Task 1: Sentiment Analysis on Product Reviews

Description:

1. Dataset (Recommended): IMDB Dataset of 50K Movie Reviews (Kaggle)
2. Analyze product reviews to determine whether the sentiment is positive or negative
3. Clean and preprocess text (e.g., lowercasing, removing stopwords)
4. Convert text to numerical format using TF-IDF or CountVectorizer.

5. Train a binary classifier (e.g., logistic regression) and evaluate its performance

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("lakshmi25npathi/imdb-dataset-of-50k-movie-reviews")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/imdb-dataset-of-50k-movie-reviews
```

```
import os
import pandas as pd

# Construct the full path to the CSV file
csv_file_path = os.path.join(path, 'IMDB Dataset.csv')

# Load the dataset
df = pd.read_csv(csv_file_path)

# Display the first few rows
display(df.head())
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Data Preprocessing

"Preprocess the text data to ensure it is clean and ready for sentiment analysis. The steps should include:

Lowercasing all text to maintain uniformity.

Removing special characters, punctuation, and numbers while keeping essential words.

Removing stopwords (e.g., 'is', 'the', 'and') to reduce noise.

Tokenizing text into individual words.

Stemming or Lemmatization to reduce words to their root form.

Handling emojis, slang, or abbreviations to interpret sentiment correctly.

Handling negations (e.g., 'not good' should not become 'good').

Converting text into numerical form (e.g., TF-IDF, Bag-of-Words, or Word Embeddings) for machine learning models. Ensure that preprocessing preserves sentiment-bearing words and context."

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer

# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('punkt')

# Initialize stemmer and stopwords
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Lowercasing
    text = text.lower()

    # Handling negations (simple approach: replace 'not' with 'not_')
    text = re.sub(r'not\s(\w+)', r'not_\1', text)

    # Remove special characters, punctuation, and numbers (keeping spaces)
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenization
    tokens = text.split()

    # Remove stopwords and stem (or lemmatize if preferred)
    # We are not stemming/lemmatizing here to preserve the negation handling
    # tokens = [stemmer.stem(word) for word in tokens if word not in stop_words]
    tokens = [word for word in tokens if word not in stop_words]


    return ' '.join(tokens)

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

# Convert text to numerical format using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000) # Limit features for demonstration
X = tfidf_vectorizer.fit_transform(df['cleaned_review'])

# Display the shape of the resulting matrix
```

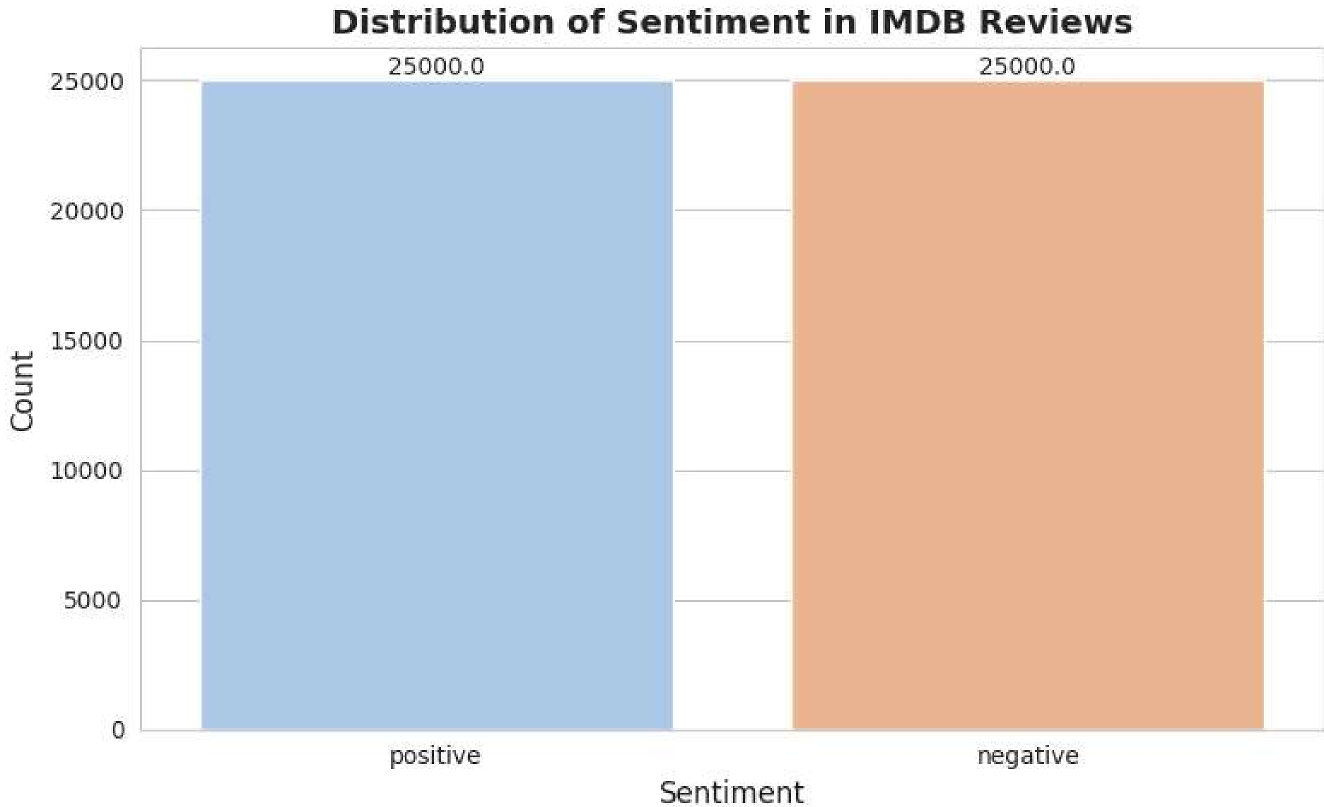
```
print("Shape of TF-IDF matrix:", X.shape)

