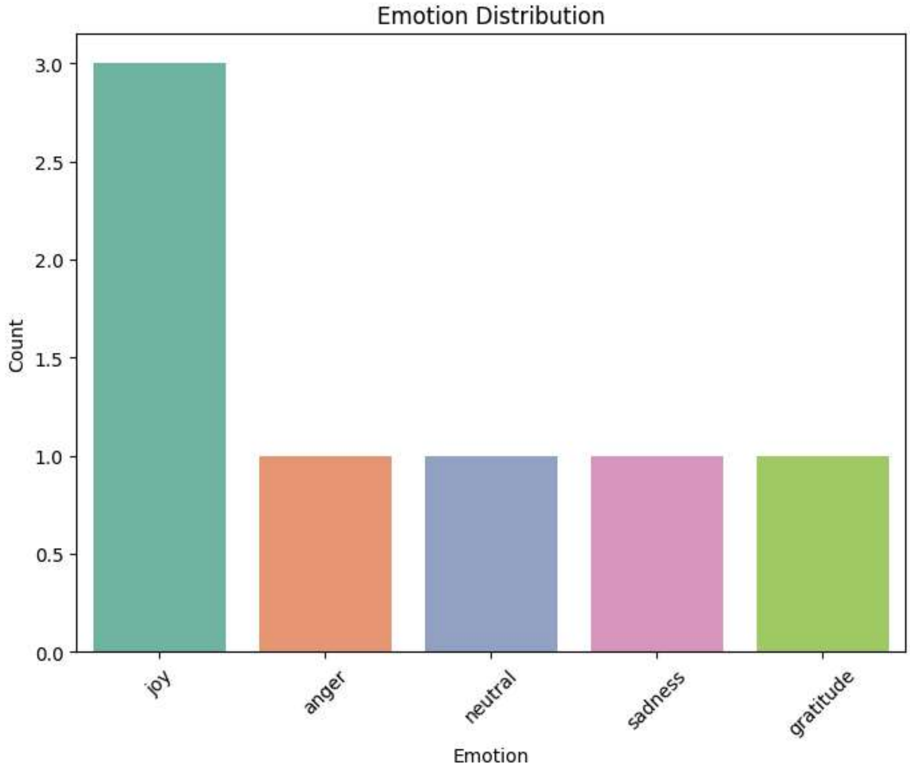
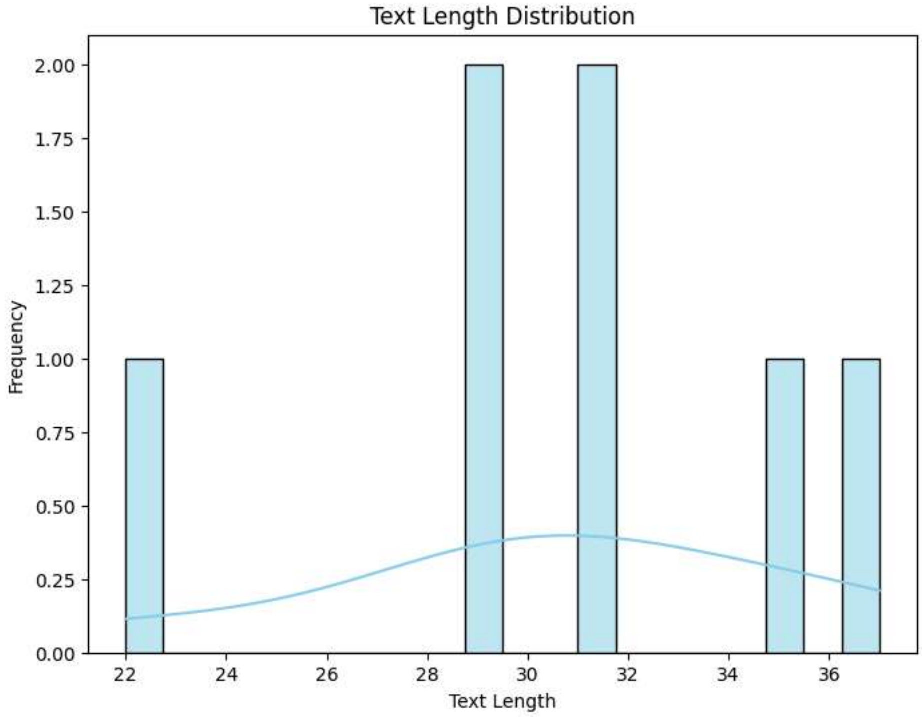
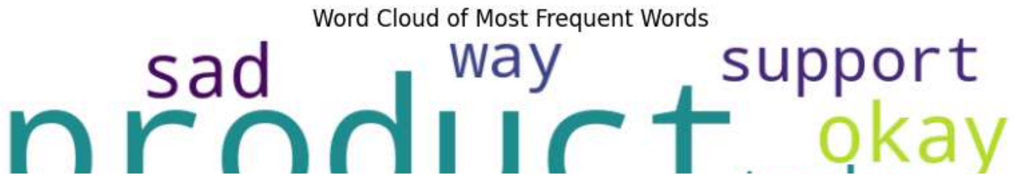
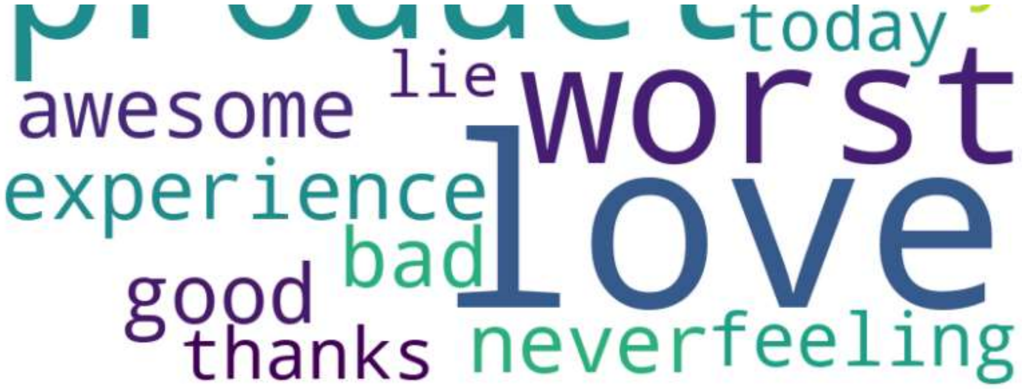
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from wordcloud import WordCloud

