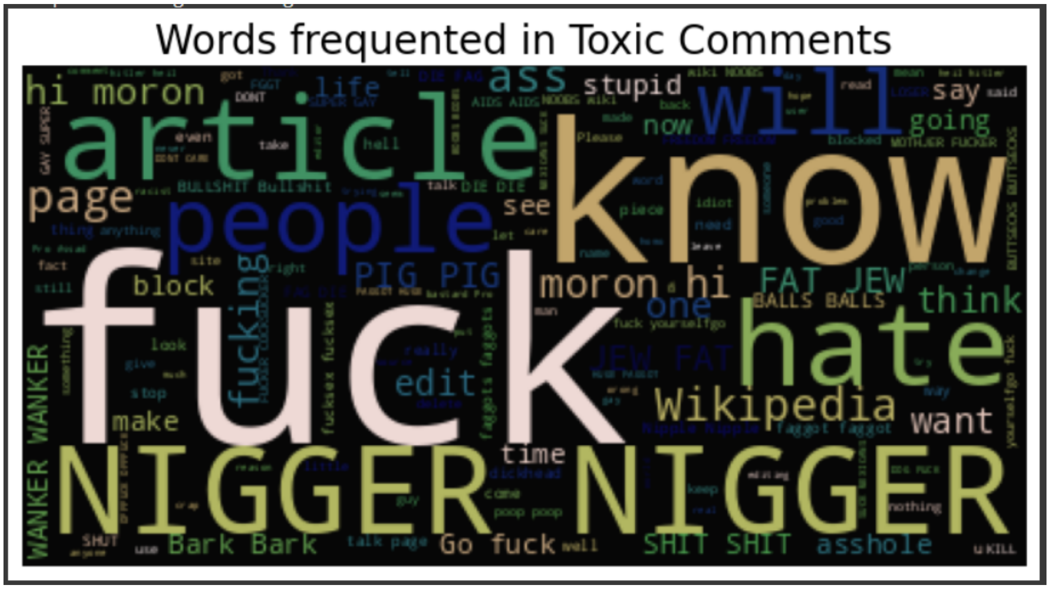
Wordclouds - Frequent words :

- ➔ Generated a visual representation of the most frequently occurring words in the text data in the 'comment_text' column of the 'train' DataFrame using a Wordcloud.
- ➔ Generates a word cloud visualization that shows the most frequently occurring words in comments that have been labeled as toxic comments.
- ➔ Similarly, generated word cloud visualization for words labeled as severely toxic Comments, Threat comments, Insult comments, Identity_hate comments.

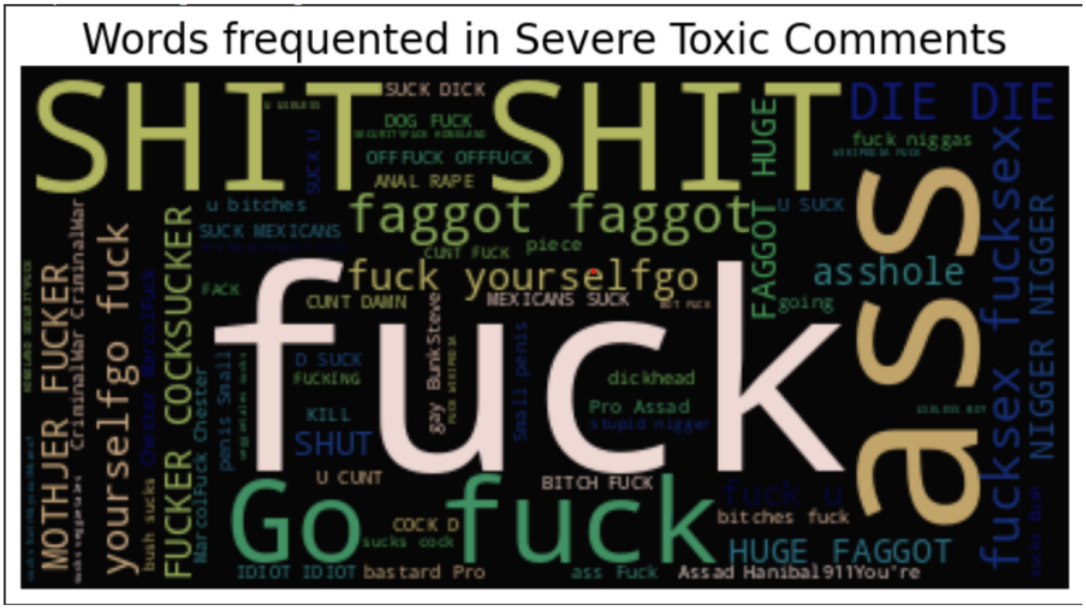
Wordcloud for clean comments :



