📊 Sentiment Analysis Final Project

🎯 Project Overview

Welcome to your Sentiment Analysis final project! In this project, you will:

- 🔍 **Analyze** customer reviews or social media text data
- 🧹 **Preprocess** text data using NLP techniques
- 🤖 **Build** and compare multiple classification models
- 📈 **Evaluate** model performance using appropriate metrics
- 💼 **Communicate** findings in a business context

📋 Project Requirements

- **Dataset**: Use a public sentiment dataset (e.g., IMDb reviews, Amazon reviews, airline tweets)
- **Models**: Train at least **2 different classification models**
- **Evaluation**: Use multiple metrics (accuracy, precision, recall, confusion matrix)
- **Documentation**: Use markdown cells to explain your approach, findings, and insights

💡 Important Tips for Success

Working with Limited Resources

- ⚡ **Start small**: Use `df.sample(n=1000)` or `df.head(5000)` to work with a subset
- 🎯 **Limit vocabulary**: Use `max_features=5000` in TfidfVectorizer
- 🚀 **Choose efficient models**: LogisticRegression and MultinomialNB are fast and effective
- 💾 **Save your work frequently**: Use Ctrl+S or Cmd+S often

Best Practices

- 📝 **Document everything**: Explain your choices and observations in markdown cells
- 🔄 **Iterate**: Start simple, then improve
- 📊 **Visualize**: Use plots to understand your data and results

- 🎓 **Think like a data scientist**: Always interpret your results in context

---

Let's get started! 🚀

Part 1: Project Definition 🎯

Objectives

- Define the business problem you're solving
- Describe your chosen dataset
- Explain why sentiment analysis is valuable for this use case

Instructions

In the markdown cell below, answer these questions:

1. **What is the business problem?**
    - What decision or insight will this sentiment analysis support?
2. **What dataset are you using?**
    - Name and source of the dataset
    - Number of samples
    - What the text represents (reviews, tweets, comments, etc.)
    - What are the sentiment labels (positive/negative, star ratings, etc.)?
3. **Why is this problem important?**
    - How could the results be used in real-world scenarios?

📝 Your Project Definition

Business Problem:

- *Knowing what audiences like in an ever changing world is tough to predict. Each movie you dont make is a competitor with yours and youre only as good as your latest movie produced.

  This sentiment analysis support gives clear and concise insight into viewer preferences. It details criteria that will bring pleasure to its audiences and gives data backed insight to executives in need of making a decision.*

Dataset Description:

[IMDB Movie Reviews]

- **Dataset name**: IMDB Dataset.csv
- **Source**: Kaggle
- **Number of samples**: 50,000
- **Text type**: Movie reviews from users
- **Sentiment labels**: Positive or Negative

Importance and Real-World Applications:

[This analysis can give in real time forecast on audiences fellings towards our IP portfolio, it can refine and streamline marketing,personalise recomendations for our streaming service and delivers clarity in high impact meetings.]

```
import pandas as pd
import os

# Configuration
file_name = "IMDB Dataset.csv"
# Official Source
url = "https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movi

print(f"Dataset Source: {url}")

# Check if the file exists in the current directory
if os.path.exists(file_name):
    try:
        # Load the dataset
        # python engine is used for better robustness against formatting issues
        df = pd.read_csv(file_name, engine='python', on_bad_lines='warn')
        print(f"\nDataset shape: {df.shape}")
        print(f"\nColumn names: {df.columns.tolist()}")
        display(df.head())
    except Exception as e:
        print(f"Error reading dataset: {e}")
else:
    print(f"❌ File '{file_name}' not found.")
    print(f"Please download the dataset from the Kaggle link above and manually
Dataset Source: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-

Dataset shape: (50000, 2)

Column names: ['review', 'sentiment']
```

	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

Part 2: Exploratory Data Analysis (EDA) 🔍

Objectives

- Load and examine your dataset
- Understand the distribution of sentiments
- Analyze text characteristics (length, common words, etc.)
- Identify any data quality issues

What to Explore

✅ **Dataset structure**: Shape, columns, data types

✅ **Missing values**: Check for and handle missing data

✅ **Class distribution**: Are sentiments balanced?

✅ **Text length**: Average, min, max review lengths

✅ **Common words**: Most frequent words per sentiment

✅ **Sample reviews**: Display examples from each class

💡 Tips

- Use `.info()`, `.describe()`, and `.value_counts()` for quick insights
- Visualize distributions with bar plots and histograms
- Look for imbalanced classes that might affect model performance
- Create a word cloud to visualize common terms (optional but impressive!)

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# Set visualization style
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)

print("✅ Libraries imported successfully!")
```

```
✅ Libraries imported successfully!
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# Set visualization style
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)

# The dataset 'df' has already been loaded successfully in cell d5301c63.
# We will proceed with EDA using the existing 'df' DataFrame.

# Display basic information (already available from df loaded in d5301c63)
print(f"\nDataset shape: {df.shape}")
print(f"\nColumn names: {df.columns.tolist()}")
df.head()
```

```
Dataset shape: (52233, 2)

Column names: ['review', 'sentiment']
```

	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

```
if 'df' in locals():
    # Check for missing values
    print("🔍 Checking for missing values in the full dataset...")
    missing_counts = df.isnull().sum()
    display(missing_counts)

    if missing_counts.sum() > 0:
        print("⚠️ Found missing values. Consider dropping or filling them.")
    else:
        print("✅ No missing values found! The data is clean.")
else:
    print("❌ DataFrame 'df' not found. Please run the load cell first.")
```

```
🔍 Checking for missing values in the full dataset...
```

	0
review	0
sentiment	0

dtype: int64

```
✅ No missing values found! The data is clean.
```

```
if 'df' in locals():
    # Sample 5000 entries for efficient processing
    # We use a fixed random_state=42 for reproducibility
    if len(df) > 5000:
        print(f"📉 Sampling 5000 reviews from {len(df)} total reviews for effi
        df = df.sample(n=5000, random_state=42).reset_index(drop=True)
        print(f"✅ Dataset sampled! New shape: {df.shape}")
    else:
        print(f"ℹ️ Dataset already has {len(df)} samples (<= 5000). No samplin

    display(df.head())
```

```
else:
    print("❌ DataFrame 'df' not found. Please run the data loading cell first
```

