```
# Install necessary libraries if not already installed
!pip install -q scikit-learn pandas matplotlib seaborn nltk

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```
# Load the dataset (ensure you've uploaded the file)
df = pd.read_csv('IMDB Dataset.csv')

# Show the first few rows of the dataset
df.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Download NLTK stopwords and WordNet for lemmatization
nltk.download('stopwords')
nltk.download('wordnet')

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Clean the text function
def clean_text(text):
    # Remove non-alphabetic characters and lowercase the text
    text = ''.join([char.lower() for char in text if char.isalpha() or char.isspace()])
    # Remove stopwords and lemmatize the words
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if word not in stop_words])
    return text

# Apply cleaning function to the 'review' column
df['cleaned_review'] = df['review'].apply(clean_text)

# Show a sample cleaned review
df[['review', 'cleaned_review']].head()
```

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

	review	cleaned_review
0	One of the other reviewers has mentioned that ...	one reviewer mentioned watching oz episode you...
1	A wonderful little production. <br /><br />The...	wonderful little production br br filming tech...
2	I thought this was a wonderful way to spend ti...	thought wonderful way spend time hot summer we...
3	Basically there's a family where a little boy ...	basically there family little boy jake think t...
4	Petter Mattei's "Love in the Time of Money" is...	petter matteis love time money visually stunni...

```
# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(df['cleaned_review'], df['sentiment'], test_size=0.2, random_state=42)

# Check the split sizes
print(f'Training data size: {len(X_train)}')
print(f'Test data size: {len(X_test)}')
```

```
Training data size: 40000
Test data size: 10000
```

```
# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Fit and transform the training data, then transform the test data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Check the shape of the transformed data (number of reviews and features)
X_train_tfidf.shape
```

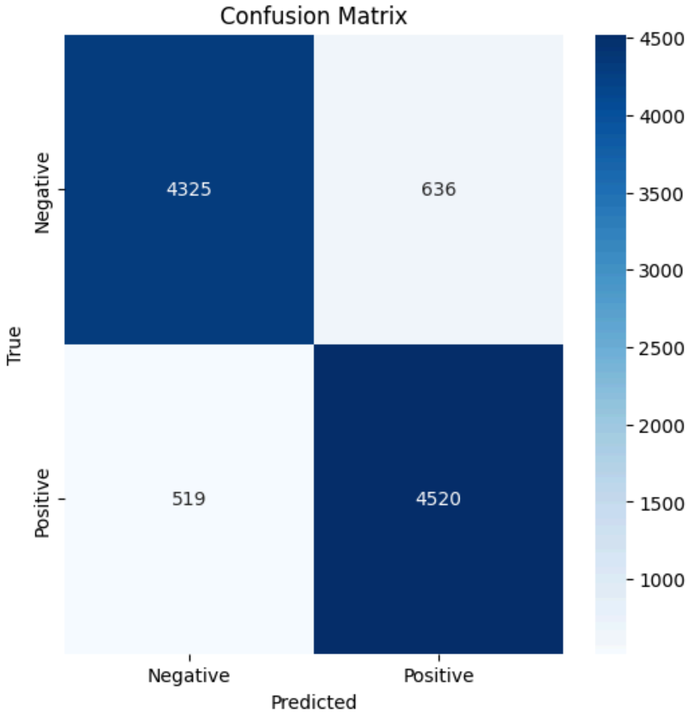
```
(40000, 5000)
```

```
# Initialize Logistic Regression model
model = LogisticRegression(max_iter=1000)

# Train the model
model.fit(X_train_tfidf, y_train)

# Predict on the test data
y_pred = model.predict(X_test_tfidf)

# Display classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    negative       0.89      0.87      0.88      4961
    positive       0.88      0.90      0.89      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

```
# Accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Classification Report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8845
```



```
              precision    recall  f1-score   support

    negative       0.89      0.87      0.88      4961
    positive       0.88      0.90      0.89      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

```
from sklearn.model_selection import GridSearchCV

# Hyperparameter grid for Logistic Regression
param_grid = {
    'C': [0.1, 1, 10],
    'max_iter': [100, 500, 1000]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=3, verbose=1, n_jobs=-1)

# Fit the grid search
grid_search.fit(X_train_tfidf, y_train)

# Get the best parameters and model
print("Best Parameters: ", grid_search.best_params_)

# Use the best model to predict
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test_tfidf)

# Evaluate the best model
print(classification_report(y_test, y_pred_best))
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
Best Parameters:  {'C': 1, 'max_iter': 100}
              precision    recall  f1-score   support

    negative       0.89      0.87      0.88      4961
    positive       0.88      0.90      0.89      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

```
# Get the feature importance (coefficients)
coefficients = model.coef_.flatten()

# Get feature names (words in the TF-IDF model)
features = np.array(tfidf_vectorizer.get_feature_names_out())

# Create a DataFrame with words and their corresponding coefficients
coef_df = pd.DataFrame({'word': features, 'coef': coefficients})
coef_df = coef_df.sort_values(by='coef', ascending=False)

# Display the top 10 words with the highest coefficients (positive sentiment)
print(coef_df.head(10))

# Display the top 10 words with the lowest coefficients (negative sentiment)
print(coef_df.tail(10))
```

```
           word      coef
1909      great  6.990502
1503  excellent  6.645027
3199    perfect  5.159948
152     amazing  4.821603
409        best  4.794773
4927  wonderful  4.787222
1617   favorite  4.616906
524   brilliant  4.532851
2630      loved  4.377516
2061  hilarious  4.179365
          word       coef
1327      dull  -5.088395
3303    poorly  -5.105705
3014   nothing  -5.415703
3302      poor  -5.849840
4426  terrible  -5.850851
484     boring  -6.595604
323        bad  -7.246964
314      awful  -7.385913
4820     waste  -7.895741
4948     worst -10.312158
```

```
from sklearn.naive_bayes import MultinomialNB

# Initialize Naive Bayes model
nb_model = MultinomialNB()

# Train the model
nb_model.fit(X_train_tfidf, y_train)

# Predict and evaluate
y_pred_nb = nb_model.predict(X_test_tfidf)
print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

    negative       0.85      0.85      0.85      4961
    positive       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
```

```
import joblib

# Save the model
joblib.dump(model, 'sentiment_model.pkl')

# Load the model for deployment
model_loaded = joblib.load('sentiment_model.pkl')
```

```
from sklearn.model_selection import cross_val_score

# Perform 5-fold cross-validation
cross_val_score(model, X_train_tfidf, y_train, cv=5, scoring='accuracy').mean()
```

```
0.88375
```