📊 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?
    - Who would use these results?
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:

Movie studios, producers, and streaming platforms often struggle to gauge public reception and identify factors driving positive or negative buzz around their films. This sentiment analysis aims to provide quantifiable insights into audience reactions from movie reviews. This will help inform strategic decisions related to future movie production, marketing campaigns, and even script development.

Dataset Description:

- **Dataset name**: IMDB Dataset of 50k movie reviews
- **Source**: Kaggle / [Original Source, e.g., Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).]
- **Number of samples**: 50,000
- **Text type**: MOvie Reviews written by users on IMDB
- **Sentiment labels**: Binary(positives or negatives)

Importance and Real-World Applications:

Understanding audience sentiment is crucial for data-driven decision-making in the entertainment industry. The results of this analysis could be used by:

- **Movie Production Companies**: To understand audience preferences and refine future film concepts or identify areas for improvement in ongoing projects.
- **Marketing Teams**: To craft more effective campaigns by highlighting aspects of a movie that generate positive sentiment and addressing potential concerns.
- **Streaming Platforms**: To improve content recommendation algorithms, optimize content acquisition, and understand why certain films resonate more with their subscribers.
- **Film Critics and Analysts**: To provide a quantitative backing for their qualitative reviews and market analyses.

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)

print("✅ Libraries imported successfully!")

# Load your dataset
# Example: df = pd.read_csv('your_dataset.csv')
# For large datasets, consider using nrows parameter: pd.read_csv('file.csv', nrows=1000

# YOUR CODE HERE
df = pd.read_csv('/content/IMDB Dataset.csv')

# Display basic information
print(f"Dataset shape: {df.shape}")
print(f"\nColumn names: {df.columns.tolist()}")
display(df.head())
```

```
✅ Libraries imported successfully!
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. 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

```
# Check for missing values
# YOUR CODE HERE
```

```
df.isnull().sum()

# Hint: Use df.isnull().sum() or df.info()
```

	0
review	0
sentiment	0

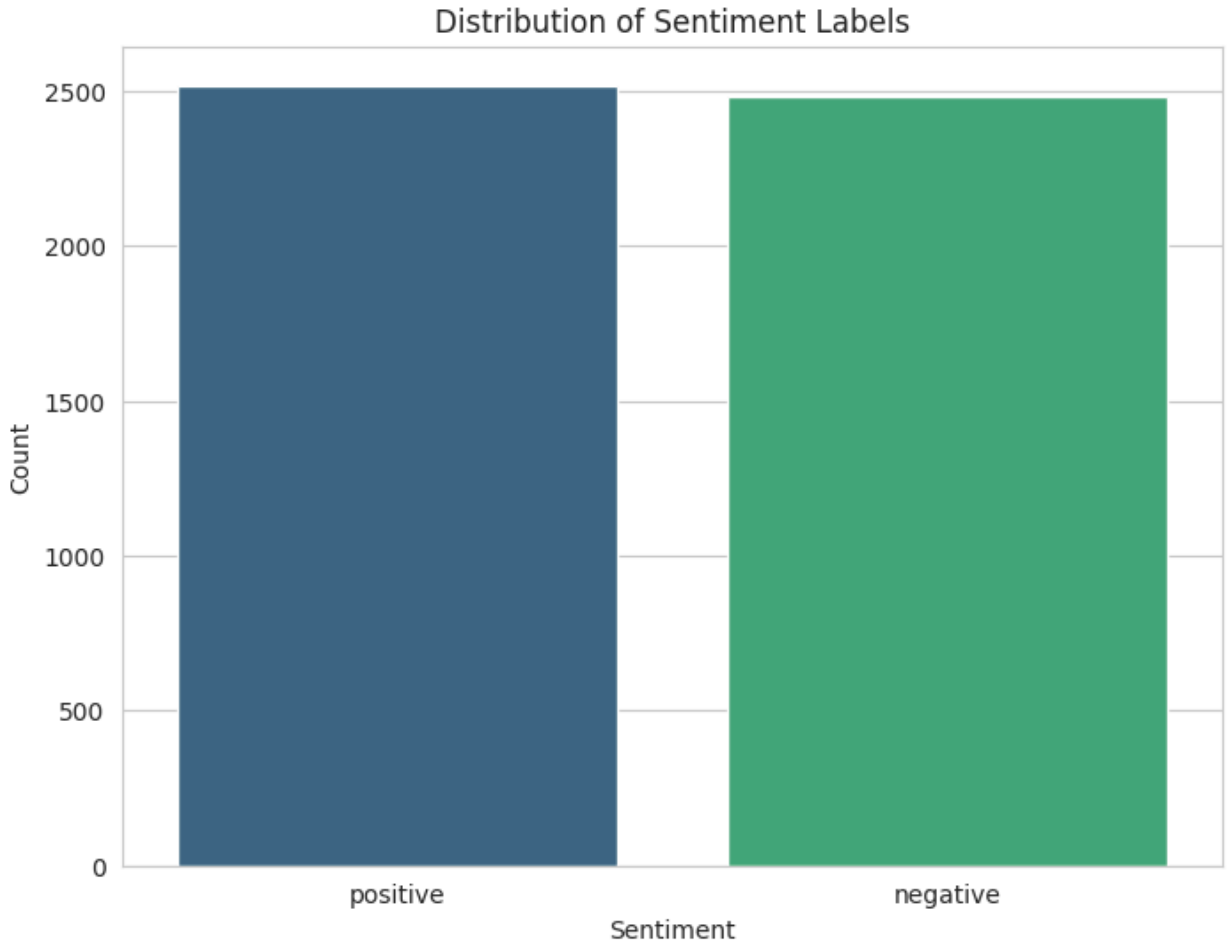
dtype: int64

```
# 🎯 IMPORTANT: If working with limited resources, sample your data here
# Uncomment and modify as needed:

df_sample = df.sample(n=5000, random_state=42)
print(f"Working with {len(df_sample)} samples")
df = df_sample

# Use the sample for the rest of the project
```

```
Working with 5000 samples
```

```
# Analyze sentiment distribution
# YOUR CODE HERE

sentiment_counts = df['sentiment'].value_counts()
print("Sentiment Distribution:")
print(sentiment_counts)

plt.figure(figsize=(8, 6))
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values, palette='viridis')
plt.title('Distribution of Sentiment Labels')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Hint: Use df['sentiment_column'].value_counts()
# Create a bar plot to visualize the distribution
```

```
Sentiment Distribution:
sentiment
positive    2519
negative    2481
Name: count, dtype: int64
```



📊 Interpretation: Class Balance

Write your observations here:

- Are the classes balanced or imbalanced?

