```
Start coding or generate with AI.
```

Removing URLs, Mentions, and Hashtags

To clean text by removing URLs, mentions (e.g., `@user`), and hashtags (e.g., `#topic`), we can use Python's `re` (regular expression) module. I will first create some sample data and then apply the cleaning functions.

```
import pandas as pd
import re

# Create a sample DataFrame for demonstration
data = {
    'text': [
        'This is a tweet with a URL https://example.com and a mention @user1 and #hashtag1.',
        'Another text with @user2 and #another_hashtag but no URL.',
        'Just a regular sentence with no special elements.',
        'Check out this link: http://test.org and a double #hashtag2 #hashtag3',
        'Hey @user3, look at this! www.example.net'
    ]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
display(df)
```

```
Original DataFrame:
```

	text
0	This is a tweet with a URL https://example.com...
1	Another text with @user2 and #another_hashtag ...
2	Just a regular sentence with no special elements.
3	Check out this link: http://test.org and a dou...
4	Hey @user3, look at this! www.example.net

Next steps: Generate code with df New interactive sheet

Now, let's define functions to remove each type of element (URLs, mentions, and hashtags) using regular expressions.

```python
# Function to remove URLs
def remove_urls(text):
    # Regex to find URLs starting with http://, https://, or www.
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

# Function to remove mentions
def remove_mentions(text):
    # Regex to find mentions starting with @
    mention_pattern = re.compile(r'@\w+')
    return mention_pattern.sub(r'', text)

# Function to remove hashtags
def remove_hashtags(text):
    # Regex to find hashtags starting with #
    hashtag_pattern = re.compile(r'#\w+')
    return hashtag_pattern.sub(r'', text)

# Apply the cleaning functions sequentially
df['cleaned_text'] = df['text'].apply(remove_urls)
df['cleaned_text'] = df['cleaned_text'].apply(remove_mentions)
df['cleaned_text'] = df['cleaned_text'].apply(remove_hashtags)

print("DataFrame after removing URLs, mentions, and hashtags:")
display(df)
```

```
DataFrame after removing URLs, mentions, and hashtags:
                                          text                                    cleaned_text
0   This is a tweet with a URL https://example.com...  This is a tweet with a URL and a mention and .
1     Another text with @user2 and #another_hashtag ...             Another text with and but no URL.
2      Just a regular sentence with no special elements.  Just a regular sentence with no special elements.
3          Check out this link: http://test.org and a dou...            Check out this link: and a double
4                 Hey @user3, look at this! www.example.net                           Hey , look at this!
```

Next steps: Generate code with df New interactive sheet

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

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

# Download 'punkt_tab' specifically as it caused a LookupError
nltk.download('punkt_tab')

# Get English stopwords
stop_words = set(stopwords.words('english'))

# Function to tokenize text
def tokenize_text(text):
    return word_tokenize(text.lower())

# Function to remove stopwords
def remove_stopwords(tokens):
    return [word for word in tokens if word.isalpha() and word not in stop_words]

# Apply tokenization and stopword removal
df['tokenized_text'] = df['cleaned_text'].apply(tokenize_text)
df['processed_text'] = df['tokenized_text'].apply(remove_stopwords)

print("DataFrame after tokenization and stopword removal:")
display(df.head())
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
DataFrame after tokenization and stopword removal:
```

	text	cleaned_text	tokenized_text	processed_text
0	This is a tweet with a URL https://example.com...	This is a tweet with a URL and a mention and .	[this, is, a, tweet, with, a, url, and, a, men...	[tweet, url, mention]
1	Another text with @user2 and #another_hashtag ...	Another text with and but no URL.	[another, text, with, and, but, no, url, .]	[another, text, url]
2	Just a regular sentence with no special	Just a regular sentence with no	[just, a, regular, sentence, with,	[regular, sentence, special,

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Convert lists of processed words back into strings for TF-IDF Vectorizer
df['text_for_tfidf'] = df['processed_text'].apply(lambda x: ' '.join(x))

# Initialize TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=100) # You can adjust max_features

# Fit and transform the text data
tfidf_matrix = tfidf_vectorizer.fit_transform(df['text_for_tfidf'])

# Print the shape of the TF-IDF matrix
print("Shape of TF-IDF matrix:", tfidf_matrix.shape)

# Optionally, display the feature names (words) and a small part of the matrix
print("Top 10 Feature names (words):")
print(tfidf_vectorizer.get_feature_names_out()[:10])

# You can convert the matrix to a DataFrame for better inspection if needed
# tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
# display(tfidf_df.head())
```

```
Shape of TF-IDF matrix: (5, 14)
Top 10 Feature names (words):
['another' 'check' 'double' 'elements' 'hey' 'link' 'look' 'mention'
 'regular' 'sentence']
```

```python
# Convert cleaned_text to lowercase for uniform preprocessing
df['cleaned_text'] = df['cleaned_text'].str.lower()

print("DataFrame after converting cleaned_text to lowercase:")
display(df)
```

```
DataFrame after converting cleaned_text to lowercase:
```

