Data Collection and Preprocessing Phase

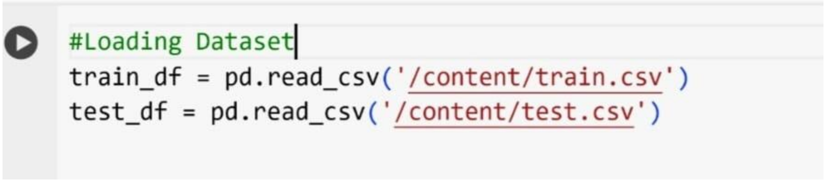
Date	20 October 2024
Team ID	739695
Project Title	Toxic Comment Classification for Social Media using NLP
Maximum Marks	6 Marks

Preprocessing Template

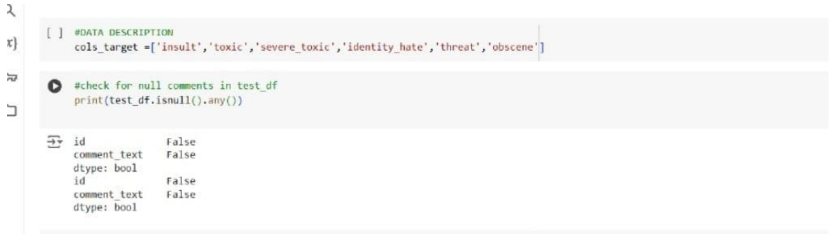
The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	The dataset consists of social media comments labeled into categories such as toxic, severe toxic, obscene, threat, insult, identity hate, and neutral. The dataset has 20,000 comments with an approximate class distribution: neutral (60%), toxic (20%), and other categories (5% each).
Resizing	Adaptation: Limit comments to a maximum number of words or characters (e.g., 100 characters). **Example Transformation**: Original: "This comment is very long and unnecessary for our processing steps." Resized: "This comment is very long."
Normalization	● Convert all text to lowercase. ● Remove unwanted characters such as URLs, HTML tags, and special characters.
Data Augmentation	• Generate synthetic data by: ○ Replacing words with synonyms. ○ Back-translation. • Example: ○ Original: "You are terrible."

	○ Synonym Replacement: "You are awful." ○ Back-translation (via Spanish): "You are horrible."
Denoising	• Remove stopwords (e.g., "is", "and", "the"). • Example: • Original: "This is an offensive comment." • Denoised: "offensive comment"
Edge Detection	• Adaptation: Extract key phrases or n-grams from text. • Example: • Original: "I hate you, you are useless." • Key Phrases: ["hate you", "are useless"]
Color Space Conversion	• Convert sentences to embeddings (e.g., Word2Vec, GloVe, or BERT embeddings).
Image Cropping	• Adaptation: Truncate text to relevant portions, e.g., first 50 words. • Example: • Original: "This is a very lengthy comment that exceeds the limit." • Cropped: "This is a very lengthy comment."
Batch Normalization	• Normalize word frequencies in text data (e.g., TF-IDF). • Example: • Comment: "This is toxic toxic toxic." • After Normalization: ["toxic": 3/6, "this": 1/6, "is": 1/6].
Data Preprocessing Code Screenshots	

Loading Data	
	`#Loading Dataset`
	`train_df = pd.read_csv('/content/train.csv')`
	`test_df = pd.read_csv('/content/test.csv')`

Data Description & Null values

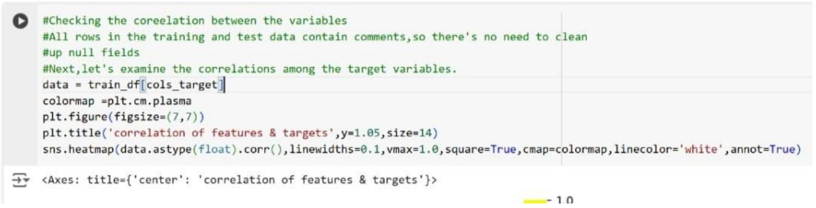


```
[ ] #DATA DESCRIPTION
    cols_target =['insult','toxic','severe_toxic','identity_hate','threat','obscene']
```

```
#check for null comments in test_df
print(test_df.isnull().any())
```

