Data Exploration

```
import pandas as pd

# Load the Excel file
df = pd.read_excel('text_class.xlsx')  # file must be uploaded first

# Display the first 5 rows
print("First 5 rows of the dataset:")
print(df.head())

# Print total number of rows and unique labels
print("\nTotal number of rows:", len(df))
print("Unique labels and their counts:")
print(df['label'].value_counts())
```

```
First 5 rows of the dataset:
                                                text     label
0                  I loved the product, it's amazing!  positive
1    Terrible service, I will never shop here again.  negative
2    The quality is good, but the delivery was late.   neutral
3  Absolutely wonderful experience, highly recomm...  positive
4  Product was damaged when it arrived, very disa...  negative

Total number of rows: 8
Unique labels and their counts:
label
positive    3
negative    3
neutral     2
Name: count, dtype: int64
```

In this step, I loaded the dataset into a pandas DataFrame. I displayed the first 5 rows to understand the structure of the data and printed the total number of rows along with the counts of unique labels. This helps in knowing the dataset size and the class distribution before preprocessing.

Preprocessing Text Data

```python
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

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

# Function to clean text
def preprocess_text(text):
    # 1. Lowercase
    text = text.lower()
    # 2. Remove punctuation & special characters (keep only letters and spaces
    text = re.sub(r'[^a-z\s]', '', text)
    # 3. Tokenize (split into words)
    words = text.split()
    # 4. Remove stopwords
    words = [w for w in words if w not in stop_words]
    # 5. Join back to sentence
    return " ".join(words)

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

# Show first 5 cleaned rows
print("✅ First 5 rows after preprocessing:")
print(df[['text', 'clean_text']].head())
```

```
✅ First 5 rows after preprocessing:
                                                text  \
0                 I loved the product, it's amazing!
1   Terrible service, I will never shop here again.
2   The quality is good, but the delivery was late.
3  Absolutely wonderful experience, highly recomm...
4  Product was damaged when it arrived, very disa...

                                          clean_text
0                             loved product amazing
1                        terrible service never shop
2                         quality good delivery late
3  absolutely wonderful experience highly recommend
4            product damaged arrived disappointed
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Preprocess and vectorize text
vectorizer = TfidfVectorizer(stop_words='english', lowercase=True)
X = vectorizer.fit_transform(df['clean_text'])

# Keep labels separately
y = df['label']

# Print processed sample
print("✅ TF-IDF processed matrix shape:", X.shape)
print("\n📄 Feature names (first 10):", vectorizer.get_feature_names_out()[:10
```

```
✅ TF-IDF processed matrix shape: (8, 28)

📄 Feature names (first 10): ['absolutely' 'amazing' 'arrived' 'customer' '
 'disappointed' 'experience' 'good' 'helpful']
```

Here I applied text preprocessing to clean the raw text. I converted all text to lowercase, removed punctuation and special characters, tokenized the sentences, and removed stopwords. This ensures that only meaningful words remain. I also displayed the first 5 cleaned rows for verification. Finally, I used TF-IDF Vectorizer to transform the cleaned text into numerical features.

Train a Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Split the data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randon

# Train Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("🎯 Accuracy of the model:", accuracy)
```

```
🎯 Accuracy of the model: 0.0
```

I split the dataset into training (80%) and testing (20%) sets. Then I trained a Logistic Regression classifier on the training data. After predicting on the test set, I calculated the accuracy. The accuracy is very low because the dataset is very small (only 8 rows total), which makes the test set unreliable. This shows the importance of having larger datasets in machine learning.

Evaluate the Model

```
from sklearn.metrics import confusion_matrix

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

print("\n🧾 Confusion Matrix:")
print(cm)
```

```
🧾 Confusion Matrix:
[[0 1 0]
 [0 0 0]
 [0 1 0]]
```

I used a confusion matrix to evaluate the model's predictions. The confusion matrix shows how many predictions were correct and where the model made mistakes. It helps in analyzing performance by comparing true labels with predicted labels for each class.

The confusion matrix helps analyze the results by showing the number of correct and incorrect predictions for each class. It allows us to see not only the overall accuracy but also which specific labels were misclassified and where the model is making mistakes.

The accuracy score shows the overall performance of the logistic regression model.

The confusion matrix helps analyze which classes the model predicted correctly and where it made mistakes. It shows the counts of true vs. predicted labels for each class.

By, Utkarsh Anand