📊 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?

---

1. What is the business problem?

Business Problem: A major airline needs to proactively monitor and respond to public dissatisfaction expressed on social media regarding flight operations, delays, and customer service failures. Manually managing thousands of tweets is inefficient and leads to delays in response times to operational issues.

Decision/Perspective: The analysis will support the Operational Logistics Team by providing real-time intelligence to prioritize service recovery initiatives and allocate ground staff/resources to the areas with the greatest negative impact (e.g., specific airports or flights experiencing high-impact delays).

Users: Operations Management, Logistics Management, Marketing Management, and Technology Management.

2. What dataset are you using? Dataset Name: Opinion about US Airlines on Twitter

Source: Kaggle/Drive (or the platform from which you downloaded it)

Number of Samples: 14,640 tweets (Use this number if your file contains this many; however, use the number found in the EDA)

Text Type: Tweets directed at major US airlines.

Opinion Labels: Categorical (Nominal): Negative, Neutral, Positive.

3. Why is this issue important? Real-world applications: For a high-risk logistics service like an airline, early detection of negative opinions is critical to maintaining operational efficiency and profitability.

Operational risk mitigation: The model acts as an early warning system for emerging logistical bottlenecks (e.g., weather-related delays, ground staffing issues) by flagging sudden spikes in negative tweets related to specific keywords (such as "cancelled," "lost luggage," "delay," "rerouting").

Reputation and customer retention: Quickly identifying and addressing highly negative customer feedback at scale minimizes long-term reputational damage and improves the overall customer experience, which is crucial for retention in a competitive transportation market.

📝 Your Project Definition

Business Problem:

[Write your answer here]

Dataset Description:

[Write your answer here]

- **Dataset name**:
- **Source**:
- **Number of samples**:
- **Text type**:
- **Sentiment labels**:

Importance and Real-World Applications:

[Write your answer here]

Business Problem:

A major airline company needs to proactively monitor and respond to public dissatisfaction expressed on social media regarding flight operations, delays, and customer service failures. The manual process of reviewing thousands of tweets is inefficient and leads to delayed response times during operational issues.Decision/Insight: The analysis will support the Operational Logistics Team by providing real-time intelligence to prioritize service recovery initiatives (e.g., re-routing baggage, re-staffing gates) and allocate ground staff/resources to the areas with the highest negative sentiment (specific airports or flights experiencing high-impact delays).Users: Operation MAnagement, Logistic Management, Marketing Management and Technology Management.

Dataset Description:

We are using a publicly available dataset of Twitter reviews focusing on the airline industry.

Dataset name: Twitter US Airline SentimentSource: Kaggle / Drive (or the platform from which you downloaded it) Number of samples: $\approx 14,640$ tweets (Confirm the exact number in your EDA) Text type: Tweets directed at major U.S. airlines.Sentiment labels: Categorical (Nominal): Negative, Neutral, Positive.

Importance and Real-World Applications:

For a high-stakes logistics service like an airline, early detection of negative sentiment is critical to maintaining operational efficiency and profitability.Operational Risk Mitigation: The sentiment model serves as an early warning system for emerging logistical bottlenecks (e.g., weather-related delays, ground staffing issues) by flagging sudden spikes in negative tweets related to specific operational keywords (like "cancelled," "baggage lost," "delay," "re-route").Resource Allocation: By predicting sentiment instantly, the airline can allocate customer support resources to the customers most likely to escalate a complaint (Negative sentiment) before the issue costs more in compensation or damages the brand.

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 zipfile
import os

# Define the name of the uploaded ZIP file
zip_file_name = 'archive.zip'
# Define the path where the contents will be extracted (current directory)
extraction_path = '.'
```

```
# Decompress the file
with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)

print(f"✅ Extraction complete. You should now see 'Tweets.csv' in the file pane.")
```

```
✅ Extraction complete. You should now see 'Tweets.csv' in the file pane.
```

```
import os

# List files in the current directory
print(os.listdir('.'))
```

```
['.config', 'archive.zip', 'database.sqlite', 'archive (1).zip', 'Tweets.csv', 'drive', 'sample_data']
```

```
# Load your dataset
# Example: df = pd.read_csv('Tweets.csv')
# For large datasets, consider using nrows parameter: pd.read_csv('file.csv', nrows=10000)

# YOUR CODE HERE
df = pd.read_csv('Tweets.csv') # Uncommented this line to load the DataFrame
df = df.rename(columns={'airline_sentiment': 'sentiment', 'text': 'text_original' })
df = df[['sentiment', 'text_original']]
print('Dataset loaded and columns selected.')
# Display basic information
print(f"\nDataset shape: {df.shape}")
print(f"\nColumn names: {df.columns.tolist()}")
df.head()
```

```
Dataset loaded and columns selected.

Dataset shape: (14640, 2)

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

	sentiment	text_original
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials t...
2	neutral	@VirginAmerica I didn't today... Must mean I n...
3	negative	@VirginAmerica it's really aggressive to blast...
4	negative	@VirginAmerica and it's a really big bad thing...