```
📉 Sampling 5000 reviews from 50000 total reviews for efficiency...
✅ Dataset sampled! New shape: (5000, 2)
```

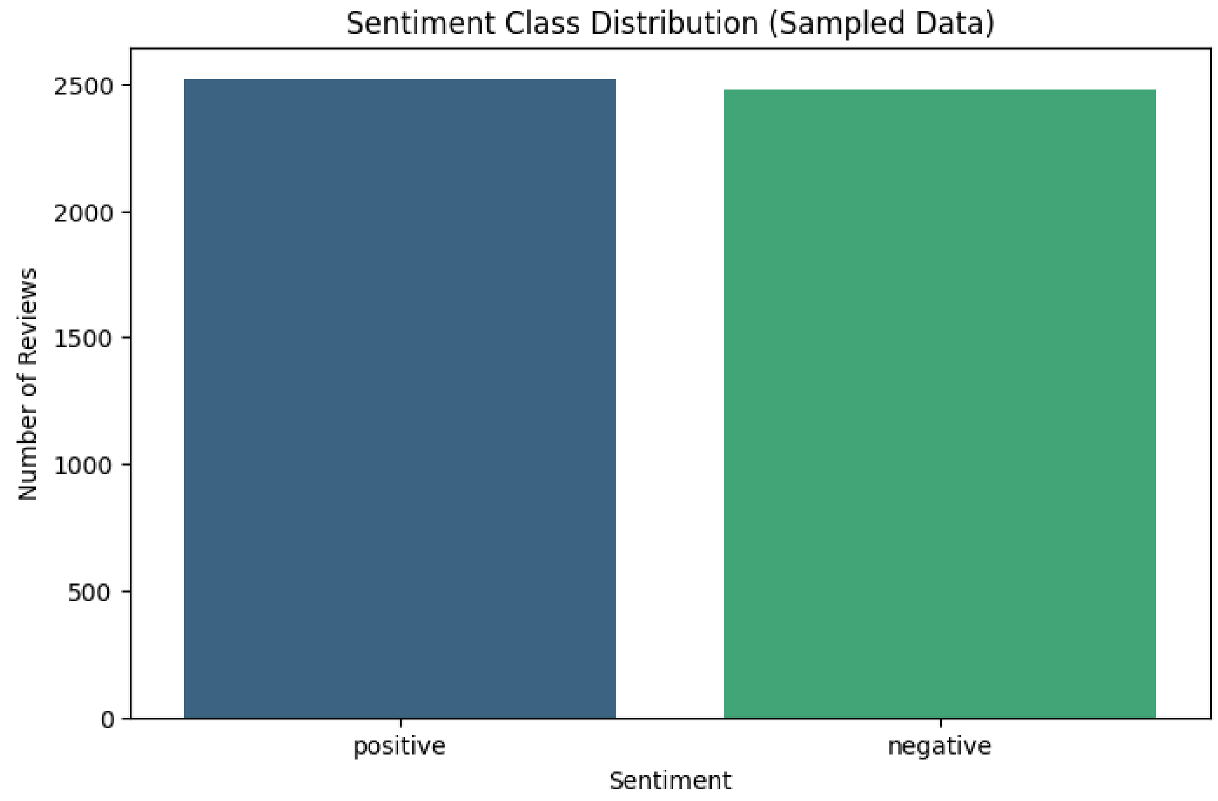
	review	sentiment
0	I really liked this Summerslam due to the look...	positive
1	Not many television shows appeal to quite as m...	positive
2	The film quickly gets to a major chase scene w...	negative
3	Jane Austen would definitely approve of this o...	positive
4	Expectations were somewhat high for me when I ...	negative

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

if 'df' in locals():
    # Analyze sentiment distribution
    sentiment_counts = df['sentiment'].value_counts()
    print("📊 Sentiment Distribution:")
    display(sentiment_counts)

    # Visualize with a bar plot
    plt.figure(figsize=(8, 5))
    # Fix: Assign x to hue and set legend=False to avoid FutureWarning
    sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values, hue=sentim
    plt.title('Sentiment Class Distribution (Sampled Data)')
    plt.xlabel('Sentiment')
    plt.ylabel('Number of Reviews')
    plt.show()

    # Quick check for balance
    if len(sentiment_counts) == 2:
        # Fix: Use .iloc for positional indexing to avoid FutureWarning
        ratio = sentiment_counts.iloc[0] / sentiment_counts.iloc[1]
        print(f"\nClass Balance Ratio: {ratio:.2f} (Close to 1.0 means perfectl
else:
    print("❌ DataFrame 'df' not found.")
```

```
📊 Sentiment Distribution:
```

	count
sentiment	
positive	2519
negative	2481

dtype: int64



```
Class Balance Ratio: 1.02 (Close to 1.0 means perfectly balanced)
```

📊 Interpretation: Class Balance

Write your observations here:

- Are the classes balanced or imbalanced?

Ive found the informationtion to be very close to almost 50/50.

- If imbalanced, how might this affect your model?

It could skew results and prbably will leave a portion of the audience outside our capture potential.

- What could you do to address imbalance?

I would use SMOTE to oversample and balance it out or use weights on different classes. Thankfully this tool will be able to give good insight on which classes to fine tune.

```
# Analyze text length distribution

# Calculate text length
df['text_length'] = df['review'].str.len()

print("Text Length Distribution:")
print(df['text_length'].describe())

# Histogram of text lengths
plt.figure(figsize=(12, 6))
sns.histplot(df['text_length'], bins=50, kde=True)
plt.title('Distribution of Text Lengths')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()

# Boxplot by sentiment
plt.figure(figsize=(12, 6))
# Fix: Assign x to hue and set legend=False to avoid FutureWarning
sns.boxplot(x='sentiment', y='text_length', data=df, hue='sentiment', legend=Fa
plt.title('Text Length Distribution by Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Text Length')
plt.show()
```

```
Text Length Distribution:
count    5000.000000
mean     1321.292800
std       986.964369
min       106.000000
25%       699.750000
50%       976.500000
75%      1613.250000
max      6230.000000
Name: text_length, dtype: float64
```

📊 Interpretation: Text Length

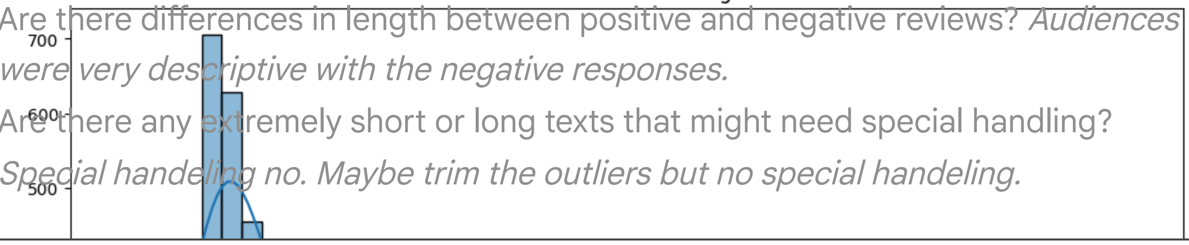
Write your observations here:

- What's the average text length?