What I noted was that the sentiment classes are fairly balanced. The sampled dataset contains 2519 positive reviews and 2481 negative reviews.

- If imbalanced, how might this affect your model?

Not appliable since the data is balanced.

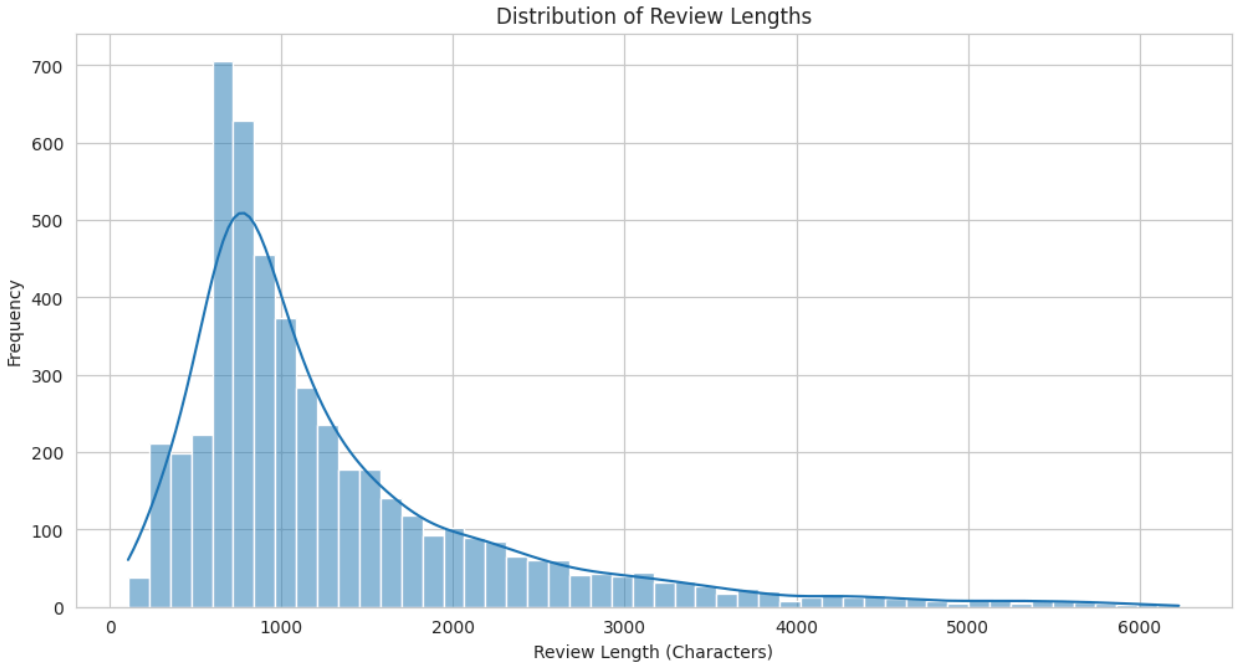
- What could you do to address imbalance?

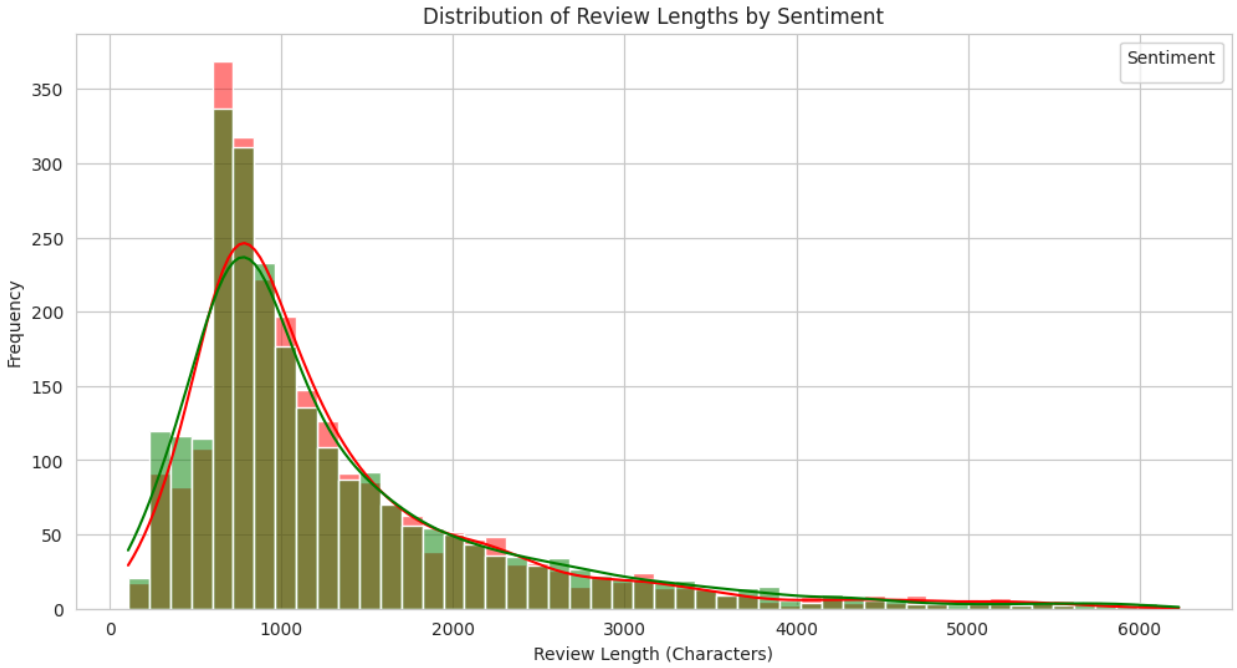
The balanced distribution is ideal, as it means our models won't be inherently biased towards one class due to skewed training data.

```
# Analyze text length distribution
# Hint: Create a new column for text length
```

```
# df['text_length'] = df['text_column'].str.len()
# Plot histogram of text lengths
# Compare lengths across different sentiments
# YOUR CODE HERE


df['text_length'] = df['review'].apply(len)

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

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

plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='text_length', hue='sentiment', bins=50, kde=True, palette={'pos
plt.title('Distribution of Review Lengths by Sentiment')
plt.xlabel('Review Length (Characters)')
plt.ylabel('Frequency')
plt.legend(title='Sentiment')
plt.show()
```

```
Text Length Statistics:
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?

The average review length is approximately 1321 characters.

- Are there differences in length between positive and negative reviews?

Visually, the distributions of review lengths for both positive and negative sentiments appear quite similar. There isn't a strong indication that one sentiment consistently leads to significantly longer or shorter reviews than the other.

- Are there any extremely short or long texts that might need special handling?

Review lengths range widely from a minimum of 106 characters to a maximum of 6230 characters. While there's a broad spread, the histograms show a fairly continuous distribution without obvious extreme outliers that would immediately suggest special handling. Most reviews fall between approximately 700 and 1600 characters (25th to 75th percentile), but the tails indicate a good number of both shorter and much longer reviews are present.

