Task 2: Sentiment Analysis with Natural Language Processing

```
# Install
!pip install -q spacy
!python -m spacy download en_core_web_sm
```

Show hidden output

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import spacy
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 classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import (
accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report
)
# Load spaCy English model
nlp = spacy.load("en_core_web_sm")
```

```
# Upload the file if running in Colab
from google.colab import files
uploaded = files.upload()
# Load the dataset
df = pd.read_csv('reviews.csv')
print(df.head())
```

Choose Files reviews.csv

- **reviews.csv**(text/csv) - 593 bytes, last modified: 7/19/2025 - 100% done

```
Saving reviews.csv to reviews (5).csv
                                     Review  Sentiment
0   Great product, really enjoyed using it!          1
1  Terrible experience. Will not buy again.          0
2          Very satisfied with the service.          1
3             Bad quality, broke in a week.          0
4      Excellent quality and fast shipping!          1
```

```
# Clean column names (remove spaces)
df.columns = df.columns.str.strip()
# Check available columns
print("Available columns:", df.columns.tolist())
```

```
Available columns: ['Review', 'Sentiment']
```

```
# Automatically identify 'review' and 'sentiment' columns
review_col = None
sentiment_col = None
for col in df.columns:
    if 'review' in col.lower():
        review_col = col
    if 'sentiment' in col.lower():
        sentiment_col = col
print(f"Review column: {review_col}")
print(f"Sentiment column: {sentiment_col}")
```

```
Review column: Review
Sentiment column: Sentiment
```

```
# Function to clean and lemmatize text
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc if not token.is_stop and token.is_alpha]
    return ' '.join(tokens)
# Apply preprocessing
df = df[[review_col, sentiment_col]].dropna().reset_index(drop=True) # Add reset_index(drop=True) here
df['Cleaned_Review'] = df[review_col].astype(str).apply(preprocess)
```

```
# Preview cleaned data
df[[review_col, 'Cleaned_Review', sentiment_col]].head()
```

	Review	Cleaned_Review	Sentiment
0	Great product, really enjoyed using it!	great product enjoy	1
1	Terrible experience. Will not buy again.	terrible experience buy	0
2	Very satisfied with the service.	satisfied service	1
3	Bad quality, broke in a week.	bad quality break week	0
4	Excellent quality and fast shipping!	excellent quality fast shipping	1

```
# Drop rows with missing values in Review or Sentiment
df.dropna(subset=['Review', 'Sentiment'], inplace=True)
# Now map sentiment to 0/1
df['Sentiment'] = df['Sentiment'].map({'positive': 1, 'negative': 0})
# Optional: drop rows where sentiment was neither positive nor negative
# df.dropna(subset=['Sentiment'], inplace=True) # Removed this line
```

```
# Initialize TfidfVectorizer
tfidf = TfidfVectorizer(max_features=5000) # You can adjust max_features as needed
```

```
# Convert text to TF-IDF vectors
if not df.empty and 'Cleaned_Review' in df.columns and not df['Cleaned_Review'].empty:
    X = tfidf.fit_transform(df['Cleaned_Review'])
    # Convert sentiment to binary
    y = df[sentiment_col]
    print("TF-IDF vectorization complete.")
else:
    print("DataFrame is empty or 'Cleaned_Review' column is missing/empty. Cannot perform TF-IDF vectorization.")
```

```
TF-IDF vectorization complete.
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

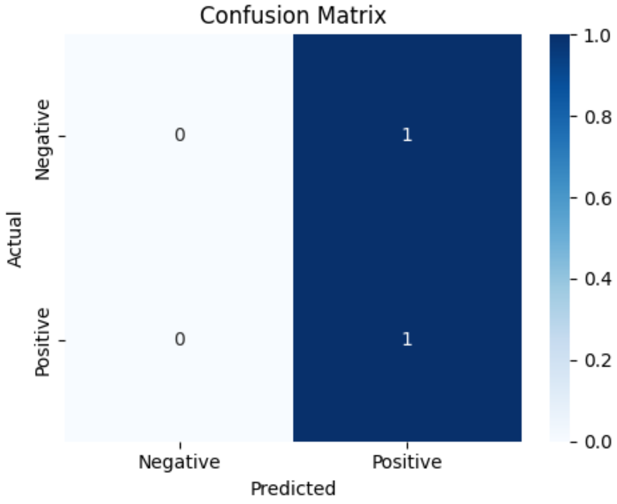
```
▾ LogisticRegression ⓘ ?
LogisticRegression()
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import numpy as np
import pandas as pd

# Step 1: Remove rows where y_test is NaN
mask = ~pd.isnull(y_test)
valid_indices = np.where(mask)[0]

# Step 2: Filter clean test sets
X_test_clean = X_test.iloc[valid_indices] if isinstance(X_test, pd.DataFrame) else X_test[valid_indices]
y_test_clean = y_test.iloc[valid_indices]

# Step 3: Predict
y_pred = model.predict(X_test_clean)

# Step 4: Evaluate
print("Model Evaluation Metrics:")

try:
    print("Accuracy:", accuracy_score(y_test_clean, y_pred))
    print("Precision:", precision_score(y_test_clean, y_pred, pos_label=1, zero_division=0))
    print("Recall:", recall_score(y_test_clean, y_pred, pos_label=1, zero_division=0))
    print("F1 Score:", f1_score(y_test_clean, y_pred, pos_label=1, zero_division=0))
except Exception as e:
    print("Error during metric computation:", e)

# Step 5: Detailed classification report
print("\nClassification Report:\n", classification_report(y_test_clean, y_pred, labels=[0, 1], zero_division=0))
```

```
Model Evaluation Metrics:
Accuracy: 0.5
Precision: 0.5
```

```
Recall: 1.0
F1 Score: 0.6666666666666666

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

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

```
# Predict on the clean test set
y_pred = model.predict(X_test_clean)
# Confusion matrix
cm = confusion_matrix(y_test_clean, y_pred)
# Plot
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()
```



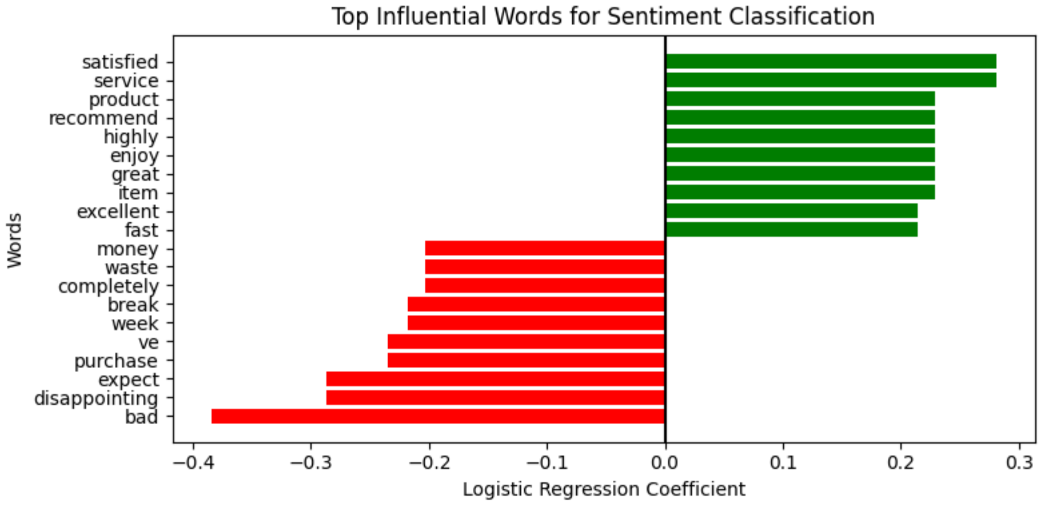
```
test_results = pd.DataFrame({
'Review Text': df.loc[y_test_clean.index][review_col], # Use y_test_clean.index and review_col
'Sentiment': y_test_clean.values, # Use y_test_clean.values
'Predicted': y_pred
})
# Correctly classified samples
print("Correctly classified positive reviews:")
print(test_results[(test_results['Sentiment'] == 1) & (test_results['Predicted'] == 1)].head(3)['Review Text'])
print("\nCorrectly classified negative reviews:")
print(test_results[(test_results['Sentiment'] == 0) & (test_results['Predicted'] == 0)].head(3)['Review Text'])
# Misclassified samples
print("\nIncorrectly classified as positive:")
print(test_results[(test_results['Sentiment'] == 0) & (test_results['Predicted'] == 1)].head(3)['Review Text'])
print("\nIncorrectly classified as negative:")
print(test_results[(test_results['Sentiment'] == 1) & (test_results['Predicted'] == 0)].head(3)['Review Text'])
```

```
Correctly classified positive reviews:
8    Good value for the price.
Name: Review Text, dtype: object

Correctly classified negative reviews:
Series([], Name: Review Text, dtype: object)

Incorrectly classified as positive:
1    Terrible experience. Will not buy again.
Name: Review Text, dtype: object

Incorrectly classified as negative:
Series([], Name: Review Text, dtype: object)
```

```
# Top positive and negative words
feature_names = np.array(tfidf.get_feature_names_out())
coefficients = model.coef_[0]
top_pos_idx = np.argsort(coefficients)[-10:]
top_neg_idx = np.argsort(coefficients)[:10]
print("Top Positive Sentiment Words:\n", feature_names[top_pos_idx])
print("\nTop Negative Sentiment Words:\n", feature_names[top_neg_idx])
```

```
Top Positive Sentiment Words:
 ['fast' 'excellent' 'item' 'great' 'enjoy' 'highly' 'recommend' 'product'
 'service' 'satisfied']

Top Negative Sentiment Words:
 ['bad' 'disappointing' 'expect' 'purchase' 've' 'week' 'break'
 'completely' 'waste' 'money']
```

```
# Extract top positive and negative word coefficients
feature_names = np.array(tfidf.get_feature_names_out())
coefficients = model.coef_[0]
# Sort features
top_n = 10
top_pos_idx = np.argsort(coefficients)[-top_n:]
top_neg_idx = np.argsort(coefficients)[:top_n]
top_features = np.concatenate([top_neg_idx, top_pos_idx])
top_words = feature_names[top_features]
top_coeffs = coefficients[top_features]
# Create bar plot
plt.figure(figsize=(8, 4))
colors = ['red'] * top_n + ['green'] * top_n
plt.barh(top_words, top_coeffs, color=colors)
plt.axvline(0, color='black')
plt.title("Top Influential Words for Sentiment Classification")
plt.xlabel("Logistic Regression Coefficient")
plt.ylabel("Words")
plt.tight_layout()
```


Top Influential Words for Sentiment Classification
satisfied
service
product
recommend
highly
enjoy
great
item
excellent
fast
money
waste
completely
break
week
ve
purchase
expect
disappointing
bad
Words
Logistic Regression Coefficient