Assignment 2.1: Tokenization, Normalization, and Descriptive Statistics

Course: ADS 509, Applied Large Language Models for Data Science

Name: Anna Shekikyan

Date: 09/15/2025

GitHub: https://github.com/AnahitShekikyan/ADS509-Assignment2.1-Lyrics-Description-EDA

ipynb: https://colab.research.google.com/drive/18TJd7apglDbLVLzP4xsM9poYBFqsliM6?usp=sharing

```
import os
import re
! pip install emoji
import emoji
import pandas as pd
import numpy as np

from collections import Counter, defaultdict
! pip install nltk
from nltk.corpus import stopwords, wordnet
from string import punctuation
```

```
Requirement already satisfied: emoji in /usr/local/lib/python3.12/dist-packages (2.14.1)
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
```

```
# Additional imports statements
import json
import unicodedata
import matplotlib.pyplot as plt

from nltk.stem import WordNetLemmatizer
from pathlib import Path
```

```
twitter_folder = "twitter/"
lyrics_folder = "lyrics/"
```

```
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_diversity),
        and num_tokens most common tokens. Return a list with the number of tokens, number
        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) if num_tokens > 0 else 0.0
    num_characters = sum(len(t) for t in 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

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

```
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.
```

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

`A:` Assertions catch violations of assumptions early, right where they occur. They serve as lightweight, executable checks for invariants and expected outputs (like your unit-test style asserts), making bugs easier to locate and preventing silent failures that would otherwise propagate.

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.

```
# Data Input, Setup
from google.colab import drive
from pathlib import Path

drive.mount('/content/drive')

# The folder that directly contains both 'lyrics' and 'twitter'
data_base = Path("/content/drive/MyDrive/M1 Assignment Data/M1 Assignment Data/M1 Results").resolve()

print("Using data base:", data_base)
print("lyrics exists:", (data_base / "lyrics").is_dir(), "| twitter exists:", (data_base / "twitter").is_dir())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Using data base: /content/drive/MyDrive/M1 Assignment Data/M1 Assignment Data/M1 Results
lyrics exists: True | twitter exists: True
```

```
# Read in the lyrics data
lyrics_dir = (data_base / "lyrics").resolve()
lyrics_data = defaultdict(dict)   # {artist: {title: full_text}}

def first_nonempty_line(text: str, fallback: str) -> str:
    for ln in text.splitlines():
        s = ln.strip()
        if s:
            return s
    return fallback

if lyrics_dir.is_dir():
    for f in lyrics_dir.rglob("*"):
        if f.is_file() and f.suffix.lower() in {".txt", ".md", ".lyr"}:
            try:
                txt = f.read_text(encoding="utf-8", errors="ignore").strip()
            except Exception:
                txt = f.read_text(encoding="latin1", errors="ignore").strip()

            artist = f.parent.name if f.parent != lyrics_dir else (
                f.stem.split("_", 1)[0].lower() if "_" in f.stem else "unknown"
            )
            title_fallback = f.stem.split("_", 1)[1].replace("_", " ").strip() if "_" in f.stem else f.stem
            title = first_nonempty_line(txt, title_fallback)

            lyrics_data[artist][title] = txt

print(f"Loaded lyrics — artists: {len(lyrics_data)} | songs: {sum(len(v) for v in lyrics_data.values())}")
```

```
Loaded lyrics — artists: 2 | songs: 406
```

```
# peek a few
shown = 0
for a, songs in lyrics_data.items():
    for t in songs:
        print(f"- {a}: {t[:60]}")
        shown += 1
        if shown >= 3: break
    if shown >= 3: break
```

```
- robyn: "Love Kills"
- robyn: "Include Me Out"
- robyn: "Electric"
```

```
# Read in the twitter data
twitter_dir = (data_base / "twitter").resolve()
twitter_desc = defaultdict(list)  # {artist: [description, ...]}

def norm(s: str) -> str:
    return re.sub(r"\s+", " ", str(s)).strip()

def candidate_files(root: Path):
    for f in root.rglob("*"):
        if f.is_file() and ("followers" in f.name.lower()) and f.suffix.lower() in {".csv", ".tsv", ".txt"}:
            yield f

def read_followers(path: Path):
    # TSV with header
    try:
        df = pd.read_csv(path, sep="\t", engine="python", encoding="utf-8", on_bad_lines="skip", dtype=str)
    except UnicodeDecodeError:
        df = pd.read_csv(path, sep="\t", engine="python", encoding="latin1", on_bad_lines="skip", dtype=str)
    except Exception:
        df = None

    if df is not None:
        cols = [c for c in df.columns if c.lower() == "description"]
        if cols:
            vals = df[cols[0]].dropna().astype(str).map(norm)
            return [v for v in vals if v]

    # TSV without header → last column
    try:
        df_noh = pd.read_csv(path, sep="\t", engine="python", header=None, encoding="utf-8",
                             on_bad_lines="skip", dtype=str)
    except UnicodeDecodeError:
        df_noh = pd.read_csv(path, sep="\t", engine="python", header=None, encoding="latin1",
                             on_bad_lines="skip", dtype=str)
    except Exception:
        df_noh = None

    if df_noh is not None and df_noh.shape[1] >= 2:
        last = df_noh.columns[-1]
        vals = df_noh[last].dropna().astype(str).map(norm)
        vals = [v for v in vals if v and v.lower() != "description"]
        if vals:
            return vals

    # CSV/unknown delimiter with header (pandas sniff)
    try:
        df_any = pd.read_csv(path, sep=None, engine="python", encoding="utf-8",
                             on_bad_lines="skip", dtype=str)
    except UnicodeDecodeError:
        df_any = pd.read_csv(path, sep=None, engine="python", encoding="latin1",
                             on_bad_lines="skip", dtype=str)
    except Exception:
        df_any = None

    if df_any is not None:
        cols = [c for c in df_any.columns if c.lower() == "description"]
        if cols:
            vals = df_any[cols[0]].dropna().astype(str).map(norm)
            return [v for v in vals if v]

    return []

if twitter_dir.is_dir():
    for f in candidate_files(twitter_dir):
        artist = f.parent.name if f.parent != twitter_dir else (
            f.stem.split("_", 1)[0].lower() if "_" in f.stem else "unknown"
        )
        descs = read_followers(f)
        if descs:
            twitter_desc[artist].extend(descs)

    # deduplicate per artist
    for a, ds in list(twitter_desc.items()):
        seen, uniq = set(), []
        for s in ds:
            if s not in seen:
                seen.add(s); uniq.append(s)
        twitter_desc[a] = uniq

print(f"Collected twitter descriptions — artists: {len(twitter_desc)} | total unique: {sum(len(v) for v in twitter_desc.values())}
```

```
Collected twitter descriptions — artists: 2 | total unique: 2032878
```

```
# Peek into a few Twitter bios per artist
for artist, descs in twitter_desc.items():
    print("="*60)
    print(f"Artist: {artist} | total bios: {len(descs)}")
    for d in descs[:5]:   # show first 5 bios
        print(" -", d[:120])  # truncate to 120 chars for readability
    print()
```

```
============================================================
Artist: cher | total bios: 1850890
 - Proud supporter of messy buns & leggings
 - 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🧡
 - csu
 - Writer @Washinformer @SpelmanCollege alumna #DCnative Award-winning journalist & PR pro @IABC Fellow & Past Chair IG: bc
 - I’m unemployed and live with my parents. MOOPS!

============================================================
Artist: robynkonichiwa | total bios: 181988
 - books, movies, music, nature & TV shows. OG Sweetee since '12 thanks to YouTube recommending 'This Feeling' on my homepa
 - (Am)auteur en herbe 🌱 - juriste en paille 🤡 - Ami des fleurs 🌸🌈 (sans la main verte) - music & books - #morecomingsoon..
 - This Twitter profile is full of sarcasm and rants with the occasional moan, dont like me dont follow me! KLF Stan Accoun
 - Flora Youssef - Blogger & Founder Posting review articles about the latest music 🎵 https://t.co/dx4hoIom7T https://t.co/
 - Recording Artist / songwriter 🌈 Fashion Stylist of The Floral House Boutique / CEO Of Arcade Mode Records / Actor
```

```
punctuation = set(punctuation) # speeds up comparison
```

```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
# clean twitter data
URL_RE   = re.compile(r'https?://\S+|www\.\S+', re.I)
EMAIL_RE = re.compile(r'\b[\w\.-]+@[\w\.-]+\.\w+\b', re.I)
NUM_RE   = re.compile(r'\b\d+(?:[\.,]\d+)?\b')
TOKEN_RE = re.compile(r"(?:[#@]?\w[\w']*)", re.UNICODE)

lemmatizer = WordNetLemmatizer()
sw_set = set(stopwords.words('english'))

def deaccent(s: str) -> str:
    return unicodedata.normalize('NFKD', s).encode('ascii', 'ignore').decode('ascii')

def pos2wn(tag: str):
    if tag.startswith('J'): return wordnet.ADJ
    if tag.startswith('V'): return wordnet.VERB
    if tag.startswith('N'): return wordnet.NOUN
    if tag.startswith('R'): return wordnet.ADV
    return wordnet.NOUN

def clean_text_tokens(text: str, keep_hashtags=True, keep_mentions=False, drop_numbers=True):
    import nltk
    t = deaccent(str(text).lower())
    t = URL_RE.sub(' ', t)
```

```
        t = EMAIL_RE.sub(' ', t)
        if drop_numbers:
            t = NUM_RE.sub(' ', t)

        raw = TOKEN_RE.findall(t)

        def keep(tok):
            if tok.startswith('#') and not keep_hashtags: return False
            if tok.startswith('@') and not keep_mentions: return False
            return True

        toks = [tok.strip(''.join(punctuation)) for tok in raw if tok and keep(tok)]
        toks = [tok for tok in toks if tok and tok not in sw_set and not emoji.is_emoji(tok)]

        if toks:
            pos = nltk.pos_tag(toks)
            toks = [lemmatizer.lemmatize(tok, pos2wn(p)) for tok, p in pos]
        return toks

    # Clean twitter data
    clean_twitter = {}
    for artist, bios in twitter_desc.items():
        combined = " ".join(bios)
        clean_twitter[artist] = clean_text_tokens(combined, keep_hashtags=True, keep_mentions=False, drop_numbers=True)
```

```
# peek a few
for a in clean_twitter:
    print("="*60, f"\n{a} — TWITTER")
    _ = descriptive_stats(clean_twitter[a], num_tokens=5, verbose=True)
```

```
============================================================
cher — TWITTER
There are 14124688 tokens in the data.
There are 635069 unique tokens in the data.
There are 79161696 characters in the data.
The lexical diversity is 0.045 in the data.
============================================================
robynkonichiwa — TWITTER
There are 1356626 tokens in the data.
There are 145797 unique tokens in the data.
There are 7726466 characters in the data.
The lexical diversity is 0.107 in the data.
```

```
# clean lyrics data
clean_lyrics = {}
for artist, songs in lyrics_data.items():
    combined = " ".join(map(str, songs.values()))
    clean_lyrics[artist] = clean_text_tokens(
        combined,
        keep_hashtags=False,  # usually irrelevant in lyrics
        keep_mentions=False,
        drop_numbers=False    # numbers can be meaningful in lyrics
    )

print("Artists (lyrics cleaned): ", list(clean_lyrics.keys())[:5])
```

```
Artists (lyrics cleaned):  ['robyn', 'cher']
```

Basic Descriptive Statistics

Call your `descriptive_stats` function on both your lyrics data and your twitter data and for both artists (four total calls).

```
# calls to descriptive_stats here
artists = list(clean_lyrics.keys())
for artist in artists[:2]:
    print("="*60)
    print(f"{artist} — LYRICS")
    _ = descriptive_stats(clean_lyrics[artist], num_tokens=5, verbose=True)

    if artist in clean_twitter:
        print("-"*60)
        print(f"{artist} — TWITTER")
        _ = descriptive_stats(clean_twitter[artist], num_tokens=5, verbose=True)
```

```
============================================================
robyn — LYRICS
There are 12280 tokens in the data.
There are 1897 unique tokens in the data.
There are 57740 characters in the data.
The lexical diversity is 0.154 in the data.
============================================================
cher — LYRICS
There are 33289 tokens in the data.
There are 3026 unique tokens in the data.
There are 155052 characters in the data.
The lexical diversity is 0.091 in the data.
------------------------------------------------------------
cher — TWITTER
There are 14124688 tokens in the data.
There are 635069 unique tokens in the data.
There are 79161696 characters in the data.
The lexical diversity is 0.045 in the data.
```

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

`A:` If stopwords were left in, the most frequent words would likely be function words such as the, and, to, of, and in. These words occur at very high frequency but contribute little semantic meaning. Their presence would mask the more distinctive and meaningful terms in both the lyrics and the Twitter bios.

`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:` Lyrics generally display higher lexical diversity than Twitter bios because songs contain more creative and varied language, while bios are short and often repetitive. The results align with this expectation: Robyn's lyrics exhibited greater lexical diversity (0.154) compared to Cher's lyrics (0.091), and Cher's Twitter bios reflected a similar level of lexical diversity (0.091). These results indicate that both text type and artist style influence lexical diversity, with Robyn's songs demonstrating more linguistic variety than Cher's.

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.

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

Emojis 😁

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

```
emoji_counts_by_artist = {}
for artist, descs in twitter_desc.items():
    # examine raw text for emojis
    chars = []
    for d in descs:
        chars.extend(list(str(d)))
    emjs = [c for c in chars if emoji.is_emoji(c)]
    emoji_counts_by_artist[artist] = Counter(emjs).most_common(10)

emoji_counts_by_artist
```

```
{'cher': [('🤍', 75640),
  ('🌈', 45460),
  ('♥', 32880),
  ('🏳', 31901),
  ('✨', 28159),
  ('💙', 20674),
  ('⍰', 20100),
  ('🌊', 19652),
  ('💜', 15786),
  ('✌', 15703)],
 'robynkonichiwa': [('🤍', 4567),
  ('🌈', 4508),
  ('🏳', 3388),
  ('♥', 3017),
  ('✨', 2104),
  ('⍰', 1462),
  ('✌', 1123),
  ('⍰', 1096),
  ('♀', 805),
  ('💙', 781)]}
```

Hashtags

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

```
hashtag_counts_by_artist = {}
for artist, descs in twitter_desc.items():
    toks = []
    for d in descs:
        toks.extend(re.split(r"\s+", str(d).strip().lower()))
    hashtags = [t.strip("".join(punctuation)) for t in toks if t.startswith("#") and len(t) > 1]
    hashtags = [h for h in hashtags if h]  # clean empties
    hashtag_counts_by_artist[artist] = Counter(hashtags).most_common(10)

for artist, tags in hashtag_counts_by_artist.items():
    print("="*40)
    print(f"{artist} — Top 10 Hashtags")
    for h, c in tags:
        print(f"{h} : {c}")
```

```
========================================
cher — Top 10 Hashtags
resist : 10166
blm : 9285
blacklivesmatter : 7223
theresistance : 3098
fbr : 3066
resistance : 2697
1 : 2364
voteblue : 2008
lgbtq : 1757
music : 1411
========================================
robynkonichiwa — Top 10 Hashtags
blacklivesmatter : 547
blm : 336
music : 284
1 : 192
teamfollowback : 122
edm : 106
lgbtq : 81
resist : 76
art : 67
travel : 64
```

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.

```
def title_tokens(title: str):
    # reuse your cleaner; keep numbers (can be meaningful in titles)
    return clean_text_tokens(title, keep_hashtags=False, keep_mentions=False, drop_numbers=False)

title_words_by_artist = {}
for artist, songs in lyrics_data.items():
    toks = []
    for title in songs.keys():
        toks.extend(title_tokens(title))
    title_words_by_artist[artist] = Counter(toks).most_common(5)

# display (dict + readable printout)
title_words_by_artist
```

```
for artist, items in title_words_by_artist.items():
    print("="*40)
    print(f"{artist} — Top 5 Title Words")
    for w, c in items:
        print(f"{w} : {c}")
```