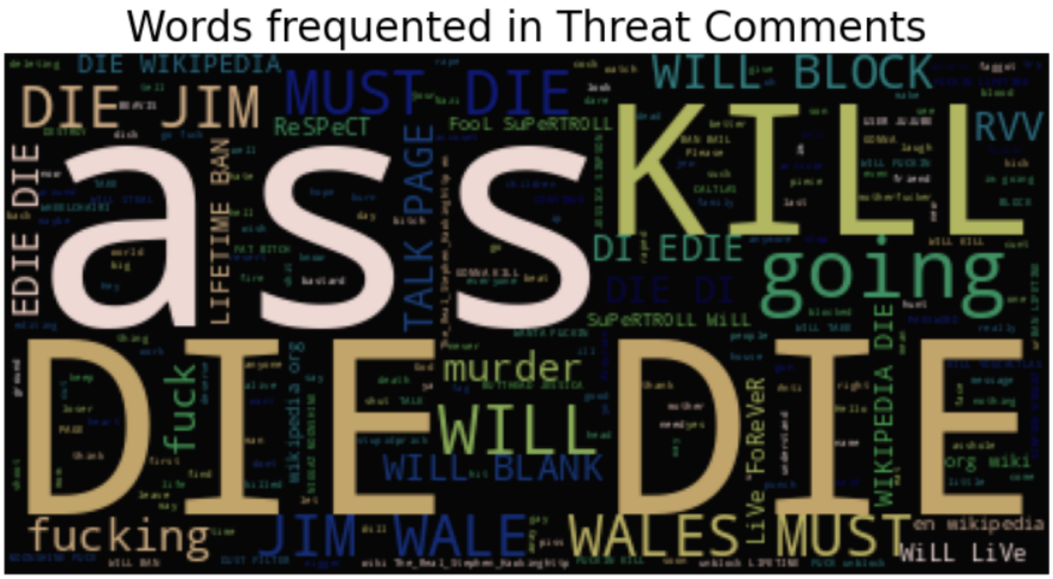
Wordcloud for Toxic comments :



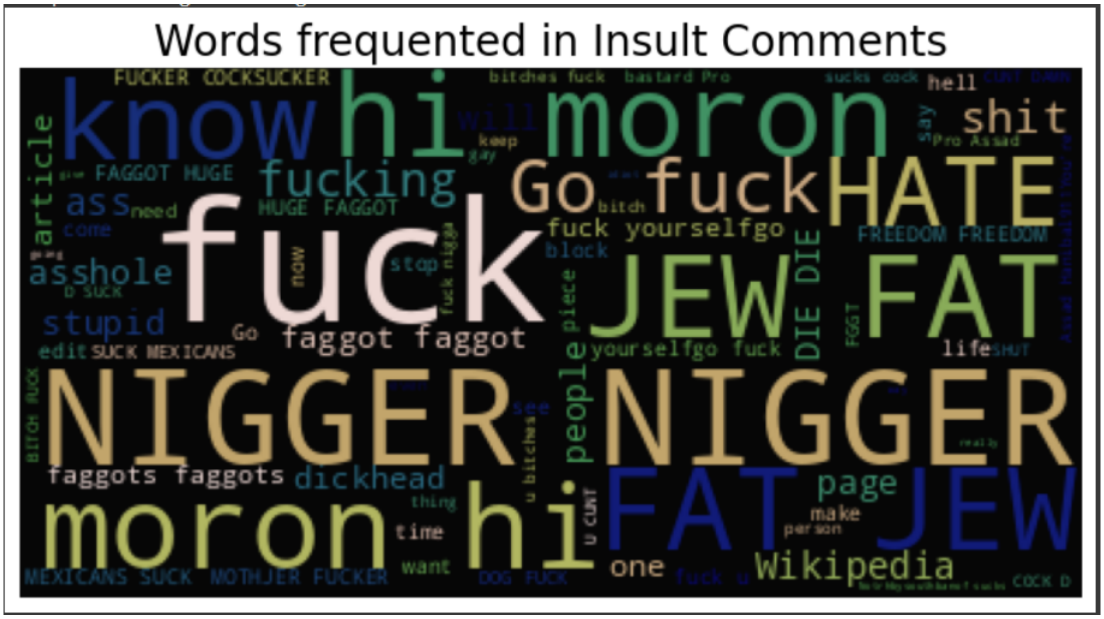
Wordcloud for Severely Toxic Comments :



