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.

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.*

In [38]:
```
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")
```

In [53]:
```
# Add any additional import statements you need here
import string
import matplotlib.pyplot as plt
```

In [17]:
```
# change `data_location` to the location of the folder on your machine.
data_location = "/users/landonpadgett/Desktop/M1 Results/"

twitter_folder = "/users/landonpadgett/Desktop/M1 Results/twitter/"
lyrics_folder = "/users/landonpadgett/Desktop/M1 Results/lyrics/"
```

In [20]:
```
def read_file(file_path):
    """Reads the content of a file and returns it as a string."""
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()

# Function to process all files in a folder, including subfolders
def process_folder(folder_path):
    """Processes all .txt files in a folder and its subfolders, returns tokenized text as a list of words."""
    all_data = ""
    # Walk through all subfolders and files
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.endswith(".txt"):  # Only process .txt files
                file_path = os.path.join(root, file)
                print(f"Processing file: {file_path}")  # Debugging output
                all_data += read_file(file_path) + " "
    return all_data.split()  # Tokenize by splitting on whitespace


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.
    """
    # Total number of tokens
    total_tokens = len(tokens)

    # Number of unique tokens
    num_unique_tokens = len(set(tokens))

    # Lexical diversity
    lexical_diversity = num_unique_tokens / total_tokens if total_tokens > 0 else 0

    # Number of characters
    num_characters = sum(len(token) for token in tokens)

    # Find the most common tokens
    token_counts = Counter(tokens)
    most_common_tokens = token_counts.most_common(num_tokens)

    if verbose:
        print(f"There are {total_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(f"The {num_tokens} most common tokens are:")
        for token, count in most_common_tokens:
            print(f"{token}: {count}")

    return [total_tokens, num_unique_tokens, lexical_diversity, num_characters]

# Paths to Twitter and Lyrics folders
data_location = "/users/landonpadgett/Desktop/M1 Results/"
twitter_folder = os.path.join(data_location, "twitter/")
lyrics_folder = os.path.join(data_location, "lyrics/")

# Process and analyze Twitter data
twitter_tokens = process_folder(twitter_folder)
print("Twitter Data Stats:")
descriptive_stats(twitter_tokens, verbose=True)

# Process and analyze Lyrics data (including subfolders like cher/ and robyn/)
lyrics_tokens = process_folder(lyrics_folder)
print("\nLyrics Data Stats:")
descriptive_stats(lyrics_tokens, verbose=True)
```

```
Processing file: /users/landonpadgett/Desktop/M1 Results/twitter/cher_followers_data.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/twitter/robynkonichiwa_followers_data.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/twitter/cher_followers.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/twitter/robynkonichiwa_followers.txt
Twitter Data Stats:
There are 58532931 tokens in the data.
There are 12777699 unique tokens in the data.
There are 368240283 characters in the data.
The lexical diversity is 0.218 in the data.
The 5 most common tokens are:
and: 598725
a: 409768
the: 400525
I: 393219
of: 365284
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_includemeout.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_electric.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_beach2k20.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_lovekills.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_timemachine.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_lovekills114524.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_givingyouback.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_noneofdem114527.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_noneofdem.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_bemine.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_fembot114519.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_shouldhaveknown.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_underneaththeheart.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_eclipse.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_robynishere.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_dontstopthemusic.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_criminalintent.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_myonlyreason.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_humanbeing.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_obaby.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_how.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_loveisfree.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_longgone.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_indestructibleacousticversion.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_hangwithme.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_shouldhaveknown106828.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_moonlight.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_getmyselftogether.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/robyn/robyn_universalwoman.txt
```

```
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_onesmallstep.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_shoppin.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_chastitysun.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_fastcompany.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_elusivebutterfly.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_giveourloveafightinchance.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_justwhativebeenlookinfor.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_donttrytoclosearose.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_saywhatsonyourmind.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_thenameofthegame.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_allireallywanttodo.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_whenlovecallsyourname.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_sittinonthedockofthebay.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_ifiknewthen.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_thegreatestthing.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_allbecauseofyou.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_onehonestman.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_loversforever.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_lovehurts.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_forwhatitsworth.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_holdinoutforlove.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_takemehome.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_thebookoflove.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_chastityssongbandofthieves.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_startingover.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_shadowdreamsong.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_wasntitgood.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_youwouldntknowlove.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_thisgodforsakenday.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_time.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_iwalkalone.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_lietome.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_backonthestreetagain.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_loveonarooftop.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_hardenoughgettingoveryou.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_takeitfromtheboys.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_dreambaby.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_pleasedonttellme.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_ihopeyoufindit.txt
Processing file: /users/landonpadgett/Desktop/M1 Results/lyrics/cher/cher_classified1a.txt

Lyrics Data Stats:
There are 99415 tokens in the data.
There are 7674 unique tokens in the data.
There are 391566 characters in the data.
The lexical diversity is 0.077 in the data.
The 5 most common tokens are:
I: 3480
you: 3427
the: 2942
to: 2174
me: 1881
```