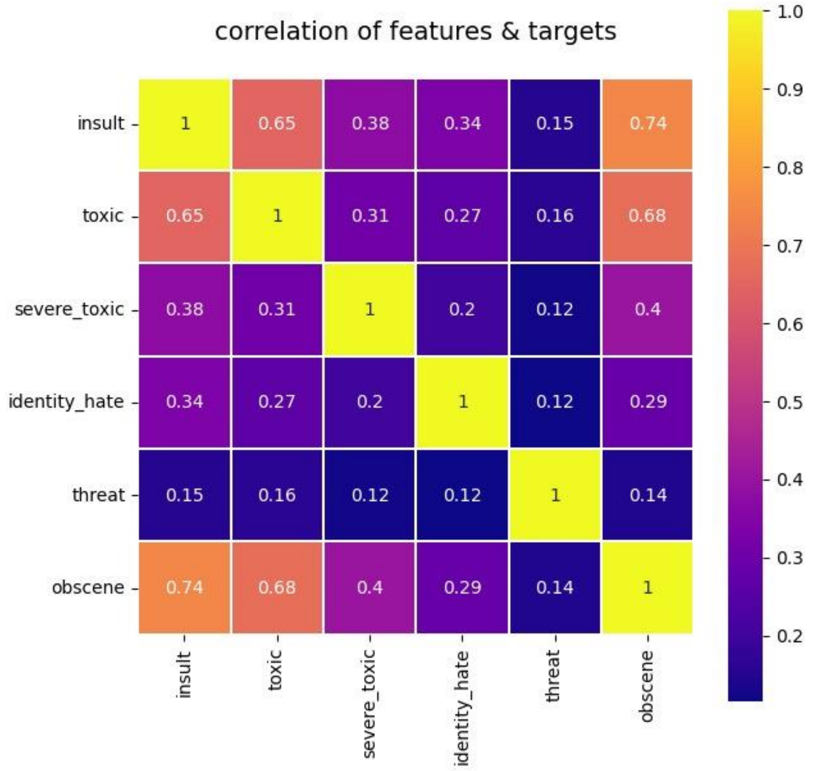
```
id              False
comment_text    False
dtype: bool
id              False
comment_text    False
dtype: bool
```

Coreleation between variables



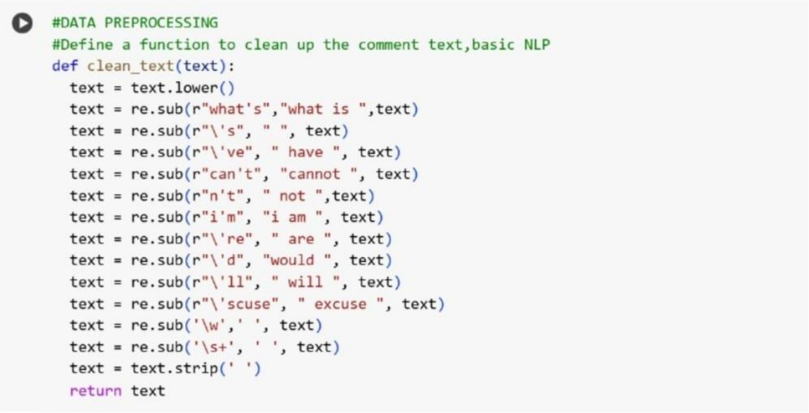
```
#Checking the coreelation between the variables
#All rows in the training and test data contain comments,so there's no need to clean
#up null fields
#Next,let's examine the correlations among the target variables.
data = train_df[cols_target]
colormap =plt.cm.plasma
plt.figure(figsize=(7,7))
plt.title('correlation of features & targets',y=1.05,size=14)
sns.heatmap(data.astype(float).corr(),linewidths=0.1,vmax=1.0,square=True,cmap=colormap,linecolor='white',annot=True)
```

```
<Axes: title={'center': 'correlation of features & targets'}>
```