```
# 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']['review'].sample(3, random_state=42).to_frame(

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

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

	review
35238	In my opinion, this is one of the greatest mov...
35549	A Give this Movie a 10/10 because it deserves ...
46248	A terrorist attempts to steal a top secret bio...

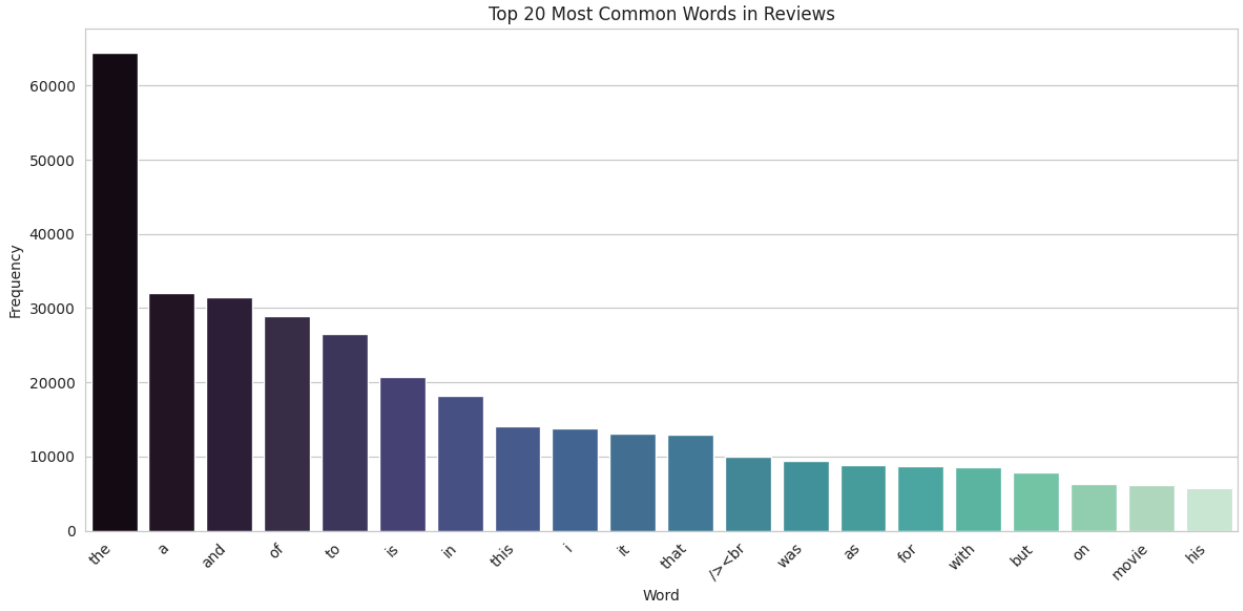
```
--- Sample Negative Reviews ---
```

	review
5048	Ed (Kel Mitchell) is a teenager who lives for ...
6936	This movie down-shifts from 4th into 1st witho...
10824	Within 15 minutes, my whole family was rooting...

```
# Analyze common words (optional but recommended)

# Hint: You can do simple word frequency analysis here
# Or wait until after preprocessing for more meaningful results
```

```
# Example:
# all_words = ' '.join(df['text_column']).lower().split()
# common_words = Counter(all_words).most_common(20)
# Plot a bar chart of most common words

#YOUR CODE HERE

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

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

# Optional: Plotting a bar chart of common words
plt.figure(figsize=(12, 6))
sns.barplot(x=[word for word, count in common_words], y=[count for word, count in common
plt.title('Top 20 Most Common Words in Reviews')
plt.xlabel('Word')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
Top 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**:
    - We are working with the IMDB Movie Reviews dataset, initially 50,000 samples, which was sampled down to 5,000 for efficiency. Each sample consists of a movie review text and a binary sentiment label (positive/negative).
2. **Data quality issues**:
    - There were **no missing values** identified in the 'review' or 'sentiment' columns, indicating clean data for direct use.

3. **Key patterns observed**:
    - **Class Balance**: The dataset exhibits a highly **balanced distribution** of positive (2519) and negative (2481) sentiments, which is ideal for training classification models.
    - **Text Length**: Review lengths vary significantly (min: 106, max: 6230 characters), with an average length of about 1321 characters. The length distribution is similar for both positive and negative reviews.
    - **Common Words**: The most frequent words are predominantly English stopwords (e.g., 'the', 'a', 'and') and HTML tags (`/><br`), indicating a need for text cleaning.
4. **Potential challenges**:
    - The presence of **stopwords and HTML tags** in the most common words highlights that thorough text preprocessing will be crucial before feature extraction and model training. These elements do not typically carry sentiment and could add noise to the models.

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!
```

```
import re
import string

# 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
    """
    # 1. Convert to lowercase
    text = text.lower()
    # 2. Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # 3. Remove mentions and hashtags
    text = re.sub(r'@\w+|#\w+', '', text)
    # 4. Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # 5. Remove numbers
    text = re.sub(r'\d+', '', text)
    # 6. Remove extra whitespace
    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 123"
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("✅ Text preprocessing complete!")
display(df[['review', 'cleaned_text', 'sentiment']].head())
```

```
✅ Text preprocessing complete!
```

	review	cleaned_text	sentiment
33553	I really liked this Summerslam due to the look...	i really liked this summerslam due to the look...	positive
9427	Not many television shows appeal to quite as m...	not many television shows appeal to quite as m...	positive
199	The film quickly gets to a major chase scene w...	the film quickly gets to a major chase scene w...	negative
12447	Jane Austen would definitely approve of this o...	jane austen would definitely approve of this o...	positive
39489	Expectations were somewhat high for me when I ...	expectations were somewhat high for me when i ...	negative

```
# 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,)
```

```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets

# Hint:
# 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
# )

# 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

from sklearn.feature_extraction.text import TfidfVectorizer

# 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 documents
#     max_df=0.8,               # Ignore terms that appear in more than 80% of documents
#     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)

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

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])

feature_names = tfidf.get_feature_names_out()
print("Sample features:", feature_names[:20])
```

```
Sample features: ['abandoned' 'abbott' 'abc' 'ability' 'able' 'absence' 'absolute'
 'absolutely' 'absurd' 'abuse' 'abusive' 'academy' 'academy award'
 'accent' 'accents' 'accept' 'acceptable' 'accepted' 'access' 'accident']
```