Out[20]:
```
[99415, 7674, 0.07719157068852789, 391566]
```

In [21]:
```
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 5 most common tokens are:
text: 3
here: 2
example: 2
is: 1
some: 1
```

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

A: Assertion statatements are hekpful in ensuring certain code conditions are accurate, and if they aren't, the points in the code where these inaccuracies are occurring are highlighted, allowing for a streamlined debugging process.

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.

In [43]:
```
#Lyrics read in

def read_lyrics_data(folder_path):
    lyrics_data = {}
    for artist in os.listdir(folder_path):
        artist_folder = os.path.join(folder_path, artist)
        if os.path.isdir(artist_folder):
            lyrics_data[artist] = {}
            for song_file in os.listdir(artist_folder):
                if song_file.endswith(".txt"):
                    song_path = os.path.join(artist_folder, song_file)
                    with open(song_path, 'r', encoding='utf-8') as file:
                        song_title = os.path.splitext(song_file)[0]
                        lyrics_data[artist][song_title] = file.read()
    return lyrics_data

lyrics_folder = "/users/landonpadgett/Desktop/M1 Results/lyrics/"
lyrics_data = read_lyrics_data(lyrics_folder)
```

In [44]:
```
#Twitter read in

def read_twitter_data(folder_path):
    twitter_data = {}
    for twitter_file in os.listdir(folder_path):
        if twitter_file.endswith(".txt"):
            artist = os.path.splitext(twitter_file)[0]
            twitter_data[artist] = []
            file_path = os.path.join(folder_path, twitter_file)
            with open(file_path, 'r', encoding='utf-8') as file:
                for line in file:
                    description = line.strip()
                    if description:
                        twitter_data[artist].append(description)
    return twitter_data

twitter_folder = "/users/landonpadgett/Desktop/M1 Results/twitter/"
twitter_data = read_twitter_data(twitter_folder)
```

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.

In [39]:
```
punctuation = set(punctuation) # speeds up comparison
```

In [45]:
```
#Twitter cleaning

stop_words = set(stopwords.words('english'))

# Function to clean and tokenize Twitter descriptions
def clean_twitter_data(twitter_data):
    """Cleans and tokenizes the Twitter descriptions by removing punctuation, stopwords, and lowercasing."""
    cleaned_data = {}
    for artist, descriptions in twitter_data.items():
        cleaned_descriptions = []
        for description in descriptions:
            # Remove punctuation
            description = description.translate(str.maketrans("", "", string.punctuation))

            # Convert to lowercase
            description = description.lower()

            # Split on whitespace to tokenize
            tokens = description.split()

            # Remove stopwords
            tokens = [word for word in tokens if word not in stop_words]

            cleaned_descriptions.append(tokens)

        cleaned_data[artist] = cleaned_descriptions

    return cleaned_data

cleaned_twitter_data = clean_twitter_data(twitter_data)

for artist, descriptions in cleaned_twitter_data.items():
    print(f"Artist: {artist}")
    for description in descriptions[:2]:
        print(f"  Cleaned description: {description}")
```

```
Artist: cher_followers_data
  Cleaned description: ['screenname', 'name', 'id', 'location', 'followerscount', 'friendscount', 'description'
]
  Cleaned description: ['hsmcnp', 'country', 'girl', '35152213', '1302', '1014']
Artist: robynkonichiwa_followers_data
  Cleaned description: ['screenname', 'name', 'id', 'location', 'followerscount', 'friendscount', 'description'
]
  Cleaned description: ['angelxoarts', 'angelxo', '1424055675030806529', 'zacatlan', 'puebla', 'mexico', '29',
'535', 'love', 'chill', '•facebook', 'instagram', 'soundcloud', 'angelxoarts•', 'httpstco447okklkza…']
Artist: cher_followers
  Cleaned description: ['id']
  Cleaned description: ['35152213']
Artist: robynkonichiwa_followers
  Cleaned description: ['id']
  Cleaned description: ['1424055675030806529']
```