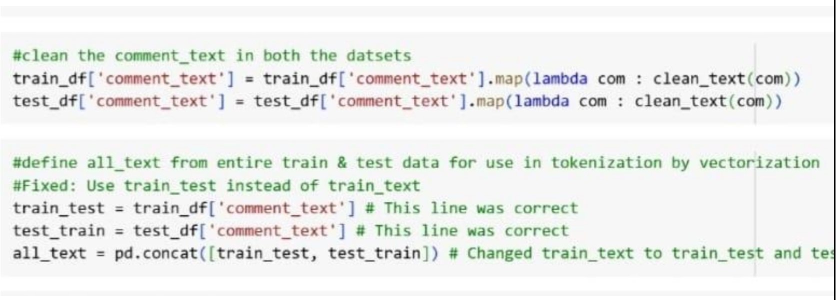
	insult	toxic	severe_toxic	identity_hate	threat	obscene
insult	1	0.65	0.38	0.34	0.15	0.74
toxic	0.65	1	0.31	0.27	0.16	0.68
severe_toxic	0.38	0.31	1	0.2	0.12	0.4
identity_hate	0.34	0.27	0.2	1	0.12	0.29
threat	0.15	0.16	0.12	0.12	1	0.14
obscene	0.74	0.68	0.4	0.29	0.14	1

<table>
<tr>
<td>Data Preprocessing</td>
<td>



```
#DATA PREPROCESSING
#Define a function to clean up the comment text,basic NLP
def clean_text(text):
  text = text.lower()
  text = re.sub(r"what's","what is ",text)
  text = re.sub(r"\'s", " ", text)
  text = re.sub(r"\'ve", " have ", text)
  text = re.sub(r"can't", "cannot ", text)
  text = re.sub(r"n't", " not ",text)
  text = re.sub(r"i'm", "i am ", text)
  text = re.sub(r"\'re", " are ", text)
  text = re.sub(r"\'d", "would ", text)
  text = re.sub(r"\'ll", " will ", text)
  text = re.sub(r"\'scuse", " excuse ", text)
  text = re.sub('\W',' ', text)
  text = re.sub('\s+', ' ', text)
  text = text.strip(' ')
  return text
```

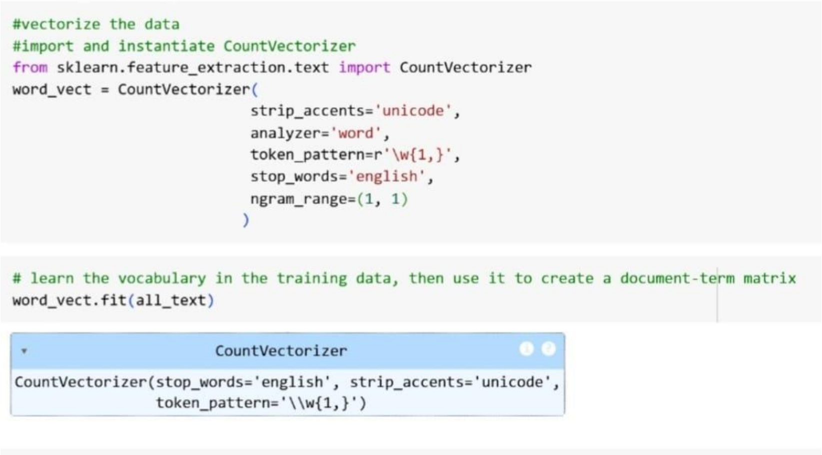
</td>
</tr>
<tr>
<td>clean the comment _text in both the datasets. & training and testing</td>
<td>



```
#clean the comment_text in both the datsets
train_df['comment_text'] = train_df['comment_text'].map(lambda com : clean_text(com))
test_df['comment_text'] = test_df['comment_text'].map(lambda com : clean_text(com))
```

```
#define all_text from entire train & test data for use in tokenization by vectorization
#Fixed: Use train_test instead of train_text
train_test = train_df['comment_text'] # This line was correct
test_train = test_df['comment_text'] # This line was correct
all_text = pd.concat([train_test, test_train]) # Changed train_text to train_test and tes
```