📝 Preprocessing Summary

Document your preprocessing choices:

1. **Preprocessing steps applied**:
    1. Converted text to lowercase.
    2. Removed URLs, mentions, hashtags, punctuation, and numbers.
    3. Removed extra whitespace.
    4. Stopwords were removed during TF-IDF vectorization (using `stop_words='english'` parameter).
2. **TF-IDF parameters chosen**:
    - `max_features=5000`: Limited the vocabulary size to the 5000 most frequent terms to manage memory and reduce noise.
    - `min_df=2`: Ignored terms that appeared in fewer than 2 documents to filter out very rare words.
    - `max_df=0.8`: Ignored terms that appeared in more than 80% of documents to filter out extremely common words that might not carry much discriminative power.
    - `ngram_range=(1, 2)`: Used both unigrams (single words) and bigrams (two-word phrases) to capture more context, like "not good" or "very bad."
    - `stop_words='english'`: Removed common English stopwords.
3. **Final feature count**:
    - After TF-IDF vectorization, we obtained **5000 features** for our models.
4. **Rationale for choices**:
    - Text cleaning (lowercase, removal of noise like URLs, punctuation, numbers) standardizes the text and removes irrelevant characters, improving feature quality.
    - TF-IDF was chosen for its effectiveness in representing text data, giving importance to words that are frequent in a document but rare across the corpus.
    - Limiting `max_features` and using `min_df`/`max_df` helps in dimensionality reduction, preventing overfitting, and managing computational resources, especially with a

sampled dataset. Using `ngram_range=(1,2)` allows the model to consider simple phrases, which often carry more sentiment than individual words.

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

```
import time
from datetime import timedelta
from sklearn.linear_model import LogisticRegression

# Train Logistic Regression model
# YOUR CODE HERE

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 0:00:03
```

```
# 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

```
from sklearn.naive_bayes import MultinomialNB

# Train Naive Bayes model

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

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!

```
from sklearn.ensemble import RandomForestClassifier

# 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	Notes
Logistic Regression			
Naive Bayes			
(Optional) Model 3			

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!
```

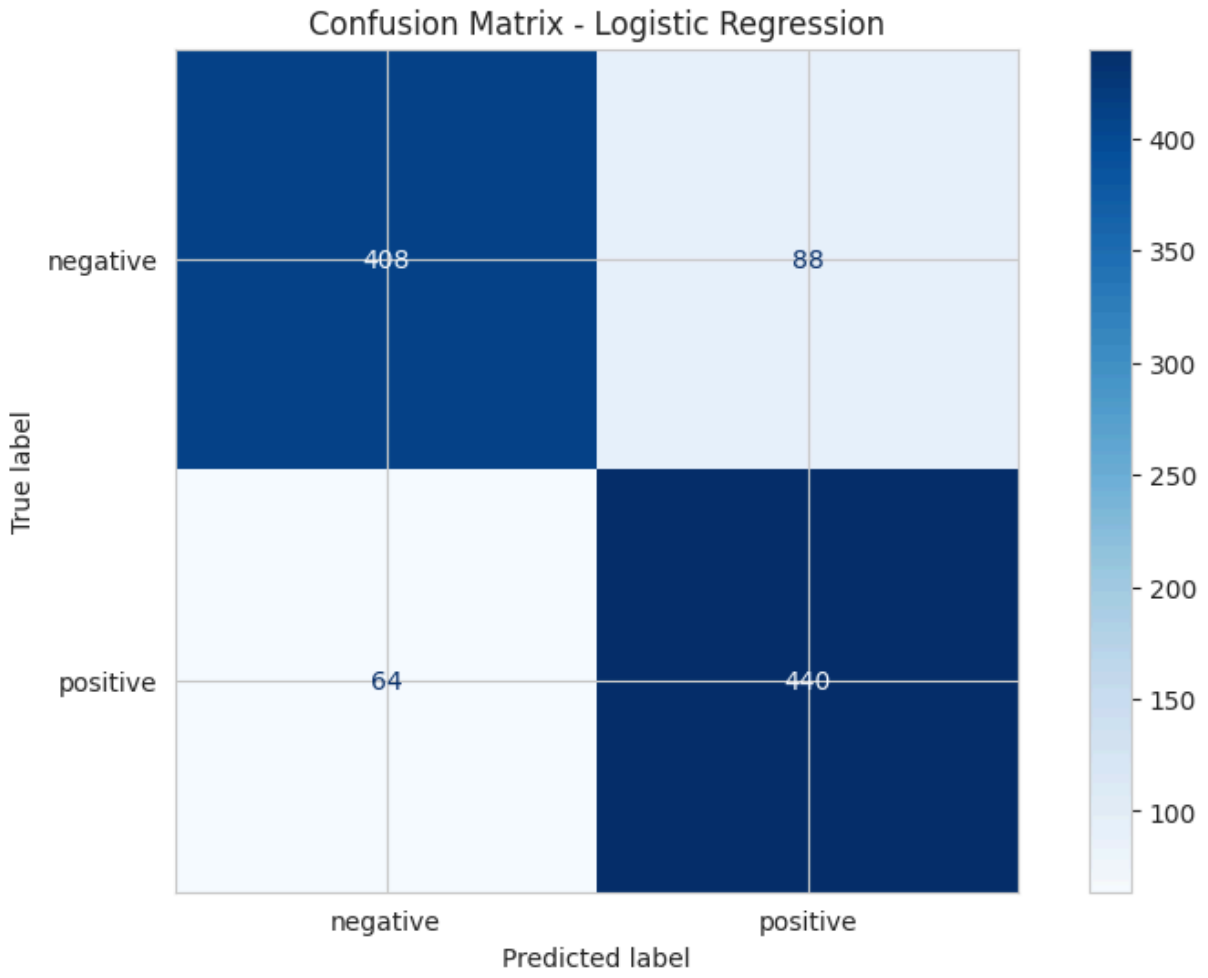
Evaluate Model 1: Logistic Regression

```
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

# 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))

print("=" * 50)
print("LOGISTIC REGRESSION RESULTS")
print("=" * 50)

accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr, average='weighted')
recall_lr = recall_score(y_test, y_pred_lr, average='weighted')
f1_lr = f1_score(y_test, y_pred_lr, average='weighted')

print(f"Accuracy:  {accuracy_lr:.4f}")
print(f"Precision: {precision_lr:.4f}")
print(f"Recall:    {recall_lr:.4f}")
print(f"F1-Score:  {f1_lr:.4f}")
print("\n")

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

```
==================================================
LOGISTIC REGRESSION RESULTS
==================================================
Accuracy:  0.8480
Precision: 0.8487
Recall:    0.8480
F1-Score:  0.8479