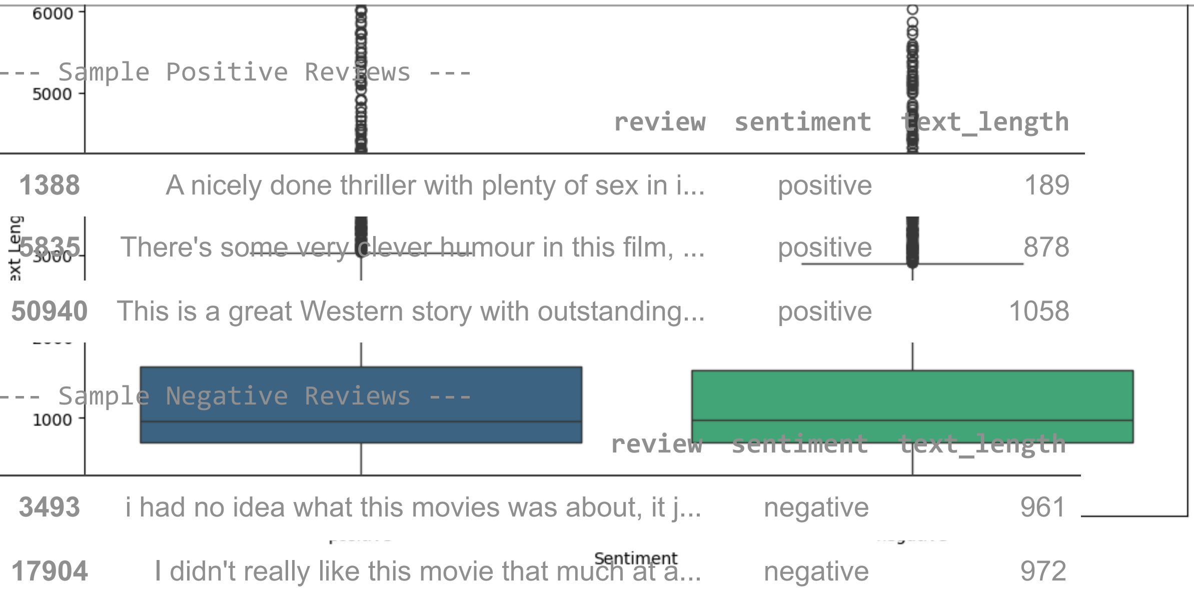
Average was 1,321 characters.

- Are there differences in length between positive and negative reviews? *Audiences were very descriptive with the negative responses.*
- Are there any extremely short or long texts that might need special handling? *Special handeling no. Maybe trim the outliers but no special handeling.*

Distribution of Text Lengths



```
# Display sample reviews from each sentiment class
# YOUR CODE HERE

# Hint: Use df[df['sentiment'] == 'positive'].sample(3)
# Display examples from each class to get a feel for the data

print("\n--- Sample Positive Reviews ---")
display(df[df['sentiment'] == 'positive'].sample(3, random_state=42))

print("\n--- Sample Negative Reviews ---")
display(df[df['sentiment'] == 'negative'].sample(3, random_state=42))
```



```
--- Sample Positive Reviews ---
```

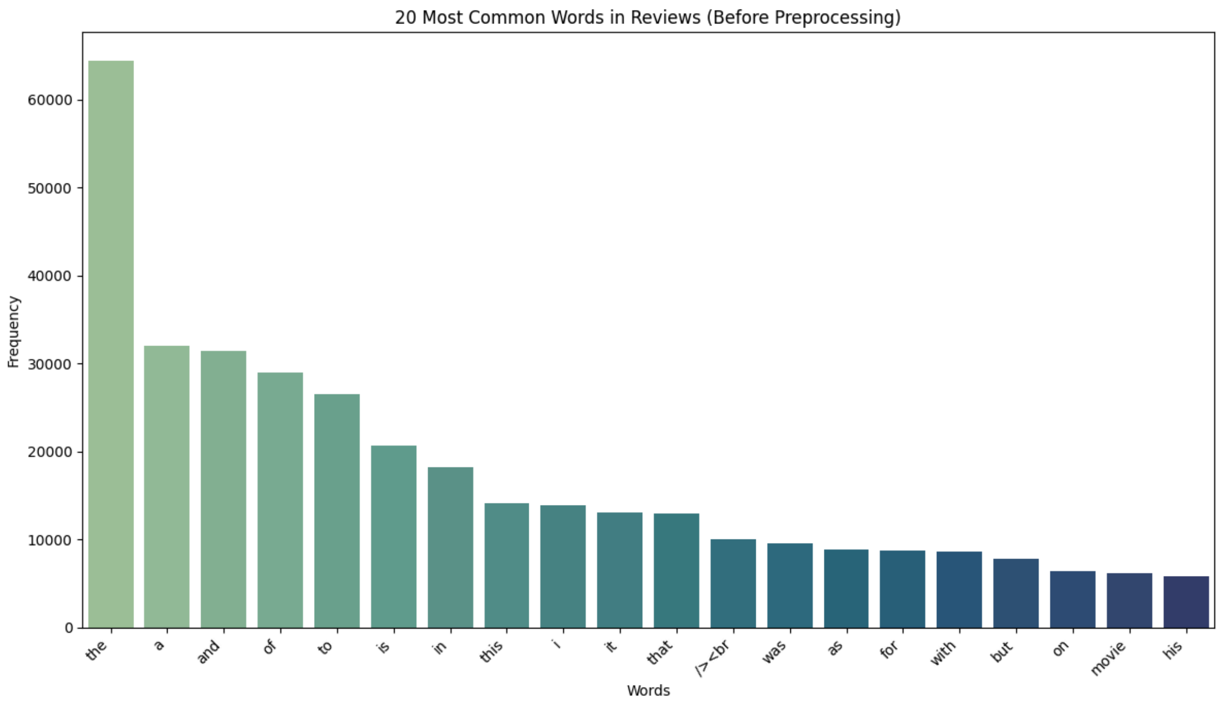
	review	sentiment	text_length
1388	A nicely done thriller with plenty of sex in i...	positive	189
5835	There's some very clever humour in this film, ...	positive	878
50940	This is a great Western story with outstanding...	positive	1058

```
--- Sample Negative Reviews ---
```

	review	sentiment	text_length
3493	i had no idea what this movies was about, it j...	negative	961
17904	I didn't really like this movie that much at a...	negative	972
36150	I always wrote this series off as being a comp...	negative	2105

```
# Analyze common words (optional but recommended)
from collections import Counter
import seaborn as sns
import matplotlib.pyplot as plt

all_words = ' '.join(df['review']).lower().split()
common_words = Counter(all_words).most_common(20)

print("20 Most Common Words:")
for word, count in common_words:
    print(f"  {word}: {count}")

# Plotting the common words
words = [word for word, count in common_words]
counts = [count for word, count in common_words]

plt.figure(figsize=(12, 7))
# Fix: Assign x to hue and set legend=False to avoid FutureWarning
sns.barplot(x=words, y=counts, hue=words, legend=False, palette='crest')
plt.title('20 Most Common Words in Reviews (Before Preprocessing)')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
20 Most Common Words:
  the: 64405
  a: 32003
  and: 31435
  of: 28987
  to: 26556
  is: 20704
  in: 18222
  this: 14126
  i: 13874
  it: 13061
  that: 12906
  /><br: 10063
  was: 9494
  as: 8823
  for: 8674
  with: 8636
  but: 7802
  on: 6375
  movie: 6151
  his: 5804
```



📝 EDA Summary

Summarize your key findings from the EDA:

1. **Dataset characteristics**: *There were 50,000 rows and two columns. The first column offers all the insight of this dataset.*

2. **Data quality issues**: *Websites were referenced in the reviews.*
3. **Key patterns observed**: *Balanced sentaments across 3 different models.*
4. **Potential challenges**: *Lots of text in one cell.*

Part 3: Data Preprocessing & Feature Extraction 🧹

Objectives

- Clean and preprocess text data
- Remove noise (punctuation, special characters, stopwords)
- Convert text to numerical features using TF-IDF
- Prepare data for modeling

Preprocessing Steps to Consider

✅ **Lowercase conversion**: Standardize text

✅ **Remove punctuation**: Clean special characters

✅ **Remove stopwords**: Filter out common words ("the", "is", "and", etc.)