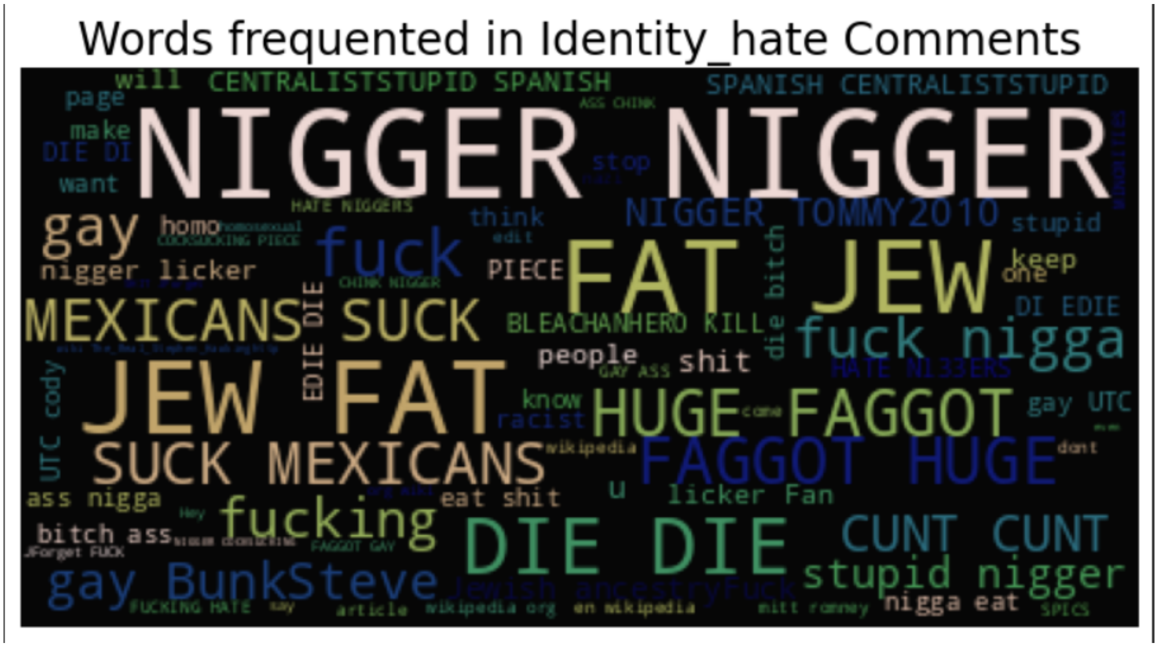
Wordcloud for Threat Comments :


Words frequented in Threat Comments

Wordcloud for Insult Comments :


Words frequented in Insult Comments

Wordcloud for Identity_hate Comments :



Text Cleaning :

➔ Made several functions to clean the text. We made functions to remove URLs, html tags, non-ascii values and the punctuations from the text data.