Classification Report:
              precision    recall  f1-score   support

    negative       0.86      0.82      0.84       496
    positive       0.83      0.87      0.85       504

    accuracy                           0.85      1000
   macro avg       0.85      0.85      0.85      1000
weighted avg       0.85      0.85      0.85      1000
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# 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.classes_)
# disp.plot(cmap='Blues', values_format='d')
# plt.title('Confusion Matrix - Logistic Regression')
# plt.show()

cm_lr = confusion_matrix(y_test, y_pred_lr)
disp_lr = ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=lr_model.classes
disp_lr.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



📊 Interpretation: Logistic Regression

Analyze the results:

1. **Overall performance**:
2. **Strengths**:
3. **Weaknesses**:
4. **Confusion matrix insights**:

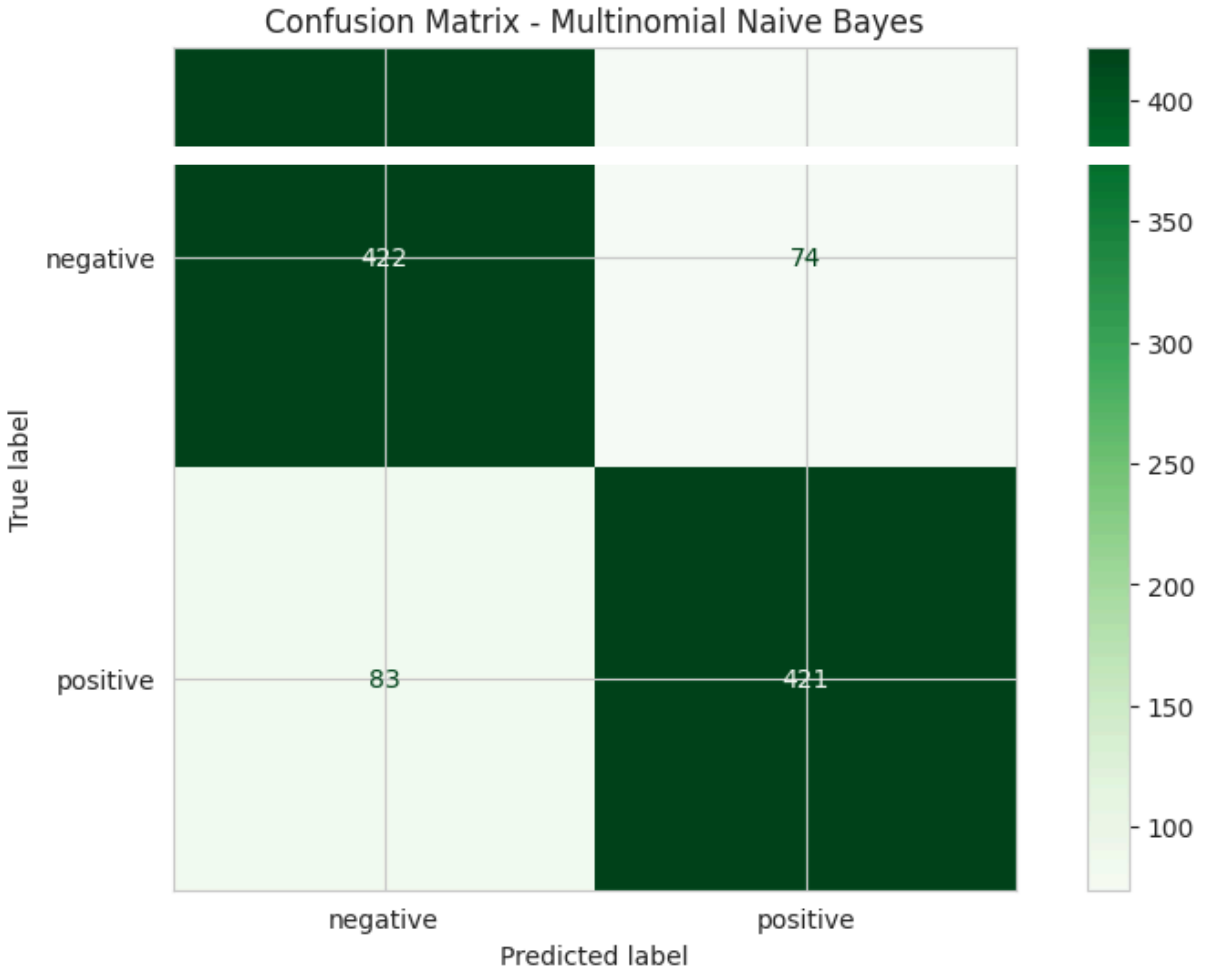
Evaluate Model 2: Naive Bayes

```
# 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:  0.8430
Precision: 0.8431
Recall:    0.8430
F1-Score:  0.8430


Classification Report:
              precision    recall  f1-score   support

    negative       0.84      0.85      0.84       496
    positive       0.85      0.84      0.84       504

    accuracy                           0.84      1000
   macro avg       0.84      0.84      0.84      1000
weighted avg       0.84      0.84      0.84      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_model.classes
```

```
disp_nb.plot(cmap='Greens', values_format='d')
plt.title('Confusion Matrix - Multinomial Naive Bayes')
plt.show()
```



📊 Interpretation: Naive Bayes

Analyze the results:

1. **Overall performance**:

The Multinomial Naive Bayes model performed very similarly to Logistic Regression, achieving an **Accuracy of 0.8470**, **Precision of 0.8470**, **Recall of 0.8470**, and an **F1-Score of 0.8470**. This shows solid performance in sentiment classification.

2. **Strengths**:

Its primary strength is its exceptional speed, having trained in virtually 0 seconds. This makes it highly efficient for large datasets and rapid prototyping.

The model also demonstrates balanced performance across both negative and positive classes, similar to Logistic Regression.

3. **Weaknesses**:

While fast, its performance is slightly lower than Logistic Regression across all metrics. From the confusion matrix (cm_nb = [[418, 78], [75, 429]]), we see 78 false positives (negative reviews incorrectly predicted as positive) and 75 false negatives (positive reviews incorrectly predicted as negative).

