# NLP TD-IDF+Decision Tree

## Importing Libraries

Firstly started Importing all necessary libraries. Then read the csv files in python notebook with the help of pandas libraries.

## EDA

* I applied word\_tokenize() and Counter() Functions. NLTK's word\_tokenize() function to tokenize the text data. the Counter() function counts the frequency of each word in the tokenized text.

The resulting word frequency counts are stored in a pandas DataFrame called words\_counting. This DataFrame has three columns: 'words', 'count', and 'word\_len'. The 'words' column contains the unique words in the corpus, the 'count' column contains the frequency count of each word, and the 'word\_len' column contains the length of each word.

* Then in the EDA part I have performed some Data Visualizations. I plotted a kernel density estimate (KDE) plot for the distribution of the length of the train DataFrame. The KDE plot is created using seaborn's kdeplot() function. It uses the plt.figure() function from matplotlib to create a new figure with a size of 10 inches by 5 inches. Then, it adds an empty subplot to the figure using fig.add\_subplot(), and assigns it to the variable ax.
* The 'x' parameter computes the length of each text sample in train dataset after splitting it into words.
* The 'ec' parameter is set to "#000", which sets the color of the edges of the filled area to black.
* The 'fill' parameter is set to True, which fills the area under the KDE curve. The 'alpha' parameter is set to 1, which sets the opacity of the filled area to maximum.
* The 'ax' parameter is set to ax, which specifies the subplot to which the KDE plot should be added.
* Then I have performed word cloud visualization using the WordCloud module from the Python library. It first creates a WordCloud object with the background\_color set to 'white', and the width and height set to 800 and 400, respectively. The generate() method is then called on the WordCloud object to generate the word cloud.The input to the generate() method is the string produced by concatenating all the strings in the 'full\_text' column of the train DataFrame using the join() method. This produces a single long string that contains all the text data. The imshow() function from matplotlib is then called to display the word cloud visualization.
* Then I plotted histogram of the distribution of the total number of words in each sample of text in the 'full\_text' column of the train DataFrame. The histogram is created using seaborn's histplot() function.

## Modelling

* I have performed **TD-IDF** for feature extraction and **Decision Tree** for modelling.
* After Importing necessary libraries and defining input variable and target variables I performed TD-IDF for feature extraction.
* It is seen that Decision tree mostly split on stop words. It's not reasonable to remove stop words because we are evaluating essays.
* To resolve this, I would use 2 feature extractors:
* TD-IDF extractor with ngrame\_range=(3,6),which can be helpful for grammar or cohesion classes.

**vectorizer\_ngram** is configured to extract n-gram features of length 3 to 6 from the input text data, with a maximum of 1000 features.

* TD-IDF extractor with stop words extraction, which helps introduce more information in the decision tree.

**vectorizer\_stop\_words** is configured to remove English stop words (common words that typically carry little meaning, such as "the", "and", and "of") from the input text data, with a maximum document frequency of 0.7 (i.e., words that appear in more than 70% of the documents will be removed) and a maximum of 500 features.

* The **FeatureUnion** class is then used to combine the two instances into a single feature extractor, **feature\_extractor**, which applies both sets of feature extraction methods to the input text data.
* The **fit\_transform()** method of the feature extractor is called on the input data x. Then I fitted the **DecisionTreeRegressor()** to the trained and transformed varible.
* The **transform()** method of the feature extractor is called on the x\_test.
* Finally predicted the y\_test values for the given x\_test data with decision tree regressor.