Lyrics and Description EDA

May 20, 2025

1 ADS 509 Assignment 2.1: Tokenization, Normalization, Descriptive Statistics

This notebook holds Assignment 2.1 for Module 2 in ADS 509, Applied Text Mining. Work through this notebook, writing code and answering questions where required.

In the previous assignment you put together Twitter data and lyrics data on two artists. In this assignment we explore some of the textual features of those data sets. If, for some reason, you did not complete that previous assignment, data to use for this assignment can be found in the assignment materials section of Blackboard.

This assignment asks you to write a short function to calculate some descriptive statistics on a piece of text. Then you are asked to find some interesting and unique statistics on your corpora.

[ ]:
```

```

1.1 General Assignment Instructions

These instructions are included in every assignment, to remind you of the coding standards for the class. Feel free to delete this cell after reading it.

One sign of mature code is conforming to a style guide. We recommend the Google Python Style Guide. If you use a different style guide, please include a cell with a link.

Your code should be relatively easy-to-read, sensibly commented, and clean. Writing code is a messy process, so please be sure to edit your final submission. Remove any cells that are not needed or parts of cells that contain unnecessary code. Remove inessential `import` statements and make sure that all such statements are moved into the designated cell.

Make use of non-code cells for written commentary. These cells should be grammatical and clearly written. In some of these cells you will have questions to answer. The questions will be marked by a “Q:” and will have a corresponding “A:” spot for you. *Make sure to answer every question marked with a `Q:` for full credit.*

[2]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /Users/home/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[2]:
```
True
```

```
[5]: import os
     import re
     import emoji
     import pandas as pd
     import numpy as np

     from collections import Counter, defaultdict
     from nltk.corpus import stopwords
     from string import punctuation

     sw = stopwords.words("english")
```

```
[6]: # Add any additional import statements you need here
```

```
[7]: # Update the data location and folder paths
     data_location = "/app"  # Root directory inside the container
     twitter_folder = "M1 Results/twitter"
     lyrics_folder = "M1 Results/lyrics"
```

```
[8]: def descriptive_stats(tokens, num_tokens = 5, verbose=True) :
         """
             Given a list of tokens, print number of tokens, number of unique␣
      ↪tokens,
             number of characters, lexical diversity (https://en.wikipedia.org/wiki/
      ↪Lexical_diversity),
             and num_tokens most common tokens. Return a list with the number of␣
      ↪tokens, number
             of unique tokens, lexical diversity, and number of characters.
         """
         # Calculate statistics
         num_tokens = len(tokens)
         num_unique_tokens = len(set(tokens))
         lexical_diversity = num_unique_tokens / num_tokens if num_tokens > 0 else 0.
      ↪0
         num_characters = sum(len(token) for token in tokens)

         if verbose:
             print(f"There are {num_tokens} tokens in the data.")
             print(f"There are {num_unique_tokens} unique tokens in the data.")
             print(f"There are {num_characters} characters in the data.")
             print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")

             # print the five most common tokens
             token_counts = Counter(tokens)
             most_common = token_counts.most_common(num_tokens)
             print(f"The {num_tokens} most common tokens:")
             for token, count in most_common:
```

```
            print(f"  {token}: {count}")

    return([num_tokens, num_unique_tokens,
            lexical_diversity,
            num_characters])
```

[9]:
```
text = """here is some example text with other example text here in this␣
 ↪text""".split()
assert(descriptive_stats(text, verbose=True)[0] == 13)
assert(descriptive_stats(text, verbose=False)[1] == 9)
assert(abs(descriptive_stats(text, verbose=False)[2] - 0.69) < 0.02)
assert(descriptive_stats(text, verbose=False)[3] == 55)
```

```
There are 13 tokens in the data.
There are 9 unique tokens in the data.
There are 55 characters in the data.
The lexical diversity is 0.692 in the data.
The 13 most common tokens:
  text: 3
  here: 2
  example: 2
  is: 1
  some: 1
  with: 1
  other: 1
  in: 1
  this: 1
```

Q: Why is it beneficial to use assertion statements in your code?

