

# Group Comparison

September 21, 2024

## 0.1 Ravita Kartawinata

[Ravita's Git](#)

## 1 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 for the two artists you selected in Module 1 and the Twitter descriptions pulled for Robyn and Cher. 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).

### 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.*

```
[1]: import os
import re
import emoji
import pandas as pd

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

from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer

[2]: # Use this space for any additional import statements you need
from nltk.tokenize import word_tokenize
from nltk.tokenize import TweetTokenizer

from matplotlib import pyplot as plt

[3]: # Place any additional functions or constants you need here.
punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

# 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,
    ↪ and number of characters
    # Fill in the correct values here.
    num_tokens_tot = len(tokens)
```

```

num_unique_tokens = len(set(tokens))
num_characters = sum(
    len(token) for token in tokens)

# Get lexical diversity (https://en.wikipedia.org/wiki/Lexical_diversity),
# and num_tokens, most common tokens.
lexical_diversity = num_unique_tokens / num_tokens_tot \
    if num_tokens_tot > 0 else 0.0
token_counts = Counter(tokens)
most_common_tokens = token_counts.most_common(num_tokens)

if verbose :
    print(f"There are {num_tokens_tot} 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 a list with the number of tokens, number of unique tokens,
# lexical diversity, and number of characters, most common tokens
return [num_tokens_tot, num_unique_tokens, lexical_diversity, \
        num_characters, most_common_tokens]

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
    stop_words = set(stopwords.words('english'))
    cleaned_tokens = [word for word in tokens if word not in stop_words]
    return(cleaned_tokens)

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

def lyrics_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. """
    tokens = word_tokenize(text.lower())
    return(tokens)

def twitter_tokenize(text) :

```

```

    tokens = text.split()
    return(tokens)

def normalize_whitespace(text):
    text = re.sub(r'\n+', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip()

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

    for transform in pipeline :
        tokens = transform(tokens)
    return(tokens)

```

## 1.2 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.

```

[4]: # Feel free to use the below cells as an example or read in the data in a way
      ↳ you prefer

```

```

data_location = "/users/rkartawi/Desktop/Ravita/MSADS/509/ads509-tm-scrape" #
      ↳ change to your location if it is not in the same directory as your notebook
twitter_folder = os.path.join(data_location, "twitter")
lyrics_folder = os.path.join(data_location, "lyrics")

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

```

```

[5]: file_path_cher = os.path.join(twitter_folder, artist_files['cher'])
      twitter_data_cher = pd.read_csv(file_path_cher, sep="\t", quoting=3)
      twitter_data_cher['artist'] = 'cher'

```

```

[6]: file_path_robyn = os.path.join(twitter_folder, artist_files['robyn'])
      twitter_data_robyn = pd.read_csv(file_path_robyn, sep="\t", quoting=3)
      twitter_data_robyn['artist'] = 'robyn'

      twitter_data = pd.concat([twitter_data_cher, twitter_data_robyn],
      ↳ ignore_index=True)
      twitter_data = twitter_data.dropna(subset=['description'])

      # print(twitter_data.head())
      # pd.set_option('display.max_colwidth', None)
      # twitter_data['description'].head(12)

```

```
[7]: # Read in the lyrics data
def read_lyrics_data(lyrics_folder):
    lyrics_data = []
    song_title_pattern = re.compile(r'"s*("[^"]+)\s*"'')
    for artist in os.listdir(lyrics_folder):
        artist_folder = os.path.join(lyrics_folder, artist)

        for song_file in os.listdir(artist_folder):
            song_path = os.path.join(artist_folder, song_file)
            with open(song_path, 'r', encoding='utf-8') as f:
                lyrics_content = f.read().strip()
                #removing some of words and newlines in lyrics
                lyrics_content = re.sub(r'^^[^\\n]*\\n+"[^"]*"\\s*lyrics\\n+', '', lyrics_content)
                lyrics_content = lyrics_content.replace('Lyrics', '')
                lyrics_content = lyrics_content.replace('AZLyrics.com', '')
                #try to get the title only from lyrics
                match = song_title_pattern.search(lyrics_content) #chatgpt
                if match:
                    song_title = match.group(1)
                else:
                    song_title = os.path.splitext(song_file)[0]
            lyrics_data.append({
                'artist': artist,
                'song': song_file,
                'song_title': song_title,
                'lyrics': lyrics_content
            })
    lyrics_df = pd.DataFrame(lyrics_data)
    return lyrics_df
lyrics_data = read_lyrics_data(lyrics_folder)
#lyrics_data.head()
```

