Spotify Recommendation System

Introduction

This notebook provides the code snippets, instructions, and descriptions of creating a Spotify recommendation system with Python. Clustering, an unsupervised machine learning algorithm, is incorporated into this project to split the dataset into clusters and reduce the processing time of the distance formula. The data used in this project is a large collection of songs from Spotify containing basic information about the track as well as numerical features derived from Spotify's API. These numerical features include interesting factors such as danceability, energy, speechiness, etc. and these factors will be used to find other similar songs. The goal of this recommendation algorithm is to find similar songs using only the numerical features provided by Spotify and does not include categorical variables such as artist or album.

Things you need before you start:

- Download Spotify 1.2m+ Songs dataset from Kaggle: https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs?reso= (https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs?reso=)
- A Spotify account (free or premium)
- Sign up for Spotify For Developers with your personal Spotify account (https://developer.spotify.com/discover/ (https://developer.spotify.com/discover/)), create a new app, and retrieve your client id and secret client id

Set up

Install packages as needed.

In [1]:
```
# Import packages
import numpy as np
import os
import pandas as pd
import io
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from collections import defaultdict
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist
import difflib

# to make this notebook's output stable across runs
np.random.seed(42)

# to plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

Uploading the dataset

This dataset found on Kaggle contains over 1.2 million Spotify songs. Download the datset at https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs?reso= (https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs?reso=)

In [2]:
```
# load dataset from local file
# data downloaded from https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs?reso=
spotify = pd.read_csv(r'<insert local directory of dataset>')
```

Exploratory Analysis and Data Cleaning

Feature Descriptions

Here are descriptions for each of the 24 features in this dataset:

- **id**: this is the unique id given to a song by Spotify (categorical)
- **name**: name of the song (categorical)
- **album**: the album the song is from (categorical)
- **album_id**: unique id given to each album by Spotify (categorical)
- **artists**: a list of the artists from each song (categorical)
- **artist_ids**: a list of the unique ids given to each artist by Spotify (categorical)
- **track_number**: the track number of the song i.e. where it appears on the album (categorical - ordinal)

- **disc_number**: disc number the song appears on (categorical - ordinal)
- **explicit**: whether or not the song is explicit, False for not explicit, True for explicit (categorical)
- **danceability**: measure of how dancable the song is on a contiuous scale 0-1, 0 being not danceable and 1 being the most danceable (numerical)
- **energy**: measure of intensity/activeness of a song on a continuous scale 0 to 1, 0 being low energy and 1 being high energy (numerical)
- **key** integer scale 0 to 11, each correspondinng to a musical key in order of stand pitch class notation - 0 = C, 1 = C#/Db, 2 = D, and so on (categorical mapped to integers)
- **loudness**: describes loudness of a song in decibels ranging from -60 to 7.23 (numerical)
