In [1]:
```
pip install emoji==1.7
```

```
Requirement already satisfied: emoji==1.7 in c:\users\abana\anaconda3\lib\site-packages
(1.7.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import os
import re
import emoji
import pandas as pd
import nltk
import numpy as np

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

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

In [3]:
```
# Place any addtional functions or constants you need here.

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

# Stopwords
#from textbook
sw = set(nltk.corpus.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.UNICODE_EMOJI :
    for em in emoji.UNICODE_EMOJI[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
    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    num_characters = sum(len(token) for token in tokens)
    lexical_diversity = num_unique_tokens / num_tokens
```

```
    return {
        'num_tokens': num_tokens,
        'num_unique_tokens': num_unique_tokens,
        'num_characters': num_characters,
        'lexical_diversity': lexical_diversity
    }



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
    #from textbook
    return [t for t in tokens if t.lower() not in sw]

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
    text = text.split()
    return(text)

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

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

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 [4]:
```
# change `data_location` to the location of the folder on your machine.
data_location = "./"
# These subfolders should still work if you correctly stored the
# data from the Module 1 assignment
twitter_folder = "/Users/Abana/Downloads/twitter"
lyrics_folder = "/Users/Abana/Downloads/lyrics"
```

In [5]:
```
df_lyrics = {
    'artist': [],
    'song_name': [],
    'contents': []
```

```
}

for artist_folder in os.listdir(lyrics_folder):
    for song_file in os.listdir(os.path.join(lyrics_folder, artist_folder)):
        with open(os.path.join(lyrics_folder, artist_folder, song_file), 'r') as f:
            song_lyrics = f.read()
        df_lyrics['artist'].append(artist_folder)
        df_lyrics['song_name'].append(song_file.replace('www_azlyrics_comcher_', '').re
        df_lyrics['contents'].append(song_lyrics)
```

In [6]:

```
df_lyrics = pd.DataFrame(df_lyrics)
df_lyrics.head()
```

Out[6]:

	artist	song_name	contents
0	cher	cher_88degrees	"88 Degrees"\n\n\n\nStuck in L.A., ain't got n...
1	cher	cher_adifferentkindoflovesong	"A Different Kind Of Love Song"\n\n\n\nWhat if...
2	cher	cher_afterall	"After All"\n\n\n\nWell, here we are again\nI ...
3	cher	cher_again	"Again"\n\n\n\nAgain evening finds me at your ...
4	cher	cher_alfie	"Alfie"\n\n\n\nWhat's it all about, Alfie?\nIs...

In [7]:

```
#Reading in data
df_twitter = {
    'artist': [],
    'description': []
}
for filename in os.listdir(twitter_folder):
    if 'data' in filename:
        artist = filename.split('_')[0]
        with open(os.path.join(twitter_folder, filename), 'r', encoding='utf-8') as f:
            for line in f:
                fields = [t.strip() for t in line.split('\t') if t.strip()]
                description = fields[-1]
                if description=='description':
                    continue
                df_twitter['artist'].append(artist)
                df_twitter['description'].append(description)
```

In [8]:

```
df_twitter = pd.DataFrame(df_twitter)
df_twitter.head()
```

Out[8]:

	artist	description
0	cher	1014
1	cher	Proud supporter of messy buns & leggings
2	cher	163㎝ / 愛かっぷ💜26歳🍒 工〇好きな女の子💞 フォローしてくれたらDMします🧡
3	cher	csu
4	cher	Writer @Washinformer @SpelmanCollege alumna #D...

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 [9]:
```
# apply the `pipeline` techniques from BTAP Ch 1 or 5

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

df_lyrics["tokens"] = df_lyrics['contents'].apply(prepare, pipeline=my_pipeline)
df_lyrics["num_tokens"] = df_lyrics["tokens"].map(len)

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

In [10]:
```
df_twitter['has_emoji'] = df_twitter["description"].apply(contains_emoji)
```

In [11]:
```
df_twitter[df_twitter.has_emoji].sample(1)[["artist","description","tokens"]]
```

Out[11]:

	artist	description	tokens
1181215	cher	Probably drunk and making people uncomfortable...	[probably, drunk, making, people, uncomfortabl...

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: We can use WordPiece splitting a text into words or subwords.

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

In [12]:
```
df_lyrics.head()
```

Out[12]:

	artist	song_name	contents	tokens	num_tokens
0	cher	cher_88degrees	"88 Degrees"\n\n\n\nStuck in L.A., ain't got n...	[88, degrees, stuck, la, aint, got, friends, h...	182
1	cher	cher_adifferentkindoflovesong	"A Different Kind Of Love Song"\n\n\n\nWhat if...	[different, kind, love, song, world, crazy, sa...	137
2	cher	cher_afterall	"After All"\n\n\n\nWell, here we are again\nI ...	[well, guess, must, fate, weve, tried, deep, i...	120
3	cher	cher_again	"Again"\n\n\n\nAgain evening finds me at your ...	[evening, finds, door, ask, could, try, dont, ...	34
4	cher	cher_alfie	"Alfie"\n\n\n\nWhat's it all about, Alfie?\nIs...	[alfie, whats, alfie, moment, live, whats, sor...	67

In [13]:

```
for artist in df_lyrics['artist'].unique():
    print(artist)
    print(descriptive_stats(np.hstack(df_lyrics[df_lyrics['artist']==artist]['tokens'].
```

```
cher
{'num_tokens': 35916, 'num_unique_tokens': 3703, 'num_characters': 172696, 'lexical_dive
rsity': 0.10310168170174852}
robyn
{'num_tokens': 15227, 'num_unique_tokens': 2156, 'num_characters': 73988, 'lexical_diver
sity': 0.14159059565245943}
```

Q: what observations do you make about these data?

A: Robyn has lexical_diversity higher than cher but Cher has more tokens than Robyn.

Find tokens uniquely related to a corpus

In [14]:

```
df_twitter.head()
```

Out[14]:

	artist	description	tokens	num_tokens	has_emoji
0	cher	1014	[1014]	1	False
1	cher	Proud supporter of messy buns & leggings	[Proud, supporter, of, messy, buns, leggings]	6	False
2	cher	163㎝ / 愛かっぷ💜26歳🍒 工〇好きな女の子💓 フォローしてくれたらDMします❤️	[163㎝ / 愛かっぷ💜26歳🍒, 工〇好きな女の子💓, フォローしてくれたらdmします❤️]	3	True
3	cher	csu	[csu]	1	False
4	cher	Writer @Washinformer @SpelmanCollege alumna #D...	[writer, washinformer, spelmancollege, alumna,...	17	False

In [15]:
```
def get_pattern(text, num_words):
    total_tokens = 1
    unique_tokens = 0
    avg_token_len = 0.0
    lex_diversity = 0.0
    top_words = []

    total_tokens = len(text)
    unique_tokens = len(set(text))
    avg_token_len = np.mean([len(w) for w in text])
    lex_diversity = unique_tokens/total_tokens

    top_words = FreqDist(text).most_common(num_words)

    results = {
        'total_tokens': total_tokens,
        'unique_tokens': unique_tokens,
        'avg_token_len': avg_token_len,
        'lex_diversity': lex_diversity,
        'top_words': top_words
    }

    return results
```

In [16]:
```
for artist in df_twitter['artist'].unique():
    print(artist)
    print(get_pattern(np.hstack(df_twitter[df_twitter['artist']==artist]['tokens'].to_l
```

```
cher
{'total_tokens': 17667726, 'unique_tokens': 1557989, 'avg_token_len': 5.582487751960835,
'lex_diversity': 0.08818276896528733, 'top_words': [('love', 213522), ('im', 139051),
('life', 122680), ('music', 86733), ('de', 72970), ('follow', 62166), ('lover', 60191),
('like', 58566), ('mom', 53465), ('sheher', 47181)]}
robynkonichiwa
{'total_tokens': 1664119, 'unique_tokens': 261185, 'avg_token_len': 5.751987688380458,
'lex_diversity': 0.15695091516892723, 'top_words': [('music', 14858), ('love', 11615),
('im', 9049), ('och', 7922), ('life', 7354), ('de', 6382), ('follow', 5570), ('like', 49
44), ('en', 4833), ('•', 4829)]}
```

In [17]:
```
def get_word_frac(word, fd_corpus, length):
    if word in fd_corpus:
        return (fd_corpus[word]/length)
    else:
        return 0

def get_ratio(word, fd_corpus_1, fd_corpus_2, len_1, len_2):
    frac_1 = get_word_frac(word, fd_corpus_1, len_1)
    frac_2 = get_word_frac(word, fd_corpus_2, len_2)

    if frac_2>0:
        return frac_1/frac_2
    else:
        return float("NaN")
```

In [22]:
```
def compare_texts(df, num_words=10, ratio_cutoff=5):
    results = {}
```

```
    artist_1 = df['artist'].unique()[0]
    artist_2 = df['artist'].unique()[1]

    results[artist_1] = get_pattern(np.hstack(df[df['artist']==artist_1]['tokens'].to_l
    results[artist_2] = get_pattern(np.hstack(df[df['artist']==artist_2]['tokens'].to_l

    fd_1 = FreqDist(np.hstack(df[df['artist']==artist_1]['tokens'].to_list()))
    fd_2 = FreqDist(np.hstack(df[df['artist']==artist_2]['tokens'].to_list()))

    fd_1_words = set(fd_1.keys())
    fd_2_words = set(fd_2.keys())

    holder = dict()

    results['{}_vs_{}'.format(artist_1, artist_2)] = dict()
    results['{}_vs_{}'.format(artist_2, artist_1)] = dict()

    for word, count in fd_1.items():
        if count > ratio_cutoff:
            if word in fd_2_words and fd_2[word] > ratio_cutoff:
                holder[word] = get_ratio(word, fd_1, fd_2, results[artist_1]['total_tok

    num_added = 0

    for word, frac in sorted(holder.items(), key=lambda item: -1*item[1]):
        results['{}_vs_{}'.format(artist_1, artist_2)][word] = frac
        num_added+=1
        if num_added == num_words:
            break

    holder = dict()
    for word, count in fd_2.items():
        if count > ratio_cutoff:
            if word in fd_1_words and fd_1[word] > ratio_cutoff:
                holder[word] = get_ratio(word, fd_2, fd_1, results[artist_2]['total_tok

    num_added = 0

    for word, frac in sorted(holder.items(), key=lambda item: -1*item[1]):
        results['{}_vs_{}'.format(artist_2, artist_1)][word] = frac
        num_added+=1
        if num_added == num_words:
            break

    return results
```

In [19]:

```
compare_texts(df_twitter)
```

Out[19]:

```
{'cher': {'total_tokens': 17667726,
  'unique_tokens': 1557989,
  'avg_token_len': 5.582487751960835,
  'lex_diversity': 0.08818276896528733,
  'top_words': [('love', 213522),
   ('im', 139051),
   ('life', 122680),
   ('music', 86733),
   ('de', 72970),
   ('follow', 62166),
```

```
  ('lover', 60191),
  ('like', 58566),
  ('mom', 53465),
  ('sheher', 47181)]},
 'robynkonichiwa': {'total_tokens': 1664119,
  'unique_tokens': 261185,
  'avg_token_len': 5.751987688380458,
  'lex_diversity': 0.15695091516892723,
  'top_words': [('music', 14858),
  ('love', 11615),
  ('im', 9049),
  ('och', 7922),
  ('life', 7354),
  ('de', 6382),
  ('follow', 5570),
  ('like', 4944),
  ('en', 4833),
  ('•', 4829)]},
 'cher_vs_robynkonichiwa': {'grandmother': 35.35586421669004,
  '#fbr': 24.575680108275773,
  'resister': 24.448972445172465,
  'nana': 23.665179024736968,
  'rbsoul': 20.58719198740444,
  'grandma': 19.901948971604853,
  '#theresistance': 18.902708917520002,
  'hiphoprap': 17.96126408031319,
  'gop': 17.330917176324785,
  'grandchildren': 16.707815639048892},
 'robynkonichiwa_vs_cher': {'sveriges': 208.0905449670366,
  'träning': 203.489903666745,
  'brinner': 198.68703637523177,
  'följ': 195.5272552623941,
  'gärna': 194.96423971865212,
  'arbetar': 187.24288369019064,
  'varje': 184.02565201166504,
  'familj': 180.48669716528684,
  'projektledare': 174.41991742863857,
  'detta': 165.31974782366612}}
```

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

A: Both cher and robynkonichiwa have music and love in their tokens. But cher has mom and family in her tokens.

Build word clouds for all four corpora.

In [20]:

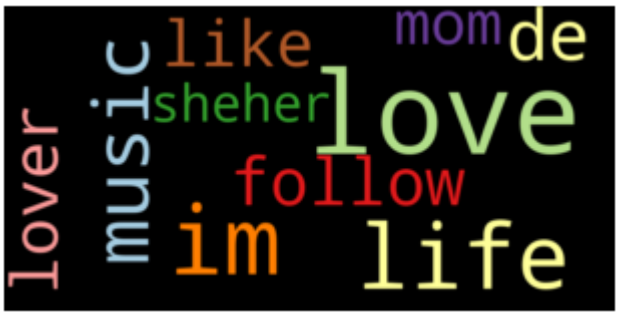
```
from matplotlib import pyplot as plt

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
```

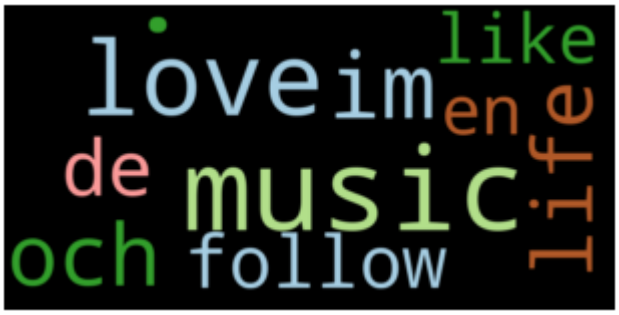
```
    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                              if token not in stopwords}
    wc.generate_from_frequencies(counter)

    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)
```

In [21]:

```
for idx, artist in enumerate(df_twitter['artist'].unique()):
    print(artist)
    freq = count_words(df_twitter[df_twitter['artist']==artist])
    plt.figure(figsize=(12,4))
    plt.subplot(1,2, idx+1)
    wordcloud(freq['freq'], max_words=10)
```

```
cher
robynkonichiwa
```





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

A: Both have love and music in wordclouds as main topic.