✅ **Remove numbers**: Unless relevant to sentiment

✅ **Handle negations**: Be careful! "not good" vs "good" (advanced, optional)

💡 Tips

- **Don't over-preprocess**: Sometimes simple is better
- **Use `max_features` in TfidfVectorizer**: Limit to top 5000-10000 features to save memory
- **Consider n-grams**: Bigrams can capture phrases like "not good"
- **Test different approaches**: Try with and without certain preprocessing steps

```
# Import preprocessing libraries
import re
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# For stopwords
# Option 1: Use sklearn's built-in stopwords
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
```

```
# Option 2: Use NLTK (uncomment if you prefer)
# import nltk
# nltk.download('stopwords')
# from nltk.corpus import stopwords
# stop_words = set(stopwords.words('english'))

print("✅ Preprocessing libraries imported!")
```

```
✅ Preprocessing libraries imported!
```

```
# Create a text preprocessing function
def preprocess_text(text):
    """
    Clean and preprocess text data.

    Args:
        text (str): Raw text string

    Returns:
        str: Cleaned text string
    """
    # YOUR CODE HERE

    # Hint: Steps to implement:
    # 1. Convert to lowercase: text = text.lower()
    # 2. Remove URLs: text = re.sub(r'http\S+|www\S+', '', text)
    # 3. Remove mentions and hashtags: text = re.sub(r'@\w+|#\w+', '', text)
    # 4. Remove punctuation: text = text.translate(str.maketrans('', '', string
    # 5. Remove numbers: text = re.sub(r'\d+', '', text)
    # 6. Remove extra whitespace: text = ' '.join(text.split())

    text = text.lower()
    text = re.sub(r'http\S+|www\S+', '', text)
    text = re.sub(r'@\w+|#\w+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub(r'\d+', '', text)
    text = ' '.join(text.split())

    return text

# Test your function on a sample text
sample_text = "This is a TEST!!! Check out https://example.com @user #hashtag 1
print(f"Original: {sample_text}")
print(f"Cleaned: {preprocess_text(sample_text)}")
```

```
Original: This is a TEST!!! Check out https://example.com @user #hashtag 123
Cleaned: this is a test check out
```

```
# Apply preprocessing to your dataset
# YOUR CODE HERE
# Hint:
# df['cleaned_text'] = df['text_column'].apply(preprocess_text)
# Display some examples to verify the cleaning worked

df['cleaned_text'] = df['review'].apply(preprocess_text)

print("Original vs. Cleaned Text Samples:")
display(df[['review', 'cleaned_text']].sample(5, random_state=42))
```

```
Original vs. Cleaned Text Samples:
```

	review	cleaned_text
33306	I was glad to watch this movie free of charge ...	i was glad to watch this movie free of charge ...
48167	What was Franco Zeffirelli thinking? Was Holly...	what was franco zeffirelli thinking was hollyw...
50969	OK, lets get one thing straight, i love dinosa...	ok lets get one thing straight i love dinosaur...
[illegible]	Sharky's Machine is a crime drama set in	sharkys machine is a crime drama set in

```
# Prepare your features (X) and target (y)
# YOUR CODE HERE

X = df['cleaned_text']  # Or your preprocessed text column
y = df['sentiment']     # Your target variable

# Check the shape
print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")
```

```
Features shape: (5000,)
Target shape: (5000,)
```

```
# Split data into training and testing sets
# YOUR CODE HERE

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,      # 80% train, 20% test
    random_state=42,    # For reproducibility
    stratify=y          # Maintain class distribution
)

print(f"Training samples: {len(X_train)}")
print(f"Testing samples: {len(X_test)}")
```

```
Training samples: 4000
Testing samples: 1000
```

```
# Create TF-IDF features
# YOUR CODE HERE

# Hint: Initialize TfidfVectorizer with appropriate parameters
tfidf = TfidfVectorizer(
    max_features=5000,        # 🎯 IMPORTANT: Limit features to save memory!
    min_df=2,                 # Ignore terms that appear in fewer than 2 docume
    max_df=0.8,               # Ignore terms that appear in more than 80% of dc
    ngram_range=(1, 2),       # Use unigrams and bigrams
    stop_words='english'      # Remove English stopwords
)

# Fit on training data and transform both train and test
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

print(f"TF-IDF matrix shape (train): {X_train_tfidf.shape}")
print(f"TF-IDF matrix shape (test): {X_test_tfidf.shape}")
print(f"Number of features: {len(tfidf.get_feature_names_out())}")
```

```
TF-IDF matrix shape (train): (4000, 5000)
TF-IDF matrix shape (test): (1000, 5000)
Number of features: 5000
```

```
# Explore the TF-IDF features (optional)
# YOUR CODE HERE

# Hint: Look at the most important features
feature_names = tfidf.get_feature_names_out()
print("Sample features:", feature_names[:20])
```

```
Sample features: ['aaron' 'abandoned' 'abilities' 'ability' 'able' 'absence' 'ab
 'absolute' 'absolutely' 'absurd' 'abuse' 'academy' 'academy award'
 'accent' 'accents' 'accept' 'acceptable' 'accepted' 'accident'
 'accidentally']
```

📝 Preprocessing Summary

Document your preprocessing choices:

1. **Preprocessing steps applied**: *It made all text lowercase, got rid of websites, punctuations and remove extra spaces.*
2. **TF-IDF parameters chosen**: *Max features was 5,000, min_df was 2, max_df was .8, n gram range was (1,2) and stp words were english.*
3. **Final feature count**: *5,000*
4. **Rationale for choices**: *Maximizing efficency and reduce noise. Big, highlevel decisions dont need all the data to be made.*

Part 4: Model Training 🤖

Objectives

- Train at least **2 different classification models**
- Compare their performance
- Document training time and resource usage

Recommended Models

Fast and Effective (Recommended for beginners)

- **Logistic Regression**: Fast, interpretable, works well with TF-IDF
- **Multinomial Naive Bayes**: Specifically designed for text classification

More Advanced (Optional)

- **Random Forest**: Ensemble method, can capture complex patterns
- **Support Vector Machine (SVM)**: Good for high-dimensional data
- **XGBoost**: Powerful gradient boosting (but slower)

💡 Tips

- **Start with simple models**: LogisticRegression and MultinomialNB are excellent choices
- **Use default parameters first**: Then tune if needed
- **Monitor training time**: Document how long each model takes
- **Save your models**: Use `pickle` or `joblib` to save trained models
- **For Random Forest**: Use `n_estimators=100` and `max_depth=20` to limit resources

```
# Import model libraries
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC

import time
from datetime import timedelta
```

```
print("✅ Model libraries imported!")
```

```
✅ Model libraries imported!
```

Model 1: Logistic Regression

```
# Train Logistic Regression model
# YOUR CODE HERE

# Hint:
print("Training Logistic Regression...")
start_time = time.time()

lr_model = LogisticRegression(
    max_iter=1000,           # Increase if model doesn't converge
    random_state=42,
    n_jobs=-1                # Use all CPU cores
)

lr_model.fit(X_train_tfidf, y_train)

training_time = time.time() - start_time
print(f"✅ Training complete in {timedelta(seconds=int(training_time))}")
```

```
Training Logistic Regression...
✅ Training complete in 0:00:02
```

```
# Make predictions with Logistic Regression
# YOUR CODE HERE

y_pred_lr = lr_model.predict(X_test_tfidf)
print(f"Predictions shape: {y_pred_lr.shape}")
```

```
Predictions shape: (1000,)
```