# Display the first 5 cleaned reviews
print("\nFirst 5 cleaned reviews:")
for i in range(5):
    print(f"Review {i+1}: {df['cleaned_review'].iloc[i]}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Shape of TF-IDF matrix: (50000, 5000)

First 5 cleaned reviews:
Review 1: one reviewers mentioned watching oz episode youll hooked right exactly happene
Review 2: wonderful little production br br filming technique unassuming oldtimebbc fash
Review 3: thought wonderful way spend time hot summer weekend sitting air conditioned th
Review 4: basically theres family little boy jake thinks theres zombie closet parents fi
Review 5: petter matteis love time money visually stunning film watch mr mattei offers u
```

Exploratory Data Analysis EDA

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set a clean style
sns.set_style("whitegrid")

# Create the plot
plt.figure(figsize=(8, 5))
ax = sns.countplot(data=df, x='sentiment', palette='pastel')

# Add title and labels
plt.title('Distribution of Sentiment in IMDB Reviews', fontsize=14, weight='bold')
plt.xlabel('Sentiment', fontsize=12)
plt.ylabel('Count', fontsize=12)

# Add value labels on top of each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height()}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-3139953124.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  ax = sns.countplot(data=df, x='sentiment', palette='pastel')
```




```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all cleaned reviews into a single string
all_reviews = " ".join(review for review in df['cleaned_review'])

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, random_state=21, max_font_size=110, collocatior

# Display the word cloud
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

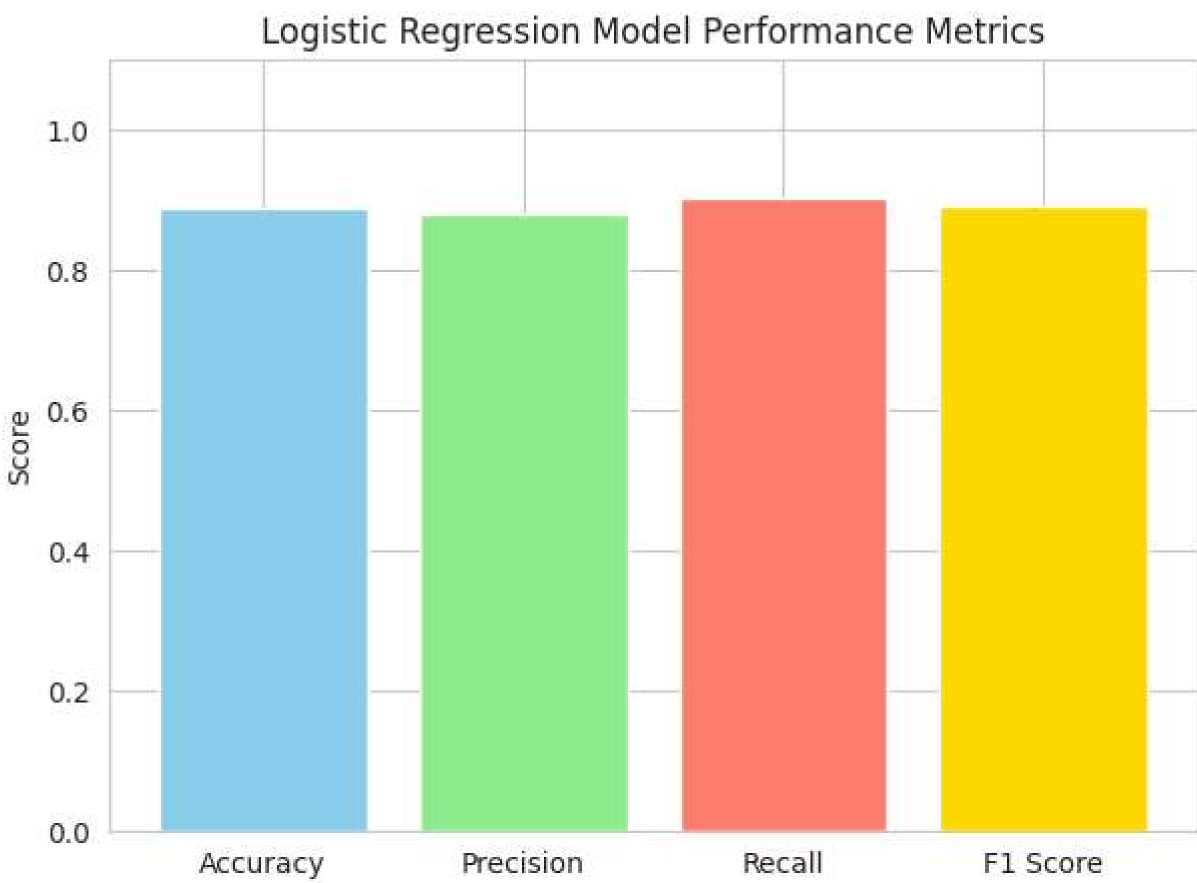
LogisticRegression Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Convert sentiment labels to numerical format (0 for negative, 1 for positive)
df['sentiment_numeric'] = df['sentiment'].apply(lambda x: 1 if x == 'positive' else 0)
y = df['sentiment_numeric']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.8905
Precision: 0.8819
Recall: 0.9038
F1 Score: 0.8927
```