# Sample dataset
data = {
    'text': [
        'I love this product! 😊 #happy',
        'Worst experience ever. Never again!',
        np.nan,
        'Just okay... not good, not bad.',
        'This is awesome!!! http://example.com',
        'This is awesome!!! http://example.com',  # Duplicate
        'Feeling sad today... 😔',
        '@user Thanks for the support!',
        '',
        'Love the way you lie... #Eminem'
    ],
    'emotion': [
        'joy', 'anger', 'sadness', 'neutral', 'joy', 'joy', 'sadness', 'gratitude', np.nan, 'joy'
    ]
}

# Create a DataFrame from the data
df = pd.DataFrame(data)

# Step 1: Handle Missing Values
print("Missing values before cleanup:\n", df.isnull().sum())

# Drop rows with missing text or emotion
df.dropna(subset=['text', 'emotion'], inplace=True)

# Step 2: Remove Duplicates
df.drop_duplicates(subset='text', inplace=True)

# Step 3: Basic Data Inspection
print("\nData Sample After Cleanup:\n", df.head())

# Step 4: Visualizing the Distribution of Emotions
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='emotion', palette='Set2')
plt.title('Emotion Distribution')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()

# Step 5: Visualizing Text Length Distribution
df['text_length'] = df['text'].apply(lambda x: len(str(x)))
plt.figure(figsize=(8, 6))
sns.histplot(df['text_length'], bins=20, color='skyblue', kde=True)
plt.title('Text Length Distribution')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()

# Step 6: Clean the Text (remove URLs, mentions, hashtags, etc.)
def clean_text(text):
    text = re.sub(r"http\S+", "", text)       # Remove URLs
    text = re.sub(r"@\w+", "", text)          # Remove mentions
    text = re.sub(r"#\w+", "", text)          # Remove hashtags
    text = re.sub(r"[^A-Za-z\s]", "", text)   # Remove non-alphabetic characters
    return text.lower().strip()

df['clean_text'] = df['text'].apply(clean_text)