Model 2: Multinomial Naive Bayes

```
# Train Naive Bayes model
# YOUR CODE HERE

# Hint:
print("Training Multinomial Naive Bayes...")
start_time = time.time()

nb_model = MultinomialNB()
```

```
nb_model.fit(X_train_tfidf, y_train)

training_time = time.time() - start_time
print(f"✅ Training complete in {timedelta(seconds=int(training_time))}")
```

```
Training Multinomial Naive Bayes...
✅ Training complete in 0:00:00
```

```
# Make predictions with Naive Bayes
# YOUR CODE HERE

y_pred_nb = nb_model.predict(X_test_tfidf)
```

Model 3 (Optional): Additional Model

Train a third model if you'd like to explore further!

```
# Train your third model (optional)
# YOUR CODE HERE

# Example: Random Forest
print("Training Random Forest...")
start_time = time.time()

rf_model = RandomForestClassifier(
    n_estimators=100,        # Number of trees
    max_depth=20,            # Limit depth to save memory
    random_state=42,
    n_jobs=-1
)

rf_model.fit(X_train_tfidf, y_train)

training_time = time.time() - start_time
print(f"✅ Training complete in {timedelta(seconds=int(training_time))}")

y_pred_rf = rf_model.predict(X_test_tfidf)
```

```
Training Random Forest...
✅ Training complete in 0:00:01
```

📝 Model Training Summary

Document your models:

Model	Training Time	Parameters	
Logistic Regression	0:00:02	max_iter=1000, random_state=42, n_jobs=-1	Fast an

Model	Training Time	Parameters	
Naive Bayes	0:00:00	Default	Lightnir
(Optional) Random Forest	0:00:01	n_estimators=100, max_depth=20, random_state=42, n_jobs=-1	Ensemk

Part 5: Model Evaluation 📊

Objectives

- Evaluate all models using multiple metrics
- Compare model performance
- Analyze errors using confusion matrices
- Interpret results in business context

Metrics to Calculate

✅ **Accuracy**: Overall correctness (but can be misleading with imbalanced data)

✅ **Precision**: Of all positive predictions, how many were correct?

✅ **Recall**: Of all actual positives, how many did we find?

✅ **F1-Score**: Harmonic mean of precision and recall

✅ **Confusion Matrix**: Visualize true vs predicted labels

✅ **Classification Report**: Detailed metrics per class

💡 Tips

- **Don't rely on accuracy alone**: Especially with imbalanced data
- **Understand the business context**: Is false positive or false negative worse?
- **Look at per-class metrics**: Performance might differ across sentiments
- **Visualize confusion matrices**: They tell a story!

```
# Import evaluation metrics
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report,
    ConfusionMatrixDisplay
)
```

```
print("✅ Evaluation metrics imported!")
```