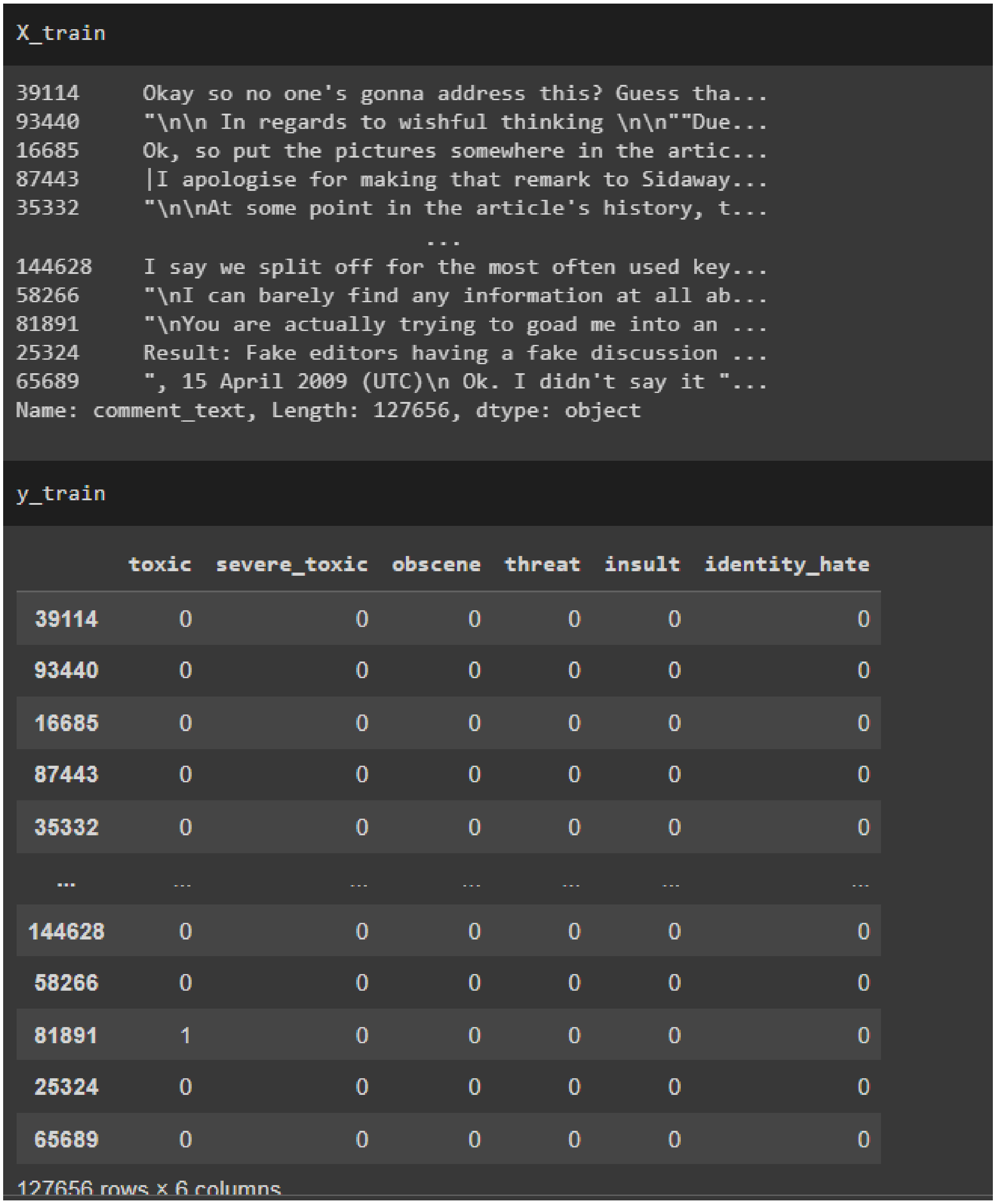
➔ Imported this library : nltk.download('punkt') and tokenized the text by using word_tokenize.

➔ Then used the library : from nltk.corpus import stopwords to remove commonly used words that are unlikely to contribute to the meaning of a text, such as 'the', 'a', 'and', etc.

➔ Used the Lancaster stemming algorithm from the NLTK library to reduce each word in a list to its base or root form. This can help to reduce the dimensionality of the text data and group together words that are similar in meaning.

Splitting the dataset :

➔ Split the dataset into train and test sets for further calculation.



```
X_train
```

```
39114     Okay so no one's gonna address this? Guess tha...
93440     "\n\n In regards to wishful thinking \n\n""Due...
16685     Ok, so put the pictures somewhere in the artic...
87443     |I apologise for making that remark to Sidaway...
35332     "\n\nAt some point in the article's history, t...
                                ...
144628    I say we split off for the most often used key...
58266     "\nI can barely find any information at all ab...
81891     "\nYou are actually trying to goad me into an ...
25324     Result: Fake editors having a fake discussion ...
65689     ", 15 April 2009 (UTC)\n Ok. I didn't say it "...
Name: comment_text, Length: 127656, dtype: object
```

```
y_train
```

	toxic	severe_toxic	obscene	threat	insult	identity_hate
39114	0	0	0	0	0	0
93440	0	0	0	0	0	0
16685	0	0	0	0	0	0
87443	0	0	0	0	0	0
35332	0	0	0	0	0	0
...	...	...	...	...	...	...
144628	0	0	0	0	0	0
58266	0	0	0	0	0	0
81891	1	0	0	0	0	0
25324	0	0	0	0	0	0
65689	0	0	0	0	0	0

127656 rows x 6 columns

Vectorization and Model Training :

➔ We have tokenize text by splitting on punctuation marks, except for certain special punctuation by using ‘re_tok’ as a regular expression.