# Step 7: Visualizing Word Cloud (Most Frequent Words)
text_data = ' '.join(df['clean_text'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text_data)

plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Most Frequent Words')
plt.axis('off')
plt.show()

# Step 8: Check for Correlation (if numeric columns exist)
# For now, we only have text_length as a numeric column
plt.figure(figsize=(8, 6))
sns.heatmap(df[['text_length']].corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()

# Step 9: Displaying Most Common Emotions
emotion_counts = df['emotion'].value_counts()
print("\nMost Common Emotions:\n", emotion_counts)

# Step 10: Final Data Inspection after EDA
print("\nCleaned Data Sample:\n", df[['text', 'clean_text', 'emotion']].head())
```

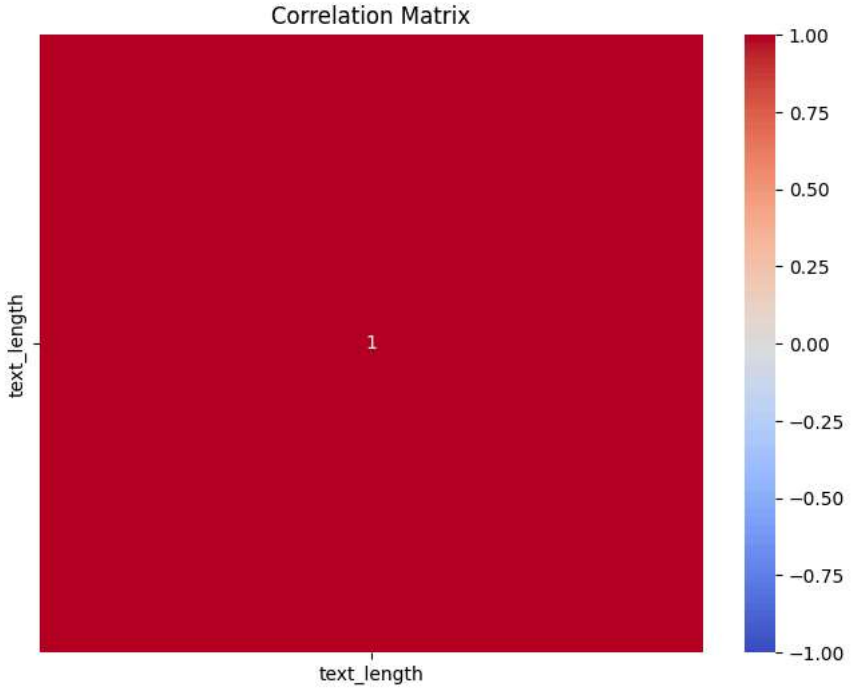
```
Missing values before cleanup:
 text       1
emotion    1
dtype: int64

