

```
!unzip /content/Tweets.csv.zip -d /content/
```

```
Archive: /content/Tweets.csv.zip
inflating: /content/Tweets.csv
```

```
import pandas as pd

df = pd.read_csv('/content/Tweets.csv')
display(df.head())
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negative
0	570306133677760513	neutral		

1	570301130888122368	positive	
---	--------------------	----------	--

2	570301083672813571	neutral	
---	--------------------	---------	--

3	570301031407624196	negative	
---	--------------------	----------	--

4	570300817074462722	negative	
---	--------------------	----------	--

```
def __init__(font_path=r'./font/NotoColorEmoji-Regular.ttf',
            ranks_only=None, preference_func=None, max_stopwords=None, random_state=None,
            max_font_size=None, font_color='black', relative_scaling=1.0, colormap='viridis',
            contour_color='black', min_word_length=0, ccw=True)
```

[Open in tab](#) [View source](#)
Word cloud object for generating word clouds.

Parameters

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

# Download necessary NLTK data (if not already downloaded)
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
```

```

nltk.download('punkt')
# Explicitly download 'punkt_tab' if needed, as indicated by previous error
try:
    # This attempts to find a common component of punkt_tab to verify it's there
    nltk.data.find('tokenizers/punkt_tab/english/averaged_perceptron_tagger.pic')
except LookupError:
    nltk.download('punkt_tab')

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

    # 2. Tokenize and remove stopwords
    tokens = word_tokenize(text.lower()) # Tokenize and convert to lowercase
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.isalpha() and word not in stop_words]

    return " ".join(filtered_tokens)

df['processed_text'] = df['text'].apply(preprocess_text)

print("Original Text Sample:")
display(df[['text', 'processed_text']].head())

```

[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
Original Text Sample:

	text	processed_text	
0	@VirginAmerica What @dhepburn said.	said	
1	@VirginAmerica plus you've added commercials t...	plus added commercials experience tacky	
2	@VirginAmerica I didn't today... Must mean I n...	today must mean need take another trip	
3	@VirginAmerica it's really aggressive to blast...	really aggressive blast obnoxious entertainmen...	
4	@VirginAmerica and it's a really big bad thing...	really big bad thing	

```

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

    # 2. Tokenize and remove stopwords

```

def __init__(font_path=r'./font/Ubuntu-Regular.ttf', ranks_only=None, preference='frequency', max_stopwords=0, random_state=None, max_font_size=100, relative_scaling='auto', colormap='viridis', normalizer='l1', min_word_length=0, count_colormap=False, color_fn=None, **kwargs):
 self._font_path = font_path
 self._ranks_only = ranks_only
 self._preference = preference
 self._stopwords = stopwords.words('english')
 self._random_state = random_state
 self._max_font_size = max_font_size
 self._relative_scaling = relative_scaling
 self._colormap = colormap
 self._normalizer = normalizer
 self._min_word_length = min_word_length
 self._count_colormap = count_colormap
 self._color_fn = color_fn
 self._kwargs = kwargs

Word cloud object for generating word clouds.

Parameters

```

tokens = word_tokenize(text.lower()) # Tokenize and convert to lowercase
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.isalpha() and word not in stop_words]

return " ".join(filtered_tokens)

```

```
df['processed_text'] = df['text'].apply(preprocess_text)
```

```
print("Original Text Sample:")
display(df[['text', 'processed_text']].head())
```

Original Text Sample:

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

```

def __init__(font_path='C:/Windows/Fonts/Arial.ttf',
            ranks_only=None, prefer_color_func=None, max_stopwords=None, random_state=42,
            max_font_size=None, font_color='black', relative_scaling='auto', colormap='viridis', normalizer='l1',
            contour_color='black', min_word_length=0, cc_min_size=10, cc_max_size=100, cc_alpha=0.5):
    self._font_path = font_path
    self.ranks_only = ranks_only
    self._prefer_color_func = prefer_color_func
    self._max_stopwords = max_stopwords
    self._random_state = random_state
    self._max_font_size = max_font_size
    self._font_color = font_color
    self._relative_scaling = relative_scaling
    self._colormap = colormap
    self._normalizer = normalizer
    self._contour_color = contour_color
    self._min_word_length = min_word_length
    self._cc_min_size = cc_min_size
    self._cc_max_size = cc_max_size
    self._cc_alpha = cc_alpha

```

[Open in tab](#) [View source](#)

today must mean need take another trip
Word cloud object for generating word clouds.