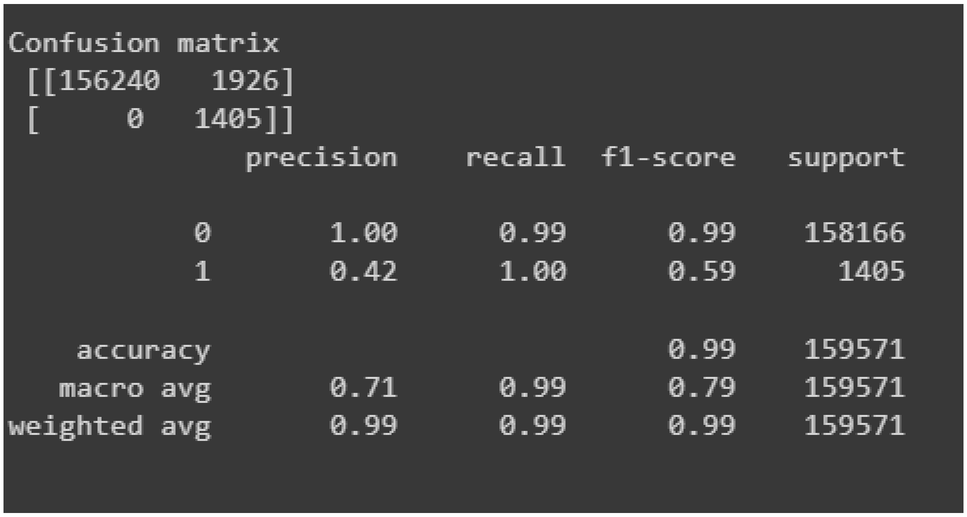
➔ Here we used “TfidfVectorizer” and using the tokenizer function tokenize() to tokenize the text, and are removing stop words.

➔ After the vectorizers are defined, the text data in train['comment_text'] and test['comment_text'] are transformed using these vectorizers, and the resulting feature vectors are stored in tr_vect and ts_vect for word-level vectorizer.

<u>For logistic regression model :</u>

➔ Trained a logistic regression model for each column (target variable) then the trained models are used to predict probabilities of each class (0 or 1) for the test set.

➔ Evaluated the performance of the logistic regression model. Printed the column name, confusion matrix, and classification report(precision, recall, F1-score, and support) of the model's predictions.

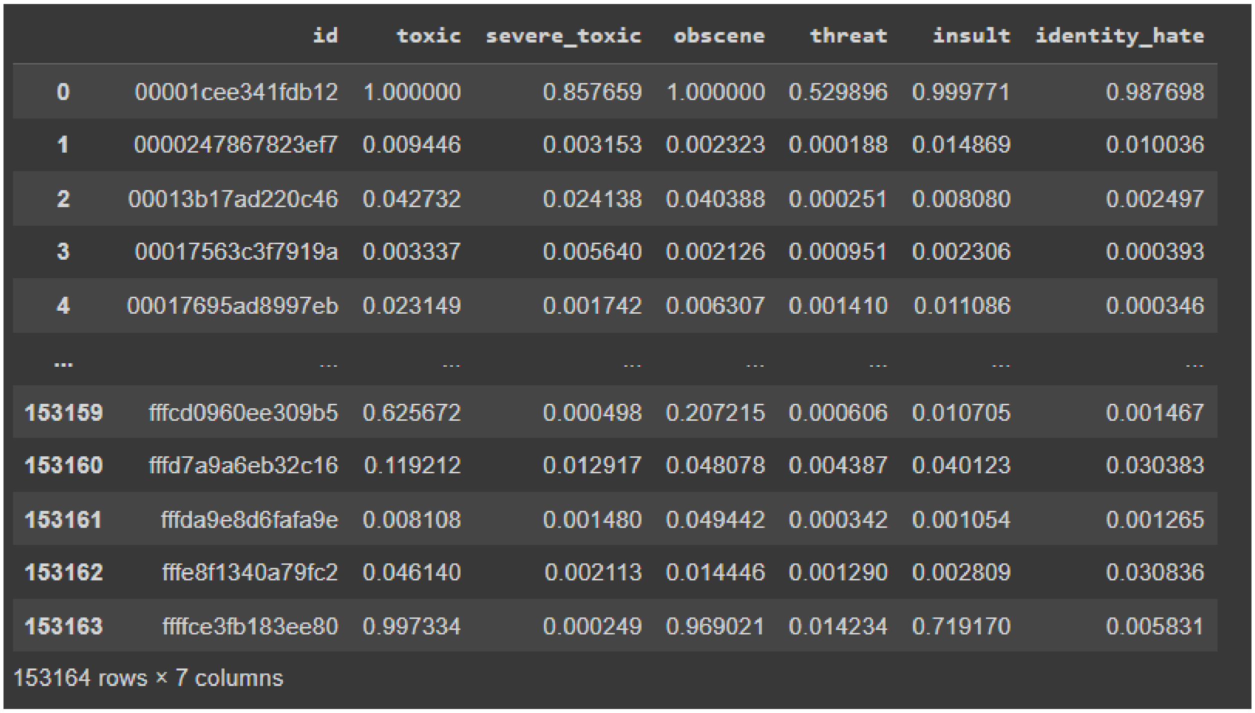


```
Confusion matrix
 [[156240   1926]
 [     0   1405]]
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    158166
           1       0.42      1.00      0.59      1405

    accuracy                           0.99    159571
   macro avg       0.71      0.99      0.79    159571
weighted avg       0.99      0.99      0.99    159571
```