</td>
</tr>
<tr>
<td>Vectorize the data</td>
<td>

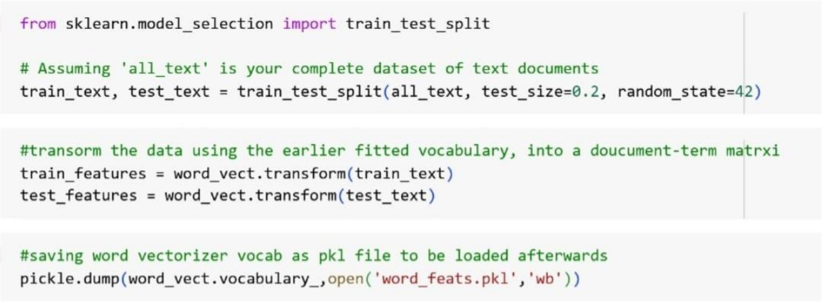


```
#vectorize the data
#import and instantiate CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
word_vect = CountVectorizer(
                           strip_accents='unicode',
                           analyzer='word',
                           token_pattern=r'\w{1,}',
                           stop_words='english',
                           ngram_range=(1, 1)
                          )
```

```
# learn the vocabulary in the training data, then use it to create a document-term matrix
word_vect.fit(all_text)
```

```
CountVectorizer
CountVectorizer(stop_words='english', strip_accents='unicode',
                token_pattern='\\w{1,}')
```

</td>
</tr>
<tr>
<td>Train _test_ split& Transform the data& saving word vectorizer</td>
<td>



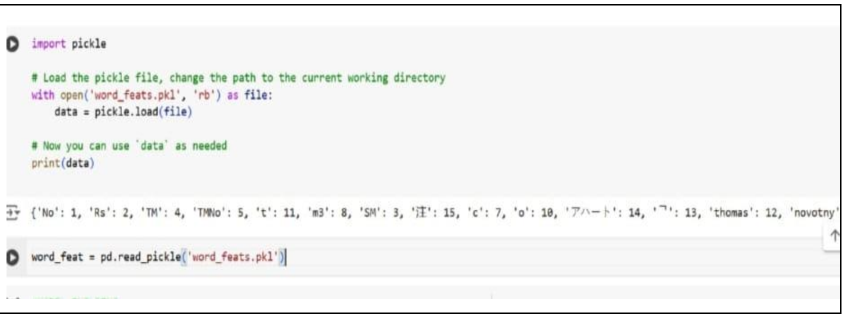
```
from sklearn.model_selection import train_test_split

# Assuming 'all_text' is your complete dataset of text documents
train_text, test_text = train_test_split(all_text, test_size=0.2, random_state=42)
```

```
#transorm the data using the earlier fitted vocabulary, into a doucument-term matrxi
train_features = word_vect.transform(train_text)
test_features = word_vect.transform(test_text)
```

```
#saving word vectorizer vocab as pkl file to be loaded afterwards
pickle.dump(word_vect.vocabulary_,open('word_feats.pkl','wb'))
```

</td>
</tr>
</table>

Loading the pickle file	
	import pickle
	
	# Load the pickle file, change the path to the current working directory
	with open('word_feats.pkl', 'rb') as file:
	    data = pickle.load(file)
	
	# Now you can use `data` as needed
	print(data)
	
	{'No': 1, 'Rs': 2, 'TM': 4, 'TMNo': 5, 't': 11, 'm3': 8, 'SM': 3, '注': 15, 'c': 7, 'o': 10, 'アハート': 14, '¬': 13, 'thomas': 12, 'novotny'
	
	word_feat = pd.read_pickle('word_feats.pkl')