Data Sample After Cleanup:
                                     text  emotion
0          I love this product! 😊 #happy      joy
1    Worst experience ever. Never again!    anger
3        Just okay... not good, not bad.  neutral
4  This is awesome!!! http://example.com      joy
6                 Feeling sad today... 😔  sadness
<ipython-input-4-7651579265fd>:45: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.countplot(data=df, x='emotion', palette='Set2')
```







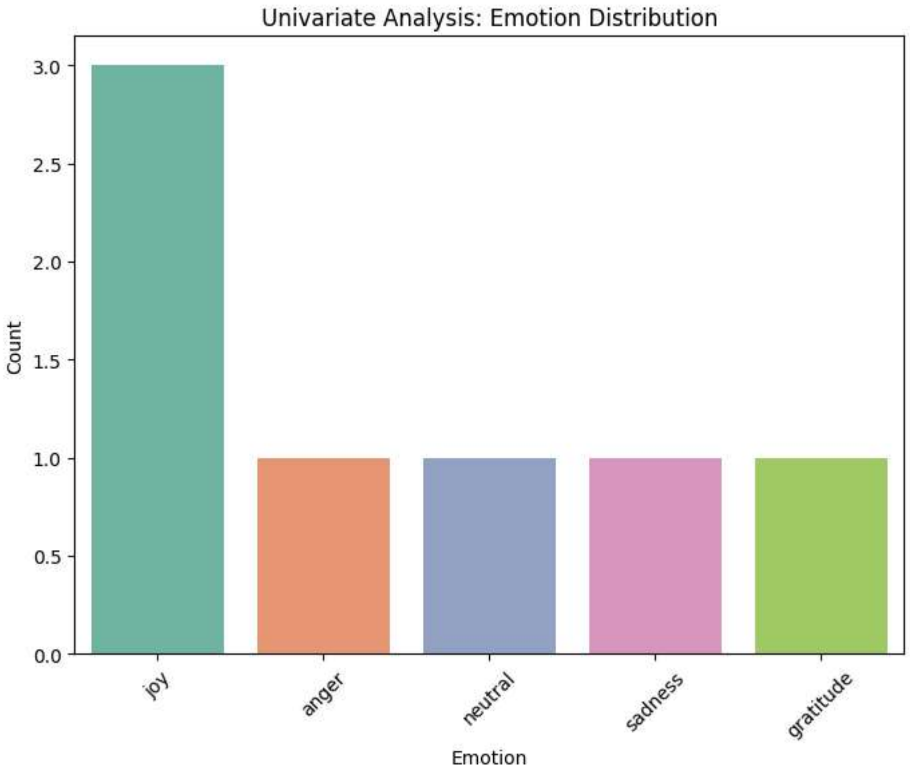


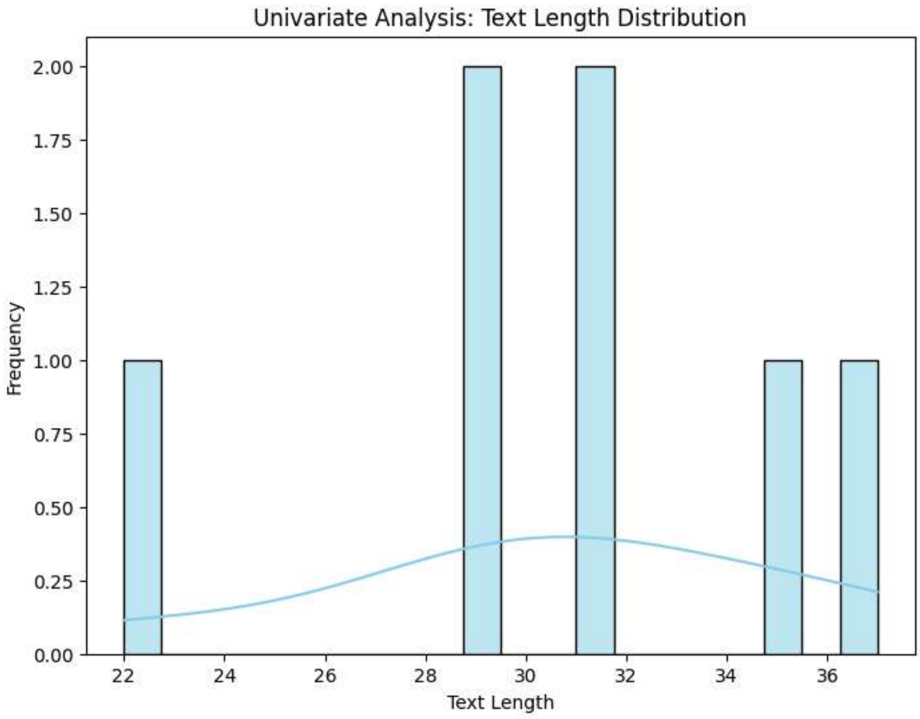


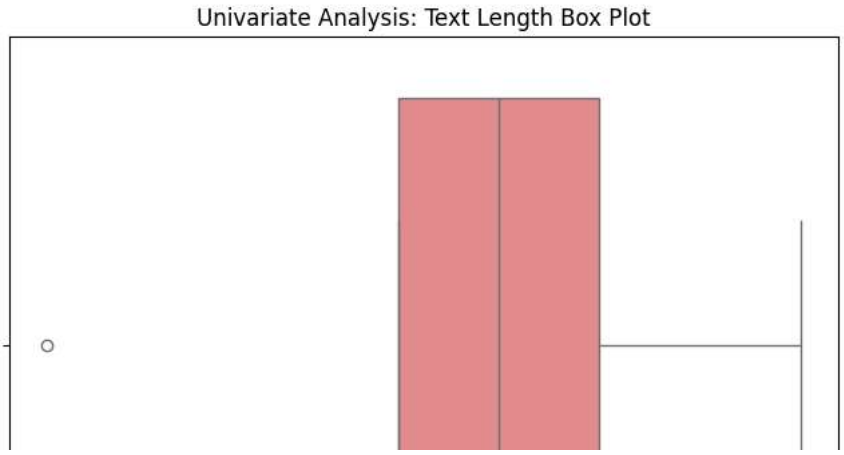
```
Most Common Emotions:
 emotion
joy          3
anger        1
neutral      1
sadness      1
gratitude    1
Name: count, dtype: int64