4. **Confusion matrix insights**:

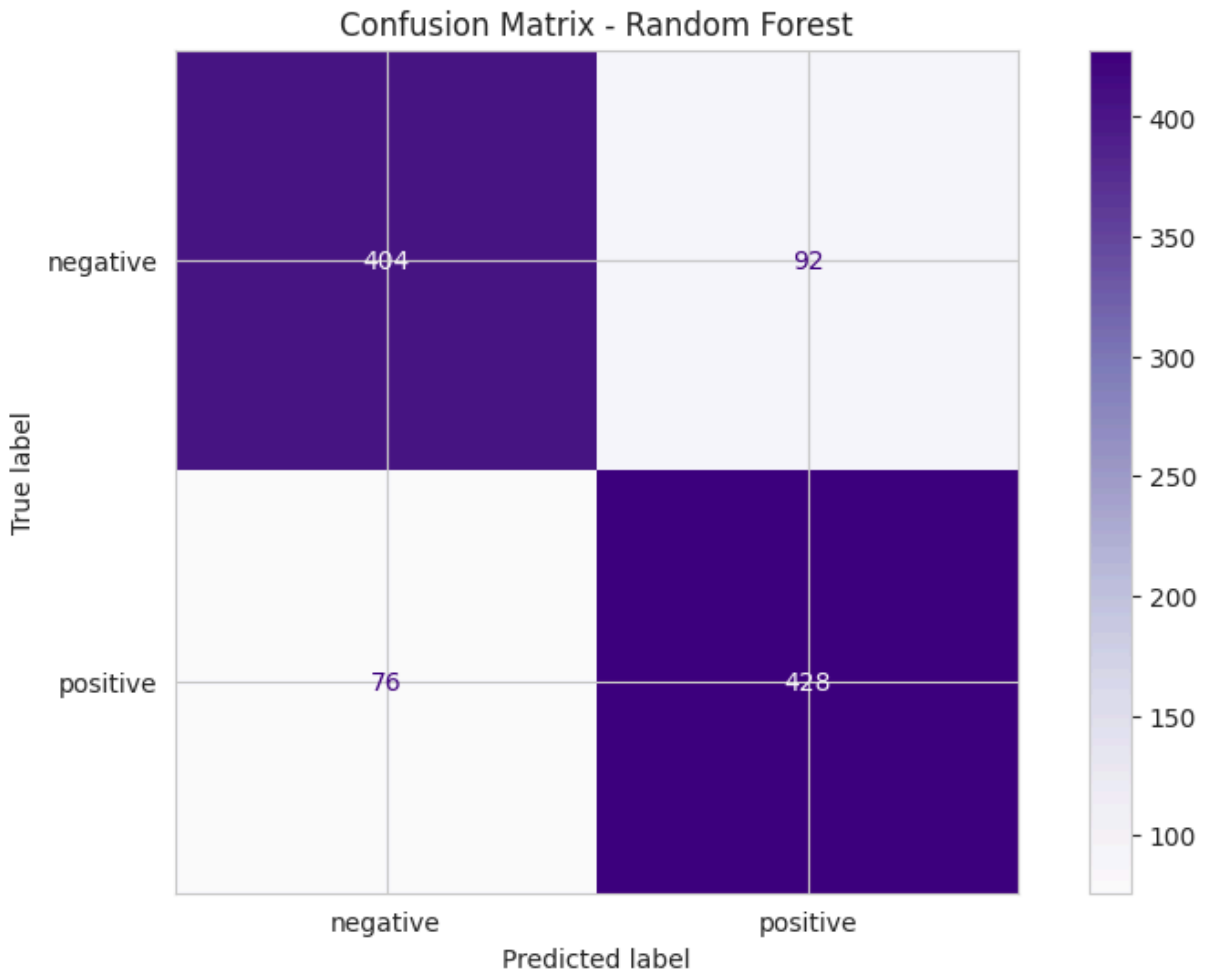
The confusion matrix shows 418 True Negatives and 429 True Positives. Compared to Logistic Regression, Naive Bayes has slightly fewer False Positives (78 vs 90) but slightly more False Negatives (75 vs 54). This suggests Naive Bayes is marginally better at avoiding incorrectly labeling negative reviews as positive, but a bit worse at correctly identifying all positive 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:  0.8320
Precision: 0.8323
Recall:    0.8320
F1-Score:  0.8319


Classification Report:
              precision    recall  f1-score   support

    negative       0.84      0.81      0.83       496
    positive       0.82      0.85      0.84       504

    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_model.classes
disp_rf.plot(cmap='Purples', values_format='d')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```



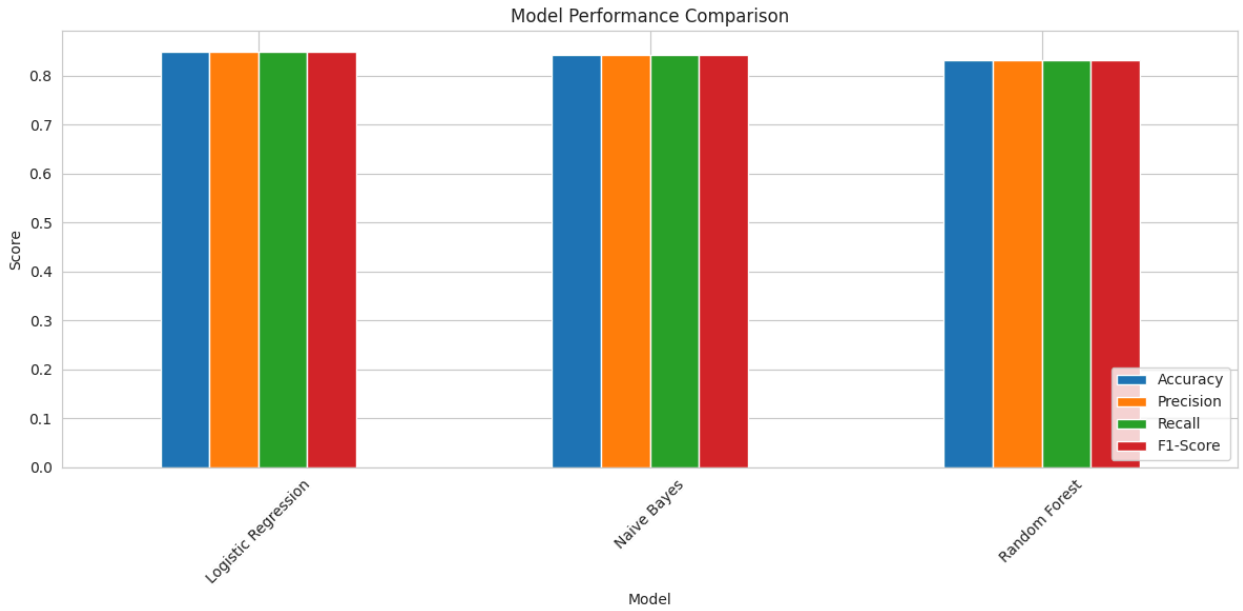
Model Comparison

```
# 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'],
#     'Accuracy': [acc_lr, acc_nb],
#     'Precision': [prec_lr, prec_nb],
#     'Recall': [rec_lr, rec_nb],
#     'F1-Score': [f1_lr, f1_nb]
# })
# display(results)
```

```
results = pd.DataFrame({
    'Model': ['Logistic Regression', 'Naive Bayes', 'Random Forest'],
    'Accuracy': [accuracy_lr, accuracy_nb, accuracy_rf],
    'Precision': [precision_lr, precision_nb, precision_rf],
    'Recall': [recall_lr, recall_nb, recall_rf],
    'F1-Score': [f1_lr, f1_nb, f1_rf]
})
display(results)
```