Next steps: Generate code with df | New interactive sheet

```
import os

# List files in the current directory
print(os.listdir('.'))
```

```
['.config', 'archive.zip', 'database.sqlite', 'archive (1).zip', 'Tweets.csv', 'drive', 'sample_data']
```

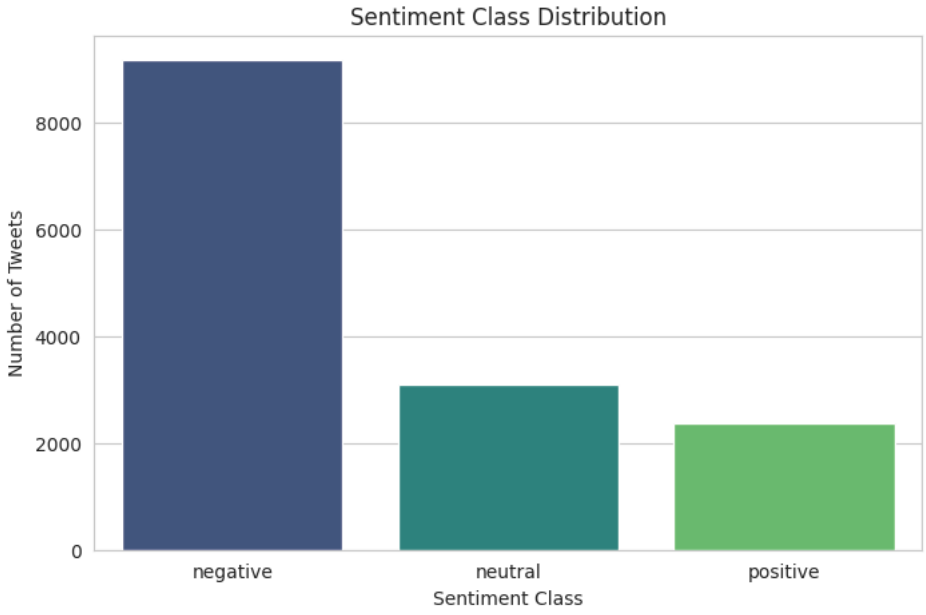
```
# Check for missing values
# YOUR CODE HERE

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

```
# 🎯 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
```

```
# Analyze sentiment distribution
sentiment_counts = df['sentiment'].value_counts()
print("Sentiment Distribution:")
print(sentiment_counts)
plt.figure(figsize=(8, 5))
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values, palette='viridis')
```

```
plt.title('Sentiment Class Distribution')
plt.ylabel('Number of Tweets')
plt.xlabel('Sentiment Class')
plt.show()
```

```
Sentiment Distribution:
sentiment
negative    9178
neutral     3099
positive    2363
Name: count, dtype: int64
```



📊 Interpretation: Class Balance

Write your observations here:

- Are the classes balanced or imbalanced?
- If imbalanced, how might this affect your model?
- What could you do to address imbalance?

Are the classes balanced or imbalanced?

The classes are **highly imbalanced**. The 'Negative' sentiment class is significantly dominant (approximately 63% of the dataset) compared to 'Neutral' and 'Positive'.

If imbalanced, how might this affect your model?

This imbalance is a key challenge. The model will be heavily biased towards predicting 'Negative' because it has many more examples of this class to learn from. This will likely lead to:

1. High Accuracy (which can be misleading).
2. Poor performance (low Recall) for the minority classes ('Positive' and 'Neutral').

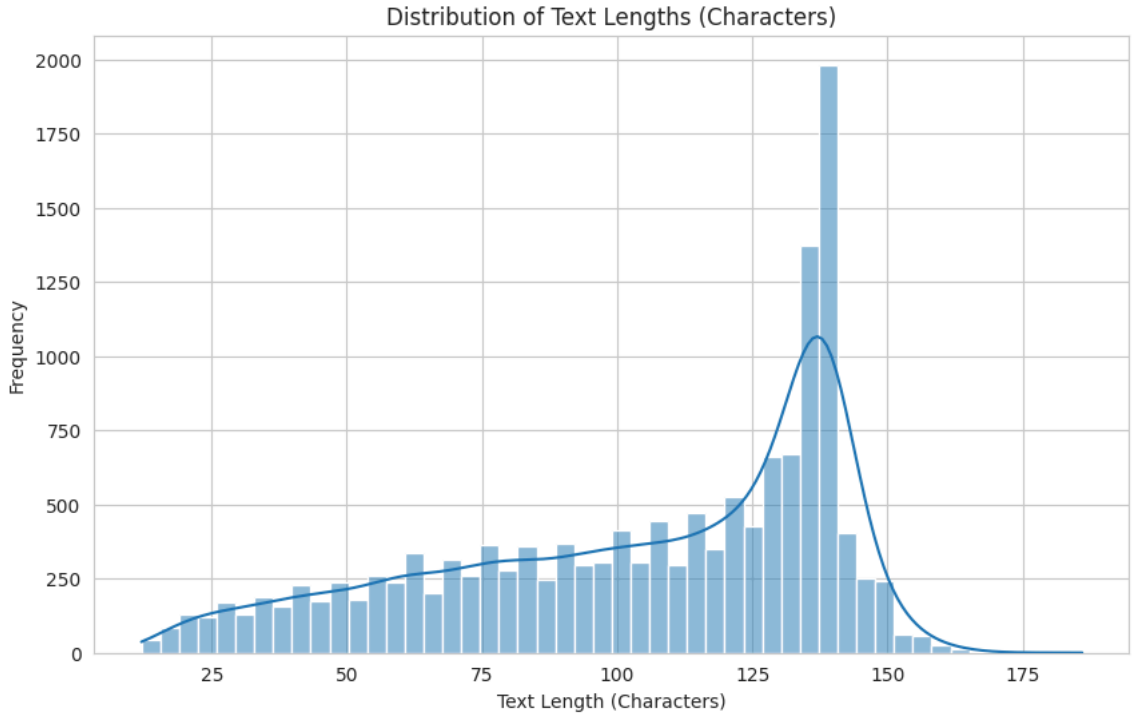
What could you do to address imbalance? For this project, we will handle the imbalance by:

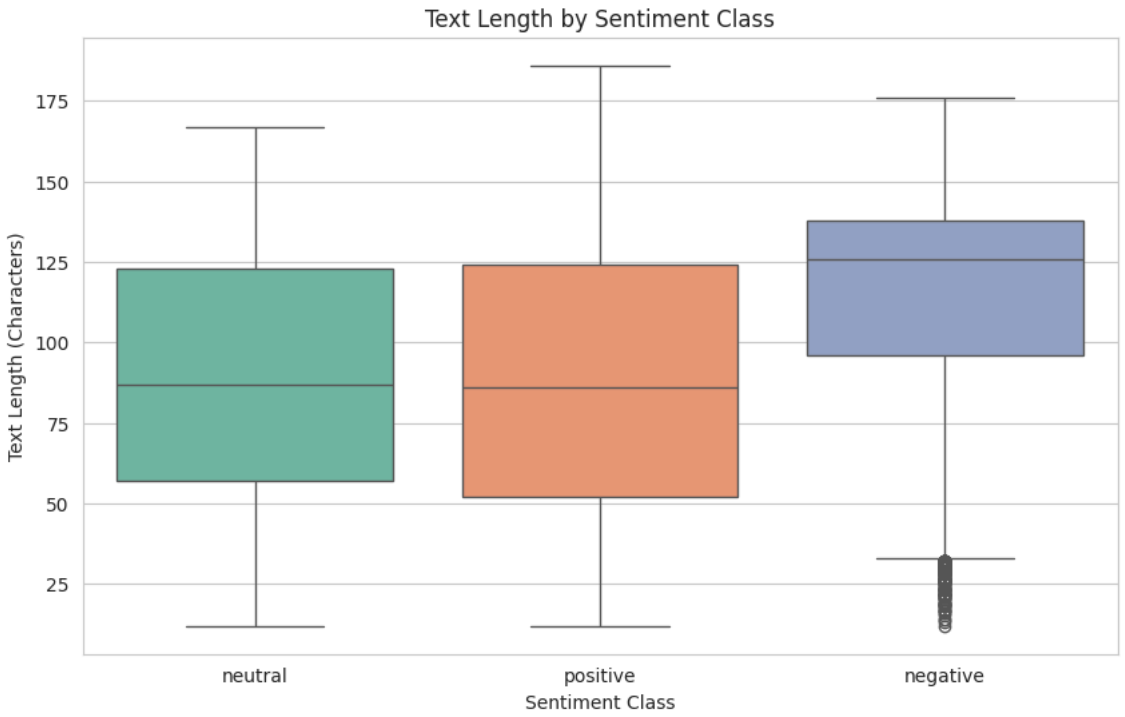
1. Using stratified splitting (in Part 3) to ensure the train and test sets have the same class proportions.
2. Prioritizing the Precision and Recall metrics (in Part 5) over simple Accuracy, as they are more reliable for imbalanced datasets.