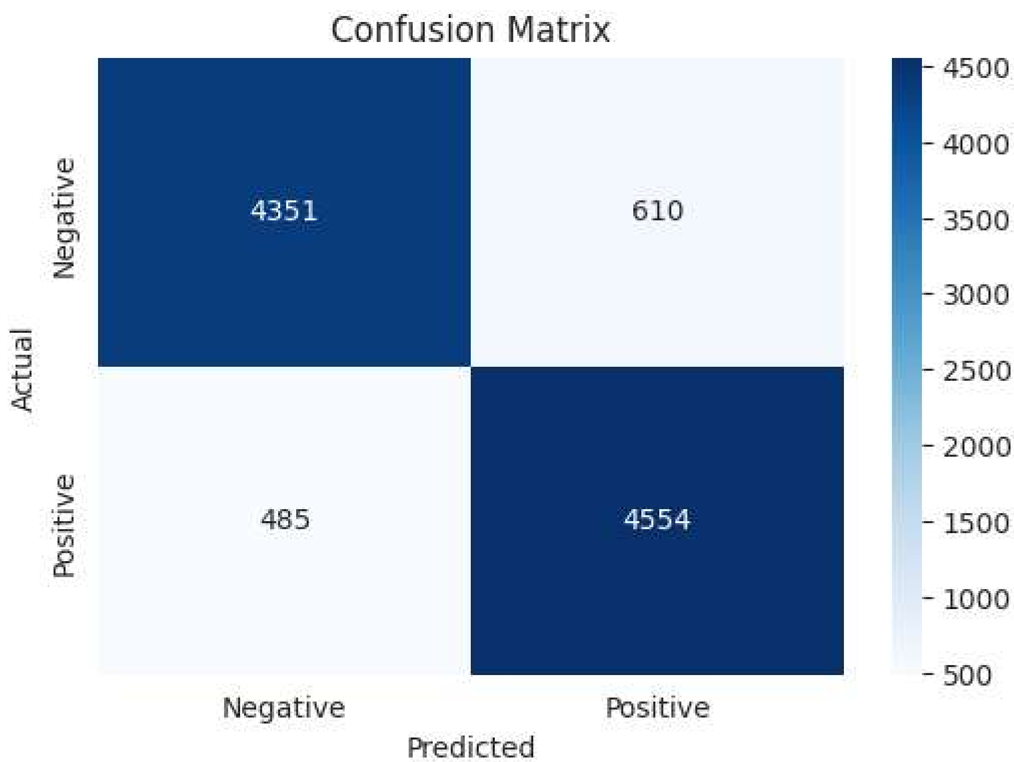
```
import matplotlib.pyplot as plt
import numpy as np

# Performance metrics (assuming these variables are available from the previous model traini
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
values = [accuracy, precision, recall, f1]

plt.figure(figsize=(7, 5))
plt.bar(metrics, values, color=['skyblue', 'lightgreen', 'salmon', 'gold'])
plt.ylim(0, 1.1) # Set y-axis limit from 0 to 1.1 for better visualization of scores
plt.ylabel('Score')
plt.title('Logistic Regression Model Performance Metrics')
plt.show()
```



```
from sklearn.metrics import classification_report

# Generate and print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       0.90      0.88      0.89      4961
           1       0.88      0.90      0.89      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

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

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix using seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yti
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

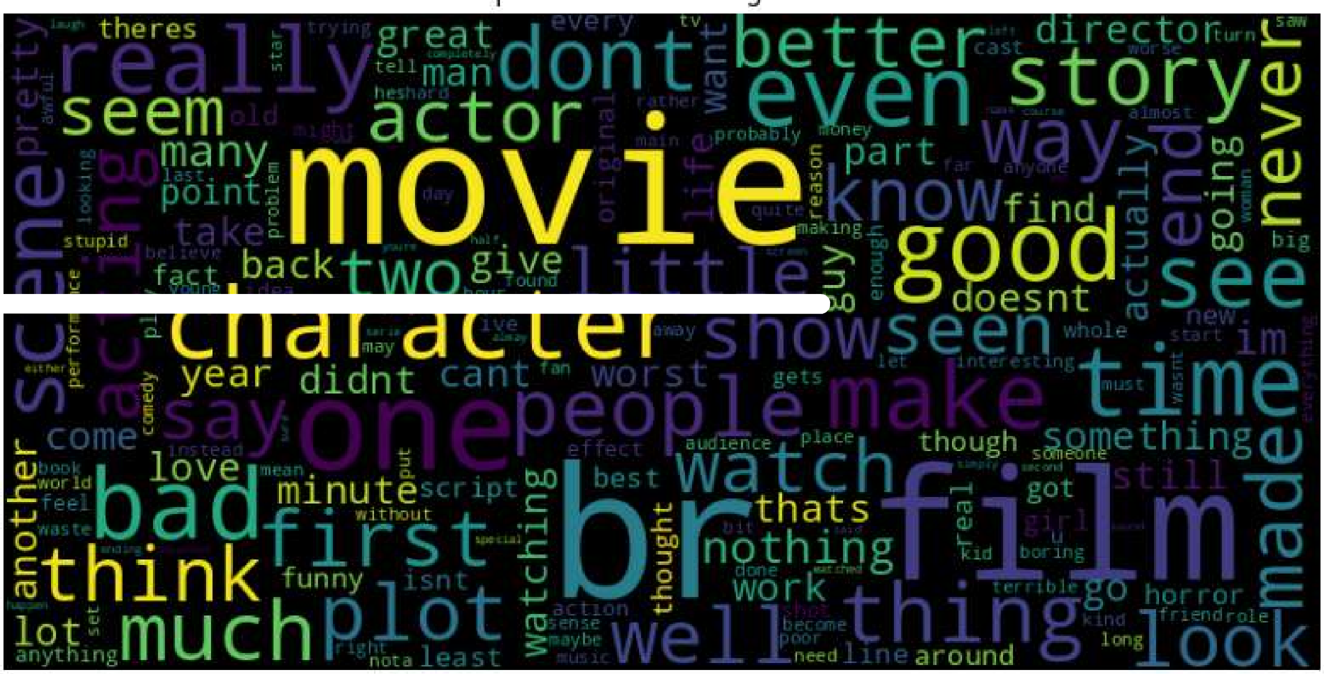



