ADS 509 Module 3: Group Comparison

The task of comparing two groups of text is fundamental to textual analysis. There are innumerable applications: survey respondents from different segments of customers, speeches by different political parties, words used in Tweets by different constituencies, etc. In this assignment you will build code to effect comparisons between groups of text data, using the ideas learned in reading and lecture.

This assignment asks you to analyze the lyrics and Twitter descriptions for the two artists you selected in Module 1. If the results from that pull were not to your liking, you are welcome to use the zipped data from the “Assignment Materials” section. Specifically, you are asked to do the following:

- Read in the data, normalize the text, and tokenize it. When you tokenize your Twitter descriptions, keep hashtags and emojis in your token set.
- Calculate descriptive statistics on the two sets of lyrics and compare the results.
- For each of the four corpora, find the words that are unique to that corpus.
- Build word clouds for all four corpora.

Each one of the analyses has a section dedicated to it below. Before beginning the analysis there is a section for you to read in the data and do your cleaning (tokenization and normalization).

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 [3]:

```
import os
import re
import emoji
import pandas as pd

from collections import Counter, defaultdict
```

```
from nltk.corpus import stopwords
from string import punctuation
#!pip install wordcloud
from wordcloud import WordCloud

from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer
```

In [4]:

```
# Use this space for any additional import statements you need
```

In [6]:

```
# Place any addtional functions or constants you need here.

# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison
tw_punct = punctuation - {"#"}

# Stopwords
sw = stopwords.words("english")

# Two useful regex
whitespace_pattern = re.compile(r"\s+")
hashtag_pattern = re.compile(r"^#[0-9a-zA-Z]+")

# It's handy to have a full set of emojis
all_language_emojis = set()

for country in emoji.EMOJI_DATA :
    for em in emoji.EMOJI_DATA[country] :
        all_language_emojis.add(em)

# and now our functions
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, and num_tokens most common
        tokens. Return a list of
    """

    # Place your Module 2 solution here
    # 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])
    return(0)



def contains_emoji(s):

    s = str(s)
    emojis = [ch for ch in s if emoji.is_emoji(ch)]

    return(len(emojis) > 0)


def remove_stop(tokens) :
    # modify this function to remove stopwords
    return(tokens)

def remove_punctuation(text, punct_set=tw_punct) :
    return("".join([ch for ch in text if ch not in punct_set]))

def tokenize(text) :
    """ Splitting on whitespace rather than the book's tokenize function. That
        function will drop tokens like '#hashtag' or '2A', which we need for Twitter. "

    # modify this function to return tokens
    return(text)

def prepare(text, pipeline) :
    tokens = str(text)

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

In [7]:

```
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 [7], 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 [6], in descriptive_stats(tokens, num_tokens, verbose)
```

```
     42     # print the five most common tokens
     43     if num_tokens > 0:
---> 44         print(counts.most_common(num_tokens))
     45 return([len(tokens),
     46        len(set(tokens)),
     47        len("".join(tokens)),
     48        len(set(tokens))/len(tokens)])
     50 return([num_tokens, num_unique_tokens,
     51         lexical_diversity,
     52         num_characters])

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

Data Ingestion

Use this section to ingest your data into the data structures you plan to use. Typically this will be a dictionary or a pandas DataFrame.

In [10]:

```
# Feel fre to use the below cells as an example or read in the data in a way you prefer

data_location = "C:/Users/Grigor/OneDrive/Desktop/Master_Program/ADS509/Module_2/"
# change to your location if it is not in the same directory as your notebook
twitter_folder = "twitter/"
lyrics_folder = "lyrics/"

artist_files = {'cher':'cher_followers_data.txt',
                'robyn':'robynkonichiwa_followers_data.txt'}
```

In [11]:

```
twitter_data = pd.read_csv(data_location + twitter_folder + artist_files['cher'],
                           sep="\t",
                           quoting=3)

twitter_data['artist'] = "cher"
```