Cleaned Data Sample:
                                    text                         clean_text  \
0          I love this product! 😊 #happy                i love this product
1    Worst experience ever. Never again!  worst experience ever never again
3        Just okay... not good, not bad.         just okay not good not bad
4  This is awesome!!! http://example.com                    this is awesome
6                  Feeling sad today... 😔                  feeling sad today

   emotion
0      joy
1    anger
3  neutral
4      joy
6  sadness
```

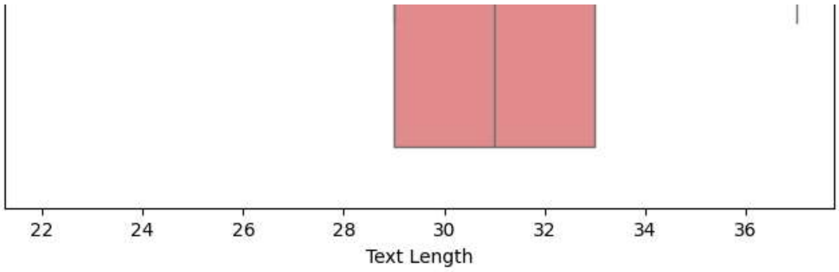
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Sample dataset
data = {
    'text': [
        'I love this product! 😊 #happy',
        'Worst experience ever. Never again!',
        np.nan,
        'Just okay... not good, not bad.',
        'This is awesome!!! http://example.com',
        'This is awesome!!! http://example.com',  # Duplicate
        'Feeling sad today... 😔',
        '@user Thanks for the support!',
        '',
        'Love the way you lie... #Eminem'
    ],
    'emotion': [
        'joy', 'anger', 'sadness', 'neutral', 'joy', 'joy', 'sadness', 'gratitude', np.nan, 'joy'
    ]
}

# Create a DataFrame from the data
df = pd.DataFrame(data)

# Handle Missing Values
df.dropna(subset=['text', 'emotion'], inplace=True)

# Remove Duplicates
df.drop_duplicates(subset='text', inplace=True)

# 1. Univariate Analysis of 'emotion' (Categorical Data)
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='emotion', palette='Set2')
plt.title('Univariate Analysis: Emotion Distribution')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()

# 2. Univariate Analysis of Text Length (Numeric Data)
df['text_length'] = df['text'].apply(lambda x: len(str(x)))

plt.figure(figsize=(8, 6))
sns.histplot(df['text_length'], bins=20, color='skyblue', kde=True)
plt.title('Univariate Analysis: Text Length Distribution')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()

# 3. Box Plot for Text Length (To check for outliers)
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['text_length'], color='lightcoral')
plt.title('Univariate Analysis: Text Length Box Plot')
plt.xlabel('Text Length')
plt.show()

# 4. Univariate Analysis of Text Length (Basic Statistics)
print("\nBasic Statistics of Text Length:")
print(df['text_length'].describe())

# 5. Checking for Skewness (Optional)
print("\nSkewness of Text Length:", df['text_length'].skew())
```

```
<ipython-input-5-9477f63755de>:36: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.countplot(data=df, x='emotion', palette='Set2')
```









```
Basic Statistics of Text Length:
count     7.000000
mean     30.571429
std       4.825527
min      22.000000
25%      29.000000
50%      31.000000
75%      33.000000
max      37.000000
Name: text_length, dtype: float64

Skewness of Text Length: -0.5914354055317241
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
import string
import nltk
from nltk.corpus import stopwords
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Download necessary NLTK data files (run this once)
nltk.download('stopwords')

# Sample dataset (replace with your dataset)
data = {
    'text': [
        "I am so happy today!",
        "This is a sad day.",
        "I love programming.",
        "I am angry about this situation.",
        "Feeling so good and relaxed!",
        "I'm so frustrated with everything."
    ],
    'emotion': ['joy', 'sadness', 'joy', 'anger', 'joy', 'anger']
}

# Create DataFrame
df = pd.DataFrame(data)

# Function to preprocess the text
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove punctuation
    text = ''.join([char for char in text if char not in string.punctuation])
    # Tokenize the text and remove stopwords
    stop_words = set(stopwords.words('english'))
    words = text.split()