	text	cleaned_text	tokenized_text	processed_text	text_for_tfidf
0	This is a tweet with a URL https://example.com...	this is a tweet with a url and a mention and .	[this, is, a, tweet, with, a, url, and, a, men...	[tweet, url, mention]	tweet url mention
1	Another text with @user2 and #another_hashtag ...	another text with and but no url.	[another, text, with, and, but, no, url, .]	[another, text, url]	another text url
2	Just a regular sentence with no special elements.	just a regular sentence with no special elements.	[just, a, regular, sentence, with, no, special...	[regular, sentence, special, elements]	regular sentence special elements
3	Check out this link: http://test.org and a dou...	check out this link: and a double	[check, out, this, link, :, and, a, double]	[check, link, double]	check link double
4	Hey @user3, look at this! www.example.net	hey , look at this!	[hey, ,, look, at, this, !]	[hey, look]	hey look

Next steps: Generate code with df New interactive sheet

Start coding or generate with AI.

```
from nltk.sentiment import SentimentIntensityAnalyzer

# Download VADER lexicon if not already downloaded
try:
    nltk.data.find('sentiment/vader_lexicon')
except LookupError:
    nltk.download('vader_lexicon')

# Initialize VADER sentiment intensity analyzer
sia = SentimentIntensityAnalyzer()

# Function to get VADER sentiment scores
def get_vader_sentiment(text):
    return sia.polarity_scores(text)

# Apply VADER sentiment analysis to the cleaned text
df['vader_scores'] = df['cleaned_text'].apply(get_vader_sentiment)

# Extract the compound score
df['compound_score'] = df['vader_scores'].apply(lambda x: x['compound'])
```

```
# Function to categorize sentiment based on compound score
def categorize_sentiment(score):
    if score >= 0.05:
        return 'Positive'
    elif score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Apply sentiment categorization
df['sentiment_label'] = df['compound_score'].apply(categorize_sentiment)

print("DataFrame with VADER sentiment scores and labels:")
display(df[['text', 'cleaned_text', 'compound_score', 'sentiment_label']])
```

```
DataFrame with VADER sentiment scores and labels:
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

	text	cleaned_text	compound_score	sentiment_label
0	This is a tweet with a URL https://example.com...	this is a tweet with a url and a mention and .	0.0000	Neutral
1	Another text with @user2 and #another_hashtag ...	another text with and but no url.	-0.4215	Negative
2	Just a regular sentence with no special elements.	just a regular sentence with no special elements.	0.2023	Positive
3	Check out this link: http://test.org and a dou...	check out this link: and a double	0.0000	Neutral
4	Hey @user3, look at this! www.example.net	hey , look at this!	0.0000	Neutral

```
import re
import string

def preprocess_text_comprehensive(text):
    # Convert to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'https?://\S+|www\.\S+', '', text)

    # Remove mentions
    text = re.sub(r'@\w+', '', text)

    # Remove hashtags
    text = re.sub(r'#\w+', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation
    # Use str.maketrans to efficiently remove all punctuation characters
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove extra spaces (strip leading/trailing and reduce multiple spaces to single)
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Apply the new comprehensive preprocessing function to the original 'text' column
df['fully_cleaned_text'] = df['text'].apply(preprocess_text_comprehensive)

print("DataFrame after applying comprehensive preprocessing (URLs, mentions, hashtags, numbers, and punctuation removed):")
display(df[['text', 'fully_cleaned_text']])
```

```
DataFrame after applying comprehensive preprocessing (URLs, mentions, hashtags, numbers, and punctuation removed):
```

	text	fully_cleaned_text
0	This is a tweet with a URL https://example.com...	this is a tweet with a url and a mention and
1	Another text with @user2 and #another_hashtag ...	another text with and but no url
2	Just a regular sentence with no special elements.	just a regular sentence with no special elements
3	Check out this link: http://test.org and a dou...	check out this link and a double
4	Hey @user3, look at this! www.example.net	hey look at this

