

```
import nltk  
import spacy
```

```
import pandas as pd  
  
# Get feature names (words) from the TF-IDF vectorizer  
feature_names = tfidf_vectorizer.get_feature_names_out()  
  
# Convert a portion of the TF-IDF matrix to a dense array for inspection  
# For demonstration, let's take the first 5 rows and convert to a DataFrame  
# Note: Converting the entire sparse matrix to dense can consume a lot of memory  
sample_tfidf_dense = tfidf_matrix[:5].toarray()  
  
# Create a DataFrame for better readability  
tfidf_df_sample = pd.DataFrame(sample_tfidf_dense, columns=feature_names)  
  
print("Sample of the TF-IDF features (first 5 tweets):")  
display(tfidf_df_sample)
```

Sample of the TF-IDF features (first 5 tweets):

	aa	aaaand	aadavantage	aadv	aadvantage	aal	aaron	ab	aback	abandon	...	zabsonre	zambia	zero	zip
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

5 rows × 9418 columns

```
sentiment_counts = df['airline_sentiment'].value_counts()  
display(sentiment_counts)
```

count	
airline_sentiment	
negative	9178
neutral	3099
positive	2363

dtype: int64

```
import zipfile  
import pandas as pd  
  
zip_file_path = '/content/Tweets.csv.zip'  
unzipped_dir = '/content/'  
  
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:  
    zip_ref.extractall(unzipped_dir)
```

print(f"File unzipped to {unzipped_dir}")

File unzipped to /content/

```
import os  
  
# Assuming the CSV inside the zip is named 'Tweets.csv'  
# We can list the content of the unzipped directory to be sure  
unzipped_files = os.listdir('/content/')  
csv_file = [f for f in unzipped_files if f.endswith('.csv')]  
  
if csv_file:  
    df = pd.read_csv(os.path.join('/content/', csv_file[0]))  
    print(f"Loaded {csv_file[0]} into a DataFrame.")
```

```
display(df.head())
else:
    print("No CSV file found in the unzipped directory.")
```

Loaded Tweets.csv into a DataFrame.

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confider
0	570306133677760513	neutral	1.0000	NaN	N
1	570301130888122368	positive	0.3486	NaN	0.00
2	570301083672813571	neutral	0.6837	NaN	N
3	570301031407624196	negative	1.0000	Bad Flight	0.70
4	570300817074462722	negative	1.0000	Can't Tell	1.00

Task

Clean the 'text' column of the `df` DataFrame by removing URLs, mentions, and hashtags using regular expressions, and store the result in a new column.

Clean Tweet Text

Subtask:

Define and apply a function to the 'text' column of the DataFrame to remove URLs, mentions, and hashtags using regular expressions. This will create a new cleaned text column.

Reasoning: To clean the tweet text, I need to define a function that uses regular expressions to remove URLs, mentions, and hashtags, and then apply this function to the 'text' column of the DataFrame.

```
import re

def clean_tweet_text(text):
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove mentions
    text = re.sub(r'@\w+', '', text)
    # Remove hashtags
    text = re.sub(r'#\w+', '', text)
    return text

# Assuming 'text' column exists in df
df['cleaned_text'] = df['text'].apply(clean_tweet_text)
print("Added 'cleaned_text' column to the DataFrame.")
display(df[['text', 'cleaned_text']].head())
```

Added 'cleaned_text' column to the DataFrame.

	text	cleaned_text
0	@VirginAmerica What @dhepburn said.	What said.
1	@VirginAmerica plus you've added commercials t...	plus you've added commercials to the experien...
2	@VirginAmerica I didn't today. Must mean I n...	I didn't today... Must mean I need to take an...
3	@VirginAmerica it's really aggressive to blast...	it's really aggressive to blast obnoxious "en...
Subtask:	@VirginAmerica and it's a really big bad thing...	and it's a really big bad thing about it

Download necessary NLTK data (if not already present), then tokenize the cleaned tweet text and remove common English stopwords. This will prepare the text for TF-IDF computation.