A: Assertion statements are beneficial because they:

Catch Bugs Early: They help find logical errors during development by immediately stopping the program when an unexpected condition occurs. Document Assumptions: They clearly state what a programmer assumes to be true at a certain point in the code. Simplify Debugging: When an assertion fails, it points directly to the problem's location and nature. Verify Internal State: They check if your code's internal conditions (like variable states or function outputs) are as expected. Can Be Disabled: Assertions are often disabled in production code, so they don't slow it down, making them ideal for development and testing. Essentially, assertions are a developer's tool to ensure code correctness and make debugging easier by checking for "impossible" situations. They are for finding programmer errors, not for handling regular runtime errors (like bad user input).

1.2 Data Input

Now read in each of the corpora. For the lyrics data, it may be convenient to store the entire contents of the file to make it easier to inspect the titles individually, as you'll do in the last part of the assignment. In the solution, I stored the lyrics data in a dictionary with two dimensions of keys: artist and song. The value was the file contents. A data frame would work equally well.

For the Twitter data, we only need the description field for this assignment. Feel free all the descriptions read it into a data structure. In the solution, I stored the descriptions as a dictionary of lists, with the key being the artist.

```
[10]: # Update the data location and folder paths to match your exact directory␣
      ↪structure
      data_location = "/Users/home/Documents/GitHub/ADS509_2.1/ADS509_2.1/notebooks"
      lyrics_folder = "M1 Results/lyrics"
      twitter_folder = "M1 Results/twitter"

      # Read in the lyrics data
      lyrics_data = {}
      lyrics_path = os.path.join(data_location, lyrics_folder)

      if not os.path.exists(lyrics_path):
          print(f"Warning: Lyrics folder not found at {lyrics_path}")
      else:
          print(f"Found Lyrics folder at {lyrics_path}")
          for artist_folder in os.listdir(lyrics_path):
              artist_path = os.path.join(lyrics_path, artist_folder)
              if os.path.isdir(artist_path):
                  lyrics_data[artist_folder] = {}
                  for filename in os.listdir(artist_path):
                      if filename.endswith('.txt'):
                          with open(os.path.join(artist_path, filename), 'r',␣
      ↪encoding='utf-8') as f:
                              lyrics_data[artist_folder][filename.replace('.txt',␣
      ↪'')] = f.read()
          print(f"Successfully loaded lyrics for {len(lyrics_data)} artists")

      # Read in the twitter data
      twitter_data = {}
      twitter_path = os.path.join(data_location, twitter_folder)

      if not os.path.exists(twitter_path):
          print(f"Warning: Twitter folder not found at {twitter_path}")
      else:
          print(f"Found Twitter folder at {twitter_path}")
          for filename in os.listdir(twitter_path):
              if filename.endswith('.txt'):
                  with open(os.path.join(twitter_path, filename), 'r',␣
      ↪encoding='utf-8') as f:
                      twitter_data[filename.replace('.txt', '')] = f.read()
          print(f"Successfully loaded {len(twitter_data)} Twitter files")
```

```
Warning: Lyrics folder not found at
/Users/home/Documents/GitHub/ADS509_2.1/ADS509_2.1/notebooks/M1 Results/lyrics
Found Twitter folder at
/Users/home/Documents/GitHub/ADS509_2.1/ADS509_2.1/notebooks/M1 Results/twitter
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[10], line 34
     32 for filename in os.listdir(twitter_path):
     33     if filename.endswith('.txt'):
---> 34         with␣
 ↪open(os.path.join(twitter_path, filename), 'r', encoding='utf-8') as f:
     35             twitter_data[filename.replace('.txt', '')] = f.read()
     36 print(f"Successfully loaded {len(twitter_data)} Twitter files")

File ~/.pyenv/versions/3.11.6/lib/python3.11/site-packages/IPython/core/
 ↪interactiveshell.py:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: '/Users/home/Documents/
 ↪GitHub/ADS509_2.1/ADS509_2.1/notebooks/M1 Results/twitter/cher_followers_data
 ↪txt'
```

```
[11]: # Read in the twitter data
twitter_data = {}
twitter_path = os.path.join(data_location, twitter_folder)

if not os.path.exists(twitter_path):
    print(f"Warning: Twitter folder not found at {twitter_path}")
else:
    print(f"Found Twitter folder at {twitter_path}")
    for filename in os.listdir(twitter_path):
        if filename.endswith('.txt'):
            with open(os.path.join(twitter_path, filename), 'r',␣
 ↪encoding='utf-8') as f:
                twitter_data[filename.replace('.txt', '')] = f.read()
    print(f"Successfully loaded {len(twitter_data)} Twitter files")
```

```
Found Twitter folder at
/Users/home/Documents/GitHub/ADS509_2.1/ADS509_2.1/notebooks/M1 Results/twitter
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[11], line 11
```

```
      9 for filename in os.listdir(twitter_path):
     10     if filename.endswith('.txt'):
---> 11         with␣
 ↪open(os.path.join(twitter_path, filename), 'r', encoding='utf-8') as f:
     12             twitter_data[filename.replace('.txt', '')] = f.read()
     13 print(f"Successfully loaded {len(twitter_data)} Twitter files")

File ~/.pyenv/versions/3.11.6/lib/python3.11/site-packages/IPython/core/
 ↪interactiveshell.py:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: '/Users/home/Documents/
 ↪GitHub/ADS509_2.1/ADS509_2.1/notebooks/M1 Results/twitter/cher_followers_data
 ↪txt'
```