In [46]:
```python
#Lyrics cleaning

stop_words = set(stopwords.words('english'))

def clean_lyrics_data(lyrics_data):
    cleaned_data = {}
    for artist, songs in lyrics_data.items():
        cleaned_data[artist] = {}
        for song_title, lyrics in songs.items():
            lyrics = lyrics.translate(str.maketrans("", "", string.punctuation))  # Remove punctuation
            lyrics = lyrics.lower()  # Convert to lowercase
            tokens = lyrics.split()  # Tokenize by splitting on whitespace
            tokens = [word for word in tokens if word not in stop_words]  # Remove stopwords
            cleaned_data[artist][song_title] = tokens
    return cleaned_data

cleaned_lyrics_data = clean_lyrics_data(lyrics_data)

# Example output to inspect cleaned data
for artist, songs in cleaned_lyrics_data.items():
    print(f"Artist: {artist}")
    for song_title, tokens in songs.items():
        print(f"  Song: {song_title}")
        print(f"  Cleaned lyrics: {tokens[:10]}")  # Print first 10 tokens for brevity
        break
```

```
Artist: robyn
  Song: robyn_includemeout
  Cleaned lyrics: ['include', 'really', 'simple', 'single', 'pulse', 'repeated', 'regular', 'interval', 'mmm',
'hmm']
Artist: cher
  Song: cher_comeandstaywithme
  Cleaned lyrics: ['come', 'stay', 'ill', 'send', 'away', 'false', 'pride', 'ill', 'forsake', 'life']
```