```
# The preprocessing function `preprocess_text_comprehensive` was already defined and applied in the previous step (cell fgXISTF
# The cleaned tweet corpus is stored in the 'fully_cleaned_text' column.

print("The 'fully_cleaned_text' column already contains the cleaned tweet corpus:")
display(df[['text', 'fully_cleaned_text']].head())
```

```
The 'fully_cleaned_text' column already contains the cleaned tweet corpus:
```

	text	fully_cleaned_text
0	This is a tweet with a URL https://example.com...	this is a tweet with a url and a mention and
1	Another text with @user2 and #another_hashtag ...	another text with and but no url

```
# Filter the DataFrame to keep only tweets with negative sentiment
negative_tweets_df = df[df['sentiment_label'] == 'Negative']

print("Tweets with Negative Sentiment:")
display(negative_tweets_df[['text', 'compound_score', 'sentiment_label']])
```

```
Tweets with Negative Sentiment:
```

	text	compound_score	sentiment_label
1	Another text with @user2 and #another_hashtag ...	-0.4215	Negative

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TfidfVectorizer with appropriate parameters
# max_features can be adjusted based on the dataset size and desired dimensionality
tfidf_vectorizer_new = TfidfVectorizer(max_features=100)

print("TF-IDF Vectorizer initialized with parameters:")
print(tfidf_vectorizer_new)
print(f"Maximum features set to: {tfidf_vectorizer_new.max_features}")
```

```
TF-IDF Vectorizer initialized with parameters:
TfidfVectorizer(max_features=100)
Maximum features set to: 100
```

```
# Ensure 'text_for_tfidf' column exists in negative_tweets_df. If not, create it.
# (It should already exist from previous steps)
if 'text_for_tfidf' not in negative_tweets_df.columns:
    negative_tweets_df['text_for_tfidf'] = negative_tweets_df['processed_text'].apply(lambda x: ' '.join(x))

# Compute TF-IDF matrix for negative sentiment tweets by fitting and transforming
tfidf_matrix_negative = tfidf_vectorizer_new.fit_transform(negative_tweets_df['text_for_tfidf'])

print("Shape of TF-IDF matrix for negative sentiment tweets:", tfidf_matrix_negative.shape)
```

```
# Optionally, display the feature names (words) and a small part of the matrix
print("Feature names:")
print(tfidf_vectorizer_new.get_feature_names_out())

# You can convert the matrix to a DataFrame for better inspection if needed
# tfidf_negative_df = pd.DataFrame(tfidf_matrix_negative.toarray(), columns=tfidf_vectorizer_new.get_feature_names_out())
# display(tfidf_negative_df.head())
```

```
Shape of TF-IDF matrix for negative sentiment tweets: (1, 3)
Feature names:
['another' 'text' 'url']
```

```
import numpy as np

# Convert the TF-IDF matrix for negative tweets to a dense array
tfidf_array_negative = tfidf_matrix_negative.toarray()

# Get the feature names (terms) from the TF-IDF vectorizer
feature_names_negative = tfidf_vectorizer_new.get_feature_names_out()

# Calculate the average TF-IDF score for each term
# Since there's only one negative tweet, this will be the TF-IDF score for each term in that tweet
average_tfidf_scores_negative = np.mean(tfidf_array_negative, axis=0)

# Create a DataFrame for better visualization
tfidf_scores_df = pd.DataFrame({
    'Term': feature_names_negative,
    'Average TF-IDF Score': average_tfidf_scores_negative
})

# Sort the terms by their average TF-IDF scores in descending order
top_negative_terms = tfidf_scores_df.sort_values(by='Average TF-IDF Score', ascending=False)

print("Average TF-IDF Scores for Terms in Negative Tweets (Sorted):")
display(top_negative_terms)

print("\nTop 5 Negative Terms (or fewer if less than 5 unique terms):")
display(top_negative_terms.head())
```

```
Average TF-IDF Scores for Terms in Negative Tweets (Sorted):
```