➔ Then we get our required probabilities for each of the target column.



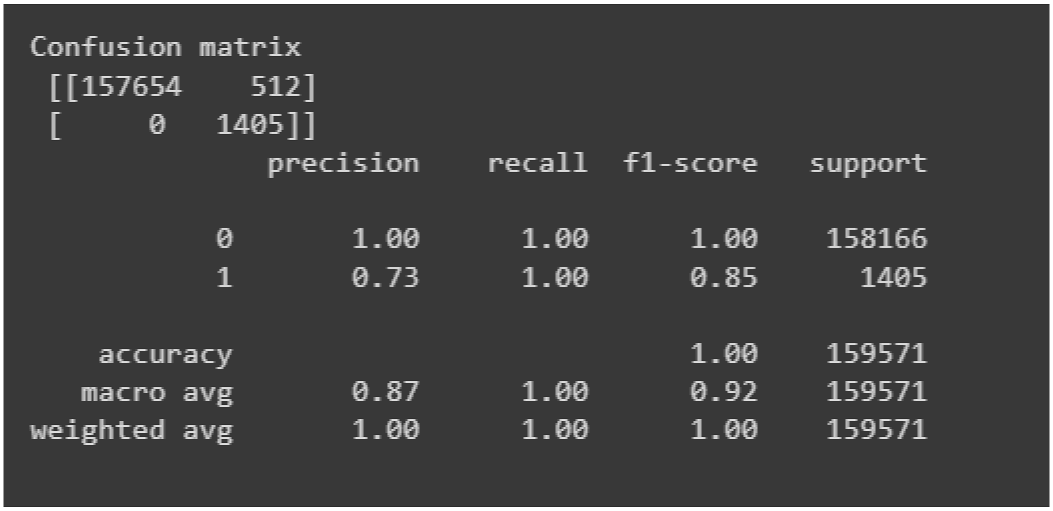
	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	1.000000	0.857659	1.000000	0.529896	0.999771	0.987698
1	0000247867823ef7	0.009446	0.003153	0.002323	0.000188	0.014869	0.010036
2	00013b17ad220c46	0.042732	0.024138	0.040388	0.000251	0.008080	0.002497
3	00017563c3f7919a	0.003337	0.005640	0.002126	0.000951	0.002306	0.000393
4	00017695ad8997eb	0.023149	0.001742	0.006307	0.001410	0.011086	0.000346
...	...	...	...	...	...	...	...
153159	fffcd0960ee309b5	0.625672	0.000498	0.207215	0.000606	0.010705	0.001467
153160	fffd7a9a6eb32c16	0.119212	0.012917	0.048078	0.004387	0.040123	0.030383
153161	fffda9e8d6fafa9e	0.008108	0.001480	0.049442	0.000342	0.001054	0.001265
153162	fffe8f1340a79fc2	0.046140	0.002113	0.014446	0.001290	0.002809	0.030836
153163	ffffce3fb183ee80	0.997334	0.000249	0.969021	0.014234	0.719170	0.005831

153164 rows × 7 columns

<u>For LinearSVC model :</u>

➔ Performed binary classification using a Linear Support Vector Classifier (LinearSVC). The CalibratedClassifierCV is then used to calibrate the probabilities of the predicted values. Then we trained the classifier on the training data (X). The output is then the predicted probabilities of the test data.

➔ Printed the column name, confusion matrix, and classification report (precision, recall, F1-score, and support) of the model's predictions.