```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Separate positive and negative reviews
positive_reviews = df[df['sentiment'] == 'positive']['cleaned_review']
negative_reviews = df[df['sentiment'] == 'negative']['cleaned_review']

# Combine positive reviews into a single string
all_positive_reviews = " ".join(review for review in positive_reviews)

# Combine negative reviews into a single string
all_negative_reviews = " ".join(review for review in negative_reviews)

# Generate word cloud for positive reviews
wordcloud_positive = WordCloud(width=800, height=400, random_state=21, max_font_size=110, cc

# Generate word cloud for negative reviews
wordcloud_negative = WordCloud(width=800, height=400, random_state=21, max_font_size=110, cc

# Display word cloud for positive reviews
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud_positive, interpolation='bilinear')
plt.title('Most Frequent Words in Positive Reviews')
plt.axis('off')
plt.show()

# Display word cloud for negative reviews
plt.figure(figsize=(10, 7))
```

```
plt.imshow(wordcloud_negative, interpolation='bilinear')
plt.title('Most Frequent Words in Negative Reviews')
plt.axis('off')
plt.show()
```

Most Frequent Words in Positive Reviews

Most Frequent Words in Negative Reviews



Bonus:

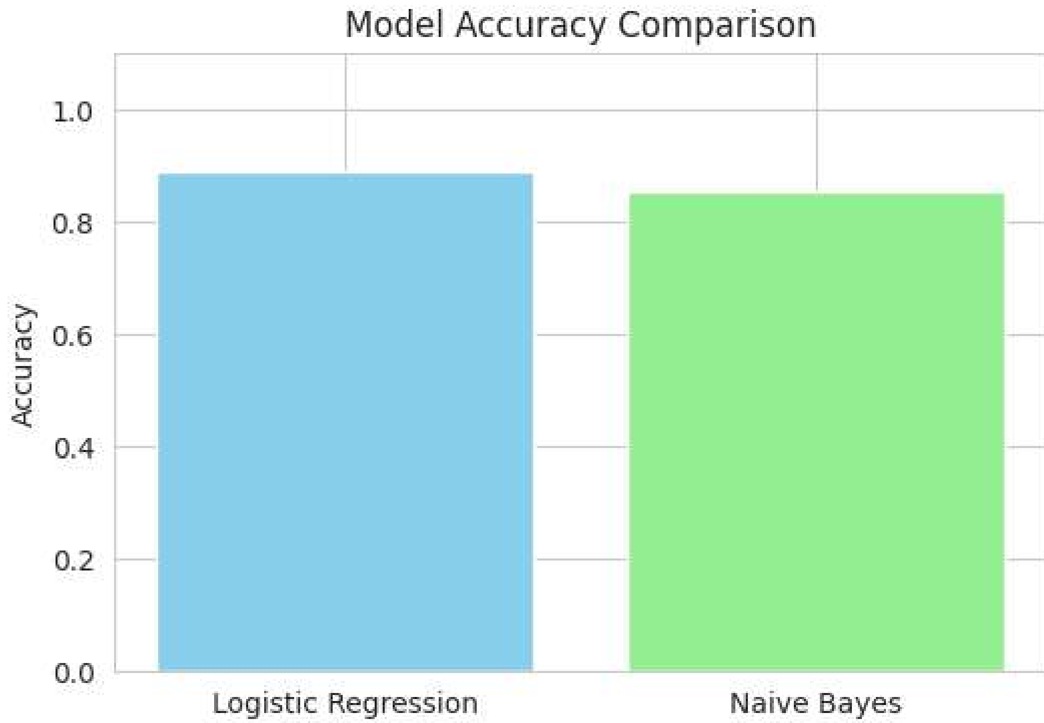
Visualize the most frequent positive and negative words

Try using a Naive Bayes classifier and compare accuracy

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Train a Naive Bayes classifier
naive_bayes_model = MultinomialNB()
naive_bayes_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_nb = naive_bayes_model.predict(X_test)

# Evaluate the Naive Bayes model
accuracy_nb = accuracy_score(y_test, y_pred_nb)

print(f"Naive Bayes Accuracy: {accuracy_nb:.4f}")
print(f"Logistic Regression Accuracy: {accuracy:.4f}")

# Compare the accuracies
if accuracy_nb > accuracy:
    print("Naive Bayes performed better than Logistic Regression.")
elif accuracy_nb < accuracy:
    print("Logistic Regression performed better than Naive Bayes.")
else:
    print("Naive Bayes and Logistic Regression performed equally well.")
```

```
Naive Bayes Accuracy: 0.8531
Logistic Regression Accuracy: 0.8905
Logistic Regression performed better than Naive Bayes.
```

```
import matplotlib.pyplot as plt
import numpy as np

# Model names and their accuracies
models = ['Logistic Regression', 'Naive Bayes']
accuracies = [accuracy, accuracy_nb]

plt.figure(figsize=(6, 4))
plt.bar(models, accuracies, color=['skyblue', 'lightgreen'])
plt.ylim(0, 1.1) # Set y-axis limit from 0 to 1.1
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.show()
```




```
from sklearn.metrics import classification_report

print("Naive Bayes Classification Report:")
print(classification_report(y_test, y_pred_nb))

print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       0.85      0.85      0.85      4961
           1       0.85      0.86      0.85      5039

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000


Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      4961
           1       0.88      0.90      0.89      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