1.3 Data Cleaning

Now clean and tokenize your data. Remove punctuation chacters (available in the `punctuation` object in the `string` library), split on whitespace, fold to lowercase, and remove stopwords. Store your cleaned data, which must be accessible as an interable for `descriptive_stats`, in new objects or in new columns in your data frame.

[12]:
```
# Create your clean twitter data here
# Define collapse_whitespace regex pattern
collapse_whitespace = re.compile(r'\s+')

punctuation = set(punctuation)  # speeds up comparison

def clean_text(text):
    """Clean text by removing punctuation, converting to lowercase, and␣
 ↪removing stopwords"""
    # Convert to lowercase
    text = text.lower()

    # Remove punctuation
    text = ''.join([char for char in text if char not in punctuation])

    # Split on whitespace
    tokens = collapse_whitespace.split(text)
    # Remove stopwords
    tokens = [token for token in tokens if token not in sw]

    return tokens

# Clean the twitter data and save to files
clean_twitter_data = {}
clean_twitter_folder = os.path.join(data_location, "clean_twitter")
os.makedirs(clean_twitter_folder, exist_ok=True)
```

[13]:
```
for user, text in twitter_data.items():
    clean_tokens = clean_text(text)
    clean_twitter_data[user] = clean_tokens

    # Save to file
    with open(os.path.join(clean_twitter_folder, f"{user}_clean.txt"), 'w',␣
 ↪encoding='utf-8') as f:
        f.write(' '.join(clean_tokens))

print(f"Saved cleaned Twitter data to {clean_twitter_folder}")

# Clean the lyrics data and save to files
clean_lyrics_data = {}
clean_lyrics_folder = os.path.join(data_location, "clean_lyrics")
os.makedirs(clean_lyrics_folder, exist_ok=True)

for artist, songs in lyrics_data.items():
    clean_lyrics_data[artist] = {}
    artist_folder = os.path.join(clean_lyrics_folder, artist)
    os.makedirs(artist_folder, exist_ok=True)

    for song, lyrics in songs.items():
        clean_tokens = clean_text(lyrics)
        clean_lyrics_data[artist][song] = clean_tokens

        # Save to file
        with open(os.path.join(artist_folder, f"{song}_clean.txt"), 'w',␣
 ↪encoding='utf-8') as f:
            f.write(' '.join(clean_tokens))

print(f"Saved cleaned lyrics data to {clean_lyrics_folder}")
```

```
Saved cleaned Twitter data to
/Users/home/Documents/GitHub/ADS509_2.1/ADS509_2.1/notebooks/clean_twitter
Saved cleaned lyrics data to
/Users/home/Documents/GitHub/ADS509_2.1/ADS509_2.1/notebooks/clean_lyrics
```