In [12]:

```
twitter_data_2 = pd.read_csv(data_location + twitter_folder + artist_files['robyn'],
                             sep="\t",
                             quoting=3)
twitter_data_2['artist'] = "robyn"

twitter_data = pd.concat([
    twitter_data,twitter_data_2])

del(twitter_data_2)
```

In [13]:

```
# read in the lyrics here
lyrics_data = pd.read_csv(data_location + twitter_folder + artist_files['cher'],
                           sep="\t",
                           quoting=3)

lyrics_data['artist'] = "cher"
```

In [14]:
```
# read in the lyrics here
lyrics_data_2 = pd.read_csv(data_location + twitter_folder + artist_files['robyn'],
                             sep="\t",
                             quoting=3)
lyrics_data_2['artist'] = "robyn"

lyrics_data = pd.concat([
    lyrics_data,lyrics_data_2])

del(lyrics_data_2)
```

Tokenization and Normalization

In this next section, tokenize and normalize your data. We recommend the following cleaning.

Lyrics

- Remove song titles
- Casefold to lowercase
- Remove stopwords (optional)
- Remove punctuation
- Split on whitespace

Removal of stopwords is up to you. Your descriptive statistic comparison will be different if you include stopwords, though TF-IDF should still find interesting features for you. Note that we remove stopwords before removing punctuation because the stopword set includes punctuation.

Twitter Descriptions

- Casefold to lowercase
- Remove stopwords
- Remove punctuation other than emojis or hashtags
- Split on whitespace

Removing stopwords seems sensible for the Twitter description data. Remember to leave in emojis and hashtags, since you analyze those.

In [18]:
```
artists = []
songs = []
lyrics = []
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","")
            artists.append(artist)
            songs.append(song)

            with open(data_location + lyrics_folder + item + "/" + lyric_page) as infil
                next(infile) # skip title
```

```
                next(infile) # skip blank
                next(infile) # skip blank
                next(infile) # skip final blank

                lyrics.append(infile.read())

lyrics_data = pd.DataFrame()
lyrics_data['artist'] = artists
lyrics_data['song'] = songs
lyrics_data['lyrics'] = lyrics
```

In [19]:

```
# apply the `pipeline` techniques from BTAP Ch 1 or 5

my_pipeline = [str.lower, remove_punctuation, tokenize, remove_stop]

lyrics_data["tokens"] = lyrics_data["lyrics"].apply(prepare,pipeline=my_pipeline)
lyrics_data["num_tokens"] = lyrics_data["tokens"].map(len)

twitter_data["tokens"] = twitter_data["description"].apply(prepare,pipeline=my_pipeline
twitter_data["num_tokens"] = twitter_data["tokens"].map(len)
```

In [20]:

```
twitter_data['has_emoji'] = twitter_data["description"].apply(contains_emoji)
```

Let's take a quick look at some descriptions with emojis.

In [21]:

```
twitter_data[twitter_data.has_emoji].sample(10)[["artist","description","tokens"]]
```

Out[21]:

	artist	description	tokens
557473	cher	University of Alabama 2019. She/her. Life is t...	university of alabama 2019 sheher life is too ...
593346	cher	It's not about whether you get knocked down, i...	its not about whether you get knocked down its...
2632495	cher	Azul de profesión 💙 T_Tniversidad de Chile 💙 ❤️ ...	azul de profesión 💙 ttniversidad de chile 💙 ❤️ ...
899665	cher	🌎 🌍 🌏 man miss me wit alllll the negative shit	🌎 🌍 🌏 man miss me wit alllll the negative shit
1987035	cher	i love drag ❤️ ,i love music ,i love harry pott...	i love drag ❤️ i love music i love harry potter...
1745904	cher	🍇 Cheers 🍷 🍇	🍇 cheers 🍷 🍇
289672	cher	💕 🌼 Bloom where you are planted. 🌷 💕	💕 🌼 bloom where you are planted 🌷 💕
168127	cher	she/her - ou 25 - tpwk - ♍	sheher ou 25 tpwk ♍
3132800	cher	#CysticFibrosis. Lung Transplant @PennMedicine...	#cysticfibrosis lung transplant pennmedicine 1...
677820	cher	Always be a little kinder than necessary. 🤘🏻	always be a little kinder than necessary 🤘🏻

With the data processed, we can now start work on the assignment questions.