Basic Descriptive Statistics

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

In [48]:
```python
def descriptive_stats(tokens, num_tokens=5, verbose=True):
    total_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    lexical_diversity = num_unique_tokens / total_tokens if total_tokens > 0 else 0
    num_characters = sum(len(token) for token in tokens)
    token_counts = Counter(tokens)
    most_common_tokens = token_counts.most_common(num_tokens)

    if verbose:
        print(f"There are {total_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(f"The {num_tokens} most common tokens are:")
        for token, count in most_common_tokens:
            print(f"{token}: {count}")

    return [total_tokens, num_unique_tokens, lexical_diversity, num_characters, most_common_tokens]

# Function to flatten token lists
def flatten_tokens(data):
    all_tokens = []
    for content in data.values():
        all_tokens.extend(content)
    return all_tokens

# Call descriptive_stats for lyrics data
for artist, songs in cleaned_lyrics_data.items():
    print(f"Descriptive stats for {artist}'s lyrics:")
    artist_lyrics_tokens = flatten_tokens(songs)  # Flatten all song tokens
    stats = descriptive_stats(artist_lyrics_tokens, verbose=True)
    print(f"Top 5 words for {artist}'s lyrics: {stats[4]}")  # Top 5 words
```

```python
# Call descriptive_stats for twitter data
for artist, descriptions in cleaned_twitter_data.items():
    print(f"Descriptive stats for {artist}'s Twitter descriptions:")
    artist_twitter_tokens = flatten_tokens({'desc': desc for desc in descriptions})  # Flatten all descriptions
    stats = descriptive_stats(artist_twitter_tokens, verbose=True)
    print(f"Top 5 words for {artist}'s Twitter descriptions: {stats[4]}")  # Top 5 words
```

```
Descriptive stats for robyn's lyrics:
There are 15227 tokens in the data.
There are 2156 unique tokens in the data.
There are 73787 characters in the data.
The lexical diversity is 0.142 in the data.
The 5 most common tokens are:
know: 308
dont: 301
im: 299
love: 275
got: 251
Top 5 words for robyn's lyrics: [('know', 308), ('dont', 301), ('im', 299), ('love', 275), ('got', 251)]
Descriptive stats for cher's lyrics:
There are 35916 tokens in the data.
There are 3703 unique tokens in the data.
There are 172634 characters in the data.
The lexical diversity is 0.103 in the data.
The 5 most common tokens are:
love: 1004
im: 513
know: 486
dont: 440
youre: 333
Top 5 words for cher's lyrics: [('love', 1004), ('im', 513), ('know', 486), ('dont', 440), ('youre', 333)]
Descriptive stats for cher_followers_data's Twitter descriptions:
There are 6 tokens in the data.
There are 6 unique tokens in the data.
There are 38 characters in the data.
The lexical diversity is 1.000 in the data.
The 5 most common tokens are:
missypooh34: 1
melissa: 1
melendez: 1
556532344: 1
0: 1
Top 5 words for cher_followers_data's Twitter descriptions: [('missypooh34', 1), ('melissa', 1), ('melendez', 1
), ('556532344', 1), ('0', 1)]
Descriptive stats for robynkonichiwa_followers_data's Twitter descriptions:
There are 12 tokens in the data.
There are 11 unique tokens in the data.
There are 66 characters in the data.
The lexical diversity is 0.917 in the data.
The 5 most common tokens are:
stand: 2
takemeback: 1
christine: 1
15022058: 1
new: 1
Top 5 words for robynkonichiwa_followers_data's Twitter descriptions: [('stand', 2), ('takemeback', 1), ('chris
tine', 1), ('15022058', 1), ('new', 1)]
Descriptive stats for cher_followers's Twitter descriptions:
There are 1 tokens in the data.
There are 1 unique tokens in the data.
There are 9 characters in the data.
The lexical diversity is 1.000 in the data.
The 5 most common tokens are:
338867174: 1
Top 5 words for cher_followers's Twitter descriptions: [('338867174', 1)]
Descriptive stats for robynkonichiwa_followers's Twitter descriptions:
There are 1 tokens in the data.
There are 1 unique tokens in the data.
There are 8 characters in the data.
The lexical diversity is 1.000 in the data.
The 5 most common tokens are:
44946331: 1
Top 5 words for robynkonichiwa_followers's Twitter descriptions: [('44946331', 1)]
```

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

A: If stopwords were left in the data, the top 5 words would be completely different and dominated as they're more frequent. They are simultaneously less insightful and would provide less clarity on the topics the arists cover.

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: I previously believed Cher would exhibit more lexical diversity due to her longer career and overall, more experience. However, the above data shows that Robyn's lyrics are slightly more lexically diverse than Cher's.

Specialty Statistics

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.

In [49]:
```
assert(emoji.is_emoji("❤ "))
assert(not emoji.is_emoji(":-)"))
```

Emojis 😁

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

In [50]:
```
def extract_emojis(text):
    return [char for char in text if char in emoji.EMOJI_DATA]

def most_common_emojis_by_artist(twitter_data):
    common_emojis = {}

    for artist, descriptions in twitter_data.items():
        all_emojis = []
        for description in descriptions:
            all_emojis.extend(extract_emojis(description))

        emoji_counter = Counter(all_emojis)
        common_emojis[artist] = emoji_counter.most_common(10)

    return common_emojis

common_emojis = most_common_emojis_by_artist(twitter_data)

for artist, emojis in common_emojis.items():
    print(f"Top 10 emojis for {artist}: {emojis}")
```

```
Top 10 emojis for cher_followers_data: [('❤', 94506), ('', 66291), ('♥', 48059), ('', 47174), ('✨', 45846), ('
', 31234), ('□', 31050), ('□', 25195), ('✌', 21963), ('□', 21571)]
Top 10 emojis for robynkonichiwa_followers_data: [('', 6086), ('❤', 5635), ('', 4641), ('♥', 4249), ('✨', 3217
), ('□', 1751), ('✌', 1495), ('♀', 1347), ('□', 1340), ('□', 1200)]
Top 10 emojis for cher_followers: []
Top 10 emojis for robynkonichiwa_followers: []
```