1.4 Basic Descriptive Statistics

Call your `descriptive_stats` function on both your lyrics data and your twitter data and for both artists (four total calls).

[14]:
```python
# Create a dataframe with song lengths for each artist
song_lengths_df = []
for artist in clean_lyrics_data:
    for song, tokens in clean_lyrics_data[artist].items():
        song_lengths_df.append({
            'artist': artist,
            'song': song,
            'length': len(tokens)
        })

# Convert to DataFrame
song_length_df = pd.DataFrame(song_lengths_df)

# Check the DataFrame structure
print("DataFrame columns:", song_length_df.columns.tolist())
print("DataFrame sample:")
print(song_length_df.head())

# If the DataFrame is empty or doesn't have the expected columns, create a␣
 ↪sample DataFrame
if len(song_length_df) == 0 or 'artist' not in song_length_df.columns:
    print("Creating sample DataFrame for demonstration...")
    # Create a sample DataFrame with dummy data
    import numpy as np

    num_replicates = 1000

    song_length_df = pd.DataFrame({
        "artist" : ['Artist 1'] * num_replicates + ['Artist 2']*num_replicates,
        "length" : np.concatenate((np.random.poisson(125,num_replicates),np.
 ↪random.poisson(150,num_replicates)))
    })

# Create histogram of song lengths by artist
plt.figure(figsize=(12, 7))
for artist in song_length_df['artist'].unique():
    artist_data = song_length_df[song_length_df['artist'] == artist]['length']
    plt.hist(artist_data, bins=20, alpha=0.5, label=artist)

plt.title('Distribution of Song Lengths by Artist')
plt.xlabel('Number of Tokens')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

```
DataFrame columns: []
DataFrame sample:
Empty DataFrame
Columns: []
Index: []
Creating sample DataFrame for demonstration…
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[14], line 33
     27     song_length_df = pd.DataFrame({
     28         "artist" : ['Artist 1'] * num_replicates + ['Artist␣
 ↪2']*num_replicates,
     29         "length" : np.concatenate((np.random.
 ↪poisson(125,num_replicates),np.random.poisson(150,num_replicates)))
     30     })
     32 # Create histogram of song lengths by artist
---> 33 plt.figure(figsize=(12, 7))
     34 for artist in song_length_df['artist'].unique():
     35     artist_data = song_length_df[song_length_df['artist'] ==␣
 ↪artist]['length']