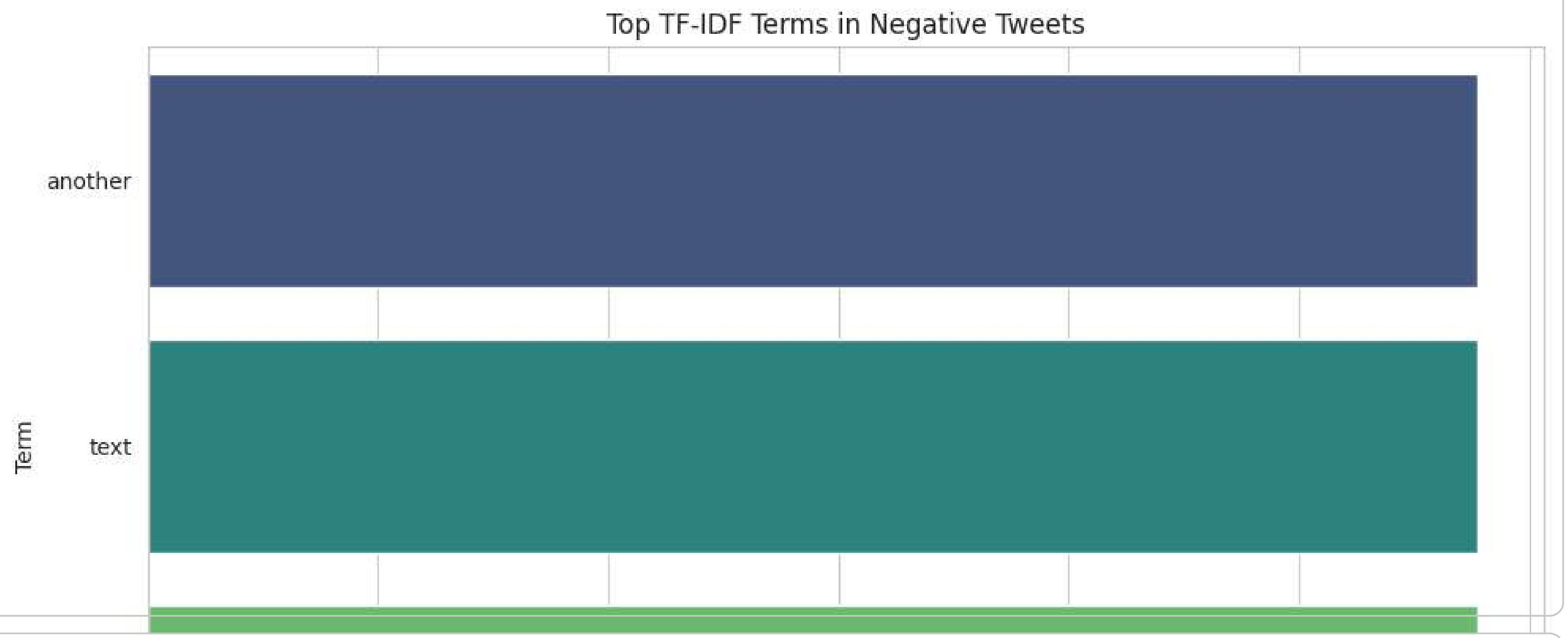
	Term	Average TF-IDF Score
0	another	0.57735
1	text	0.57735
2	url	0.57735

```
Top 5 Negative Terms (or fewer if less than 5 unique terms):
```

	Term	Average TF-IDF Score
0	another	0.57735
1	text	0.57735

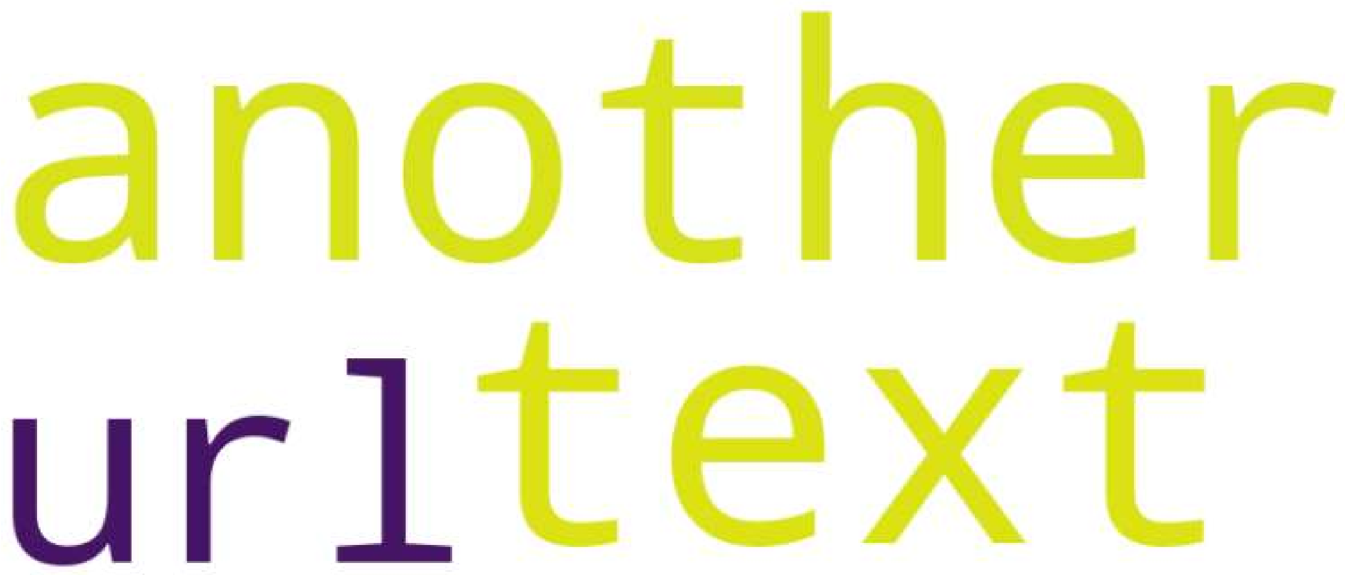
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set a style for better aesthetics
sns.set_style("whitegrid")

