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 [65]:

```
import nltk
#nltk.download()
```

In [66]:

```
import os
import re
#!pip install emoji
import emoji
import nltk
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 [67]:
```
# change `data_location` to the location of the folder on your machine.
data_location = "C:/Users/Grigor/OneDrive/Desktop/Master_Program/ADS509/Module_2/"

# These subfolders should still work if you correctly stored the
# data from the Module 1 assignment
twitter_folder = "twitter/"
lyrics_folder = "lyrics"
```

In [68]:
```
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_
        and num_tokens most common tokens. Return a list with the number of tokens, num
        of unique tokens, lexical diversity, and number of characters.

    """

    # Fill in the correct values here.
    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    lexical_diversity = num_unique_tokens/num_tokens
    num_characters = len("".join(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
        if num_tokens > 0:
            print(counts.most_common(num_tokens))
    return([len(tokens),
           len(set(tokens)),
           len("".join(tokens)),
           len(set(tokens))/len(tokens)])

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

In [69]:
```
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.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [69], in <cell line: 2>()
      1 text = """here is some example text with other example text here in this tex
t""".split()
----> 2 assert(descriptive_stats(text, verbose=True)[0] == 13)
      3 assert(descriptive_stats(text, verbose=False)[1] == 9)
      4 assert(abs(descriptive_stats(text, verbose=False)[2] - 0.69) < 0.02)

Input In [68], in descriptive_stats(tokens, num_tokens, verbose)
     22     # print the five most common tokens
     23     if num_tokens > 0:
---> 24         print(counts.most_common(num_tokens))
     25 return([len(tokens),
     26         len(set(tokens)),
     27         len("".join(tokens)),
     28         len(set(tokens))/len(tokens)])
     30 return([num_tokens, num_unique_tokens,
     31          lexical_diversity,
     32          num_characters])

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

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

A: Assert statements check for conditions and helps to find issues wihtin your code. If the asserrtion is set to false then an AssertionError will be thrown.

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 [70]:
```python
# Read in the lyrics data
lyrics_data = defaultdict(list)

for item in os.listdir(data_location + lyrics_folder):
    if os.path.isdir(data_location + lyrics_folder + item):
        for lyric_page in os.listdir(data_location + lyrics_folder + item) :
            artist,song = lyric_page.split("_")
            song = song.replace(".txt","")
            lyrics_data[item][song]= open(data_location + lyrics_folder + item)
```

In [71]:
```python
# Read in the twitter data
twitter_files = os.listdir(data_location + twitter_folder)
desc_files = [f for f in twitter_files if "followers_data" in f]

twitter_data = defaultdict(list)
```

```
for f in desc_files:
    artist = f.split("_")[0]

    with open(data_location + twitter_folder + f, 'r', encoding = 'utf8') as infile:
        next(infile)
        for idx, line in enumerate(infile.readlines()):
            line = line.strip().split("\t")
            if len(line) == 7 :
                twitter_data[artist].append(line[6])
```

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 [72]:
```
punctuation = set(punctuation) # speeds up comparison
```

In [73]:
```
# create your clean twitter data here
```

In [75]:
```
# create your clean lyrics data here
lyrics_processed = defaultdict(list)

for artist in lyrics_data :
    for song in lyrics_data[artist] :
        lyrics = "".join([ch for ch in lyrics_data[artist][song] if ch ])
        lyrics = [item.lower().strip() for item in lyrics.split()]
        lyrics = [item for item in lyrics if item not in sw]
        lyrics_processed[artist].extend(lyrics)
```

```
defaultdict(<class 'list'>, {})
```

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 [63]:
```
# calls to descriptive_stats here
descriptive_stats(lyrics_processed['robyn'], num_tokens = 10)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [63], in <cell line: 2>()
      1 # calls to descriptive_stats here
----> 2 descriptive_stats(lyrics_processed['robyn'], num_tokens = 10)

Input In [56], in descriptive_stats(tokens, num_tokens, verbose)
     11 num_tokens = len(tokens)
     12 num_unique_tokens = len(set(tokens))
```

```
---> 13 lexical_diversity = num_unique_tokens/num_tokens
     14 num_characters = len("".join(tokens))
     16 if verbose :

ZeroDivisionError: division by zero
```

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

A: Stop words are used to exclude certian words from searches. By removing these words we remove low level information. If we left these stopwords the top 5 words would be different as more low level words may have shown up in the searches.

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: Lexical diversity between artists i thought would be very diverse. I was unable to run this line of code so I am unable to answer the second part of this question.

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 [76]:
```
assert(emoji.is_emoji("❤️"))
assert(not emoji.is_emoji(":-)"))
```

Emojis 😁

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

In [77]:
```
# Your code here
emojis = defaultdict(list)

for artist in twitter_data :
    for desc in twitter_data[artist] :
        emojis[artist].extend([ch for ch in desc if emoji.is_emoji(ch)])
```

In [78]:
```
for artist in emojis :
    print(artist)
    print(Counter(emojis[artist]).most_common(10))
```

```
cher
[('🖤', 79223), ('🌈', 47549), ('♥', 33978), ('🏳', 33412), ('✨', 29468), ('💙', 2137
9), ('⍰', 20930), ('🌊', 20223), ('✌', 16773), ('💜', 16550)]
robynkonichiwa
[('🖤', 4783), ('🌈', 4685), ('🏳', 3528), ('♥', 3103), ('✨', 2223), ('⍰', 1495),
('✌', 1189), ('⍰', 1139), ('♀', 836), ('💙', 809)]
```

Hashtags

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

In [79]:
```python
# Your code here
hashtags = defaultdict(list)

for artist in twitter_data :
    for desc in twitter_data[artist] :

        hashtags[artist].extend([item.lower() for item in desc.split()])
```

In [80]:
```python
for artist in hashtags :
    print(artist)
    print(Counter(hashtags[artist]).most_common(10))
```