Hashtags

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

In [51]:
```
def extract_hashtags(text):
    return re.findall(r"#\w+", text)

def most_common_hashtags_by_artist(twitter_data):
    common_hashtags = {}

    for artist, descriptions in twitter_data.items():
        all_hashtags = []
        for description in descriptions:
            all_hashtags.extend(extract_hashtags(description))

        hashtag_counter = Counter(all_hashtags)
        common_hashtags[artist] = hashtag_counter.most_common(10)

    return common_hashtags

common_hashtags = most_common_hashtags_by_artist(twitter_data)

for artist, hashtags in common_hashtags.items():
    print(f"Top 10 hashtags for {artist}: {hashtags}")
```

```
Top 10 hashtags for cher_followers_data: [('#BLM', 10100), ('#Resist', 6161), ('#BlackLivesMatter', 4888), ('#r
esist', 3860), ('#FBR', 3330), ('#1', 3111), ('#TheResistance', 3044), ('#blacklivesmatter', 2738), ('#Resistan
ce', 1953), ('#RESIST', 1878)]
Top 10 hashtags for robynkonichiwa_followers_data: [('#BlackLivesMatter', 356), ('#BLM', 345), ('#1', 228), ('#
blacklivesmatter', 222), ('#music', 175), ('#Music', 114), ('#EDM', 87), ('#LGBTQ', 76), ('#blm', 60), ('#TeamF
ollowBack', 59)]
Top 10 hashtags for cher_followers: []
Top 10 hashtags for robynkonichiwa_followers: []
```

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.

In [52]:
```
def extract_first_line(text):
    return text.split('\n', 1)[0]

def most_common_words_in_song_titles(lyrics_data):
    common_words = {}

    for artist, songs in lyrics_data.items():
        all_words = []
        for song_title, lyrics in songs.items():
            first_line = extract_first_line(lyrics)
            words = first_line.split()
            all_words.extend(words)

        word_counter = Counter(all_words)
        common_words[artist] = word_counter.most_common(5)

    return common_words

common_words_in_titles = most_common_words_in_song_titles(lyrics_data)

for artist, words in common_words_in_titles.items():
    print(f"Top 5 words in song titles for {artist}: {words}")
```

```
Top 5 words in song titles for robyn: [('Me', 7), ('You', 7), ('The', 7), ('My', 6), ('To', 6)]
Top 5 words in song titles for cher: [('The', 29), ('To', 28), ('"The', 24), ('Of', 21), ('"I', 21)]
```

Song Lengths

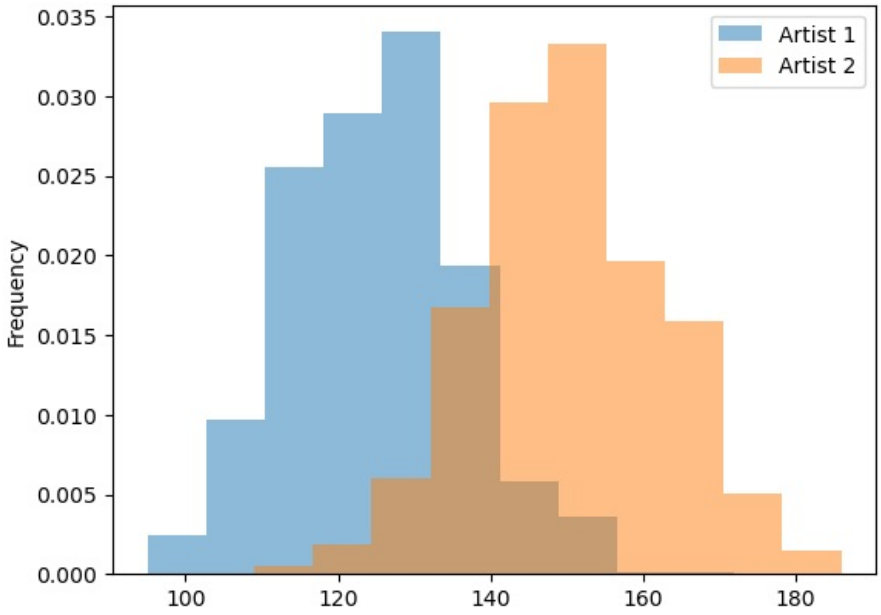
For each artist, a histogram of song lengths (in terms of number of tokens). If you put the song lengths in a data frame with an artist column, matplotlib will make the plotting quite easy. An example is given to help you out.