Parameters

really aggressive blast
entertainmen...

really big bad thing

```

# 3. Compute TF-IDF on tweet text
tfidf_vectorizer = TfidfVectorizer(max_features=5000) # Limiting to 5000 features
tfidf_matrix = tfidf_vectorizer.fit_transform(df['processed_text'])

print("TF-IDF Matrix Shape:", tfidf_matrix.shape)
print("Sample TF-IDF Features (first 5 tweets, first 10 features):")
display(pd.DataFrame(tfidf_matrix[:5, :10].toarray(), columns=tfidf_vectorizer.get_feature_names()))

```

TF-IDF Matrix Shape: (14640, 5000)

Sample TF-IDF Features (first 5 tweets, first 10 features):

	aa	aadvantage	abandoned	abc	ability	able	aboard	abq	absolute	absolut
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
2	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
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
# Filter for negative sentiment tweets
```

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

```
# Transform the processed text of negative tweets using the *already fitted*
negative_tfidf_matrix = tfidf_vectorizer.transform(negative_tweets_df['proce
# Sum the TF-IDF scores for each term across all negative tweets
# The sum will give us an idea of the overall importance of each term in neg
# Use .A1 to ensure a 1-dimensional numpy array from the sparse matrix sum
sum_tfidf_scores = negative_tfidf_matrix.sum(axis=0).A1

# Get feature names (terms)
feature_names = tfidf_vectorizer.get_feature_names_out()

# Create a DataFrame to hold terms and their summed TF-IDF scores
term_scores_df = pd.DataFrame({'term': feature_names, 'tfidf_score': sum_tfidf_scores})

# Sort by TF-IDF score in descending order to get the top terms
top_negative_terms = term_scores_df.sort_values(by='tfidf_score', ascending=False, c

