1. The dataset contains two columns namely, “Comment” and “Emotion”. We have included preprocessing techniques including text cleaning, tokenization and stop word removal.

- Lowercasing: This ensures the model that, the words are treated the same without making it case sensitive.
- Punctuation removal: Removing the unnecessary punctuations simplifies the text.
- Tokenization: Breaks down the text into separate words. This is essential for feeding the text into models.
- Stop word Removal: Removing the words like ‘is’, ‘I’ which doesn’t have significant role. So that the model can focus on the job.

2. For feature extraction, I choose to implement TF-IDF (Term Frequency-Inverse Document Frequency) using TfidfVectorizer. The TF-IDF vectorizer has successfully transformed the text data into a numerical matrix where each row represents a comment, and each column represents a unique word. The values are the TF-IDF scores, indicates the relative importance of each word. The matrix contains many zeros, indicating that most words appear only in a subset of the documents. TF-IDF scores are given for the words that occur frequently in a specific comment but not in others are given higher scores, helping the model focus on distinguishing words for emotion classification.

Model Suitability:

1. Naive Bayes**:** This algorithm is suitable for text classification because it assumes the features (words) are conditionally independent given the class label. Despite this strong assumption, Naive Bayes often performs surprisingly well on text data and is computationally efficient.

   Pros**:** Works well for high-dimensional data (e.g., TF-IDF), simple to implement, and fast.

   Cons**:** The assumption of independence between features is rarely true in real-world text.

2. Support Vector Machine (SVM)**:** SVMs are robust classifiers that perform well with high-dimensional data like text. The linear kernel works effectively in cases where the classes are linearly separable in the feature space (e.g., with TF-IDF).

   Pros**:** High accuracy, especially when classes are well-separated in feature space. It is effective in high-dimensional spaces and with a clear margin of separation.

   Cons**:** Computationally intensive for large datasets, and tuning hyperparameters like the kernel and regularization is important for good performance.