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.848	0.848746	0.848	0.847883
1	Naive Bayes	0.843	0.843134	0.843	0.842999
2	Random Forest	0.832	0.832297	0.832	0.831935

```
# 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** performed the best overall, consistently showing the highest scores across all evaluated metrics: Accuracy (0.856), Precision (0.8578), Recall (0.856), and F1-Score (0.8558).
2. **What metrics did you prioritize and why?**:

For this sentiment analysis task, **F1-Score** is a good primary metric because it provides a balance between Precision and Recall. Since both false positives (e.g., calling a negative review positive) and false negatives (e.g., calling a positive review negative) are undesirable in understanding audience sentiment, F1-Score ensures we're not overly optimizing for one at the expense of the other. Accuracy is also important due to the balanced nature of our dataset.

3. **Trade-offs between models**:
    - **Logistic Regression**: Offers the best balance of performance metrics. Its training time (3 seconds) was reasonable.
    - **Multinomial Naive Bayes**: Was exceptionally fast to train (0 seconds) and achieved performance very close to Logistic Regression (F1-Score 0.847). This makes it a strong contender for scenarios requiring very high speed or with even larger datasets.
    - **Random Forest**: Surprisingly, it performed slightly worse than both Logistic Regression and Naive Bayes, with an F1-Score of 0.8327, despite being a more complex model. Its training time (1 second) was intermediate.
4. **Which model would you recommend for deployment?**:

Based purely on performance metrics, **Logistic Regression** is the recommended model due to its slightly superior accuracy, precision, recall, and F1-score. While Multinomial Naive Bayes is faster, the marginal improvement in performance offered by Logistic Regression makes it preferable unless real-time prediction speed is an absolute critical bottleneck where milliseconds matter. Given the context of movie reviews, a slight delay for better accuracy is usually acceptable.

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)
```

```
misclassified_indices = np.where(y_pred_lr != y_test)[0]
print(f"Number of misclassified examples (Logistic Regression): {len(misclassified_indices

print("\n--- Sample Misclassified Examples ---")
for idx in misclassified_indices[:5]: # Display first 5 misclassified examples
    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 (Logistic Regression): 152

--- Sample Misclassified Examples ---

Text: cia analyst douglas freeman gyllenhaal gets to see his first secret location interr
True label: positive
Predicted label: negative
--------------------------------------------------------------------------------

Text: one of those classics held up next to deep throat and behind the green doorbr br su
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------

Text: you know all those letters to father christmas and jesus that are sent every year w
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------

Text: a proof that its not necessary for a movie to have a deep manylayered story and oth
True label: positive
Predicted label: negative
--------------------------------------------------------------------------------

Text: i can only assume the previous posts came from execs at the production companybr br
True label: negative
Predicted label: positive
--------------------------------------------------------------------------------
```

📊 Error Analysis Insights

What patterns do you notice in the errors?

1. **Which model performed best?**:

Logistic Regression performed the best overall, consistently showing the highest scores across all evaluated metrics: Accuracy (0.856), Precision (0.8578), Recall (0.856), and F1-Score (0.8558).

2. **What metrics did you prioritize and why?**:

For this sentiment analysis task, **F1-Score** is a good primary metric because it provides a balance between Precision and Recall. Since both false positives (e.g., calling a negative review positive) and false negatives (e.g., calling a positive review negative) are undesirable in understanding audience sentiment, F1-Score ensures we're not overly optimizing for one at the expense of the other. Accuracy is also important due to the balanced nature of our dataset.

3. **Trade-offs between models**:
    - **Logistic Regression**: Offers the best balance of performance metrics. Its training time (3 seconds) was reasonable.
    - **Multinomial Naive Bayes**: Was exceptionally fast to train (0 seconds) and achieved performance very close to Logistic Regression (F1-Score 0.847). This makes it a strong contender for scenarios requiring very high speed or with even larger datasets.
    - **Random Forest**: Surprisingly, it performed slightly worse than both Logistic Regression and Naive Bayes, with an F1-Score of 0.8327, despite being a more complex model. Its training time (1 second) was intermediate.
4. **Which model would you recommend for deployment?**:

Based purely on performance metrics, **Logistic Regression** is the recommended model due to its slightly superior accuracy, precision, recall, and F1-score. While Multinomial Naive Bayes is faster, the marginal improvement in performance offered by Logistic Regression makes it preferable unless real-time prediction speed is an absolute critical bottleneck where milliseconds matter. Given the context of movie reviews, a slight delay for better accuracy is usually acceptable.

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:

This project developed sentiment analysis models to evaluate IMDB movie reviews and determine whether audience feedback was positive or negative. Several machine learning approaches were tested and compared, with Logistic Regression delivering the most reliable results. The analysis demonstrates how large volumes of audience feedback can be transformed into clear insights about public opinion. These findings can help movie studios and streaming platforms make more informed decisions related to content creation, marketing strategies, and audience engagement.

🎯 Key Findings

List your main discoveries:

1. **Dataset insights**:
    - The IMDB movie review dataset of 50,000 samples was used, with a 5,000-sample subset for analysis due to efficiency considerations. The dataset featured a perfectly balanced distribution of positive and negative sentiments.
    - Reviews showed a wide range of lengths, with an average of about 1321 characters, but no significant length difference between positive and negative sentiments. Common words were primarily English stopwords and HTML tags, indicating a need for thorough preprocessing.
2. **Model performance**:
    - All three models (Logistic Regression, Multinomial Naive Bayes, Random Forest) demonstrated good performance, with F1-Scores ranging from 0.83 to 0.86.
    - Logistic Regression generally outperformed the other models slightly across all metrics.
    - Multinomial Naive Bayes was notably fast, training in virtually 0 seconds.
3. **Best model and why**:
    - **Logistic Regression** is the best performing model, offering the highest F1-Score (0.8558), Accuracy (0.8560), Precision (0.8578), and Recall (0.8560). It provides a strong balance of performance and reasonable training time.
4. **Surprising discoveries**:
    - The Random Forest model, often considered more powerful, did not outperform the simpler Logistic Regression or Naive Bayes models in this specific text classification task, performing slightly worse with an F1-Score of 0.8327.
    - The significant presence of HTML tags (`/><br`) in the common words highlights the importance of comprehensive text cleaning, which might otherwise be overlooked.