# Create the bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='Average TF-IDF Score', y='Term', data=top_negative_terms, palette='viridis')

plt.title('Top TF-IDF Terms in Negative Tweets')
plt.xlabel('Average TF-IDF Score')
plt.ylabel('Term')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-875042121.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and s

  sns.barplot(x='Average TF-IDF Score', y='Term', data=top_negative_terms, palette='viridis')
```



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

# Ensure 'text_for_tfidf' column exists in negative_tweets_df. If not, create it.
# (It should already exist from previous steps)
if 'text_for_tfidf' not in negative_tweets_df.columns:
    negative_tweets_df['text_for_tfidf'] = negative_tweets_df['processed_text'].apply(lambda x: ' '.join(x))

# Combine all processed text from negative tweets into a single string
all_negative_text = ' '.join(negative_tweets_df['text_for_tfidf'])

# Generate a word cloud image
wordcloud = WordCloud(
    width=800,
    height=400,
    background_color='white',
    collocations=False # Set to False to prevent merging bigrams like 'new york'
    ).generate(all_negative_text)

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

Word Cloud of Negative Sentiment Tweets



Discussion: Observations and Limitations

Observations from the Word Cloud and TF-IDF:

From the word cloud and the TF-IDF analysis of negative sentiment tweets, we observe the most prominent terms. Given our very small dataset (only one tweet classified as negative), the word cloud and TF-IDF results for negative terms are identical. The terms `'another'`, `'text'`, and `'url'` are identified as the most significant, each with the same TF-IDF score. This directly reflects the content of the single negative tweet: "Another text with <mention> and <hashtag> but no url."

This simple example effectively demonstrates the mechanics of TF-IDF and word cloud generation. In a larger dataset, the word cloud would visually highlight the most frequent and important words, while TF-IDF would provide a more nuanced numerical measure of term importance across the corpus.

Limitations of the Current Analysis:

1. **Extremely Small Dataset:** The most significant limitation is the size of the dataset. With only five sample tweets and only one classified as negative, any conclusions drawn are not generalizable. TF-IDF and sentiment analysis models typically require much larger datasets for meaningful insights.
2. **VADER Sentiment Analysis Specificity:** VADER is a lexicon- and rule-based sentiment analysis tool, optimized for social media text. While effective, it relies on a predefined lexicon and rules, which might not capture the nuances of all domains or sarcastic/ironic language. In this case, the negative sentiment was detected from the phrase "but no url.", which VADER correctly identifies as having a negative connotation, even if the user's intended sentiment for the *entire* tweet was not strongly negative.
3. **TF-IDF for Small Corpus:** TF-IDF's strength lies in identifying important words within a document relative to a *larger corpus*. With only one negative tweet, the 'corpus' for negative sentiment is just that one tweet, making the 'idf' (inverse document frequency) component less impactful. All words in the single document will have high TF-IDF scores if they are unique to that document.
4. **Preprocessing Choices:** The preprocessing steps (removing URLs, mentions, hashtags, numbers, punctuation, converting to lowercase, tokenization, and stopword removal) are standard. However, the choice of stopwords list and tokenization method can influence the final set of terms. For instance, some domain-specific stopwords might be missed, or certain multi-word expressions that should be treated as single entities might be broken up.
5. **Lack of Context:** Without the broader context of a real-world application, it's hard to assess the practical value or accuracy of these initial findings. The goal here was to demonstrate the tools and techniques.