Data Collection and Preprocessing Phase

Date	20 October 2024
Team ID	739734
Project Title	Ai-powered vehicle damage assessment and cost estimation for insurance claims
Maximum Marks	6 Marks

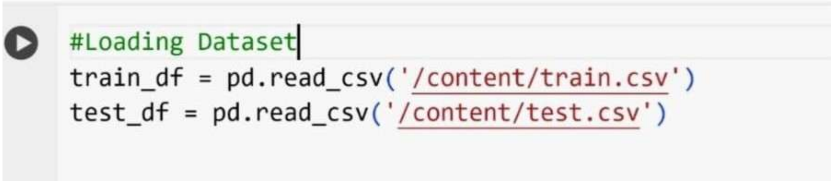
Preprocessing Template

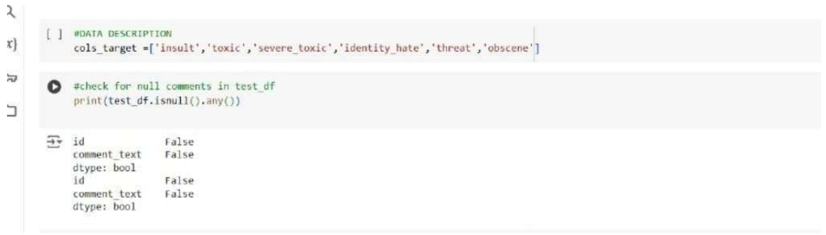
The preprocessing stage involves collecting and processing vehicle damage images and corresponding claims data. This includes data cleaning, image resizing and normalization, and annotation of damage locations and types. Additionally, data augmentation techniques will be applied to increase dataset diversity and reduce overfitting.

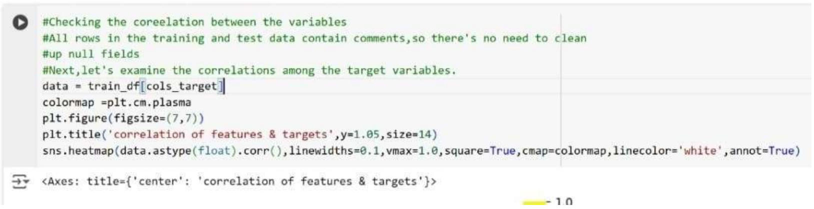
The preprocessed data will be split into training, validation, and testing sets to support the development and evaluation of the AI-powered damage assessment and cost estimation model.

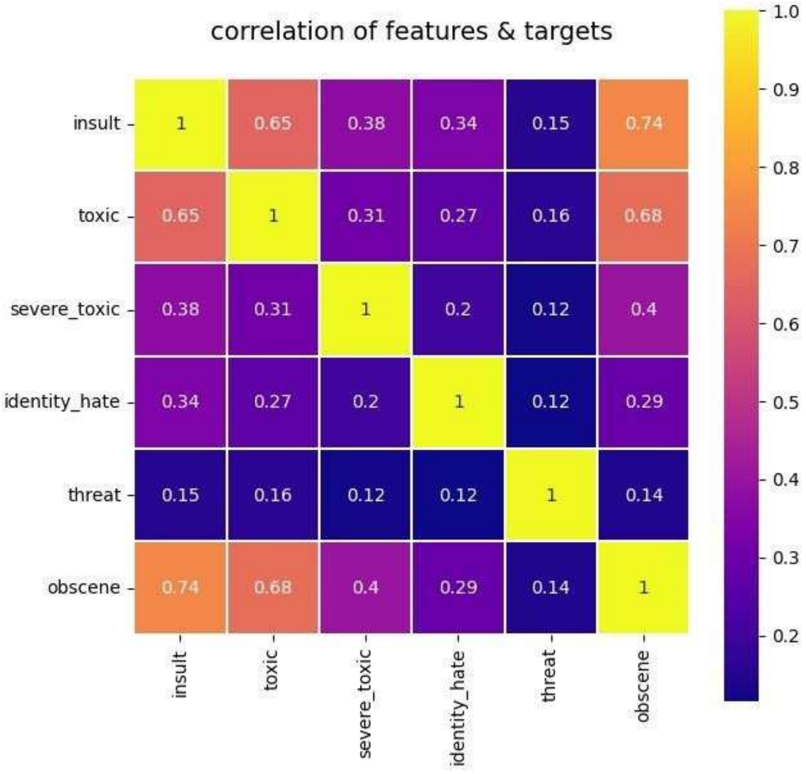
Section	**Description**
Data Overview	The data for AI-powered vehicle damage assessment and cost estimation consists of approximately 100,000+ vehicle damage images and 500,000+ corresponding claims records, sourced from insurance company records and public databases.
Resizing	Resizing images to a uniform size (e.g., 512x512 pixels) to facilitate model training and improve efficiency. - Maintaining aspect ratio to preserve image integrity and prevent distortion. - Using interpolation techniques (e.g., bilinear, bicubic) to minimize image degradation during resizing.
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	