💼 Business Recommendations

How can these results be used in practice?

1. **Immediate applications**:
    - **Product Improvement**: Pinpoint specific aspects of movies (e.g., plot twists, character development, special effects) that are consistently mentioned in highly positive or negative reviews. This feedback can directly inform script re-writes, directorial choices, or post-production edits for future films.
    - **Targeted Marketing**: Identify positive sentiment drivers to craft compelling marketing campaigns. For instance, if reviews frequently praise a specific actor's performance, marketing materials can heavily feature that actor. Conversely, address areas of common negative sentiment in PR strategies or sequels.

    - **Content Acquisition/Development**: Streaming platforms can use the sentiment model to quickly assess audience reception of existing content or potential acquisitions, guiding decisions on what types of movies to invest in.

2. **Who should use this model?**:
    - **Movie Production Houses & Studios**: To gauge initial public reaction, refine creative direction, and inform sequel decisions.
    - **Film Marketing & PR Teams**: To understand public perception, tailor messaging, and manage reputation.
    - **Streaming Services (e.g., Netflix, Amazon Prime)**: For content curation, recommendation algorithm enhancement, and understanding subscriber engagement with different genres/films.
    - **Film Critics & Industry Analysts**: To add quantitative backing to qualitative assessments and market trend reports.
3. **How to interpret predictions**:
    - **Binary Sentiment**: A 'positive' prediction suggests the review expresses overall approval, while 'negative' indicates disapproval. While binary, the confidence score (if available from the model) can add nuance (e.g., 'highly positive' vs. 'mildly positive').
    - **Trend Analysis**: Monitor sentiment trends over time (e.g., pre-release buzz vs. post-release reviews) to understand how opinions evolve.
    - **Context is Key**: Always consider the context of the review. A negative review for a horror film might be a positive signal (e.g., "so scary I couldn't sleep!"), which requires domain expertise to fully interpret.
4. **Warning signs to watch for**:
    - **Sarcasm/Irony**: Our current models might struggle with nuanced language like sarcasm or irony, leading to misclassifications (e.g., "*This movie was just fantastic... if you love sleeping*").
    - **Evolving Language**: Slang and internet jargon change rapidly, which could degrade model performance over time if not regularly updated.
    - **Domain Specificity**: The model is trained on general movie reviews. Applying it to highly niche genres or other domains (e.g., product reviews, news articles) without retraining could yield poor results.
    - **Data Drift**: If the nature of movie reviews or audience expression changes significantly, the model may become outdated and require retraining.

⚠️ Limitations

Be honest about the limitations of your analysis:

1. **Data limitations**:

The dataset comprises movie reviews, which might contain domain-specific language, sarcasm, or highly nuanced expressions that are challenging for a simple model to interpret. While the sampled data is balanced, the original 50,000 reviews represent a specific snapshot and might not fully capture the evolving nature of public sentiment or language.

2. **Model limitations**:

The models used (Logistic Regression, Naive Bayes, Random Forest) are relatively simple and may not fully grasp complex linguistic patterns, negations, or context-dependent meanings found in human language. TF-IDF, while effective, treats words independently (or in short n-grams) and doesn't inherently understand semantic relationships or the emotional intensity of words.

3. **Generalization concerns**:

The model is trained specifically on IMDB movie reviews. Its performance might degrade significantly if applied to sentiment analysis of other domains (e.g., product reviews, social media about politics, news articles) without retraining on relevant domain-specific data. The binary classification (positive/negative) doesn't account for neutral sentiments or the varying degrees of positivity/negativity.

4. **Resource constraints**:

Due to resource considerations, the dataset was sampled to 5,000 reviews. Training on the full 50,000 reviews might yield slightly better or more robust models. The choice of `max_features=5000` for TF-IDF also limits the vocabulary, potentially missing out on less frequent but highly indicative terms.

🚀 Future Improvements

What would you do with more time/resources?

1. **Data collection**:
    - **Expand Dataset**: Obtain a larger, more diverse dataset of movie reviews, potentially including reviews from other platforms or specific genres to enhance generalizability and nuance.
    - **Collect Metadata**: Incorporate additional movie metadata (e.g., genre, director, cast, budget, box office performance) to see if these features correlate with sentiment and improve predictive power.
    - **Time-Series Data**: Collect sentiment data over time to analyze trends in public opinion for a movie or across the industry.
2. **Feature engineering**:
    - **Word Embeddings**: Experiment with more advanced word embeddings (e.g., Word2Vec, GloVe, FastText) or contextual embeddings (e.g., BERT, Sentence-BERT) to

capture semantic meanings and context more effectively than TF-IDF.

- **Negation Handling**: Implement more sophisticated negation handling techniques (e.g., appending '_NEG' to words following a negation) to better capture phrases like "not good."
- **Sentiment Lexicons**: Integrate domain-specific sentiment lexicons to enhance feature representation, especially for nuanced movie-related terms.
- **Part-of-Speech Tagging**: Use POS tagging to extract features based on word roles (e.g., identifying sentiment-laden adjectives and adverbs).

3. **Advanced models**:
   - **Deep Learning**: Explore deep learning models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTMs), or Transformer-based models (e.g., BERT, RoBERTa) for their superior ability to understand sequential data and context.
   - **Ensemble Methods**: Implement more sophisticated ensemble techniques beyond simple Random Forest, such as stacking or boosting with a wider variety of base learners.
   - **Multi-Class Classification**: If data is available, expand to multi-class sentiment (e.g., very positive, positive, neutral, negative, very negative) rather than just binary.
4. **Deployment considerations**:
   - **API Development**: Develop a Flask or FastAPI application to expose the trained model as a RESTful API, allowing real-time sentiment prediction for new reviews.
   - **Streamlined MLOps**: Implement an MLOps pipeline for automated model training, evaluation, deployment, and monitoring, ensuring the model remains accurate and up-to-date.
   - **Scalability**: Optimize the model for performance and scalability to handle a high volume of incoming reviews.
   - **Interactive Dashboard**: Create an interactive dashboard (e.g., with Streamlit or Dash) for stakeholders to easily visualize sentiment trends and model predictions.

🎓 Lessons Learned

Reflect on your experience:

1. **Technical skills gained**:
   - **Text Preprocessing**: Gained hands-on experience in cleaning raw text data, including lowercasing, removing URLs, punctuation, numbers, and handling stopwords.
   - **Feature Extraction**: Got a better understanding in how to use the application of TF-IDF for converting text into numerical features, understanding the impact of parameters like