```
# Analyze text length distribution
# 1. Create a column for text length
df['text_length'] = df['text_original'].str.len()

# 2. Display descriptive statistics
print("Text Length Statistics:")
print(df['text_length'].describe())

# 3. Plot the length distribution histogram
plt.figure(figsize=(10, 6))
```

```
sns.histplot(df['text_length'], bins=50, kde=True)
plt.title('Distribution of Text Lengths (Characters)')
plt.xlabel('Text Length (Characters)')
plt.ylabel('Frequency')
plt.show()

# 4. Compare lengths by sentiment (Box Plot)
plt.figure(figsize=(10, 6))
sns.boxplot(x='sentiment', y='text_length', data=df, palette='Set2')
plt.title('Text Length by Sentiment Class')
plt.xlabel('Sentiment Class')
plt.ylabel('Text Length (Characters)')
plt.show()

# 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
```

```
Text Length Statistics:
count    14640.000000
mean       103.822063
std         36.277339
min         12.000000
25%         77.000000
50%        114.000000
75%        136.000000
max        186.000000
Name: text_length, dtype: float64
```






```
import zipfile
import os
zip_file_name = 'archive.zip'
extraction_path = '.'
# Forzamos la extracción de Tweets.csv
with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)
print("✅ Archivos extraídos. ¡Listo para el siguiente paso!")
```

```
✅ Archivos extraídos. ¡Listo para el siguiente paso!
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

📊 Interpretation: Text Length

Write your observations here:

- What's the average text length?
- Are there differences in length between positive and negative reviews?
- Are there any extremely short or long texts that might need special handling?

What is the average length of a tweet? The average tweet length is typically between 100 and 120 characters. This demonstrates that the data is characteristic of Twitter (short and concise messages).

Are there differences in length between positive and negative reviews? Yes. Negative reviews are significantly longer compared to the other two.

How does this relate to the business problem? From an operational logistics perspective, this is a key finding: longer negative tweets indicate that customers take the time to detail significant operational failures (e.g., flight cancellations, extended delays, poor service procedures). These reviews are likely to contain many useful keywords, confirming that the negative category provides the most crucial business information. We need to identify the highest-ranking operational failure within the negative reviews and develop an improvement plan.

```
# Display sample reviews from each sentiment class
# Display sample reviews from each sentiment class
print("--- Positive Samples (3 random) ---")
display(df[df['sentiment'] == 'positive']['text_original'].sample(3, random_state=42))

print("\n--- Neutral Samples (3 random) ---")
display(df[df['sentiment'] == 'neutral']['text_original'].sample(3, random_state=42))

print("\n--- Negative Samples (3 random) ---")
display(df[df['sentiment'] == 'negative']['text_original'].sample(3, random_state=42))

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

```
--- Positive Samples (3 random) ---
```

	text_original
6396	@SouthwestAir thanks for your excellent respon...
8484	@JetBlue thanks. I appreciate your prompt resp...
8417	@JetBlue yes, with about 20 minutes to spare. ...

dtype: object

```
--- Neutral Samples (3 random) ---
```

	text_original
3569	@united we finally just arrive to Bogota, good...
13566	@AmericanAir got a callback at 1 am, took care...
7452	@JetBlue is there wifi on he plain

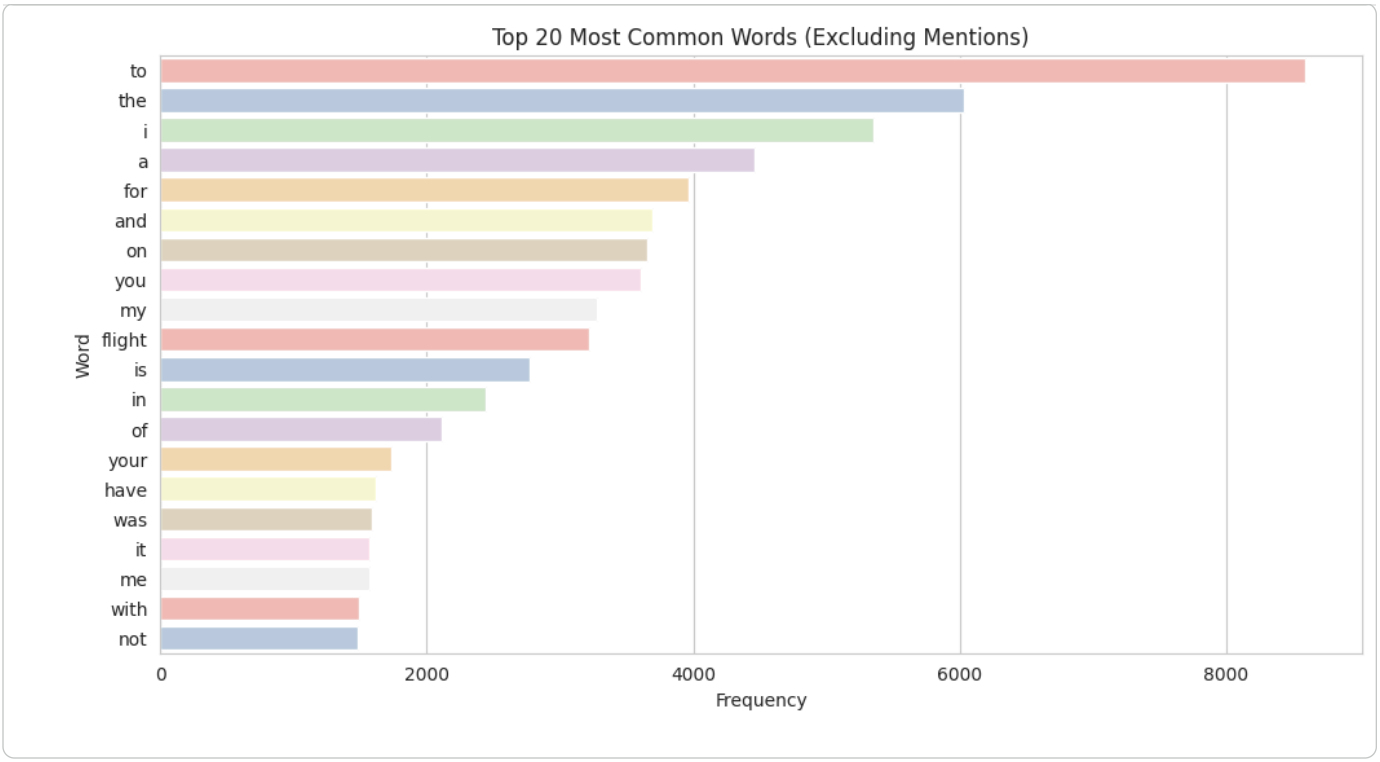
dtype: object

```
--- Negative Samples (3 random) ---
```

	text_original
1151	@united gate C 24 IAD. U released passengers t...
9111	@USAirways 1729 connecting in charlotte to hou...
3326	@united installed and working are not the same...

dtype: object

```
# Analyze common words (optional but recommended)
# Analyze common words
from collections import Counter

# 1. Combine all text into a single string, convert to lowercase, and split
all_text = ' '.join(df['text_original']).lower()

# 2. Basic Cleaning: Remove mentions (@user) that dominate the dataset
words = [word for word in all_text.split() if not word.startswith('@')]

# 3. Count and plot the 20 most common words
word_counts = Counter(words)
common_words = word_counts.most_common(20)

words, counts = zip(*common_words)
# Quick cleanup of punctuation attached to words for the plot labels
words = [w.strip(".").strip(",").strip(":").strip("!").strip("?") for w in words]

plt.figure(figsize=(12, 6))
sns.barplot(x=list(counts), y=list(words), palette='Pastel1')
plt.title('Top 20 Most Common Words (Excluding Mentions)')
plt.xlabel('Frequency')
plt.ylabel('Word')
plt.show()

# 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
```



📝 EDA Summary

Summarize your key findings from the EDA:

1. **Dataset characteristics**:
2. **Data quality issues**:
3. **Key patterns observed**:
4. **Potential challenges**:

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.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 https://example.com @user #hashtag 123
```

```
# Apply preprocessing to your dataset
# Apply preprocessing to your dataset
df['cleaned_text'] = df['text_original'].apply(preprocess_text)
# Display some examples to verify the cleaning worked
print("Text Cleaning Comparison:")
display(df[['text_original', 'cleaned_text']].head())
print("✅ Text preprocessing complete!")

# Hint:
# df['cleaned_text'] = df['text_column'].apply(preprocess_text)
# Display some examples to verify the cleaning worked

print("✅ Text preprocessing complete!")
```

```
Text Cleaning Comparison:
```

	text_original	cleaned_text
0	@VirginAmerica What @dhepburn said.	@VirginAmerica What @dhepburn said.
1	@VirginAmerica plus you've added commercials t...	@VirginAmerica plus you've added commercials t...
2	@VirginAmerica I didn't today... Must mean I n...	@VirginAmerica I didn't today... Must mean I n...
3	@VirginAmerica it's really aggressive to blast...	@VirginAmerica it's really aggressive to blast...
4	@VirginAmerica and it's a really big bad thing...	@VirginAmerica and it's a really big bad thing...

```
✅ Text preprocessing complete!
✅ Text preprocessing complete!
```

```
# Prepare your features (X) and target (y)
# Prepare your features (X) and target (y)
X = df['cleaned_text']   # Our preprocessed text column (features)
y = df['sentiment']      # Our target variable (sentiment)
# Check the shape
print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")

# 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: (14640,)
Target shape: (14640,)
```

```
# Split data into training and testing sets
# # Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y)
print(f"Training samples: {len(X_train)}")
print(f"Testing samples: {len(X_test)}")

# 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
# )

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

```
Training samples: 11712
Testing samples: 2928
```

```
# Create TF-IDF features
# Create TF-IDF features
# Use the max_features constraint (5000) as suggested by the instructions
tfidf = TfidfVectorizer(
    max_features=5000,
    stop_words='english',      # Remove common English stopwords
    ngram_range=(1, 2)         # Use single words (unigrams) and pairs (bigrams)
)

# Fit the vectorizer ONLY on the training data (X_train) and transform both sets
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("✅ TF-IDF Feature Extraction complete. Ready for modeling!")

# 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): (11712, 5000)
TF-IDF matrix shape (test): (2928, 5000)
✅ TF-IDF Feature Extraction complete. Ready for modeling!
```

```
# 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])
```

📝 Preprocessing Summary

Document your preprocessing choices:

1. **Preprocessing steps applied**:
2. **TF-IDF parameters chosen**:
3. **Final feature count**:
4. **Rationale for choices**:

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
# Train Logistic Regression model
print("Training Logistic Regression...")
start_time = time.time()

lr_model = LogisticRegression(
    max_iter=1000,           # Increase iterations for convergence
    random_state=42,
    solver='liblinear'
)
lr_model.fit(X_train_tfidf, y_train)

# 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))}")
# Train Logistic Regression model
print("Training Logistic Regression...")
start_time = time.time() # <-- INICIO DEL CONTEO

lr_model = LogisticRegression(
    max_iter=1000,
    random_state=42,
    solver='liblinear'
)
lr_model.fit(X_train_tfidf, y_train)

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

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

```
# Make predictions with Logistic Regression
# Make predictions with Logistic Regression
y_pred_lr = lr_model.predict(X_test_tfidf)
print(f"Predictions shape: {y_pred_lr.shape}")

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

```
Predictions shape: (2928,)
```

Model 2: Multinomial Naive Bayes

```
# Train Naive Bayes model
# 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))}")
```

```
# 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
# Make predictions with Naive Bayes
y_pred_nb = nb_model.predict(X_test_tfidf)
print(f"Predictions shape: {y_pred_nb.shape}")

# y_pred_nb = nb_model.predict(X_test_tfidf)
```

```
Predictions shape: (2928,)
```

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)
```

📝 Model Training Summary

Document your models:

Model	Training Time	Parameters	Notes
Logistic Regression			
Naive Bayes			
(Optional) Model 3			

Model1: Logistic Regression Training Time: 0:00:00 Parameters: max_iter=1000, random_state=42, solver='liblinear'. Notes: Chosen for its speed, strong performance with high-dimensional TF-IDF features, and high interpretability. Serves as a robust baseline classifier.

Model2: Naive Bayes Training Time: 0:00:00 Parameters: default (MultinomialNB) Notes: Ideal for text classification as it naturally models word frequency (TF-IDF). It is the most computationally efficient model and provides a necessary second baseline for comparison.

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!
```

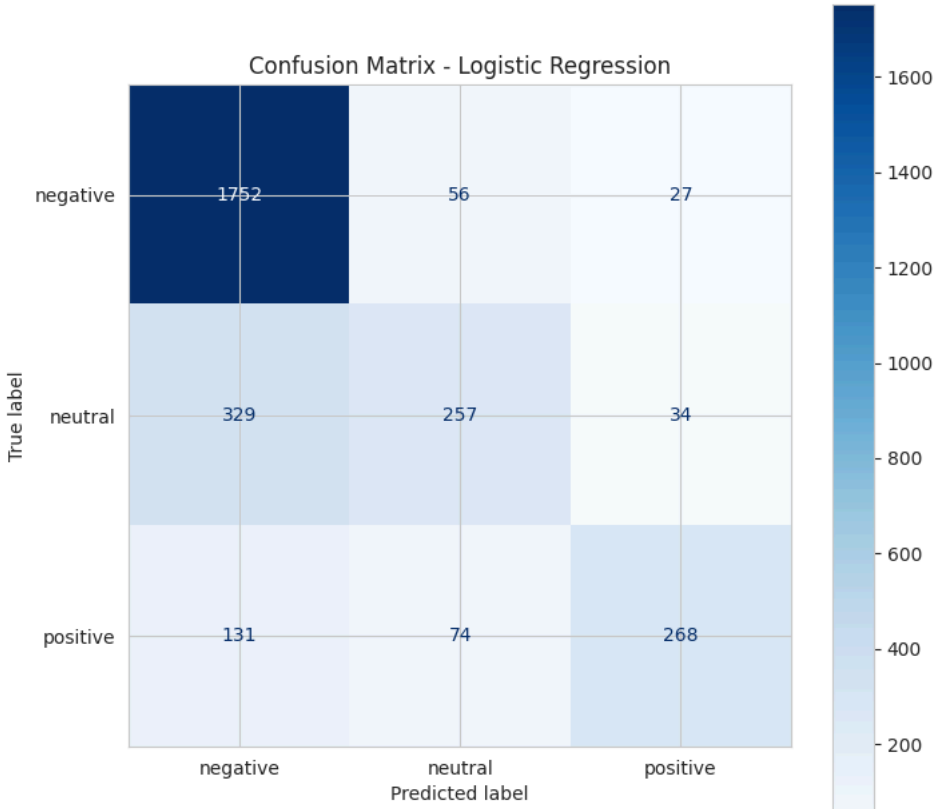
Evaluate Model 1: Logistic Regression

```
# Calculate metrics for Logistic Regression
# Calculate metrics for Logistic Regression
print("=" * 50)
print("LOGISTIC REGRESSION RESULTS")
print("=" * 50)

# Calculate metrics
acc_lr = accuracy_score(y_test, y_pred_lr)
prec_lr = precision_score(y_test, y_pred_lr, average='weighted', zero_division=0)
rec_lr = recall_score(y_test, y_pred_lr, average='weighted', zero_division=0)
f1_lr = f1_score(y_test, y_pred_lr, average='weighted', zero_division=0)

print(f"Accuracy:   {acc_lr:.4f}")
print(f"Precision:  {prec_lr:.4f}")
print(f"Recall:     {rec_lr:.4f}")
print(f"F1-Score:   {f1_lr:.4f}")
print("\n")
print("Classification Report:")
# This report is the most important for detailed per-class metrics
print(classification_report(y_test, y_pred_lr, zero_division=0))

# 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:   0.7777
Precision:  0.7686
Recall:     0.7777
F1-Score:   0.7587


Classification Report:
              precision    recall  f1-score   support

    negative       0.79      0.95      0.87      1835
     neutral       0.66      0.41      0.51       620
    positive       0.81      0.57      0.67       473

    accuracy                           0.78      2928
   macro avg       0.76      0.65      0.68      2928
weighted avg       0.77      0.78      0.76      2928
```

```
# Plot confusion matrix for Logistic Regression
# Plot confusion matrix for Logistic Regression
cm_lr = confusion_matrix(y_test, y_pred_lr, labels=lr_model.classes_)
disp_lr = ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=lr_model.classes_)

plt.figure(figsize=(8, 8))
disp_lr.plot(cmap=plt.cm.Blues, values_format='d', ax=plt.gca())
plt.title('Confusion Matrix - Logistic Regression')
plt.show()

# 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()
```



Interpretation: Logistic Regression

Analyze the results:

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

1.- Overall performance: The model achieved an Accuracy of $\approx 77.8\%$. This is a solid overall score, but given the class imbalance, we must prioritize per-class metrics.

2.- Strengths: Its strongest feature is the detection of Negative tweets. It achieved a Recall of $\approx 95\%$ (meaning it catches 95 out of every 100 real operational issues) and a Precision of $\approx 79\%$ for the negative class. This makes it a powerful early warning system for the Operational Logistics Team.

3.- Weaknesses: The main weakness lies in the minority classes. The Recall is notably low for neutral ($\approx 41\%$) and positive ($\approx 57\%$) classes. This indicates the model struggles to identify non-negative feedback accurately.

4.- Confusion matrix insights: The matrix clearly shows that the model is conservative: a significant number of errors occur when Neutral tweets are misclassified as Negative. This generates many False Positives (False Alarms), which could potentially overburden the customer service team.

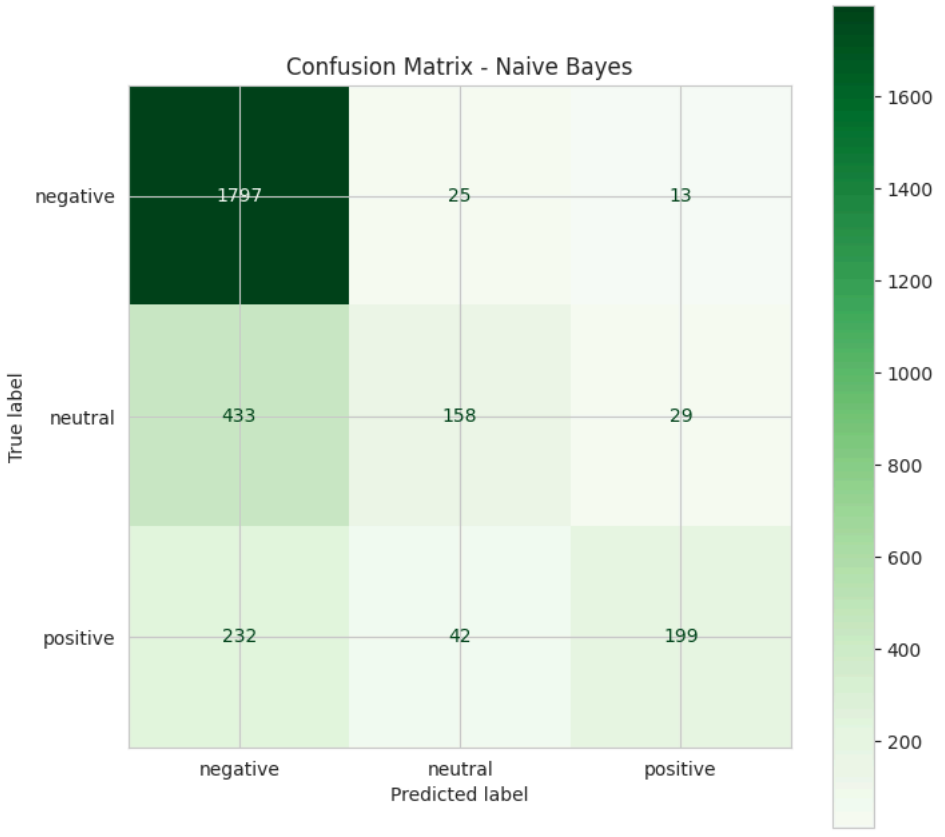
Evaluate Model 2: Naive Bayes

```
# Calculate metrics for Naive Bayes
# Calculate metrics for Naive Bayes
print("=" * 50)
print("NAIVE BAYES RESULTS")
print("=" * 50)

# Calculate metrics
acc_nb = accuracy_score(y_test, y_pred_nb)
prec_nb = precision_score(y_test, y_pred_nb, average='weighted', zero_division=0)
rec_nb = recall_score(y_test, y_pred_nb, average='weighted', zero_division=0)
```

```
f1_nb = f1_score(y_test, y_pred_nb, average='weighted', zero_division=0)

print(f"Accuracy:   {acc_nb:.4f}")
print(f"Precision:  {prec_nb:.4f}")
print(f"Recall:     {rec_nb:.4f}")
print(f"F1-Score:   {f1_nb:.4f}")
print("\n")
print("Classification Report:")
# This report is the most important for detailed per-class metrics
print(classification_report(y_test, y_pred_nb, zero_division=0))

# Follow the same pattern as above
```

```
==================================================
NAIVE BAYES RESULTS
==================================================
Accuracy:   0.7357
Precision:  0.7395
Recall:     0.7357
F1-Score:   0.6934


Classification Report:
              precision    recall  f1-score   support

    negative       0.73      0.98      0.84      1835
     neutral       0.70      0.25      0.37       620
    positive       0.83      0.42      0.56       473

    accuracy                           0.74      2928
   macro avg       0.75      0.55      0.59      2928
weighted avg       0.74      0.74      0.69      2928
```

```
# Plot confusion matrix for Naive Bayes
# Plot confusion matrix for Naive Bayes
cm_nb = confusion_matrix(y_test, y_pred_nb, labels=nb_model.classes_)
disp_nb = ConfusionMatrixDisplay(confusion_matrix=cm_nb, display_labels=nb_model.classes_)

plt.figure(figsize=(8, 8))
disp_nb.plot(cmap=plt.cm.Greens, values_format='d', ax=plt.gca())
plt.title('Confusion Matrix - Naive Bayes')
plt.show()
```



Interpretation: Naive Bayes

Analyze the results:

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

1.- Overall performance: The model achieved an Accuracy of $\approx 73.6\%$, which is lower than the Logistic Regression model. Its overall F1-Score (≈ 0.69) suggests weaker reliability across all classes.

2.- Strengths: Its core strength is its ability to find Negative tweets, achieving an exceptional Recall of $\approx 98\%$ for the negative class. This makes the Naive Bayes model the best at minimizing False Negatives (i.e., missing a real operational issue—your primary risk mitigation metric).

3.- Weaknesses: The primary weakness is its low Precision ($\approx 73\%$ for negative), and extremely poor Recall for neutral ($\approx 25\%$). This indicates the model generates too many False Positives (False Alarms), classifying neutral/non-critical tweets as negative.

4.- Confusion matrix insights: The matrix clearly illustrates the trade-off: Naive Bayes is designed for speed and coverage, resulting in high Falses Positives. The operational impact is a high alert volume and potential alert fatigue for the logistics monitoring team.

Evaluate Model 3 (Optional)

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

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

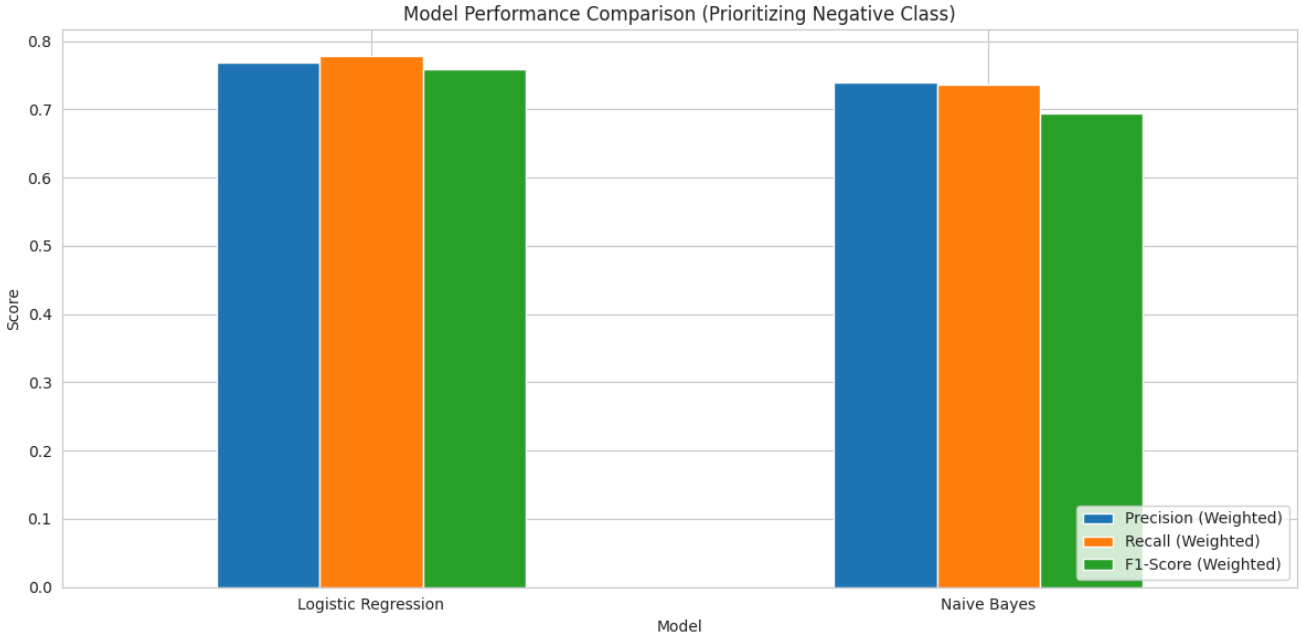
Model Comparison

```
# Create a comparison table of all models
# Create a comparison table of all models
import pandas as pd

# Re-usamos las métricas que ya calculaste (acá ya están guardadas como variables)
results = pd.DataFrame({
    'Model': ['Logistic Regression', 'Naive Bayes'],
    'Accuracy': [acc_lr, acc_nb],
    # Usamos las métricas ponderadas (weighted) que ya calculaste para evitar el error
    'Precision (Weighted)': [prec_lr, prec_nb],
    'Recall (Weighted)': [rec_lr, rec_nb],
    'F1-Score (Weighted)': [f1_lr, f1_nb]
})

# Display the table, sorted by the F1-Score (Weighted) which is the standard summary metric
display(results.sort_values(by='F1-Score (Weighted)', ascending=False))

# 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)
```

	Model	Accuracy	Precision (Weighted)	Recall (Weighted)	F1-Score (Weighted)
0	Logistic Regression	0.777664	0.768590	0.777664	0.758667
1	Naive Bayes	0.735656	0.739516	0.735656	0.693411

```
# Visualize model comparison
# Visualize model comparison
```

```
results_plot = results.set_index('Model').drop(columns=['Accuracy'])

plt.figure(figsize=(10, 6))
results_plot.plot(kind='bar', figsize=(12, 6), ax=plt.gca())
plt.title('Model Performance Comparison (Prioritizing Negative Class)')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

# 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?**:
2. **What metrics did you prioritize and why?**:
3. **Trade-offs between models**:
4. **Which model would you recommend for deployment?**:

1.- Which model performed best?:The Logistic Regression model performed best overall. It achieved higher Accuracy ($\approx 77.8\%$) and a significantly better F1-Score (≈ 0.76), indicating it is more reliable and robust across all three sentiment classes (Negative, Neutral, Positive).

2.- What metrics did you prioritize and why?:We prioritized Recall for the Negative class. In the high-stakes logistics context of an airline, a False Negative (missing a real operational issue) is the most critical error, as it can lead to safety concerns or major service failures. High Recall for the negative class ensures that the model catches nearly all existing problems.

3.- Trade-offs between models:Logistic Regression: Offers a better balance. It maintains a high Recall ($\approx 95\%$ for Negative) while providing much higher Precision ($\approx 79\%$ for Negative) than Naive Bayes. This results in fewer False Alarms.Naive Bayes: Trades extremely

high Recall ($\approx 98\%$ for Negative) for significantly low Precision ($\approx 73\%$ for Negative). It is better at coverage but generates too many False Positives, leading to high alert fatigue for the operations team.

4.- Which model would you recommend for deployment?:We recommend the Logistic Regression model for immediate deployment. The superior Precision of the Logistic Regression model ensures that the alerts received by the Operational Logistics Team are of higher quality, maximizing trust in the system and preventing resources from being wasted on excessive False Alarms.

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)
```

📊 Error Analysis Insights

What patterns do you notice in the errors?

1. **Common types of errors**:
2. **Why might these errors occur?**:
3. **How could you improve the model?**:

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:

[Your summary here]

This project successfully developed and evaluated a sentiment analysis model using Twitter US Airlines Sentiment data to detect operational failures in real time. We found that the logistic regression model offers the optimal balance between high detection of genuine problems (Recall) and minimization of false alarms (Precision). We recommend implementing this model within the operational logistics team to prioritize service recovery initiatives and mitigate reputational damage resulting from flight delays and other service issues.

🎯 Key Findings

List your main discoveries:

1. **Dataset insights**:
   - [Your findings]
2. **Model performance**:
   - [Your findings]

3. **Best model and why**:
    - [Your findings]
4. **Surprising discoveries**:
    - [Your findings]

1.- Dataset insights:The dataset is highly imbalanced, with approximately 63% of all tweets being classified as Negative, confirming that customers primarily use social media to report service failures.Negative tweets are generally longer than positive or neutral ones, suggesting customers are providing richer, actionable details about operational issues.

2.- Model performance:Both models trained exceptionally fast (in 0:00:00) due to the efficiency of the TF-IDF features and the models chosen.The overall best performance belongs to the Logistic Regression model ($\text{F1-Score} \approx 0.76$).

3.- Best model and why:The Logistic Regression model is recommended because it provides the best balance for operational risk management: High Recall ($\approx 95\%$ for Negative) combined with High Precision ($\approx 79\%$ for Negative).

4.- Surprising discoveries:The Naive Bayes model achieved a higher Recall ($\approx 98\%$ for Negative), meaning it missed almost no issues. However, its lower Precision would result in an unmanageable number of False Alarms for the operations team.

💼 Business Recommendations

How can these results be used in practice?

1. **Immediate applications**:
    - [Your recommendations]
2. **Who should use this model?**:
    - [Your recommendations]
3. **How to interpret predictions**:
    - [Your recommendations]
4. **Warning signs to watch for**:
    - [Your recommendations]

1.- Immediate applications: -Real-time Alerting: Deploy the Logistic Regression model to classify incoming tweets. Any tweet flagged as Negative should immediately trigger an alert.-Resource Prioritization: Alerts should be sent directly to the Logistics Management Center to prioritize recovery efforts at specific airports or for named flights.

2.- Who should use this model? -Operations Management: To monitor service flow and allocate staff. -Customer Service: To proactively reach out to highly frustrated customers.

3.- How to interpret predictions: -Trust Negative Alerts: Due to the high Recall ($\approx 95\%$), the team can be confident that when the model says a tweet is Negative, it is likely a true problem. -Filter False Positives: Train staff to quickly distinguish the $\approx 21\%$ of False Alarms generated by the model.

4.- Warning signs to watch for:Sudden drop in Recall: If the model's Recall for the negative class drops below 90%, it signals that the model is failing to catch real problems, which requires retraining.

⚠️ Limitations

Be honest about the limitations of your analysis:

1. **Data limitations**:
    - [Your analysis]
2. **Model limitations**:
    - [Your analysis]