In [54]:
```
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)
```

Out[54]:
```
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
```

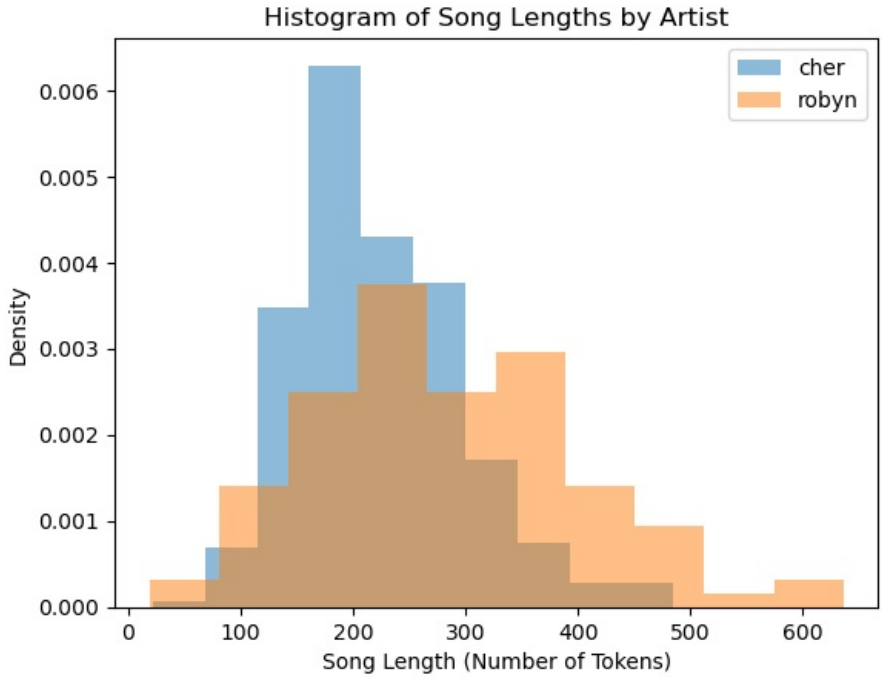


In [58]:
```
def get_song_lengths(lyrics_data):
    song_lengths = []
    for artist, songs in lyrics_data.items():
        for song_title, lyrics in songs.items():
            tokens = lyrics.split()
            song_lengths.append({"artist": artist, "length": len(tokens)})
    return song_lengths

song_lengths = get_song_lengths(lyrics_data)

df = pd.DataFrame(song_lengths)
```

```
df.groupby('artist')['length'].plot(kind="hist", density=True, alpha=0.5, legend=True)
plt.xlabel('Song Length (Number of Tokens)')
plt.ylabel('Density')
plt.title('Histogram of Song Lengths by Artist')
plt.show()
```



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: Matches one or more whitespace characters

In [56]:
```
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)])
```

In [59]:
```
def collapse_whitespace(text):
    return re.sub(r'\s+', ' ', text).strip()

def get_song_lengths(lyrics_data):
    song_lengths = []
    for artist, songs in lyrics_data.items():
        for song_title, lyrics in songs.items():
            lyrics = collapse_whitespace(lyrics)
            tokens = lyrics.split()
            song_lengths.append({"artist": artist, "length": len(tokens)})
    return song_lengths

song_lengths = get_song_lengths(lyrics_data)

df = pd.DataFrame(song_lengths)

df.groupby('artist')['length'].plot(kind="hist", density=True, alpha=0.5, legend=True)
plt.xlabel('Song Length (Number of Tokens)')
plt.ylabel('Density')
plt.title('Histogram of Song Lengths by Artist')
plt.show()
```