Task 1: Data Exploration (2 Marks)

1. Load the sample dataset provided in text_class.csv

```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving text_class.csv to text_class.csv
```

1. Display the first 5 rows of the dataset.

```
import pandas as pd
df = pd.read_csv('text_class.csv')

print("First 5 rows of the dataset:")
print(df.head())
```

```
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
```

1. Print the total number of rows and the count of unique labels in the dataset

```
print("\nTotal number of rows:", len(df))
print("Number of unique labels:", df['label'].nunique())
print("Unique labels:", df['label'].unique())
```

```
Total number of rows: 8
Number of unique labels: 3
Unique labels: ['positive' 'negative' 'neutral']
```

1. Complete any necessary step which is required before preprocessing

```
print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
text     0
label    0
dtype: int64
```

```
print("\nDataset Info:")
print(df.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    8 non-null      object
 1   label   8 non-null      object
dtypes: object(2)
memory usage: 260.0+ bytes
None
```

Task 2: Preprocessing Text Data (3 Marks)

1. Convert all text to lowercase.

```
# Step 1: Import necessary libraries
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')

from nltk.tokenize import word_tokenize

df['text_lower'] = df['text'].str.lower()
print("Step 1: Lowercased text")
print(df['text_lower'].head())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.

Step 1: Lowercased 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...
Name: text_lower, dtype: object

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

1. Remove punctuation and special characters

```
# Step 2: Remove punctuation and special characters
df['text_clean'] = df['text_lower'].apply(lambda x: re.sub(r'[^a-zA-
Z0-9\s]', '', x))
print("\nStep 2: Cleaned text (no punctuation/special characters)")
print(df['text_clean'].head())
```

```
Step 2: Cleaned text (no punctuation/special characters)
0                       i loved the product its 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 recommend
4      product was damaged when it arrived very disap...
Name: text_clean, dtype: object
```

1. Tokenize the text and remove stopwords. Provide the processed version of the first 5 rows.

```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Download required NLTK resources
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')


# Step 3: Tokenize and remove stopwords
stop_words = set(stopwords.words('english'))
df['text_tokens'] = df['text_clean'].apply(word_tokenize)
df['text_tokens_nostop'] = df['text_tokens'].apply(lambda x: [word for
word in x if word not in stop_words])

print("\nStep 3: Tokenized text without stopwords")
print(df[['text', 'text_tokens_nostop']].head())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.


Step 3: Tokenized text without stopwords
                                                  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...

                                  text_tokens_nostop
0                         [loved, product, amazing]
1                  [terrible, service, never, shop]
2                   [quality, good, delivery, late]
```

```
3  [absolutely, wonderful, experience, highly, re...
4          [product, damaged, arrived, disappointed]

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Task 3: Train a Classifier (3 Marks)

1. Split the data into training and test sets (80% training, 20% testing).
2. Train a simple logistic regression model.
3. Predict the labels on the test set and calculate accuracy. Provide the accuracy score and a brief comment on the result.

```
#Let's do Split, Train, and Predit step-by-step.

# Step 1: Import required libraries
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Step 2: Join tokens back into strings for TF-IDF vectorization
df['clean_text'] = df['text_tokens_nostop'].apply(lambda x: '
'.join(x))
print(df['clean_text'])

# Step 3: Convert text to TF-IDF features
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['clean_text'])

# Step 4: Extract labels
y = df['label']

# Step 5: Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Train Logistic Regression model
LGR_model = LogisticRegression()
LGR_model.fit(X_train, y_train)

# Step 7: Make predictions
y_pred = LGR_model.predict(X_test)

# Step 8: Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

```
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
5                      customer support helpful polite
6                         worst purchase ive ever made
7                         satisfied product price high
Name: clean_text, dtype: object
Accuracy Score: 0.5
```

Task 4: Evaluate the Model (2 Marks)

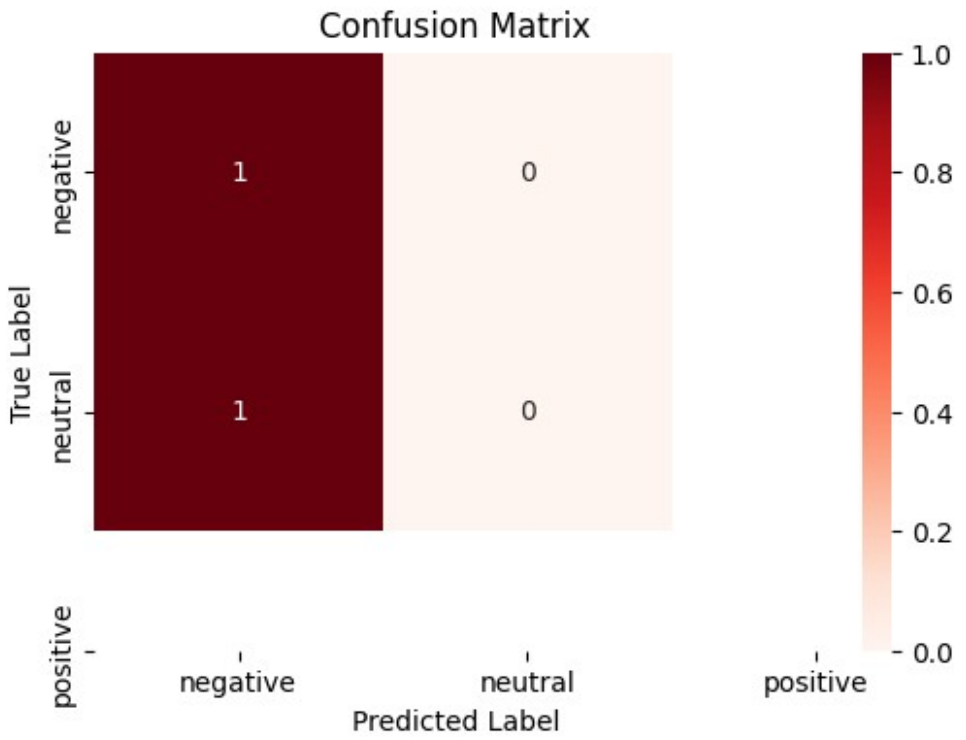
1. Evaluate the performance of the model.

```
'''
The confusion matrix helps us understand where the model is making
errors—for instance,
if it's confusing one label for another. Combined with the F1-score,
it gives a clearer view of model performance beyond just accuracy.
'''

from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds',
xticklabels=LGR_model.classes_, yticklabels=LGR_model.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Step 2: Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```




```
Classification Report:
              precision    recall  f1-score   support

    negative       0.50      1.00      0.67         1
    positive       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2


/usr/local/lib/python3.11/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
```

```
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

1. Write one or two sentences on how the confusion matrix helps analyze the results.

Ans) The confusion matrix helps us understand where the model is making errors—for instance, if it's confusing one label for another.

Combined with the F1-score, it gives a clearer view of model performance beyond just accuracy.

The confusion matrix helps analyze classification results by showing the number of correct and incorrect predictions for each class.

It highlights specific misclassifications, making it easier to identify patterns of errors and assess model performance beyond overall accuracy.

-----Thank You------