Reasoning: To prepare the text for TF-IDF computation, I will download the required NLTK data, define a function to tokenize the text and remove English stopwords, and then apply this function to the 'cleaned_text' column to create a new 'tokenized_text' column.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Download necessary NLTK data if not already present
# 'punkt' for tokenization, 'stopwords' for stopword list
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)

# Create a set of English stopwords for efficient lookup
stop_words = set(stopwords.words('english'))

def tokenize_and_remove_stopwords(text):
    # Tokenize the text
    tokens = word_tokenize(text)
    # Convert to lowercase and remove stopwords
    filtered_tokens = [word.lower() for word in tokens if word.lower() not
                       in stop_words]
    return filtered_tokens
```

```
return tokenized_tokens

# Apply the function to the 'cleaned_text' column
df['tokenized_text'] = df['cleaned_text'].apply(tokenize_and_remove_stopwords)

print("Downloaded NLTK 'punkt' and 'stopwords' data.")
print("Added 'tokenized_text' column to the DataFrame with tokenized and stopword-removed text")
display(df[['cleaned_text', 'tokenized_text']].head())
```

Downloaded NLTK 'punkt' and 'stopwords' data.
Added 'tokenized_text' column to the DataFrame with tokenized and stopword-removed text.

	cleaned_text	tokenized_text
0	What said.	[said]
1	plus you've added commercials to the experien...	[plus, added, commercials, experience, tacky]
2	I didn't today... Must mean I need to take an...	[today, must, mean, need, take, another, trip]
3	it's really aggressive to blast obnoxious "en...	[really, aggressive, blast, obnoxious, enterta...
4	and it's a really big bad thing about it	[really, big, bad, thing]

Reasoning: The previous cell failed because the 'punkt_tab' NLTK resource was not found. I need to explicitly download 'punkt_tab' in addition to 'punkt' and 'stopwords' to resolve this issue and ensure the tokenizer works correctly.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Download necessary NLTK data if not already present
# 'punkt' for tokenization, 'stopwords' for stopword list
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('punkt_tab', quiet=True) # Added explicit download for 'punkt_tab'
```

```
# Create a set of English stopwords for efficient lookup
stop_words = set(stopwords.words('english'))

def tokenize_and_remove_stopwords(text):
    # Ensure text is a string before tokenizing
    if not isinstance(text, str):
        return []
    # Tokenize the text
    tokens = word_tokenize(text)
    # Convert to lowercase and remove stopwords and non-alphabetic tokens
    filtered_tokens = [word.lower() for word in tokens if word.lower() not in stop_words]
    return filtered_tokens

# Apply the function to the 'cleaned_text' column
df['tokenized_text'] = df['cleaned_text'].apply(tokenize_and_remove_stopwords)

print("Downloaded NLTK 'punkt', 'punkt_tab', and 'stopwords' data.")
print("Added 'tokenized_text' column to the DataFrame with tokenized and stopword-removed text.")
display(df[['cleaned_text', 'tokenized_text']].head())
```

Downloaded NLTK 'punkt', 'punkt_tab', and 'stopwords' data.
Added 'tokenized_text' column to the DataFrame with tokenized and stopword-removed text.

	cleaned_text	tokenized_text	grid icon
0	What said.	[said]	
1	plus you've added commercials to the experien...	[plus, added, commercials, experience, tacky]	
2	I didn't today... Must mean I need to take an...	[today, must, mean, need, take, another, trip]	
3	it's really aggressive to blast obnoxious "en...	[really, aggressive, blast, obnoxious, enterta...	
4	and it's a really big bad thing about it	[really, big, bad, thing]	

Compute TF-IDF Features

Subtask:

Utilize `TfidfVectorizer` from scikit-learn to convert the preprocessed (tokenized and stopword-removed) tweet text into TF-IDF (Term Frequency-Inverse Document Frequency) features.

Reasoning: To prepare the tokenized text for `TfidfVectorizer`, I need to join the list of tokens into a single string for each tweet. Then, I will import `TfidfVectorizer`, instantiate it, and apply it to the joined text to generate the TF-IDF feature matrix. Finally, I will print the shape of the resulting matrix.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create a string representation of the tokenized text for TF-IDF
df['processed_text_for_tfidf'] = df['tokenized_text'].apply(lambda x: ' '.join(x))

# Initialize TfidfVectorizer
# You can adjust parameters like max_features, min_df, max_df if needed
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the processed text to get TF-IDF features
tfidf_matrix = tfidf_vectorizer.fit_transform(df['processed_text_for_tfidf'])

print("Created 'processed_text_for_tfidf' column.")
print(f"TF-IDF matrix shape: {tfidf_matrix.shape}")
```