- **mode**: whether the song is in major or minor mode, 0 for minor, 1 for major (categorical mapped to integers)
- **speechiness**: proportion of spoken words in a track ranging from 0 to 1 (numerical)
- **acousticness**: a continuous confidence measure if a song is acoustic ranging from 0 to 1, 0 is likely not acoustic, 1 is likely acoustic (numerical)
- **instrumentalness**: proportion of instrumental parts of a song ranging from 0 to 1, 0 for less instrumentals and 1 for more instrumentals (numerical)
- **liveness**: probability that a track was performed live (detects for live audience sounds) ranging from 0 to 1, 0 is not likely performed live, 1 is likely performed live
- **valence**: measure of positivity ranging from 0 to 1, 0 for not very postive and 1 for very postive (numerical)
- **tempo**: overall tempo of a track in beats per minute (BPM) ranging from 0 to 249 (numerical)
- **duration_ms**: the length of a song in milliseconds (numerical)
- **time_signature**: the overall time signature of a track in notational convention that measures how many beats per bar (numerical)
- **year**: year song was released (categorical)
- **release_date**: full release date of a song in YYYY-MM-DD format (categorical)

Below shows a snippet of what the dataset looks like. We can see a few of the tracks in the dataset and their features.

In [3]:
```
spotify.head()
```

Out[3]:

	id	name	album	album_id	artists	artist_ids	track_number	disc_number	explicit	dan
0	7lmeHLHBe4nmXzuXc0HDjk	Testify	The Battle Of Los Angeles	2eia0myWFgoHuttJytCxgX	['Rage Against The Machine']	['2d0hyoQ5ynDBnkvAbJKORj']	1	1	False	
1	1wsRitfRRtWyEapl0q22o8	Guerrilla Radio	The Battle Of Los Angeles	2eia0myWFgoHuttJytCxgX	['Rage Against The Machine']	['2d0hyoQ5ynDBnkvAbJKORj']	2	1	True	
2	1hR0fIFK2qRG3f3RF70pb7	Calm Like a Bomb	The Battle Of Los Angeles	2eia0myWFgoHuttJytCxgX	['Rage Against The Machine']	['2d0hyoQ5ynDBnkvAbJKORj']	3	1	False	
3	2lbASgTSoDO7MTuLAXITW0	Mic Check	The Battle Of Los Angeles	2eia0myWFgoHuttJytCxgX	['Rage Against The Machine']	['2d0hyoQ5ynDBnkvAbJKORj']	4	1	True	
4	1MQTmpYOZ6fcMQc56Hdo7T	Sleep Now In the Fire	The Battle Of Los Angeles	2eia0myWFgoHuttJytCxgX	['Rage Against The Machine']	['2d0hyoQ5ynDBnkvAbJKORj']	5	1	False	

5 rows × 24 columns

Here we return some of the information about the dataset including the size of the dataset and feature name, type, and non-null count.

In [4]:
```
spotify.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1204025 entries, 0 to 1204024
Data columns (total 24 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   id                1204025 non-null  object
 1   name              1204025 non-null  object
 2   album             1204025 non-null  object
 3   album_id          1204025 non-null  object
 4   artists           1204025 non-null  object
 5   artist_ids        1204025 non-null  object
 6   track_number      1204025 non-null  int64
 7   disc_number       1204025 non-null  int64
 8   explicit          1204025 non-null  bool
 9   danceability      1204025 non-null  float64
 10  energy            1204025 non-null  float64
 11  key               1204025 non-null  int64
 12  loudness          1204025 non-null  float64
 13  mode              1204025 non-null  int64
 14  speechiness       1204025 non-null  float64
 15  acousticness      1204025 non-null  float64
 16  instrumentalness  1204025 non-null  float64
 17  liveness          1204025 non-null  float64
 18  valence           1204025 non-null  float64
 19  tempo             1204025 non-null  float64
 20  duration_ms       1204025 non-null  int64
 21  time_signature    1204025 non-null  float64
 22  year              1204025 non-null  int64
 23  release_date      1204025 non-null  object
dtypes: bool(1), float64(10), int64(6), object(7)
memory usage: 212.4+ MB
```

This will give us the count, mean, standard devaition, min, Q1, median, Q3, and max of each of the numeric columns.

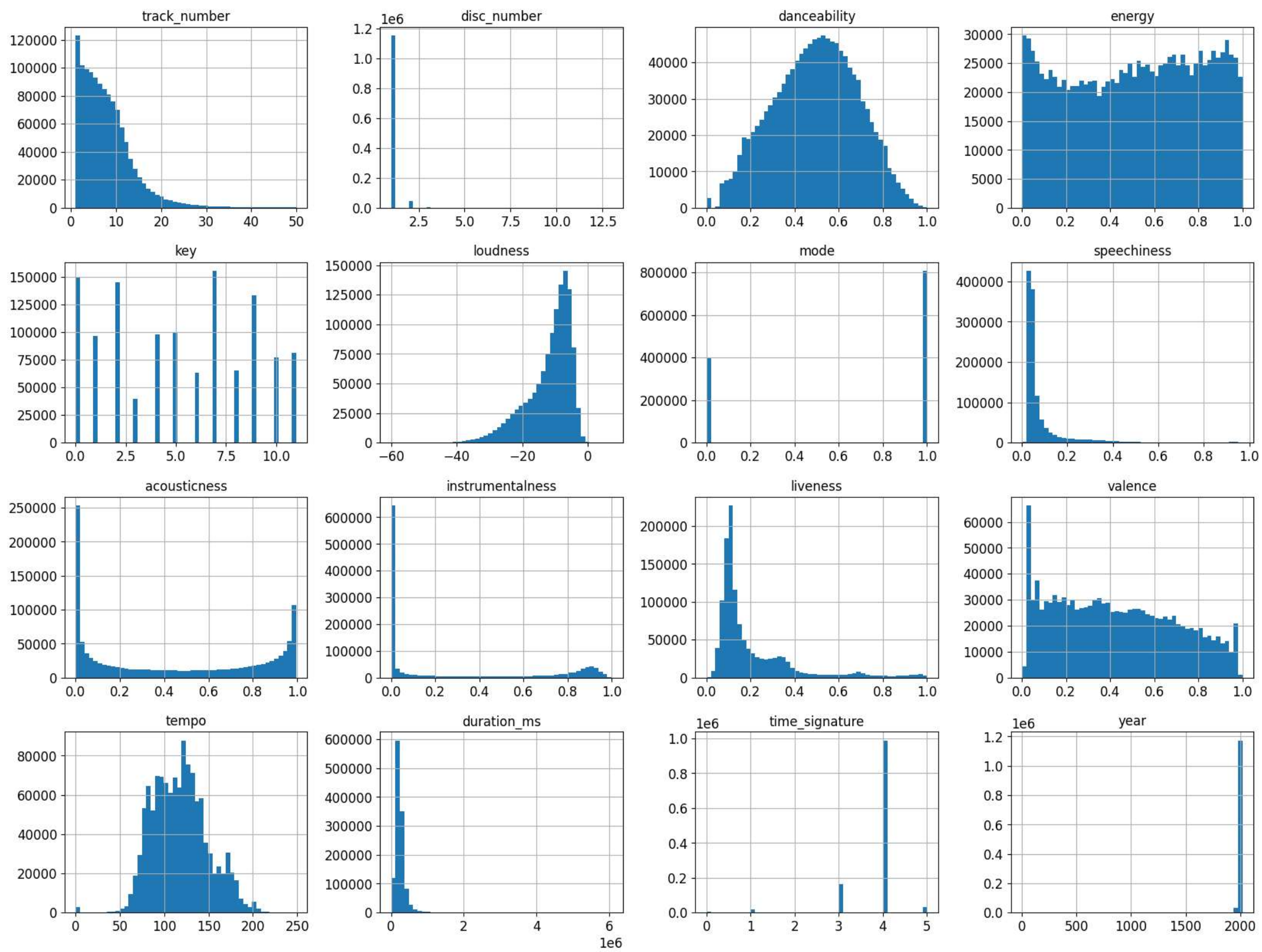
In [5]:
```
spotify.describe()
```

Out[5]:

	track_number	disc_number	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness
count	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06	1.204025e+06
mean	7.656352e+00	1.055906e+00	4.930565e-01	5.095363e-01	5.194151e+00	-1.180870e+01	6.714595e-01	8.438219e-02	4.467511e-01	2.828605e-01
std	5.994977e+00	2.953752e-01	1.896694e-01	2.946839e-01	3.536731e+00	6.982132e+00	4.696827e-01	1.159914e-01	3.852014e-01	3.762844e-01
min	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	-6.000000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	3.000000e+00	1.000000e+00	3.560000e-01	2.520000e-01	2.000000e+00	-1.525400e+01	0.000000e+00	3.510000e-02	3.760000e-02	7.600000e-06
50%	7.000000e+00	1.000000e+00	5.010000e-01	5.240000e-01	5.000000e+00	-9.791000e+00	1.000000e+00	4.460000e-02	3.890000e-01	8.080000e-03
75%	1.000000e+01	1.000000e+00	6.330000e-01	7.660000e-01	8.000000e+00	-6.717000e+00	1.000000e+00	7.230000e-02	8.610000e-01	7.190000e-01
max	5.000000e+01	1.300000e+01	1.000000e+00	1.000000e+00	1.100000e+01	7.234000e+00	1.000000e+00	9.690000e-01	9.960000e-01	1.000000e+00

Now we will plot the frequency of values in each of our numerical datasets using histograms.

In [6]:
```python
spotify.hist(bins=50, figsize=(20,15))
plt.show()
```



Exploratory Analysis Summary

Fortunately, this data looks very clean! There are some interesting distributions shown in the plot of histograms where some are skewed and some are normal. There are no missing values and many of the features are already scaled 0 to 1. Adding a standard scalar to our clustering pipeline should be sufficient to deal with any numerical data that is not already between 0 and 1.

Data Cleaning

For cleaning, all we will being doing is creating a another dataframe containing only the numerical values. This is for the purpose of clustering with numerical audio features that are available on the Spotify API.

In [8]:
```python
# separate numerical values
spotify_metrics = spotify.iloc[:, 9:22]
list(spotify_metrics)
```

Out[8]:
```
['danceability',
 'energy',
 'key',
 'loudness',
 'mode',
 'speechiness',
 'acousticness',
 'instrumentalness',
 'liveness',
 'valence',
 'tempo',
 'duration_ms',
 'time_signature']
```

Clustering

The unsupervised machine learning algorithm clustering will be used in this project to reduce the processing time of the distance algorithm. This works by sorting the data into groups of other 'nearby' data. The distance formula will then only search within the input song's cluster rather than the entire dataset.

KMeans

We will be using a KMeans clustering algorithm on this data because this particular algorithm works well with large datasets. We will start with 10 clsuters to see how it does, and then search through different numbers of clusters to see what performs best.

In [9]:
```
# kmeans algorithm with 10 clusters to start
song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),
                                  ('kmeans', KMeans(n_clusters=10))
                                  ], verbose=False)
number_cols = list(spotify_metrics.columns)
kmeans10 = song_cluster_pipeline.fit(spotify_metrics)
song_cluster_labels = song_cluster_pipeline.predict(spotify_metrics)
spotify['cluster_label'] = song_cluster_labels
```

```
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

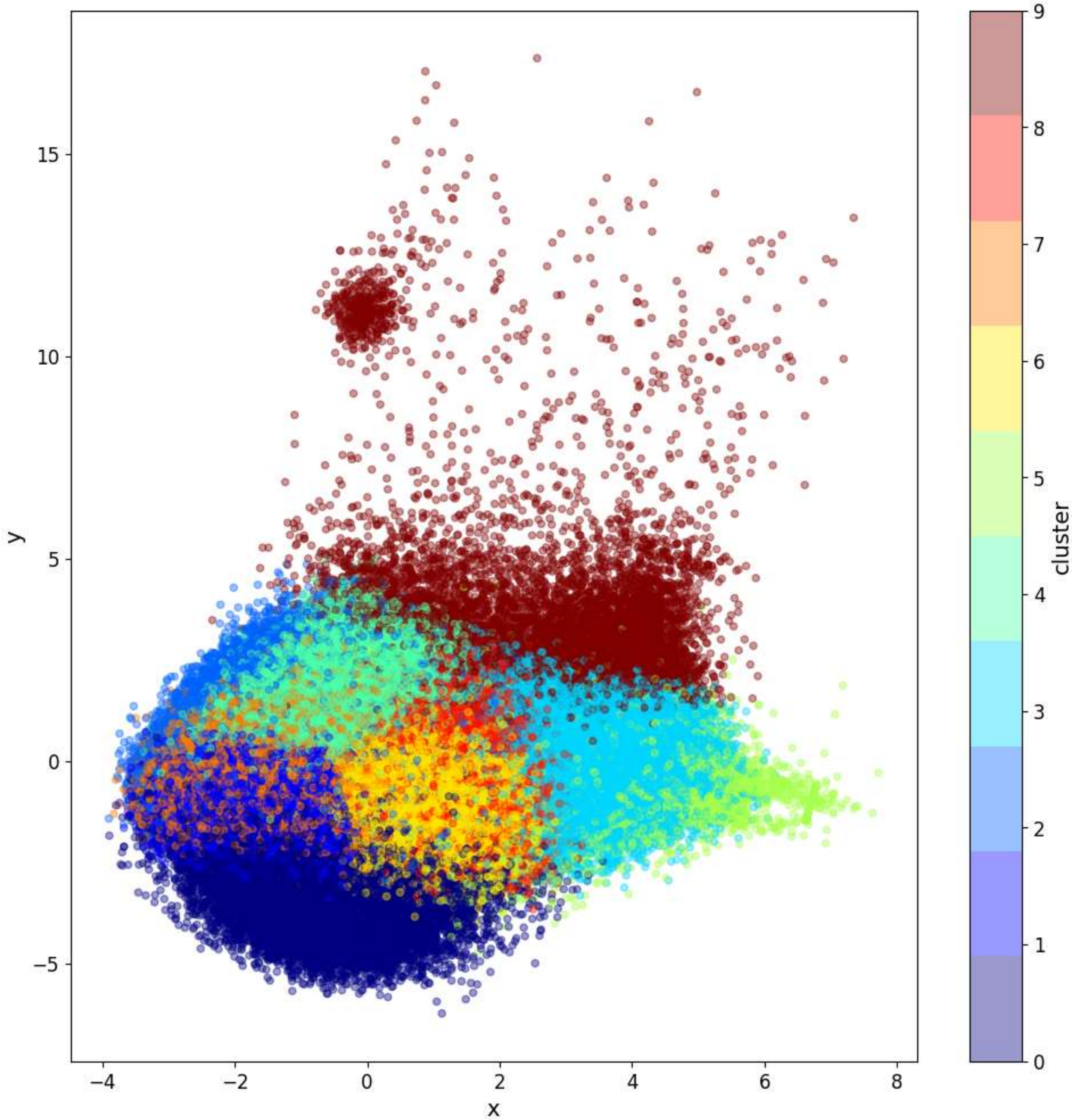
In [10]:
```
# check the inertia (we want a low number)
kmeans10['kmeans'].inertia_
```

Out[10]: 8343806.648790574

Next, we will visualize these clusters on the dataset. Since there are so many features we will perform a Principle Component Analysis to project the dataset down to 2 components, and then plot a scatterplot of these two components.

In [11]:

```
# perform a PCA on the features so we can visualize the results of the clustering
# by projecting the features to 2 components
pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])
song_embedding = pca_pipeline.fit_transform(spotify_metrics)
projection = pd.DataFrame(columns=['x','y'], data=song_embedding)
projection['title'] = spotify['name']
projection['cluster'] = spotify['cluster_label']

cluster_plot = projection.plot(kind="scatter", x="x", y="y", alpha=0.4, figsize=(12,12),
                               c="cluster", cmap=plt.get_cmap("jet",10), sharex=False)
plt.show()
```

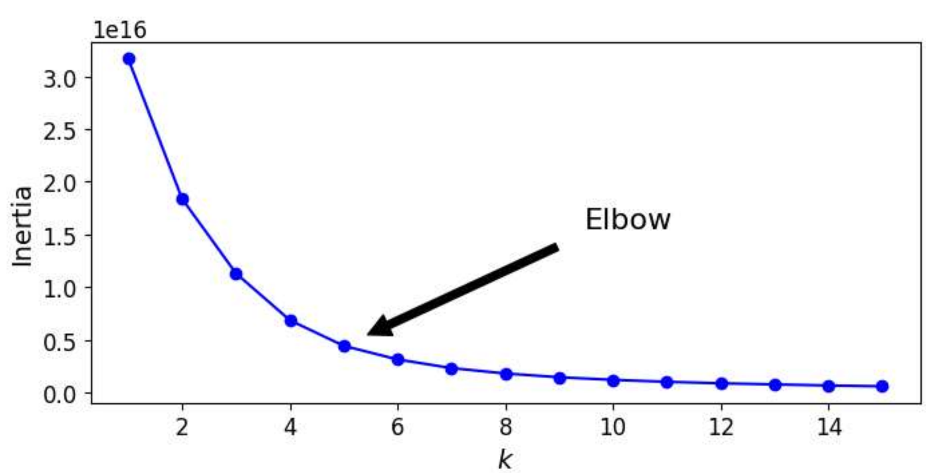


It looks as though there are some definitive clusters in the dataset. It is a bit muddy in places, particularly because of the projection. Let's try other numbers of clusters and compare their inertias.

Finding optimal number of clusters

In [12]:
```
## fit kmeans models with 1-15 clusters and plot inertia vs number of clusters
# NOTE: this may take some time to run
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(spotify_metrics)
               for k in range (1,16)]
inertias = [model.inertia_ for model in kmeans_per_k]

plt.figure(figsize=(8,3.5))
plt.plot(range(1,16), inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.annotate('Elbow',
            xy=(5,inertias[4]),
            xytext=(0.55,0.55),
            textcoords='figure fraction',
            fontsize=16,
            arrowprops=dict(facecolor='black', shrink=0.1))
plt.show()
```

```
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

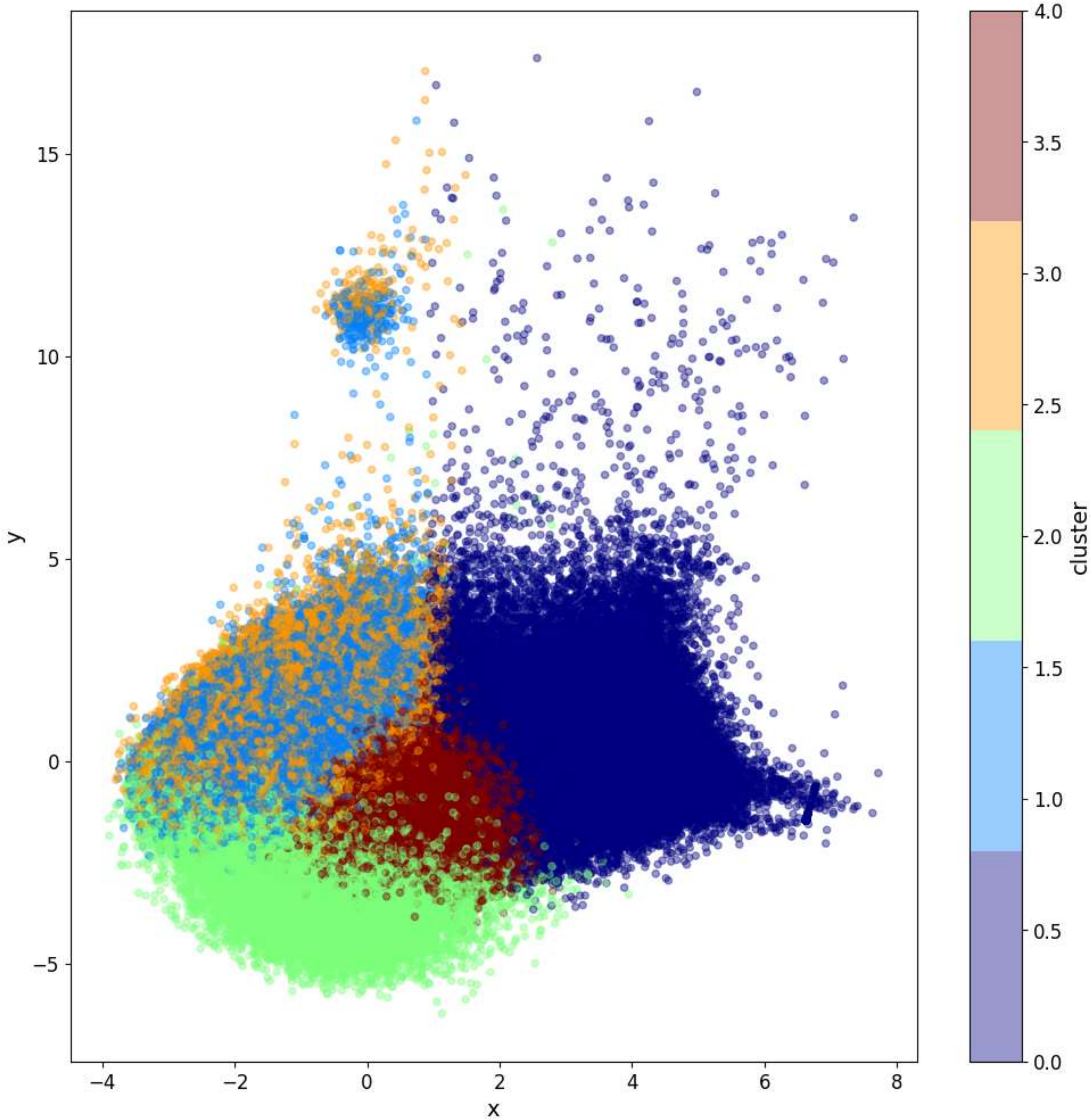


Our comparison of inertias for clusters 1-15 shows a very nice curve demonstrating the tradeoff of inertia and number of clusters. Five clusters appears to be where the "elbow" occurs and has the best tradeoff of inertia for number of clusters. Let's more closely examine how k=5 performs for our clustering algorithm.

In [13]:

```
# plot best number of clusters (k=5)
song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),
                                  ('kmeans', KMeans(n_clusters=5))
                                 ], verbose=False)
number_cols = list(spotify_metrics.columns)
kmeans5 = song_cluster_pipeline.fit(spotify_metrics)
song_cluster_labels = song_cluster_pipeline.predict(spotify_metrics)
spotify['cluster_label'] = song_cluster_labels

pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])
song_embedding = pca_pipeline.fit_transform(spotify_metrics)
projection = pd.DataFrame(columns=['x','y'], data=song_embedding)
projection['title'] = spotify['name']
projection['cluster'] = spotify['cluster_label']

cluster_plot = projection.plot(kind="scatter", x="x", y="y", alpha=0.4, figsize=(12,12),
                              c="cluster", cmap=plt.get_cmap("jet",5), sharex=False, colorbar = True)
plt.show()
```

```
C:\Users\shans\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



The clusters in this plot look to be fairly defined, and the areas that are not as defined as likely due to the projection.

In [15]:
```
# check the inertia (we want a low number)
kmeans5['kmeans'].inertia_
```

Out[15]:
```
10423612.880860724
```

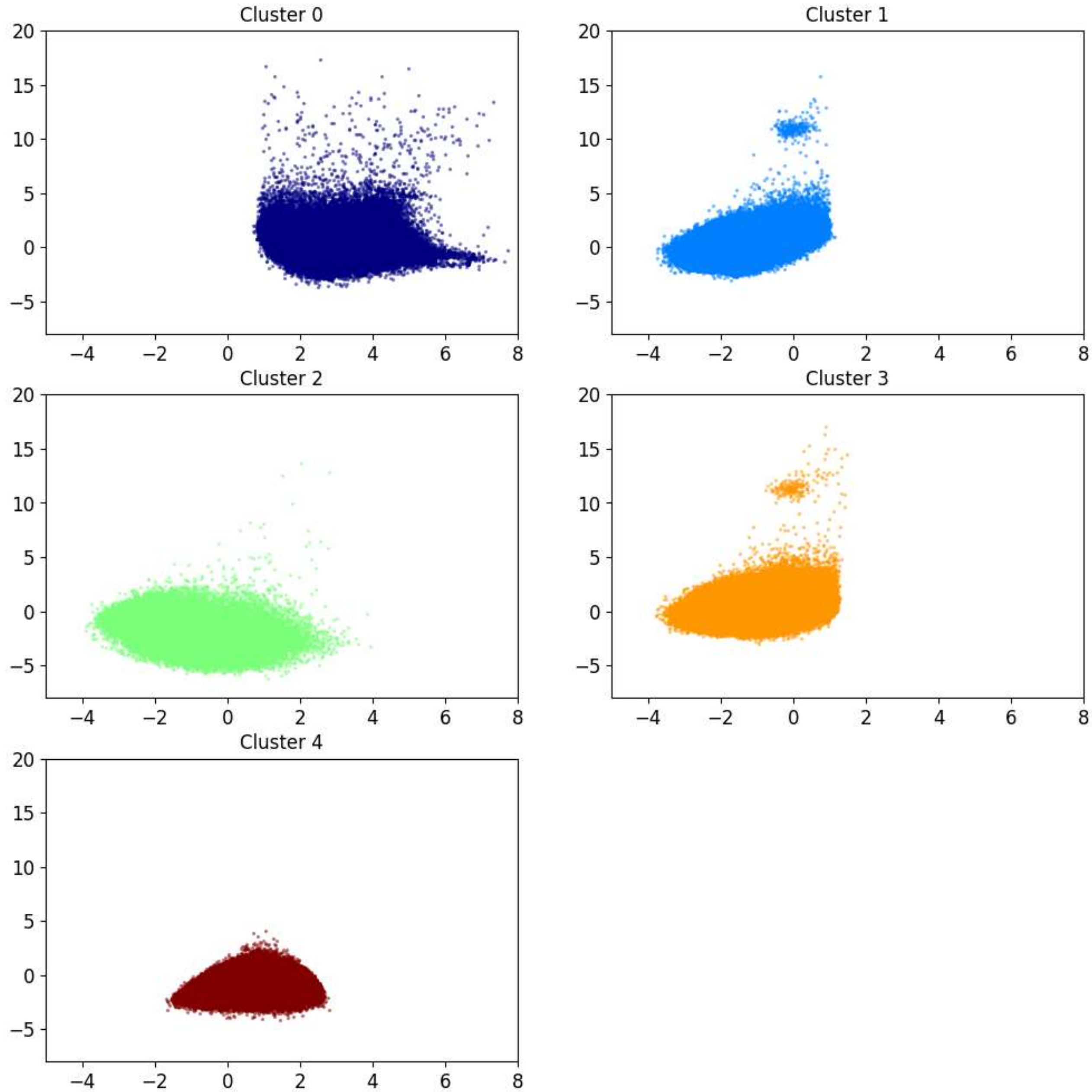
The next plots separate the clusters so we can see the whole cluster.

In [16]:

```
plt.figure(figsize=(12, 12))
color_jet = plt.get_cmap("jet",5)

plt.subplot(321)
projection0 = projection.query("cluster == 0")
plt.scatter(projection0['x'], projection0['y'], alpha = 0.4, s = 2,
            color = color_jet(0))
plt.axis([-5, 8, -8, 20])
#projection0.plot(kind="scatter", x="x", y="y", alpha=0.4)
plt.title("Cluster 0")

plt.subplot(322)
projection1 = projection.query("cluster == 1")
plt.scatter(projection1['x'], projection1['y'], alpha = 0.4, s = 2,
            color = color_jet(0.2))
plt.axis([-5, 8, -8, 20])
#projection1.plot(kind="scatter", x="x", y="y", alpha=0.4)
plt.title("Cluster 1")

plt.subplot(323)
projection2 = projection.query("cluster == 2")
plt.scatter(projection2['x'], projection2['y'], alpha = 0.4, s = 2,
            color = color_jet(0.4))
plt.axis([-5, 8, -8, 20])
#projection2.plot(kind="scatter", x="x", y="y", alpha=0.4)
plt.title("Cluster 2")

plt.subplot(324)
projection3 = projection.query("cluster == 3")
plt.scatter(projection3['x'], projection3['y'], alpha = 0.4, s = 2,
            color = color_jet(0.6))
plt.axis([-5, 8, -8, 20])
#projection3.plot(kind="scatter", x="x", y="y", alpha=0.4)
plt.title("Cluster 3")

plt.subplot(325)
projection4 = projection.query("cluster == 4")
plt.scatter(projection4['x'], projection4['y'], alpha = 0.4, s = 2,
            color = color_jet(0.8))
plt.axis([-5, 8, -8, 20])
#projection4.plot(kind="scatter", x="x", y="y", alpha=0.4)
plt.title("Cluster 4")

plt.show
```

Out[16]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Recommendation System

This recommendation system will be utilizing Spotify's API and the package `spotipy` . Our clustering algorithm can help reduce the computational time by only checking the similarities within the input song's cluster.

This recommendation will make use of the cosine similarity formula, which is commonly used for recommendation systems. Cosine similarity measures the cosine angle between sequences of numbers as vectors.

Any song found on Spotify can be input into this algorithm whether it is in the existing data or not. The recommended songs, however, are currently limited to what is in this dataset.

To use this code, you will need a Spotify account (free or premium both work) and sign up for Spotify for Developers. You will then create a new app, and add your client id and client secret id to the environment variables.

```
os.environ['SPOTIFY_CLIENT_ID'] = 'your_client_id'
os.environ['SPOTIFY_CLIENT_SECRET'] = 'your_client_secret_id'
```

In [17]:

```
# fill in with your Spotify for Developers app information as detailed above
os.environ['SPOTIFY_CLIENT_ID'] = 'your_client_id'
os.environ['SPOTIFY_CLIENT_SECRET'] = 'your_client_secret_id'
```

In [18]:
```
# Input your unique authenticating information to utilize the Spotify API
sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=os.environ["SPOTIFY_CLIENT_ID"],
                                                           client_secret=os.environ["SPOTIFY_CLIENT_SECRET"]))
```

The function below will retrieve the input song info. To input a song, choose a song from Spotify and copy the song URL from Spotify. You can do this by clicking the 3 dots to the right of a song -> Share -> Copy Song Link. Input the URL as a string.

This function will first search the dataset for the song. If the song is not in the dataset, then it will call the Spotify API to retrieve the song information.

In [19]:
```
# Function that will retrieve song info from song URL
# You can copy a song URL from spotify and save it as a string
def get_song_info(URL, data = spotify):
  id = URL.split('/')[-1].split("?")[0]
  song_in_dataset = data.loc[(data['id'] == id)]
  new_song = []
  scaler = song_cluster_pipeline.steps[0][1]

  if song_in_dataset.empty:
    new_song = pd.DataFrame(sp.audio_features(id))
    new_song_metrics = new_song.drop(columns = ['type', 'id', 'uri',
                                            'track_href','analysis_url'])
    new_song['cluster_label'] = kmeans5['kmeans'].predict(scaler.transform
                                                    (new_song_metrics))[0]
    new_song['name'] = sp.track(id)["name"]
    new_song['album'] = sp.track(id)["album"]["name"]
    return new_song

  else:
    return song_in_dataset
```

In [20]:
```
# Test function on Lost by Frank Ocean
LostFrankOcean = 'https://open.spotify.com/track/3GZD6HmiNUhxXYf8Gch723?si=8a7c6b5e1e1b4c78'
get_song_info(LostFrankOcean)
```

Out[20]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	type	i
0	0.913	0.603	8	-4.892	1	0.226	0.0272	0.000503	0.167	0.497	...	audio_features	3GZD6HmiNUhxXYf8Gch72

1 rows × 21 columns

Our last function will be the actual song recommender. It takes in the song URL as a string and will return the top 5 most similar songs as recommendations.

In [21]:
```
# This is our final function that will give song recommendations given a song
# list, our dataset, and the number of songs to return
def recommend_songs(URL, data = spotify, n_songs=10):
    number_cols = list(spotify_metrics)
    song = get_song_info(URL)
    metadata_cols = ['name', 'year', 'artists']

    # filter data by predicted cluster in song
    clustered_data = data.loc[(data['cluster_label'] == song['cluster_label'][0])]

    # scale data and new song
    scaler = song_cluster_pipeline.steps[0][1]
    scaled_data = scaler.transform(clustered_data[number_cols])
    scaled_song = scaler.transform(song[number_cols])

    # find the distances and the indices of the most similar songs
    distances = cdist(scaled_song, scaled_data, 'cosine')
    index = list(np.argsort(distances)[:, :n_songs][0])

    # collect the recommended songs
    rec_songs = clustered_data.iloc[index]
    rec_songs = rec_songs[~rec_songs['name'].isin(song['name'])]
    return rec_songs#[metadata_cols].to_dict(orient='records')
```

In [22]:
```
LostFrankOcean = 'https://open.spotify.com/track/3GZD6HmiNUhxXYf8Gch723?si=8a7c6b5e1e1b4c78'
rec_songs = recommend_songs(LostFrankOcean)
rec_songs.head()
```

Out[22]:

	id	name	album	album_id	artists	artist_ids	track_number	disc_numbe
612458	3NSTniwtVXJ5iYf5NVudIt	Do You Miss Me at All (Marian Hill Remix)	Do You Miss Me at All (Marian Hill Remix)	1X6bsuOMwgsfZDYIl6G48a	['Bridgit Mendler', 'Marian Hill']	['4VhL8KLjVso4vLfOLVViTb', '1xHQO9GJIW9OXHxGBI...	1	1
301515	77dgyxbuL53WfkLZU3fk3o	Ain't No Other Man	Keeps Gettin' Better: A Decade of Hits	2019iQx5MmA6byqYqdK7zS	['Christina Aguilera']	['1l7ZsJRRS8wlW3WfJfPfNS']	8	1
1189899	6L5uDZkqqe6L8ObBkYMpm8	Mann Family	Born to Drip	2fzaiLb2045T3gzmaoYCaB	['Sosamann']	['3Bj81IbILbuj2uEwWXMdXI']	1	1
1070117	6dcCZaxoyiSqUgY5vXZqwA	All in It	Invite Only	1eohATSMTwYTEpDco488fX	['Bandhunta Izzy', 'Yella Beezy']	['5nnmjpedVxTOH8KwpDdSZ2', '7kwCkEJ384PWm0UQW3...	2	1

Final Discussion

So that concludes our Spotify recommendation system! Whether or not the recommendations are good are up to the individual user. Personally, I have found some interesting new songs from this that I probably would not have found otherwise. I believe that excluding features such as artist and album allow for the recommendations to focus more on how the composition of the song is similar as opposed to limiting recommendations by common categorical variables.