Q: What is one area of improvement to your tokenization that you could theoretically carry out? (No need to actually do it; let's not make perfect the enemy of good enough.)

A: I feel like there are special characters within the description and tokens that may need some more explaining.

Calculate descriptive statistics on the two sets of lyrics and compare the results.

In [57]:

```
# your code here
#descriptive_stats(lyrics_data['artist'], num_tokens = 10)
cher = lyrics_data[lyrics_data['artist'] == 'cher']
cher_dstats = [element for list_ in cher['tokens'].values for element in list_]

robyn = lyrics_data[lyrics_data['artist'] == 'robyn']
robyn_dstats = [element for list_ in robyn['tokens'].values for element in list_]
```

In [60]:

```
print("\nCher :")
print(descriptive_stats(cher_dstats))
```

```
Cher :
There are 332560 tokens in the data.
There are 48 unique tokens in the data.
There are 332560 characters in the data.
The lexical diversity is 0.000 in the data.

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [60], in <cell line: 2>()
      1 print("\nCher :")
----> 2 print(descriptive_stats(cher_dstats))

Input In [6], in descriptive_stats(tokens, num_tokens, verbose)
     42     # print the five most common tokens
     43     if num_tokens > 0:
---> 44         print(counts.most_common(num_tokens))
     45 return([len(tokens),
     46        len(set(tokens)),
     47        len("".join(tokens)),
     48        len(set(tokens))/len(tokens)])
     50 return([num_tokens, num_unique_tokens,
     51         lexical_diversity,
     52         num_characters])

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

In [61]:

```
print("\nRobyn :")
print(descriptive_stats(robyn_dstats))
```

```
Robyn :
There are 141288 tokens in the data.
There are 51 unique tokens in the data.
There are 141288 characters in the data.
The lexical diversity is 0.000 in the data.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [61], in <cell line: 2>()
      1 print("\nRobyn :")
----> 2 print(descriptive_stats(robyn_dstats))

Input In [6], in descriptive_stats(tokens, num_tokens, verbose)
     42     # print the five most common tokens
     43     if num_tokens > 0:
---> 44         print(counts.most_common(num_tokens))
     45 return([len(tokens),
     46        len(set(tokens)),
     47        len("".join(tokens)),
     48        len(set(tokens))/len(tokens)])
     50 return([num_tokens, num_unique_tokens,
     51         lexical_diversity,
     52         num_characters])

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

Q: what observations do you make about these data?

A: There are many tokens compared to unique tokens. Also for both artists it appears the lexical diversity is 0.

Find tokens uniquely related to a corpus

Typically we would use TF-IDF to find unique tokens in documents. Unfortunately, we either have too few documents (if we view each data source as a single document) or too many (if we view each description as a separate document). In the latter case, our problem will be that descriptions tend to be short, so our matrix would be too sparse to support analysis.

To avoid these problems, we will create a custom statistic to identify words that are uniquely related to each corpus. The idea is to find words that occur often in one corpus and infrequently in the other(s). Since corpora can be of different lengths, we will focus on the *concentration* of tokens within a corpus. "Concentration" is simply the count of the token divided by the total corpus length. For instance, if a corpus had length 100,000 and a word appeared 1,000 times, then the concentration would be $\frac{1000}{100000} = 0.01$. If the same token had a concentration of 0.005 in another corpus, then the concentration ratio would be $\frac{0.01}{0.005} = 2$. Very rare words can easily create infinite ratios, so you will also add a cutoff to your code so that a token must appear at least n times for you to return it.

An example of these calculations can be found in this spreadsheet. Please don't hesitate to ask questions if this is confusing.

In this section find 10 tokens for each of your four corpora that meet the following criteria:

1. The token appears at least `n` times in all corpora
2. The tokens are in the top 10 for the highest ratio of appearances in a given corpora vs appearances in other corpora.