```
cher
[('and', 565427), ('i', 436327), ('a', 416133), ('the', 401094), ('to', 347286), ('of',
325564), ('my', 278188), ('in', 225889), ('love', 198887), ('is', 188702)]
robynkonichiwa
[('and', 44386), ('the', 34224), ('i', 34046), ('a', 30302), ('of', 25780), ('to', 2237
8), ('in', 20833), ('my', 19064), ('&', 17131), ('|', 15556)]
```

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 [81]:
```python
title_tokens = defaultdict(list)

for artist in lyrics_data :
    for song, page in lyrics_data[artist].items() :

        page = page.split("\n")

        title_tokens[artist].extend([item.lower() for item in page[0].lyrics_data])
```

In [82]:
```python
for artist in title_tokens :
    print(artist)
    print(Counter(title_tokens[artist]).most_common(5))
```

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 [83]:
```
artist_vec = []
song_vec = []
lyrics_vec = []

for artist in lyrics_data :
    for song, lyrics in lyrics_data[artist].items():
        artist_vec.append(artist)
        song_vec.append(song)
        lyrics_vec.append(lyrics)

lyrics_df = pd.DataFrame()
lyrics_df['artist'] = artist_vec
lyrics_df['song'] = song_vec
lyrics_df['lyrics'] = lyrics_vec

lyrics_df.head()
```

Out[83]:

artist	song	lyrics

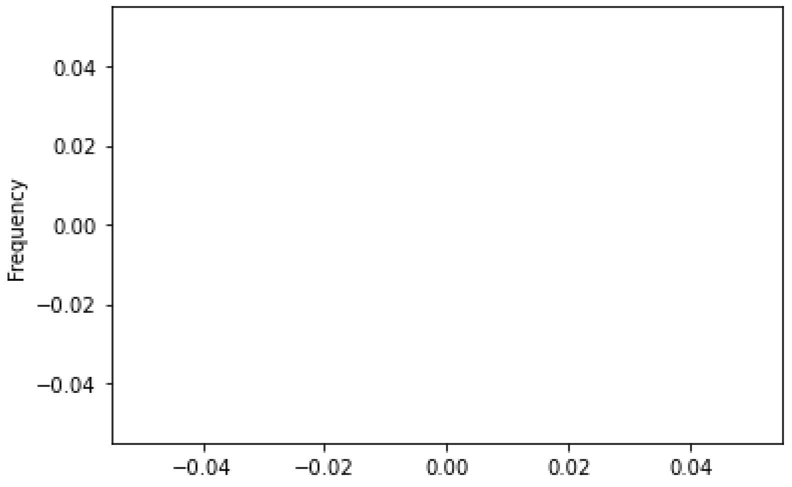
In [84]:
```
collapse_whitespace = re.compile(r'\s+')

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

In [85]:
```
lyrics_df['lyric_token'] = lyrics_df['lyrics'].apply(tokenize_lyrics)
lyrics_df['lyric_len'] = lyrics_df['lyric_token'].apply(len)
```

In [86]:
```
lyrics_df.groupby('artist')["lyric_len"].plot(kind = "hist", density = True)
```

```
C:\Users\Grigor\.conda\envs\ADS_502\lib\site-packages\numpy\lib\histograms.py:905: Runti
meWarning: invalid value encountered in true_divide
  return n/db/n.sum(), bin_edges
```

Out[86]:
```
Series([], Name: lyric_len, dtype: float64)
```

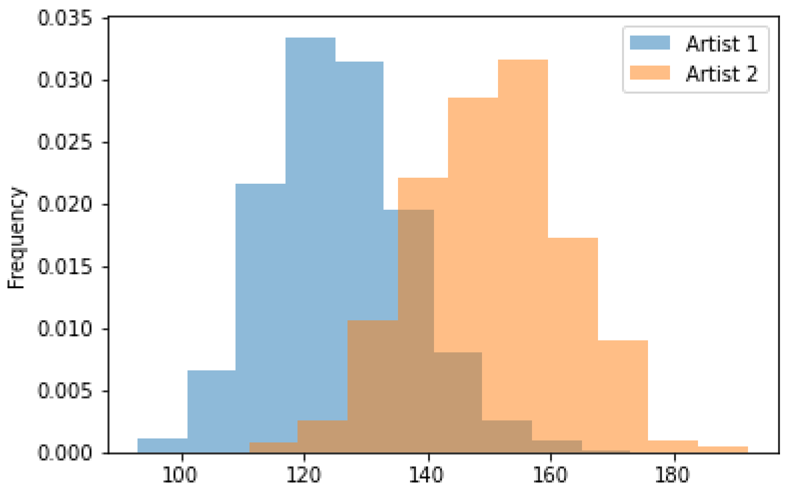


In [87]:

```
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(
})

df.groupby('artist')['length'].plot(kind="hist",density=True,alpha=0.5,legend=True)
```

Out[87]:

```
artist
Artist 1    AxesSubplot(0.125,0.125;0.775x0.755)
Artist 2    AxesSubplot(0.125,0.125;0.775x0.755)
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:

In [88]:

```
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 [ ]: # Your lyric length comparison chart here.
```