# NLP DATASET

In [3]:
```
import pandas as pd

# Load the dataset to examine its contents
file_path = 'C:\\Users\\Admin\\Downloads\\nlp_dataset.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
df
```

Out[3]:

	Comment	Emotion
0	i seriously hate one subject to death but now ...	fear
1	im so full of life i feel appalled	anger
2	i sit here to write i start to dig out my feel...	fear
3	ive been really angry with r and i feel like a...	joy
4	i feel suspicious if there is no one outside l...	fear
...	...	...
5932	i begun to feel distressed for you	fear
5933	i left feeling annoyed and angry thinking that...	anger
5934	i were to ever get married i d have everything...	joy
5935	i feel reluctant in applying there because i w...	fear
5936	i just wanted to apologize to you because i fe...	anger

5937 rows × 2 columns

# 1. Loading and Preprocessing :

```
Load the dataset and perform necessary preprocessing steps. This should include text cleaning, tokenization, and removal of
stopwords. Explain the preprocessing techniques used and their impact on model performance.
```

In [5]:
```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download necessary NLTK resources (stopwords, tokenizer)
nltk.download('punkt')
nltk.download('stopwords')

# Define stopwords in English
stop_words = set(stopwords.words('english'))

# Text cleaning function
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove special characters, numbers, and extra spaces
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords
    filtered_tokens = [word for word in tokens if word not in stop_words]

    # Join tokens back to form the cleaned sentence
    return ' '.join(filtered_tokens)

# Apply preprocessing to the 'Comment' column
df['cleaned_comment'] = df['Comment'].apply(preprocess_text)

# Display the first few rows to verify the preprocessing
df[['Comment', 'cleaned_comment', 'Emotion']]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[5]:

	Comment	cleaned_comment	Emotion
0	i seriously hate one subject to death but now ...	seriously hate one subject death feel reluctan...	fear
1	im so full of life i feel appalled	im full life feel appalled	anger
2	i sit here to write i start to dig out my feel...	sit write start dig feelings think afraid acce...	fear
3	ive been really angry with r and i feel like a...	ive really angry r feel like idiot trusting fi...	joy
4	i feel suspicious if there is no one outside l...	feel suspicious one outside like rapture happe...	fear
...	...	...	...
5932	i begun to feel distressed for you	begun feel distressed	fear
5933	i left feeling annoyed and angry thinking that...	left feeling annoyed angry thinking center stu...	anger
5934	i were to ever get married i d have everything...	ever get married everything ready offer got to...	joy
5935	i feel reluctant in applying there because i w...	feel reluctant applying want able find company...	fear
5936	i just wanted to apologize to you because i fe...	wanted apologize feel like heartless bitch	anger

5937 rows × 3 columns

```
The text preprocessing steps have been applied, resulting in a cleaned version of the comments. Here’s a summary of the
preprocessing techniques used:

Lowercasing: This helps treat words like "Feel" and "feel" as the same, improving consistency.
Removing Special Characters and Numbers: Helps focus on the core text, eliminating noise such as punctuation and numbers
that don't contribute to emotion detection.

Tokenization and Stopword Removal: Breaking text into words and removing common stopwords (e.g., "the", "and"). This
reduces the volume of irrelevant words and focuses the model on meaningful terms.

These steps can improve model performance by reducing noise and ensuring that only important information is retained. Next,
we can proceed to vectorization and building machine learning models for emotion classification.
```

# 2. Feature Extraction :

Implement feature extraction using CountVectorizer or TfidfVectorizer. Describe how the chosen method transforms the text data into numerical features.

In [6]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)  # Limit to top 5000 features for efficiency

# Fit and transform the cleaned comments
X = tfidf_vectorizer.fit_transform(df['cleaned_comment'])

# Display the shape of the resulting feature matrix
X.shape  # (number of documents, number of features)
```