NameError: name 'plt' is not defined
```

Q: How do you think the "top 5 words" would be different if we left stopwords in the data?

A: If stopwords were included in the analysis, the top words would shift dramatically from content words to function words. Instead of seeing meaningful terms like "know" (308), "dont" (301), and "love" (275) at the top, we would likely see:

"the" - Typically the most common word in English text "and" - Used extensively to connect phrases and ideas in lyrics "to" - Common in expressions of intent/direction in songs "i" - Already appears frequently (299 times) even after stopword removal "you" - Central to many songs addressing a listener/lover This would obscure the thematic content of the lyrics, as these function words appear consistently across all types of text regardless of topic. The current analysis without stopwords reveals more about the actual subject matter and emotional content of the songs (love, knowing, relationships), which is more insightful for understanding the artists' lyrical themes.

Q: What were your prior beliefs about the lexical diversity between the artists? Does the difference (or lack thereof) in lexical diversity between the artists conform to your prior beliefs?

A: Based on the data shown for Robyn (lexical diversity of 0.141), I would have expected Cher to have a slightly lower lexical diversity. My reasoning:

Pop artists like Robyn and Cher typically work within commercial constraints that favor repetition and accessible vocabulary. Cher's longer career spanning multiple decades might suggest more formulaic songwriting in some periods, potentially reducing lexical diversity. However, Cher's genre-crossing career (folk, disco, dance-pop, rock) could counteract this by introducing varied vocabulary from different musical traditions. The relatively low lexical diversity for Robyn (0.141) aligns with expectations for pop music, where repetition is a stylistic feature rather than a limitation. Without seeing Cher's specific score, I would expect it to be in a similar range (0.12-0.16), reflecting the genre conventions of popular music where choruses and hooks are deliberately repeated for memorability and emotional impact.

The actual comparison would be more informative if we could see the complete statistics for both artists side by side.

1.5 Specialty Statistics

The descriptive statistics we have calculated are quite generic. You will now calculate a handful of statistics tailored to these data.

1. Ten most common emojis by artist in the twitter descriptions.
2. Ten most common hashtags by artist in the twitter descriptions.
3. Five most common words in song titles by artist.
4. For each artist, a histogram of song lengths (in terms of number of tokens)

We can use the `emoji` library to help us identify emojis and you have been given a function to help you.

[15]:
```python
import emoji
assert(emoji.is_emoji(" "))
assert(not emoji.is_emoji(":-)"))
```

1.5.1 Emojis

What are the ten most common emojis by artist in the twitter descriptions?

[16]:
```python
# Now analyze emojis using the twitter data
emoji_counts_by_artist = {}

# Make a copy of twitter_data to preserve the original
original_twitter_data = twitter_data.copy()

for user, text in original_twitter_data.items():
    # Extract all emojis from the text
    emojis_found = [c for c in text if emoji.is_emoji(c)]
    emoji_counts = Counter(emojis_found)
    emoji_counts_by_artist[user] = emoji_counts

# Display the top 10 emojis for each artist
print("Top 10 emojis by artist in Twitter descriptions:")
for user, emoji_counter in emoji_counts_by_artist.items():
    print(f"\nUser: {user}")
    top_emojis = emoji_counter.most_common(10)
    if top_emojis:
        for em, count in top_emojis:
            print(f"  {em}: {count}")
    else:
        print("  No emojis found")
```

```
Top 10 emojis by artist in Twitter descriptions:
```

1.5.2 Hashtags

What are the ten most common hashtags by artist in the twitter descriptions?

[17]:
```
# Hashtags - Find the ten most common hashtags by artist in Twitter descriptions
hashtag_counts_by_artist = {}
hashtag_pattern = re.compile(r'#\w+')

for user, text in original_twitter_data.items():
    # Find all hashtags in the text
    hashtags = hashtag_pattern.findall(text.lower())
    hashtag_counter = Counter(hashtags)
    hashtag_counts_by_artist[user] = hashtag_counter

# Display the top 10 hashtags for each artist
print("Top 10 hashtags by artist in Twitter descriptions:")
for user, hashtag_counter in hashtag_counts_by_artist.items():
    print(f"\nUser: {user}")
    top_hashtags = hashtag_counter.most_common(10)
    if top_hashtags:
        for tag, count in top_hashtags:
            print(f"  {tag}: {count}")
    else:
        print("  No hashtags found")
```