```
# Example of predicting sentiment for a new review
new_review = "This movie was fantastic! I loved every minute of it."

# Preprocess the new review using the same function
cleaned_new_review = preprocess_text(new_review)

# Vectorize the cleaned review using the fitted TF-IDF vectorizer
# We use transform, not fit_transform, as the vectorizer is already fitted on the training c
new_review_vectorized = tfidf_vectorizer.transform([cleaned_new_review])

# Predict the sentiment using the trained Logistic Regression model
predicted_sentiment_numeric = model.predict(new_review_vectorized)

# Convert the numerical prediction back to a sentiment label
predicted_sentiment = 'positive' if predicted_sentiment_numeric[0] == 1 else 'negative'

print(f"Original review: '{new_review}'")
print(f"Cleaned review: '{cleaned_new_review}'")
print(f"Predicted sentiment: {predicted_sentiment}")

# Example with a negative review
new_review_negative = "The plot was terrible and the acting was awful."

cleaned_new_review_negative = preprocess_text(new_review_negative)
new_review_negative_vectorized = tfidf_vectorizer.transform([cleaned_new_review_negative])
predicted_sentiment_negative_numeric = model.predict(new_review_negative_vectorized)
predicted_sentiment_negative = 'positive' if predicted_sentiment_negative_numeric[0] == 1 el

print(f"\nOriginal review: '{new_review_negative}'")
print(f"Cleaned review: '{cleaned_new_review_negative}'")
print(f"Predicted sentiment: {predicted_sentiment_negative}")
```

```
Original review: 'This movie was fantastic! I loved every minute of it.'
Cleaned review: 'movie fantastic loved every minute'
Predicted sentiment: positive

Original review: 'The plot was terrible and the acting was awful.'
Cleaned review: 'plot terrible acting awful'
Predicted sentiment: negative
```

Deployment

```
!pip install -q gradio

import gradio as gr

# Define a prediction function that takes text input and returns the predicted sentiment
def predict_sentiment(review):
    # Preprocess the review
```

```
        cleaned_review = preprocess_text(review)

        # Vectorize the cleaned review
        review_vectorized = tfidf_vectorizer.transform([cleaned_review])

        # Predict the sentiment using the trained model (using Logistic Regression model)
        predicted_sentiment_numeric = model.predict(review_vectorized)

        # Convert the numerical prediction back to a sentiment label
        predicted_sentiment = 'positive' if predicted_sentiment_numeric[0] == 1 else 'negative'

        return predicted_sentiment

    # Create the Gradio interface
    interface = gr.Interface(
        fn=predict_sentiment,
        inputs=gr.Textbox(lines=5, label="Enter your movie review:"),
        outputs=gr.Textbox(label="Predicted Sentiment:"),
        title="Movie Review Sentiment Analysis",
        description="Enter a movie review to get a sentiment prediction (positive or negative)."
    )

    # Launch the interface
    interface.launch(debug=True)
```

```
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share

Colab notebook detected. This cell will run indefinitely so that you can see errors and
* Running on public URL: https://c33609b5f2d0d91c23.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gra
```

Movie Review Sentiment Analysis