print("\nTop 20 TF-IDF terms for negative sentiment:")
display(top_negative_terms.head(20))
```

[Open in tab](#) [View source](#)
Word cloud object for ger

Parameters

Top 20 TF-IDF terms for negative sentiment:

	term	tfidf_score	grid
1716	flight	409.940860	
1883	get	192.412607	

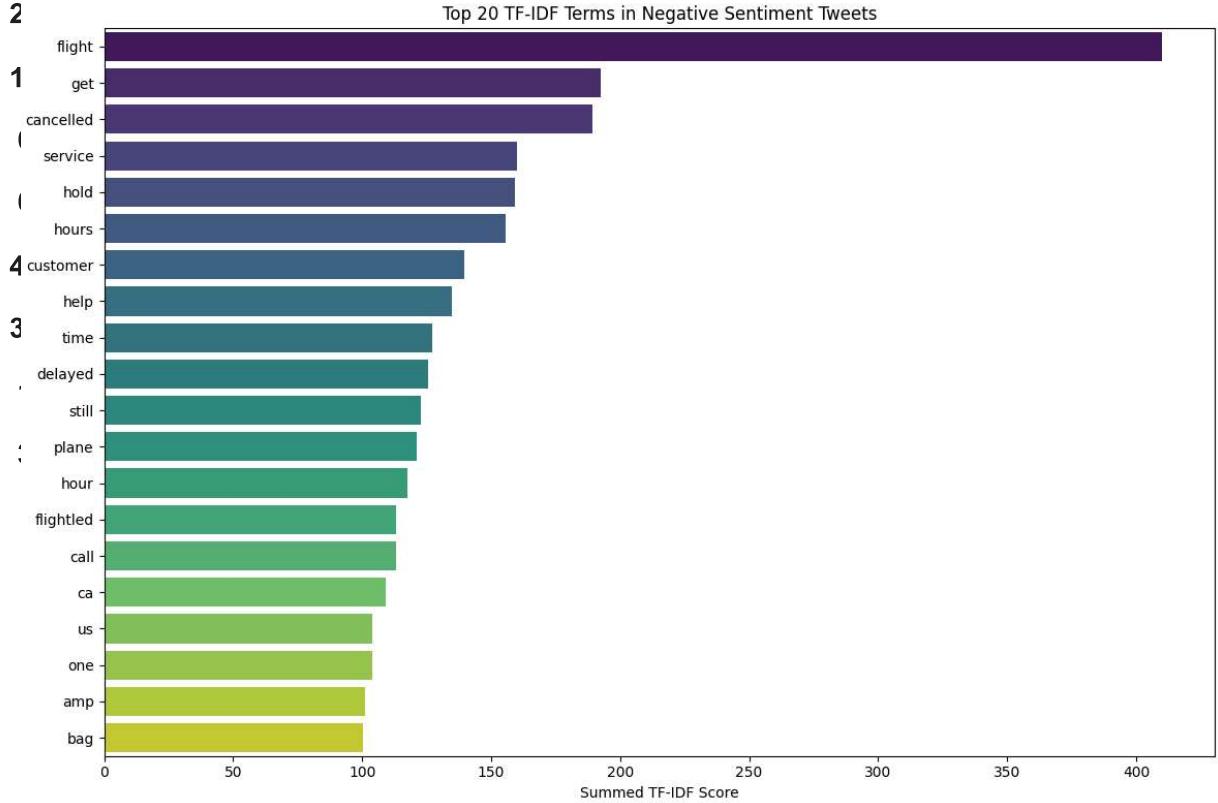
```
import matplotlib.pyplot as plt
import seaborn as sns

# Bar chart for top 20 terms
plt.figure(figsize=(12, 8))
sns.barplot(x='tfidf_score', y='term', data=top_negative_terms)
plt.title('Top 20 TF-IDF Terms in Negative Sentiment Tweets')
plt.xlabel('Summed TF-IDF Score')
plt.ylabel('Term')
plt.tight_layout()
plt.show()
```

1194 delayed 125.439821
 impipython-input-2168000755.py:6: FutureWarning:

4251 still 122.741445
 missing `palette` without assigning `hue` is deprecated and will be removed in v6.0.0

3260 plane 121.326106
 sns.barplot(x='tfidf_score', y='term', data=top_negative_terms.head(20), palette='viridis')



```
def __init__(font_path=None, ranks_only=None, prefer_color_func=None, max_stopwords=None, random_state=None, max_font_size=None, figure=None, palette='viridis', relative_scaling='auto', colormap=None, normalise_colours=True, contour_color='black', min_word_length=0, cc_min_word_length=0, **kwargs):
    self._font_path = font_path
    self._ranks_only = ranks_only
    self._prefer_color_func = prefer_color_func
    self._max_stopwords = max_stopwords
    self._random_state = random_state
    self._max_font_size = max_font_size
    self._figure = figure
    self._relative_scaling = relative_scaling
    self._colormap = colormap
    self._normalise_colours = normalise_colours
    self._contour_color = contour_color
    self._min_word_length = min_word_length
    self._cc_min_word_length = cc_min_word_length
    self._**kwargs = kwargs
```

[Open in tab](#) [View source](#)

Word cloud object for generating word clouds.

Parameters

```
from wordcloud import WordCloud
```

```
# Create a dictionary of terms and their TF-IDF scores for the word cloud
# We'll use all terms, but the word cloud will naturally emphasize higher scoring terms
```

```
wordcloud_data = dict(zip(top_negative_terms['term'], top_negative_terms['tf_idf']))

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white', color_func=lambda x: 'black' if x in negative_words else 'red')

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Top TF-IDF Terms in Negative Sentiment Tweets')
plt.show()
```