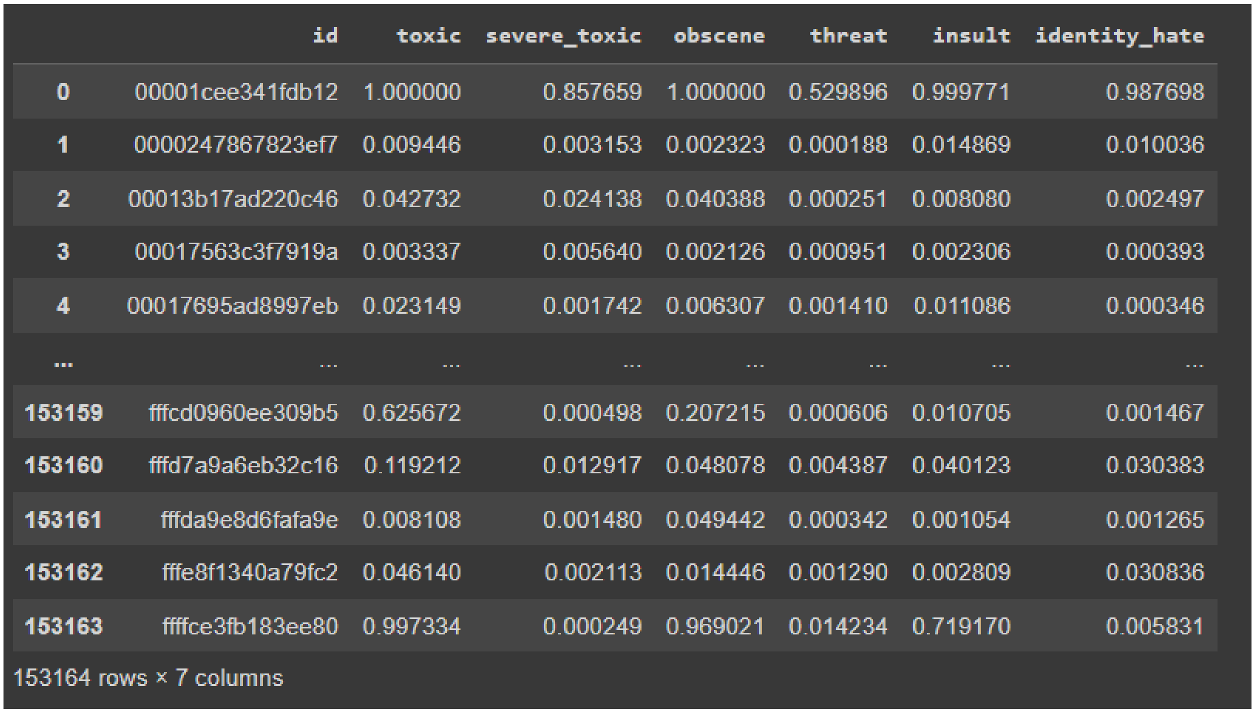


```
Confusion matrix
 [[157654    512]
 [     0   1405]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    158166
           1       0.73      1.00      0.85      1405

    accuracy                           1.00    159571
   macro avg       0.87      1.00      0.92    159571
weighted avg       1.00      1.00      1.00    159571
```

➔ Then we get our required probabilities for each of the target column.