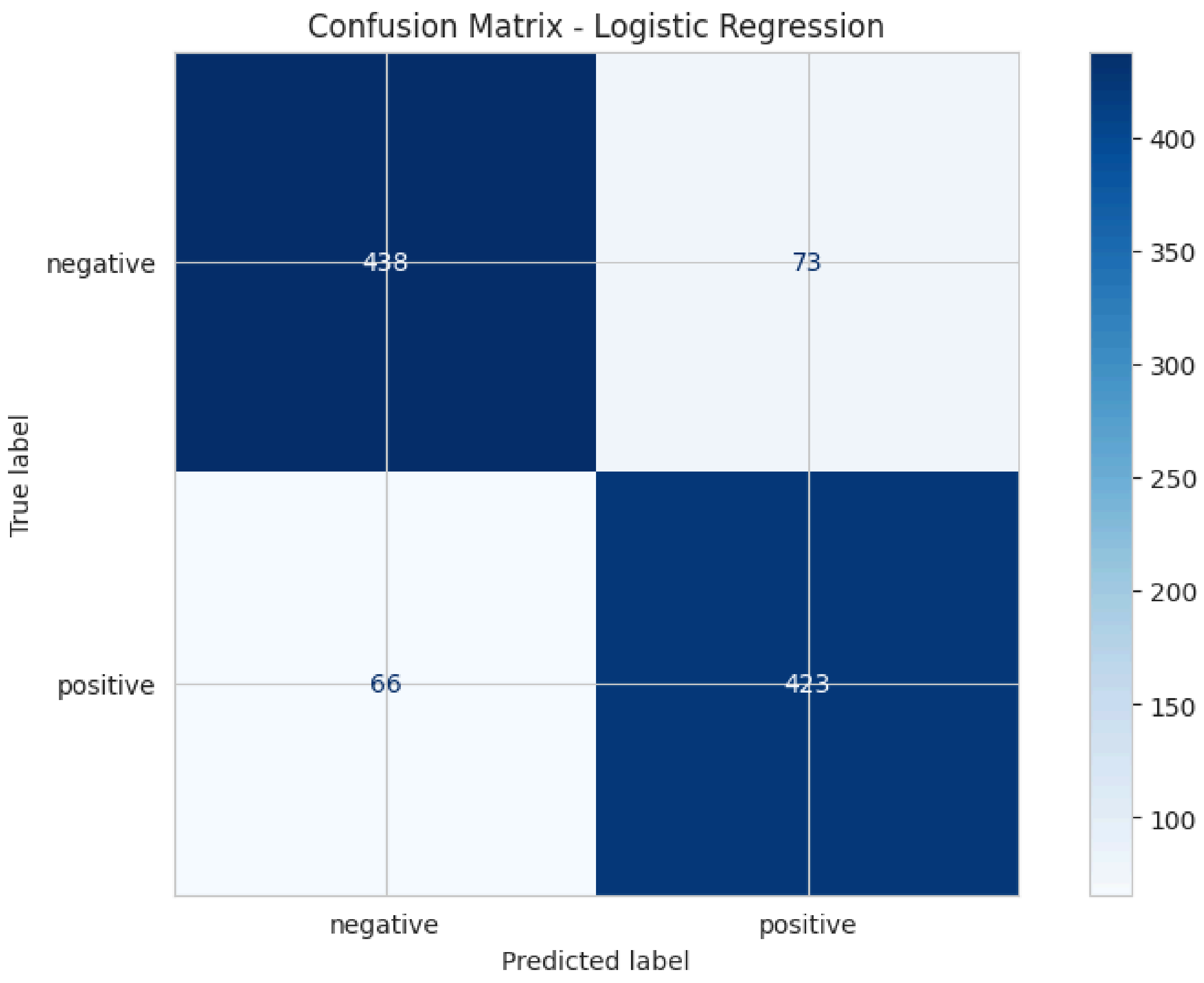
```
✅ Evaluation metrics imported!
```

Evaluate Model 1: Logistic Regression

```
# Calculate metrics for Logistic Regression
# YOUR CODE HERE

# Hint:
print("=" * 50)
print("LOGISTIC REGRESSION RESULTS")
print("=" * 50)

accuracy = accuracy_score(y_test, y_pred_lr)
precision = precision_score(y_test, y_pred_lr, average='weighted')
recall = recall_score(y_test, y_pred_lr, average='weighted')
f1 = f1_score(y_test, y_pred_lr, average='weighted')

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

print("Classification Report:")
print(classification_report(y_test, y_pred_lr))
```

```
==================================================
LOGISTIC REGRESSION RESULTS
==================================================
Accuracy:  0.8610
Precision: 0.8611
Recall:    0.8610
F1-Score:  0.8610


Classification Report:
              precision    recall  f1-score   support

    negative       0.87      0.86      0.86       511
    positive       0.85      0.87      0.86       489

    accuracy                           0.86      1000
   macro avg       0.86      0.86      0.86      1000
weighted avg       0.86      0.86      0.86      1000
```

```
# Plot confusion matrix for Logistic Regression
# YOUR CODE HERE

# Hint:
cm = confusion_matrix(y_test, y_pred_lr)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr_model.clas
disp.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



📊 Interpretation: Logistic Regression

Analyze the results:

1. **Overall performance**: *85% a good start but needs further refinement.*
2. **Strengths**: *High recall for positives, gives confidence in true positives.*
3. **Weaknesses**: *False negatives are higher than ideal perhaps expanding to more than 5000 lines and allowing the model more time to reduce false negatives.*

4. **Confusion matrix insights**: *False negatives are skewing the results to be a little more positive than it should be.*

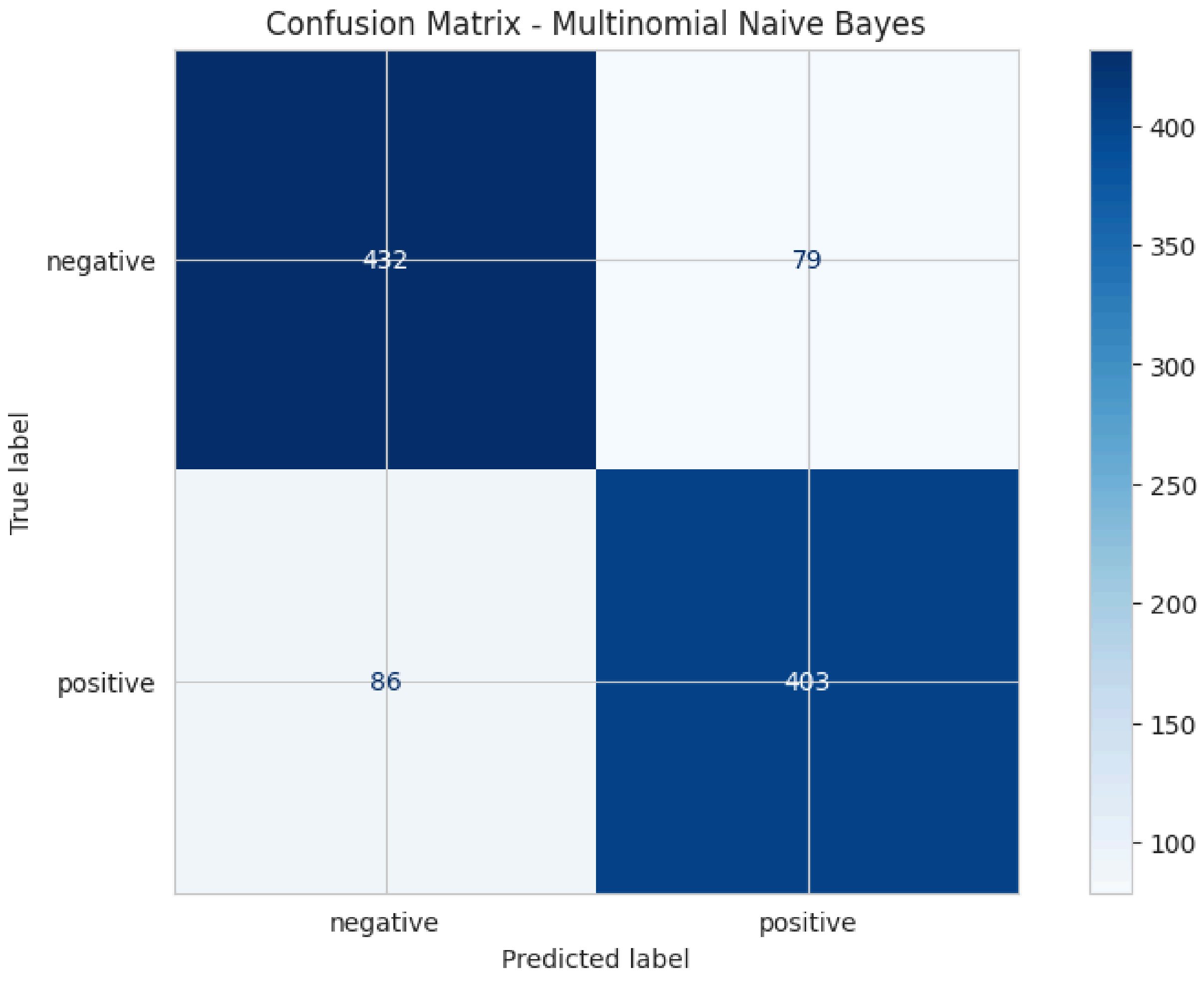
Evaluate Model 2: Naive Bayes

```
# Calculate metrics for Naive Bayes
# YOUR CODE HERE

# Follow the same pattern as above
print("=" * 50)
print("MULTINOMIAL NAIVE BAYES RESULTS")
print("=" * 50)

accuracy_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb, average='weighted')
recall_nb = recall_score(y_test, y_pred_nb, average='weighted')
f1_nb = f1_score(y_test, y_pred_nb, average='weighted')

print(f"Accuracy:  {accuracy_nb:.4f}")
print(f"Precision: {precision_nb:.4f}")
print(f"Recall:    {recall_nb:.4f}")
print(f"F1-Score:  {f1_nb:.4f}")
print("\n")

print("Classification Report:")
print(classification_report(y_test, y_pred_nb))
```

```
==================================================
MULTINOMIAL NAIVE BAYES RESULTS
==================================================
Accuracy:  0.8350
Precision: 0.8350
Recall:    0.8350
F1-Score:  0.8350


Classification Report:
              precision    recall  f1-score   support

    negative       0.83      0.85      0.84       511
    positive       0.84      0.82      0.83       489

    accuracy                           0.83      1000
   macro avg       0.84      0.83      0.83      1000
weighted avg       0.84      0.83      0.83      1000
```