### 1.3 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 stopwords 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.

```
[8]: # apply the `pipeline` techniques from BTAP Ch 1 or 5

my_pipeline = [str.lower, normalize_whitespace, remove_punctuation,
               ↳ lyrics_tokenize, remove_stop]

lyrics_data["tokens"] = lyrics_data["lyrics"].apply(lambda text: prepare(text,
               ↳ my_pipeline))
lyrics_data["num_tokens"] = lyrics_data["tokens"].apply(len)
```

```
[9]: # lyrics_data.head(5)
```

```
[10]: my_pipeline = [str.lower, normalize_whitespace, remove_punctuation,
                    ↳ twitter_tokenize, remove_stop]

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

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

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

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

```
[12]:      artist      description \
817496  cher      The best Diet Programs of the Momen...
43442   cher      +3+      +3+
2409875 cher  rroll dor-f n @arrolladora @mikemiramontexz @...
1317226 cher  Press Secretary for @NC_Governor, previously a...
3961033 cher      Love Cher since 30 years,...she's my heart
1794890 cher  Broke Educator and student. Animal enthusiast...
1390733 cher      15 years old Bulgaria
279970  cher      me encantan los
1397367 cher  Always Happy  Always Positive Too Busy Enjoy...
772179  cher  Photographer | Runner | Fitness Freak | Sports...

tokens
817496  [ , best, diet, programs, moment, fin...
```

```

43442                                [3,      , 3]
2409875  [rroll dorf n, arrolladora, mikemiramontexz, ...
1317226  [press, secretary, ncgovernor, previously, sar...
3961033      [love, cher, since, 30, yearsshes, heart ]
1794890  [broke, educator, student, animal, enthusiast,...
1390733      [15, years, old bulgaria,  ]
279970      [encantan, los,  ]
1397367  [always, happy, , always, positive, busy, en...
772179  [photographer, runner, fitness, freak, sportsf...

```

```
[13]: # twitter_data.head(12)
```

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: Tokenization between lyrics and twitter has a different approach. In twitter/description, we would like to maintain #(hashtag) integrity to the word next to it. Using `word_tokenize(text)` or the book `tokenize`, it will separate # as a token by itself. In addition, building a pipeline also helps on what data cleaning sequence should be performed first. If we use `TweetTokenizer()`, words such as 'I'll', 'it's' are still part of the token since maintaining original context would be an important piece for sentiment analysis especially from social media data. In this HW, I used `text.split()` to maintain # as it splits "I'll" and remove these tokens from `remove_punctuation` and `stopword` functions as I excluded a subject sentiment analysis.

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

```

[14]: # your code here
# Focus on tokens and flatten it
# lyrics_tokens = [token for tokens_list in lyrics_data['tokens'] for token in
↳ tokens_list]
# print("Descriptive Statistics for Lyrics Data:")
# lyrics_stats = descriptive_stats(lyrics_tokens)

# Group by artist and concatenate tokens for each artist
grouped_tokens = lyrics_data.groupby('artist')['tokens'].apply(lambda x: [token
↳ for tokens_list in x for token in tokens_list])

for artist, tokens in grouped_tokens.items():
    print(f"\nDescriptive Statistics lyrics for Artist: {artist}")
    lyrics_stats = descriptive_stats(tokens)

```

Descriptive Statistics lyrics for Artist: cher

There are 5396 tokens in the data.

There are 1395 unique tokens in the data.

There are 25854 characters in the data.

The lexical diversity is 0.259 in the data.

The 5 most common tokens are:

come: 93

dont: 90

baby: 83

see: 76

love: 58

Descriptive Statistics lyrics for Artist: robyn

There are 3097 tokens in the data.

There are 656 unique tokens in the data.

There are 15026 characters in the data.

The lexical diversity is 0.212 in the data.

The 5 most common tokens are:

love: 99

robyn: 75

show: 74

know: 72

dont: 60

```
[15]: grouped_tokens = twitter_data.groupby('artist')['tokens'].apply(lambda x:
    ↪ [token for tokens_list in x for token in tokens_list])

for artist, tokens in grouped_tokens.items():
    print(f"\nDescriptive Statistics twitter for Artist: {artist}")
    twitter_stats = descriptive_stats(tokens)
```

Descriptive Statistics twitter for Artist: cher

There are 15673833 tokens in the data.

There are 1555372 unique tokens in the data.

There are 93389117 characters in the data.

The lexical diversity is 0.099 in the data.

The 5 most common tokens are:

love: 213522

im: 139051

life: 122679

music: 86733

de: 72970

Descriptive Statistics twitter for Artist: robyn

There are 1495769 tokens in the data.

There are 258391 unique tokens in the data.

There are 9158537 characters in the data.

The lexical diversity is 0.173 in the data.

The 5 most common tokens are:

music: 14858

love: 11615



im: 9049  
och: 7922  
life: 7354

Q: what observations do you make about these data?

A: Comparing lyric stats between the artists, we found that there are some common word such as “love”, “dont” that are used quiet often in their songs. While comparing twitter stat between the artists, we can see that the follower also have the same sentiment of “love” in the artists’ music and how their songs might relate to “life”. When we compare lyric and twitter for the same artist, “love” is the most common token that is used in lyric, twitter for both of the artists.

## 1.5 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.

```
[16]: n = 5
# calculate concentration ratios
def concentration_ratios_by_artist(artist_corpus, artist_name, other_corpus,
    other_name):
    token_counts_artist = Counter()
    for tokens in artist_corpus['tokens']:
        token_counts_artist.update(tokens)

    total_tokens_artist = sum(token_counts_artist.values())
```

```

        filtered_tokens = ((token, count) for token, count in token_counts_artist.
↳items() if count >= n)
        concentration_artist = {token: (count, count / total_tokens_artist) for
↳token, count in filtered_tokens}

        token_counts_other = Counter()
        for tokens in other_corpus['tokens']:
            token_counts_other.update(tokens)

        total_tokens_other = sum(token_counts_other.values())

        results = []
        for token, (count_artist, concentration_artist_value) in
↳concentration_artist.items():
            count_other = token_counts_other.get(token, 0)
            concentration_other_value = count_other / total_tokens_other if
↳total_tokens_other > 0 else 0

            results.append({
                'Token': token,
                f'{artist_name} Count': count_artist,
                f'{other_name} Count': count_other,
                f'Concentration {artist_name}': f"{concentration_artist_value * 100:
↳.2f}%",
                f'Concentration {other_name}': f"{concentration_other_value * 100:
↳.2f}%",
            })

        df = pd.DataFrame(results)

        # get the top 10
        return df.sort_values(by=f'{artist_name} Count', ascending=False).head(10)

```

```

[17]: # Twitter data analysis
robyn_twitter_corpus = twitter_data[twitter_data['artist'] == 'robyn']
cher_twitter_corpus = twitter_data[twitter_data['artist'] == 'cher']

# Compare unique tokens between Robyn vs Cher
robyn_vs_cher_twitter = concentration_ratios_by_artist(robyn_twitter_corpus,
↳'Robyn', cher_twitter_corpus, 'Cher')
cher_vs_robyn_twitter = concentration_ratios_by_artist(cher_twitter_corpus,
↳'Cher', robyn_twitter_corpus, 'Robyn')

print("\nCher (Top 10 - Twitter Data)")
cher_vs_robyn_twitter

```

Cher (Top 10 - Twitter Data)

```
[17]:
```

	Token	Cher Count	Robyn Count	Concentration	Cher Concentration	Robyn
47	love	213522	11615		1.36%	0.78%
53	im	139051	9049		0.89%	0.60%
60	life	122679	7354		0.78%	0.49%
486	music	86733	14858		0.55%	0.99%
632	de	72970	6382		0.47%	0.43%
65	follow	62166	5570		0.40%	0.37%
332	lover	60191	4760		0.38%	0.32%
78	like	58566	4944		0.37%	0.33%
197	mom	53465	1118		0.34%	0.07%
34	sheher	47181	3479		0.30%	0.23%

```
[18]: print("\nRobyn (Top 10 - Twitter Data)")
      robyn_vs_cher_twitter
```

Robyn (Top 10 - Twitter Data)

```
[18]:
```

	Token	Robyn Count	Cher Count	Concentration	Robyn Concentration	Cher
6	music	14858	86733		0.99%	0.55%
0	love	11615	213522		0.78%	1.36%
98	im	9049	139051		0.60%	0.89%
1416	och	7922	767		0.53%	0.00%
96	life	7354	122679		0.49%	0.78%
288	de	6382	72970		0.43%	0.47%
40	follow	5570	62166		0.37%	0.40%
39	like	4944	58566		0.33%	0.37%
18	en	4833	25970		0.32%	0.17%
198	•	4829	44347		0.32%	0.28%

```
[19]: # Lyrics data analysis
      robyn_lyrics_corpus = lyrics_data[lyrics_data['artist'] == 'robyn']
      cher_lyrics_corpus = lyrics_data[lyrics_data['artist'] == 'cher']

      # Get unique tokens related to Robyn vs Cher
      robyn_vs_cher_lyrics = concentration_ratios_by_artist(robyn_lyrics_corpus,
      ↪ 'Robyn', cher_lyrics_corpus, 'Cher')
      cher_vs_robyn_lyrics = concentration_ratios_by_artist(cher_lyrics_corpus,
      ↪ 'Cher', robyn_lyrics_corpus, 'Robyn')

      print("\nCher (Top 10 - Lyrics Data)")
      cher_vs_robyn_lyrics
```

Cher (Top 10 - Lyrics Data)

```
[19]:
```

	Token	Cher Count	Robyn Count	Concentration Cher	Concentration Robyn
32	come	93	9	1.72%	0.29%
10	dont	90	60	1.67%	1.94%
7	baby	83	30	1.54%	0.97%
33	see	76	13	1.41%	0.42%
73	love	58	99	1.07%	3.20%
31	girl	57	6	1.06%	0.19%
90	know	53	72	0.98%	2.32%
145	im	52	35	0.96%	1.13%
2	want	50	53	0.93%	1.71%
41	like	50	15	0.93%	0.48%

```
[20]: print("\nRobyn (Top 10 - Lyrics Data)")
      robyn_vs_cher_lyrics
```

Robyn (Top 10 - Lyrics Data)

```
[20]:
```

	Token	Robyn Count	Cher Count	Concentration Robyn	Concentration Cher
38	love	99	58	3.20%	1.07%
0	robyn	75	0	2.42%	0.00%
70	show	74	1	2.39%	0.02%
16	know	72	53	2.32%	0.98%
15	dont	60	90	1.94%	1.67%
67	want	53	50	1.71%	0.93%
9	got	50	21	1.61%	0.39%
72	go	41	50	1.32%	0.93%
64	time	38	38	1.23%	0.70%
11	heart	38	19	1.23%	0.35%

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

A: Cher has significantly larger twitter presence especially in words like “love” and “im” which are far more frequently than Robyn. In the Lyrics data, Robyn uses self-referential words like “robyn” and “show” more often, while Cher’s lyrics have words like “baby” and “come.” Both artists share common tokens in lyric like “love” and in twitter like “life”. Overall, Cher’s Twitter data shows a higher concentration of personal and emotional words like “love” and “life” compared to Robyn. The overlapped tokens such as “love”, “music” and “im” seems balanced distributed. In conclusion, the token ratios highlight that while both artists share common words, they each have distinct linguistic patterns, particularly in their lyrics, where Robyn’s use of words like “show” stands out.

## 1.6 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.

```

[21]: 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

    if isinstance(word_freq, pd.DataFrame):
        counter = word_freq['freq'].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)

```



words with fewer than 5 occurrences. The word clouds visually represent the most frequent words in each artist’s lyrics and Twitter posts. The larger the token in the word cloud, the more frequently that word is used. This allows for easy comparison of word variation and usage differences (or similarities) between the artist’s lyrics and their Twitter activity. As we can see that words “love” is found a lot in both of artists lyrics and “life” is in the twitter.

- Dib, F. (n.d.). Build, test, and debug regex. regex101. <https://regex101.com/>
- OpenAI. (2024). ChatGPT (September 24 version) [Large language model]. <https://chat.openai.com/chat>
- Jens, A. (n.d.). Blueprints-text/CH01/first\_insights.ipynb at master · blueprints-for-text-analytics-python/blueprints-text. GitHub. [https://github.com/blueprints-for-text-analytics-python/blueprints-text/blob/master/ch01/First\\_Insights.ipynb](https://github.com/blueprints-for-text-analytics-python/blueprints-text/blob/master/ch01/First_Insights.ipynb)