

step1:Import Required Libraries

```
# Import pandas and numpy for data handling
import pandas as pd
import numpy as np

# Import libraries for text cleaning
import re
import string

# Import TF-IDF for feature extraction
from sklearn.feature_extraction.text import TfidfVectorizer

# Import Logistic Regression model
from sklearn.linear_model import LogisticRegression

# Import function to split data
from sklearn.model_selection import train_test_split

# Import evaluation metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Import libraries for visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

step2:Load IMDB Dataset

```
# Load the IMDB movie review dataset
# Make sure IMDB Dataset.csv is in the same folder as this file
df = pd.read_csv("IMDB Dataset.csv")

# Display first 5 rows of the dataset
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	

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
# Display column names and data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  -- 
 0   review      50000 non-null   object 
 1   sentiment   50000 non-null   object 
dtypes: object(2)
memory usage: 781.4+ KB
```

step3:Convert Sentiment Labels to Numeric

```
# Convert sentiment labels into numeric form
# positive -> 1, negative -> 0
df["sentiment"] = df["sentiment"].map({
    "positive": 1,
    "negative": 0
})

# Check conversion
df["sentiment"].value_counts()
```

```

count
sentiment
   1    25000
   0    25000

dtype: int64

```

step4:Text Preprocessing

```

def clean_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation
    text = text.translate(
        str.maketrans('', '', string.punctuation)
    )

    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text

```

```

# Apply text cleaning to all reviews
df["clean_review"] = df["review"].apply(clean_text)

# Display original and cleaned reviews
df[["review", "clean_review"]].head()

```

	review	clean_review	
0	One of the other reviewers has mentioned that ...	one of the other reviewers has mentioned that ...	
1	A wonderful little production. The...	a wonderful little production the filming tech...	
2	I thought this was a wonderful way to spend ti...	i thought this was a wonderful way to spend ti...	
3	Basically there's a family where a little boy ...	basically theres a family where a little boy j...	
4	Petter Mattei's "Love in the Time of Money" is...	petter matteis love in the time of money is a ...	

step5:Feature Extraction using TF-IDF

```

# Initialize TF-IDF Vectorizer
tfidf = TfidfVectorizer(
    stop_words="english",
    max_features=5000  # Limit features for efficiency
)

# Convert cleaned text into TF-IDF feature matrix
X = tfidf.fit_transform(df["clean_review"])

# Target variable
y = df["sentiment"]

# Print vocabulary size
print("Vocabulary size:", len(tfidf.vocabulary_))

```

Vocabulary size: 5000

step6:Train-Test Split

```

# Split dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,

```

```
    test_size=0.2,
    random_state=42,
    stratify=y # Keeps class balance
)
```

step7:Train Logistic Regression Model

```
# Create Logistic Regression model
# max_iter increased to ensure convergence
model = LogisticRegression(max_iter=1000)

# Train the model on training data
model.fit(X_train, y_train)
```

```
▼ LogisticRegression ⓘ (?)  
LogisticRegression(max_iter=1000)
```

step8:Make Predictions

```
# Predict sentiment for test data
y_pred = model.predict(X_test)
```

step9:Model Evaluation

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

```
Accuracy: 0.8865
```

Precision, Recall, F1-Score

```
# Print detailed classification report
print(classification_report(y_test, y_pred))
```

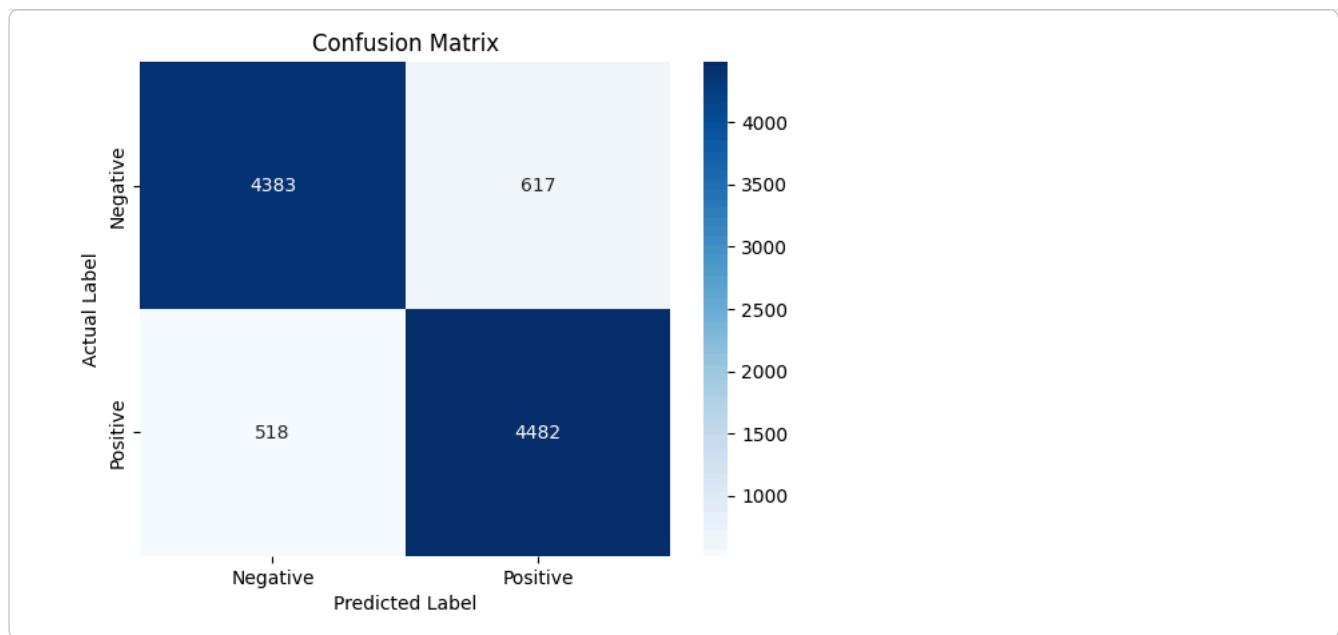
	precision	recall	f1-score	support
0	0.89	0.88	0.89	5000
1	0.88	0.90	0.89	5000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

Confusion Matrix

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

# Plot confusion matrix
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Negative", "Positive"],
    yticklabels=["Negative", "Positive"]
)

plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.title("Confusion Matrix")
plt.show()
```



step10: Analyze Important Words (Model Interpretation)

```
# Get feature names (words)
feature_names = tfidf.get_feature_names_out()

# Get Logistic Regression coefficients
coefficients = model.coef_[0]

# Create DataFrame of words and weights
coef_df = pd.DataFrame({
    "Word": feature_names,
    "Weight": coefficients
})

# Top positive words
print("Top Positive Words:")
coef_df.sort_values(by="Weight", ascending=False).head(10)

# Top negative words
print("\nTop Negative Words:")
coef_df.sort_values(by="Weight").head(10)
```

Top Positive Words:

Top Negative Words:

	Word	Weight	grid icon
4954	worst	-9.907928	
4826	waste	-8.345805	
305	awful	-7.759973	
315	bad	-7.287905	
468	boring	-6.011418	
3297	poor	-5.555322	
3298	poorly	-5.410190	
4431	terrible	-5.303510	
4953	worse	-5.282704	
1350	dull	-5.233018	