You will choose a cutoff for yourself based on the side of the corpus you're working with. If you're working with the Robyn-Cher corpora provided, n=5 seems to perform reasonably well.

In [111…

```
def compare_conc(corpus_1, corpus_2, req_tokens = 5, num_word = 10):
    tokens_1 = Counter(corpus_1)
    tokens_1 = pd.DataFrame.from_dict(tokens_1, orient = 'index').reset_index()
    tokens_1 = tokens_1.rename(columns = {'index' : 'token', 0 : 'count'})
    tokens_1 = tokens_1[tokens_1['count'] >= req_tokens]
    cor_length_1 = len(corpus_1)
    tokens_1['Conc_one'] = tokens_1['count']/cor_length_1

    tokens_2 = Counter(corpus_2)
    tokens_2 = pd.DataFrame.from_dict(tokens_2, orient = 'index').reset_index()
    tokens_2 = tokens_2.rename(columns = {'index' : 'token', 0 : 'count'})
    tokens_2 = tokens_2[tokens_2['count'] >= req_tokens]
    cor_length_2 = len(corpus_2)
    tokens_2['Conc_two'] = tokens_2['count']/cor_length_2

    merge_token = pd.merge(tokens_1, tokens_2, how = "outer", on = ['token', 'token'])
    merge_token = merge_token.dropna()
    merge_token['one vs two'] = merge_token['Conc_one']/merge_token['Conc_two']
    merge_token['two vs one'] = merge_token['Conc_two']/merge_token['Conc_one']
    one_vs_two_res = merge_token[['token', 'one vs two']].sort_values(by = 'one vs two'
                                                                      ascending = False)
    two_vs_one_res = merge_token[['token', 'two vs one']].sort_values(by = 'two vs one'
                                                                      ascending = False)
    print("Corpus One vs. Two:\n", one_vs_two_res, "\n\nCorpus Two vs. One:\n", two_vs_
```

In [112…

```
compare_conc(cher_dstats, robyn_dstats)
```

```
Corpus One vs. Two:
    index token  one vs two
0     27     q    2.063555
1     21     v    1.210375
2     12     f    1.131808
3      9     a    1.112474
4     17     h    1.101959
5     19     w    1.094263
6     13     r    1.078300
7      0     s    1.058977
8     14     e    1.058545
9     15     d    1.058419

Corpus Two vs. One:
    index token  two vs one
0     35     ¢   12.945756
1     36     €   12.945756
2     32     â    8.099751
3     38     1    7.649765
4     30     ã    7.061322
5     25     8    6.355189
6     29     2    5.884435
7     31     ƒ    5.456476
8     28     x    1.575360
9      4     k    1.423613
```

Q: What are some observations about the top tokens? Do you notice any interesting items on the list?

A: ¢ and € are identical with 12.945756 value.

Build word clouds for all four corpora.

For building wordclouds, we'll follow exactly the code of the text. The code in this section can be found here. If you haven't already, you should absolutely clone the repository that accompanies the book.

In [110…

```
from matplotlib import pyplot as plt

def wordcloud(word_freq, title=None, max_words=200, stopwords=None):

    wc = WordCloud(width=800, height=400,
                   background_color= "black", colormap="Paired",
                   max_font_size=150, max_words=max_words)

    # convert data frame into dict
    if type(word_freq) == pd.Series:
        counter = Counter(word_freq.fillna(0).to_dict())
    else:
        counter = word_freq

    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                              if token not in stopwords}
    wc.generate_from_frequencies(counter)

    plt.title(title)

    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")


def count_words(df, column='tokens', preprocess=None, min_freq=2):

    # process tokens and update counter
    def update(doc):
        tokens = doc if preprocess is None else preprocess(doc)
        counter.update(tokens)

    # create counter and run through all data
    counter = Counter()
    df[column].map(update)

    # transform counter into data frame
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    freq_df = freq_df.query('freq >= @min_freq')
    freq_df.index.name = 'token'

    return freq_df.sort_values('freq', ascending=False)
```

Q: What observations do you have about these (relatively straightforward) wordclouds?

A: