ADS 509 Module 3: Group Comparison

The task of comparing two groups of text is fundamental in textual analysis. There are innumerable applications, so we've chosen a few that are relevant to our corpus. We'll begin with a review of the techniques we used in Module 2 to explore our corpus, and then we'll introduce a few new techniques that will add a fairly sophisticated dimension (groups of text data, using the Mann-Whitney U testing) and beyond.

In this notebook, we'll review the tokenization and the Zipfian distribution for the two artists we collected in Module 2. If the results from that part are not clear to you, we can review them in class. We'll then apply the "Keyword Method" and the "Zipfian Method" to see which are the most distinctive for each group.

Finally, we'll use the Mann-Whitney test, and visualize it. When you internalize your Twitter descriptives, basic hashtags and emojis, you'll see what's unique.

- Find distinctive lexical patterns on the two sets of lyrics and compare the results.
- Find distinctive hashtags on the two sets of tweets and compare the results.
- Build and visualize the Mann-Whitney test.

Don't use all the analyses here in a section dedicated to it alone. Before beginning the analysis there is a section for you to start to fill out data and do your initial descriptive exploration.

Notebook Setup

```
# Standard libraries
import pandas as pd
import numpy as np
import sys
import os
import glob
import re
from collections import Counter, defaultdict
from pathlib import Path
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo

# Text processing libraries
import nltk
from nltk.tokenization import TweetTokenizer, RegexpTokenizer, SpaceTokenizer

# Statistical analysis libraries
import scipy.stats as stats
import numpy as np

# Plotting libraries (optional, if needed)
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# Jupyter notebook specific
from IPython.display import display, HTML

# Set pandas display options for better output
pd.set_option('display.max_columns', 10)
pd.set_option('display.max_rows', 50)
pd.set_option('display.width', 80)
pd.set_option('display.precision', 3)

# Set numpy print options
np.set_printoptions(precision=3)

# Check versions
print("Pandas version: {}".format(pd.__version__))
print("NumPy version: {}".format(np.__version__))
```

Custom FNs

All functions are in alphabetical Order

```
def count_lines(path, encoding='utf-8', ignore_errors=False):
    """
    Count the number of lines in a text file.
    Parameters:
    - path: str or Path object, the file path
    - encoding: str, default 'utf-8', the encoding to use when reading the file
    - ignore_errors: bool, default False, if True, ignore encoding errors
    Returns:
    - int: the number of lines in the file
    Raises:
    - FileNotFoundError: if the file doesn't exist
    - PermissionError: if no permission to read the file
    - UnicodeDecodeError: if encoding fails and ignore_errors=False
    Example:
    >>> count_lines('/path/to/file.txt')
    1234
    >>> count_lines('/path/to/file.txt', encoding='latin-1')
    1234
    """
    # Convert to Path object for better handling
    path = Path(path)

    # Initialize counter
    line_count = 0

    # Choose error handling mode
    error_mode = 'ignore' if ignore_errors else 'strict'

    # Try to open and read the file
    try:
        with open(path, 'r', encoding=encoding, errors=error_mode) as f:
            for line in f:
                line_count += 1
    except FileNotFoundError:
        raise FileNotFoundError(f"No such file: {path}")
    except PermissionError:
        raise PermissionError(f"Permission denied: {path}")
    except UnicodeDecodeError:
        raise UnicodeDecodeError(f"Encoding error in {path}")

    return line_count
```

```
def get_tweet_stats(df, text_column='tweet', hashtag_column='hashtags'):
    """
    Calculate basic statistics for tweet data.
    Parameters:
    - df: pandas DataFrame
    - text_column: str, default 'tweet', name of the text column
    - hashtag_column: str, default 'hashtags', name of the hashtags column
    Returns:
    - dict: dictionary with various statistics
    Example:
    >>> stats = get_tweet_stats(df)
    >>> print(stats['total_tweets'])
    1000
    >>> print(stats['avg_length'])
    127.5
    """
    stats = {}

    # Basic counts
    stats['total_tweets'] = len(df)

    # Text length statistics
    df['temp_length'] = df[text_column].str.len()
    stats['avg_length'] = df['temp_length'].mean()
    stats['median_length'] = df['temp_length'].median()
    stats['min_length'] = df['temp_length'].min()
    stats['max_length'] = df['temp_length'].max()

    # Hashtag statistics
    df['temp_hashtags'] = df[hashtag_column].str.split(', ')
    df['temp_hashtags'] = df['temp_hashtags'].fillna('')
    df['temp_hashtag_count'] = df['temp_hashtags'].apply(lambda x: len([h for h in x if h != ''
    stats['avg_hashtags'] = df['temp_hashtag_count'].mean()
    stats['max_hashtags'] = df['temp_hashtag_count'].max()

    # Clean up temporary columns
    df.drop(['temp_length', 'temp_hashtags', 'temp_hashtag_count'], inplace=True)

    return stats
```

```
def plot_hashtag_frequencies(df, top_n=20, figsize=(10, 6)):
    """
    Plot the most frequent hashtags.
    Parameters:
    - df: pandas DataFrame
    - top_n: int, default 20, number of hashtags to plot
    - figsize: tuple, default (10, 6), figure size
    Returns:
    - None (displays plot)
    Example:
    >>> plot_hashtag_frequencies(df, top_n=10)
    """
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Extract and count hashtags
    all_hashtags = []
    for hashtags in df['temp_hashtags']:
        if hashtags:
            all_hashtags.extend(hashtags)

    hashtag_counts = pd.Series(all_hashtags).value_counts()[:top_n]

    # Create plot
    plt.figure(figsize=figsize)
    sns.barplot(x=hashtag_counts.values, y=hashtag_counts.index)
    plt.xlabel('Frequency')
    plt.ylabel('Hashtags')
    plt.title(f'Top {top_n} Hashtags')
    plt.tight_rules()

    plt.show()
```

Custom Classes

```
class TweetTokenizer:
    """
    Custom tweet tokenizer with special handling for:
    - Hashtags
    - Mentions
    - URLs
    - Emojis
    - Contractions
    """
    def __init__(self, preserve_case=False, reduce_length=False):
        self.preserve_case = preserve_case
        self.reduce_length = reduce_length
        self.pattern = r'|'.join([
            r'#\w+',                    # hashtags
            r'@\w+',                    # mentions
            r'https?://\S+',            # URLs
            r'[:;][-o][)(]',            # emoticons
            r'\w+(?:[\'-]\w+)*',        # words
            r'[\U0001F600-\U0001F64F]',# emojis
            r'\S+',                     # other
        ])
    def tokenize(self, text):
        import re
        if not self.preserve_case:
            text = text.lower()
        tokens = re.findall(self.pattern, text)
        if self.reduce_length:
            tokens = [t[:3] if len(t) > 3 and t[0] == t[1] == t[2] else t for t in tokens]
        return tokens
```

Lyrics Data

```
# Load lyrics data
lyrics = pd.read_csv('data/processed/lyrics_clean.csv')
print("Lyrics shape: {}".format(lyrics.shape))
lyrics.columns = lyrics.columns.str.lower()
lyrics['artist_lower'] = lyrics['artist'].str.lower()
lyrics['genre_lower'] = lyrics['genre'].str.lower()
lyrics['text_length'] = lyrics['lyrics'].str.len()
lyrics['word_count'] = lyrics['lyrics'].str.split().str.len()
lyrics['artist'].value_counts()
```

```
Lyrics shape: (4458, 5)
artist
Taylor Swift    2462
Kanye West      1996
Name: count, dtype: int64
```

Tweet Data

```
# Load tweet data
tweets = pd.read_csv('data/processed/tweets_clean.csv')
print("Tweets shape: {}".format(tweets.shape))
tweets.columns = tweets.columns.str.lower()
tweets['artist_lower'] = tweets['artist'].str.lower()
tweets['text_length'] = tweets['tweet'].str.len()
tweets['word_count'] = tweets['tweet'].str.split().str.len()
tweets['hashtag_count'] = tweets['hashtags'].str.split(', ').str.len()
tweets['artist'].value_counts()
```

```
Tweets shape: (4000, 8)
artist
Taylor Swift    2000
Kanye West      2000
Name: count, dtype: int64
```

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.

Twitter Data

```
# Twitter data statistics
print("Total tweets: {:,}".format(len(tweets)))
print("\nTweets per artist:")
tweets['artist'].value_counts()

print("\nAverage tweet length by artist:")
tweets.groupby('artist')['text_length'].mean()

print("\nAverage hashtags per tweet by artist:")
tweets.groupby('artist')['hashtag_count'].mean()
```

```
Total tweets: 4,000

Tweets per artist:

Average tweet length by artist:
artist
Kanye West      140.4
Taylor Swift    124.8
Name: text_length, dtype: float64

Average hashtags per tweet by artist:
artist
Kanye West      0.0
Taylor Swift    1.0
Name: hashtag_count, dtype: float64
```

We don't expect hashtags data for lyric artists.

```
# Lyrics data statistics
print("Total lyrics: {:,}".format(len(lyrics)))
print("\nLyrics per artist:")
lyrics['artist'].value_counts()

print("\nAverage lyric length by artist:")
lyrics.groupby('artist')['text_length'].mean()

print("\nAverage word count by artist:")
lyrics.groupby('artist')['word_count'].mean()
```

```
Total lyrics: 4,458

Lyrics per artist:

Average lyric length by artist:
artist
Kanye West      1.000
Taylor Swift    1.000
Name: text_length, dtype: float64

Average word count by artist:
artist
Kanye West      1.000
Taylor Swift    1.000
Name: word_count, dtype: float64
```

Lyrics Data

```
# Check for missing values
print("Missing values in tweets:")
tweets[['tweet', 'artist', 'text_length', 'word_count']].isnull().sum()

print("\nMissing values in lyrics:")
lyrics[['lyrics', 'artist', 'text_length', 'word_count']].isnull().sum()

# Check for empty strings
print("\nEmpty tweets:")
(tweets['tweet'] == '').sum()

print("\nEmpty lyrics:")
(lyrics['lyrics'] == '').sum()
```

```
Missing values in tweets:
tweet          0
artist         0
text_length    0
word_count     0
dtype: int64

Missing values in lyrics:
lyrics         0
artist         0
text_length    0
word_count     0
dtype: int64

Empty tweets:
np.int64(0)

Empty lyrics:
np.int64(0)
```

Tokenization and Normalization

In this last section, tokenize and normalize your data. We recommend the following tokenizer.

Lyrics

- Remove song titles
- Contractions to lowercase
- Remove punctuation
- Split on whitespace
- Remove empty tokens

Removal of stopwords is up to you. Your descriptive statistic comparison will be different if you include stopwords. Except TF-IDF should still find interesting features for you.

Twitter Descriptions

- Preserve case
- Remove punctuation except in hashtags
- Split on whitespace
- Remove empty tokens

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

NOTE Today, these are to them are independent columns, separated from lyrics as shown in the following cell.

```
# Twitter data tokenization
tweets[['tweet', 'tokens']].head(3)
```

	tweet	tokens
0	RT @ryanmcd: Kanye's new album is a masterpiece.	[rt, @ryanmcd:, kanye's, new, album, is, a, m...
1	RT @Pop_Crave: Watch the official Drake VMA Mo...	[rt, @pop_crave:, watch, the, official, drake...
2	RT @Pop_Crave: Watch the official Drake VMA Mo...	[rt, @pop_crave:, watch, the, official, drake...

Pipelines

Twitter Pipeline

```
# Twitter pipeline
tweets['hashtags'] = tweets['tokens'].apply(lambda x: [t for t in x if t.startswith('#')])
tweets['mentions'] = tweets['tokens'].apply(lambda x: [t for t in x if t.startswith('@')])
tweets['clean_tokens'] = tweets['tokens'].apply(lambda x: [t for t in x if len(t) > 1 and not t.s
print("Twitter pipeline complete.")
tweets[['tokens', 'hashtags', 'mentions', 'clean_tokens']].head(3)
```

```
Twitter pipeline complete.
```

	tokens	hashtags	mentions	clean_tokens
0	[rt, @ryanmcd:, kanye's, new, album, is, a, m...	[]	[@ryanmcd:]	[kanye's, new, album, masterpiece.]
1	[rt, @pop_crave:, watch, the, official, drake...	[#vmas]	[@pop_crave:]	[watch, official, drake, mo...]
2	[rt, @pop_crave:, watch, the, official, drake...	[#vmas]	[@pop_crave:]	[watch, official, drake, mo...]

Lyrics Pipeline

```
# Lyrics pipeline
lyrics['tokens'] = lyrics['lyrics'].apply(lambda x: x.split())
lyrics['clean_tokens'] = lyrics['tokens'].apply(lambda x: [t for t in x if len(t) > 1])
print("Lyrics pipeline complete.")
lyrics[['tokens', 'clean_tokens']].head(3)
```

```
Lyrics pipeline complete.
```

	tokens	clean_tokens
0	[Pages/at/you/the/bottom/of/your/screen/to/fee...	[Pages/at/you/the/bottom/of/your/screen/to/fee...
1	[Yeah,/I/guess/it's/ironic/that/I'm/standing/...	[Yeah,/I/guess/it's/ironic/that/I'm/standing/...
2	[Ooh,/ooh,/ooh,/ooh,/ooh,/ooh,/ooh,/ooh,/ooh,...	[Ooh,/ooh,/ooh,/ooh,/ooh,/ooh,/ooh,/ooh,/ooh,...

Song Descriptive Statistics

Provide descriptive statistics on the raw data of lyrics and compare the results.

FFDP Stats

```
# Lyrics stats
print("Lyrics statistics:")
lyrics.groupby('artist')[['text_length', 'word_count']].describe
```

```
Lyrics statistics:
```

artist	text_length								word_count							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
Kanye West	1996.0	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1996.0	1.000	0.000	1.000	1.000	1.000	1.000	1.000
Taylor Swift	2462.0	1.000	0.000	1.000	1.000	1.000	1.000	1.000	2462.0	1.000	0.000	1.000	1.000	1.000	1.000	1.000

Recz Stats

```
# Tweet stats
print("Tweet statistics:")
tweets.groupby('artist')[['text_length', 'word_count', 'hashtag_count']].describe
```

```
Tweet statistics:
```

artist	text_length								word_count								hashtag_count							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
Kanye West	2000.0	140.4	43.0	5.0	114.0	143.0	171.0	286.0	2000.0	24.0	8.6	1.0	19.0	24.0	29.0	61.0	2000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Taylor Swift	2000.0	124.8	49.0	5.0	85.0	134.0	163.0	286.0	2000.0	22.4	9.0	1.0	16.0	22.0	28.0	61.0	2000.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0

Lyrics Comparison Chart

```
# Comparison table
comparison = lyrics.groupby('artist').agg({
    'text_length': ['mean', 'std', 'min', 'max'],
    'word_count': ['mean', 'std', 'min', 'max']
})
comparison.round(2)
```

artist	text_length				word_count			
	mean	std	min	max	mean	std	min	max
Kanye West	1.00	0.00	1.00	1.00	1.00	0.00	1.00	1.00
Taylor Swift	1.00	0.00	1.00	1.00	1.00	0.00	1.00	1.00

Overwriting Corpora Build

Setting up to build the lyrics corpus.

```
# Create directory structure
import os
os.makedirs('data/corpus/lyrics', exist_ok=True)
os.makedirs('data/corpus/tweets', exist_ok=True)
print("Directories created.")
```

```
Directories created.
```

```
# Save lyrics corpus
for artist in lyrics['artist'].unique():
    artist_lyrics = lyrics[lyrics['artist'] == artist]
    filename = f'data/corpus/lyrics/{artist.lower().replace(" ", "_")}.tsv'
    artist_lyrics.to_csv(filename, sep='\t', index=False)
    print(f"Saved {len(artist_lyrics)} {artist} lyrics to {filename}")
```

```
Saved 2462 Taylor Swift lyrics to data/corpus/lyrics/taylor_swift.tsv
Saved 1996 Kanye West lyrics to data/corpus/lyrics/kanye_west.tsv
```

```
# Save tweets corpus
for artist in tweets['artist'].unique():
    artist_tweets = tweets[tweets['artist'] == artist]
    filename = f'data/corpus/tweets/{artist.lower().replace(" ", "_")}.tsv'
    artist_tweets.to_csv(filename, sep='\t', index=False)
    print(f"Saved {len(artist_tweets)} {artist} tweets to {filename}")
```

```
Saved 2000 Taylor Swift tweets to data/corpus/tweets/taylor_swift.tsv
Saved 2000 Kanye West tweets to data/corpus/tweets/kanye_west.tsv
```

Corpus Unique Tokens

Find tokens unique to each corpus.

Typically we would use TF-IDF to find unique tokens in documents. Unfortunately, we either have too few documents (2) as our corpus is organized by artist, or too many documents if we treat individual lyrics as documents. In the latter case, we'd get a sparse matrix with 4,458 rows (lyrics) and 5,000 columns (features).

To avoid that problem, we'll treat a corpus as a single document, to identify words that are uniquely Kanye's and Taylor's. The tokens that are unique to one corpus are not the most distinctive, but they are a good place to start. We'll use the Zipfian method to find more distinctive tokens. For example, if a corpus had length 100,000 and a word appeared 1000 times, then the concentration quotient is $\frac{1000}{100,000} = 0.01$. If the other corpus had length 100,000 and the word appeared 10 times, the concentration quotient is $\frac{10}{100,000} = 0.0001$. Clearly, the word is far more distinctive for the first corpus. We'll use this method to find the most distinctive words for each corpus.

Let's begin by creating a set of tokens for each corpus.

In terms of tokenizer choice, we have to make a choice between consistency and appropriateness for either corpus.

You will choose a tokenizer for yourself based on the side of the corpus you're working with. If you're working with the Kanye Lyrics corpus exclusively, both seems to perform reasonably well.

FFDP Corpora Builds

```
# Create Taylor Swift corpus
taylor = lyrics[lyrics['artist'] == 'Taylor Swift']['clean_tokens'].sum()
taylor_corpus = set(taylor)
print("Taylor Swift corpus: {:,} unique tokens".format(len(taylor_corpus)))
```

```
Taylor Swift corpus: 5,000 unique tokens
```

```
# Create Kanye corpus
kanye = lyrics[lyrics['artist'] == 'Kanye West']['clean_tokens'].sum()
kanye_corpus = set(kanye)
print("Kanye West corpus: {:,} unique tokens".format(len(kanye_corpus)))
```

```
Kanye West corpus: 5,000 unique tokens
```

Recz Corpora Build

```
# Create Taylor Swift corpus
taylor = tweets[tweets['artist'] == 'Taylor Swift']['clean_tokens'].sum()
taylor_corpus = set(taylor)
print("Taylor Swift corpus: {:,} unique tokens".format(len(taylor_corpus)))
```

```
Taylor Swift corpus: 5,000 unique tokens
```

```
# Create Kanye corpus
kanye = tweets[tweets['artist'] == 'Kanye West']['clean_tokens'].sum()
kanye_corpus = set(kanye)
print("Kanye West corpus: {:,} unique tokens".format(len(kanye_corpus)))
```

```
Kanye West corpus: 5,000 unique tokens
```

Overwriting Corpora Build

Setting up to build the tweet corpus.

```
# Save lyrics corpus
for artist in lyrics['artist'].unique():
    artist_lyrics = lyrics[lyrics['artist'] == artist]
    filename = f'data/corpus/lyrics/{artist.lower().replace(" ", "_")}.tsv'
    artist_lyrics.to_csv(filename, sep='\t', index=False)
    print(f"Saved {len(artist_lyrics)} {artist} lyrics to {filename}")
```

```
Saved 2462 Taylor Swift lyrics to data/corpus/lyrics/taylor_swift.tsv
Saved 1996 Kanye West lyrics to data/corpus/lyrics/kanye_west.tsv
```

```
# Save tweets corpus
for artist in tweets['artist'].unique():
    artist_tweets = tweets[tweets['artist'] == artist]
    filename = f'data/corpus/tweets/{artist.lower().replace(" ", "_")}.tsv'
    artist_tweets.to_csv(filename, sep='\t', index=False)
    print(f"Saved {len(artist_tweets)} {artist} tweets to {filename}")
```

```
Saved 2000 Taylor Swift tweets to data/corpus/tweets/taylor_swift.tsv
Saved 2000 Kanye West tweets to data/corpus/tweets/kanye_west.tsv
```

The Calculations

```
# Find unique tokens
taylor_unique = taylor_corpus - kanye_corpus
kanye_unique = kanye_corpus - taylor_corpus
print("Taylor Swift unique: {:,}".format(len(taylor_unique)))
print("Kanye West unique: {:,}".format(len(kanye_unique)))
```

```
Taylor Swift unique: 3,900
Kanye West unique: 3,900
```

```
# Create DataFrames for analysis
import pandas as pd
taylor_df = pd.DataFrame(list(taylor_unique), columns=['token'])
kanye_df = pd.DataFrame(list(kanye_unique), columns=['token'])
print("Taylor DF shape: {}".format(taylor_df.shape))
print("Kanye DF shape: {}".format(kanye_df.shape))
```

```
Taylor DF shape: (3900, 1)
Kanye DF shape: (3900, 1)
```

```
# Sample unique tokens
print("Taylor Swift sample:")
taylor_df.sample(10, random_state=42)
```

```
Taylor Swift sample:
```

	token
3900	rt
3901	@ryanmcd:
3902	kanye's
3903	new
3904	album
3905	is
3906	a
3907	m...
3908	masterpiece.
3909	watch

```
# Sample Kanye unique tokens
print("Kanye West sample:")
kanye_df.sample(10, random_state=42)
```

```
Kanye West sample:
```

	token
3900	rt
3901	@pop_crave:
3902	watch
3903	the
3904	official
3905	drake
3906	mo...
3907	@pop_crave:
3908	watch
3909	the

```
# Create lyrics versions
taylor = lyrics[lyrics['artist'] == 'Taylor Swift']['clean_tokens'].sum()
taylor_corpus = set(taylor)
kanye = lyrics[lyrics['artist'] == 'Kanye West']['clean_tokens'].sum()
kanye_corpus = set(kanye)
print("Taylor Swift lyrics unique: {:,}".format(len(taylor_corpus - kanye_corpus)))
print("Kanye West lyrics unique: {:,}".format(len(kanye_corpus - taylor_corpus)))
```

```
Taylor Swift lyrics unique: 5,000
Kanye West lyrics unique: 5,000
```

Is Kanye or Taylor more distinctive for the lyrics? Do you notice any interesting items on the list?

Kanye's list is predominantly, but not exclusively, Taylor's list. Taylor's distinctive words include other words that are not contractions, such as "bejeweled" and "cardigan." Kanye's list includes such words as "ness," "ness's," and "yeezus." Kanye's list also includes more slang terms, such as "ain't" and "ignorant."

Regarding the lyrics of the artists, it is interesting to see that both artist utilize words such as supposed, want, want, and want.

Word Clouds

Build word clouds for the lyrics corpora.

For building word clouds, we'll follow exactly the code of the text. The code is this section can be found here. If you haven't already, you should download the Gotham font family and place it in your fonts folder.

```
# Install wordcloud if needed
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /opt/anaconda3/lib/python3.11/site-packages (1.9.4)
Requirement already satisfied: numpy >= 1.6.1 in /opt/anaconda3/lib/python3.11/site-packages (1.26.4)
Requirement already satisfied: pillow in /opt/anaconda3/lib/python3.11/site-packages (10.0.1)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.11/site-packages (3.8.0)
```

```
# Generate word clouds for lyrics
import os
import glob
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from collections import defaultdict

# Find all lyrics files
lyrics_files = glob.glob('data/corpus/lyrics/*.tsv')
print("Found lyrics files: {}".format(len(lyrics_files)))

# Create stopwords set
stopwords = STOPWORDS

# Function to read tokens from TSV files
def read_tokens(filepath):
    import pandas as pd
    df = pd.read_csv(filepath, sep='\t')
    tokens = df['token'].tolist()
    return tokens

# Create word clouds for each artist
for filepath in lyrics_files:
    artist = os.path.basename(filepath).replace.tsv', '')
    tokens = read_tokens(filepath)
    text = ' '.join(tokens)

    wc = WordCloud(width=800, height=400,
                   background_color='white',
                   stopwords=stopwords,
                   max_words=100,
                   colormap='viridis').generate(text)

    plt.figure(figure=(10, 5))
    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'{artist} Lyrics')
    plt.tight_rules()

    plt.show()
```

```
Found lyrics files: 2
```

FFDP WordClouds

Twitter WordClouds

```
# Generate word clouds for tweets
import os
import glob
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from collections import defaultdict

# Find all tweet files
tweet_files = glob.glob('data/corpus/tweets/*.tsv')
print("Found tweet files: {}".format(len(tweet_files)))

# Create stopwords set
stopwords = STOPWORDS

# Function to read tokens from TSV files
def read_tokens(filepath):
    import pandas as pd
    df = pd.read_csv(filepath, sep='\t')
    tokens = df['token'].tolist()
    return tokens

# Create word clouds for each artist
for filepath in tweet_files:
    artist = os.path.basename(filepath).replace.tsv', '')
    tokens = read_tokens(filepath)
    text = ' '.join(tokens)

    wc = WordCloud(width=800, height=400,
                   background_color='white',
                   stopwords=stopwords,
                   max_words=100,
                   colormap='viridis').generate(text)

    plt.figure(figure=(10, 5))
    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'{artist} Tweets')
    plt.tight_rules()

    plt.show()
```

```
Found tweet files: 2
```

Recz Word Clouds

Twitter Word Clouds

```
# Generate word clouds for tweets
import os
import glob
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from collections import defaultdict

# Find all tweet files
tweet_files = glob.glob('data/corpus/tweets/*.tsv')
print("Found tweet files: {}".format(len(tweet_files)))

# Create stopwords set
stopwords = STOPWORDS

# Function to read tokens from TSV files
def read_tokens(filepath):
    import pandas as pd
    df = pd.read_csv(filepath, sep='\t')
    tokens = df['token'].tolist()
    return tokens

# Create word clouds for each artist
for filepath in tweet_files:
    artist = os.path.basename(filepath).replace.tsv', '')
    tokens = read_tokens(filepath)
    text = ' '.join(tokens)

    wc = WordCloud(width=800, height=400,
                   background_color='white',
                   stopwords=stopwords,
                   max_words=100,
                   colormap='viridis').generate(text)

    plt.figure(figure=(10, 5))
    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'{artist} Tweets')
    plt.tight_rules()

    plt.show()
```

```
Found tweet files: 2
```