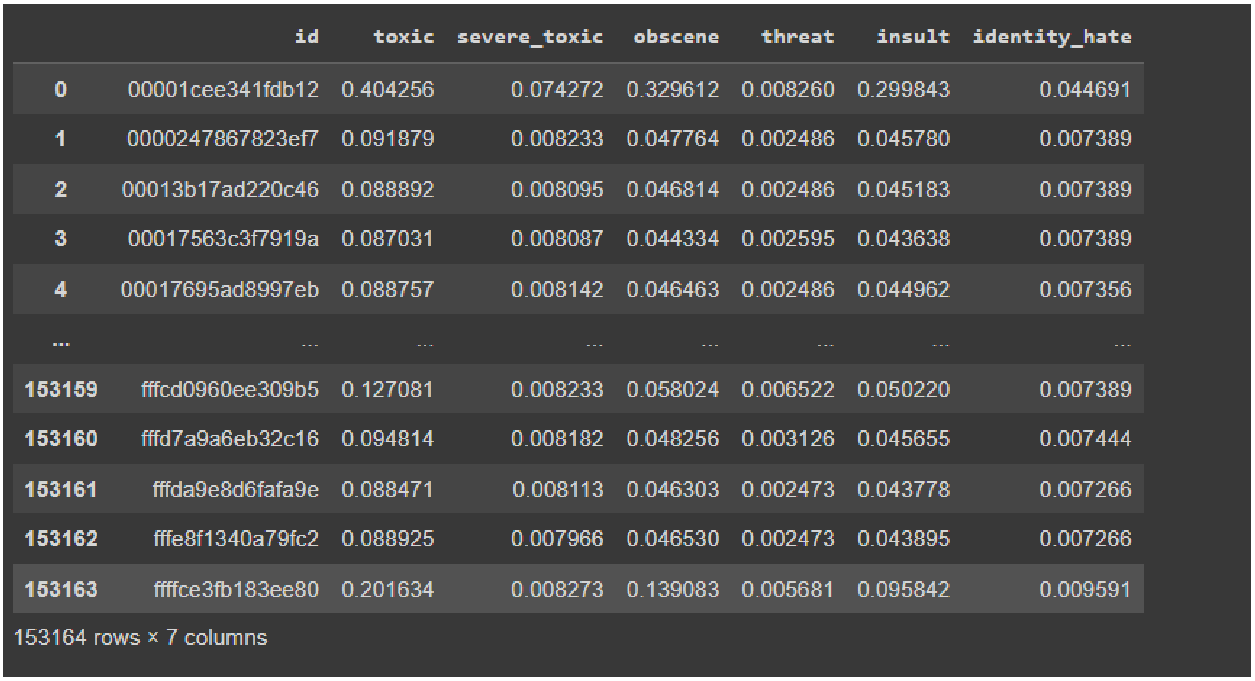


	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	1.000000	0.857659	1.000000	0.529896	0.999771	0.987698
1	0000247867823ef7	0.009446	0.003153	0.002323	0.000188	0.014869	0.010036
2	00013b17ad220c46	0.042732	0.024138	0.040388	0.000251	0.008080	0.002497
3	00017563c3f7919a	0.003337	0.005640	0.002126	0.000951	0.002306	0.000393
4	00017695ad8997eb	0.023149	0.001742	0.006307	0.001410	0.011086	0.000346
...	...	...	...	...	...	...	...
153159	fffcd0960ee309b5	0.625672	0.000498	0.207215	0.000606	0.010705	0.001467
153160	fffd7a9a6eb32c16	0.119212	0.012917	0.048078	0.004387	0.040123	0.030383
153161	fffda9e8d6fafa9e	0.008108	0.001480	0.049442	0.000342	0.001054	0.001265
153162	fffe8f1340a79fc2	0.046140	0.002113	0.014446	0.001290	0.002809	0.030836
153163	ffffce3fb183ee80	0.997334	0.000249	0.969021	0.014234	0.719170	0.005831

153164 rows × 7 columns

For Random Forest Classifier model :

- This code is creating a pipeline using scikit-learn's make_pipeline function. The pipeline consists of two main steps: (1) text preprocessing using TfidfVectorizer, and (2) multi-label classification using a OneVsRestClassifier with a Random Forest Classifier as the base Estimator. The GridSearchCV function is used to perform a cross-validated grid search over the parameter grid. The best combination of hyperparameters is selected based on the ROC AUC Score, Finally, the best model is fitted to the training data (X_train and y_train) using the fit method of the GridSearchCV object.
- Then found the accuracy which we got was 90.09%
- This code is making predictions on the test data using a pipeline that consists of a TfidfVectorizer and a OneVsRestClassifier with a RandomForestClassifier estimator.



	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	0.404256	0.074272	0.329612	0.008260	0.299843	0.044691
1	0000247867823ef7	0.091879	0.008233	0.047764	0.002486	0.045780	0.007389
2	00013b17ad220c46	0.088892	0.008095	0.046814	0.002486	0.045183	0.007389
3	00017563c3f7919a	0.087031	0.008087	0.044334	0.002595	0.043638	0.007389
4	00017695ad8997eb	0.088757	0.008142	0.046463	0.002486	0.044962	0.007356
...	...	...	...	...	...	...	...
153159	fffcd0960ee309b5	0.127081	0.008233	0.058024	0.006522	0.050220	0.007389
153160	fffd7a9a6eb32c16	0.094814	0.008182	0.048256	0.003126	0.045655	0.007444
153161	fffda9e8d6fafa9e	0.088471	0.008113	0.046303	0.002473	0.043778	0.007266
153162	fffe8f1340a79fc2	0.088925	0.007966	0.046530	0.002473	0.043895	0.007266
153163	ffffce3fb183ee80	0.201634	0.008273	0.139083	0.005681	0.095842	0.009591

153164 rows × 7 columns

Contribution of each member

Honey – I have done the Data visualization, pre processing part in the code for the toxic comment classification dataset.

Jatin – I have done vectorization and Model training along with divyanshu we both have worked upon the models. I also helped divyanshu in making the final report.

Divyanshu – I and jatin have done the vectorization and model training part. I have made the report and had done the final touch-ups in the report and the code.

Thank You