```
========================================
robyn — Top 5 Title Words
love : 5
get : 4
u : 3
know : 3
girl : 3
========================================
cher — Top 5 Title Words
love : 38
man : 12
song : 11
go : 10
time : 8
```

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.

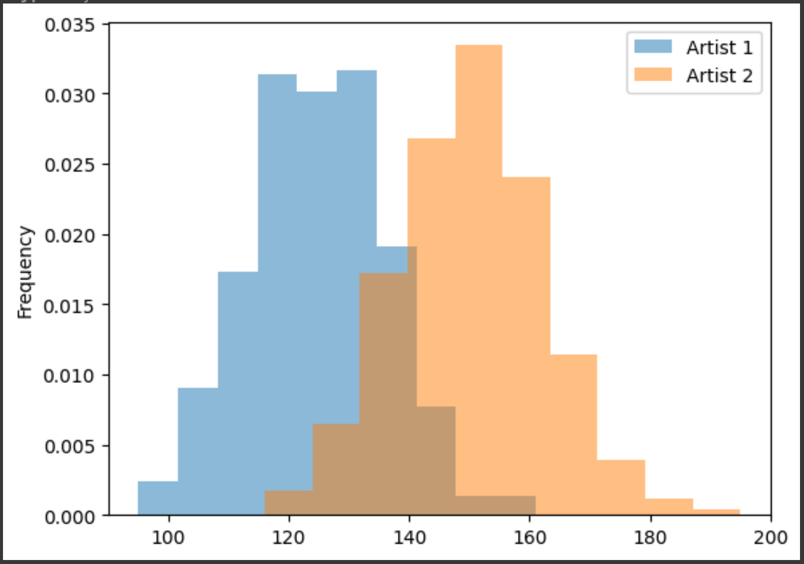
```
num_replicates = 1000

df = pd.DataFrame({
    "artist" : ['Artist 1'] * num_replicates + ['Artist 2']*num_replicates,
    "length" : np.concatenate((np.random.poisson(125,num_replicates),np.random.poisson(150,num_replicates)))
})

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

artist	length
Artist 1	Axes(0.125,0.11;0.775x0.77)
Artist 2	Axes(0.125,0.11;0.775x0.77)

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:` The pattern \s+ matches one or more whitespace characters, including spaces, tabs, and newlines. It is useful for collapsing multiple forms of whitespace into a single delimiter when splitting text into tokens.

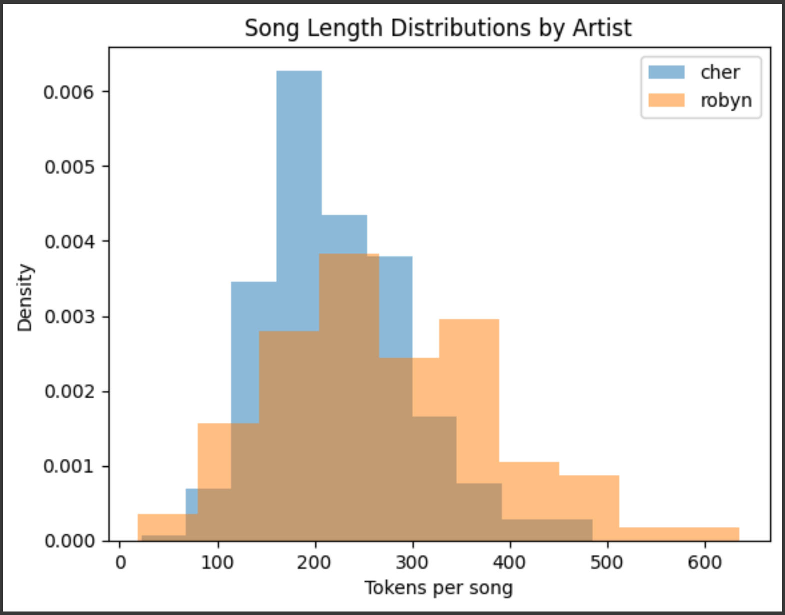
```
collapse_whitespace = re.compile(r'\s+')

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

```
# lyric length comparison chart here.
length_rows = []
for artist, songs in lyrics_data.items():
    for title, text in songs.items():
        toks = tokenize_lyrics(text.strip())
        length_rows.append({"artist": artist, "length": len(toks)})

length_df = pd.DataFrame(length_rows)

if not length_df.empty:
    length_df.groupby("artist")["length"].plot(
        kind="hist", density=True, alpha=0.5, legend=True
    )
    plt.xlabel("Tokens per song")
    plt.ylabel("Density")
    plt.title("Song Length Distributions by Artist")
    plt.show()
```


Song Length Distributions by Artist
cher
robyn
Density
Tokens per song