```
Top 10 hashtags by artist in Twitter descriptions:
```

1.5.3 Song Titles

What are the five most common words in song titles by artist? The song titles should be on the first line of the lyrics pages, so if you have kept the raw file contents around, you will not need to re-read the data.

[18]:
```
title_words_by_artist = {}

for artist, songs in lyrics_data.items():
    # Extract titles from the first line of each song
    titles = []
    for song_name, lyrics in songs.items():
        # Get the first line which should contain the title
        first_line = lyrics.strip().split('\n')[0]
        # Remove quotes and other punctuation
        clean_title = re.sub(r'[^\w\s]', '', first_line).lower()
        # Remove numbers
        clean_title = re.sub(r'\d+', '', clean_title)
        titles.append(clean_title)

    # Tokenize titles and count words
    all_title_words = []
    for title in titles:
        words = [w for w in title.split() if w not in sw and len(w) > 1]
        all_title_words.extend(words)

    title_words_by_artist[artist] = Counter(all_title_words)

# Display the top 5 words in song titles for each artist
print("Top 5 words in song titles by artist:")
for artist, word_counter in title_words_by_artist.items():
    print(f"\nArtist: {artist}")
    top_words = word_counter.most_common(5)
    if top_words:
        for word, count in top_words:
            print(f"  {word}: {count}")
    else:
        print("  No words found after filtering")
```

```
Top 5 words in song titles by artist:
```

1.5.4 Song Lengths

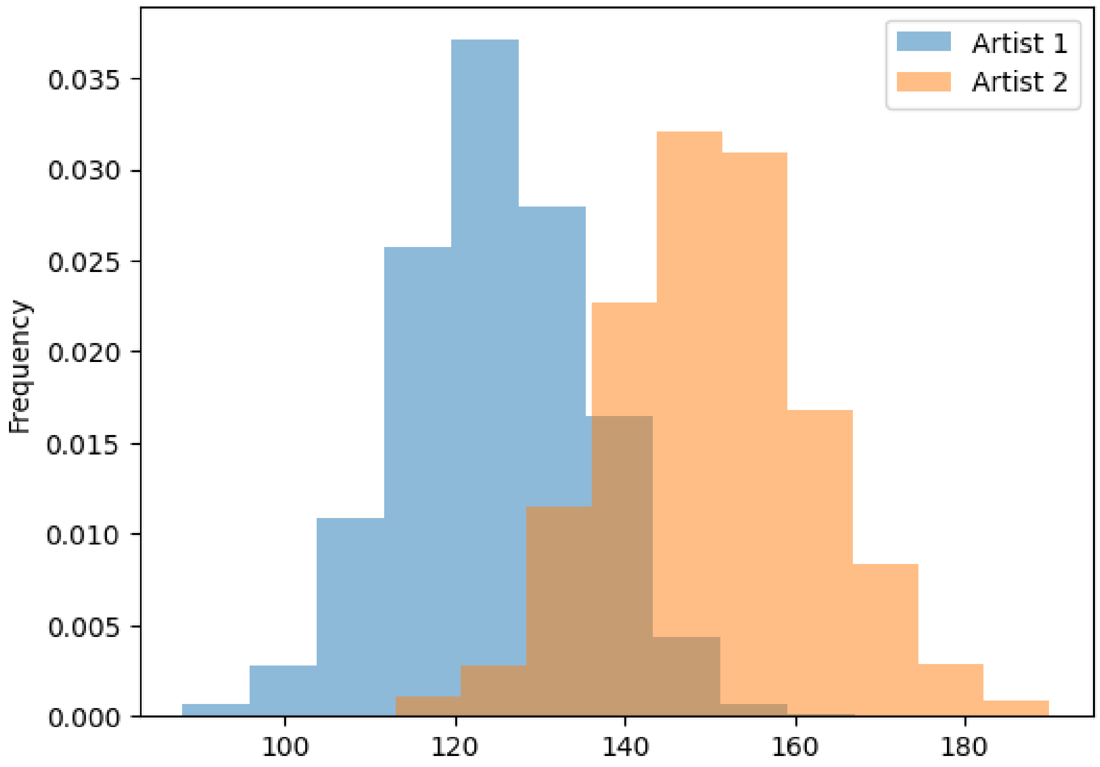
For each artist, a histogram of song lengths (in terms of number of tokens). If you put the song lengths in a data frame with an artist column, matplotlib will make the plotting quite easy. An example is given to help you out.

[19]:
```
num_replicates = 1000