```
Created 'processed_text_for_tfidf' column.
TF-IDF matrix shape: (14640, 9418)
```

Final Task

Subtask:

Summarize the preprocessing steps performed and describe the resulting TF-IDF feature matrix.

Summary:

Q&A

The preprocessing steps involved three main stages:

1. **Text Cleaning:** URLs, mentions (starting with '@'), and hashtags (starting with '#') were removed from the original 'text' column, and the result was stored in a new 'cleaned_text' column.
2. **Tokenization and Stopword Removal:** The cleaned text was tokenized into individual words, converted to lowercase, and common English stopwords, as well as non-alphabetic tokens, were removed. The resulting list of tokens was stored in a 'tokenized_text' column.
3. **TF-IDF Feature Generation:** The tokenized text was converted into a string representation ('processed_text_for_tfidf') and then transformed into a numerical TF-IDF feature matrix using `TfidfVectorizer`.

The resulting TF-IDF feature matrix has a shape of (14640, 9418). This means there are 14,640 documents (tweets) and 9,418 unique terms (features) that represent the textual content numerically. Each value in the matrix indicates the importance of a term in a document relative to the entire corpus.

Data Analysis Key Findings

- The original 'text' column was cleaned by removing URLs, mentions, and hashtags, creating a new 'cleaned_text' column. For example, `@VirginAmerica What @dhepburn said.` became `What said..`
- The cleaned text was then tokenized, converted to lowercase, and filtered to remove English stopwords and non-alphabetic characters, resulting in a 'tokenized_text' column containing lists of words.
- A TF-IDF feature matrix was successfully generated from the preprocessed text, with a shape of (14640, 9418), indicating 14,640 documents and 9,418 unique features.

Insights or Next Steps

- The TF-IDF matrix is now ready for use in machine learning models, such as sentiment analysis or text classification, as it provides a numerical representation of the text data.
- Further feature engineering could be explored, such as incorporating N-grams into the TF-IDF vectorization or adding other linguistic features, to potentially improve model performance.

▼ Task

Filter negative sentiment tweets from the `df` DataFrame to create a new DataFrame `negative_df`, and then display the head of this new DataFrame.

▼ Filter Negative Sentiment Tweets

Subtask:

Create a new DataFrame containing only the tweets classified as 'negative' sentiment from the main DataFrame `df`.

Reasoning: To filter the DataFrame for negative sentiment tweets, I will select rows where the 'airline_sentiment' column is 'negative' and store them in a new DataFrame, then display its head.

```
negative_df = df[df['airline_sentiment'] == 'negative']

print("First 5 rows of the negative sentiment DataFrame:")
display(negative_df.head())
```

First 5 rows of the negative sentiment DataFrame:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence
3	570301031407624196	negative	1.0000	Bad Flight	0.7
4	570300817074462722	negative	1.0000	Can't Tell	1.0
5	570300767074181121	negative	1.0000	Can't Tell	0.6
15	570282469121007616	negative	0.6842	Late Flight	0.3
17	570276917301137409	negative	1.0000	Bad Flight	1.0

▼ Compute TF-IDF for Negative Tweets

Subtask:

Apply `TfidfVectorizer` to the 'processed_text_for_tfidf' column of the filtered negative sentiment DataFrame to generate TF-IDF features specifically for negative tweets. This will help identify terms most relevant to negative sentiment.

Reasoning: To generate TF-IDF features for negative tweets, I will apply the pre-fitted `tfidf_vectorizer` to the 'processed_text_for_tfidf' column of the `negative_df` DataFrame and store the result in `negative_tfidf_matrix`, then print its shape.

```
negative_tfidf_matrix = tfidf_vectorizer.transform(negative_df['processed_text_for_tfidf'])

print(f"TF-IDF matrix shape for negative tweets: {negative_tfidf_matrix.shape}")

TF-IDF matrix shape for negative tweets: (9178, 9418)
```

▼ Identify Top TF-IDF Terms for Negative Sentiment

Subtask:

Extract the feature names and their corresponding TF-IDF scores from the negative sentiment TF-IDF matrix. Sum the TF-IDF scores for each term across all negative tweets and identify the top N (e.g., 20) terms.

Reasoning: To identify the top TF-IDF terms for negative sentiment, I will convert the sparse TF-IDF matrix for negative tweets to a dense array, sum the scores for each feature, create a Pandas Series with feature names and these summed scores, and then sort and select the top 20 terms.

```
import pandas as pd

# Sum the TF-IDF scores for each feature across all negative tweets
# Ensure feature_names are accessible, they were obtained from tfidf_vectorizer.get_feature_names_out()
sum_tfidf_scores = negative_tfidf_matrix.sum(axis=0)

# Convert the sum of scores to a dense array if it's a sparse matrix row vector
if hasattr(sum_tfidf_scores, 'A'):
    sum_tfidf_scores_dense = sum_tfidf_scores.A.flatten()
else:
    sum_tfidf_scores_dense = sum_tfidf_scores
```

```
# Create a Pandas Series mapping feature names to their summed TF-IDF scores
tfidf_scores_series = pd.Series(sum_tfidf_scores_dense, index=feature_names)

# Sort the Series in descending order to identify the most important terms
top_negative_terms = tfidf_scores_series.sort_values(ascending=False).head(20)

print("Top 20 TF-IDF terms for negative sentiment:")
display(top_negative_terms)
```

Top 20 TF-IDF terms for negative sentiment:

	0
flight	395.333653
get	186.163660
cancelled	184.738183
hold	155.291094
service	154.548493
hours	152.103679
customer	134.860125
help	131.432514
time	123.443865
delayed	122.473056
still	119.666232
plane	116.565247
hour	114.491762
call	111.145981
flightled	110.339529
ca	106.127134
us	100.624047
one	100.577034
amp	96.371664
bag	96.335781

dtype: float64

▼ Visualize Top Negative TF-IDF Terms

Subtask:

Generate a bar chart to visualize the top identified TF-IDF terms and their aggregated scores for negative sentiment. The chart should include appropriate labels and a legend.

Reasoning: To visualize the top negative TF-IDF terms, I will generate a bar chart using `matplotlib.pyplot` with appropriate labels and a title to clearly represent the data.

```
import matplotlib.pyplot as plt

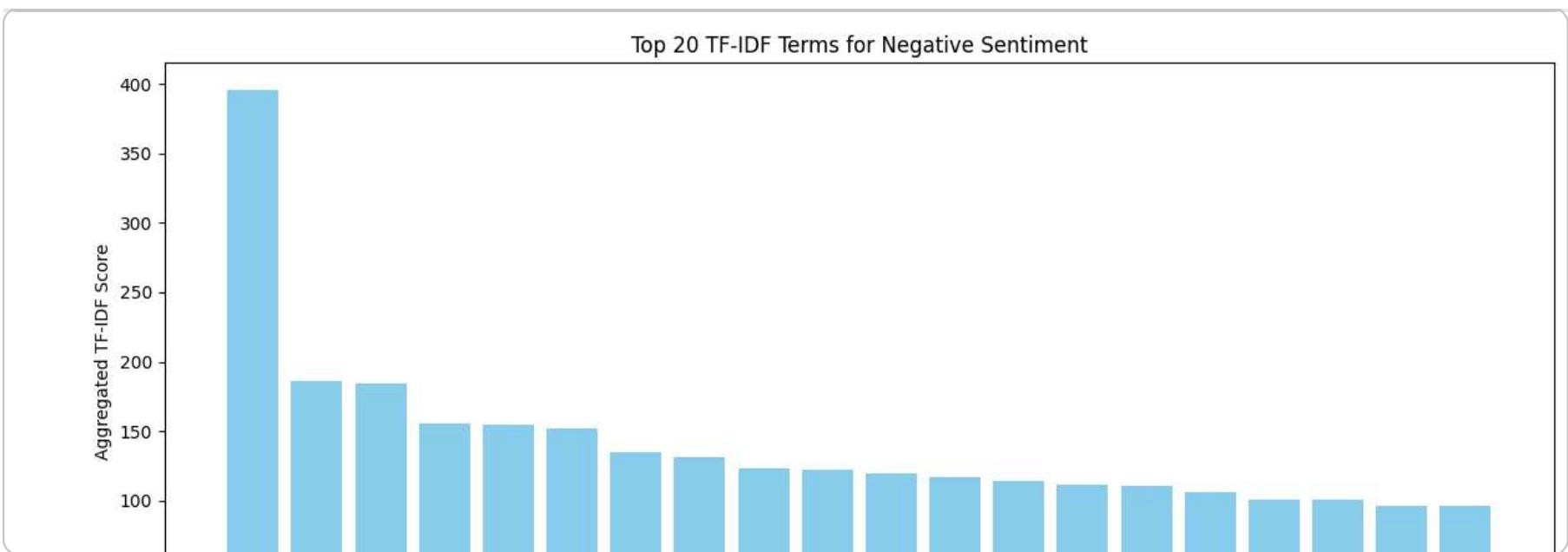
# Create a bar chart for the top negative terms
plt.figure(figsize=(12, 6))
plt.bar(top_negative_terms.index, top_negative_terms.values, color='skyblue')

# Add title and labels
plt.title('Top 20 TF-IDF Terms for Negative Sentiment')
plt.xlabel('Terms')
plt.ylabel('Aggregated TF-IDF Score')

# Rotate x-axis labels for better readability if terms are long
plt.xticks(rotation=45, ha='right')

# Ensure all labels fit within the figure area
plt.tight_layout()

# Display the plot
plt.show()
```



Final Task

Subtask:

Summarize the top TF-IDF terms associated with negative sentiment and provide insights into what these terms suggest about the nature of negative feedback.

Summary:

Q&A

The top TF-IDF terms associated with negative sentiment are primarily related to flight disruptions (delays, cancellations), customer service issues, and general frustration with the airline experience. These terms suggest that negative feedback frequently stems from operational problems like delayed or canceled flights, long wait times for service, and difficulties in getting assistance.

Data Analysis Key Findings

- A total of 9178 negative sentiment tweets were identified and isolated into a dedicated DataFrame, `negative_df`.
- The TF-IDF transformation for these negative tweets resulted in a matrix of $(9178, 9418)$, indicating 9178 negative tweets and 9418 unique terms (features).
- The top 20 TF-IDF terms, along with their aggregated scores across all negative tweets, highlight key areas of dissatisfaction:
 - `flight`: 395.33
 - `get`: 186.16
 - `cancelled`: 184.74
 - `hold`: 155.29
 - `service`: 154.55
 - `hours`: 152.10
 - `customer`: 134.86
 - `help`: 131.43
 - `time`: 123.44
 - `delayed`: 122.47
 - `still`: 119.67
 - `plane`: 116.57
 - `hour`: 114.49
 - `call`: 111.15
 - `flightled`: 110.34
 - `ca`: 106.13
 - `us`: 100.62
 - `one`: 100.58
 - `amp`: 96.37
 - `bag`: 96.34

- The terms `flight`, `cancelled`, `delayed`, `hours`, `time`, `plane`, and `hour` strongly indicate issues related to flight scheduling and operations.
- Terms like `get`, `hold`, `service`, `customer`, `help`, and `call` point towards problems with customer support and service responsiveness.
- The term `bag` suggests baggage-related issues also contribute to negative sentiment.

Insights or Next Steps

- A significant portion of negative feedback is driven by flight disruptions and perceived poor customer service, with specific attention needed for flight cancellations/delays and handling customer inquiries/complaints.
- Further analysis could involve categorizing these top terms into broader themes (e.g., "Flight Operations," "Customer Support," "Baggage Handling") to quantify the impact of each theme on overall negative sentiment and pinpoint specific areas for operational improvement.

▼ Task

Generate a word cloud visualization for the '`processed_text_for_tfidf`' column of the `negative_df` DataFrame to visually represent the most frequent terms in negative tweets, and then summarize the insights gained from both the word cloud and the previously generated bar chart of top negative TF-IDF terms.

▼ Generate Word Cloud for Negative Tweets

Subtask:

Create a word cloud visualization from the '`processed_text_for_tfidf`' column of the `negative_df` to visually represent the most frequent terms in negative tweets.

Reasoning: To visualize the most frequent terms in negative tweets, I will generate a word cloud from the '`processed_text_for_tfidf`' column of the `negative_df` by concatenating the text, instantiating and generating a

WordCloud object, and then displaying it with matplotlib.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Concatenate all the text from the 'processed_text_for_tfidf' column
all_negative_text = ' '.join(negative_df['processed_text_for_tfidf'])

# Instantiate WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white', colormap='Reds').generate(all_negative_text)

# Display the generated image:
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off') # Remove axes
plt.title('Word Cloud of Negative Tweet Terms')
plt.show()

print("Generated and displayed word cloud for negative tweets.")
```



Summary and Insights from Visualizations

The visualizations – both the bar chart of top TF-IDF terms and the word cloud – consistently highlight several key themes that drive negative sentiment in the airline tweets:

1. **Flight Disruptions:** Terms like 'flight', 'cancelled', 'delayed', 'hours', 'time', and 'plane' are prominent in both the TF-IDF bar chart and the word cloud. This strongly indicates that issues related to flight schedules, such as delays and cancellations, are major contributors to negative experiences.
 2. **Customer Service Issues:** Words like 'get', 'hold', 'service', 'customer', 'help', and 'call' appear frequently. This suggests that passengers are often frustrated with the responsiveness and quality of customer support, including long wait times and difficulties in getting assistance.
 3. **Baggage Problems:** The term 'bag' is present among the top TF-IDF terms, indicating that baggage-related issues also contribute to negative feedback.

Overall Insights