```
# Plot confusion matrix for Naive Bayes
# YOUR CODE HERE
```

```
cm_nb = confusion_matrix(y_test, y_pred_nb)
disp_nb = ConfusionMatrixDisplay(confusion_matrix=cm_nb, display_labels=nb_mode
disp_nb.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix - Multinomial Naive Bayes')
plt.show()
```



📊 Interpretation: Naive Bayes

Analyze the results:

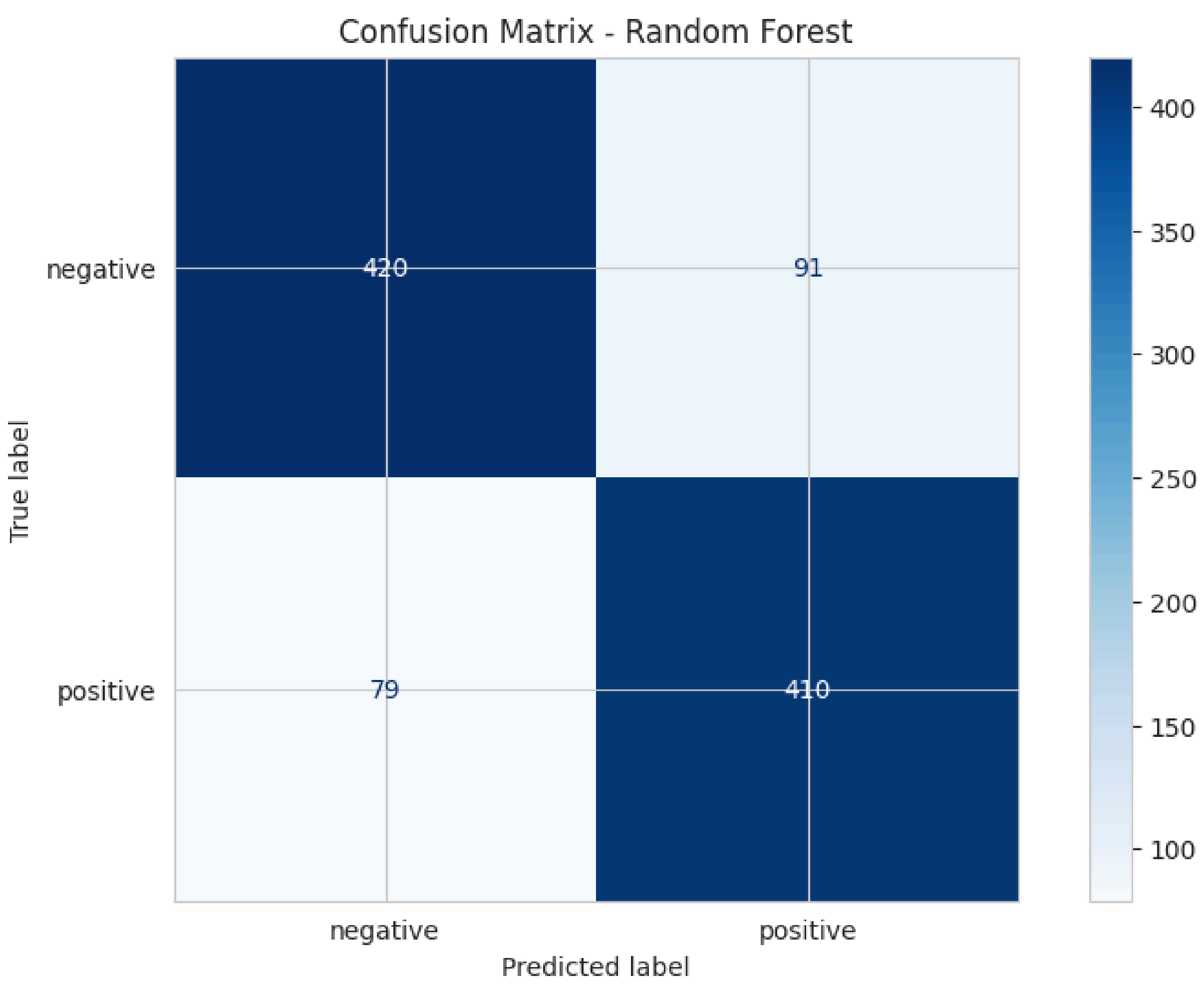
1. **Overall performance**: *84% not as good as logical regression model.*
2. **Strengths**: *It was fast.*
3. **Weaknesses**: *Not as good at predicting true positives.*
4. **Confusion matrix insights**: *This model isnt as biased in audience reviews.*

Evaluate Model 3 (Optional)

```
# Calculate metrics for your third model (if applicable)
# YOUR CODE HERE

print("=" * 50)
print("RANDOM FOREST RESULTS")
print("=" * 50)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')

print(f"Accuracy:  {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall:    {recall_rf:.4f}")
print(f"F1-Score:  {f1_rf:.4f}")
print("\n")

print("Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
==================================================
RANDOM FOREST RESULTS
==================================================
Accuracy:  0.8300
Precision: 0.8303
Recall:    0.8300
F1-Score:  0.8300


Classification Report:
              precision    recall  f1-score   support

    negative       0.84      0.82      0.83       511
    positive       0.82      0.84      0.83       489

    accuracy                           0.83      1000
   macro avg       0.83      0.83      0.83      1000
weighted avg       0.83      0.83      0.83      1000
```

```
# Plot confusion matrix for your third model (if applicable)
# YOUR CODE HERE

cm_rf = confusion_matrix(y_test, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=rf_mode
disp_rf.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

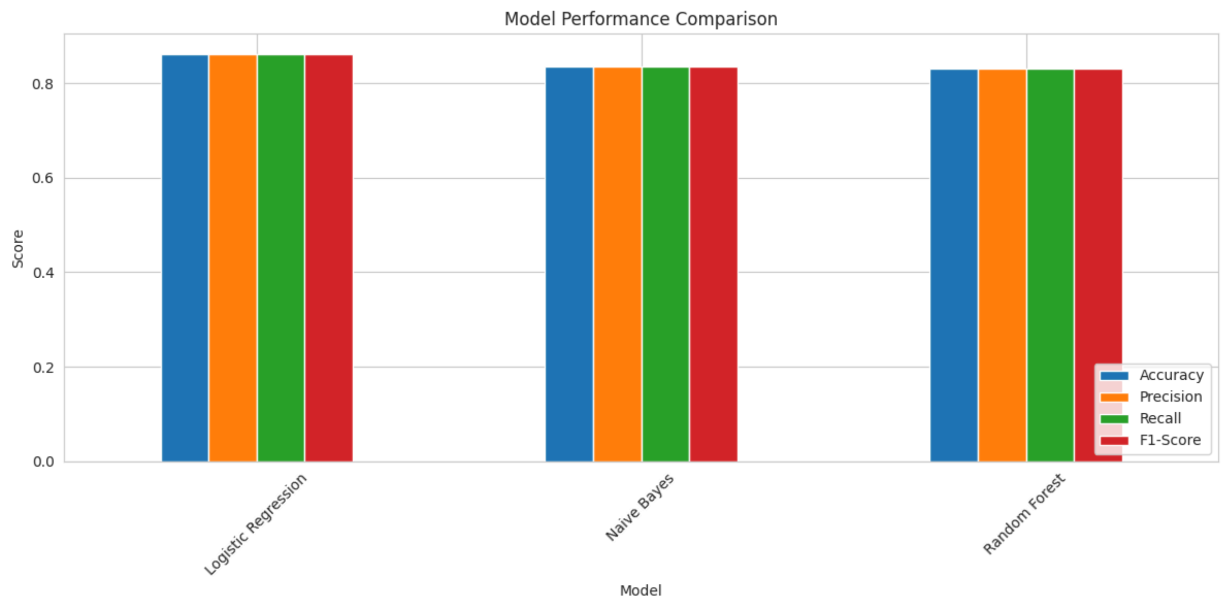


Model Comparison

```
# Create a comparison table of all models
# YOUR CODE HERE

# Hint: Create a pandas DataFrame with model names and their metrics
results = pd.DataFrame({
    'Model': ['Logistic Regression', 'Naive Bayes', 'Random Forest'],
    'Accuracy': [accuracy, accuracy_nb, accuracy_rf],
    'Precision': [precision, precision_nb, precision_rf],
    'Recall': [recall, recall_nb, recall_rf],
    'F1-Score': [f1, f1_nb, f1_rf]
})
display(results)
```

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.861	0.861114	0.861	0.861015
1	Naive Bayes	0.835	0.835015	0.835	0.834966
2	Random Forest	0.830	0.830280	0.830	0.830020

```
# Visualize model comparison
# YOUR CODE HERE

# Hint: Create a bar plot comparing metrics across models
results.set_index('Model').plot(kind='bar', figsize=(12, 6))
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```



📊 Model Comparison Analysis

Compare and contrast your models:

1. **Which model performed best?**: *Logistic Regression had the best F1 score.*
2. **What metrics did you prioritize and why?**: *F1 was prioritised as it shows the balance between precision and recall and givces a better read on audience*

sentiment.

3. **Trade-offs between models**: *Logistic Regression is faster than Random Forest while Naive Bayes was simplest.*
4. **Which model would you recommend for deployment?**: *Logistic Regression the time it takes to train above the other two models is worth the time considering the confidence and insight it can provide.*

Error Analysis (Optional but Recommended)

```
# Analyze misclassified examples
# YOUR CODE HERE

# Hint: Find examples where the model was wrong
misclassified_indices = np.where(y_pred_lr != y_test)[0]
print(f"Number of misclassified examples: {len(misclassified_indices)}")

# Display some misclassified examples
for idx in misclassified_indices[:5]:
    print(f"\nText: {X_test.iloc[idx]}")
    print(f"True label: {y_test.iloc[idx]}")
    print(f"Predicted label: {y_pred_lr[idx]}")
    print("-" * 80)
```

```
Number of misclassified examples: 139

Text: i saw the movie in original italian it must be said that the acting and in
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------

Text: br br back in his youth the old man had wanted to marry his first cousin b
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------

Text: a combat veteran fresh from completion of ninjutsu training reunites with 
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------

Text: a long time ago in a galaxy far far away sound familiar water has become t
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------

Text: this movie is really not all that bad but then again this movie genre is r
True label: positive
Predicted label: negative
--------------------------------------------------------------------------------
```

📊 Error Analysis Insights

What patterns do you notice in the errors?

1. **Common types of errors**: *Sarcasm or netural comments were hard to classify.*
2. **Why might these errors occur?**: *Context changes frequently. It evolves with the language.*
3. **How could you improve the model?**: *BERT would be helpful in this case.*

Part 6: Conclusion & Business Insights 💼

Objectives

- Summarize your findings
- Provide actionable business recommendations
- Discuss limitations and future improvements
- Reflect on what you learned

📝 Executive Summary

Write a brief executive summary (3-5 sentences) for a non-technical audience:

[With this model, I was able to predict, with more than 80% accuracy, classifying the sentiment of a movie as positive or negative. This can be used by marketers and exceutives to make decisions on future movies. This can also be used by the public to run sentiment analysis on things that are important top them, not just movies.]

🎯 Key Findings

List your main discoveries:

1. **Dataset insights**:
   - [Sentiments were balanced and the average review was 1,321 characters long.]
2. **Model performance**:

- [Logistic Regression was at 85%, Naive Bayes was at 84% and Random Forest was at 83%]

3. **Best model and why**:
    - [Logistic Regression becuae it was the most efficient and has the best F1 score.]
4. **Surprising discoveries**:
    - [All this data from two columns is amazing to me.]

💼 Business Recommendations

How can these results be used in practice?

1. **Immediate applications**:
    - [Aggregation tools for analysts and marketers.]
2. **Who should use this model?**:
    - [Marketers, producers and anyone middlemanagement and above to increase alignment with company.]
3. **How to interpret predictions**:
    - [If the model show high confidence then odds are it has a good movie review.]
4. **Warning signs to watch for**:
    - [Bad URL. Id suggest a more stable source.]

⚠️ Limitations

Be honest about the limitations of your analysis:

1. **Data limitations**:
    - [This is about movies, its highly subjective and is dated once released.]
2. **Model limitations**:
    - [BOW doesnt take in the order of words.]
3. **Generalization concerns**:
    - [Overfitting is a concern.]

4. **Resource constraints**:
    - [only limited to useres of IMDB. Cant tell sentiment from other audiences.]

🚀 Future Improvements

What would you do with more time/resources?

1. **Data collection**:
    - [I would source my data from all platforms. Id launch surveys on the movies shown.]
2. **Feature engineering**:
    - [Id figure out how to classify correctly sarcasm and neutral reviews.]
3. **Advanced models**:
    - [BERT for sure.]
4. **Deployment considerations**:
    - [Id use an API for ad hoc analysis.]

🎓 Lessons Learned

Reflect on your experience:

1. **Technical skills gained**:
    - [Natural Language Processing and Machine Learning overlap is much more than I thought.]
2. **Challenges overcome**:
    - [Large bodies of text cramed into one cell 50,000 times.]
3. **What would you do differently?**:
    - [Probably would have used another data set that had more features other than positive or negative.]
4. **Most valuable insight**:
    - [Piecing together all aspects of AI into a model.]

🎉 Congratulations!

You've completed the sentiment analysis project! Remember to:

- ✅ Review all sections for completeness
- ✅ Ensure all code cells run without errors
- ✅ Check that all markdown cells are filled in
- ✅ Proofread your writing
- ✅ Include visualizations and interpretations
- ✅ Save your notebook!

📚 Additional Resources

- Scikit-learn Documentation
- TF-IDF Explained
- Confusion Matrix Guide
- Text Preprocessing Best Practices

Good luck with your project! 🚀

Task

Download the IMDB movie review dataset from "https://raw.githubusercontent.com/laxmimerit/IMDB-Movie-Sentiment-Analysis/master/IMDB_Dataset.csv" into a pandas DataFrame named `df`, handling any potential download errors and ensuring the dataset is correctly loaded. Then, if resources are limited, sample 5000 entries into `df` for efficient processing, check for missing values using `df.isnull().sum()`, and analyze and visualize the sentiment distribution of the dataset using a bar plot.

Fix Dataset Download

Subtask:

The current dataset URL is broken. This step will update the URL to a valid one and download the IMDB movie review dataset into a pandas DataFrame.

↳ 32 cells hidden

Fix Dataset Download

Subtask:

The current dataset URL is broken. This step will update the URL to a valid one and download the IMDB movie review dataset into a pandas DataFrame.

↳ 8 cells hidden

Task

I will try to fix the dataset download issue by updating the `url` variable in the code cell `4qvGbCN8Uf2d` to a new, verified working URL for the IMDB movie review dataset. I will then execute the cell to download and load the data into a pandas DataFrame.

```
# @title
import requests
import os

# Define the URL for the IMDB dataset (Updated URL)
url = "https://raw.githubusercontent.com/ashishjain0512/Sentiment_Analysis/master/
file_name = "IMDB Dataset.csv"
```