```python
import requests
import csv
import os
from dotenv import load_dotenv
import time
import json
from pathlib import Path

print("CWD:", os.getcwd())
print("Script dir:", Path(__file__).parent)
print(".env exists in CWD?", os.path.exists(".env"))


# Load environment variables from .env file
load_dotenv()

CLIENT_ID = os.getenv("SPOTIFY_CLIENT_ID")
CLIENT_SECRET = os.getenv("SPOTIFY_CLIENT_SECRET")

print("CLIENT_ID loaded:", bool(CLIENT_ID))
print("CLIENT_SECRET loaded:", bool(CLIENT_SECRET))

CSV_FILENAME = 'spotify_data.csv'

def init_csv():
    # Create CSV file with header if it does not exist
    if not os.path.isfile(CSV_FILENAME):
        with open(CSV_FILENAME, mode='w', newline='', encoding='utf-8') as file:
            writer = csv.writer(file)
            writer.writerow(['timestamp', 'track_name', 'artist_name', 'album_name', 'popularity'])
        print(f"Created new CSV file: {CSV_FILENAME}")

def append_tracks_to_csv(rows, filename=CSV_FILENAME):
    # Append rows of track data to CSV file
    with open(filename, mode='a', encoding='utf-8', newline='') as file:
        writer = csv.writer(file)
        writer.writerows(rows)

def get_token():
    # Function to get Spotify API token (placeholder implementation)
    auth = (CLIENT_ID, CLIENT_SECRET)
    r = requests.post(
        "https://accounts.spotify.com/api/token",
        data={"grant_type": "client_credentials"},
        auth=auth,
        timeout=30
    )
    r.raise_for_status()
    return r.json()['access_token']

def spotify_get(url, token, params=None, max_retries=5):
    headers = {"Authorization": f"Bearer {token}"}
    last_response = None

    for attempt in range(max_retries):
        r = requests.get(url, headers=headers, params=params, timeout=30)
        last_response = r

        if r.status_code == 429:
            retry = int(r.headers.get("retry-after", "1"))
            time.sleep(retry)
            continue

        if r.status_code in (500, 502, 503, 504):
            time.sleep(2 ** attempt)
            continue

        r.raise_for_status()
        return r.json()

    if last_response is not None:
        last_response.raise_for_status()
    raise Exception("Failed to get a valid response from Spotify API after retries.")

def get_genre_seeds(token):
    url = "https://api.spotify.com/v1/recommendations/available-genre-seeds"
    return spotify_get(url, token)

def get_categories(token, limit=50):
    url = "https://api.spotify.com/v1/browse/categories"
    return spotify_get(url, token, params={"limit": limit})

def get_category_playlists(category_id, token, limit=50):
    url = f"https://api.spotify.com/v1/browse/categories/{category_id}/playlists"
    return spotify_get(url, token, params={"limit": limit})

def search_playlists(keyword, token, limit=50):
    url = "https://api.spotify.com/v1/search"
    return spotify_get(url, token, params={"q": keyword, "type": "playlist", "limit": limit})

def scrape_genre(genre_keyword, token, playlist_limit=20):
    # Search public playlists by a given genre keyword
    # fetch all tracks for each playlist found
    # return list of track data rows via save_callback
    print(f"Scraping genre: '{genre_keyword}' ===")
    try:
        search_results = search_playlists(genre_keyword, token, limit=playlist_limit)
        playlist_items = search_results.get('playlists', {}).get('items', [])

        if not playlist_items:
            print(f"No playlists found for genre '{genre_keyword}'")
            return

        for idx, p in enumerate(playlist_items, start=1):
            if p is None:
                print(f"Skipping playlist #{idx} due to None value")
                continue

            playlist_id = p.get("id")
            playlist_name = p.get("name")
            track_info = p.get("tracks") or {}
            total_tracks = track_info.get("total")

            if not playlist_id:
                print(f"Skipping playlist #{idx} due to missing ID")
                continue
            print(f"Playlist: {playlist_id} - {playlist_name} ({total_tracks} tracks)")

            # Fetch all tracks in the playlist
            tracks = get_playlist_tracks(playlist_id, token)
            print(" -> tracks fetched:", len(tracks))

            # Build CSV rows
```

```
            rows = []
            for t in tracks:
                track = t.get("track")
                if track is None:
                    continue

                track_id = track.get("id")
                track_name = track.get("name")
                artists = ", ". join(a["name"] for a in track.get("artists", []))

                rows.append([
                    genre_keyword,
                    track_id,
                    playlist_name,
                    track_id,
                    track_name,
                    artists,
                ])

            # Append to CSV
            append_tracks_to_csv(rows)

    except requests.HTTPError as e:
        print(f"HTTP error occurred while scraping genre '{genre_keyword}': {e}")

def get_playlist_tracks(playlist_id, token):
    url = f"https://api.spotify.com/v1/playlists/{playlist_id}/tracks"
    items = []
    params = {"limit": 100, "offset": 0}
    while True:
        data = spotify_get(url, token, params=params)
        items.extend(data.get("items", []))

        next_url = data.get("next")
        if next_url:
            url = next_url
            params = None  # next_url already contains all params
        else:
            break
    return items

if __name__ == "__main__":
    init_csv()
    token = get_token()

    # Try genre seeds retrieval with error handling
    try:
        seeds = get_genre_seeds(token).get("genres", [])
        print("Genres (seed list):", seeds[30])
    except requests.HTTPError as e:
        if e.response.status_code == 404:
            print("Genre seeds endpoint not found (404). Skipping genre seed retrieval.")
        else:
            raise

    # Get categories, just to see if it works
    cats = get_categories(token)
    print("\nFirst 20 categories:")
    for c in cats.get("categories", {}).get("items", [])[:20]:
        print("cateogories:", c["id"], "-", c["name"])

    # Define list of genres to scrape
    genres_to_scrape = [
        "rock",
        "pop",
        "hip hop",
        "metal",
        "jazz",
        "classical",
        "country",
        "electronic",
        "r&b",
        "indie",
        "folk",
        "blues",
        "reggae",
        "punk",
        "soul",
        "latin",
        "k-pop",
    ]

    # Loop over all genres and scrape each one
    for genre in genres_to_scrape:
        scrape_genre(genre, token, playlist_limit=50)

    print("\nDone datascraping all genres.")
```

Visualization and clustering

NOTE: After completing the analysis, a fixed random seed was added to the notebook to ensure reproducible results in future runs. Because the seed was added only after the original experiments had been completed, some numerical outputs (cluster labels, cluster sizes, plots) in the current notebook may differ slightly from the screenshots and interpretations included in the report. These differences do not affect the methodology or conclusions — the clustering structure remains the same across runs, and only minor numeric values changed. The complete workflow is now fully reproducible for future use.

```
#libraries

import pandas as pd
import os
import json
import random
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from collections import Counter
import re

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import AgglomerativeClustering
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN
from mpl_toolkits.mplot3d import Axes3D


import umap

#seed
SEED = 1234
random.seed(SEED)
np.random.seed(SEED)
```

1. Dataset preparation and visualization

Collected datasets/files:

- Million Playlist Dataset: 1000 .json files x 1000 playlists (2017)
- Scraped Playlist Dataset: 76 playlists (2025)
- Million Songs Dataset: ~1.1 mio tracks with audio features

Goal: to create a dataset that contains playlists from both 2017 and 2025 playlist datasets, with added audio features for each track to enable better comparisons between tracks for our recommendation system. Because Spotify Audio Features API is deprecated and data cannot be retrieved from it anymore, I have to rely on the datasets that were made before November 2024, to collect the data about tracks. To achieve this, a sample subset of 1000 playlists will be selected from MPD. Coverage of tracks from MSD will be calculated and playlists with a coverage above a certain treshold will be added to the subset. This will limit the amount of missing values. The same principle applies to the scraped PD, but I expect the treshold there to be lower, since there are not as many playlists to choose from, and it should be taken into consideration that the songs might be newer and therefore not included in the MSD that was created before their release. After selecting the playlists, the .json file will be converted into a .csv file and a subset of scraped playlists will be created, merged together and features added.

1.1 Load raw data and create subset

Million Song Dataset & MDS Track ID list:

```
features_df = pd.read_csv("data/_raw/spotify_data.csv")   # your 1M-song feature dataset
msd_track_ids = set(features_df["track_id"].astype(str))
len(msd_track_ids)
```

Million Playlist Dataset subset

Create a subset of sampled playlists with coverage above a certain threshold (JSON): Threshold of 0.75 was chosen after a few trials. It should not be too low to avoid having too many missing values, and not too high - there must be at least one playlist from each file, and we need to be careful to avoid bias, since MSD contains mostly popular songs, which woul eliminate certain playlists with more obscure tracks.

```
# --- helper function to determine playlist coverage ---------------

def playlist_coverage(pl):

    tracks = pl.get("tracks", [])
    pl_track_ids = [track["track_uri"].split(":")[-1] for track in tracks]
    matched = sum(1 for tid in pl_track_ids if tid in msd_track_ids)

    return matched / len(pl_track_ids)

#-----------------------------------------------------------------------
```

```
data_dir = "data/_raw/data"   #directory with 1000 MPD files
output_file = "data/mpd_sample_1000.json"

#at least 75% of tracks in a playlist must be able to get audio
features from MSD
threshold = 0.75

#list of playlists that fit the criteria - 1 from each file
sampled_playlists = []

#list of files
files = sorted(os.listdir(data_dir))

#reproductibility!!!!!
random.seed(SEED)
np.random.seed(SEED)

for filename in files:
    filepath = os.path.join(data_dir, filename)
    #print(filename)
    with open(filepath, 'r', encoding='utf-8') as f:
        data = json.load(f)
        playlists = data.get('playlists', [])

        #find the first high coverage playlist
        candidates = [pl for pl in playlists if playlist_coverage(pl)
>= threshold]
        chosen = random.choice(candidates)
        sampled_playlists.append(chosen)
        #print(playlist_coverage(chosen))

#check length (should be 1000 if all files had at least 1 playlist fit
the criteria)
print("Sampled playlists:", len(sampled_playlists))


#save into a new json file
with open(output_file, "w", encoding="utf-8") as f:
    json.dump(sampled_playlists, f, indent=2)
```

```
Sampled playlists: 1000
```

Transform the subset of playlists to a .csv file:

```
#read json sample file
with open("data/mpd_sample_1000.json") as f:
    data_json = json.load(f)

#a list of dictionaries (1 dict - 1 track)
rows = []

'''
example of track data:

"pos": 0,
"artist_name": "G-Eazy",
"track_uri": "spotify:track:40YcuQysJ0KlGQTeGUosTC",
"artist_uri": "spotify:artist:02kJSzxNuaWGqwubyUba0Z",
"track_name": "Me, Myself & I",
"album_uri": "spotify:album:09Q3WwGYsQe5ognkvVkmCu",
"duration_ms": 251466,
"album_name": "When It's Dark Out"

'''

for playlist in data_json:
    playlist_id = playlist["pid"]
    playlist_name = playlist["name"]

    for track in playlist["tracks"]:
        #clean uris
        track_id = track["track_uri"].split(":")[-1]
        artist_id = track["artist_uri"].split(":")[-1]
        album_id = track["album_uri"].split(":")[-1]

        rows.append({
            "playlist_id": playlist_id,
            "playlist_name": playlist_name,
            "artist_id": artist_id,
            "artist_name": track["artist_name"],
            "track_id": track_id,
            "track_name": track["track_name"],
            "album_id": album_id,
            "album_name": track["album_name"],
            "duration_ms": track["duration_ms"],
            "position": track["pos"]
        })

df_json = pd.DataFrame(rows)
df_json.to_csv("data/mpd_playlists.csv", index=False)

df_json.sample(n=5, random_state=SEED)
```

{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"playlist_id","rawType":"int64","type":"integer"},
{"name":"playlist_name","rawType":"object","type":"string"},
{"name":"artist_id","rawType":"object","type":"string"},
{"name":"artist_name","rawType":"object","type":"string"},
{"name":"track_id","rawType":"object","type":"string"},
{"name":"track_name","rawType":"object","type":"string"},
{"name":"album_id","rawType":"object","type":"string"},
{"name":"album_name","rawType":"object","type":"string"},
{"name":"duration_ms","rawType":"int64","type":"integer"},
{"name":"position","rawType":"int64","type":"integer"}],"ref":"a77cbb6
b-2b09-44e9-bb3b-f759e0d1e7cc","rows":
[["16461","58871","mine.","06HL4z0CvFAxyc27GXpf02","Taylor
Swift","5VwFkx7JOimOGTYfha5rs1","I Almost
Do","1EoDsNmgTLtmwe1BDAVxV5","Red","242573","122"],
["8578","340423","bet","7iZtZyCzp3LItcw1wtPI3D","Rae
Sremmurd","4scpF6J5uMBvoh6sFB7EL1","No
Type","6eDx949ONWDCN0O22wFZf7","SremmLife","200080","24"],
["249","107683","Good Country","0BvkDsjIUla7X0k6CSWh1I","Luke
Bryan","1PoGWZbJPGmViVi7CYbDUK","Drink A
Beer","5M8gr5RV2eR6UkztC69ogB","Crash My Party","202626","55"],
["18437","617005","BRUH","31RyGWziBGfdbaZk9ZSmTE","Max
Minelli","3S1o3PmUNYOwoeAPcTxNdF","They Can Hate (feat. Max
Minelli)","6m6h4C733ycfTkMz5nJJDe","\"I Don't Know What To Call It\"
Vol. 1","265812","63"],
["15311","553590","Wedding","29ywwKkxfoH7iWwNY1UezA","Francesca
Battistelli","7fSWlTKRsn3jkhe7BiMbIs","Worth
It","1s6NqofRkHJN1eThDu948E","Hundred More
Years","226213","6"]],"shape":{"columns":10,"rows":5}}

Scraped Dataset subset

Check coverage:

```
#read file
scraped = pd.read_csv("data/_raw/scraped/spotify_tracks.csv")

#mask column: is each track present in msd
scraped['in_msd'] = scraped['track_id'].apply(lambda x: x in
msd_track_ids)

#coverage portion of playlist
coverage_df = (
    scraped.groupby('playlist_id')['in_msd']
           .agg(['sum', 'count'])
           .reset_index()
)

#average per playlist
```

```
coverage_df['coverage'] = coverage_df['sum'] / coverage_df['count']

#check range
print("Best coverage: ", np.max(coverage_df['coverage']))
print("Mean coverage: ", np.mean(coverage_df['coverage']))
```

```
Best coverage:  0.6507936507936508
Mean coverage:  0.14339813567913973
```

The coverage of scaped data by MSD is significantly lower than with the MPD. That is because this data is newer and contains songs that were released after MSD was already made, and because scraped data contains only 76 playlists, which is very little to pick from. After a few trials I decided on a threshold of 0.15, which is a little bit above average and eliminates the worst 2/3 of the data.

```
threshold = 0.15

#get ids of good playlists
good_ids = coverage_df[coverage_df['coverage'] >= threshold]
['playlist_id']
print(len(good_ids))

#filter and keep good playlists
scraped_filtered = scraped[scraped['playlist_id'].isin(good_ids)]

#drop mask column
scraped_filtered = scraped_filtered.drop(columns=('in_msd'))

#save new subset to csv
scraped_filtered.to_csv("data/scraped_filtered.csv", index=False)
```

```
26
```

Merge both subsets

```
#read files
scraped_subset = pd.read_csv("data/scraped_filtered.csv")
mpd_subset = pd.read_csv("data/mpd_playlists.csv")

#rename to fit mpd column
scraped_subset = scraped_subset.rename(columns={"artists":
"artist_name"})

#combine
df_all = pd.concat([scraped_subset, mpd_subset], ignore_index=True)

#fix dtype conflict
df_all['playlist_id'] = df_all['playlist_id'].astype(str)

#save
```

```
df_all.to_csv("data/combined_data.csv")

df_all.sample(n=10)
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"genre","rawType":"object","type":"unknown"},
{"name":"playlist_id","rawType":"object","type":"string"},
{"name":"playlist_name","rawType":"object","type":"string"},
{"name":"track_id","rawType":"object","type":"string"},
{"name":"track_name","rawType":"object","type":"string"},
{"name":"artist_name","rawType":"object","type":"unknown"},
{"name":"artist_id","rawType":"object","type":"unknown"},
{"name":"album_id","rawType":"object","type":"unknown"},
{"name":"album_name","rawType":"object","type":"unknown"},
{"name":"duration_ms","rawType":"float64","type":"float"},
{"name":"position","rawType":"float64","type":"float"}],"ref":"141d0f2
7-796b-4e9a-a29b-c52546d9deab","rows":
[["27862",null,"534264","LIT","1wHZx0LgzFHyeIZkUydNXq","Antidote","Tra
vis
Scott","0Y5tJX1MQlPlqiwlOH1tJY","4PWBTB6NYSKQwfo79I3prg","Rodeo","2626
93.0","7.0"],["33117",null,"666832","Lord of the
Rings","0dlAN15Cr7eIOqhqtFjfHV","The Passing Of Théoden","Howard
Shore","0OcclcP5o8VKH2TRqSY2A7","2Re6RMBEfD8iv1HMxLMdzI","The Lord Of
The Rings - The Return Of The King - The Complete
Recordings","136320.0","73.0"],["27240",null,"52601","Country
Favs","6Ymvlzom4TQeoKqAWsZRD8","Somethin' 'Bout A Truck","Kip
Moore","2hJPr4lk7Q8SSvCVBl9fWM","191BU6Uvnf7oNTjO4n36Yu","Up All
Night","213826.0","9.0"],["40088",null,"873452","summer
2015","5w9L4OCR45z7Y0KeKlDVkA","Stand by Your Gun","George
Ezra","2ysnwxxNtSgbb9t1m2Ur4j","5tF2lAa2rh2kU2xIiBzWia","Wanted On
Voyage","184493.0","3.0"],["38797",null,"833674","Luke Bryan
Concert","2aYudnZ9lYxIxmhRr1iByK","See About A Girl","Lee
Brice","5Zq7R5qmi58ByYyBQTlNuk","0RnVSSUbSBEjk5MlQZhYYP","Hard 2
Love","236960.0","29.0"],
["16568",null,"198377","HAMILTON","7EsSVPxaYoAZjQwhspJBs2","Who Lives,
Who Dies, Who Tells Your Story","Original Broadway Cast of
Hamilton","3UUJfRbrA2nTbcg4i0MOwu","1kCHru7uhxBUdzkm4gzRQc","Hamilton"
,"217229.0","45.0"],["27254",null,"52601","Country
Favs","0cITLOYn1Sv4q27zZPqlNK","Red","Taylor
Swift","06HL4z0CvFAxyc27GXpf02","1EoDsNmgTLtmwe1BDAVxV5","Red","220826
.0","23.0"],["262","rock","29fOXPbR3qClySksICQ2Yn","ROCK
MUSIK","2pxAohyJptQWTQ5ZRWYijN","The Trooper - 1998 Remastered
Version",null,null,null,null,null,null],
["31320",null,"615893","favorite songs","7MbII8jWqFiTFYU3MPYEfW","Rock
Me","One
Direction","4AK6F7OLvEQ5QYCBNiQWHq","5SxEsi1PNyo1XfEKDYcFKF","Take Me
Home: Yearbook Edition","200040.0","33.0"],
["39764",null,"861267","summer country","7mldq42yDuxiUNn08nvzHO","Body
Like A Back Road","Sam
```

Hunt","2kucQ9jQwuD8jWdtR9Ef38","2N7kidh1wA9EoLdf16QWrz","Body Like A Back Road","165386.0","2.0"]],"shape":{"columns":11,"rows":10}}

Add the MSD data:

For the overlapping columns the data from playlist datasets has priority.

```
features_df = pd.read_csv("data/_raw/spotify_data.csv")
df_all = pd.read_csv("data/combined_data.csv")

#column names check
print(features_df.columns)
print(df_all.columns)

#drop index column
features_df = features_df.drop(columns="Unnamed: 0")
df_all = df_all.drop(columns="Unnamed: 0")
```

```
Index(['Unnamed: 0', 'artist_name', 'track_name', 'track_id',
'popularity',
       'year', 'genre', 'danceability', 'energy', 'key', 'loudness',
'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'duration_ms', 'time_signature'],
      dtype='object')
Index(['Unnamed: 0', 'genre', 'playlist_id', 'playlist_name',
'track_id',
       'track_name', 'artist_name', 'artist_id', 'album_id',
'album_name',
       'duration_ms', 'position'],
      dtype='object')
```

```
#merge based on track id
merged = df_all.merge(
    features_df,
    on="track_id",
    how="left",
    suffixes=("", "_feat")
)

#filling NAs with info from features df
feat_cols = [c for c in merged.columns if c.endswith("_feat")]
for f in feat_cols:
    base = f[:-5]
    merged[base] = merged[base].fillna(merged[f])

#removing duplicated features
merged = merged.drop(columns=feat_cols)

#added this later - replace genre "hip hop" with "hip-hop"
```

```
merged['genre'] = merged['genre'].replace("hip hop", "hip-hop")

merged.to_csv("data/.csv", index=False)

merged.sample(n=10)
```

1.2 Exploratory data analysis of the sampled data set

```
#load sampled data set
df = pd.read_csv("data/sample_merged.csv", low_memory=False)
```

Number of playlists, tracks, artists, genres:

```
print(f"Number of rows: {len(df)}")

playlist_col = "playlist_id"
track_col    = "track_id"
artist_col   = "artist_id"
genre_col = "genre"

print(f"Number of unique playlists: {df[playlist_col].nunique()}")
print(f"Number of unique tracks: {df[track_col].nunique()}")
print(f"Number of unique artists: {df[artist_col].nunique()}")
print(f"Number of genres: {df[genre_col].nunique()}")
```

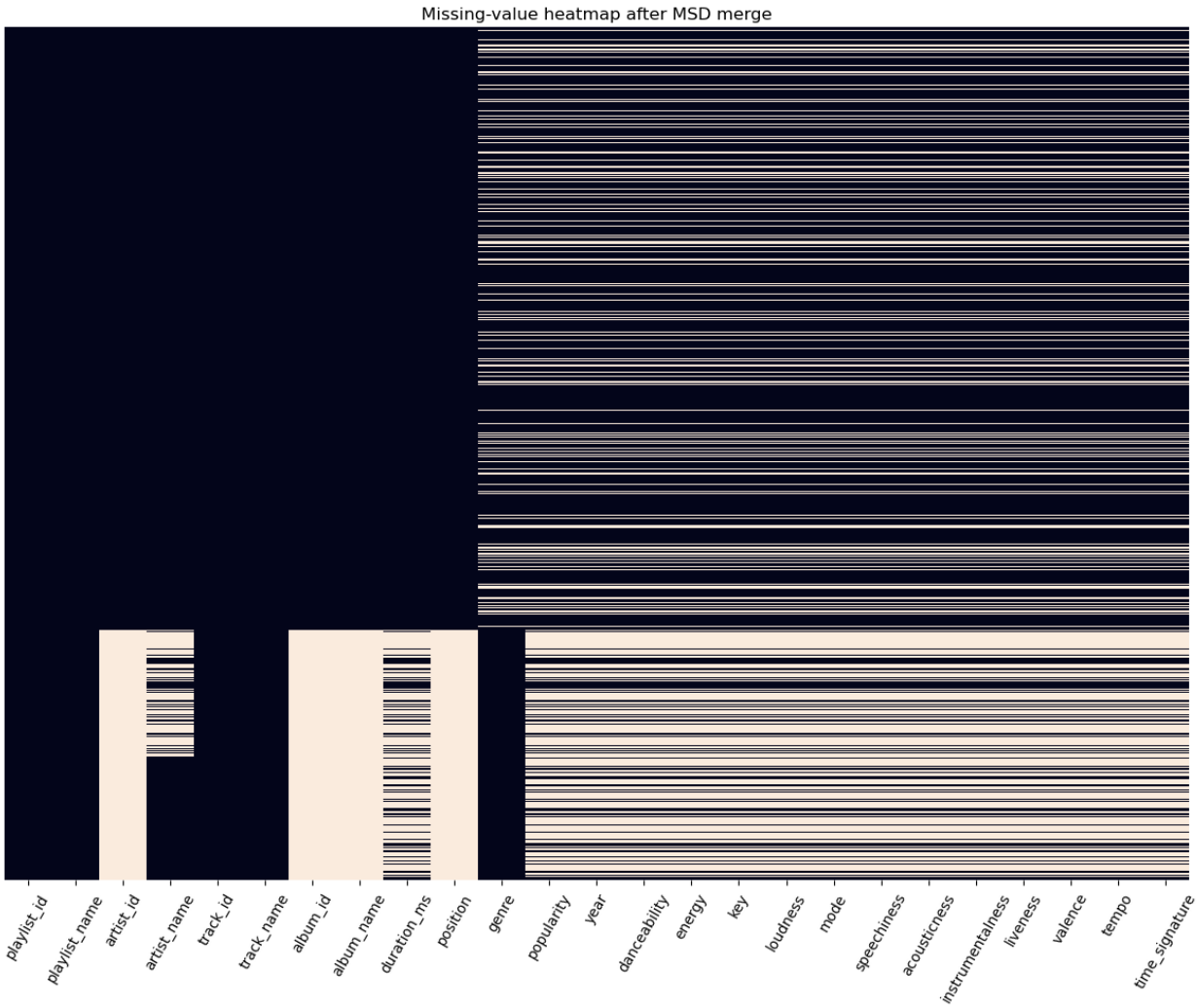
```
Number of rows: 44472
Number of unique playlists: 1026
Number of unique tracks: 16393
Number of unique artists: 1812
Number of genres: 60
```

```
df[genre_col].unique()
```

```
array(['dance', nan, 'country', 'edm', 'hip-hop', 'pop', 'indie-pop',
       'emo', 'hardcore', 'metalcore', 'punk', 'rock', 'electro',
       'gospel', 'dub', 'chill', 'swedish', 'jazz', 'show-tunes',
'house',
       'acoustic', 'k-pop', 'metal', 'alt-rock', 'sleep', 'german',
       'piano', 'hard-rock', 'classical', 'soul', 'drum-and-bass',
'folk',
       'singer-songwriter', 'trance', 'funk', 'club', 'punk-rock',
       'death-metal', 'goth', 'french', 'deep-house', 'disco',
       'minimal-techno', 'power-pop', 'blues', 'breakbeat',
'songwriter',
       'comedy', 'dancehall', 'new-age', 'ambient', 'ska', 'sad',
       'indian', 'groove', 'black-metal', 'psych-rock', 'hardstyle',
       'electronic', 'guitar', 'hip hop'], dtype=object)
```

Missing value distribution and handling:

```
missing_count = df.isna().sum()
missing_percent = df.isna().mean() * 100

missing_summary = pd.DataFrame({
    "missing_count": missing_count,
    "missing_percent": missing_percent.round(2)
}).sort_values("missing_percent", ascending=False)

def plot_missing_values(df, title="Missing-value heatmap after MSD
merge"):
    plt.figure(figsize=(12,10))
    sns.heatmap(df.isna(), cbar=False)
    plt.title(title)
    plt.xticks(rotation=60, fontsize=10)
    plt.yticks([])
    plt.tight_layout()
    plt.show()

plot_missing_values(df)
print(missing_summary)
```



	missing_count	missing_percent
year	14954	33.63
danceability	14954	33.63
tempo	14954	33.63
valence	14954	33.63
liveness	14954	33.63
instrumentalness	14954	33.63
acousticness	14954	33.63
speechiness	14954	33.63
mode	14954	33.63
loudness	14954	33.63
key	14954	33.63
energy	14954	33.63
time_signature	14954	33.63
popularity	14954	33.63
position	13050	29.34
album_name	13050	29.34
album_id	13050	29.34

```
artist_id                13050            29.34
duration_ms               8869            19.94
genre                     6085            13.68
artist_name               4472            10.06
playlist_name                0             0.00
track_name                   2             0.00
track_id                     0             0.00
playlist_id                  0             0.00
```

Approximately 33% of all tracks are missing every audio feature. Since these features are essential for PCA and clustering, I will proceed with two parallel strategies:

Dropping Missing-Feature Tracks A second dataset will be created by removing all tracks that lack audio features entirely, ensuring a fully complete dataset for comparison.

Per-Genre Imputation To preserve as much data as possible, missing audio features will be imputed using the median value within each genre. If a track has no valid genre label, the most common genre within its playlist will be used as a fallback.

Almost 14% of the tracks are missing genre as well. Since genre information is able to come from either dataset (playlist subset or MSD), it can be assumed that tracks who lack genre lack audio features as well. After dropping Missing-Feature tracks, all tracks should have genre. Before per-genre imputation, missing values will be filled with the mode genre of the track's playlist.

Summary statistics for audio features:

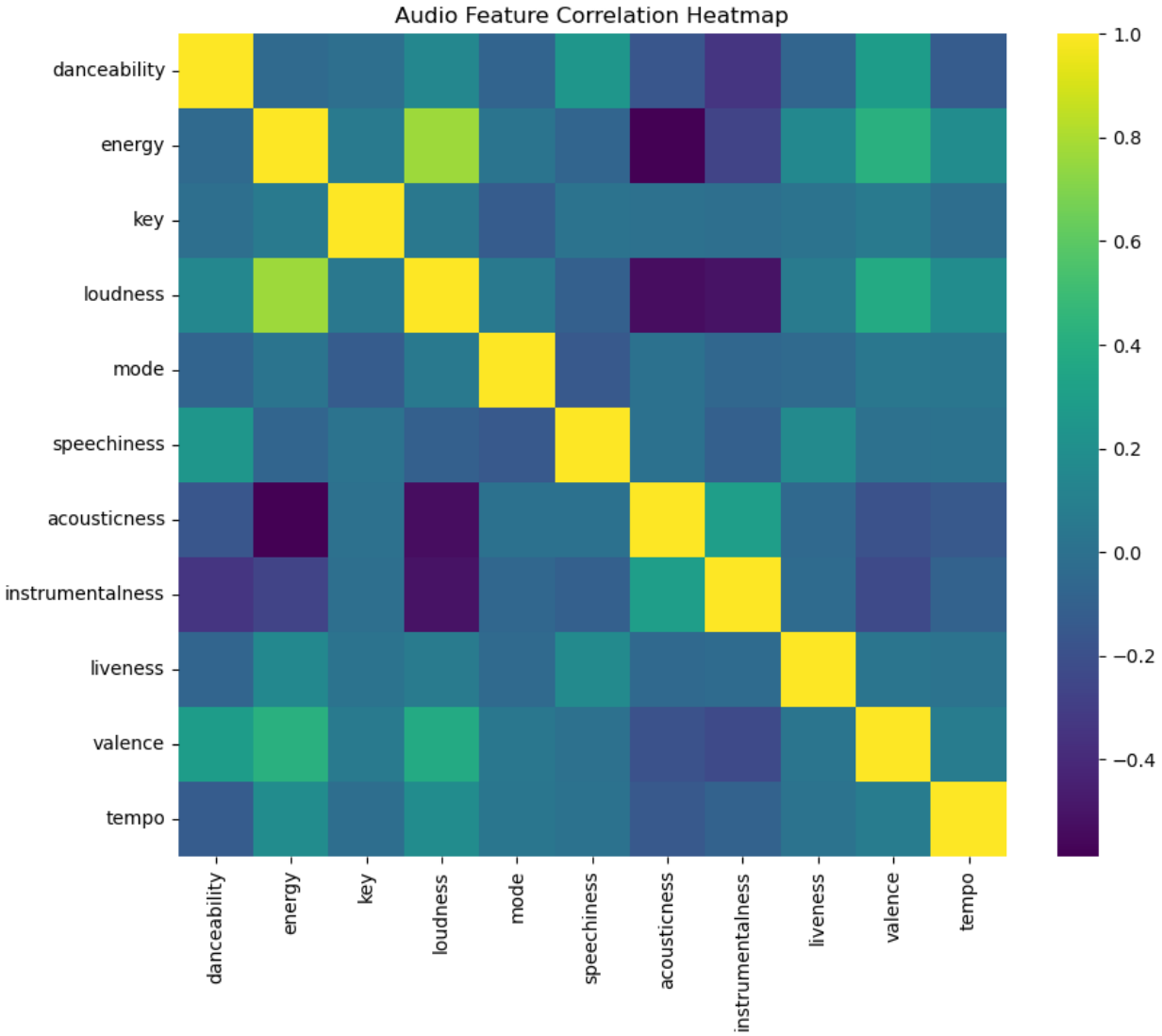
```
#list of audio features
audio_features = [
    'danceability', 'energy', 'key',
    'loudness', 'mode', 'speechiness', 'acousticness',
'instrumentalness',
    'liveness', 'valence', 'tempo'
]

#check if all are numerical
df[audio_features].dtypes

#summary statistics
summary_stats = df[audio_features].agg(["mean", "median", "std"]).T
print(summary_stats)
```

```
                      mean    median       std
danceability      0.609654    0.6090  0.156048
energy            0.688284    0.7140  0.190969
key               5.223660    5.0000  3.645743
loudness         -6.300783   -5.7500  3.032534
mode              0.672369    1.0000  0.469357
speechiness       0.119751    0.0580  0.121692
acousticness      0.174601    0.0803  0.221068
instrumentalness  0.027520    0.0000  0.140618
```

```
liveness              0.199071    0.1350   0.156264
valence               0.479494    0.4750   0.225224
tempo               124.780383  125.0100  30.151431
```

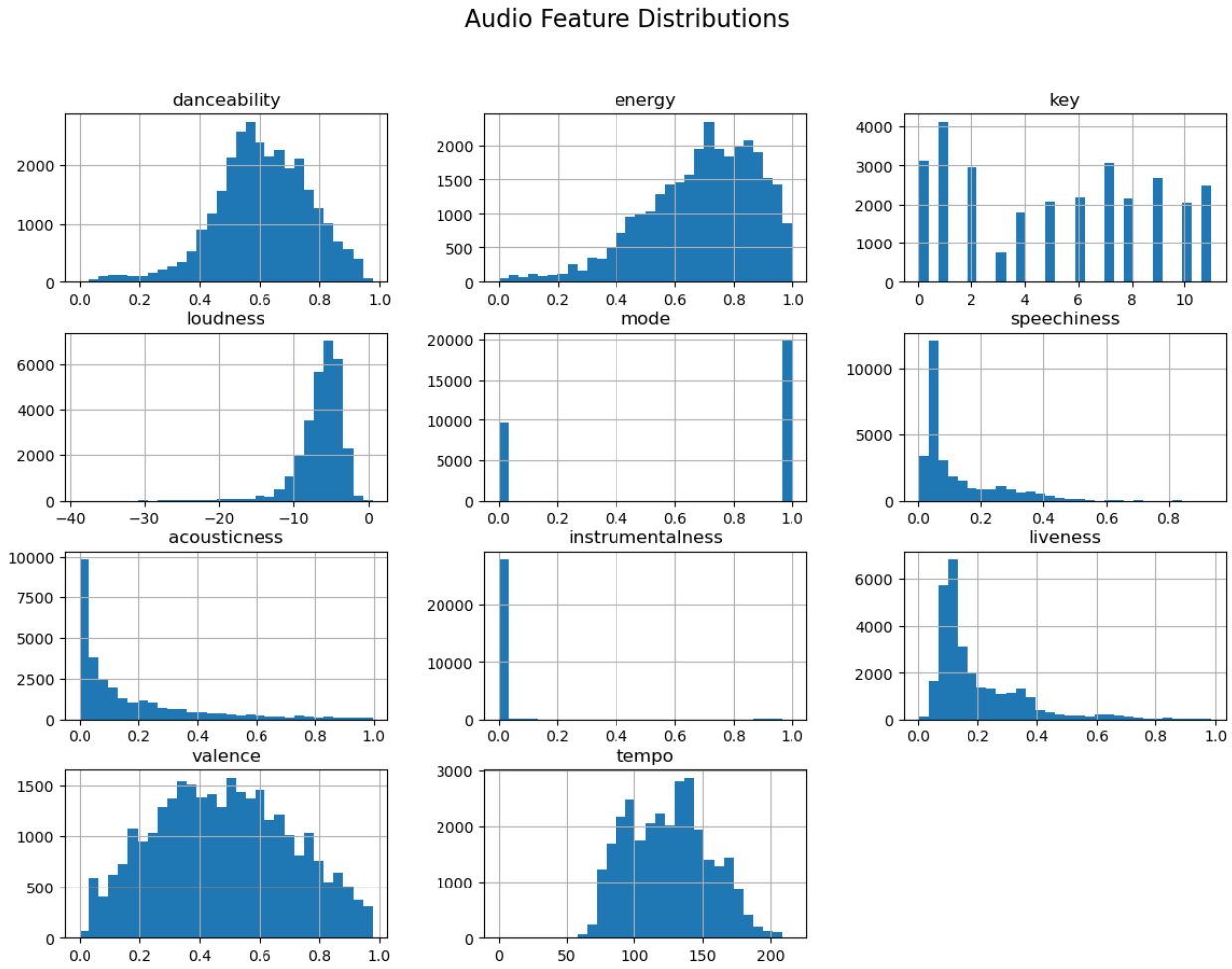
Correlation heatmap of audio features:

```
def plot_audio_correlations(df, audio_cols):
    plt.figure(figsize=(10,8))
    corr = df[audio_cols].corr()
    sns.heatmap(corr, cmap="viridis", annot=False)
    plt.title("Audio Feature Correlation Heatmap")
    plt.show()

plot_audio_correlations(df, audio_features)
```



Distribution plots for audio features:

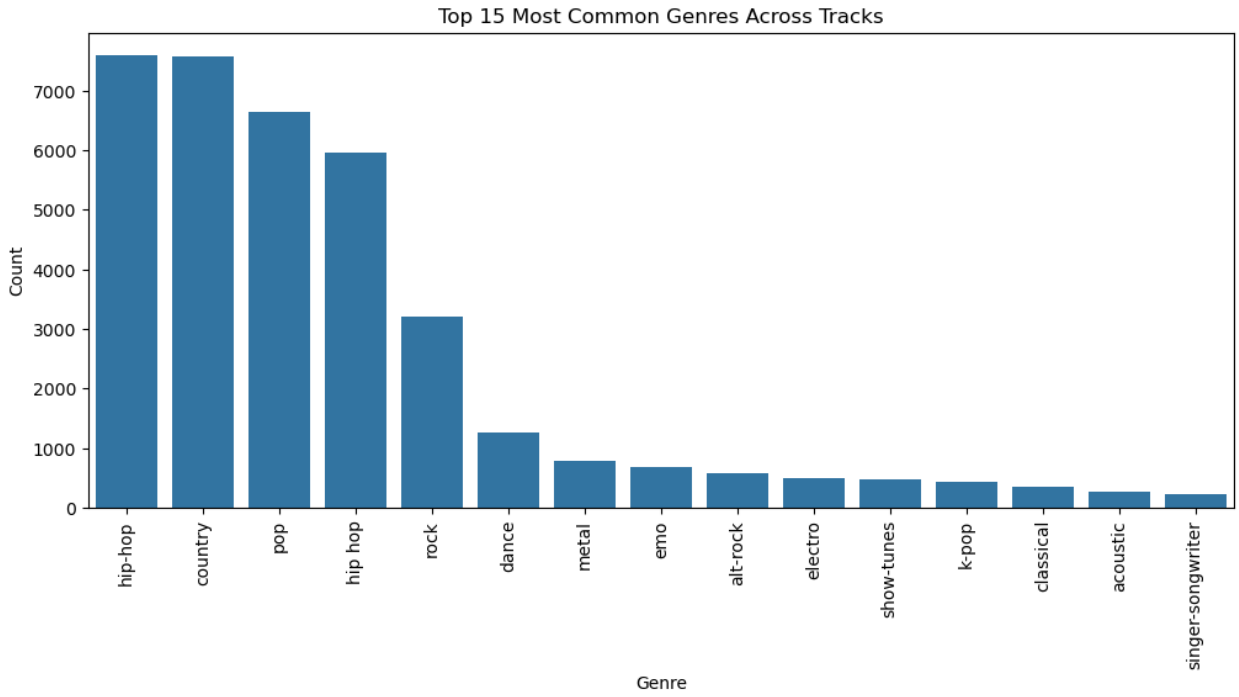
```
def plot_audio_distributions(df, audio_cols):
    df[audio_cols].hist(figsize=(14,10), bins=30)
    plt.suptitle("Audio Feature Distributions", fontsize=16)
    plt.show()

plot_audio_distributions(df, audio_features)
```



Genre distribution

```
def plot_top_genres(df, n=15):
    plt.figure(figsize=(12, 5))

    #get top n genres
    top = df['genre'].value_counts().head(n).index

    sns.countplot(
        data= df[df['genre'].isin(top)],
        x='genre',
        order=top
    )
```

```
    plt.xticks(rotation=90)
    plt.title(f"Top {n} Most Common Genres Across Tracks")
    plt.xlabel("Genre")
    plt.ylabel("Count")
    plt.show()

plot_top_genres(df, 15)
```



This shows that sampled data is very biased, which will likely affect clustering and further analysis. This happened because MSD covers mainly hip-hop and popular tracks. Hip-hop accounts for more than 1/3 of the data. To fix this, genres will be normalized at playlist level, which will hopefully enable playlist composition to be presented in the clusters.

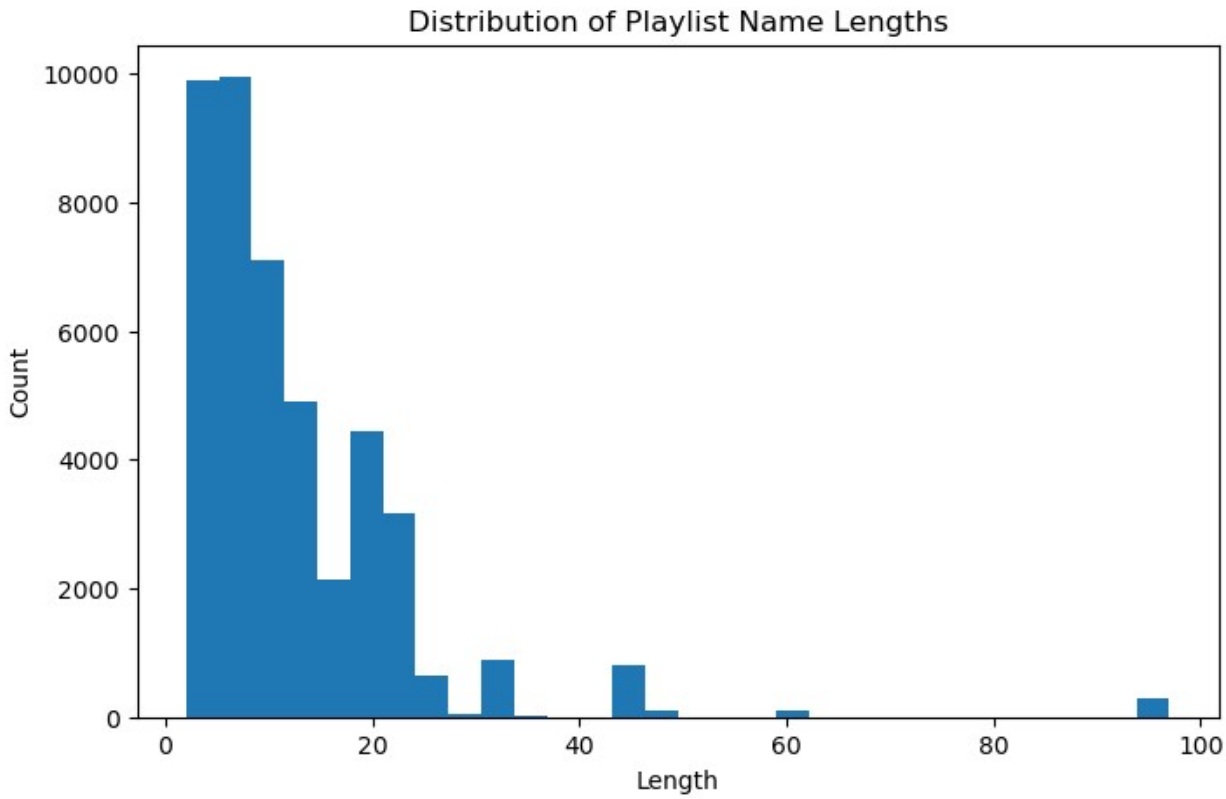
Another option would be to downsample hip-hop tracks for clustering. However, that is not a realistic approach for building a recommender.

Playlist titles statistics

Playlist name lengths:

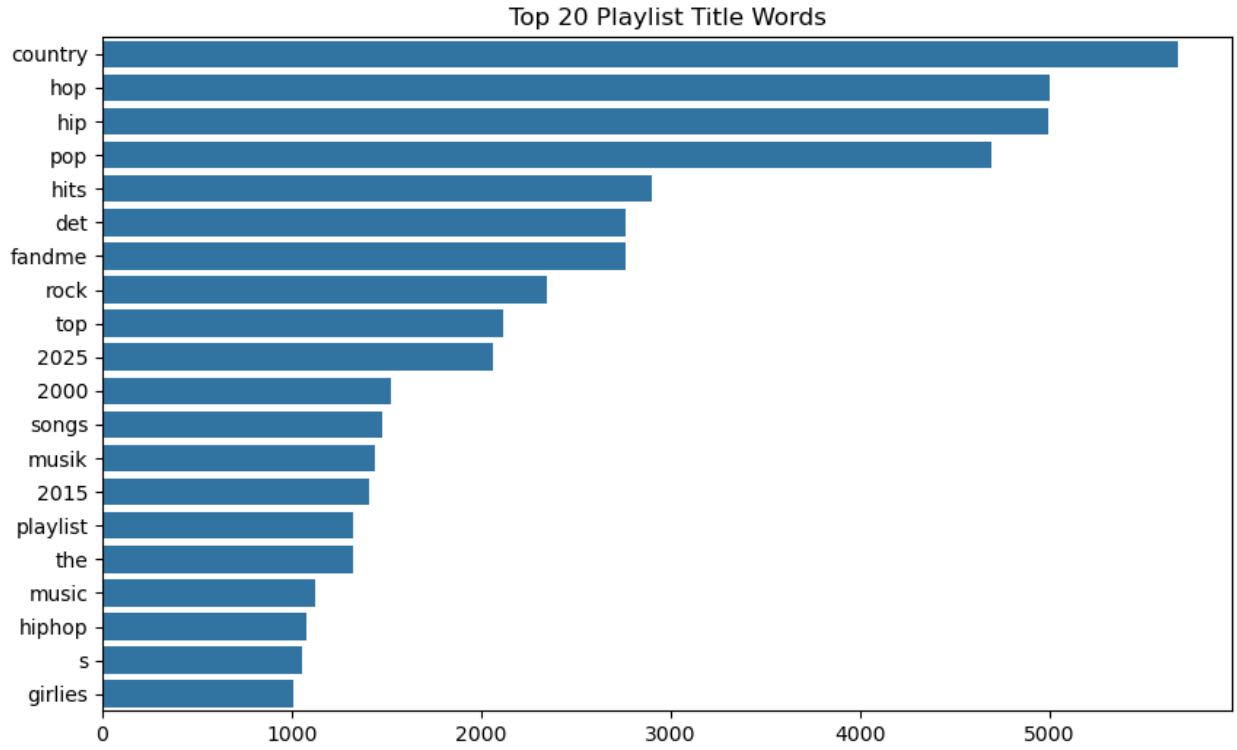
```
def plot_playlist_name_lengths(df):
    lengths = df['playlist_name'].apply(len)
    plt.figure(figsize=(8,5))
    plt.hist(lengths, bins=30)
    plt.title("Distribution of Playlist Name Lengths")
    plt.xlabel("Length")
    plt.ylabel("Count")
```

```
    plt.show()

plot_playlist_name_lengths(df)
```



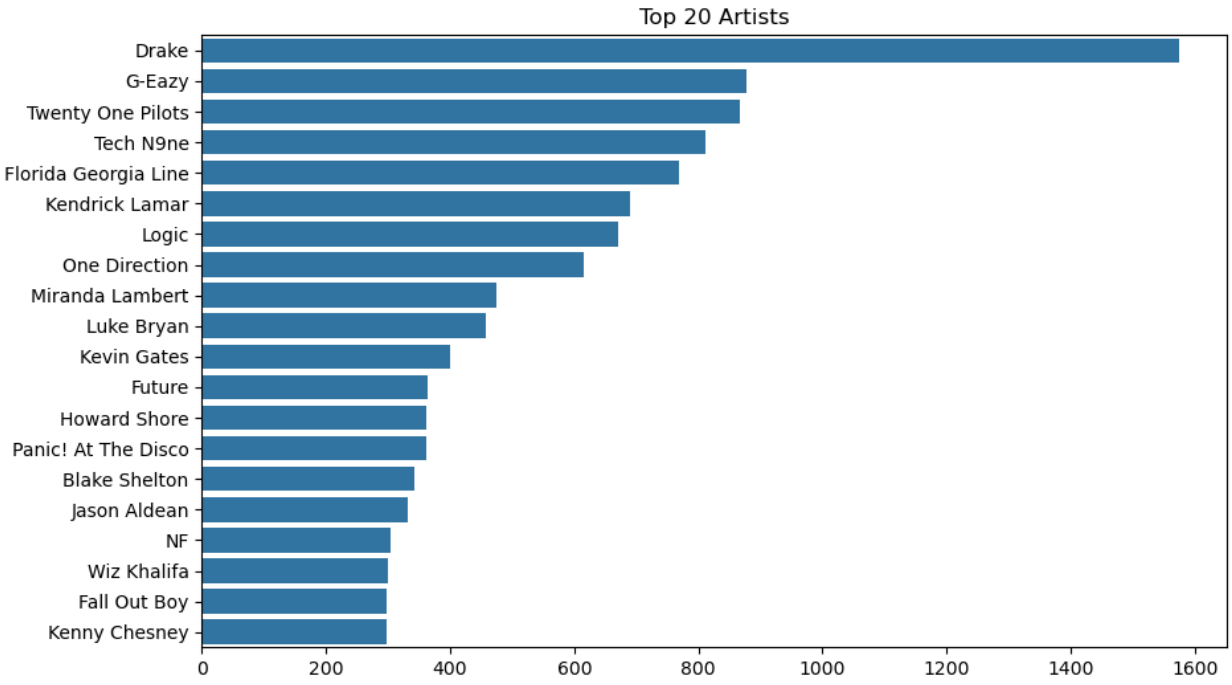
Common playlist title words:

```
def plot_common_playlist_words(df, n=20):
    names = df['playlist_name'].dropna().str.lower()
    words = re.findall(r'\w+', " ".join(names))

    counter = Counter(words)
    most = counter.most_common(n)

    plt.figure(figsize=(10,6))
    sns.barplot(x=[c[1] for c in most], y=[c[0] for c in most])
    plt.title(f"Top {n} Playlist Title Words")
    plt.show()

plot_common_playlist_words(df, 20)
```



Common artists:

```
def plot_common_artists(df, n=20):
    artists = df['artist_name'].dropna()
    counter = Counter(artists)
    most = counter.most_common(n)

    plt.figure(figsize=(10,6))
    sns.barplot(x=[c[1] for c in most], y=[c[0] for c in most])
    plt.title(f"Top {n} Artists")
    plt.show()

plot_common_artists(df, 20)
```



1.3 The three analysis datasets

Dataset with dropped NAs

```
def create_dropna_dataset(df, cols, audio_features):
    df = df[cols]
    return df.dropna(subset=cols).copy()

cols = ['playlist_id', 'genre'] + audio_features
df_drop = create_dropna_dataset(df, cols, audio_features)

df_drop.to_csv("data/data_audio_dropped.csv")
```

Dataset with imputed values (per-genre)

```
#calculating global medians
global_medians = {}

#select only genre + audio columns
audio_df = df[cols]

for c in audio_features:
    med = audio_df[c].median(skipna=True)
    global_medians[c] = med

#print(global_medians)

#calculating per-genre medians
genres = audio_df['genre'].unique()
genre_medians = audio_df.groupby('genre')[audio_features].median()
```

```
#print(genre_medians.sample(n=5))

#find mode of genre per playlist and fill missing genre values
num_missing_genre = audio_df['genre'].isna().sum()
#print(num_missing_genre)

playlist_genre_mode = (audio_df.groupby('playlist_id')
['genre'].agg(lambda x: x.mode()[0]))
#print(playlist_genre_mode.sample(n=5))

#fill missing genres in tracks
df_imp = audio_df.merge(playlist_genre_mode.rename("playlist_genre"),
                    on="playlist_id", how='left')
df_imp['genre'] = df_imp['genre'].fillna(df_imp['playlist_genre'])
df_imp = df_imp.drop(columns=['playlist_genre'])
#df_imp.sample(n=5)

#fill missing audio features in tracks based on genre
for c in audio_features:
    df_imp[c] = df_imp[c].fillna(
        df_imp['genre'].map(genre_medians[c])
    )

#verify no missing values
df_imp.isna().sum().sum()

df_imp.to_csv("data/data_audio_imputed.csv")
```

Dataset with text

```
df.columns
text_cols = ['playlist_id', 'playlist_name', 'track_name',
'artist_name', 'album_name']
df_text = df[text_cols]
df_text.to_csv("data/data_text.csv")
```

1.4 Per-playlist feature aggregation:

Audio + genre:

```
#PLAYLIST-LEVEL FEATURE AGGREGATION
def aggregate_playlists(df):

    playlist_id = "playlist_id"
    track_id = "track_id"
    num_audio_cols = ['danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness',
'instrumentalness',
       'liveness', 'valence', 'tempo']
```

```
    genre = "genre"

    #numeric aggregation
    mean_df = df.groupby(playlist_id)[num_audio_cols].mean()
    mean_df.columns = [f"{c}_mean" for c in mean_df.columns]

    #genre proportion in a playlist: one-out-of-K encoding
    genre_dummies = pd.get_dummies(df[genre], prefix="genre")
    genre_df = pd.concat([df[[playlist_id]], genre_dummies], axis=1)
    genre_props = genre_df.groupby(playlist_id).mean()

    #print(genre_props.sample(n=5))

    #join
    playlist_features = (
        mean_df
        .join(genre_props, how="left")
    )

    #print(playlist_features.columns)

    return playlist_features

df_imp = pd.read_csv("data/data_audio_imputed.csv")
df_drop = pd.read_csv("data/data_audio_dropped.csv")

aggr_imp = aggregate_playlists(df_imp)
aggr_drop = aggregate_playlists(df_drop)

aggr_imp.to_csv("data/audio_imputed_aggr.csv")
aggr_drop.to_csv("data/audio_dropped_aggr.csv")

aggr_imp.sample(n=5, random_state=SEED)
```


```
{"columns":[{"name":"playlist_id","rawType":"object","type":"string"},
{"name":"danceability_mean","rawType":"float64","type":"float"},
{"name":"energy_mean","rawType":"float64","type":"float"},
{"name":"key_mean","rawType":"float64","type":"float"},
{"name":"loudness_mean","rawType":"float64","type":"float"},
{"name":"mode_mean","rawType":"float64","type":"float"},
{"name":"speechiness_mean","rawType":"float64","type":"float"},
{"name":"acousticness_mean","rawType":"float64","type":"float"},
{"name":"instrumentalness_mean","rawType":"float64","type":"float"},
{"name":"liveness_mean","rawType":"float64","type":"float"},
{"name":"valence_mean","rawType":"float64","type":"float"},
{"name":"tempo_mean","rawType":"float64","type":"float"},
{"name":"genre_acoustic","rawType":"float64","type":"float"},
{"name":"genre_alt-rock","rawType":"float64","type":"float"},
{"name":"genre_ambient","rawType":"float64","type":"float"},
{"name":"genre_black-metal","rawType":"float64","type":"float"},
```

```
{"name":"genre_blues","rawType":"float64","type":"float"},
{"name":"genre_breakbeat","rawType":"float64","type":"float"},
{"name":"genre_chill","rawType":"float64","type":"float"},
{"name":"genre_classical","rawType":"float64","type":"float"},
{"name":"genre_club","rawType":"float64","type":"float"},
{"name":"genre_comedy","rawType":"float64","type":"float"},
{"name":"genre_country","rawType":"float64","type":"float"},
{"name":"genre_dance","rawType":"float64","type":"float"},
{"name":"genre_dancehall","rawType":"float64","type":"float"},
{"name":"genre_death-metal","rawType":"float64","type":"float"},
{"name":"genre_deep-house","rawType":"float64","type":"float"},
{"name":"genre_disco","rawType":"float64","type":"float"},
{"name":"genre_drum-and-bass","rawType":"float64","type":"float"},
{"name":"genre_dub","rawType":"float64","type":"float"},
{"name":"genre_edm","rawType":"float64","type":"float"},
{"name":"genre_electro","rawType":"float64","type":"float"},
{"name":"genre_electronic","rawType":"float64","type":"float"},
{"name":"genre_emo","rawType":"float64","type":"float"},
{"name":"genre_folk","rawType":"float64","type":"float"},
{"name":"genre_french","rawType":"float64","type":"float"},
{"name":"genre_funk","rawType":"float64","type":"float"},
{"name":"genre_german","rawType":"float64","type":"float"},
{"name":"genre_gospel","rawType":"float64","type":"float"},
{"name":"genre_goth","rawType":"float64","type":"float"},
{"name":"genre_groove","rawType":"float64","type":"float"},
{"name":"genre_guitar","rawType":"float64","type":"float"},
{"name":"genre_hard-rock","rawType":"float64","type":"float"},
{"name":"genre_hardcore","rawType":"float64","type":"float"},
{"name":"genre_hardstyle","rawType":"float64","type":"float"},
{"name":"genre_hip hop","rawType":"float64","type":"float"},
{"name":"genre_hip-hop","rawType":"float64","type":"float"},
{"name":"genre_house","rawType":"float64","type":"float"},
{"name":"genre_indian","rawType":"float64","type":"float"},
{"name":"genre_indie-pop","rawType":"float64","type":"float"},
{"name":"genre_jazz","rawType":"float64","type":"float"},
{"name":"genre_k-pop","rawType":"float64","type":"float"},
{"name":"genre_metal","rawType":"float64","type":"float"},
{"name":"genre_metalcore","rawType":"float64","type":"float"},
{"name":"genre_minimal-techno","rawType":"float64","type":"float"},
{"name":"genre_new-age","rawType":"float64","type":"float"},
{"name":"genre_piano","rawType":"float64","type":"float"},
{"name":"genre_pop","rawType":"float64","type":"float"},
{"name":"genre_power-pop","rawType":"float64","type":"float"},
{"name":"genre_psych-rock","rawType":"float64","type":"float"},
{"name":"genre_punk","rawType":"float64","type":"float"},
{"name":"genre_punk-rock","rawType":"float64","type":"float"},
{"name":"genre_rock","rawType":"float64","type":"float"},
{"name":"genre_sad","rawType":"float64","type":"float"},
{"name":"genre_show-tunes","rawType":"float64","type":"float"},
```

{"name":"genre_singer-songwriter","rawType":"float64","type":"float"},
{"name":"genre_ska","rawType":"float64","type":"float"},
{"name":"genre_sleep","rawType":"float64","type":"float"},
{"name":"genre_songwriter","rawType":"float64","type":"float"},
{"name":"genre_soul","rawType":"float64","type":"float"},
{"name":"genre_swedish","rawType":"float64","type":"float"},
{"name":"genre_trance","rawType":"float64","type":"float"}],"ref":"3d6
e667a-922f-476d-bf43-67b59fba587e","rows":
[["7xkmRjcU7adwvRRtdHCLGr","0.6576534653465347","0.7050712871287128","
5.19009900990099","-
5.367219801980198","0.8118811881188119","0.06646900990099004","0.11001
159207920791","0.0010486187326732674","0.14315564356435642","0.5445136
633663367","122.94561188118811","0.0","0.0","0.0","0.0","0.0","0.0","0
.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0",
"0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0
","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0
.0","0.0","0.0","0.0","1.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0",
"0.0","0.0","0.0","0.0","0.0","0.0","0.0"],
["562675","0.5408947368421053","0.7542105263157894","4.842105263157895
","-
5.201605263157894","0.8421052631578947","0.03968157894736842","0.09546
763157894737","3.1323684210526316e-
06","0.16102894736842108","0.559921052631579","130.7342105263158","0.0
","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","1.0","0.0","0
.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0",
"0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0
","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0
.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0",
"0.0"],
["180017","0.5052222222222222","0.8579629629629629","5.555555555555555
","-
4.22","0.5925925925925926","0.10706666666666667","0.03374571111111111"
,"0.00818208888888889","0.23835555555555554","0.48007037037037037","12
7.16503703703704","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0
","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.11111
11111111111","0.0","0.037037037037037035","0.0","0.0","0.0","0.0","0.0
","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0
.0","0.0","0.5925925925925926","0.0","0.0","0.0","0.0","0.0","0.0","0.
0","0.0","0.0","0.2222222222222222","0.0","0.0","0.0","0.0","0.0","0.0
","0.0","0.0","0.037037037037037035"],
["901647","0.5531150442477877","0.8800265486725664","5.451327433628318
","-
3.605141592920354","0.7610619469026548","0.08846017699115044","0.05746
4477876106194","0.003889635663716814","0.22550619469026548","0.6968451
327433628","123.4163185840708","0.0","0.0","0.0","0.0","0.0","0.0","0.
0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","
0.0","0.0","0.0","0.07964601769911504","0.0","0.0","0.0","0.0","0.0","
0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0"
,"0.0","0.0","0.0","0.0","0.0","0.0","0.008849557522123894","0.0","0.0

```
","0.37168141592920356","0.0","0.0","0.0","0.0","0.0","0.5398230088495
575","0.0","0.0","0.0","0.0","0.0"],
["337283","0.5874615384615385","0.8193076923076923","5.769230769230769
","-
4.785461538461538","0.3076923076923077","0.044461538461538455","0.1596
8046153846155","5.661538461538461e-
07","0.4887692307692308","0.5184615384615385","116.62123076923078","0.
0","0.38461538461538464","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.
0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.6153846153
846154","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0
.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0",
"0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0","0.0
","0.0","0.0","0.0","0.0","0.0","0.0"]],"shape":
{"columns":71,"rows":5}}
```


2. Clustering

2.1 Audio based clustering

Clustering based on audio features will be done in parallel with both the imputed and the dropped NAs version of the data sets. At the end results will be evaluated based on which data set performs better and yields more reasonable results.

Preparation

Load data:

```
df_imp_all = pd.read_csv("data/audio_imputed_aggr.csv")
df_drop_all = pd.read_csv("data/audio_dropped_aggr.csv")

#remove columns with genre - need only audio features for this
clustering
genre_cols = [col for col in df_imp_all.columns if
col.startswith("genre_")]
df_imp = df_imp_all.drop(columns=genre_cols)
df_drop = df_drop_all.drop(columns=genre_cols)


df_imp.sample(n=5, random_state=SEED)
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"playlist_id","rawType":"object","type":"string"},
{"name":"danceability_mean","rawType":"float64","type":"float"},
{"name":"energy_mean","rawType":"float64","type":"float"},
{"name":"key_mean","rawType":"float64","type":"float"},
{"name":"loudness_mean","rawType":"float64","type":"float"},
{"name":"mode_mean","rawType":"float64","type":"float"},
{"name":"speechiness_mean","rawType":"float64","type":"float"},
{"name":"acousticness_mean","rawType":"float64","type":"float"},
```

{"name":"instrumentalness_mean","rawType":"float64","type":"float"},
{"name":"liveness_mean","rawType":"float64","type":"float"},
{"name":"valence_mean","rawType":"float64","type":"float"},
{"name":"tempo_mean","rawType":"float64","type":"float"}],"ref":"ac194
03a-3d0e-406e-ba38-c8c482c0303e","rows":
[["803","7xkmRjcU7adwvRRtdHCLGr","0.6576534653465347","0.7050712871287
128","5.19009900990099","-
5.367219801980198","0.8118811881188119","0.06646900990099","0.11001159
20792079","0.0010486187326732","0.1431556435643564","0.544513663366336
7","122.94561188118811"],
["523","562675","0.5408947368421053","0.7542105263157894","4.842105263
157895","-
5.201605263157894","0.8421052631578947","0.0396815789473684","0.095467
6315789473","3.132368421052632e-
06","0.161028947368421","0.559921052631579","130.7342105263158"],
["89","180017","0.5052222222222222","0.8579629629629629","5.5555555555
55555","-
4.22","0.5925925925925926","0.1070666666666666","0.0337457111111111","
0.0081820888888888","0.2383555555555555","0.4800703703703703","127.165
03703703704"],
["916","901647","0.5531150442477877","0.8800265486725664","5.451327433
628318","-
3.605141592920354","0.7610619469026548","0.0884601769911504","0.057464
4778761061","0.0038896356637168","0.2255061946902654","0.6968451327433
628","123.4163185840708"],
["270","337283","0.5874615384615385","0.8193076923076923","5.769230769
230769","-
4.785461538461538","0.3076923076923077","0.0444615384615384","0.159680
4615384615","5.661538461538461e-
07","0.4887692307692308","0.5184615384615385","116.62123076923078"]],"
shape":{"columns":12,"rows":5}}

Scaling:

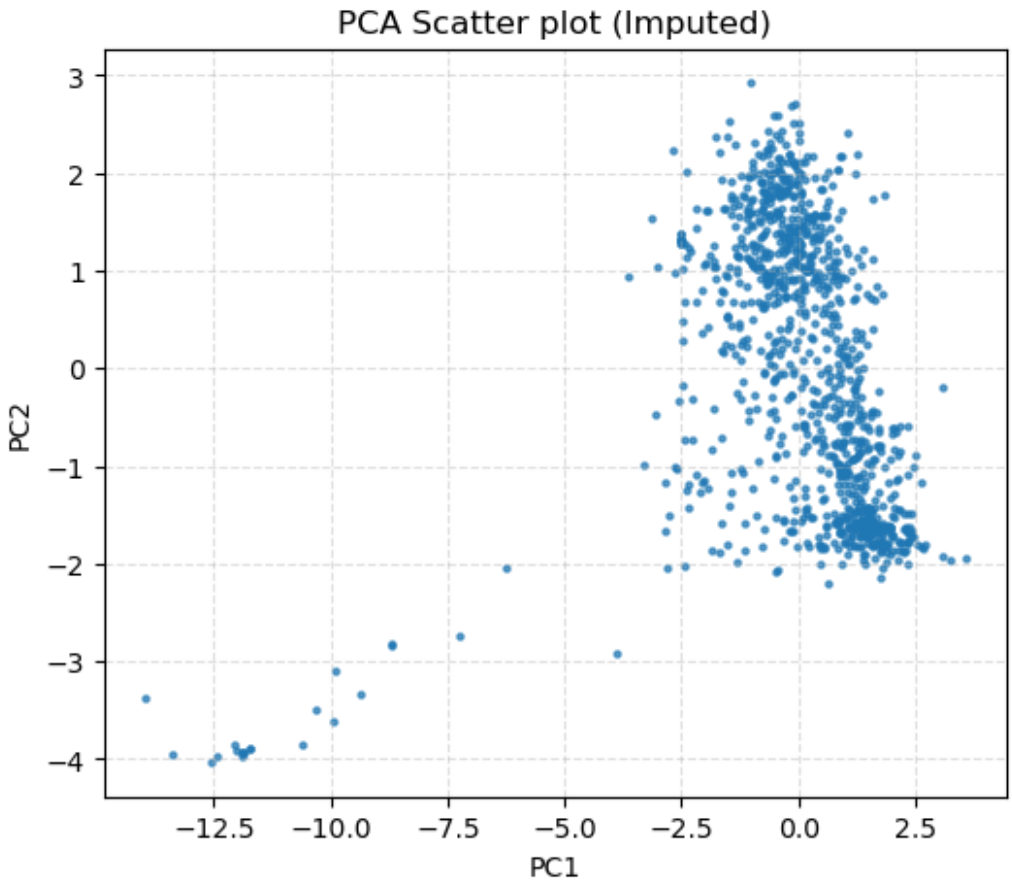
```
def scale_features(df, feature_cols):

    scaler = StandardScaler()
    scaled_array = scaler.fit_transform(df[feature_cols])

    #array -> df
    df_scaled = pd.DataFrame(scaled_array,
                             columns=feature_cols,
                             index=df.index)

    #make a scaled full df
    df_scaled_full = pd.concat([df.drop(columns=feature_cols),
df_scaled], axis=1)

    return df_scaled_full
```

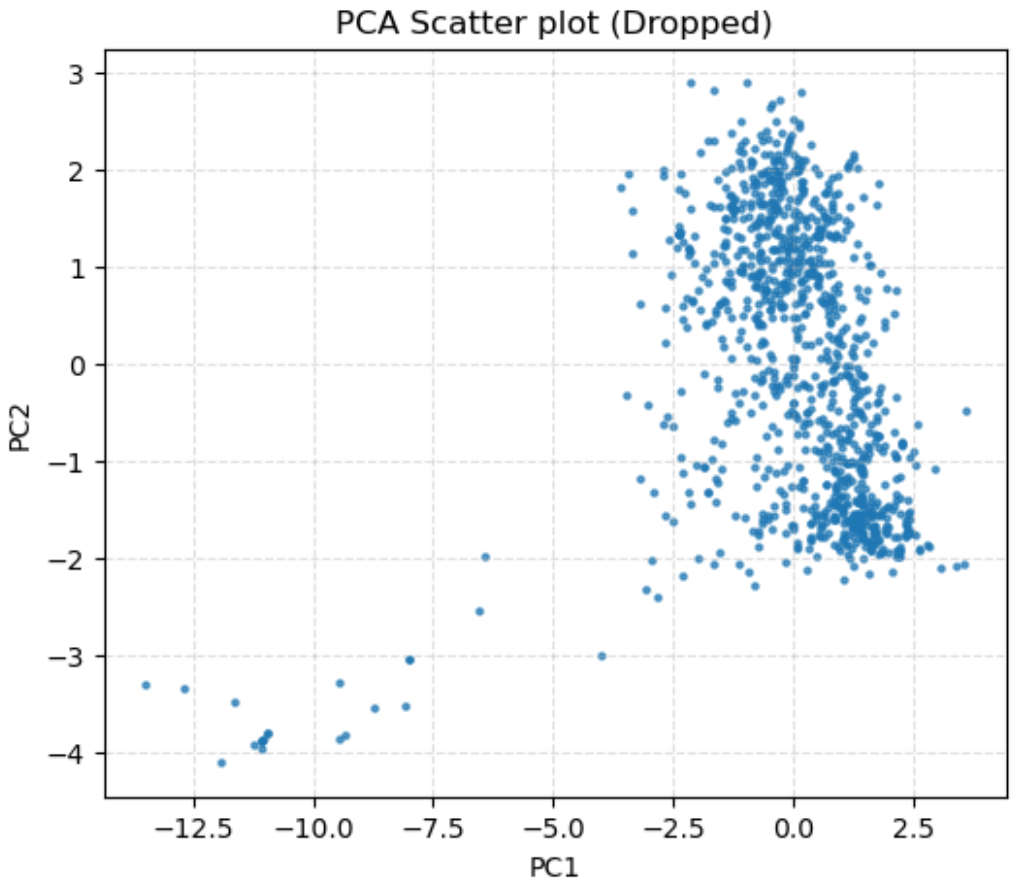
```
cols = df_drop.columns[1:].to_list()
df_imp_scaled = scale_features(df_imp, cols)
df_drop_scaled = scale_features(df_drop, cols)

df_imp_scaled.sample(n=5, random_state=SEED)
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"playlist_id","rawType":"object","type":"string"},
{"name":"danceability_mean","rawType":"float64","type":"float"},
{"name":"energy_mean","rawType":"float64","type":"float"},
{"name":"key_mean","rawType":"float64","type":"float"},
{"name":"loudness_mean","rawType":"float64","type":"float"},
{"name":"mode_mean","rawType":"float64","type":"float"},
{"name":"speechiness_mean","rawType":"float64","type":"float"},
{"name":"acousticness_mean","rawType":"float64","type":"float"},
{"name":"instrumentalness_mean","rawType":"float64","type":"float"},
{"name":"liveness_mean","rawType":"float64","type":"float"},
{"name":"valence_mean","rawType":"float64","type":"float"},
{"name":"tempo_mean","rawType":"float64","type":"float"}],"ref":"72a29
ad4-2715-4aae-b8dd-e660eb984ace","rows":
[["803","7xkmRjcU7adwvRRtdHCLGr","0.32372534677879644","0.292181940438
16054","-
0.041233482017281865","0.4917254185103496","0.5283411613179622","-
0.6919703942380506","-0.44220564304905596","-0.21860853991692406","-
0.7435472750314797","0.7943817084750829","-0.17650793136885706"],
["523","562675","-0.7506784031254132","0.6830157591562412","-
0.4214849026540583","0.560708442854473","0.6962227967353025","-
1.0385839943973685","-0.5518047985739505","-0.22692878635831662","-
0.40733365621969947","0.9311224227172104","0.5839599493248546"],
["89","180017","-
1.078933806100799","1.5082210673162624","0.35809947142440085","0.96957
41151937509","-0.689711930388186","-0.16666059525260532","-
1.0169236608349106","-
0.161838569775335","1.0472524155016802","0.2224469205183555","0.235470
84070735705"],["916","901647","-
0.6382281883121436","1.6837059873140843","0.24420980586557162","1.2256
79604273264","0.24606230530650353","-0.4074176273070706","-
0.8381857711460045","-
0.19599900447132299","0.8055438783841495","2.146324829115107","-
0.13054878710181436"],["270","337283","-
0.3221747107520725","1.2007725334416546","0.5915815506723633","0.73404
37846397975","-2.2722095154450535","-0.9767341298586323","-
0.06791514985685557","-
0.22694920894670678","5.757768661770665","0.5631688724988001","-
0.7940116595803772"]],"shape":{"columns":12,"rows":5}}
```


PCA and plot check before clustering:

```
#PCA reduction
def apply_pca(X, n_components):
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X)
    X_pca_df = pd.DataFrame(X_pca, index=X.index, columns=[f"PC{i+1}"
for i in range(X_pca.shape[1])])
    #return data frame and pca model
    return X_pca_df, pca

#PCA variance plot before clustering
def pca_plot(X_pca_df, tip, labels=None):
    plt.figure(figsize=(6, 5))

    if labels is None:
        plt.scatter(X_pca_df['PC1'], X_pca_df['PC2'], s=5, alpha=0.7)
    else:
        #encode labels
        le = LabelEncoder()
        labels_enc = le.fit_transform(labels)
        plt.scatter(
            X_pca_df['PC1'], X_pca_df['PC2'],
            c=labels_enc, cmap='viridis', s=5, alpha=0.8
        )

    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.title(f"PCA Scatter plot ({tip})")
    plt.grid(True, linestyle="--", alpha=0.4)
    plt.show()

#imputed
X_imp = df_imp_scaled.drop(columns=['playlist_id'])
X_pca_imp, _ = apply_pca(X_imp, n_components=3)
pca_plot(X_pca_imp, tip="Imputed")

#dropped
X_drop = df_drop_scaled.drop(columns=['playlist_id'])
X_pca_drop, _ = apply_pca(X_drop, n_components=3)
pca_plot(X_pca_drop, tip="Dropped")
```


PCA Scatter plot (Imputed)
PC2
PC1
3
2
1
0
−1
−2
−3
−4
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5



Clustering 1: K-means and PCA -> K-means

```
#KMEANS
def run_kmeans(X, k):
    kmeans = KMeans(n_clusters=k, random_state=SEED)
    labels = kmeans.fit_predict(X)
    sil = silhouette_score(X, labels)
    db = davies_bouldin_score(X, labels)
    return labels, sil, db, kmeans

#score plot
def plot_scores(results_dict, metric_name="Silhouette Score"):
    ks = list(results_dict.keys())
    scores = [results_dict[k] for k in ks]

    plt.figure(figsize=(6, 5))
    plt.plot(ks, scores, marker='o')
    plt.xlabel("k")
    plt.ylabel(metric_name)
    plt.title(f"{metric_name} vs k")
    plt.grid(True, linestyle="--", alpha=0.4)
    plt.show()
```

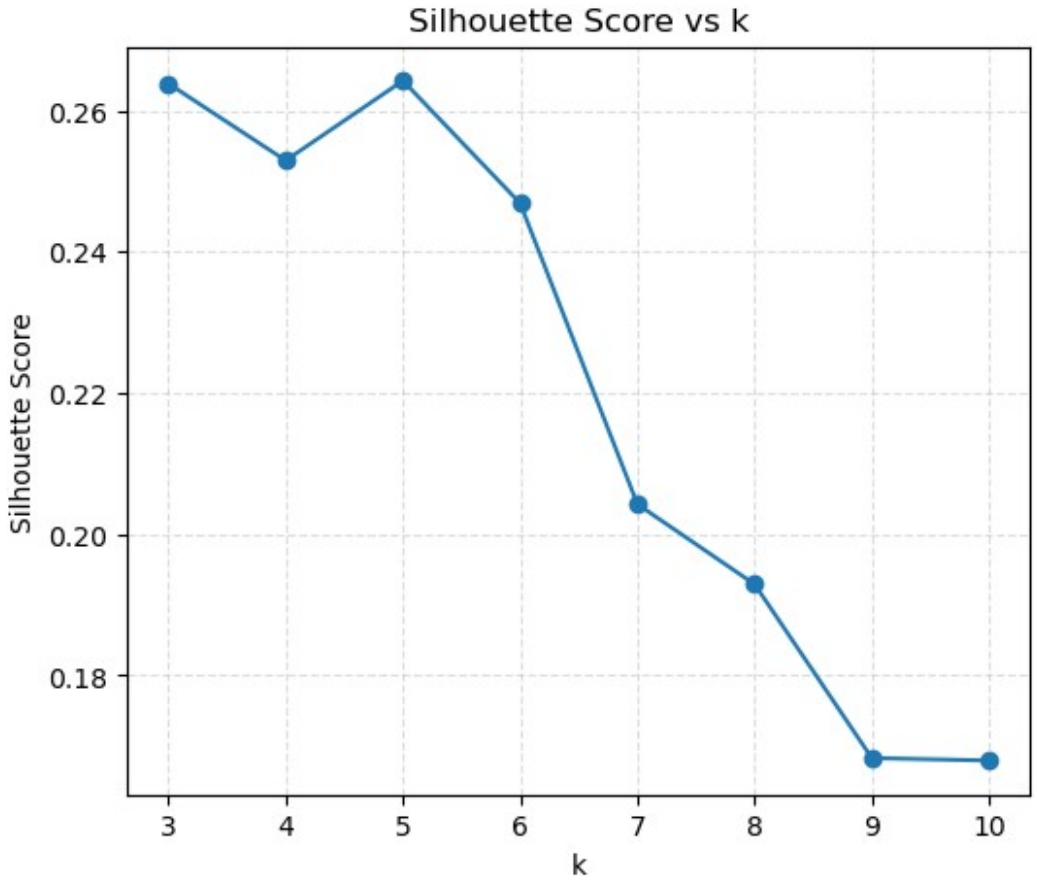
```
#centroid heatmap
def plot_centroid_heatmap(centroids, title="Cluster Centroid
Heatmap"):
    plt.figure(figsize=(8, 4))
    sns.heatmap(centroids, annot=False, cmap="viridis")
    plt.title(title)
    plt.show()

#cluster summary by genres
def genre_cluster_summary(df_all, labels):

    #select genre columns
    genre_cols = [c for c in df_all.columns if c.startswith("genre_")]

    #copy and add cluster labels
    df_tmp = df_all.copy()
    df_tmp['cluster'] = labels

    #mean per cluster
    cluster_means = df_tmp.groupby('cluster')[genre_cols].mean()

    #remove 'genre_' prefix
    cluster_means.columns = [c.replace('genre_', '') for c in
cluster_means.columns]

    #keep only top 5 genres per cluster
    top5_dict = {}
    for cluster_id, row in cluster_means.iterrows():
        row_clean = row.dropna()
        top5 = row_clean.nlargest(10)
        top5_dict[cluster_id] = list(top5.items())

    return top5_dict


#pipeline
def kmeans_pipeline(X, df_all, k_range=range(3, 11),
drop_cols=['playlist_id'], plot_pca=True, pca_data=False):

    # run k-means for multiple k
    results_sil = {}
    results_db = {}
    models = {}
    labels_dict = {}

    for k in k_range:
        labels, sil, db, model = run_kmeans(X, k)
        print(f"K:{k}, SIL:{sil:.3f}, DB:{db:.3f}")
        results_sil[k] = sil
        results_db[k] = db
```

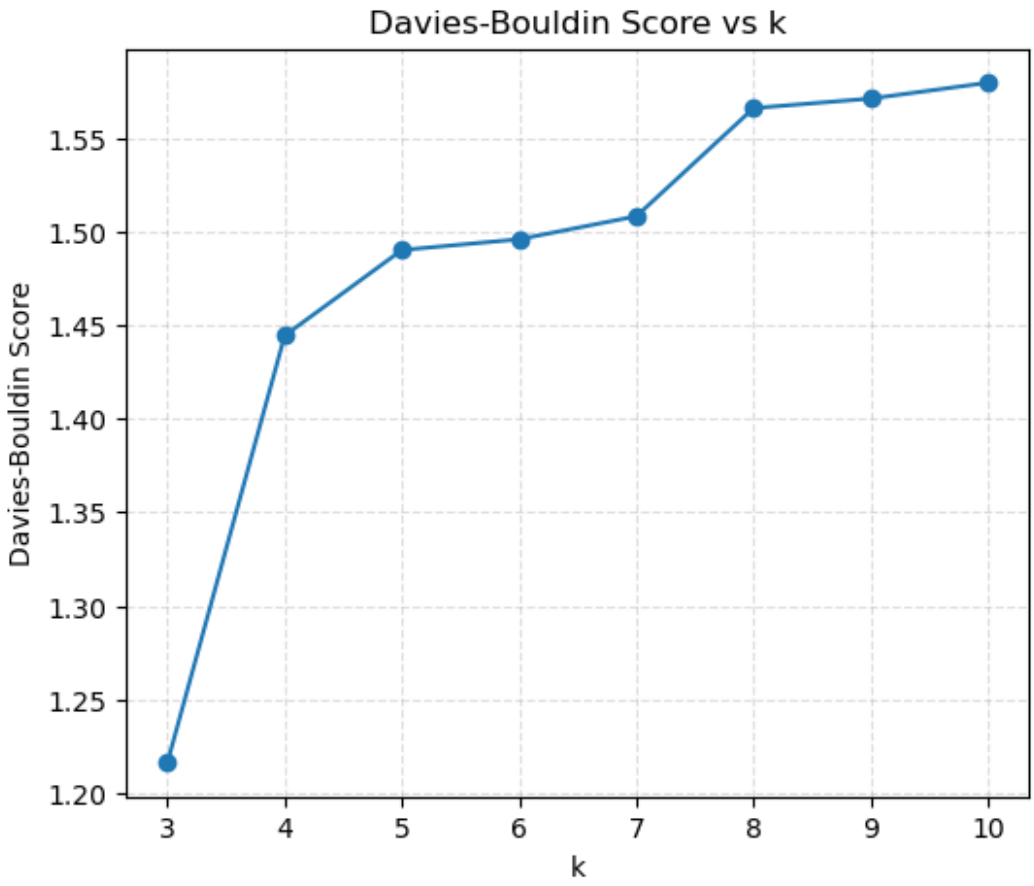
```
        models[k] = model
        labels_dict[k] = labels

    # plot silhouette and DB scores
    plot_scores(results_sil, metric_name="Silhouette Score")
    plot_scores(results_db, metric_name="Davies-Bouldin Score")

    # pick best k based on silhouette
    best_k = max(results_sil, key=results_sil.get)
    best_labels = labels_dict[best_k]
    best_model = models[best_k]
    print(f"Best k based on silhouette: {best_k}")

    # PCA for plotting
    if plot_pca:
        X_pca_df, _ = apply_pca(X, n_components=2)
        pca_plot(X_pca_df, tip=f"k={best_k}", labels=best_labels)

    #centroids
    if pca_data == False:
        centroids = best_model.cluster_centers_
        centroids_df = pd.DataFrame(centroids, columns=[c for c in
X.columns if c != 'playlist_id'])
        plot_centroid_heatmap(centroids_df)
        print(centroids_df)

    #genre summary
    genre_summary = genre_cluster_summary(df_all, best_labels)
    for c, it in genre_summary.items():
        print(f"Cluster {c}:")
        for x in it:
            print(f"{x[0]}, {x[1]:.3f}")
        print()

    # return results
    return {
        "best_k": best_k,
        "best_labels": best_labels,
        "best_model": best_model,
        "silhouette_scores": results_sil,
        "db_scores": results_db,
        "genre_summary": genre_summary
    }
```

K-means on original feature space:

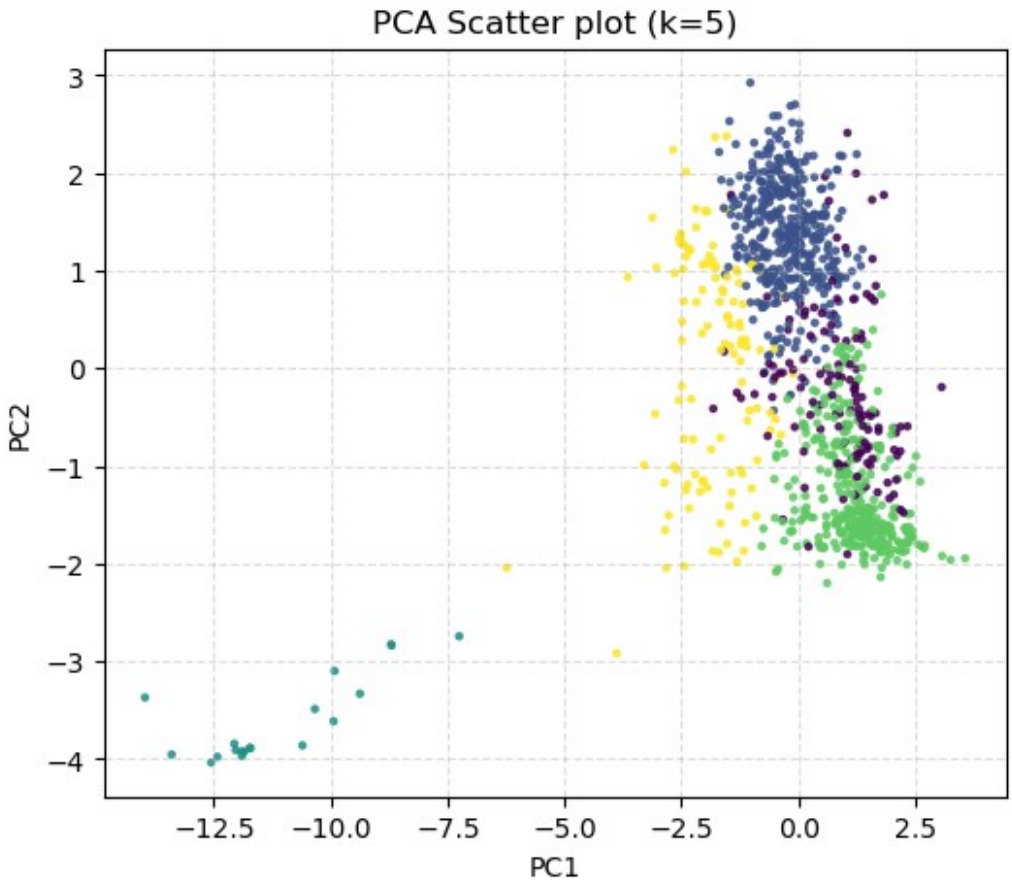
```
#imputed
results_imp = kmeans_pipeline(X_imp, df_imp_all)
```

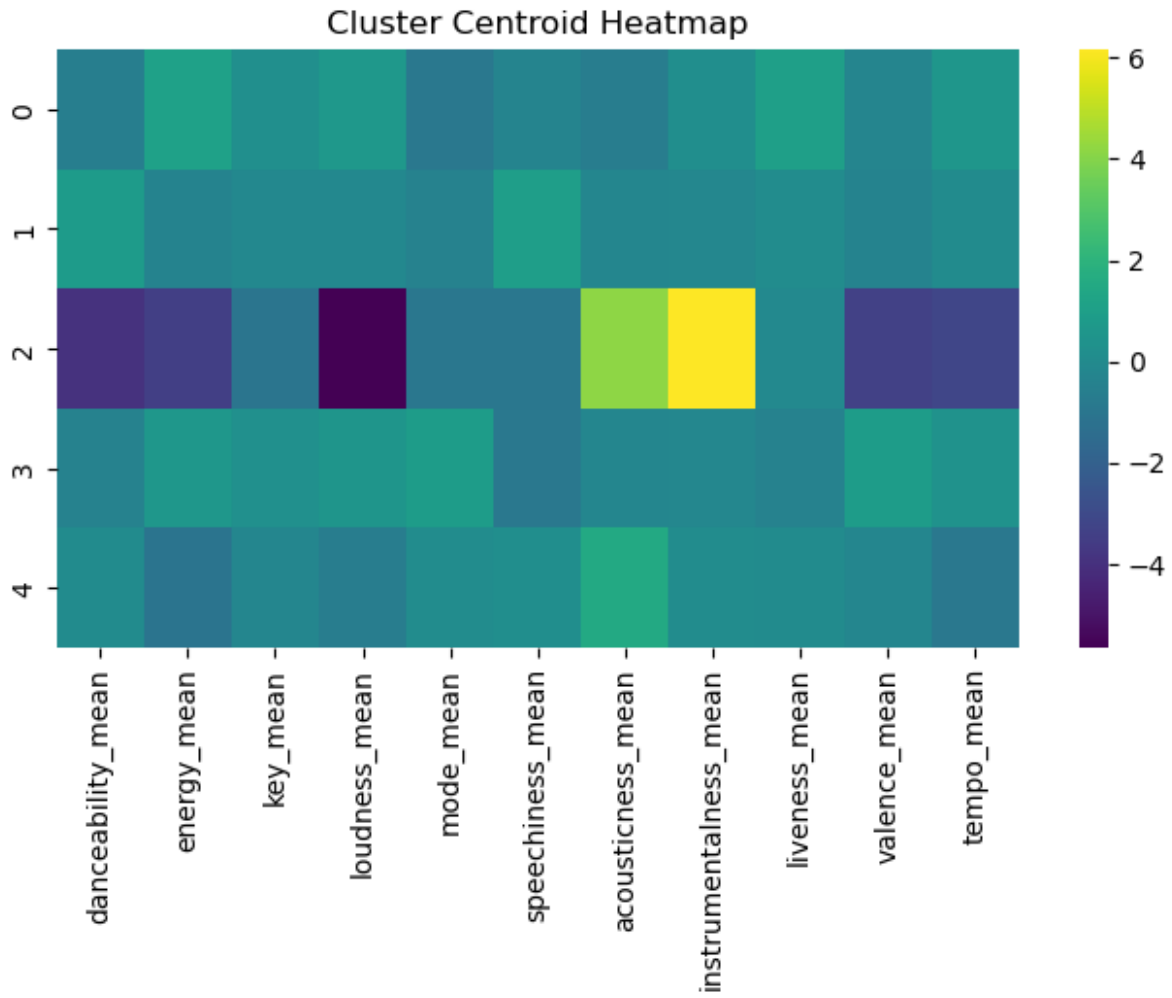
```
c:\Users\lunka\miniforge3\Lib\site-packages\threadpoolctl.py:1226:
RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see

https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)

K:3, SIL:0.264, DB:1.216
K:4, SIL:0.253, DB:1.444
K:5, SIL:0.264, DB:1.490
K:6, SIL:0.247, DB:1.496
K:7, SIL:0.204, DB:1.508
K:8, SIL:0.193, DB:1.566
K:9, SIL:0.168, DB:1.571
K:10, SIL:0.168, DB:1.580
```






```
Best k based on silhouette: 5
```


PCA Scatter plot (k=5)
PC2
PC1
3
2
1
0
−1
−2
−3
−4
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5

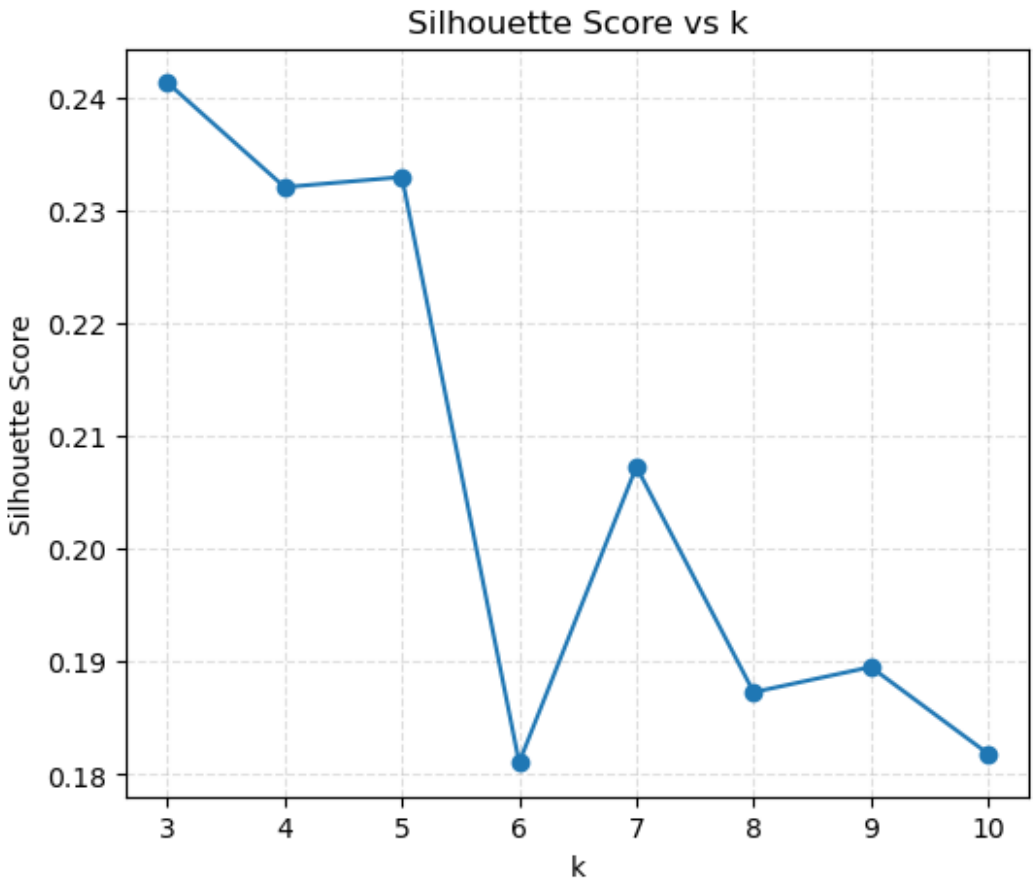


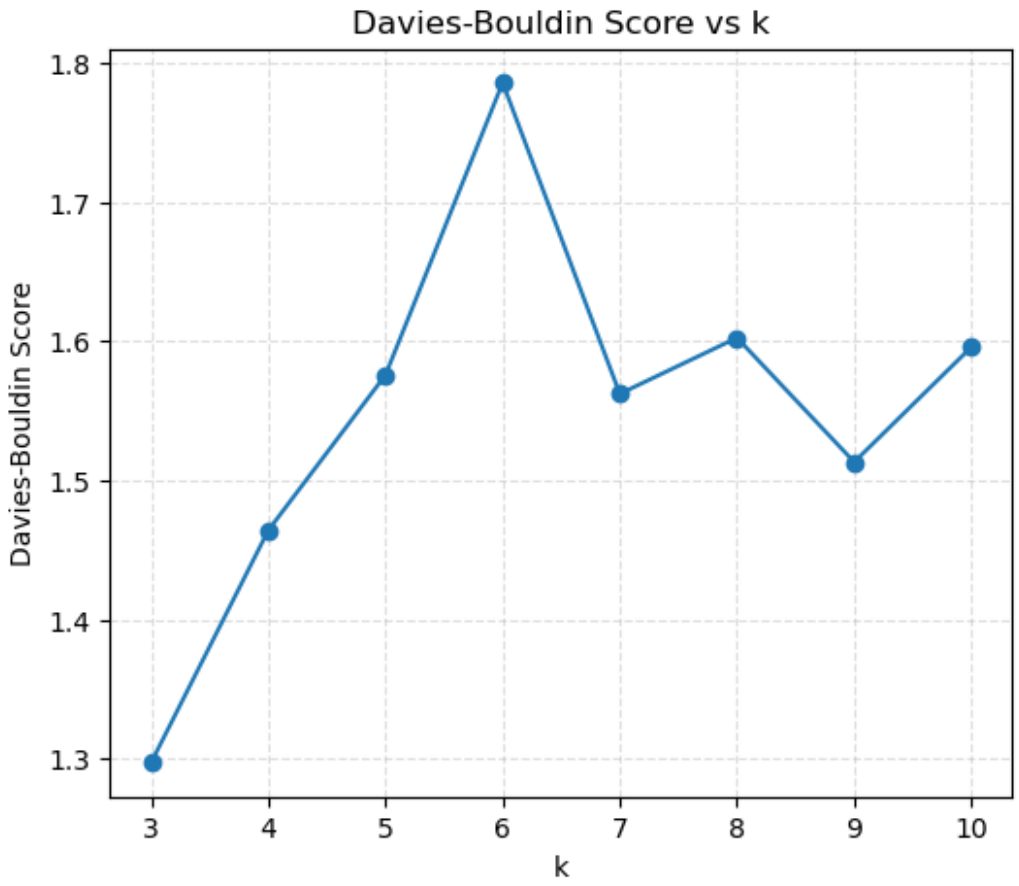
	danceability_mean	energy_mean	key_mean	loudness_mean	mode_mean
0	-0.665609	1.070173	0.175783	0.644122	-0.936813
1	0.786210	-0.354082	-0.134564	-0.149978	-0.407826
2	-3.958971	-3.475843	-1.076082	-5.664986	-0.982566
3	-0.410617	0.581414	0.221868	0.491259	0.848833
4	0.016095	-1.129341	-0.220042	-0.707608	0.057279

	speechiness_mean	acousticness_mean	instrumentalness_mean	liveness_mean
0	-0.326844	-0.685030	0.150474	0.958175
1	0.924393	-0.227237	-0.198783	0.030198

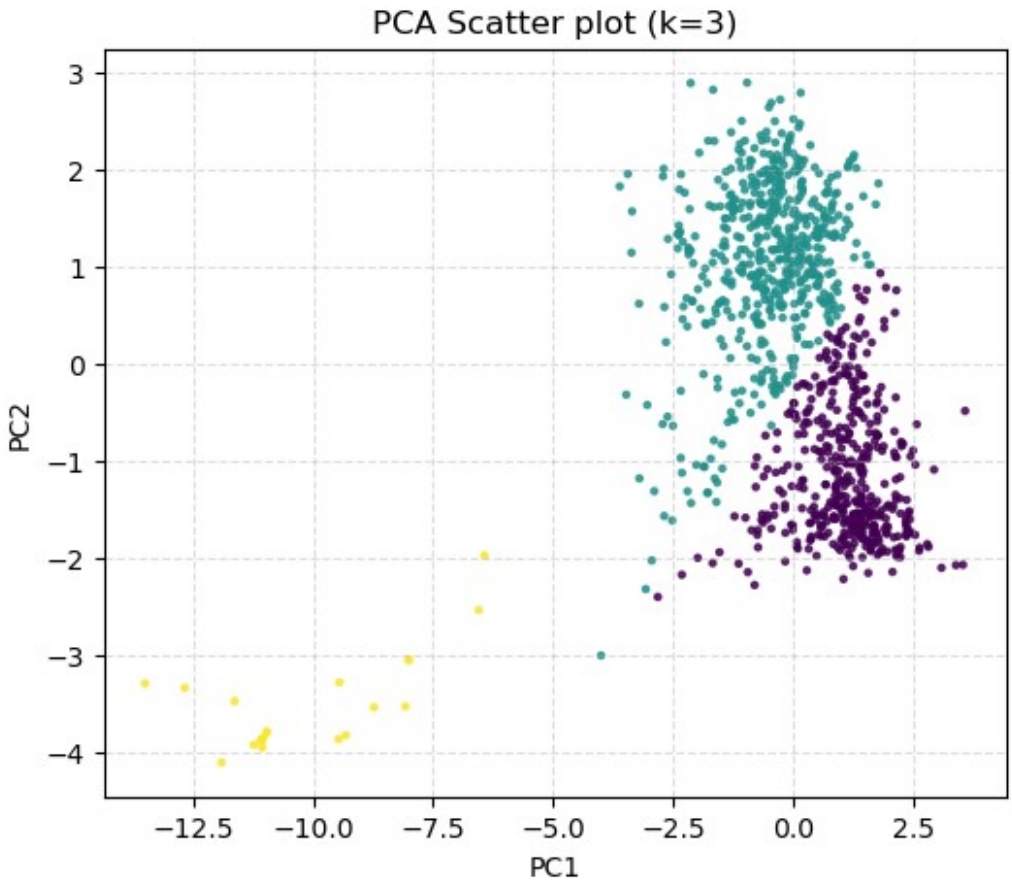
```
2         -0.977203           4.179093                6.175474      -
0.082315
3         -0.904290          -0.245405               -0.205941      -
0.400945
4          0.134111           1.502651                0.062945
0.022976

   valence_mean  tempo_mean
0     -0.299527    0.539597
1     -0.383487   -0.005781
2     -3.370460   -3.179987
3      0.813344    0.308674
4     -0.225232   -0.938040
Cluster 0:
metal, 0.158
dance, 0.148
electro, 0.115
hip-hop, 0.094
alt-rock, 0.088
rock, 0.087
emo, 0.073
gospel, 0.035
hardcore, 0.035
edm, 0.034

Cluster 1:
hip-hop, 0.771
dance, 0.088
pop, 0.032
hip hop, 0.028
emo, 0.016
electro, 0.011
house, 0.009
rock, 0.008
country, 0.008
indie-pop, 0.006

Cluster 2:
classical, 0.446
sleep, 0.246
german, 0.131
new-age, 0.097
ambient, 0.033
rock, 0.020
alt-rock, 0.017
singer-songwriter, 0.007
metal, 0.004
acoustic, 0.000

Cluster 3:
```

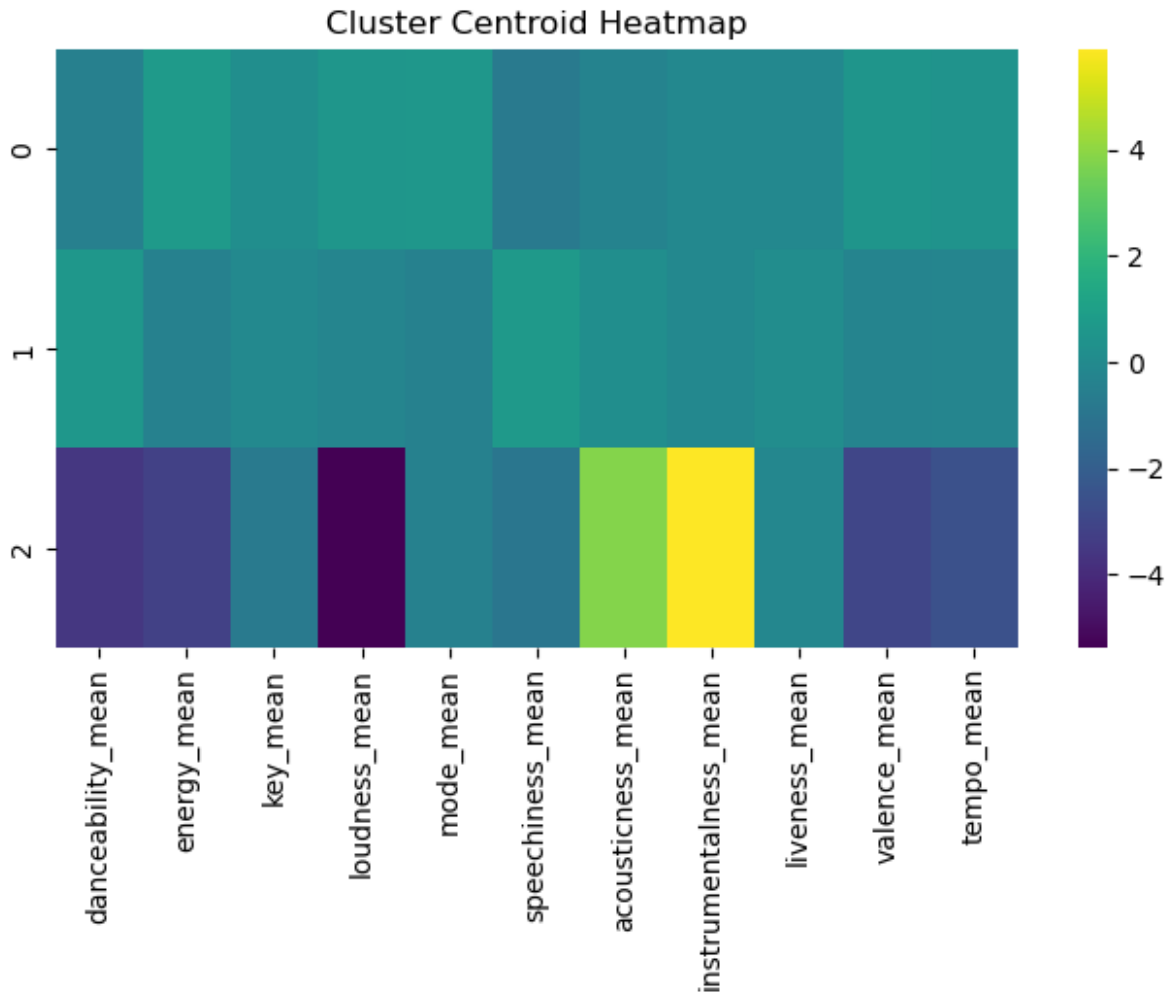
```
country, 0.591
pop, 0.108
rock, 0.089
alt-rock, 0.065
dance, 0.063
hip-hop, 0.017
k-pop, 0.014
electro, 0.009
indie-pop, 0.008
blues, 0.005

Cluster 4:
hip-hop, 0.240
pop, 0.191
show-tunes, 0.120
singer-songwriter, 0.060
dance, 0.048
jazz, 0.048
folk, 0.038
acoustic, 0.037
alt-rock, 0.033
electro, 0.031


#dropped
results_drop = kmeans_pipeline(X_drop, df_drop_all)

K:3, SIL:0.241, DB:1.297
K:4, SIL:0.232, DB:1.464
K:5, SIL:0.233, DB:1.576
K:6, SIL:0.181, DB:1.786
K:7, SIL:0.207, DB:1.562
K:8, SIL:0.187, DB:1.602
K:9, SIL:0.190, DB:1.513
K:10, SIL:0.182, DB:1.596
```


Silhouette Score vs k
Silhouette Score
0.24
0.23
0.22
0.21
0.20
0.19
0.18
3
4
5
6
7
8
9
10
k




```
Best k based on silhouette: 3
```


PCA Scatter plot (k=3)
PC2
PC1
3
2
1
0
−1
−2
−3
−4
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5



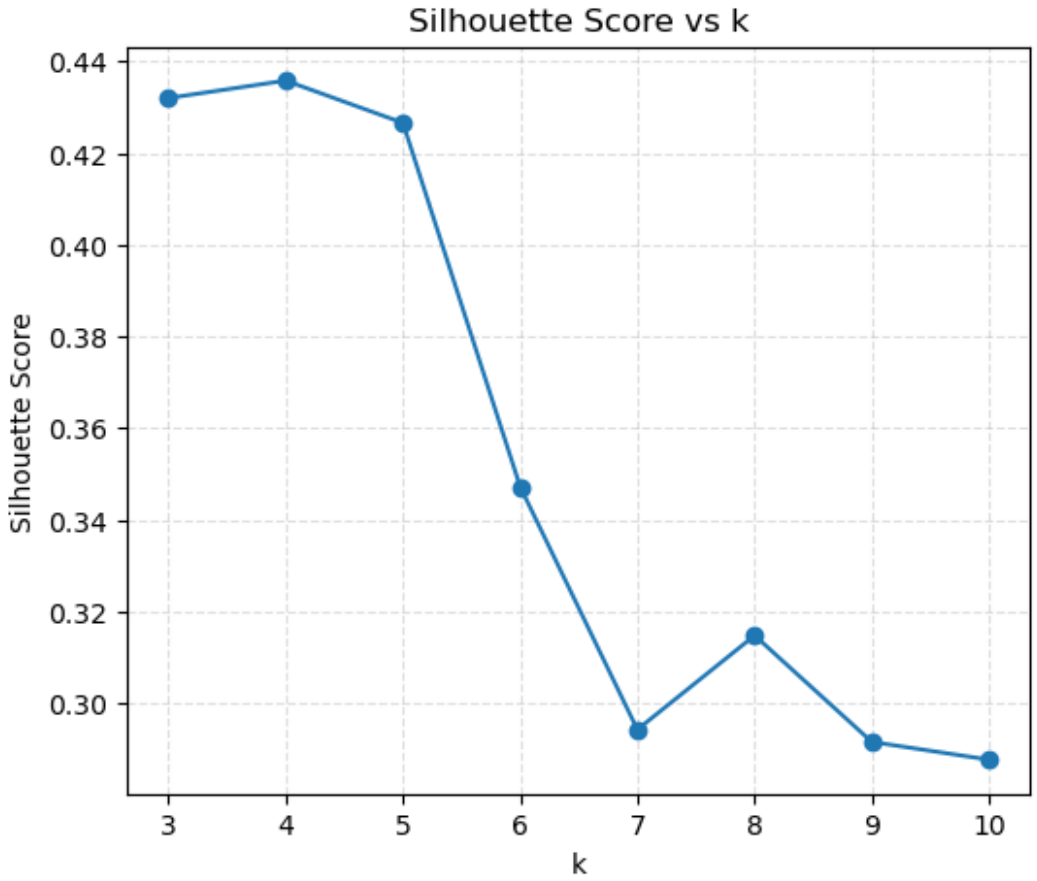

```
   danceability_mean  energy_mean  key_mean  loudness_mean  mode_mean
\
0          -0.517249     0.695809  0.124907       0.527721   0.577568

1           0.569692    -0.458163 -0.074213      -0.233962  -0.463298

2          -3.609931    -3.186254 -0.781611      -5.378923  -0.479119


   speechiness_mean  acousticness_mean  instrumentalness_mean
liveness_mean  \
0         -0.781159          -0.331100              -0.104999      -
0.106601
1          0.688440           0.131250              -0.138054
0.094356
2         -0.965492           3.780376               5.887523      -
0.142407

   valence_mean  tempo_mean
```

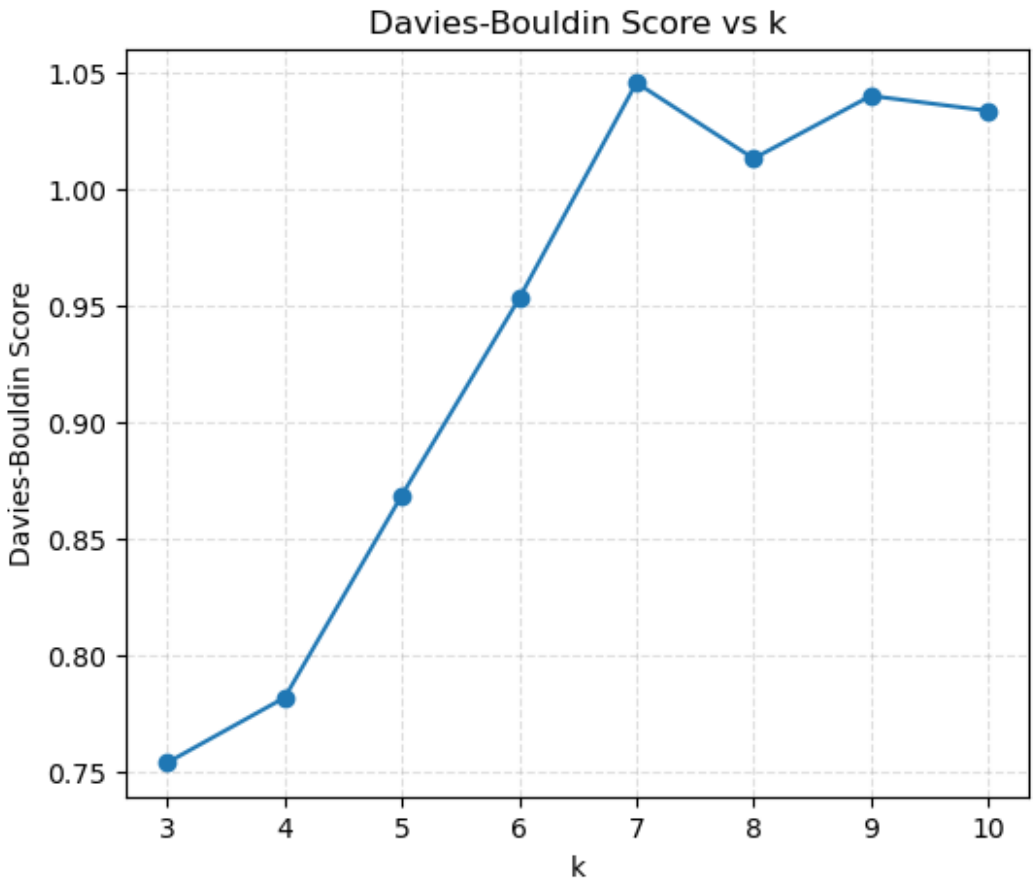
```
0      0.516700    0.406602
1     -0.315582   -0.239253
2     -3.009195   -2.605054
Cluster 0:
country, 0.451
rock, 0.088
pop, 0.085
dance, 0.083
alt-rock, 0.074
metal, 0.047
hip-hop, 0.028
emo, 0.022
electro, 0.018
k-pop, 0.010

Cluster 1:
hip-hop, 0.605
dance, 0.088
pop, 0.078
electro, 0.034
show-tunes, 0.026
hip hop, 0.020
emo, 0.018
rock, 0.015
country, 0.015
singer-songwriter, 0.012

Cluster 2:
classical, 0.425
sleep, 0.233
german, 0.121
new-age, 0.091
folk, 0.046
ambient, 0.028
rock, 0.024
alt-rock, 0.019
singer-songwriter, 0.007
metal, 0.005
```

K-means on PCA-reduced space:

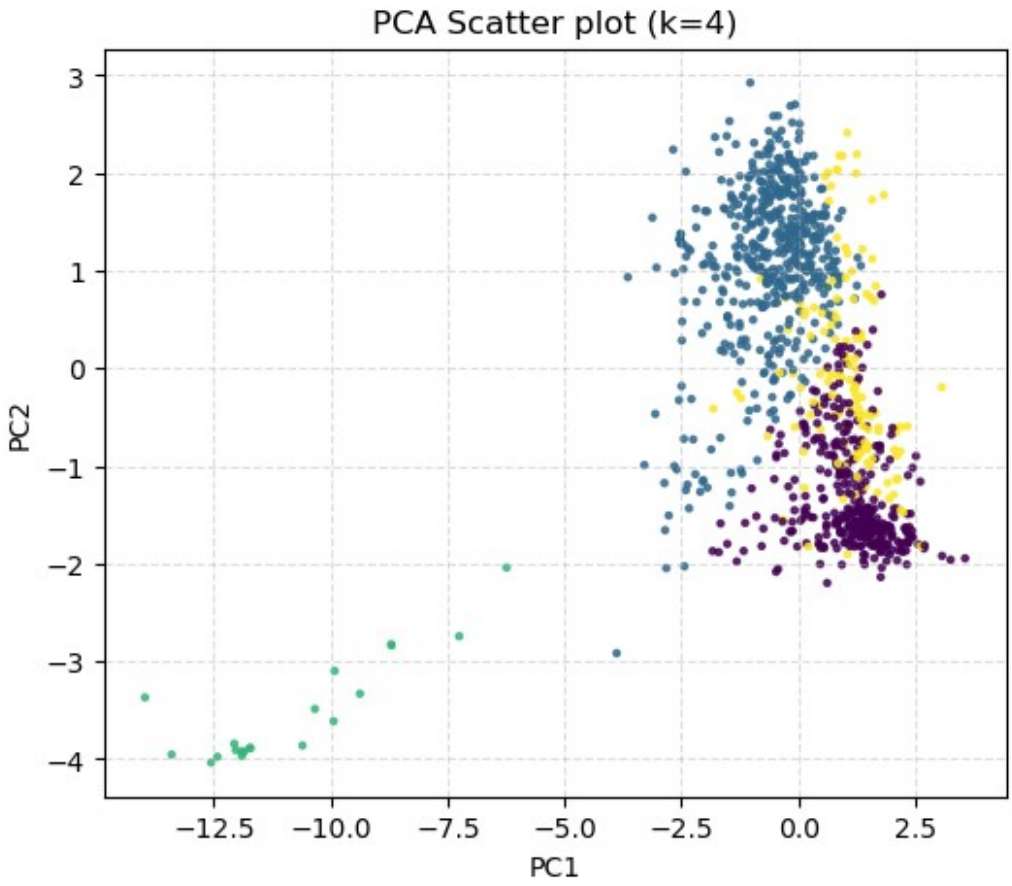
```
#imputed
results_imp_pca = kmeans_pipeline(X_pca_imp, df_imp_all,
pca_data=True)

K:3, SIL:0.432, DB:0.754
K:4, SIL:0.436, DB:0.782
K:5, SIL:0.427, DB:0.869
```

```
K:6, SIL:0.347, DB:0.953
K:7, SIL:0.294, DB:1.046
K:8, SIL:0.315, DB:1.013
K:9, SIL:0.292, DB:1.040
K:10, SIL:0.288, DB:1.033
```



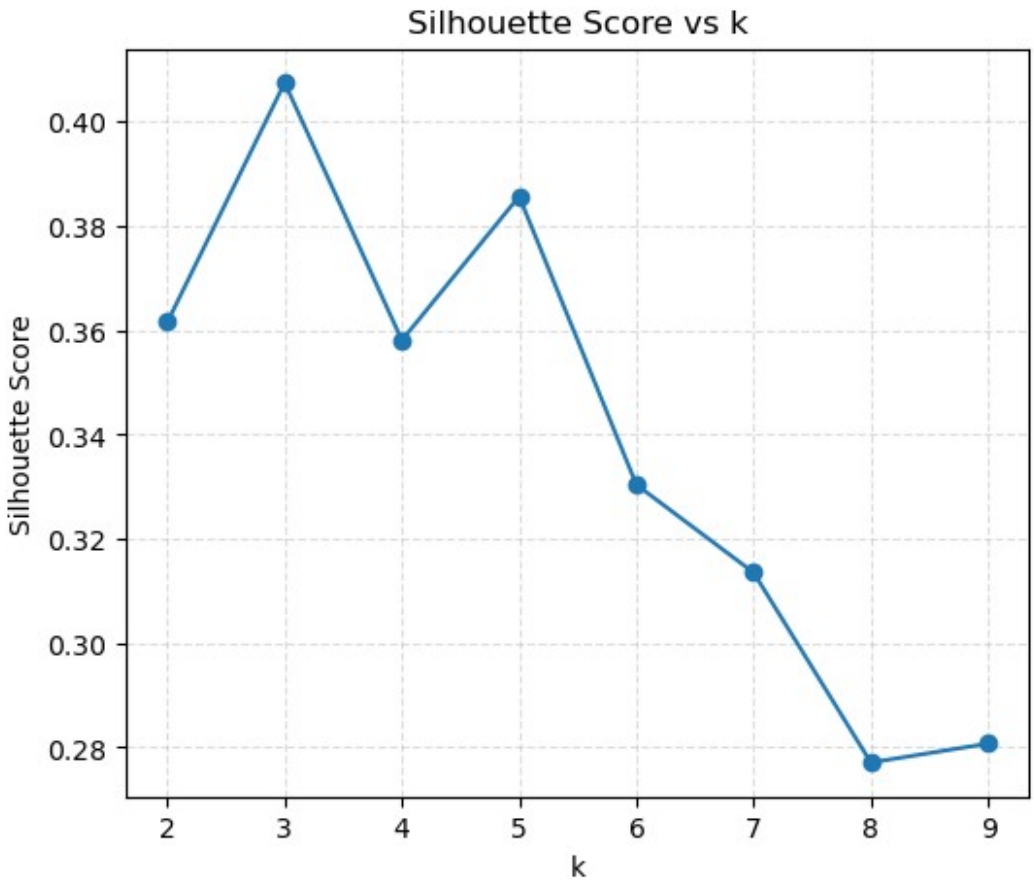


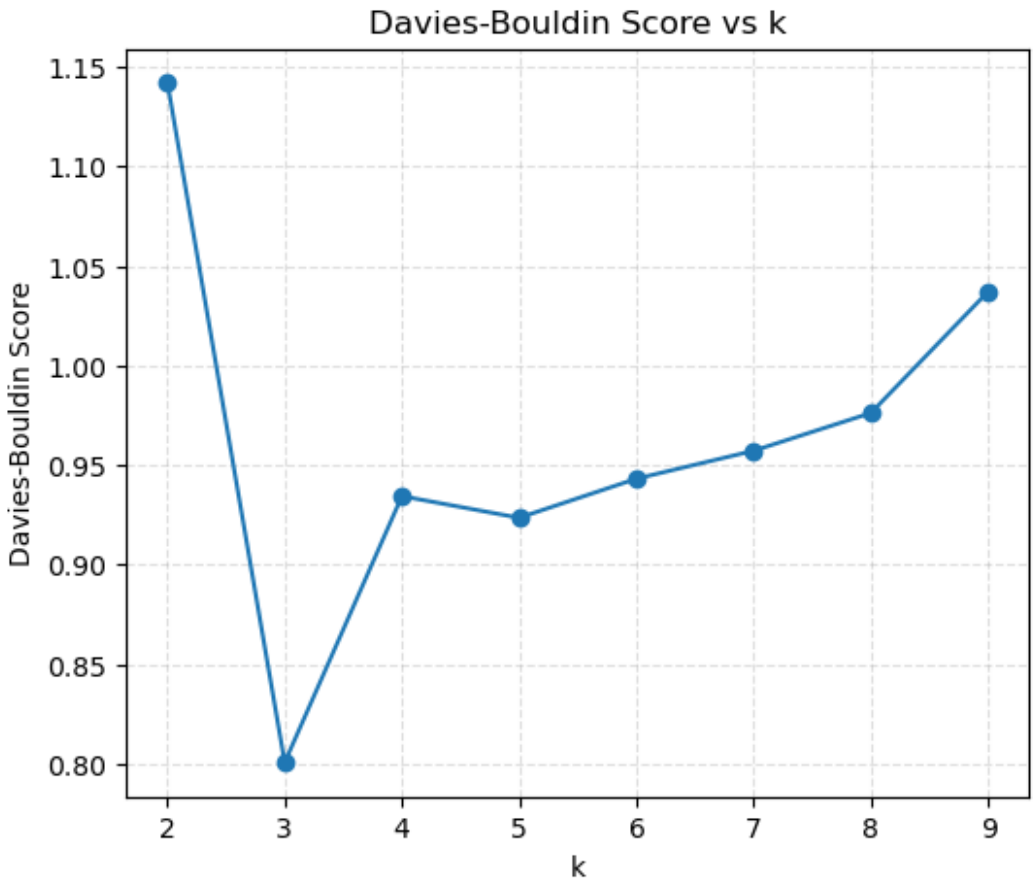

```
Best k based on silhouette: 4
```



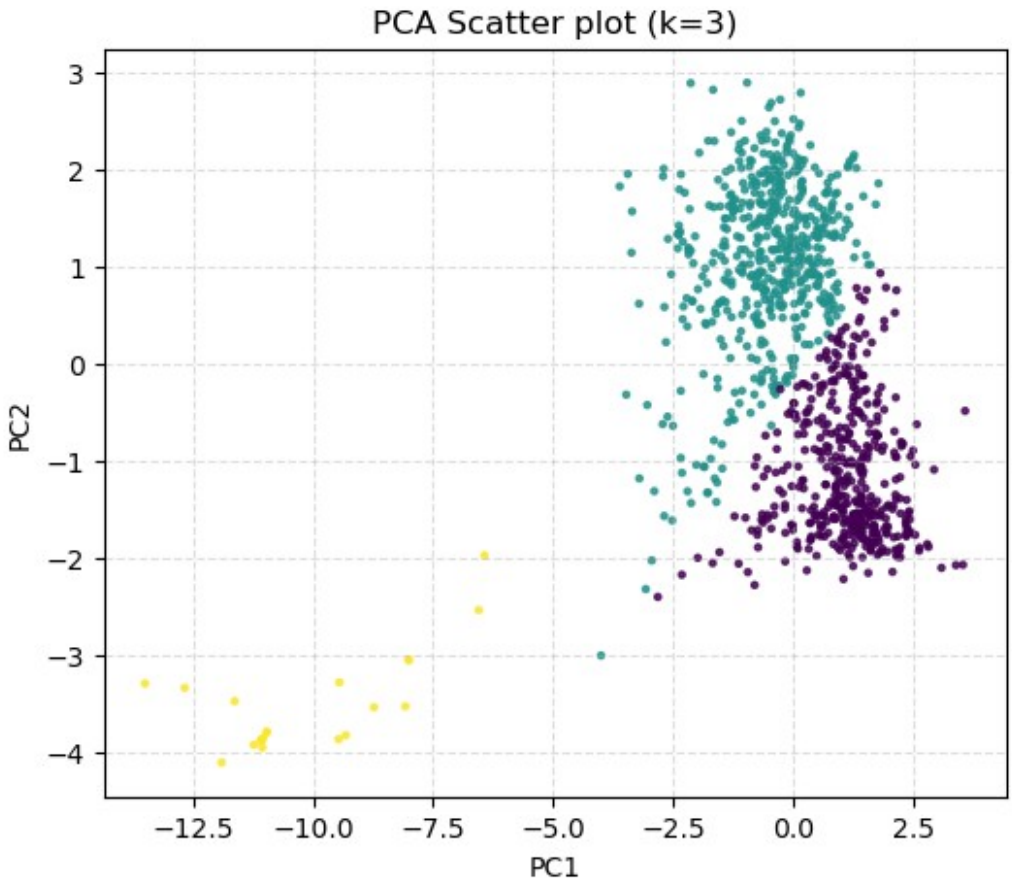

```
Cluster 0:
country, 0.573
pop, 0.107
rock, 0.087
alt-rock, 0.072
dance, 0.061
hip-hop, 0.017
k-pop, 0.011
folk, 0.010
indie-pop, 0.009
electro, 0.008

Cluster 1:
hip-hop, 0.640
dance, 0.077
pop, 0.074
electro, 0.034
show-tunes, 0.029
hip hop, 0.022
emo, 0.017
singer-songwriter, 0.013
jazz, 0.012
```

```
rock, 0.011

Cluster 2:
classical, 0.425
sleep, 0.234
german, 0.124
new-age, 0.092
folk, 0.046
ambient, 0.032
rock, 0.019
alt-rock, 0.016
singer-songwriter, 0.006
metal, 0.004

Cluster 3:
dance, 0.163
hip-hop, 0.161
metal, 0.156
rock, 0.079
alt-rock, 0.071
emo, 0.070
electro, 0.047
country, 0.038
hardcore, 0.035
gospel, 0.035


#dropped
results_drop_pca = kmeans_pipeline(X_pca_drop, df_drop_all,
pca_data=True, k_range=range(2, 10))

K:2, SIL:0.362, DB:1.142
K:3, SIL:0.407, DB:0.801
K:4, SIL:0.358, DB:0.934
K:5, SIL:0.386, DB:0.924
K:6, SIL:0.330, DB:0.943
K:7, SIL:0.314, DB:0.957
K:8, SIL:0.277, DB:0.976
K:9, SIL:0.281, DB:1.037
```


Silhouette Score vs k
Silhouette Score
0.40
0.38
0.36
0.34
0.32
0.30
0.28
2
3
4
5
6
7
8
9
k




```
Best k based on silhouette: 3
```




```
Cluster 0:
country, 0.449
rock, 0.088
pop, 0.085
dance, 0.083
alt-rock, 0.074
metal, 0.047
hip-hop, 0.028
emo, 0.022
electro, 0.018
gospel, 0.011

Cluster 1:
hip-hop, 0.605
dance, 0.088
pop, 0.078
electro, 0.034
show-tunes, 0.026
hip hop, 0.020
emo, 0.018
country, 0.017
rock, 0.015
```

```
singer-songwriter, 0.012

Cluster 2:
classical, 0.425
sleep, 0.233
german, 0.121
new-age, 0.091
folk, 0.046
ambient, 0.028
rock, 0.024
alt-rock, 0.019
singer-songwriter, 0.007
metal, 0.005
```

Results

1. Imputed Audio Features (Full Feature Space)

(Most expressive but noisy due to imputation)

The imputed, non-reduced clustering produced four clusters, that most differ from each other in acousticness, energy, loudness, and instrumentalness.

Cluster A — Pop / Acoustic / Show-Tunes

This cluster shows moderate acousticness and moderate energy but lacks extreme values. Genre distribution includes pop (18.4%), show-tunes (16.7%), hip-hop (12.8%), and a broad mix of acoustic and folk genres. The lack of strong acoustic signatures indicates a heterogeneous, mixed-purpose playlist group, likely blending mainstream pop with lighter acoustic tracks.

Cluster B — Energetic: Country / Rock / Pop

This cluster has high energy, high loudness, and high valence, characteristic of upbeat, amplified music. Genres align with this: country (45.7%), rock (9.7%), alt-rock (9.5%), and dance (8.8%).

Cluster C — Hip-Hop / Dance / Electronic

Highest danceability and highest speechiness, this cluster captures playlists dominated by hip-hop (70.2%), with supporting roles from dance (10.2%) and electronic genres. The acoustic profile (low acousticness, moderately loud, rhythmic) matches the trap / rap / club music sound profile.

Cluster D — Classical / Ambient / Instrumental

The most distinct cluster, defined by extremely low danceability, very low energy, very negative loudness, and very high acousticness and instrumentalness. Genre distribution confirms: classical (50.3%), sleep/ambient (28.9%), and piano/new-age music dominate. This cluster represents quiet, instrumental playlists used for studying, relaxing, or background ambience.

Interpretation: All four clusters are coherent and musically meaningful. However, the mixed nature of Cluster A indicates some noise introduced by imputation, which blurs boundaries between pop/acoustic/hip-hop playlists. Could it indicate overfitting?

2. Dropped-NA Audio Features (Full Feature Space)

(Cleaner structure, fewer playlists, stronger clusters)

When removing tracks with missing audio features, three clear and stable clusters appear.

Cluster 0 — Classical / Ambient / Instrumental

Similar to Cluster D above, this group contains extremely low-energy, quiet, instrumental playlists. Genres confirm: classical (41.1%), sleep (23.1%), folk, French acoustic, and piano/new-age. This cluster is the most homogeneous across all models.

Cluster 1 — Country / Rock / Pop (High Energy)

The energetic, major-key playlists again form a strong cluster. The dominant genres (country 43.0%, pop, alt-rock, dance) match the audio profile of high-energy and loud modern mainstream music.

Cluster 2 — Hip-Hop / Dance

The danceable, speech-heavy playlists again form a stable cluster dominated by hip-hop (59.0%), dance, pop, and electro. Audio features (high speechiness, high danceability) reinforce this.

Interpretation: Dropping NA values improves cluster clarity and creates tighter, more musically consistent groups. Compared to the imputed model, no noisy mixed-pop cluster appears; the structure becomes cleaner.

3 & 4. PCA-Reduced Models (Imputed + Dropped NA)

(Very clean grouping, but acoustic interpretability limited)

Using PCA to compress audio features into principal components produces three very clean and visually well-separated clusters, but interpretability of centroids decreases because PCA mixes features into abstract linear combinations. However, the genre composition of each cluster remains extremely stable and matches the non-reduced solutions:

Cluster A — Classical / Ambient / Instrumental Cluster B — Country / Rock / Pop Cluster C — Hip-Hop / Dance / Electronic

These clusters replicate the three dominant archetypes found previously, confirming that the structure is not dependent on missing values or high dimensionality.

Interpretation: The PCA clustering is visually the cleanest and most balanced. However, it sacrifices the ability to explain clusters directly in terms of audio features, which weakens interpretability.

Model	Clarity	Interpretability	Stability	Comments
Imputed (no	Medi	High	OK	4 clusters but one is noisy/mixed

Model	Clarity	Interpretability	Stability	Comments
PCA)	um			
Dropped NA (no PCA)	High	High	High	Clear clusters + good interpretability
Imputed PCA	High	Low	Medium	Clean, but audio meaning lost
Dropped PCA	High	Low	High	Best visually; weak interpretability

Clustering 2: Hierarchical clustering

I chose agglomerative clustering for the next option because it does not require a predefined number of clusters and builds a gull clustering tree which allows us to inspect the structure of the data, see natural separations and choose number of clusters after that. Valuable, since playlist audio distributions do not form obvious clusters.

Our final similarity matrix of 1026 playlists is also small enough for this to be computationally feasible.

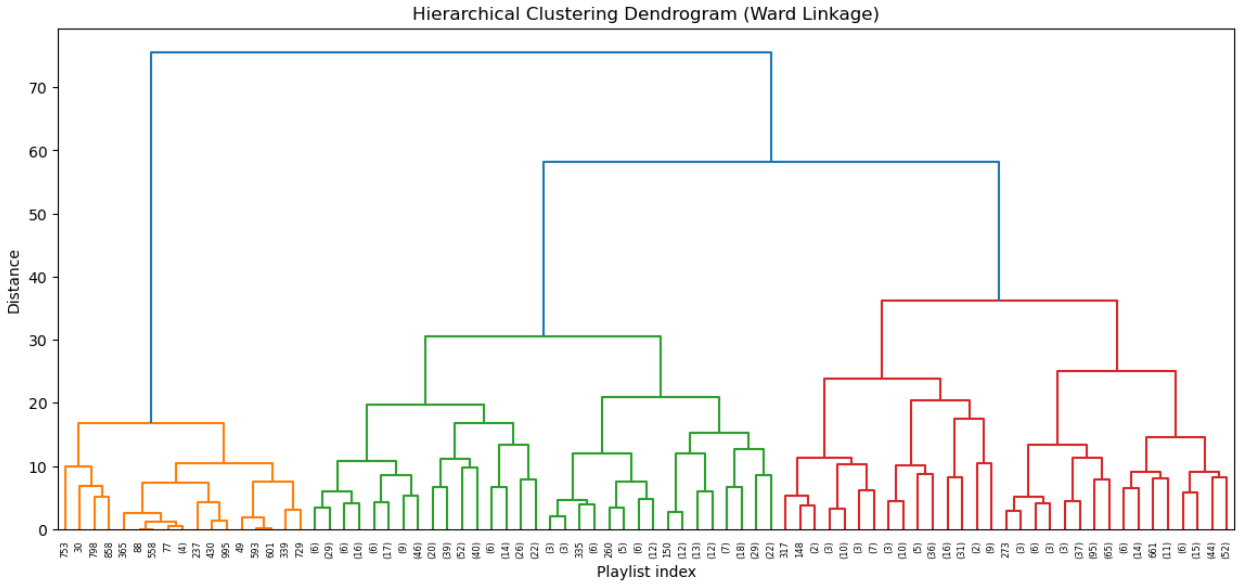
DBSCAN in this case would not work very well, because our clusters are continuous and overlapping and there is no individual dense blobs, only one continuous blob and large empty regions. Interpreting the clusters would also be very difficult.

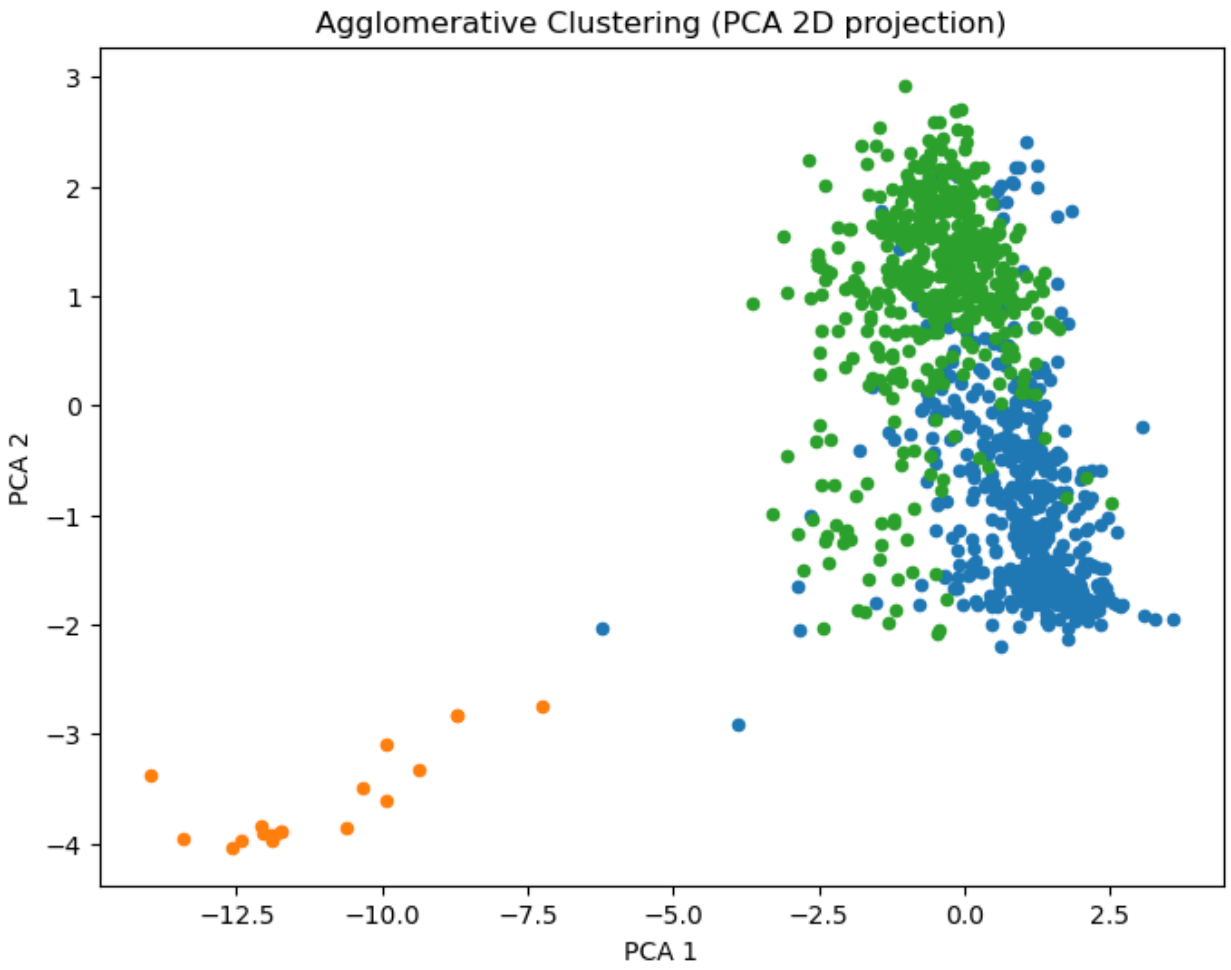
```
def hierarchical_clustering(X):
    # linkage: how clusters merge. 'ward' works best for numeric,
scaled data
    Z = linkage(X, method='ward')

    plt.figure(figsize=(14, 6))
    plt.title("Hierarchical Clustering Dendrogram (Ward Linkage)")
    plt.xlabel("Playlist index")
    plt.ylabel("Distance")
    dendrogram(Z, truncate_mode='level', p=6)  # shows only the top
merges for clarity
    plt.show()

def plot_pca_agg(X, n_clusters):
    agg = AgglomerativeClustering(
        n_clusters=n_clusters,
        linkage='ward'
    )

    clusters_agg = agg.fit_predict(X)
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)

    plt.figure(figsize=(8, 6))
    for cluster_id in sorted(set(clusters_agg)):
        mask = clusters_agg == cluster_id
```

```
        plt.scatter(
            X_pca[mask, 0],
            X_pca[mask, 1],
            s=20,
            label=f"Cluster {cluster_id}"
        )
    plt.title("Agglomerative Clustering (PCA 2D projection)")
    plt.xlabel("PCA 1")
    plt.ylabel("PCA 2")
    #plt.legend(title="Cluster")
    plt.show()
```

Imputed:

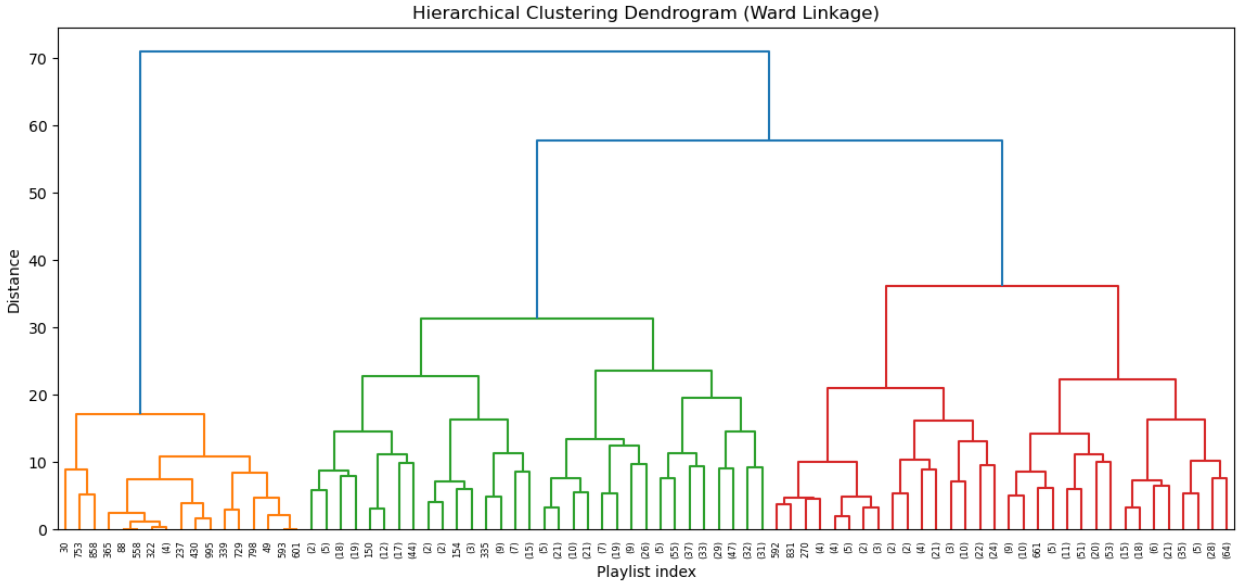
```
hierarchical_clustering(X_imp)
plot_pca_agg(X_imp, 3)
```

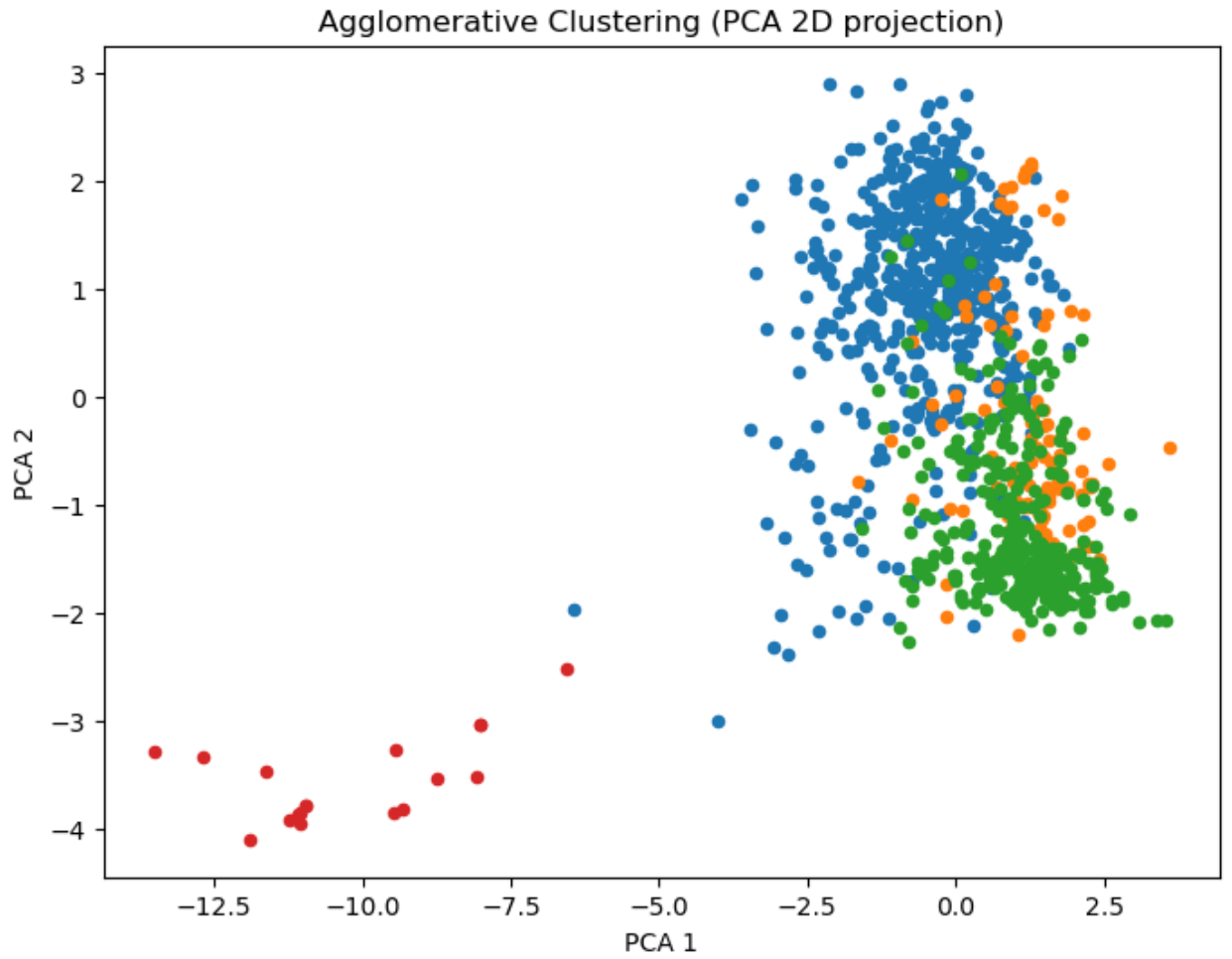




Dropped NAs:

```
hierarchical_clustering(X_drop)
plot_pca_agg(X_drop, 4)
```


Hierarchical Clustering Dendrogram (Ward Linkage)
Distance
Playlist index

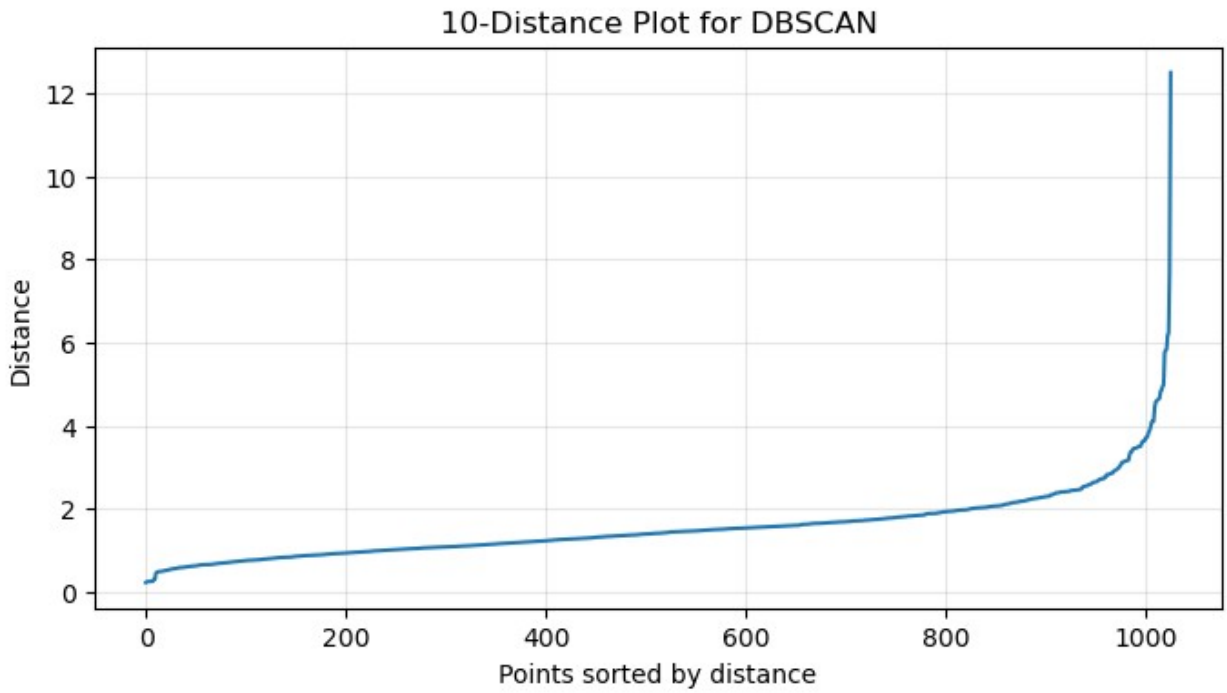

Agglomerative Clustering (PCA 2D projection)
PCA 2
PCA 1

This type of clustering is by itself not very interpretable, and it was more used to check about the choice of K. It yields results that agree on K-means and the clusters seem to be pretty much the same, even though K-means shows less overlap. Dendrogram shows that best K is either 3 or 4.

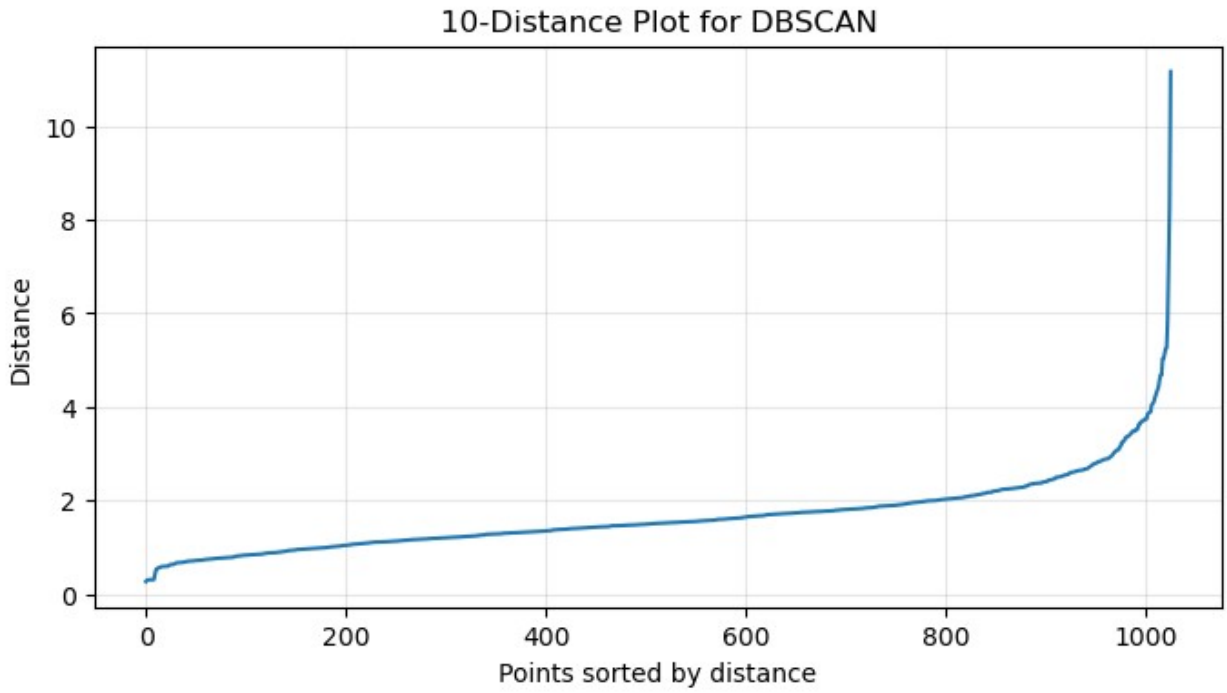
Clustering 3: DBSCAN (failed)

```
def plot_k_distance(X, k=10):

    nn = NearestNeighbors(n_neighbors=k)
    nn.fit(X)
    distances, _ = nn.kneighbors(X)

    # take the k-th distance, sort
    k_distances = np.sort(distances[:, -1])

    plt.figure(figsize=(8,4))
    plt.plot(k_distances)
    plt.title(f"{k}-Distance Plot for DBSCAN")
    plt.xlabel("Points sorted by distance")
    plt.ylabel("Distance")
    plt.grid(True, alpha=0.3)
    plt.show()

    return k_distances

print("Imputed:")
_ = plot_k_distance(X_imp)
print("Dropped NAs:")
_ = plot_k_distance(X_drop)
```

```
Imputed:
```




```
Dropped NAs:
```



eps chosen at elbow: 2.5

Very high, DBSCAN is probably not going to work.

```
def run_dbscan(X, eps, min_samples=10):
    db = DBSCAN(eps=eps, min_samples=min_samples)
    labels = db.fit_predict(X)

    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise = sum(labels == -1)

    print(f"DBSCAN found {n_clusters} clusters + {n_noise} noise
points")

    return labels, db

eps = 2.5
labels_imp, db_imp = run_dbscan(X_imp, eps)
labels_drop, db_drop = run_dbscan(X_drop, eps)
```

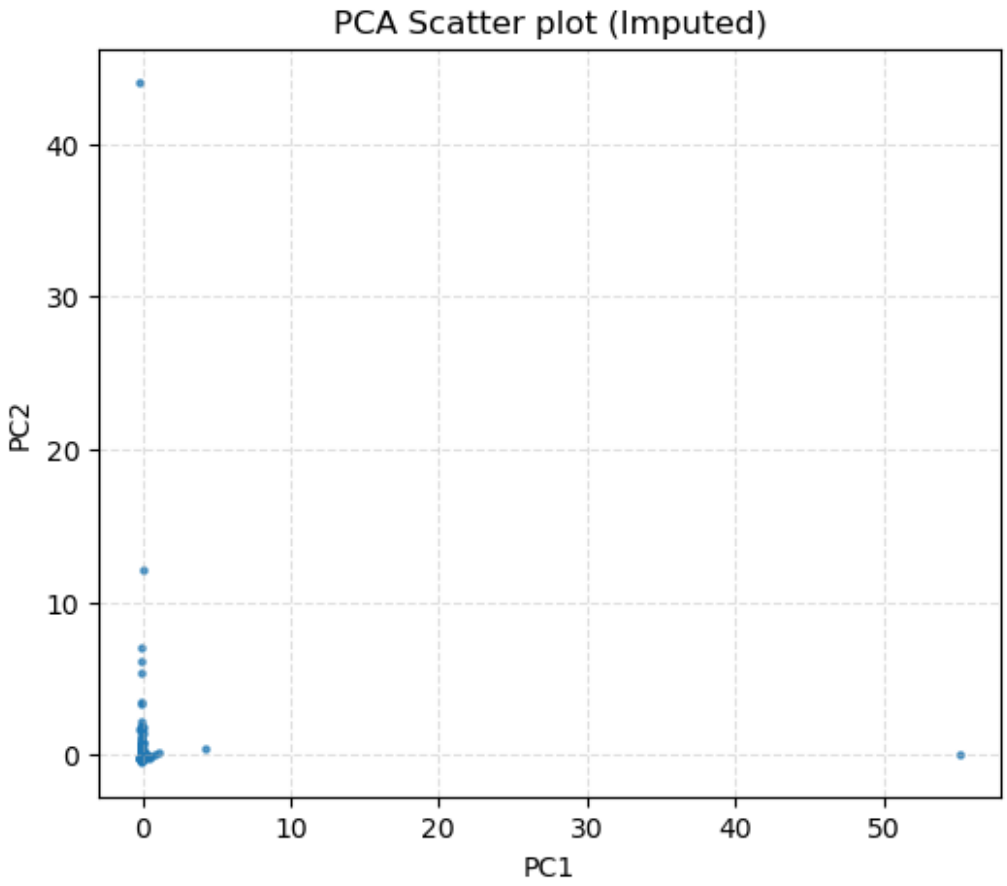
```
DBSCAN found 1 clusters + 48 noise points
DBSCAN found 2 clusters + 47 noise points
```

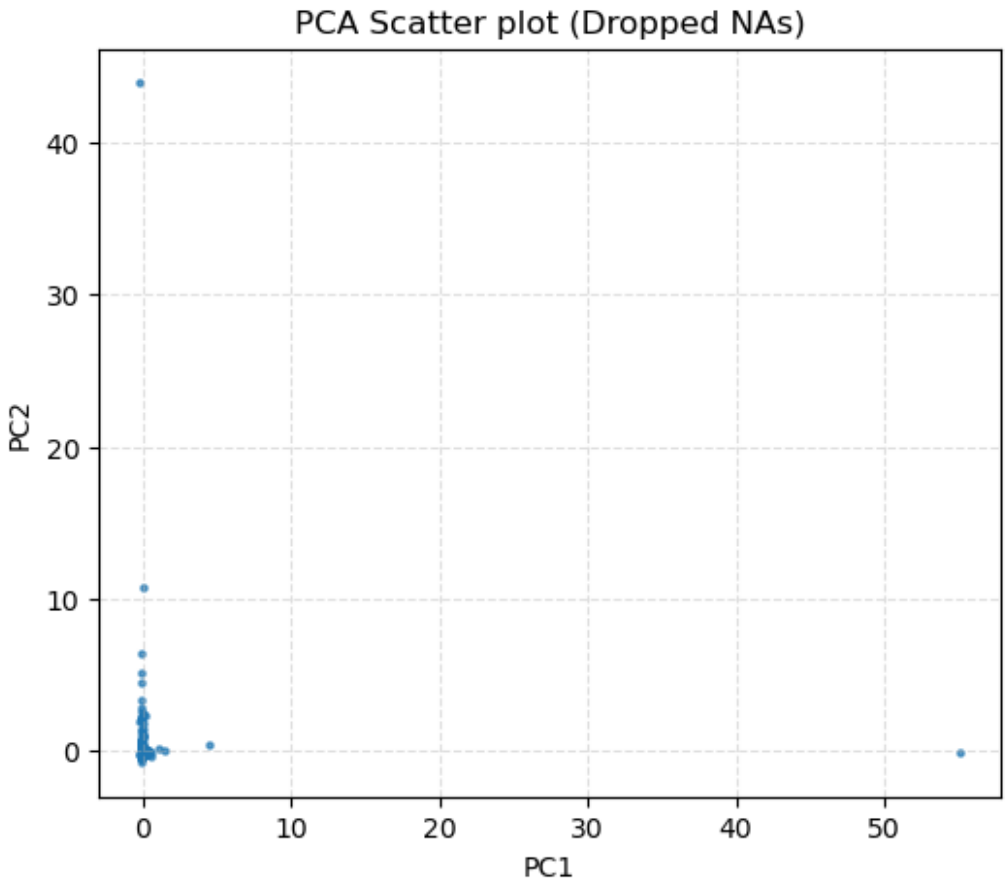
DBSCAN does not work with this data set, further testing and analysis with this is pointless.

2.2 Genre-based clustering

Data preparation:

```
#select columns
df_imp_genre = df_imp_all[genre_cols]
df_drop_genre = df_drop_all[genre_cols]

#scaling
df_imp_gsc = scale_features(df_imp_genre, genre_cols)
df_drop_gsc = scale_features(df_drop_genre, genre_cols)

#PCA
df_imp_gpca, _ = apply_pca(df_imp_gsc, n_components=0.90)
df_drop_gpca, _ = apply_pca(df_drop_gsc, n_components=0.90)

#visualisation
pca_plot(df_imp_gpca, "Imputed")
pca_plot(df_drop_gpca, "Dropped NAs")
```


PCA Scatter plot (Imputed)
PC2
PC1
40
30
20
10
0
0
10
20
30
40
50



The matrix is very sparse and that's why my PCA plots look like this but maybe it will get better.

K-Means:

```
def plot_pca_3d(X, labels, title="PCA 3D Projection of Genre
Clusters"):

    pca = PCA(n_components=3)
    X_pca = pca.fit_transform(X)

    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(111, projection='3d')

    scatter = ax.scatter(
        X_pca[:,0], X_pca[:,1], X_pca[:,2],
        c=labels, cmap='tab10', s=20, alpha=0.8
    )

    ax.set_xlabel("PC1")
    ax.set_ylabel("PC2")
    ax.set_zlabel("PC3")
    ax.set_title(title)
```

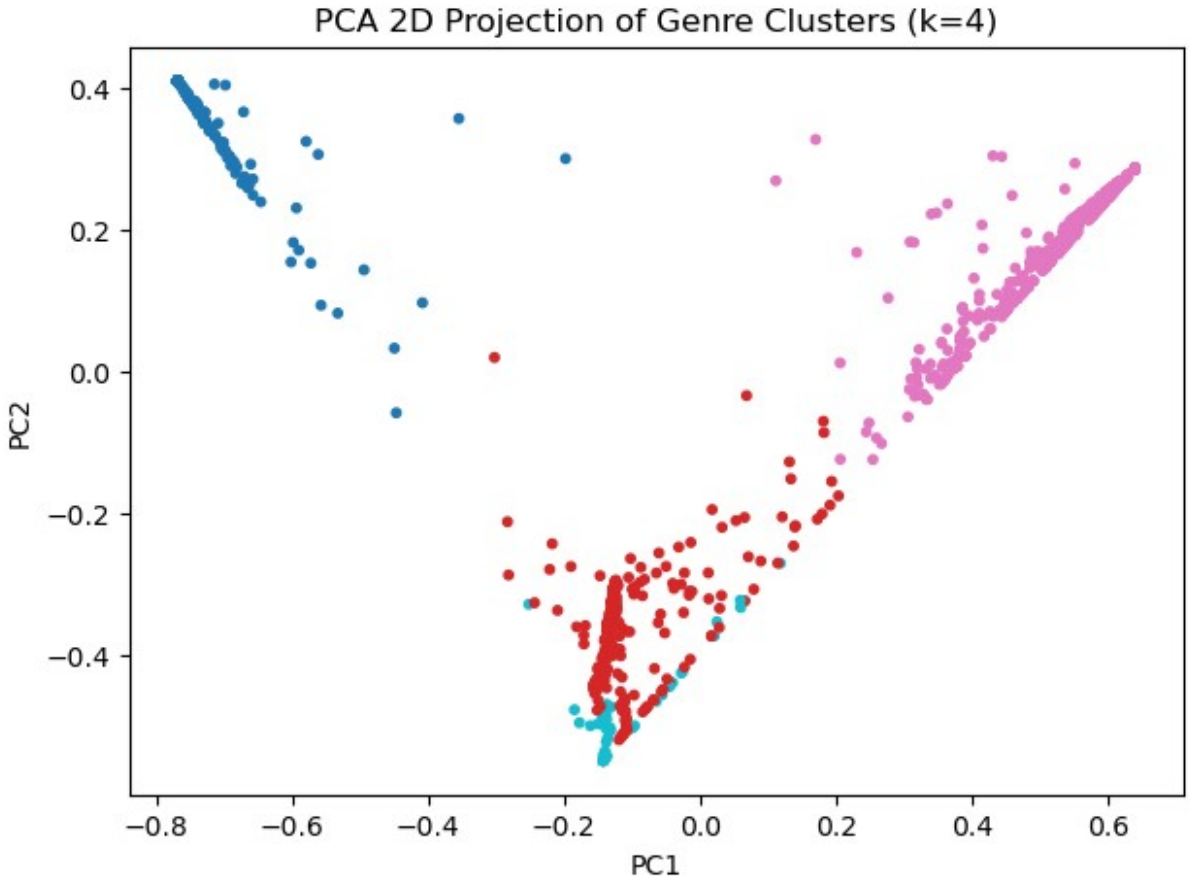
```
    legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
    ax.add_artist(legend1)

    plt.show()



def genre_kmeans_pipeline(X_genre, df_all, k_range=[3,4,5,6,7],
best_k=None, top_n=5, plot_umap=True, plot_heatmap=True,
random_state=SEED):

    results_sil = {}
    results_db = {}
    models = {}
    labels_dict = {}

    print("Running KMeans for multiple k:")
    for k in k_range:
        km = KMeans(n_clusters=k, random_state=random_state)
        labels = km.fit_predict(X_genre)
        sil = silhouette_score(X_genre, labels)
        db = davies_bouldin_score(X_genre, labels)

        results_sil[k] = sil
        results_db[k] = db
        models[k] = km
        labels_dict[k] = labels

        print(f"k={k}, silhouette={sil:.4f}, DB={db:.4f}")
    print()

    #best_k
    if best_k is None:
        best_k = max(results_sil, key=results_sil.get)
    print(f"Selected best_k = {best_k}")

    #best model
    best_model = models[best_k]
    best_labels = labels_dict[best_k]

    #cluster labels to dataframe
    df_all = df_all.copy()
    df_all['cluster_genre'] = best_labels

    #cluster sizes
    unique, counts = np.unique(best_labels, return_counts=True)
    print("\nCluster sizes:")
    for u, c in zip(unique, counts):
        print(f"Cluster {u+1}: {c} items")
```

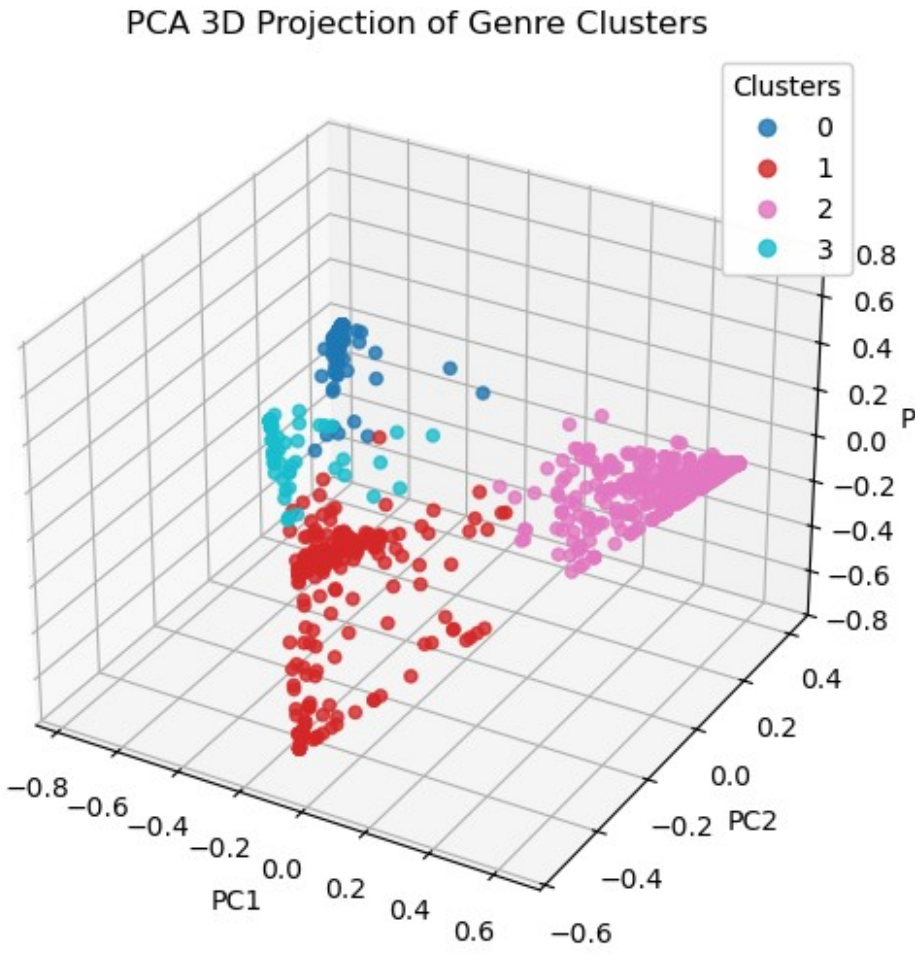
```
    #summary / centroids
    cluster_summary = df_all.groupby('cluster_genre').mean()

    #dominant genres per cluster
    for cluster_id, row in cluster_summary.iterrows():
        print(f"\n=== Cluster {cluster_id+1} ===")
        non_zero_genres = row[row > 0]
        if non_zero_genres.empty:
            print("No genres present.")
            continue
        top_genres =
non_zero_genres.sort_values(ascending=False).head(top_n)
        for genre, val in top_genres.items():
            print(f"{genre.replace('genre_', ''):20s} {val:.3f}")

    #pca plot
    pca = PCA(n_components=2, random_state=random_state)
    X_pca = pca.fit_transform(X_genre)
    plt.figure(figsize=(7,5))
    plt.scatter(X_pca[:,0], X_pca[:,1], c=df_all['cluster_genre'],
cmap='tab10', s=10)
    plt.title(f"PCA 2D Projection of Genre Clusters (k={best_k})")
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.show()

    plot_pca_3d(X_genre, df_all['cluster_genre'])

    #centroid heatmap
    if plot_heatmap:
        plt.figure(figsize=(10,6))
        sns.heatmap(cluster_summary, annot=False, cmap="viridis")
        plt.title(f"Cluster Centroid Heatmap (k={best_k})")
        plt.xlabel("Genres")
        plt.ylabel("Cluster")
        plt.show()

    #UMAP visualization
    if plot_umap:
        reducer = umap.UMAP(random_state=random_state, init="random")
        embedding = reducer.fit_transform(X_genre)
        plt.figure(figsize=(7,5))
        plt.scatter(embedding[:,0], embedding[:,1],
c=df_all['cluster_genre'], cmap='tab10', s=10)
        plt.title("UMAP Genre Clusters")
        plt.show()

    return {
```

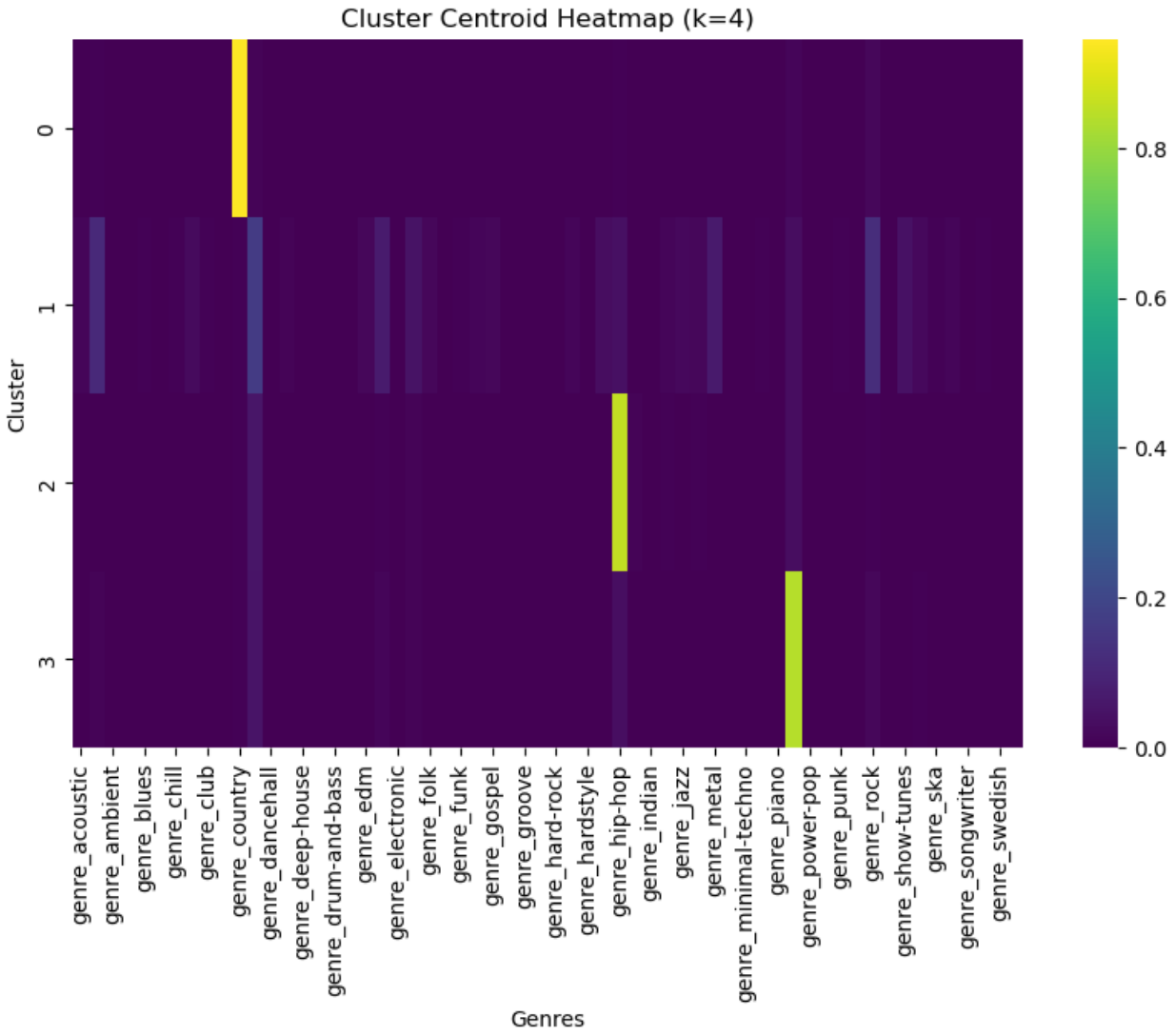
```
        "best_k": best_k,
        "model": best_model,
        "labels": best_labels,
        "silhouette_scores": results_sil,
        "db_scores": results_db,
        "cluster_summary": cluster_summary
    }

results_genre_imp = genre_kmeans_pipeline(df_imp_genre, df_imp_genre,
best_k=4)
```

```
Running KMeans for multiple k:
k=3, silhouette=0.5253, DB=1.0370
k=4, silhouette=0.5550, DB=1.0849
k=5, silhouette=0.5767, DB=1.1414
k=6, silhouette=0.6067, DB=1.1918
k=7, silhouette=0.6223, DB=1.2001

Selected best_k = 4

Cluster sizes:
Cluster 1: 224 items
Cluster 2: 347 items
Cluster 3: 394 items
Cluster 4: 61 items

=== Cluster 1 ===
country              0.944
rock                 0.016
pop                  0.015
dance                0.007
hip-hop              0.005

=== Cluster 2 ===
dance                0.166
rock                 0.121
alt-rock             0.109
electro              0.069
metal                0.063

=== Cluster 3 ===
hip-hop              0.857
dance                0.052
pop                  0.031
emo                  0.010
house                0.008

=== Cluster 4 ===
pop                  0.838
dance                0.049
```

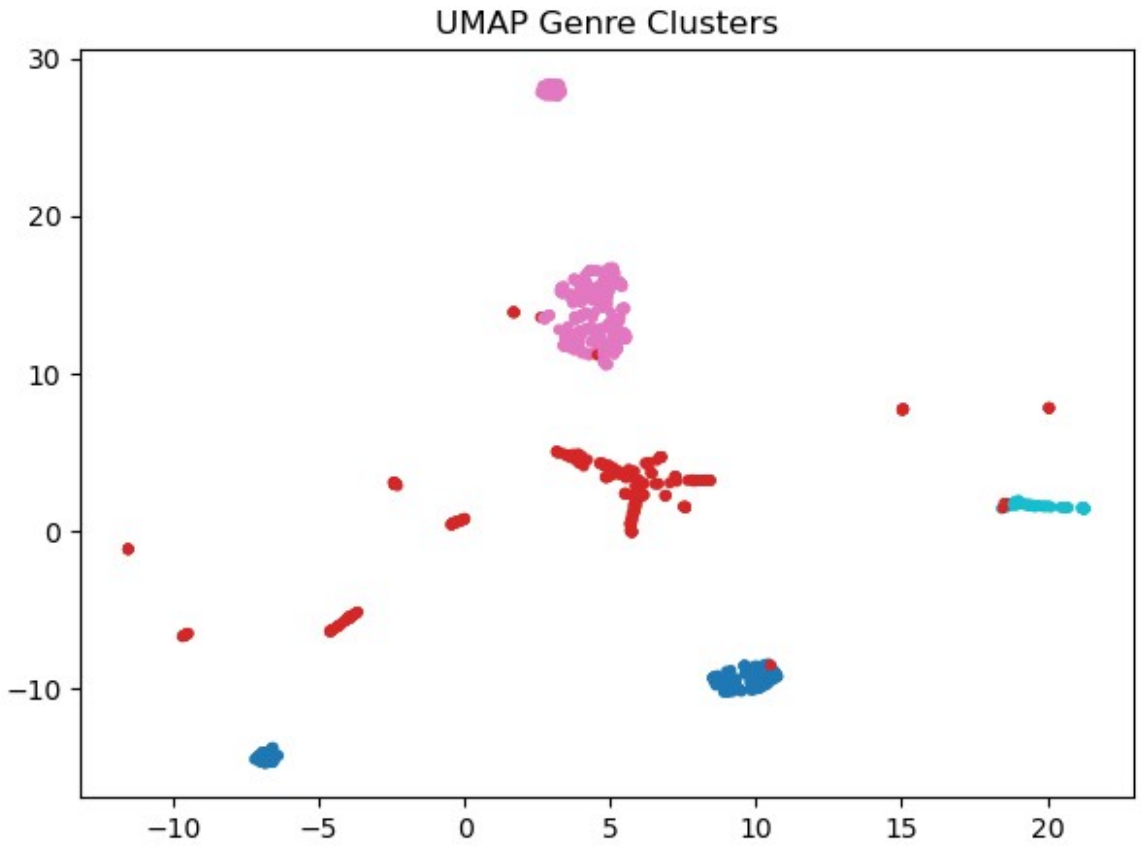
```
hip-hop                 0.035
rock                    0.017
electro                 0.014
```




PCA 3D Projection of Genre Clusters
Clusters
0
1
2
3
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
P
0.4
0.2
0.0
−0.2
PC2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
PC1

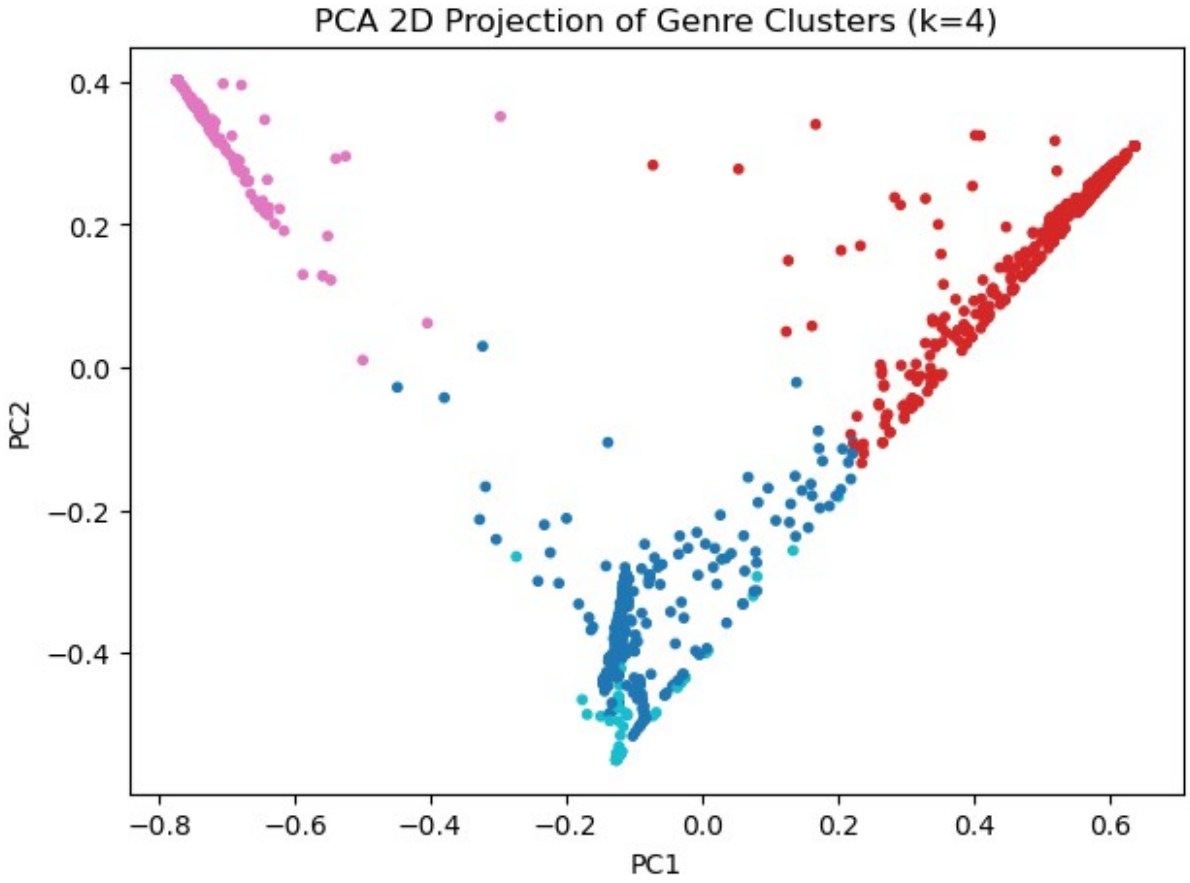


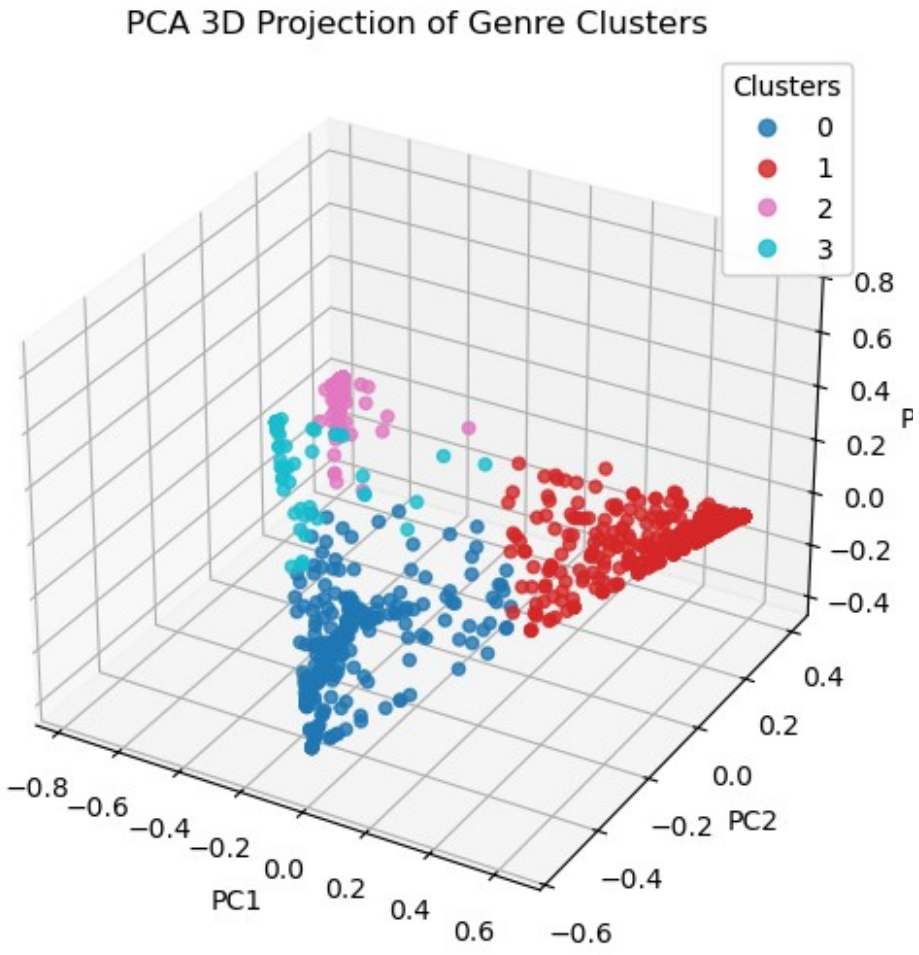

```
c:\Users\lunka\miniforge3\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state.
Use no seed for parallelism.
  warn(
```

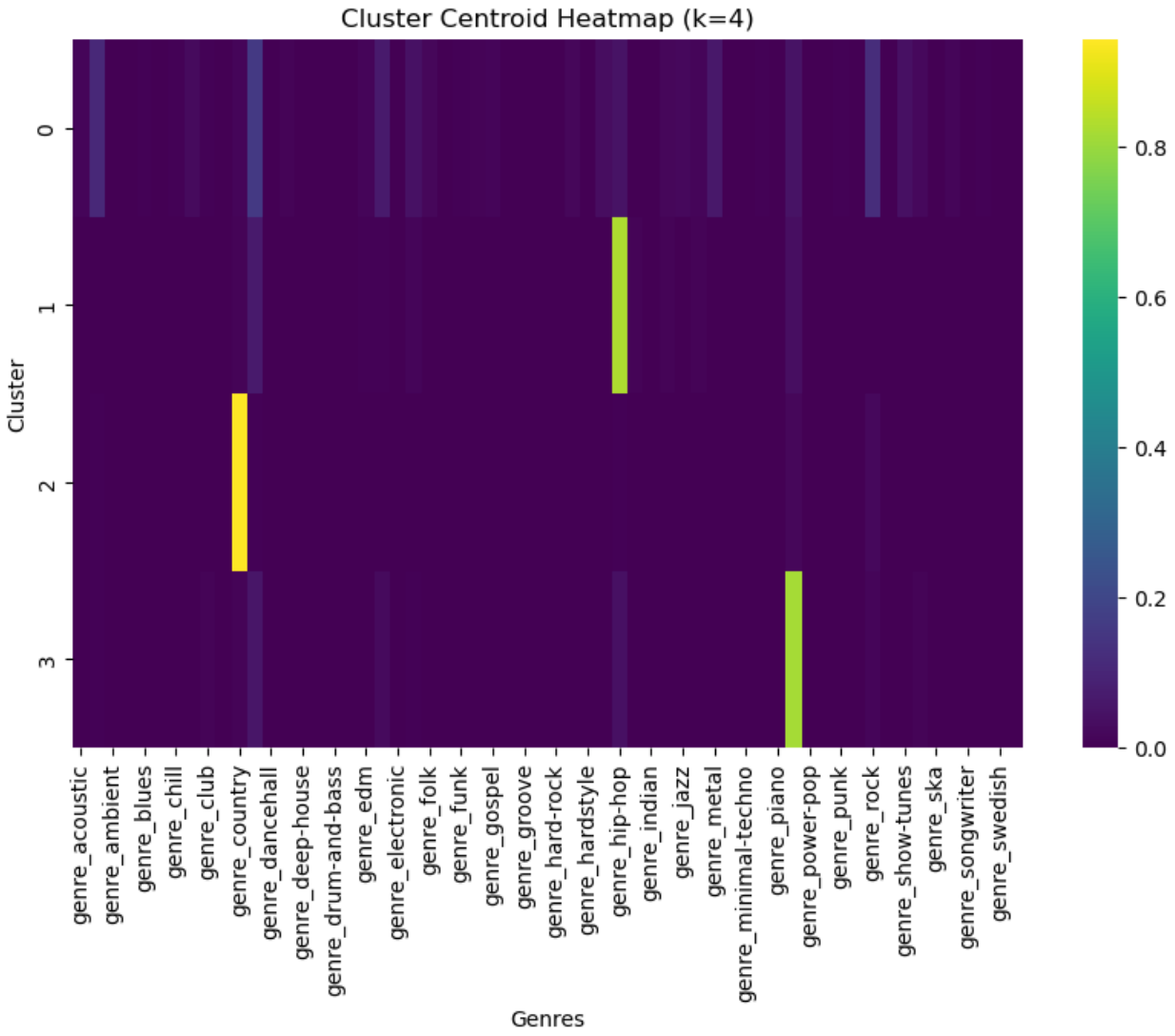



```
results_genre_drop = genre_kmeans_pipeline(df_drop_genre,
df_drop_genre, best_k=4)

Running KMeans for multiple k:
k=3, silhouette=0.4937, DB=1.0782
k=4, silhouette=0.5149, DB=1.1226
k=5, silhouette=0.5272, DB=1.5269
k=6, silhouette=0.5500, DB=1.2889
k=7, silhouette=0.5652, DB=1.5325

Selected best_k = 4

Cluster sizes:
Cluster 1: 356 items
Cluster 2: 392 items
Cluster 3: 219 items
Cluster 4: 59 items

=== Cluster 1 ===
dance                0.157
rock                 0.121
alt-rock             0.105
electro              0.066
metal                0.061
```

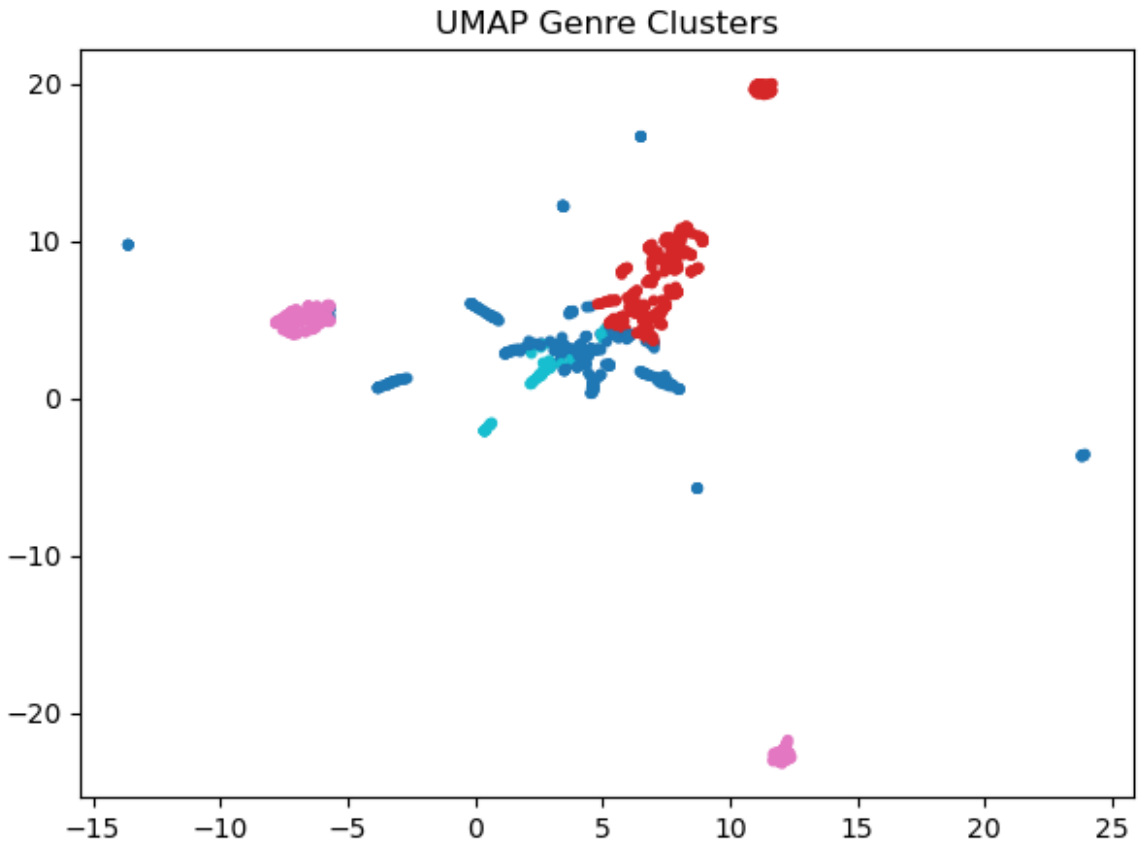
```
=== Cluster 2 ===
hip-hop                0.827
dance                  0.065
pop                    0.035
emo                    0.012
house                  0.010

=== Cluster 3 ===
country                0.943
rock                   0.019
pop                    0.017
dance                  0.007
hip-hop                0.004

=== Cluster 4 ===
pop                    0.815
dance                  0.055
hip-hop                0.040
electro                0.023
rock                   0.014
```




PCA 3D Projection of Genre Clusters
Clusters
0
1
2
3
PC1
PC2
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8




```
c:\Users\lunka\miniforge3\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state.
Use no seed for parallelism.
  warn(
```



Comparing the two KMeans runs, both selected best_k = 4, but the clusters differ in structure and cohesion. In the first run, silhouette scores were generally higher (up to 0.6057), indicating tighter, more well-separated clusters, while the second run had lower silhouette scores (up to 0.5540), suggesting slightly less distinct separation.

The cluster sizes also differ: the first run had a small alternative rock cluster (44 playlists) and a moderate country cluster (191), whereas in the second run, the country cluster became larger (189), the pop cluster expanded (403), and a small show-tunes/jazz cluster emerged (22 playlists).

Looking at dominant genres, the first run's clusters were clearer in thematic separation:

Cluster 1: pop/dance/rock

Cluster 2: country

Cluster 3: alt-rock/rock

Cluster 4: hip-hop

In the second run, clusters mix some genres more:

Cluster 1: pop/dance/alt-rock/rock

Cluster 2: hip-hop/dance/pop

Cluster 3: show-tunes/jazz

Cluster 4: country

Overall, the first clustering appears more interpretable with distinct genre focus per cluster, while the second clustering produces slightly more mixed clusters but captures niche genres like show-tunes and jazz. Depending on the application—recommendation vs. exploration—the first run may provide cleaner genre-based separation, while the second run may highlight minor niche clusters.

2.3 Text-based clustering (playlist name)

Data load:

```
df_text = pd.read_csv("data/data_text.csv")
df_text = df_text[['playlist_id', 'playlist_name']]
```

TD-IDF and K-means:

```
#TF-IDF: names to vectors
vectorizer = TfidfVectorizer(
    stop_words='english',
    max_features=3000,
    ngram_range=(1,2)
)

X_tfidf = vectorizer.fit_transform(df_text['playlist_name'])


#KMEANS
sil_scores = {}
for k in range(2, 10):
    km = KMeans(n_clusters=k, random_state=1234)
    labels = km.fit_predict(X_tfidf)
    sil = silhouette_score(X_tfidf, labels)
    sil_scores[k] = sil
    print(k, sil)
```

```
2 0.026919823660102184
3 0.07625795690911678
4 0.10085423073661584
5 0.12140489040459408
6 0.12244766928892316
7 0.151656142002267
8 0.2091412141624785
9 0.21817456208820993
```

Interpretation of different ks: It cannot really be chosen by score, because it will keep increasing as I increase k, so I am focusing more on impretablity. Big improvement from 3 to 5, very small

changes from 5 to 6 and then probably overfitting. K's bigger than 6 would probably make too specific clusters and they would capture a lot of noise and meaningless words.

```
#terms ranked by importance for the 4 considered ks
for k_try in [3, 4, 5, 6]:
    km = KMeans(n_clusters=k_try, random_state=SEED)
    labels = km.fit_predict(X_tfidf)
    centers = km.cluster_centers_
    terms = vectorizer.get_feature_names_out()

    print(f"\nk={k_try}")
    for i in range(k_try):
        top = centers[i].argsort()[::-1][:8]
        print(f"Cluster {i}: {[terms[j] for j in top]}")
```

```
k=3
Cluster 0: ['ap', 'ap rocky', 'rocky', 'mob', 'ap mob', 'rocky long',
'live', 'version']
Cluster 1: ['hip hop', 'hip', 'hop', 'det fandme', 'det', 'fandme',
'fandme hip', '2000']
Cluster 2: ['country', 'pop', 'rock', 'rock musik', 'musik', 'hits',
'pop hits', '2025']

k=4
Cluster 0: ['ap', 'ap rocky', 'rocky', 'mob', 'ap mob', 'rocky long',
'live', 'version']
Cluster 1: ['hip hop', 'hip', 'hop', 'det fandme', 'det', 'fandme',
'fandme hip', '2000']
Cluster 2: ['country', 'rock', 'pop', 'rock musik', 'musik', 'new',
'pop girlies', 'girlies']
Cluster 3: ['pop hits', '2025', 'hits', 'pop', '2015 2025', 'hits
2015', '2015', '2000s 2025']

k=5
Cluster 0: ['ap', 'ap rocky', 'rocky', 'mob', 'ap mob', 'rocky long',
'live', 'version']
Cluster 1: ['hip hop', 'hip', 'hop', 'det fandme', 'det', 'fandme',
'fandme hip', '2000']
Cluster 2: ['country', 'rock', 'pop', 'musik', 'rock musik', 'new',
'pop girlies', 'girlies']
Cluster 3: ['pop hits', '2025', 'hits', 'pop', '2015 2025', 'hits
2015', '2015', '2000s 2025']
Cluster 4: ['tech', 'tech n9ne', 'n9ne', 'øne piløts', 'film',
'future', 'funky', 'fun songs']

k=6
Cluster 0: ['ap', 'ap rocky', 'rocky', 'mob', 'ap mob', 'rocky long',
'live', 'version']
Cluster 1: ['hip hop', 'hip', 'hop', 'det fandme', 'det', 'fandme',
```

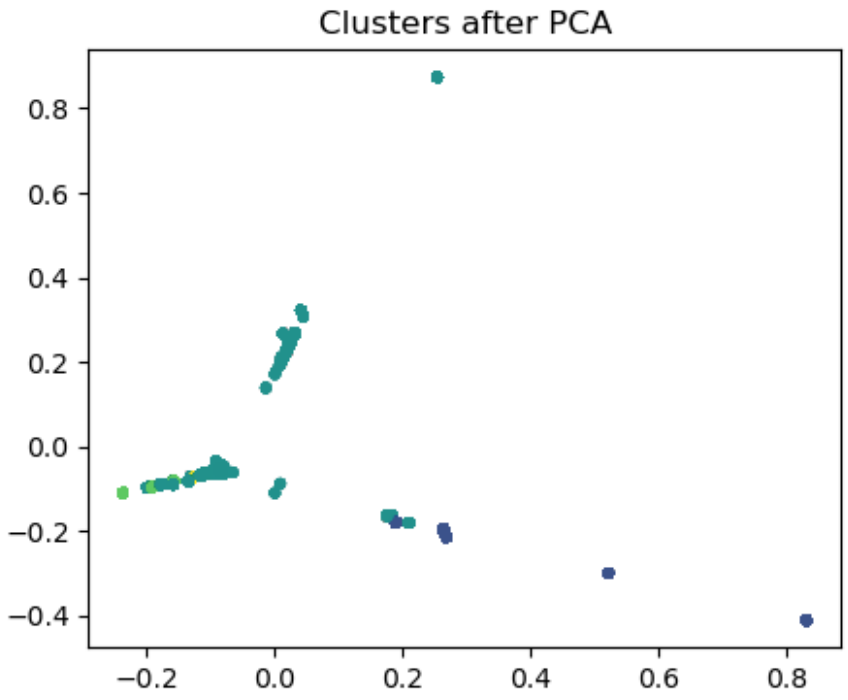
```
'fandme hip', '2000']
Cluster 2: ['country', 'rock', 'pop', 'rock musik', 'musik',
'girlies', 'pop girlies', 'eazy']
Cluster 3: ['pop hits', '2025', 'hits', 'pop', '2015 2025', 'hits
2015', '2015', '2000s 2025']
Cluster 4: ['tech', 'tech n9ne', 'n9ne', 'øne piløts', 'film',
'future', 'funky', 'fun songs']
Cluster 5: ['new', 'music', 'new country', 'country', 'country music',
'new music', 'new new', 'new playlist']

#FINAL MODEL ON K = 5

km_5 = KMeans(n_clusters=5, random_state=1234)
labels = km_5.fit_predict(X_tfidf)
centers = km_5.cluster_centers_
terms = vectorizer.get_feature_names_out()

unique, counts = np.unique(labels, return_counts=True)
for u, c in zip(unique, counts):
    print(f"Cluster {u+1}: {c} items")

Cluster 1: 244 items
Cluster 2: 3883 items
Cluster 3: 37278 items
Cluster 4: 2064 items
Cluster 5: 1003 items
```

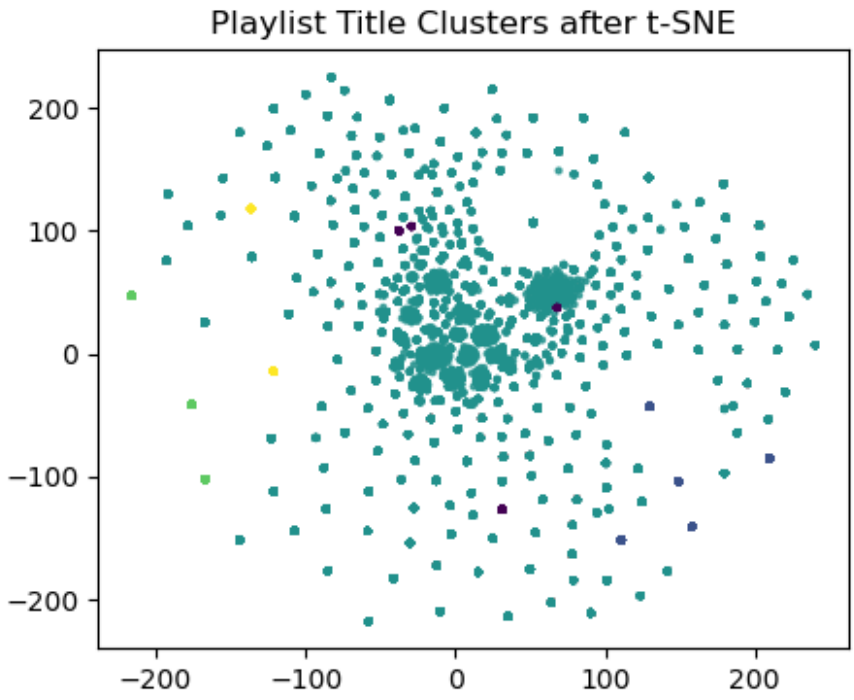
Visualization:

```
#dimensionality reduction
pca = PCA(n_components=0.90, random_state=1234)
X_pca = pca.fit_transform(X_tfidf.toarray())

tsne = TSNE(n_components=2, perplexity=30, random_state=1234)
X_tsne = tsne.fit_transform(X_pca)

plt.figure(figsize=(5, 4))
scatter = plt.scatter(
    X_pca[:, 1],
    X_pca[:, 0],
    c=labels,
    cmap="viridis",
    s=10,
    alpha=0.8
)
plt.title("Clusters after PCA")

plt.figure(figsize=(5, 4))
scatter = plt.scatter(
    X_tsne[:, 1],
```

```
    X_tsne[:, 0],
    c=labels,
    cmap="viridis",
    s=4,
    alpha=0.8
)
plt.title("Playlist Title Clusters after t-SNE")
```

```
Text(0.5, 1.0, 'Playlist Title Clusters after t-SNE')
```





Clusters are highly imbalanced as cluster 2 contains more than 80% of all playlists. Playlist titles are too short and random/uninformative and are not useful for genre separation or recommendation. In the visualisations it is also clear that clustering was very unsuccessful.

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Input file
input_file = "merged_with_features.csv"

# Output files
train_file = "train.csv"
test_file = "test.csv"

# Split settings
test_size = 0.2         # 20% test, 80% train
random_state = 42       # for reproducibility

def main():
    # Load the data
    df = pd.read_csv(input_file)

    # Split using sklearn
    train_df, test_df = train_test_split(
        df,
        test_size=test_size,
        random_state=random_state,
        shuffle=True
    )

    # Save outputs
    train_df.to_csv(train_file, index=False)
    test_df.to_csv(test_file, index=False)

    print("Done!")
    print(f"Training rows: {len(train_df)}")
    print(f"Testing rows: {len(test_df)}")

if __name__ == "__main__":
    main()
```

Playlist recommender using Locality-Sensitive Hashing and Jaccard similarity

```
import pandas as pd
import numpy as np
from math import log2
import random
```

Load train data

We begin by loading the training dataset, which contains playlists and their associated tracks.

```
df = pd.read_csv("train.csv")
df.head()
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"playlist_id","rawType":"object","type":"string"},
{"name":"playlist_name","rawType":"object","type":"string"},
{"name":"artist_id","rawType":"object","type":"unknown"},
{"name":"artist_name","rawType":"object","type":"unknown"},
{"name":"track_id","rawType":"object","type":"string"},
{"name":"track_name","rawType":"object","type":"string"},
{"name":"album_id","rawType":"object","type":"unknown"},
{"name":"album_name","rawType":"object","type":"unknown"},
{"name":"duration_ms","rawType":"float64","type":"float"},
{"name":"position","rawType":"float64","type":"float"},
{"name":"genre","rawType":"object","type":"unknown"},
{"name":"popularity","rawType":"float64","type":"float"},
{"name":"year","rawType":"float64","type":"float"},
{"name":"danceability","rawType":"float64","type":"float"},
{"name":"energy","rawType":"float64","type":"float"},
{"name":"key","rawType":"float64","type":"float"},
{"name":"loudness","rawType":"float64","type":"float"},
{"name":"mode","rawType":"float64","type":"float"},
{"name":"speechiness","rawType":"float64","type":"float"},
{"name":"acousticness","rawType":"float64","type":"float"},
{"name":"instrumentalness","rawType":"float64","type":"float"},
{"name":"liveness","rawType":"float64","type":"float"},
{"name":"valence","rawType":"float64","type":"float"},
{"name":"tempo","rawType":"float64","type":"float"},
{"name":"time_signature","rawType":"float64","type":"float"}],"ref":"e
1095064-eeed-4175-acad-8631297835f3","rows":[["0","461206","Miranda
Lambert","66lH4jAE7pqPlOlzUKbwA0","Miranda
Lambert","4Gyhy413uPALzaVg4S1DpX","Keeper of the
Flame","563h536tB6n8Dn62jr4RZG","The Weight of These
```

```
Wings","239733.0","95.0","country","31.0","2016.0","0.622","0.673","6.
0","-6.372","1.0","0.0361","0.45","2.35e-
05","0.13","0.337","116.567","4.0"],
["1","545224","mhm","3zLWOB0I86EiVgG5NrX1ht","Jarrod
Alonge","3jKrgzpPb23ZmEIAipUKnZ","Hey Jarrod, What's That Song
Again?","0XFMnZrWpEYMVRrBdHmsGZ","Beating a Dead Horse: Deluxe Ultra-
Limited Exclusive Undead
Edition","255185.0","4.0",null,null,null,null,null,null,null,null,null
,null,null,null,null,null,null],["2","0dMexqq0XIWS3QJ74z3ZhD","Hip Hop
2000s Music - Best Hip Hop Hits of the 00s Playlist (Top Hip Hop Songs
from 2000 to
2009)",null,null,"5dL5jv5GSCRoDhTtnY8maL","Mesmerize",null,null,null,n
ull,"hip
hop",null,null,null,null,null,null,null,null,null,null,null,null,null,
null],["3","279103","litty","46SHBwWsqBkxI7EeeBEQG7","Kodak
Black","34oWbFBfGEElvgO0a5c9V4","No
Flockin","12YTH28wiBXQ16gvWOCMLU","No Flockin","165290.0","18.0","hip-
hop","73.0","2015.0","0.943","0.595","5.0","-
8.372","0.0","0.191","0.0673","0.0","0.0839","0.815","117.532","4.0"],
["4","1WH6WVBwPBz35ZbWsgCpgr","Top Pop Hits 2015-2025",null,"Topic,
A7S","3H7ihDc1dqLriiWXwsc2po","Breaking
Me",null,null,"166794.0",null,"pop","75.0","2019.0","0.789","0.72","8.
0","-
5.652","0.0","0.218","0.223","0.0","0.129","0.664","122.031","4.0"]],"
shape":{"columns":25,"rows":5}}
```


Group tracks by playlist

To work with playlists as sets, we group all track IDs belonging to the same playlist. Each playlist is therefore represented as a Python set, where duplicate tracks are removed automatically.

```
playlist_dict = (
    df.groupby("playlist_id")["track_id"]
      .apply(set)
      .to_dict()
)

# make sure playlist_id are strings
playlist_dict = {
    str(pid): tracks
    for pid, tracks in playlist_dict.items()
}

sizes = {pid: len(s) for pid, s in playlist_dict.items()}
min(sizes.values()), max(sizes.values())
```

```
(1, 439)
```

MinHash signatures

Each playlist is first transformed into a MinHash signature—a compact representation that preserves similarity.
Playlists with similar signatures have a high probability of being placed in the same LSH buckets

MinHash works by hashing each element in a set using a family of independent hash functions and keeping only the minimum hashed value for each function.

A key detail is that all MinHash instances must share the exact same hash functions. Otherwise, signatures become incomparable and LSH cannot identify similar playlists. We therefore generate a fixedhash functions reuse them for all playlists.

Python's built-in `hash()` cannot be used because it changes across runs, so we define `stable_hash` function to ensure reproducible hashing. Using these components, each playlist is converted into a compact MinHash signature that captures the identity of its track set while enabling efficient comparison through LSH.

```
NUM_PERM = 256
MAX_HASH = 2**32 - 1

def generate_hash_params(num_perm, seed=42):
    random.seed(seed)

    prime = 4294967311
    params = []
    for _ in range(num_perm):
        a = random.randint(1, prime - 1)
        b = random.randint(0, prime - 1)
        params.append((a, b, prime))
    return params

# IMPORTANT: all MinHash objects must share the same hash functions
GLOBAL_HASH_PARAMS = generate_hash_params(NUM_PERM, MAX_HASH)

def stable_hash(x):
    return int.from_bytes(x.encode('utf8'), 'little') & 0xffffffff

class MinHash:
    def __init__(self, num_perm=512):
        self.num_perm = num_perm
        self.max_hash = (1 << 32) - 1
        self.hash_params = GLOBAL_HASH_PARAMS   # shared
        self.signature = [self.max_hash] * num_perm

    def update(self, value):
        h = stable_hash(value)
        for i, (a, b, prime) in enumerate(self.hash_params):
            new_hash = (a * h + b) % prime
```

```
            if new_hash < self.signature[i]:
                self.signature[i] = new_hash

    def digest(self):
        return self.signature


def create_minhash_from_tracks(track_set):
    m = MinHash()
    for t in track_set:
        m.update(str(t))
    return m



minhash_dict = {
    pid: create_minhash_from_tracks(tracks)
    for pid, tracks in playlist_dict.items()
}

sigs = [mh.digest() for mh in minhash_dict.values()]
unique_sigs = len({tuple(sig) for sig in sigs})
print("Unique signatures:", unique_sigs)
print("Total playlists:", len(minhash_dict))


Unique signatures: 932
Total playlists: 936
```

Locality-Sensitive Hashing (LSH) to find neighbors fast

Computing Jaccard similarity between all pairs of playlists is computationally expensive, especially when the dataset grows.
To address this, we use Locality-Sensitive Hashing (LSH), which enables fast approximate nearest-neighbor search. The signature of length num_perm is divided into a number of *bands*, each containing a fixed number of rows. For every band, the corresponding slice of the signature is hashed and stored in a dictionary of buckets. Playlists whose band hashes match are placed into the same bucket and then treated as candidate neighbors.

After building the LSH index over all playlists, we can query it with the MinHash signature of any playlist to obtain a list of candidate neighbors. The final line below demonstrates that the indexing and lookup works as expected.

```
class LSH:
    def __init__(self, num_perm=NUM_PERM, bands=256):
        assert num_perm % bands == 0, "num_perm must be divisible by
bands"
        self.num_perm = num_perm
        self.bands = bands
        self.rows = num_perm // bands      # rows per band
```

```
        self.buckets = [dict() for _ in range(bands)]

    def _band_hash(self, band_values):
        return hash(tuple(band_values))

    def insert(self, key, minhash_obj):
        sig = minhash_obj.digest()
        for b in range(self.bands):
            start = b * self.rows
            end = start + self.rows
            band = sig[start:end]
            h = self._band_hash(band)
            bucket = self.buckets[b].setdefault(h, [])
            bucket.append(key)

    def query(self, minhash_obj):
        sig = minhash_obj.digest()
        candidates = set()
        for b in range(self.bands):
            start = b * self.rows
            end = start + self.rows
            band = sig[start:end]
            h = self._band_hash(band)
            bucket = self.buckets[b].get(h)
            if bucket:
                candidates.update(bucket)
        return list(candidates)


lsh = LSH(num_perm=NUM_PERM, bands=256)

for pid, mh in minhash_dict.items():
    lsh.insert(pid, mh)

# Testing for a random playlist
some_pid = next(iter(minhash_dict.keys()))
print("Example LSH candidates:", lsh.query(minhash_dict[some_pid]))
```

```
Example LSH candidates: ['7DgPQwzEoUVfQYBiMLER9Z',
'4DJztJkufdlND0Hvg4nGkK', '489404', '523500',
'5B46dEOFzUu8z3uPC9lBLt', '0dMexqq0XIWS3QJ74z3ZhD',
'56un2laj6rmMUKhDlkUkAY']
```

Generating recommendations

Once we can retrieve similar playlists through LSH, we generate recommendations by aggregating tracks from the nearest neighbors.

The intuition is that playlists that share many songs with a query playlist likely contain additional relevant tracks.

For a given playlist:

1. LSH retrieves a set of similar playlists.
2. Tracks from these neighbors are scored based on Jaccard
3. Tracks already present in the playlist are removed.
4. The highest-scoring tracks are returned as recommendations.

This simple neighborhood-based strategy provides a fast and effective baseline recommender system.

```
def jaccard_similarity(set1, set2):
    if not set1 and not set2:
        return 0.0
    return len(set1 & set2) / len(set1 | set2)


def recommend(pid, top_k=10, top_neighbors=100):
    """
    Recommend tracks for a given playlist ID using Jaccard similarity
for scoring.

    """

    # get MinHash signature for the query playlist
    mh = minhash_dict[pid]

    # Retrieve candidate neighbors from LSH buckets
    candidates = lsh.query(mh)

    # Remove itself if present
    candidates = [c for c in candidates if c != pid]

    # Visible tracks of query playlist (if testing)
    # or full playlist if training
    query_tracks = playlist_dict[pid]

    # ---- Score neighbors by Jaccard similarity ----
    neighbor_scores = []
    for c in candidates:
        sim = jaccard_similarity(query_tracks, playlist_dict[c])
        neighbor_scores.append((sim, c))

    # Sort neighbors by similarity
    neighbor_scores.sort(reverse=True)
    top_neighbors = neighbor_scores[:top_neighbors]
```

```
    scores = {}
    for sim, nbr in top_neighbors:
        for track in playlist_dict[nbr]:
            if track not in query_tracks:
                scores[track] = scores.get(track, 0) + sim

    # Return top-K highest scoring tracks
    ranked = sorted(scores, key=scores.get, reverse=True)
    return ranked[:top_k]



recommendations = recommend(some_pid)
recommendations
```

```
['5ByAIlEEnxYdvpnezg7HTX',
 '7kfTqGMzIHFWeBeOJALzRf',
 '4Km5HrUvYTaSUfiSGPJeQR',
 '7dZAPeA3Of5j5Vaef0DQ6M',
 '7frWizM5FgJkaUl1rXhVtO',
 '4Kd0FzFpOgIGxlBl4HXuFn',
 '5zN3VFmNhdOKxRElarvVq5',
 '04KTF78FFg8sOHC1BADqbY',
 '05VHidqx1tV6V7MsCdAIby',
 '2NBQmPrOEEjA8VbeWOQGxO']
```

```
def print_playlist_and_recommendations(pid, top_k=10):
    # --- 1. Print current playlist tracks ---
    print(f"\n=== Tracks in Playlist {pid} ===")

    playlist_tracks = playlist_dict[pid]

    playlist_df = (
        df[df['track_id'].isin(playlist_tracks)]
        [['track_id', 'track_name', 'artist_name']]
        .drop_duplicates()
    )

    for _, row in playlist_df.iterrows():
        print(f"• {row['track_name']} — {row['artist_name']}")

    # --- 2. Compute recommendations ---
    rec_ids = recommend(pid, top_k=top_k)

    print(f"\n=== Recommended Tracks for Playlist {pid} ===")

    if len(rec_ids) == 0:
        print("No recommendations found.")
        return
```

```
    rec_df = (
        df[df['track_id'].isin(rec_ids)]
        [['track_id', 'track_name', 'artist_name']]
        .drop_duplicates()
    )

    # --- 3. Print recommended tracks ---
    for _, row in rec_df.iterrows():
        print(f"• {row['track_name']} — {row['artist_name']}")

query_pid = list(playlist_dict.keys())[2]
print_playlist_and_recommendations(query_pid, top_k=10)
```

```
=== Tracks in Playlist 101861 ===
• Summer — Marshmello
• Where Are Ü Now (with Justin Bieber) - Marshmello Remix — Jack Ü
• Alone — Marshmello

=== Recommended Tracks for Playlist 101861 ===
• Moving On — Marshmello
• Silence — Marshmello
• Home — Marshmello
• Alarm - Marshmello Remix — Anne-Marie
• Fade — Alan Walker
• Waiting For Love - Marshmello Remix — Avicii
• Take It Back — Marshmello
```

Evaluation

To evaluate the effectiveness of the recommendation system, we use a common methodology. For each test playlist, we hide a portion of the tracks (treated as ground truth) and let the model predict them based only on the visible tracks.

We compute:

- **Precision@k:** How many recommended tracks are correct.
- **Recall@k:** How many of the hidden tracks were recovered.
- **MAP@k:** How well the model ranks relevant tracks.
- **NDCG@k:** Whether relevant tracks appear near the top of the recommendation list.

These metrics allow us to assess both retrieval quality and ranking performance.
This evaluation setup is consistent with standard recommender system benchmarks, including the Spotify Million Playlist Dataset Challenge.

```
test_df = pd.read_csv("test.csv")

# Group tracks for each test playlist
playlist_tracks_test = (
```

```
    test_df.groupby("playlist_id")["track_id"]
    .apply(list)
    .to_dict()
)

visible_test = {}
hidden_test = {}

hide_ratio = 0.25   # hide 25% of tracks

for pid, tracks in playlist_tracks_test.items():
    if len(tracks) < 3:
        continue

    n_hide = max(1, int(len(tracks) * hide_ratio))
    hidden = set(random.sample(tracks, n_hide))
    visible = set(tracks) - hidden

    hidden_test[pid] = hidden
    visible_test[pid] = visible

test_pids = list(visible_test.keys())

def create_minhash_from_tracks(track_set):
    m = MinHash(num_perm=512)
    for t in track_set:
        m.update(str(t))
    return m

minhash_test = {
    pid: create_minhash_from_tracks(visible)
    for pid, visible in visible_test.items()
}


def jaccard_similarity(set1, set2):
    if not set1 and not set2:
        return 0.0
    return len(set1 & set2) / len(set1 | set2)


def recommend_from_visible(pid, top_k=10, top_neighbors=200):
    mh = minhash_test[pid]
    visible = visible_test[pid]

    # get candidate neighbors from LSH
    candidates = lsh.query(mh)

    # score neighbors by exact Jaccard similarity
    neighbor_scores = []
    for c in candidates:
```

```
        base = playlist_dict[c]  # training playlist
        sim = jaccard_similarity(visible, base)
        neighbor_scores.append((sim, c))

    # sort neighbors by Jaccard score
    neighbor_scores.sort(reverse=True)
    top_neighbors = neighbor_scores[:top_neighbors]

    # aggregate track scores
    scores = {}
    for sim, c in top_neighbors:
        for track in playlist_dict[c]:
            if track not in visible:
                scores[track] = scores.get(track, 0) + sim

    # return top-K tracks
    ranked = sorted(scores, key=scores.get, reverse=True)
    return ranked[:top_k]


def precision_at_k(rec, gt, k):
    return len(set(rec[:k]) & gt) / k

def recall_at_k(rec, gt, k):
    return len(set(rec[:k]) & gt) / len(gt)

def average_precision(rec, gt, k):
    score = 0
    hits = 0
    for i, track in enumerate(rec[:k], start=1):
        if track in gt:
            hits += 1
            score += hits / i
    return score / min(len(gt), k)

def ndcg_at_k(rec, gt, k):
    dcg = 0
    for i, track in enumerate(rec[:k], start=1):
        if track in gt:
            dcg += 1 / log2(i + 1)
    ideal = min(len(gt), k)
    idcg = sum(1 / log2(i + 1) for i in range(1, ideal + 1))
    return dcg / idcg if idcg > 0 else 0


K = 10

precisions = []
recalls = []
maps = []
```

```
ndcgs = []

for pid in test_pids:
    recs = recommend_from_visible(pid, top_k=K)
    gt = hidden_test[pid]

    precisions.append(precision_at_k(recs, gt, K))
    recalls.append(recall_at_k(recs, gt, K))
    maps.append(average_precision(recs, gt, K))
    ndcgs.append(ndcg_at_k(recs, gt, K))


print("\n=== Evaluation Results (k=10) ===")
print(f"Precision@10: {np.mean(precisions):.4f}")
print(f"Recall@10:    {np.mean(recalls):.4f}")
print(f"MAP@10:       {np.mean(maps):.4f}")
print(f"NDCG@10:      {np.mean(ndcgs):.4f}")
```

```
=== Evaluation Results (k=10) ===
Precision@10: 0.0068
Recall@10:    0.0632
MAP@10:       0.0186
NDCG@10:      0.0293
```

BERT prompt-to-playlist generator

In this section, we use the transformer BERT (all-mpnet-base-v2) to encode tracks and playlists as embeddings, as well as the given user prompt.
Using FAISS similarity scores, we generate a new playlist with k tracks which are semantically close to the user prompt.

```
import pandas as pd
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
from tqdm import tqdm
```

Prompt input (ex. "UK alternative rock 80s") and size of created playlist (ex. 30)

Here we define a list of prompts to generate playlists. Usually these would contain at least the genre or an indicative period of time.

```
#prompt = "UK alternative rock 80s"
prompts = [
    "Iconic rock 80s",
    "Summer vibe 2010s",
    "Girl power",
    "study vibe",
    "songs to scream to", #a friend's recommendation
    "Kendrick Lamar"
]

playlist_size = 30

features = [

"popularity","genre","danceability","energy","key","loudness","mode",

"speechiness","acousticness","instrumentalness","liveness","valence","
tempo",
    "duration_ms","time_signature"
]
```

Importing tracks data

```
path = "./data.csv"
df = pd.read_csv(path)

#deletes invalid tracks, we need a minimum of info
df = df.dropna(subset=["track_name", "artist_name"])

#fill NaNs with "unknown" for genres and -1 for years
```

```
df["genre"] = df["genre"].fillna("unknown")
df["year"] = df["year"].fillna(-1).astype(int)

print(f'Loaded {len(df)} tracks')
```

```
Loaded 171038 tracks

C:\Users\Elio\AppData\Local\Temp\ipykernel_15768\2794900023.py:2:
DtypeWarning: Columns (2,6) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv(path)
```

Tracks as NL descriptions

To let BERT embed tracks in a meaningful way, each track in our dataset is converted to a NL description containing basic metadata, audio features and playlist context (name + position).

ex: 'Keeper of the Flame' by Miranda Lambert. From album: The Weight of These Wings. Released in 2016. [...] From playlist: Miranda Lambert.

This allows BERT to capture semantic similarity between tracks.

```
#fill other NaNs with unknown to make sure this is understandable for
BERT
def safe(x, default="unknown"):
    return x if pd.notna(x) else default

def track_to_text(row):
    """
    Converts each row (track) of the dataset into a rich natural
language
    descritption which BERT can use efficiently and embed
meaningfully.
    """
    #
    features_text = ", ".join(
        f"{col} {safe(row[col])}" for col in features if col in row
    )
    return (
        f"{safe(row['track_name'])} by {safe(row['artist_name'])}. "
        f"From album: {safe(row['album_name'])}. "
        f"Released in {safe(row['year'])}. "
        f"Position in playlist: {safe(row['position'])}. "
        f"{features_text}. "
        f"From playlist: {safe(row['playlist_name'])}."
    )

df["text_rpz"] = df.apply(track_to_text, axis=1)
texts = df["text_rpz"].tolist()
```

Creating tracks embeddings with BERT

```
#loading BERT model (17 mins)
model = SentenceTransformer("sentence-transformers/all-mpnet-base-v2")

#encoding tracks
#track_embeddings = model.encode(texts, normalize_embeddings=True,
show_progress_bar=True)
#track_embeddings = track_embeddings.astype(np.float32)

#np.save("track_embeddings.npy", track_embeddings)

track_embeddings = np.load("./embeddings/track_embeddings.npy")

dim = track_embeddings.shape[1]
print(f"Track embeddings shape: {track_embeddings.shape}")

Track embeddings shape: (171038, 768)
```

Building FAISS index

FAISS (Facebook AI Similarity Search) is an optimized library used for similarity search between vectors. In this section of our project, each track is represented as a BERT generated embedding, and FAISS finds the tracks nearest to a prompt very fast.

- faiss.IndexFlatIP(dim) is an exact index of size dim based on the Inner Product. Since our embeddings are normalised (*normalise_embeddings=True*), this inner product is the cosine similarity.
- index.add keeps those vectors in the optimised index structure.
- index.search(query, k) (used later) returns two objects: **I**, the k nearest vectors (tracks), and **D**, the similarity score.

Documentation: https://faiss.ai/index.html

```
index = faiss.IndexFlatIP(dim) #cosine similarity
index.add(track_embeddings)
```

Playlist embeddings

Since our dataset is playlist-based, and not just random tracks that BERT and FAISS would correlate, we want to encode the information the playlists give us. This could help decide which song would fit better in a new generated playlist, based on whether they are already associated in existing playlists.

To do so, we encode the playlists as the mean of the tracks embeddings. This provides a signal of track co-occurence, improving the relevance of our generated playlists.

```
#playlist_groups = df.groupby("playlist_id")
#playlist_embeddings = {}
#for pname, group in tqdm(playlist_groups, desc="Encoding playlist
embeddings"):
#    emb = model.encode(group["text_rpz"].tolist(),
normalize_embeddings=True)
#    playlist_embeddings[pname] = np.mean(emb, axis=0)  # mean of
tracks

#np.save("playlist_embeddings.npy", playlist_embeddings)

playlist_embeddings = np.load("./embeddings/playlist_embeddings.npy",
allow_pickle=True)
```

Generating new playlist

To generate a new playlist, we first encode the prompt as a BERT embedding.

Then we combine the two types of information: the similarity between the user prompt and the tracks, and the correlation of the tracks based on the playlists in which they appear. We then mix these scores with a weight alpha.

When using FAISS's index.search(), notice we use k*3. This is because we are filtering by playlists **after**, and we would not want to miss relevant tracks. We thus have $3 \times k$ candidate tracks and we will choose the k most relevant.

```
def generate_playlist(prompt: str, k: int, alpha=0.7):
    """
    Given a natural language prompt, return the k most similar tracks.

    alpha: weight btw prompt similarity & playlist correlation
    """
    prompt_embedding = model.encode(prompt,
normalize_embeddings=True).astype(np.float32)

    #faiss search
    D, I = index.search(prompt_embedding.reshape(1, -1), k*3)
    candidate_tracks = df.iloc[I[0]].copy()

    #normalized faiss scores D
    D = (D - D.min()) / (D.max() - D.min() + 1e-9) #to avoid errors
with 0
    candidate_tracks["faiss_score"] = D[0]

    #playlist correlation score (for each track)
    def correlation_score_playlist(track_row):
        score = 0.0
        #we list the playlists in which the track appears
        for pid in df[df["track_id"]==track_row["track_id"]]
["playlist_id"].unique():
```

```
            if pid in playlist_embeddings:
                #we add a score for each time the track appears in a
playlist,
                #based on a similarity score between the user prompt
and the playlist embedding
                score += np.dot(prompt_embedding,
playlist_embeddings[pid])
        return score

    #playlist correlation score
    candidate_tracks["playlist_corr"] =
candidate_tracks.apply(correlation_score_playlist, axis=1)
    #normalize playlist correlation scores to avoid bias
    pc = candidate_tracks["playlist_corr"].values
    pc_norm = (pc - pc.min()) / (pc.max() - pc.min() + 1e-9)
    candidate_tracks["playlist_corr"] = pc_norm

    #combined with prompt similarity score, with weight alpha for
prompt (ie. 1-alpha for playlist)
    candidate_tracks["combined_score"] =
alpha*candidate_tracks["faiss_score"] + (1-
alpha)*candidate_tracks["playlist_corr"]

    #sort & remove duplicates
    result = candidate_tracks.sort_values("combined_score",
ascending=False)
    result =
result.drop_duplicates(subset=["track_name","artist_name"]).head(k)
#keep k best

    df_tracks = result[[
        "playlist_name", "track_name","artist_name", "year", "genre",
"album_name"
    ]]

    track_ids = result["track_id"].tolist()

    return df_tracks, track_ids
```

Use case example:

```
playlists = []

for prompt in prompts:
    playlist, _ = generate_playlist(prompt, playlist_size)
    playlists.append( playlist )

print("\ngenerated playlists:")
for i, pl in enumerate(playlists):
```

```
    print(f"\nPrompt: {prompts[i]}")
    print(pl.to_string(index=False))


generated playlists:

Prompt: Iconic rock 80s
                         playlist_name                track_name
artist_name  year genre album_name
             80s Hits - Best of the 80s             Crocodile Rock
-1   pop
     Classic Rock Songs 60s 70s 80s 90s                   Rockstar
-1  rock
             80s Hits - Best of the 80s             I Remember You
-1  rock
             80s Hits - Best of the 80s                   Running
Retrofile    -1  rock
     Classic Rock Songs 60s 70s 80s 90s                Photograph
-1  rock
             80s Hits - Best of the 80s      Jump - 2015 Remaster
-1  rock
     Classic Rock Songs 60s 70s 80s 90s    1979 - Remastered 2012
-1  rock
     Classic Rock Songs 60s 70s 80s 90s  Rockstar - 2020 Remaster
-1  rock
         Rock Classics (80s, 90s 2000s)               Rock Of Ages
-1  rock
             80s Hits - Best of the 80s             Crocodile Rock
Elton John    -1  rock
Top 100 Rock Classics: 80s, 90s & 2000s                 89 and 24
-1  rock
             80s Hits - Best of the 80s               Poster Child
Retrofile    -1  rock
     Classic Rock Songs 60s 70s 80s 90s  Rock and Roll - Remaster
-1  rock
                                 smooooth                        80s
Berhana  2016 chill    Berhana
             80s Hits - Best of the 80s       The Final Countdown
Europe    -1  rock
             80s Hits - Best of the 80s               Poster Child
-1  rock
     Classic Rock Songs 60s 70s 80s 90s           Info Technology
-1  rock
             80s Hits - Best of the 80s               Expectations
-1  rock
     Classic Rock Songs 60s 70s 80s 90s                      1985
-1  rock
         Rock Classics (80s, 90s 2000s)                Whatsername
-1  rock
             80s Hits - Best of the 80s               Expectations
```

```
Retrofile    -1  rock
             80s Hits - Best of the 80s All the Time in the World
Retrofile    -1  rock
             80s Hits - Best of the 80s                Two Words
Retrofile    -1  rock
     Classic Rock Songs 60s 70s 80s 90s      1969 - 2019 Remaster
-1  rock
         Rock Classics (80s, 90s 2000s)              Rock Of Ages Def
Leppard    -1  rock
             80s Hits - Best of the 80s                Rebel Yell
-1  rock
     Classic Rock Songs 60s 70s 80s 90s      Old Time Rock & Roll
-1  rock
     Classic Rock Songs 60s 70s 80s 90s         How You Remind Me
-1  rock
     Classic Rock Songs 60s 70s 80s 90s              I Wanna Rock
-1  rock
             80s Hits - Best of the 80s         Rivers of Babylon
-1  rock

Prompt: Summer vibe 2010s
                                                        playlist_name
track_name                                        artist_name  year
genre                                             album_name
                                                          summer vibes
Summer                                       Calvin Harris  2014
dance                                                Motion
                                                       Summer Vibes
Feels Like Summer                                           Weezer
2017   alt-rock                                      Pacific Daydream
                                                Pop Hits 2000s – 2025
Summer                                                        -1
pop
                                                     pop girlies□□□
Cool for the Summer
-1        pop
                                                                Chill Rap
Summertime Music                                     Shwayze & Cisco  2011
indie-pop                                          Island In The Sun
                                                       Skrillex – Recess
Summer                                             Marshmello  2016
dance                                                    Joytime
                                                               car jamz
The Summer                                            Citizen    -1
unknown                                                  Youth
                                                                  spain
Summer Song                      The Karminsky Experience Inc.    -1
unknown                                               Snapshot
                                                      Kendrick Lamar
```

Summertime - Single Edit DJ Jazzy Jeff & The Fresh Prince -1 unknown The Very Best Of D.J. Jazzy Jeff & The Fresh Prince

erging Year Of Summer Wildstylez 2012 hardstyle Hardstyle The Ultimate Collection - Best Of 2012

Focus Suddenly Summer - Original Mix Armin van Buuren 2012 edm Suddenly Summer

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Summer Travel Lofi Fruits Music, Calisson, Chill Fruits Music -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to High Summer Lofi Fruits Music, Chill Fruits Music -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Summer's End Lofi Fruits Music, Chill Fruits Music -1 hip hop

Jill Scott Spring Summer Feeling Jill Scott 2004 funk Beautifully Human: Words and Sounds Vol. 2

summer tunes Sounds Of Summer Dierks Bentley 2014 country

RISER Hiphop Summer '25 -1 hip hop

HOUSTON Summertime Slim Thug 2012 hip-hop

Summertime Country Single For The Summer Sam Hunt -1 unknown

Montevallo Hiphop Summer '24 -1 hip hop

Summer Time Summertime Sammy Adams 2012 indie-pop

Only One Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to High Summer -1 hip hop

Electronica Indian Summer - Original Mix Guy Gerber 2014 deep-house

11 11

Good Country Summer Cassadee Pope 2016

```
country                                                    Summer EP
                                                  Det fandme hip hop!
Dear Summer                                                        -1
hip hop
                                                  Rock Techno Remixes
Spring (The Four Seasons) - Techno Mix                        LANNÉ,
Blaze U, Charles B    -1       rock

                                                                Summer
Boys Of The Summer                                          Mike Stud
-1    unknown                                                 Closer
                                                               Rap
Summer On Smash                                              Nas  2012
hardcore                                          Life Is Good
                                                                summer
Summer Skin                                Death Cab for Cutie  2005
indie-pop                                                    Plans
                                                                karin
The Height Of Summer                                        The Knife
2010    electro                                 Tomorrow In A Year

Prompt: Girl power
                                                                playlist_name
track_name                                 artist_name  year    genre
album_name
                                                                 goosebumps
Power                                 Rich Homie Quan    -1  unknown
No Label Vol. 2
                                                  90s Dance Hits (Top 100)
The Power                                                      -1      pop

                                                           girls girls girls
Pacify Her                                 Melanie Martinez  2015  electro
Cry Baby
                                       Hip Hop 90's & 2000's - Classics
POWER                                                          -1  hip hop

                                                         Rock Techno Remixes
Power                                                          -1     rock

                                                                      Hiphop
Power (feat. ElCamino)
-1  hip hop
                                                                    Gorillaz
We Got The Power (feat. Jehnny Beth)
Gorillaz    -1  unknown                        Humanz
                                                       HIP-HOP CLASSICS
Fight The Power                                                 -1
hip hop
                                                                      Hiphop
```

Power (feat. ElCamino) Benny The Butcher, 38 Spesh, V Don, Elcamino
-1 hip hop
Top Pop Hits 2015-2025
GIRLS -1 pop

Concert
Girls Miranda Lambert 2014 country
Platinum
Børne rock
Flower Power Tøj (Ding Dong Bama Lama Sing Song)
-1 rock
ROCK MUSIK
Girls, Girls, Girls -1
rock
Luke Bryan Concert
Power Of A Woman Lee Brice 2010
country Love Like Crazy
90s Dance Hits (Top 100)
The Power SNAP! -1 pop

AUGUST 2017
Girls Do What They Want The Maine
2008 emo Can't Stop Won't Stop
Grace
Girl In A Country Song Maddie & Tae
-1 unknown Start Here
Top Pop Hits 2015-2025
Girls Need Love (with Drake) - Remix
-1 pop
Top Pop Hits 2015-2025
GIRLS The Kid LAROI -1 pop

queens
Money Power Glory Lana Del Rey 2014
pop Ultraviolence
femme
For the Love of Holy Ghost Power Girl
Pusher 2015 club Groceries EP 2
pop girlies□□□
How You Get The Girl (Taylor's Version)
-1 pop
Wiz
Da Power Juicy J -1 unknown
TGOD Mafia: Rude Awakening
Uplifting
There Is Power Lincoln Brewster 2014 alt-
rock Oxygen
girls girls girls
Control Halsey 2015 electro
BADLANDS

```
                                                    girls girls girls
Girls Like Girls                                 Hayley Kiyoko  2015
electro This Side of Paradise - EP
                                                                miranda
Girls Like Us                                 Pistol Annies  2013
country                   Annie Up
                                                 Det fandme hip hop!
Position Of Power                                                -1
hip hop
BEST CLASSIC ROCK  Greatest Hits of All Time (Rock & Soft Rock Music)
The Power Of Love                                                -1
rock
                                                                country
God Made Girls                                    RaeLynn    -1
unknown             God Made Girls

Prompt: study vibe
                                                    playlist_name
track_name                            artist_name  year   genre
album_name
                     Hiphop Træningsmusik (opdateres hver onsdag)
Vibe                                          -1 hip hop

                     Hiphop Træningsmusik (opdateres hver onsdag)
Vibe                                    Guè    -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Calm Study Ambience  Lofi Fruits Music, Chill Fruits Music    -1 hip
hop
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Aroma  Lofi Fruits Music, Chill Fruits Music    -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Calm Study Ambience                                          -1 hip
hop
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Lofi Background Chilling  Lofi Fruits Music, Chill Fruits Music    -1
hip hop
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Refound Motivation  Lofi Fruits Music, Chill Fruits Music    -1 hip
hop
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Study With Me  Lofi Fruits Music, Chill Fruits Music    -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Deep Think  Lofi Fruits Music, Chill Fruits Music    -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Study Break  Lofi Fruits Music, Chill Fruits Music    -1 hip hop
```

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Aroma -1 hip hop

vibez Pass the Vibes Donnie Trumpet & The Social Experiment 2015 soul Surf
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to STAY Lofi Fruits Music, Chill Fruits Music -1 hip hop

Vibe Bitch, Don’t Kill My Vibe - Remix Kendrick Lamar 2012 hip-hop good kid, m.A.A.d city
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Mount Fuji Vibe -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Studying Lofi Beats Lofi Fruits Music, Chill Fruits Music -1 hip hop
It's A Vibe It's A Vibe 2 Chainz 2017 hip-hop Pretty Girls Like Trap Music
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Calming Chill Lofi Fruits Music, Chill Fruits Music -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Study Break -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Photograph Lofi Fruits Music, Chill Fruits Music -1 hip hop

The Best Vibe Mick Jenkins -1 unknown The Water (S)
vibes Bitch, Don’t Kill My Vibe Kendrick Lamar 2012 hip-hop good kid, m.A.A.d city
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Motivational Speech Lofi Fruits Music, Chill Fruits Music -1 hip hop
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Show Me How Lofi Fruits Music, Chill Fruits Music -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Research Lofi Fruits Music, Chill Fruits Music -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to I Feel It Coming -1 hip hop

Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to Warm Breeze Lofi Fruits Music, Chill Fruits Music -1 hip hop

```
                                                                  vibes
Feels                          Calvin Harris  2017   dance       Funk
Wav Bounces Vol.1
Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
Motivational Speech                                             -1 hip
hop
                                                        boats and hoes
Good Vibe                                 Lil Haiti    -1 unknown
The New Haiti

Prompt: songs to scream to
                                      playlist_name
track_name                     artist_name  year   genre
album_name
                                        ROCK MUSIK
Let Me Hear You Scream                  Ozzy Osbourne    -1    rock

                        The Best of Michael Jackson
Scream Michael Jackson, Janet Jackson    -1    rock

                               Pop 2000-2010 Bangers
Scream & Shout      will.i.am, Britney Spears    -1     pop

                               Pop 2000-2010 Bangers
Scream                                        -1     pop

                                        ROCK MUSIK
Let Me Hear You Scream                                   -1    rock

                               Pop 2000-2010 Bangers
Scream & Shout                                    -1     pop

                                         old songs
Scream       High School Musical Cast  2008   dance High School
Musical 3: Senior Year
                               Pop 2000-2010 Bangers
Scream                            USHER    -1     pop

                                    Britney Spears
Scream & Shout                      will.i.am    -1 unknown
#willpower
                                                 BFMV
Scream              Avenged Sevenfold  2007   metal
Avenged Sevenfold
                                                   ML
Scream                               Usher    -1 unknown
Looking 4 Myself
                                   Pop FM’s bedste
Shout                Tears For Fears    -1     pop
```

ROCK MUSIK
Choking On Your Screams -1 rock

SONGS WE ALL KNOW😀
Roar Katy Perry -1 pop

Hard Rock /Metal
Shout At The Devil Mötley Crüe -1 rock

♫ Popular Techno Remixes – Rave & Club Bangers□ □
Scream & Shout NIVEK, JKRS -1 pop

ROCK MUSIK
Choking On Your Screams Motörhead -1 rock

Alesana This Is Usually The
Part Where People Scream Alesana 2008 emo
Where Myth Fades To Legend
motionless in white
Scream New Years Day 2015 emo
Malevolence
Hard Rock /Metal
Screaming for Vengeance Judas Priest -1 rock

Disturbed
Shout 2000 Disturbed 2000 metal
The Sickness
cry
Glory And Gore Lorde 2013 pop
Pure Heroine
Drowning
Scream Halestorm 2015 metal
Into The Wild Life
Pop FM’s bedste
Shout -1 pop

Classic Rock Songs 60s 70s 80s 90s
Cry Baby Janis Joplin -1 rock

Hard Rock /Metal
Stand Up and Shout Dio -1 rock

pop girlies□□□
What the Hell Avril Lavigne -1 pop

Hard Rock /Metal
Stand Up and Shout -1 rock

new In

```
Case of Emergency, Dial 411           Sleeping With Sirens  2010
emo  With Ears To See And Eyes To Hear
          Top 100 Rock Classics: 80s, 90s & 2000s
Rebel Yell                      Billy Idol    -1    rock


Prompt: Kendrick Lamar
                          playlist_name                track_name
artist_name  year   genre             album_name
                                  kenny untitled 07 | 2014 - 2016
Kendrick Lamar  2016 hip-hop   untitled unmastered.
                         kendrick lamar             Rolling Stone
Black Hippy  2011 hip-hop               Setbacks
  Kendrick Lamar – untitled unmastered. untitled 03 | 05.28.2013.
Kendrick Lamar  2016 hip-hop   untitled unmastered.
                          KING KENDRICK untitled 06 | 06.30.2014.
Kendrick Lamar  2016 hip-hop   untitled unmastered.
  Kendrick Lamar – untitled unmastered.                          u
Kendrick Lamar  2015 hip-hop    To Pimp A Butterfly
                          KING KENDRICK                King Kunta
Kendrick Lamar  2015 hip-hop    To Pimp A Butterfly
Kendrick Lamar - good kid, m.A.A.d city                  good kid
Kendrick Lamar  2012 hip-hop good kid, m.A.A.d city
  Kendrick Lamar – untitled unmastered. untitled 05 | 09.21.2014.
Kendrick Lamar  2016 hip-hop   untitled unmastered.
  Kendrick Lamar – untitled unmastered. untitled 02 | 06.23.2014.
Kendrick Lamar  2016 hip-hop   untitled unmastered.
                          KING KENDRICK                DUCKWORTH.
Kendrick Lamar  2017 hip-hop                  DAMN.
                        Kendrick Lamar             Hood Politics
Kendrick Lamar  2015 hip-hop    To Pimp A Butterfly
                         kendrick lamar                          i
Kendrick Lamar  2014 hip-hop                       i
```

Evaluation methods

It is very hard to evaluate a recommendation system accurately, especially ours, since there is no "true" perfectly generated playlist. We have no ground-truth for subjective similarity to evaluate our performance.

What we can do is evaluate playlist **reconstruction**, using several metrics.
Source: https://www.evidentlyai.com/ranking-metrics/evaluating-recommender-systems

1. Recall@k

$$Recall@k = \frac{\text{nb of 'real' tracks found in the k returned tracks}}{\text{total nb of real tracks}}$$

Since we generate a list with no specific order, this gives us a simple and robust objective measure of the playlist reconstruction.

```
# garder uniquement les playlist_id qui sont des entiers
df = df[df["playlist_id"].apply(lambda x: str(x).isdigit())]
df["playlist_id"] = df["playlist_id"].astype(int)


#filtering playlists with between 20-40 tracks
#this allows to evaluate with a minimum of material and consistency
playlist_sizes = df.groupby("playlist_id")["track_id"].count()
valid_pids = playlist_sizes[(playlist_sizes >= 20) & (playlist_sizes
<= 40)].index.tolist()

print(f"{len(valid_pids)} retained playlists for evaluation (20-40
tracks)")

np.random.seed(42)
selected_playlists = np.random.choice(valid_pids, size=min(10,
len(valid_pids)), replace=False)

#returns a list of the predicted track_ids (generated playlist),
#using the playlist name as a prompt for evaluating purposes
def predict_playlist_from_id(pid, alpha=0.7):
    playlist_df = df[df["playlist_id"] == pid]

    if playlist_df.empty:
        return [], None

    prompt = playlist_df["playlist_name"].iloc[0]
    k = len(playlist_df)

    df_playlist, ids = generate_playlist(prompt, k=k, alpha=alpha)

    return df_playlist, ids
```

```
1403 retained playlists for evaluation (20-40 tracks)
```

```
def recall_at_k(true_ids, pred_ids, k):
    true_set = set(true_ids)
    pred_set = set(pred_ids[:k])
    return len(true_set & pred_set) / len(true_set)
```

Additional Metrics

To obtain a more complete evaluation, we compute:

Precision@k: How many of the returned tracks were correct?

MAP@k: Rewars correct predictions at earlier ranks

nDCG@k: Measures ranking quality with positional discounting.

These metrics give a more nuanced analysis of playlist reconstruction quality.

```
def precision_at_k(recommended, ground_truth, k):
    recommended_k = recommended[:k]
    hits = sum([1 for r in recommended_k if r in ground_truth])
    return hits / k


def mean_average_precision_at_k(recommended, ground_truth, k):
    recommended_k = recommended[:k]
    score = 0.0
    hits = 0

    for i, r in enumerate(recommended_k, start=1):
        if r in ground_truth:
            hits += 1
            score += hits / i

    if len(ground_truth) == 0:
        return 0.0

    return score / min(len(ground_truth), k)


def ndcg_at_k(recommended, ground_truth, k):
    recommended_k = recommended[:k]
    dcg = 0.0

    for i, r in enumerate(recommended_k, start=1):
        if r in ground_truth:
            dcg += 1 / np.log2(i + 1)

    #ideal dcg
    ideal_hits = min(len(ground_truth), k)
    idcg = sum([1 / np.log2(i + 1) for i in range(1, ideal_hits + 1)])

    return dcg / idcg if idcg > 0 else 0.0
```

Full evaluation loop

```
results = []

for pid in tqdm(valid_pids, desc="Evaluating model metrics"):

    playlist_df = df[df["playlist_id"] == pid]

    if playlist_df.empty:
        continue

    true_ids = playlist_df["track_id"].tolist()
    k = 10
```

```
    _, pred_ids = predict_playlist_from_id(pid, alpha=0.7)

    res = {
        "playlist_id": pid,
        "playlist_name": playlist_df["playlist_name"].iloc[0],
        "num_tracks": k,
        "recall@k": recall_at_k(true_ids, pred_ids, k),
        "precision@k": precision_at_k(pred_ids, true_ids, k),
        "map@k": mean_average_precision_at_k(pred_ids, true_ids, k),
        "ndcg@k": ndcg_at_k(pred_ids, true_ids, k)
    }

    results.append(res)


#displaying some results
for r in results[0:10]:
    print("\n" + "="*70)
    print(f"Playlist: {r['playlist_name']} (id={r['playlist_id']},
{r['num_tracks']} tracks)")
    print(f"Recall@k:    {r['recall@k']:.3f}")
    print(f"Precision@k: {r['precision@k']:.3f}")
    print(f"MAP@k:       {r['map@k']:.3f}")
    print(f"NDCG@k:      {r['ndcg@k']:.3f}")

Evaluating model metrics: 100%|██████████| 1403/1403 [23:17<00:00,
1.00it/s]


======================================================================
Playlist: summer 2k17 (id=1043, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000

======================================================================
Playlist: The Glitch Mob (id=1192, 10 tracks)
Recall@k:    0.120
Precision@k: 0.300
MAP@k:       0.115
NDCG@k:      0.258

======================================================================
Playlist: country (id=1240, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000
```

```
================================================================================
Playlist: country (id=1915, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000

================================================================================
Playlist: starbucks  (id=2976, 10 tracks)
Recall@k:    0.154
Precision@k: 0.400
MAP@k:       0.115
NDCG@k:      0.278

================================================================================
Playlist: Country (id=3029, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000

================================================================================
Playlist: best of country (id=4181, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000

================================================================================
Playlist: Country (id=5005, 10 tracks)
Recall@k:    0.025
Precision@k: 0.100
MAP@k:       0.017
NDCG@k:      0.078

================================================================================
Playlist: ignite (id=5193, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000

================================================================================
Playlist: Upbeat (id=6771, 10 tracks)
Recall@k:    0.000
Precision@k: 0.000
MAP@k:       0.000
NDCG@k:      0.000
```

Global average scores

```
print("Average performance over tested playlists:\n")

print("Recall@10:    ", np.mean([r["recall@k"] for r in results]))
print("Precision@10: ", np.mean([r["precision@k"] for r in results]))
print("MAP@10:       ", np.mean([r["map@k"] for r in results]))
print("NDCG@10:      ", np.mean([r["ndcg@k"] for r in results]))
```

```
Average performance over tested playlists:

Recall@10:     0.03022200500049308
Precision@10:  0.08139700641482539
MAP@10:        0.04421146862166107
NDCG@10:       0.08355978166389104
```

2. LLM-as-a-judge

Beyond quantitative metrics, we can ask an LLM to evaluate thematic coherence, genre consistency, mood alignement and subjective musical similarity.
It provides a complementary qualitative angle.

We follow this example: https://towardsdatascience.com/llm-as-a-judge-what-it-is-why-it-works-and-how-to-use-it-to-evaluate-ai-models/
With the framework found on github:
https://github.com/PieroPaialungaAI/LLMAsAJudge/tree/main

```
from openai import OpenAI
import os
from llm_judge import LLMJudge

os.environ["OPENAI_API_KEY"] = "secret_key" #im hiding this

client = OpenAI()

judge_role = """You are an expert evaluator of music playlist
recommendation systems.
You have 10 years of experience in playlist generation systems and
understand the wide
choice of possible relevant songs. You understand that the dataset may
contain missing
or unknown metadata (notably values marked as -1 or 'unknown'), and
that tracks span
from 2000 to 2023. Ignore missing metadata like 'unknown' genre or -1
year. Focus on
whether the tracks match the style, mood, and era indicated by the
user prompt or
original playlist name. Your job is to assess whether the generated
playlists
are accurate and appropriate given the user prompt."""
```

```python
# Define the evaluation task
evaluation_task = """For each generated playlist, you must:
1. Read the user prompt and understand its expectations
2. Examine the tracks in the generated playlist
3. Determine if the playlist is coherent with regard to the user
prompt
4. Provide a quality score (0-100), verdict, and detailed reasoning"""

# Define evaluation criteria
evaluation_criteria = """
A generated playlist is excellent if:
- The genres of the tracks are coherent with the prompt
- The mood and vibe are coherent
- There is a temporal and style coherence (if the prompt mentions era
or style)
- There is some diversity of artists, albums, energy...
- It is appropriate overall"""

# Valid verdicts
verdicts = ["Excellent", "Good", "Average", "Poor"]

playlist_judge = LLMJudge(
    llm_client= client,
    role= judge_role,
    task_description= evaluation_task,
    evaluation_criteria= evaluation_criteria,
    valid_verdicts= verdicts,
    model_name= "gpt-4o-mini",
    temperature= 0.2
)

#we need a function to create the playlist object we will give to the
llm
def format_playlist_llm(prompt, playlist):

    def safe_year(year):
        return "unknown" if year == -1 else year

    text = f"Given prompt: {prompt}\n"
    text += f"Number of tracks: {len(playlist)}\n\n"

    text += "Tracks:\n"

    for i, row in playlist.iterrows():
        track_name = (row.get("track_name"))
        artists = (row.get("artist_name"))
        genre = (row.get("genre"))
        year = (row.get("year"))
        original_playlist = (row.get("playlist_name"))
```

```
        track_line = f"- '{track_name}' by {artists}"
        track_line += f" — Genre: {genre}"
        track_line += f" year: {safe_year(year)}"
        track_line += f" fetched from playlist: {original_playlist}"

        text += track_line + "\n"

    text += (
        "\nNotes: Some metadata may be missing or labelled as
'unknown'. "
        "This is normal for this dataset and should NOT influence
quality judgment.\n"
    )

    return {"input": prompt, "model_output": text}

generated_playlists = []
for prompt in prompts:
    df_generated, _ = generate_playlist(prompt, k=30)
    formatted = format_playlist_llm(prompt, df_generated)

    generated_playlists.append( formatted )

print("=" *50)
print("Judging generated playlists")
print("=" *50)

for i, pred in enumerate(generated_playlists, 1):
    print(f"\n### Case {i} ###")
    print(f"Prompt: {pred['input']}")
    print(f"Generated playlist: {pred['model_output']}")


    judgment = playlist_judge.judge_single(
        input_text=pred['input'],
        model_output=pred['model_output']
    )

    print(f"\n{'Judge Verdict:':<20} {judgment.verdict}")
    print(f"{'Quality Score:':<20} {judgment.score}/100")
    print(f"{'Confidence:':<20} {judgment.confidence}%")
    print(f"\nReasoning: {judgment.reasoning}")

    if judgment.notes:
        print(f"Notes: {judgment.notes}")

    print("\n" + "=" * 100)
```

```
==================================================
Judging generated playlists
==================================================
```

Case 1
Prompt: Iconic rock 80s
Generated playlist: Given prompt: Iconic rock 80s
Number of tracks: 30

Tracks:
- 'Crocodile Rock' by — Genre: pop year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Rockstar' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'I Remember You' by — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Running' by Retrofile — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Photograph' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Jump - 2015 Remaster' by — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- '1979 - Remastered 2012' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Rockstar - 2020 Remaster' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Rock Of Ages' by — Genre: rock year: unknown fetched from playlist: Rock Classics (80s, 90s 2000s)
- 'Crocodile Rock' by Elton John — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- '89 and 24' by — Genre: rock year: unknown fetched from playlist: Top 100 Rock Classics: 80s, 90s & 2000s
- 'Poster Child' by Retrofile — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Rock and Roll - Remaster' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- '80s' by Berhana — Genre: chill year: 2016 fetched from playlist: smooooth
- 'The Final Countdown' by Europe — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Poster Child' by — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Info Technology' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Expectations' by — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- '1985' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Whatsername' by — Genre: rock year: unknown fetched from playlist: Rock Classics (80s, 90s 2000s)
- 'Expectations' by Retrofile — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s

- 'All the Time in the World' by Retrofile — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Two Words' by Retrofile — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- '1969 - 2019 Remaster' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Rock Of Ages' by Def Leppard — Genre: rock year: unknown fetched from playlist: Rock Classics (80s, 90s 2000s)
- 'Rebel Yell' by — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s
- 'Old Time Rock & Roll' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'How You Remind Me' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'I Wanna Rock' by — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Rivers of Babylon' by — Genre: rock year: unknown fetched from playlist: 80s Hits - Best of the 80s

Notes: Some metadata may be missing or labelled as 'unknown'. This is normal for this dataset and should NOT influence quality judgment.

Judge Verdict: Partially Correct
Quality Score: 70.0/100
Confidence: 80.0%

Reasoning: The playlist aligns with the user prompt of 'Iconic rock 80s' in that it predominantly features rock tracks from the 80s era. However, there are inconsistencies, such as the inclusion of 'Crocodile Rock' by Elton John, which is more pop than rock, and '80s' by Berhana, which is a chill track from 2016 and does not fit the 80s rock theme. Additionally, while many tracks are indeed rock, the diversity of artists is somewhat limited, with multiple tracks by Retrofile and some repeated titles. Overall, the playlist captures the essence of 80s rock but lacks some precision and diversity, leading to a score that reflects its partial correctness.
Notes: To improve, the playlist could exclude non-rock tracks and ensure a wider variety of artists to enhance the overall quality.

==========

Case 2
Prompt: Summer vibe 2010s
Generated playlist: Given prompt: Summer vibe 2010s
Number of tracks: 30

Tracks:
- 'Summer' by Calvin Harris — Genre: dance year: 2014 fetched from

playlist: summer vibes
- 'Feels Like Summer' by Weezer — Genre: alt-rock year: 2017 fetched from playlist: Summer Vibes
- 'Summer' by — Genre: pop year: unknown fetched from playlist: Pop Hits 2000s — 2025
- 'Cool for the Summer' by — Genre: pop year: unknown fetched from playlist: pop girlies□□□
- 'Summertime Music' by Shwayze & Cisco — Genre: indie-pop year: 2011 fetched from playlist: Chill Rap
- 'Summer' by Marshmello — Genre: dance year: 2016 fetched from playlist: Skrillex — Recess
- 'The Summer' by Citizen — Genre: unknown year: unknown fetched from playlist: car jamz
- 'Summer Song' by The Karminsky Experience Inc. — Genre: unknown year: unknown fetched from playlist: spain
- 'Summertime - Single Edit' by DJ Jazzy Jeff & The Fresh Prince — Genre: unknown year: unknown fetched from playlist: Kendrick Lamar
- 'Year Of Summer' by Wildstylez — Genre: hardstyle year: 2012 fetched from playlist: erging
- 'Suddenly Summer - Original Mix' by Armin van Buuren — Genre: edm year: 2012 fetched from playlist: Focus
- 'Summer Travel' by Lofi Fruits Music, Calisson, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'High Summer' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Summer's End' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Spring Summer Feeling' by Jill Scott — Genre: funk year: 2004 fetched from playlist: Jill Scott
- 'Sounds Of Summer' by Dierks Bentley — Genre: country year: 2014 fetched from playlist: summer tunes
- 'Summer '25' by — Genre: hip hop year: unknown fetched from playlist: Hiphop
- 'Summertime' by Slim Thug — Genre: hip-hop year: 2012 fetched from playlist: HOUSTON
- 'Single For The Summer' by Sam Hunt — Genre: unknown year: unknown fetched from playlist: Country
- 'Summer '24' by — Genre: hip hop year: unknown fetched from playlist: Hiphop
- 'Summertime' by Sammy Adams — Genre: indie-pop year: 2012 fetched from playlist: Summer Time
- 'High Summer' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Indian Summer - Original Mix' by Guy Gerber — Genre: deep-house year: 2014 fetched from playlist: Electronica

- 'Summer' by Cassadee Pope — Genre: country year: 2016 fetched from playlist: Good Country
- 'Dear Summer' by — Genre: hip hop year: unknown fetched from playlist: Det fandme hip hop!
- 'Spring (The Four Seasons) - Techno Mix' by LANNÉ, Blaze U, Charles B — Genre: rock year: unknown fetched from playlist: Rock Techno Remixes
- 'Boys Of The Summer' by Mike Stud — Genre: unknown year: unknown fetched from playlist: Summer
- 'Summer On Smash' by Nas — Genre: hardcore year: 2012 fetched from playlist: Rap
- 'Summer Skin' by Death Cab for Cutie — Genre: indie-pop year: 2005 fetched from playlist: summer
- 'The Height Of Summer' by The Knife — Genre: electro year: 2010 fetched from playlist: karin

Notes: Some metadata may be missing or labelled as 'unknown'. This is normal for this dataset and should NOT influence quality judgment.

Judge Verdict: Partially Correct
Quality Score: 65.0/100
Confidence: 80.0%

Reasoning: The playlist generally aligns with the prompt of 'Summer vibe 2010s' as it includes many tracks that are either explicitly about summer or have a summery feel. However, there are several tracks with unknown metadata, which detracts from the overall quality and coherence of the playlist. Additionally, while there is a good representation of genres like dance, pop, and hip-hop, the presence of tracks from outside the 2010s era, such as 'Spring (The Four Seasons) - Techno Mix' (year unknown) and 'Summer Skin' by Death Cab for Cutie (2005), disrupts the temporal coherence. The diversity of artists is somewhat present, but the playlist could benefit from more well-known summer hits from the 2010s to enhance its appeal. Overall, while the playlist captures some summer vibes, it lacks consistency in era and quality due to the unknown metadata and some out-of-era tracks.
Notes: Consider replacing tracks with unknown metadata or from outside the 2010s to improve coherence and quality.

==

Case 3
Prompt: Girl power
Generated playlist: Given prompt: Girl power
Number of tracks: 30

Tracks:
- 'Power' by Rich Homie Quan — Genre: unknown year: unknown fetched

from playlist: goosebumps
- 'The Power' by – Genre: pop year: unknown fetched from playlist: 90s Dance Hits (Top 100)
- 'Pacify Her' by Melanie Martinez – Genre: electro year: 2015 fetched from playlist: girls girls girls
- 'POWER' by – Genre: hip hop year: unknown fetched from playlist: Hip Hop 90's & 2000's - Classics
- 'Power' by – Genre: rock year: unknown fetched from playlist: Rock Techno Remixes
- 'Power (feat. ElCamino)' by – Genre: hip hop year: unknown fetched from playlist: Hiphop
- 'We Got The Power (feat. Jehnny Beth)' by Gorillaz – Genre: unknown year: unknown fetched from playlist: Gorillaz
- 'Fight The Power' by – Genre: hip hop year: unknown fetched from playlist: HIP-HOP CLASSICS
- 'Power (feat. ElCamino)' by Benny The Butcher, 38 Spesh, V Don, Elcamino – Genre: hip hop year: unknown fetched from playlist: Hiphop
- 'GIRLS' by – Genre: pop year: unknown fetched from playlist: Top Pop Hits 2015-2025
- 'Girls' by Miranda Lambert – Genre: country year: 2014 fetched from playlist: Concert
- 'Flower Power Tøj (Ding Dong Bama Lama Sing Song)' by – Genre: rock year: unknown fetched from playlist: Børne rock
- 'Girls, Girls, Girls' by – Genre: rock year: unknown fetched from playlist: ROCK MUSIK
- 'Power Of A Woman' by Lee Brice – Genre: country year: 2010 fetched from playlist: Luke Bryan Concert
- 'The Power' by SNAP! – Genre: pop year: unknown fetched from playlist: 90s Dance Hits (Top 100)
- 'Girls Do What They Want' by The Maine – Genre: emo year: 2008 fetched from playlist: AUGUST 2017
- 'Girl In A Country Song' by Maddie & Tae – Genre: unknown year: unknown fetched from playlist: Grace
- 'Girls Need Love (with Drake) - Remix' by – Genre: pop year: unknown fetched from playlist: Top Pop Hits 2015-2025
- 'GIRLS' by The Kid LAROI – Genre: pop year: unknown fetched from playlist: Top Pop Hits 2015-2025
- 'Money Power Glory' by Lana Del Rey – Genre: pop year: 2014 fetched from playlist: queens
- 'For the Love of Holy Ghost Power' by Girl Pusher – Genre: club year: 2015 fetched from playlist: femme
- 'How You Get The Girl (Taylor's Version)' by – Genre: pop year: unknown fetched from playlist: pop girlies□□□
- 'Da Power' by Juicy J – Genre: unknown year: unknown fetched from playlist: Wiz
- 'There Is Power' by Lincoln Brewster – Genre: alt-rock year: 2014 fetched from playlist: Uplifting
- 'Control' by Halsey – Genre: electro year: 2015 fetched from playlist: girls girls girls

- 'Girls Like Girls' by Hayley Kiyoko — Genre: electro year: 2015 fetched from playlist: girls girls girls
- 'Girls Like Us' by Pistol Annies — Genre: country year: 2013 fetched from playlist: miranda
- 'Position Of Power' by — Genre: hip hop year: unknown fetched from playlist: Det fandme hip hop!
- 'The Power Of Love' by — Genre: rock year: unknown fetched from playlist: BEST CLASSIC ROCK Greatest Hits of All Time (Rock & Soft Rock Music)
- 'God Made Girls' by RaeLynn — Genre: unknown year: unknown fetched from playlist: country

Notes: Some metadata may be missing or labelled as 'unknown'. This is normal for this dataset and should NOT influence quality judgment.

Judge Verdict: Partially Correct
Quality Score: 65.0/100
Confidence: 80.0%

Reasoning: The playlist generated in response to the prompt 'Girl power' includes a variety of tracks that touch on themes of empowerment and femininity, which aligns with the concept of 'girl power'. However, the coherence of the genres is mixed; while there are pop and hip hop tracks that are generally associated with empowerment, there are also country and rock tracks that do not fit as well with the theme. Additionally, the presence of many tracks with unknown metadata raises concerns about the overall quality and relevance of the playlist. The diversity of artists is present, but the overall execution lacks a strong focus on the 'girl power' theme, leading to a score that reflects a decent but not excellent alignment with the prompt.
Notes: To improve, the playlist could focus more on contemporary female artists and tracks that explicitly convey empowerment messages, ensuring a stronger thematic coherence.

==========

Case 4
Prompt: study vibe
Generated playlist: Given prompt: study vibe
Number of tracks: 30

Tracks:
- 'Vibe' by — Genre: hip hop year: unknown fetched from playlist: Hiphop Træningsmusik (opdateres hver onsdag)
- 'Vibe' by Guè — Genre: hip hop year: unknown fetched from playlist: Hiphop Træningsmusik (opdateres hver onsdag)
- 'Calm Study Ambience' by Lofi Fruits Music, Chill Fruits Music —

Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Aroma' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Calm Study Ambience' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Lofi Background Chilling' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Refound Motivation' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Study With Me' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Deep Think' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Study Break' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Aroma' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Pass the Vibes' by Donnie Trumpet & The Social Experiment — Genre: soul year: 2015 fetched from playlist: vibez
- 'STAY' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Bitch, Don’t Kill My Vibe - Remix' by Kendrick Lamar — Genre: hip-hop year: 2012 fetched from playlist: Vibe
- 'Mount Fuji Vibe' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Studying Lofi Beats' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'It's A Vibe' by 2 Chainz — Genre: hip-hop year: 2017 fetched from playlist: It's A Vibe
- 'Calming Chill' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Study Break' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Photograph' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to

- 'Vibe' by Mick Jenkins — Genre: unknown year: unknown fetched from playlist: The Best
- 'Bitch, Don’t Kill My Vibe' by Kendrick Lamar — Genre: hip-hop year: 2012 fetched from playlist: vibes
- 'Motivational Speech' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Show Me How' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Research' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'I Feel It Coming' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Warm Breeze' by Lofi Fruits Music, Chill Fruits Music — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Feels' by Calvin Harris — Genre: dance year: 2017 fetched from playlist: vibes
- 'Motivational Speech' by — Genre: hip hop year: unknown fetched from playlist: Lofi Fruits Music - lofi hip hop beats to study, relax & sleep to
- 'Good Vibe' by Lil Haiti — Genre: unknown year: unknown fetched from playlist: boats and hoes

Notes: Some metadata may be missing or labelled as 'unknown'. This is normal for this dataset and should NOT influence quality judgment.

Judge Verdict: Partially Correct
Quality Score: 70.0/100
Confidence: 80.0%

Reasoning: The playlist generated aligns with the user prompt of 'study vibe' primarily through the inclusion of many lofi hip hop tracks, which are well-known for their calming and study-friendly qualities. However, the overwhelming presence of hip hop tracks, particularly from the same artists and playlists, leads to a lack of diversity and may not fully capture the broader range of music that could enhance a study atmosphere. Additionally, while some tracks like 'Bitch, Don’t Kill My Vibe' by Kendrick Lamar and 'It's A Vibe' by 2 Chainz are popular, they may not fit the 'study vibe' as they are more upbeat and lyrical, which could be distracting. Overall, while the playlist is coherent with the study theme, it could benefit from more variety in genres and artists to enhance the listening experience for studying.
Notes: Consider incorporating a wider range of genres such as ambient, classical, or acoustic tracks that are also conducive to studying.

```
This would improve the overall quality and diversity of the playlist.

================================================================================
==============================

### Case 5 ###
Prompt: songs to scream to
Generated playlist: Given prompt: songs to scream to
Number of tracks: 30

Tracks:
- 'Let Me Hear You Scream' by Ozzy Osbourne — Genre: rock year:
unknown fetched from playlist: ROCK MUSIK
- 'Scream' by Michael Jackson, Janet Jackson — Genre: rock year:
unknown fetched from playlist: The Best of Michael Jackson
- 'Scream & Shout' by will.i.am, Britney Spears — Genre: pop year:
unknown fetched from playlist: Pop 2000-2010 Bangers
- 'Scream' by   — Genre: pop year: unknown fetched from playlist: Pop
2000-2010 Bangers
- 'Let Me Hear You Scream' by   — Genre: rock year: unknown fetched
from playlist: ROCK MUSIK
- 'Scream & Shout' by   — Genre: pop year: unknown fetched from
playlist: Pop 2000-2010 Bangers
- 'Scream' by High School Musical Cast — Genre: dance year: 2008
fetched from playlist: old songs
- 'Scream' by USHER — Genre: pop year: unknown fetched from playlist:
Pop 2000-2010 Bangers
- 'Scream & Shout' by will.i.am — Genre: unknown year: unknown fetched
from playlist: Britney Spears
- 'Scream' by Avenged Sevenfold — Genre: metal year: 2007 fetched from
playlist: BFMV
- 'Scream' by Usher — Genre: unknown year: unknown fetched from
playlist: ML
- 'Shout' by Tears For Fears — Genre: pop year: unknown fetched from
playlist: Pop FM’s bedste
- 'Choking On Your Screams' by   — Genre: rock year: unknown fetched
from playlist: ROCK MUSIK
- 'Roar' by Katy Perry — Genre: pop year: unknown fetched from
playlist: SONGS WE ALL KNOW😜
- 'Shout At The Devil' by Mötley Crüe — Genre: rock year: unknown
fetched from playlist: Hard Rock /Metal
- 'Scream & Shout' by NIVEK, JKRS — Genre: pop year: unknown fetched
from playlist: ♫ Popular Techno Remixes — Rave & Club Bangers□ □
- 'Choking On Your Screams' by Motörhead — Genre: rock year: unknown
fetched from playlist: ROCK MUSIK
- 'This Is Usually The Part Where People Scream' by Alesana — Genre:
emo year: 2008 fetched from playlist: Alesana
- 'Scream' by New Years Day — Genre: emo year: 2015 fetched from
playlist: motionless in white
- 'Screaming for Vengeance' by Judas Priest — Genre: rock year:
```

```
unknown fetched from playlist: Hard Rock /Metal
- 'Shout 2000' by Disturbed — Genre: metal year: 2000 fetched from playlist: Disturbed
- 'Glory And Gore' by Lorde — Genre: pop year: 2013 fetched from playlist: cry
- 'Scream' by Halestorm — Genre: metal year: 2015 fetched from playlist: Drowning
- 'Shout' by   — Genre: pop year: unknown fetched from playlist: Pop FM’s bedste
- 'Cry Baby' by Janis Joplin — Genre: rock year: unknown fetched from playlist: Classic Rock Songs 60s 70s 80s 90s
- 'Stand Up and Shout' by Dio — Genre: rock year: unknown fetched from playlist: Hard Rock /Metal
- 'What the Hell' by Avril Lavigne — Genre: pop year: unknown fetched from playlist: pop girlies□□□
- 'Stand Up and Shout' by   — Genre: rock year: unknown fetched from playlist: Hard Rock /Metal
- 'In Case of Emergency, Dial 411' by Sleeping With Sirens — Genre: emo year: 2010 fetched from playlist: new
- 'Rebel Yell' by Billy Idol — Genre: rock year: unknown fetched from playlist: Top 100 Rock Classics: 80s, 90s & 2000s

Notes: Some metadata may be missing or labelled as 'unknown'. This is normal for this dataset and should NOT influence quality judgment.


Judge Verdict:       Partially Correct
Quality Score:       70.0/100
Confidence:          80.0%

Reasoning: The playlist contains a variety of songs that include the word 'scream' or are related to the concept of screaming, which aligns with the user prompt. However, the coherence of the genres is mixed; while rock and metal tracks are appropriate for a playlist meant for screaming, there are also pop tracks that may not fit the intense mood typically associated with screaming. Additionally, there are multiple entries with missing metadata and duplicates, which detracts from the overall quality and diversity of the playlist. The inclusion of emo tracks adds some depth, but the overall execution lacks the energy and intensity expected from a playlist designed for screaming. Therefore, while the playlist is relevant, it does not fully meet the expectations for coherence and quality.
Notes: To improve, the playlist could focus more on high-energy rock and metal tracks that evoke a stronger screaming vibe, and reduce the number of pop tracks that dilute the intensity.

================================================================================

### Case 6 ###
```

```
Prompt: Kendrick Lamar
Generated playlist: Given prompt: Kendrick Lamar
Number of tracks: 12

Tracks:
- 'untitled 07 | 2014 - 2016' by Kendrick Lamar — Genre: hip-hop year:
2016 fetched from playlist: kenny
- 'Rolling Stone' by Black Hippy — Genre: hip-hop year: 2011 fetched
from playlist: kendrick lamar
- 'untitled 03 | 05.28.2013.' by Kendrick Lamar — Genre: hip-hop year:
2016 fetched from playlist: Kendrick Lamar — untitled unmastered.
- 'untitled 06 | 06.30.2014.' by Kendrick Lamar — Genre: hip-hop year:
2016 fetched from playlist: KING KENDRICK
- 'u' by Kendrick Lamar — Genre: hip-hop year: 2015 fetched from
playlist: Kendrick Lamar — untitled unmastered.
- 'King Kunta' by Kendrick Lamar — Genre: hip-hop year: 2015 fetched
from playlist: KING KENDRICK
- 'good kid' by Kendrick Lamar — Genre: hip-hop year: 2012 fetched
from playlist: Kendrick Lamar - good kid, m.A.A.d city
- 'untitled 05 | 09.21.2014.' by Kendrick Lamar — Genre: hip-hop year:
2016 fetched from playlist: Kendrick Lamar — untitled unmastered.
- 'untitled 02 | 06.23.2014.' by Kendrick Lamar — Genre: hip-hop year:
2016 fetched from playlist: Kendrick Lamar — untitled unmastered.
- 'DUCKWORTH.' by Kendrick Lamar — Genre: hip-hop year: 2017 fetched
from playlist: KING KENDRICK
- 'Hood Politics' by Kendrick Lamar — Genre: hip-hop year: 2015
fetched from playlist: Kendrick Lamar
- 'i' by Kendrick Lamar — Genre: hip-hop year: 2014 fetched from
playlist: kendrick lamar

Notes: Some metadata may be missing or labelled as 'unknown'. This is
normal for this dataset and should NOT influence quality judgment.


Judge Verdict:       Correct
Quality Score:       95.0/100
Confidence:          90.0%

Reasoning: The playlist generated is highly coherent with the user
prompt, which is simply 'Kendrick Lamar'. All tracks are by Kendrick
Lamar, showcasing a strong representation of his work. The genre is
consistently hip-hop, aligning perfectly with Kendrick Lamar's style.
The selection spans various years from 2011 to 2017, reflecting a good
temporal range of his discography. Additionally, the tracks chosen are
diverse in terms of their themes and moods, which is typical of
Kendrick's music. The only minor point is that the playlist could have
included a few more varied tracks from collaborations or features to
enhance diversity, but overall, it meets the expectations set by the
prompt very well.
Notes: Consider including a few collaborative tracks or features to
```

```
add more diversity to the playlist.

================================================================================
================================
```