    words = [word for word in words if word not in stop_words]
    return ' '.join(words)

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

# Feature Engineering: Text Length
df['text_length'] = df['clean_text'].apply(lambda x: len(x.split()))  # Word count

# Feature Engineering: Sentiment Scores using TextBlob
df['polarity'] = df['clean_text'].apply(lambda x: TextBlob(x).sentiment.polarity)
df['subjectivity'] = df['clean_text'].apply(lambda x: TextBlob(x).sentiment.subjectivity)

# Feature Engineering: TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=100)  # Limit to top 100 features
X_tfidf = vectorizer.fit_transform(df['clean_text'])
```

```
# Convert TF-IDF sparse matrix to DataFrame
tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer.get_feature_names_out())
df = pd.concat([df, tfidf_df], axis=1)

# Define the target variable (emotion) and features (X)
X = df.drop(columns=['text', 'emotion', 'clean_text'])
y = df['emotion']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Accuracy: 0.5

Classification Report:
              precision    recall  f1-score   support

       anger       0.00      0.00      0.00         0
         joy       1.00      1.00      1.00         1
     sadness       0.00      0.00      0.00         1

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

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# 1. Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 2. Load Dataset (Example CSV)
# Make sure your CSV has 'text' and 'label' columns
df = pd.read_csv("your_dataset.csv")  # Replace with actual file name

# 3. Basic Preprocessing
df.dropna(inplace=True)  # Remove missing values
X = df['text']
y = df['label']

# 4. Text Vectorization (TF-IDF)
vectorizer = TfidfVectorizer(max_features=5000)
X_vec = vectorizer.fit_transform(X)

# 5. Split Data
X_train, X_test, y_train, y_test = train_test_split(X_vec, y, test_size=0.2, random_state=42)

# 6. Train Model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 7. Predict & Evaluate
y_pred = model.predict(X_test)
```

```
print("✅ Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Confusion
```

```
  File "<ipython-input-7-4bf43498532b>", line 32
    print("\n📊 Confusion
          ^
SyntaxError: unterminated string literal (detected at line 32)
```

```
# 1. Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 2. Create a dummy dataset
data = {
    'text': [
        'I love this product!',
        'This is the worst experience ever.',
        'Not bad, could be better.',
        'Absolutely fantastic!',
        'Terrible service.',
        'I am happy with the result.',
        'I hate it.',
        'It was okay, not great.',
        'Amazing quality!',
        'Worst purchase I’ve made.'
    ],
    'label': ['positive', 'negative', 'neutral', 'positive', 'negative',
              'positive', 'negative', 'neutral', 'positive', 'negative']
}

df = pd.DataFrame(data)

# 3. Vectorize text using TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['text'])
y = df['label']

# 4. Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 5. Train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 6. Make predictions
y_pred = model.predict(X_test)

# 7. Evaluate the model
print("✅ Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\n📝 Classification Report:\n", classification_report(y_test, y_pred))
```

```
✅ Accuracy: 0.3333333333333333

📊 Confusion Matrix:
 [[1 0]
 [2 0]]

📝 Classification Report:
               precision    recall  f1-score   support

    negative       0.33      1.00      0.50         1
    positive       0.00      0.00      0.00         2

    accuracy                           0.33         3
   macro avg       0.17      0.50      0.25         3
weighted avg       0.11      0.33      0.17         3

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```