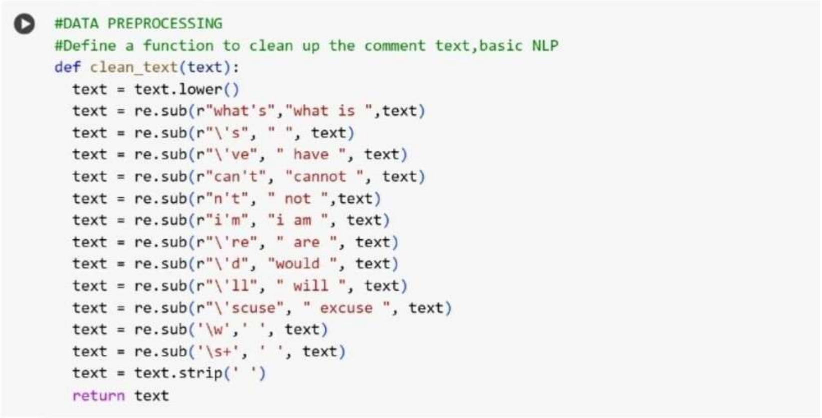
Loading Data	 `#Loading Dataset`
	`train_df = pd.read_csv('/content/train.csv')`
	`test_df = pd.read_csv('/content/test.csv')`

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'}>`

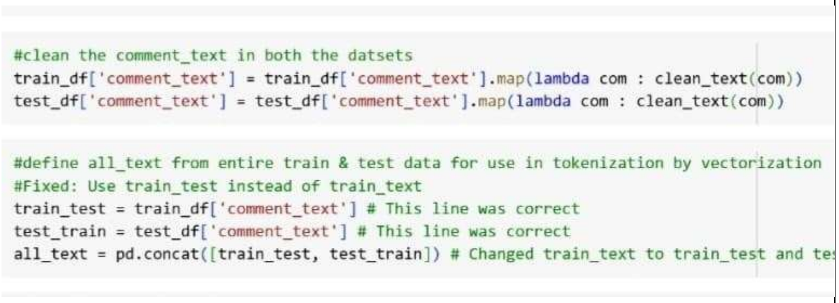
	

Data Preprocessing



```
#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
```

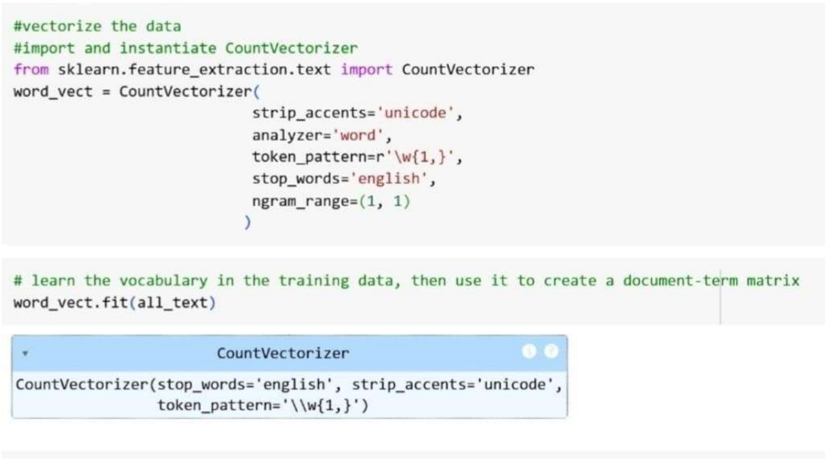
clean the comment _text in both the datasets. & training and testing



```
#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
```

Vectorize the data

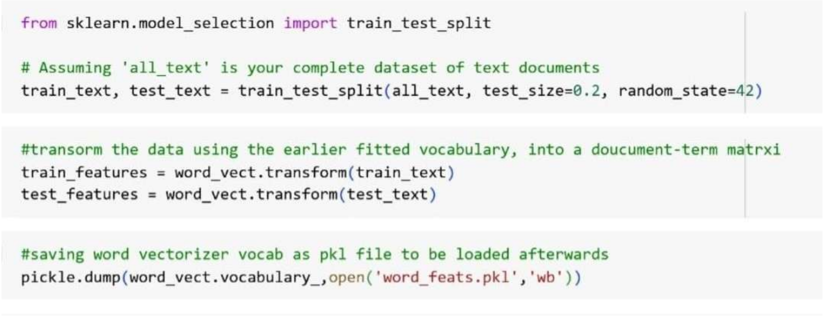


```
#vectorize the data
#import and instantiate CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
word_vect = CountVectorizer(
                            strip_accents='unicode',
                            analyzer='word',
                            token_pattern=r'\w{1,}',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

Train _test_ split& Transform the data& saving word vectorizer

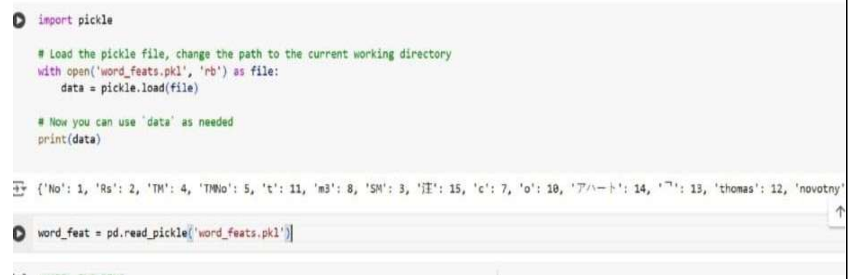


```
from sklearn.model_selection import train_test_split

# 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'))
```

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, 'アパート': 14, '¬': 13, 'thomas': 12, 'novotny'
```

```
word_feat = pd.read_pickle('word_feats.pkl')
```