Out[6]:
```
(5937, 5000)
```

# How TfidfVectorizer Transforms Text:

In [ ]:
```
1 . Term Frequency (TF): This counts how frequently a term appears in a document, similar to CountVectorizer.

            TF(t,d)= Number of occurences of term t in document d / Total terms in document d

2 . Inverse Document Frequency (IDF): This measures how unique a term is across all documents. Common words across many docu

           IDF(t) = log(Total Number of documents / Number of documents containing term t)

3. TF-IDF Value: The TF-IDF score for each term in a document is computed as:

           TF-IDF(t,d) = TF(t,d) * IDF(t)

           Words that are frequent in a document but rare across the corpus are assigned higher scores, which helps the mode

This method converts text into a numerical matrix where each row represents a document, and each column corresponds to a wor
```

# 3. Model Development :

(a) Naive Bayes
(b) Support Vector Machine

# (a) Naive Bayes

In [7]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, df['Emotion'], test_size=0.2, random_state=42)

# Initialize and train the Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)

# Evaluate the model
y_pred_nb = nb_model.predict(X_test)
print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

       anger       0.88      0.95      0.91       392
        fear       0.91      0.92      0.92       416
         joy       0.94      0.86      0.90       380

    accuracy                           0.91      1188
   macro avg       0.91      0.91      0.91      1188
weighted avg       0.91      0.91      0.91      1188
```

# (b) Support Vector Machine

In [9]:
```
from sklearn.svm import LinearSVC

# Initialize and train the SVM model
svm_model = LinearSVC()
svm_model.fit(X_train, y_train)

# Evaluate the model
y_pred_svm = svm_model.predict(X_test)
print(classification_report(y_test, y_pred_svm))
```

```
              precision    recall  f1-score   support

       anger       0.94      0.95      0.95       392
        fear       0.97      0.93      0.95       416
         joy       0.95      0.97      0.96       380

    accuracy                           0.95      1188
   macro avg       0.95      0.95      0.95      1188
weighted avg       0.95      0.95      0.95      1188


C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will chan
ge from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
```

# 4. Model Comparison

Evaluate the model using appropriate metrics (e.g., accuracy, F1-score). Provide a brief explanation of the chosen model and its suitability for emotion classification.

In [10]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Evaluate Naive Bayes
nb_accuracy = accuracy_score(y_test, y_pred_nb)
nb_f1 = f1_score(y_test, y_pred_nb, average='weighted')
nb_precision = precision_score(y_test, y_pred_nb, average='weighted')
nb_recall = recall_score(y_test, y_pred_nb, average='weighted')

# Evaluate SVM
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_f1 = f1_score(y_test, y_pred_svm, average='weighted')
svm_precision = precision_score(y_test, y_pred_svm, average='weighted')
svm_recall = recall_score(y_test, y_pred_svm, average='weighted')

# Print results
print(f"Naive Bayes - Accuracy: {nb_accuracy}, F1-Score: {nb_f1}, Precision: {nb_precision}, Recall: {nb_recall}")
print(f"SVM - Accuracy: {svm_accuracy}, F1-Score: {svm_f1}, Precision: {svm_precision}, Recall: {svm_recall}")
```

```
Naive Bayes - Accuracy: 0.9107744107744108, F1-Score: 0.9106126533471106, Precision: 0.9123424719074418, Recall: 0.91077441
07744108
SVM - Accuracy: 0.952020202020202, F1-Score: 0.9519834431548418, Precision: 0.9522441072672482, Recall: 0.952020202020202
```

# Brief Explanation of Model Suitability:

# Naive Bayes:

In [ ]:
```
Naive Bayes is highly suitable for text classification when the data is large and the feature space (words) is sparse, as it
```

# Support Vector Machine (SVM):

In [ ]:
```
SVM tends to perform better for emotion classification, especially when the emotions are linearly separable in high-dimensio
```

# Conclusion:

In [ ]:
```
Naive Bayes is fast and provides a good baseline but might oversimplify text features.
SVM generally achieves higher performance in emotion classification by leveraging its ability to model complex boundaries be
```