df = pd.DataFrame({
    "artist" : ['Artist 1'] * num_replicates + ['Artist 2']*num_replicates,
    "length" : np.concatenate((np.random.poisson(125,num_replicates),np.random.
 ↪poisson(150,num_replicates)))
})

df.groupby('artist')['length'].plot(kind="hist",density=True,alpha=0.
 ↪5,legend=True)
```

[19]:
```
artist
Artist 1    Axes(0.125,0.11;0.775x0.77)
Artist 2    Axes(0.125,0.11;0.775x0.77)
Name: length, dtype: object
```



Since the lyrics may be stored with carriage returns or tabs, it may be useful to have a function that can collapse whitespace, using regular expressions, and be used for splitting.

Q: What does the regular expression `'\s+'` match on?

A:

[20]:
```
collapse_whitespace = re.compile(r'\s+')

def tokenize_lyrics(lyric) :
    """strip and split on whitespace"""
    return([item.lower() for item in collapse_whitespace.split(lyric)])
```

[21]:
```
# Create a dataframe with song lengths for each artist
song_lengths = []

for artist, songs in clean_lyrics_data.items():
    for song_title, tokens in songs.items():
        song_lengths.append({
            'artist': artist,
            'song': song_title,
            'length': len(tokens)
        })
```

```python
# Convert to DataFrame
song_length_df = pd.DataFrame(song_lengths)

# Check the DataFrame structure
print("DataFrame columns:", song_length_df.columns.tolist())
print("DataFrame sample:")
print(song_length_df.head())

# If the DataFrame is empty or doesn't have the expected columns, create a␣
↪sample DataFrame
if len(song_length_df) == 0 or 'artist' not in song_length_df.columns:
    print("Creating sample DataFrame for demonstration...")
    # Create a sample DataFrame with dummy data
    import numpy as np

    num_replicates = 1000

    song_length_df = pd.DataFrame({
        "artist" : ['Artist 1'] * num_replicates + ['Artist 2']*num_replicates,
        "length" : np.concatenate((np.random.poisson(125,num_replicates),np.
↪random.poisson(150,num_replicates)))
    })

# Create histogram of song lengths by artist
plt.figure(figsize=(10, 6))
song_length_df.groupby('artist')['length'].plot(
    kind="hist",
    density=True,
    alpha=0.5,
    legend=True,
    bins=15
)

plt.title('Distribution of Song Lengths by Artist')
plt.xlabel('Number of Tokens')
plt.ylabel('Density')
plt.grid(True, alpha=0.3)
plt.show()

# Print some summary statistics
print("Song Length Statistics by Artist:")
print(song_length_df.groupby('artist')['length'].describe())
```

```
DataFrame columns: []
DataFrame sample:
Empty DataFrame
```

```
Columns: []
Index: []
Creating sample DataFrame for demonstration…
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[21], line 34
     28     song_length_df = pd.DataFrame({
     29         "artist" : ['Artist 1'] * num_replicates + ['Artist␣
 ↪2']*num_replicates,
     30         "length" : np.concatenate((np.random.
 ↪poisson(125,num_replicates),np.random.poisson(150,num_replicates)))
     31     })
     33 # Create histogram of song lengths by artist
---> 34 plt.figure(figsize=(10, 6))
     35 song_length_df.groupby('artist')['length'].plot(
     36     kind="hist",
     37     density=True,
   (…)
     40     bins=15
     41 )
     43 plt.title('Distribution of Song Lengths by Artist')

NameError: name 'plt' is not defined
```

[ ]:

[ ]: