In [ ]:

In [4]:
```
pip install pandas
```

```
Collecting pandas
  Using cached pandas-2.3.2-cp313-cp313-win_amd64.whl.metadata (19 kB)
Requirement already satisfied: numpy>=1.26.0 in c:\users\sowmi\onedrive\deskto
p\python\amazon\venv\lib\site-packages (from pandas) (2.3.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sowmi\onedriv
e\desktop\python\amazon\venv\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\sowmi\onedrive\desktop\
python\amazon\venv\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\sowmi\onedrive\deskto
p\python\amazon\venv\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\sowmi\onedrive\desktop\pyth
on\amazon\venv\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Using cached pandas-2.3.2-cp313-cp313-win_amd64.whl (11.0 MB)
Installing collected packages: pandas
Successfully installed pandas-2.3.2
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
pip install numpy matpotlib seaborn sklearn
```

```
Requirement already satisfied: numpy in c:\users\sowmi\onedrive\desktop\python\
amazon\venv\lib\site-packages (2.3.3)
Note: you may need to restart the kernel to use updated packages.
ERROR: Could not find a version that satisfies the requirement matpotlib (from
versions: none)
ERROR: No matching distribution found for matpotlib
```

In [1]:
```
import pandas as pd #data manipulation and analysis
import numpy as np #numerical computing
import matplotlib.pyplot as plt #data visualization
import seaborn as sns #data visualization
```

In [2]:
```
import os #operating system
from sklearn.preprocessing import StandardScaler,  MinMaxScaler #Standardizati
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering#K-means cl
from sklearn.decomposition import PCA #Principal component analysis
from sklearn.manifold import TSNE #t-distributed stochastic neighbor embedding
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import silhouette_score,davies_bouldin_score
import scipy.cluster.hierarchy as sch #dendrogram
```

In [3]:
```
import warnings #to ignore warnings
warnings.filterwarnings('ignore') #ignore warnings
```

In [4]:
```
sns.set(style="whitegrid") #set seaborn style
plt.rcParams['figure.figsize'] = (10, 6) #set figure size
pd.set_option('display.max_columns', None) #display all columns
```

In [5]:
```
zon= pd.read_csv(r"C:\Users\sowmi\OneDrive\Desktop\python\amazon\single_genre_
```

In [6]:
```
zon
```

Out[6]:

	id_songs	name_song	popularity_songs	duration_ms	ex
0	0IA0Hju8CAgYfV1hwhidBH	La Java	0	161427	
1	1b8HZQCqcqwbzlA1jRTp6E	En Douce	0	223440	
2	5d5gQxHwYovxR5pqETOIAa	J'en Ai Marre	0	208267	
3	1EO65UEEPfy7CR0NK2sDxy	Ils n'ont pas ca	0	161933	
4	6a58gXSgqbIsXUhVZ6ZJqe	La belote	0	167973	
...	...	...	...	...	
95832	44r4zta6P9flkhKaVnbsvG	Freaks	70	174800	
95833	0MmaEacabpK8Yp3Mdeo5uY	下雨天	50	265846	
95834	1dKxf4Ht2SsKLyXfSDJAgy	The Cutest Puppy	67	82500	
95835	0SjsIzJkZfDU7wlcdklEFR	John Brown's Song	66	185250	
95836	5rgu12WBIHQtvej2MdHSH0	云与海	50	258267	

95837 rows × 23 columns

In [7]:
```
zon.head(3)
```

Out[7]:

	id_songs	name_song	popularity_songs	duration_ms	explicit
0	0IA0Hju8CAgYfV1hwhidBH	La Java	0	161427	0
1	1b8HZQCqcqwbzlA1jRTp6E	En Douce	0	223440	0
2	5d5gQxHwYovxR5pqETOIAa	J'en Ai Marre	0	208267	0

In [8]:
```
zon.tail(3)
```

Out[8]:

	id_songs	name_song	popularity_songs	duration_ms	ex
95834	1dKxf4Ht2SsKLyXfSDJAgy	The Cutest Puppy	67	82500	
95835	0SjsIzJkZfDU7wIcdkIEFR	John Brown's Song	66	185250	
95836	5rgu12WBIHQtvej2MdHSH0	云与海	50	258267	

In [9]:

```
print(zon.shape)            # How many rows & columns?
print(zon.info())           # Data types, missing values?
```

```
(95837, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95837 entries, 0 to 95836
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   id_songs            95837 non-null  object 
 1   name_song           95837 non-null  object 
 2   popularity_songs    95837 non-null  int64  
 3   duration_ms         95837 non-null  int64  
 4   explicit            95837 non-null  int64  
 5   id_artists          95837 non-null  object 
 6   release_date        95837 non-null  object 
 7   danceability        95837 non-null  float64
 8   energy              95837 non-null  float64
 9   key                 95837 non-null  int64  
 10  loudness            95837 non-null  float64
 11  mode                95837 non-null  int64  
 12  speechiness         95837 non-null  float64
 13  acousticness        95837 non-null  float64
 14  instrumentalness    95837 non-null  float64
 15  liveness            95837 non-null  float64
 16  valence             95837 non-null  float64
 17  tempo               95837 non-null  float64
 18  time_signature      95837 non-null  int64  
 19  followers           95837 non-null  float64
 20  genres              95837 non-null  object 
 21  name_artists        95837 non-null  object 
 22  popularity_artists  95837 non-null  int64  
dtypes: float64(10), int64(7), object(6)
memory usage: 16.8+ MB
None
```

In [10]:

```
print(zon.isnull().sum())  # Missing values?
```

```
id_songs              0
name_song             0
popularity_songs      0
duration_ms           0
explicit              0
id_artists            0
release_date          0
danceability          0
energy                0
key                   0
loudness              0
mode                  0
speechiness           0
acousticness          0
instrumentalness      0
liveness              0
valence               0
tempo                 0
time_signature        0
followers             0
genres                0
name_artists          0
popularity_artists    0
dtype: int64
```

In [11]:
```
print(zon.duplicated().sum())  # Duplicates?
print(zon.describe())      # Summary statistics for numerical columns
```

```
0
       popularity_songs   duration_ms      explicit  danceability  \
count      95837.000000  9.583700e+04  95837.000000  95837.000000
mean          26.066394  2.087320e+05      0.029644      0.586853
std           16.254133  1.177526e+05      0.169604      0.155422
min            0.000000  6.373000e+03      0.000000      0.000000
25%           13.000000  1.573330e+05      0.000000      0.488000
50%           26.000000  2.040000e+05      0.000000      0.605000
75%           37.000000  2.502670e+05      0.000000      0.700000
max           98.000000  4.800118e+06      1.000000      0.991000

             energy           key      loudness          mode   speechiness  \
count  95837.000000  95837.000000  95837.000000  95837.000000  95837.000000
mean       0.541083      5.196782    -10.157862      0.648069      0.168832
std        0.236304      3.534923      4.748798      0.477575      0.275417
min        0.000020      0.000000    -50.174000      0.000000      0.000000
25%        0.365000      2.000000    -12.723000      0.000000      0.034100
50%        0.542000      5.000000     -9.397000      1.000000      0.046200
75%        0.727000      8.000000     -6.692000      1.000000      0.103000
max        1.000000     11.000000      5.376000      1.000000      0.968000

       acousticness  instrumentalness      liveness       valence  \
count  95837.000000      95837.000000  95837.000000  95837.000000
mean       0.458989          0.082145      0.224916      0.574281
std        0.330416          0.232440      0.185829      0.248126
min        0.000000          0.000000      0.000000      0.000000
25%        0.133000          0.000000      0.100000      0.378000
50%        0.453000          0.000004      0.149000      0.589000
75%        0.759000          0.001300      0.302000      0.780000
max        0.996000          1.000000      0.997000      1.000000

              tempo  time_signature     followers  popularity_artists
count  95837.000000    95837.000000  9.583700e+04        95837.000000
mean     117.539870        3.851362  1.979919e+05           42.819329
std       30.190399        0.544406  7.807520e+05           20.897833
min        0.000000        0.000000  0.000000e+00            0.000000
25%       94.829000        4.000000  2.563000e+03           28.000000
50%      116.595000        4.000000  1.595600e+04           40.000000
75%      135.975000        4.000000  8.495100e+04           56.000000
max      239.906000        5.000000  2.802643e+07           95.000000
```

In [12]:

```
print("shape = ",zon["id_songs"].shape)
print("\ninfo = ",zon["id_songs"].info())
print("\nisnull = ",zon["id_songs"].isnull().sum())
print("\ndescribe = ", zon["id_songs"].describe())
print("\nnunique =",zon["id_songs"].nunique())
print("\nunique =",zon["id_songs"].unique())
print("\nvalue_counts =",zon['id_songs'].value_counts())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: id_songs
Non-Null Count  Dtype
--------------  -----
95837 non-null  object
dtypes: object(1)
memory usage: 748.9+ KB

info =  None

isnull =  0

describe =  count                      95837
unique                     95837
top       0IA0Hju8CAgYfV1hwhidBH
freq                           1
Name: id_songs, dtype: object

nunique = 95837

unique = ['0IA0Hju8CAgYfV1hwhidBH' '1b8HZQCqcqwbzlA1jRTp6E'
 '5d5gQxHwYovxR5pqETOIAa' ... '1dKxf4Ht2SsKLyXfSDJAgy'
 '0SjsIzJkZfDU7wlcdklEFR' '5rgu12WBIHQtvej2MdHSH0']

value_counts = id_songs
0IA0Hju8CAgYfV1hwhidBH    1
1b8HZQCqcqwbzlA1jRTp6E    1
5d5gQxHwYovxR5pqETOIAa    1
1EO65UEEPfy7CR0NK2sDxy    1
6a58gXSgqbIsXUhVZ6ZJqe    1
                         ..
44r4zta6P9flkhKaVnbsvG    1
0MmaEacabpK8Yp3Mdeo5uY    1
1dKxf4Ht2SsKLyXfSDJAgy    1
0SjsIzJkZfDU7wlcdklEFR    1
5rgu12WBIHQtvej2MdHSH0    1
Name: count, Length: 95837, dtype: int64
```

In [13]:

```python
print("shape = ",zon["name_song"].shape)
print("\ninfo = ",zon["name_song"].info())
print("\nisnull = ",zon["name_song"].isnull().sum())
print("\ndescribe = ", zon["name_song"].describe())
print("\nnunique =",zon["name_song"].nunique())
print("\nunique =",zon["name_song"].unique())
print("\nvalue_counts =",zon['name_song'].value_counts())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: name_song
Non-Null Count  Dtype
--------------  -----
95837 non-null  object
dtypes: object(1)
memory usage: 748.9+ KB

info =  None

isnull =  0

describe =  count                    95837
unique                   85427
top       Bibi Blocksberg Lied
freq                        33
Name: name_song, dtype: object

nunique = 85427

unique = ['La Java' 'En Douce' "J'en Ai Marre" ... 'The Cutest Puppy'
 "John Brown's Song" '云与海']

value_counts = name_song
Bibi Blocksberg Lied
33
Intro
20
Benjamin Blümchen Lied
19
Time
14
Lonely
14

..
春回大地贺新年
1
春联红
1
Brown Noise
1
Fifth Sunday of Lent, Year C: Has No One Condemned You - Communion Antiphons fo
r Satb Choir, Vol. 1      1
Eddy Elephant
1
Name: count, Length: 85427, dtype: int64
```

In [14]:
```
print("shape = ",zon["id_artists"].shape)
print("\ninfo = ",zon["id_artists"].info())
print("\nisnull = ",zon["id_artists"].isnull().sum())
print("\ndescribe = ", zon["id_artists"].describe())
```

```python
print("\nnunique =",zon["id_artists"].nunique())
print("\nunique =",zon["id_artists"].unique())
print("\nvalue_counts =",zon['id_artists'].value_counts())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: id_artists
Non-Null Count  Dtype
--------------  -----
95837 non-null  object
dtypes: object(1)
memory usage: 748.9+ KB

info =  None

isnull =  0

describe =  count                      95837
unique                     18009
top       3meJIgRw7YleJrmbpbJK6S
freq                        3856
Name: id_artists, dtype: object

nunique = 18009

unique = ['4AxgXfD7ISvJSTObqm4aIE' '7DIlOK9L8d0IQ7Xk8aJxDW'
 '28pbIiOohRRZjqpAM9iqYM' ... '7vgGpuiXdNlCmc994PlMlz'
 '4MxqhahGRT4BPz1PilXGeu' '1QLBXKM5GCpyQQSVMNZqrZ']

value_counts = id_artists
3meJIgRw7YleJrmbpbJK6S    3856
0i38tQX5j4gZ0KS3eCMoIl    2006
1l6d0RIxTL3JytlLGvWzYe    1503
3t2iKODSDyzoDJw7AsD99u    1472
1hD52edfn6aNsK3fb5c2OT     812
                          ...
2gaWHbmYNQ9nbKgfmkOLP8       1
2JzTVgQ5vs7k1gGXbC5DWG       1
4AOgY5YwQw4d5pgEX3f56A       1
6gAvgPl08HUbzcqw8hquEf       1
6GCii6lkUhkzTrznRyCuVh       1
Name: count, Length: 18009, dtype: int64
```

In [15]:

```python
print("shape = ",zon["genres"].shape)
print("\ninfo = ",zon["genres"].info())
print("\nisnull = ",zon["genres"].isnull().sum())
print("\ndescribe = ", zon["genres"].describe())
print("\nnunique =",zon["genres"].nunique())
print("\nunique =",zon["genres"].unique())
print("\nvalue_counts =",zon['genres'].value_counts())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: genres
Non-Null Count  Dtype
--------------  -----
95837 non-null  object
dtypes: object(1)
memory usage: 748.9+ KB

info =  None

isnull =  0

describe =  count             95837
unique             3153
top       ['hoerspiel']
freq               8027
Name: genres, dtype: object

nunique = 3153

unique = ["['vintage chanson']" "['new orleans jazz']" "['harlem renaissance']"
...
 "['chinese classical performance']" "['chinese new year']"
 "['singaporean indie']"]

value_counts = genres
['hoerspiel']                        8027
['kleine hoerspiel']                 2081
[]                                   1876
['classic israeli pop']              1180
['vintage taiwan pop']               1097
                                     ...
['italian mezzo-soprano']               1
['deep discofox']                       1
['pinoy praise']                        1
['lithuanian metal']                    1
['chinese classical performance']       1
Name: count, Length: 3153, dtype: int64
```

In [16]:

```python
print("shape = ",zon["name_artists"].shape)
print("\ninfo = ",zon["name_artists"].info())
print("\nisnull = ",zon["name_artists"].isnull().sum())
print("\ndescribe = ", zon["name_artists"].describe())
print("\nnunique =",zon["name_artists"].nunique())
print("\nunique =",zon["name_artists"].unique())
print("\nvalue_counts =",zon['name_artists'].value_counts())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: name_artists
Non-Null Count  Dtype
--------------  -----
95837 non-null  object
dtypes: object(1)
memory usage: 748.9+ KB

info =  None

isnull =  0

describe =  count            95837
unique           17662
top       Die drei ???
freq              3856
Name: name_artists, dtype: object

nunique = 17662

unique = ['Mistinguett' 'Félix Mayol' 'Louis Lynel' ... 'Laureen Conrad'
 'Gregory Oberle' '阿YueYue']

value_counts = name_artists
Die drei ???         3856
TKKG Retro-Archiv    2006
Benjamin Blümchen    1503
Bibi Blocksberg      1472
Fünf Freunde          812
                     ... 
甄秀珍                   1
Vivi Sumanti            1
The Tempters            1
Pat Bringer             1
Batucada Band           1
Name: count, Length: 17662, dtype: int64
```

In [17]:

```
print("shape = ",zon["popularity_songs"].shape)
print("\ninfo = ",zon["popularity_songs"].info())
print("\nisnull = ",zon["popularity_songs"].isnull().sum())
print("\ndescribe = ", zon["popularity_songs"].describe())
print("\nnunique =",zon["popularity_songs"].nunique())
print("\nskew = ",zon['popularity_songs'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: popularity_songs
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean        26.066394
std         16.254133
min          0.000000
25%         13.000000
50%         26.000000
75%         37.000000
max         98.000000
Name: popularity_songs, dtype: float64

nunique = 94

skew =  0.26968048030592345
```

In [18]:

```
print("shape = ",zon["duration_ms"].shape)
print("\ninfo = ",zon["duration_ms"].info())
print("\nisnull = ",zon["duration_ms"].isnull().sum())
print("\ndescribe = ", zon["duration_ms"].describe())
print("\nnunique =",zon["duration_ms"].nunique())
print("\nskew = ",zon['duration_ms'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: duration_ms
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    9.583700e+04
mean     2.087320e+05
std      1.177526e+05
min      6.373000e+03
25%      1.573330e+05
50%      2.040000e+05
75%      2.502670e+05
max      4.800118e+06
Name: duration_ms, dtype: float64

nunique = 44685

skew =  10.035800691260967
```

In [19]:

```
print("shape = ",zon["explicit"].shape)
print("\ninfo = ",zon["explicit"].info())
print("\nisnull = ",zon["explicit"].isnull().sum())
print("\ndescribe = ", zon["explicit"].describe())
print("\nnunique =",zon["explicit"].nunique())
print("\nskew = ",zon['explicit'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: explicit
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.029644
std          0.169604
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: explicit, dtype: float64

nunique = 2

skew =  5.5466257551320926
```

In [20]:

```
print("shape = ",zon["danceability"].shape)
print("\ninfo = ",zon["danceability"].info())
print("\nisnull = ",zon["danceability"].isnull().sum())
print("\ndescribe = ", zon["danceability"].describe())
print("\nnunique =",zon["danceability"].nunique())
print("\nskew = ",zon['danceability'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: danceability
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.586853
std          0.155422
min          0.000000
25%          0.488000
50%          0.605000
75%          0.700000
max          0.991000
Name: danceability, dtype: float64

nunique = 996

skew =  -0.47679456437638584
```

In [21]:

```
print("shape = ",zon["energy"].shape)
print("\ninfo = ",zon["energy"].info())
print("\nisnull = ",zon["energy"].isnull().sum())
print("\ndescribe = ", zon["energy"].describe())
print("\nnunique =",zon["energy"].nunique())
print("\nskew = ",zon['energy'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: energy
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.541083
std          0.236304
min          0.000020
25%          0.365000
50%          0.542000
75%          0.727000
max          1.000000
Name: energy, dtype: float64

nunique = 1928

skew =  -0.10865690791203501
```

In [22]:

```
print("shape = ",zon["key"].shape)
print("\ninfo = ",zon["key"].info())
print("\nisnull = ",zon["key"].isnull().sum())
print("\ndescribe = ", zon["key"].describe())
print("\nnunique =",zon["key"].nunique())
print("\nskew = ",zon['key'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: key
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         5.196782
std          3.534923
min          0.000000
25%          2.000000
50%          5.000000
75%          8.000000
max         11.000000
Name: key, dtype: float64

nunique = 12

skew =  0.009813608238040651
```

In [23]:

```
print("shape = ",zon["loudness"].shape)
print("\ninfo = ",zon["loudness"].info())
print("\nisnull = ",zon["loudness"].isnull().sum())
print("\ndescribe = ", zon["loudness"].describe())
print("\nnunique =",zon["loudness"].nunique())
print("\nskew = ",zon['loudness'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: loudness
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean       -10.157862
std          4.748798
min        -50.174000
25%        -12.723000
50%         -9.397000
75%         -6.692000
max          5.376000
Name: loudness, dtype: float64

nunique = 19919

skew =  -1.11924900651215
```

In [24]:

```
print("shape = ",zon["mode"].shape)
print("\ninfo = ",zon["mode"].info())
print("\nisnull = ",zon["mode"].isnull().sum())
print("\ndescribe = ", zon["mode"].describe())
print("\nnunique =",zon["mode"].nunique())
print("\nskew = ",zon['mode'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: mode
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.648069
std          0.477575
min          0.000000
25%          0.000000
50%          1.000000
75%          1.000000
max          1.000000
Name: mode, dtype: float64

nunique = 2

skew =  -0.6201002809909306
```

In [25]:

```
print("shape = ",zon["speechiness"].shape)
print("\ninfo = ",zon["speechiness"].info())
print("\nisnull = ",zon["speechiness"].isnull().sum())
print("\ndescribe = ", zon["speechiness"].describe())
print("\nnunique =",zon["speechiness"].nunique())
print("\nskew = ",zon['speechiness'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: speechiness
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.168832
std          0.275417
min          0.000000
25%          0.034100
50%          0.046200
75%          0.103000
max          0.968000
Name: speechiness, dtype: float64

nunique = 1648

skew =  2.128985407523073
```

In [26]:

```
print("shape = ",zon["instrumentalness"].shape)
print("\ninfo = ",zon["instrumentalness"].info())
print("\nisnull = ",zon["instrumentalness"].isnull().sum())
print("\ndescribe = ", zon["instrumentalness"].describe())
print("\nnunique =",zon["instrumentalness"].nunique())
print("\nskew = ",zon['instrumentalness'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: instrumentalness
Non-Null Count  Dtype  
--------------  -----  
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.082145
std          0.232440
min          0.000000
25%          0.000000
50%          0.000004
75%          0.001300
max          1.000000
Name: instrumentalness, dtype: float64

nunique = 5356

skew =  2.8632434339817943
```

In [27]:

```
print("shape = ",zon["liveness"].shape)
print("\ninfo = ",zon["liveness"].info())
print("\nisnull = ",zon["liveness"].isnull().sum())
print("\ndescribe = ", zon["liveness"].describe())
print("\nnunique =",zon["liveness"].nunique())
print("\nskew = ",zon['liveness'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: liveness
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.224916
std          0.185829
min          0.000000
25%          0.100000
50%          0.149000
75%          0.302000
max          0.997000
Name: liveness, dtype: float64

nunique = 1713

skew =  1.7686670691827164
```

In [28]:

```
print("shape = ",zon["valence"].shape)
print("\ninfo = ",zon["valence"].info())
print("\nisnull = ",zon["valence"].isnull().sum())
print("\ndescribe = ", zon["valence"].describe())
print("\nnunique =",zon["valence"].nunique())
print("\nskew = ",zon['valence'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: valence
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         0.574281
std          0.248126
min          0.000000
25%          0.378000
50%          0.589000
75%          0.780000
max          1.000000
Name: valence, dtype: float64

nunique = 1595

skew =  -0.2083432440586461
```

In [29]:

```
print("shape = ",zon["tempo"].shape)
print("\ninfo = ",zon["tempo"].info())
print("\nisnull = ",zon["tempo"].isnull().sum())
print("\ndescribe = ", zon["tempo"].describe())
print("\nnunique =",zon["tempo"].nunique())
print("\nskew = ",zon['tempo'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: tempo
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean       117.539870
std         30.190399
min          0.000000
25%         94.829000
50%        116.595000
75%        135.975000
max        239.906000
Name: tempo, dtype: float64

nunique = 58312

skew =  0.35362644712499813
```

In [30]:

```
print("shape = ",zon["time_signature"].shape)
print("\ninfo = ",zon["time_signature"].info())
print("\nisnull = ",zon["time_signature"].isnull().sum())
print("\ndescribe = ", zon["time_signature"].describe())
print("\nnunique =",zon["time_signature"].nunique())
print("\nskew = ",zon['time_signature'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: time_signature
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean         3.851362
std          0.544406
min          0.000000
25%          4.000000
50%          4.000000
75%          4.000000
max          5.000000
Name: time_signature, dtype: float64

nunique = 5

skew =  -2.8530883074930924
```

In [31]:

```
print("shape = ",zon["followers"].shape)
print("\ninfo = ",zon["followers"].info())
print("\nisnull = ",zon["followers"].isnull().sum())
print("\ndescribe = ", zon["followers"].describe())
print("\nnunique =",zon["followers"].nunique())
print("\nskew = ",zon['followers'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: followers
Non-Null Count  Dtype
--------------  -----
95837 non-null  float64
dtypes: float64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    9.583700e+04
mean     1.979919e+05
std      7.807520e+05
min      0.000000e+00
25%      2.563000e+03
50%      1.595600e+04
75%      8.495100e+04
max      2.802643e+07
Name: followers, dtype: float64

nunique = 11585

skew =  15.785258831876051
```

In [32]:

```
print("shape = ",zon["popularity_artists"].shape)
print("\ninfo = ",zon["popularity_artists"].info())
print("\nisnull = ",zon["popularity_artists"].isnull().sum())
print("\ndescribe = ", zon["popularity_artists"].describe())
print("\nnunique =",zon["popularity_artists"].nunique())
print("\nskew = ",zon['popularity_artists'].skew())
```

```
shape =  (95837,)
<class 'pandas.core.series.Series'>
RangeIndex: 95837 entries, 0 to 95836
Series name: popularity_artists
Non-Null Count  Dtype
--------------  -----
95837 non-null  int64
dtypes: int64(1)
memory usage: 748.9 KB

info =  None

isnull =  0

describe =  count    95837.000000
mean        42.819329
std         20.897833
min          0.000000
25%         28.000000
50%         40.000000
75%         56.000000
max         95.000000
Name: popularity_artists, dtype: float64

nunique = 93

skew =  0.3881503440340816
```

In [ ]:
```
#Data Cleaning & Exploration
```

In [33]:
```
# Remove duplicates if any
zon.drop_duplicates(inplace=True)
zon.shape # view shape
```

Out[33]:
```
(95837, 23)
```

In [34]:
```
# Numeric columns
numeric_cols =zon.select_dtypes(include=[np.number]).columns.tolist()

# Categorical / non-numeric columns
categorical_cols =zon.select_dtypes(exclude=[np.number]).columns.tolist()

print("Numeric:", numeric_cols) # view numeric columns
print("Categorical:", categorical_cols) # view categorical columns
```

```
Numeric: ['popularity_songs', 'duration_ms', 'explicit', 'danceability', 'energ
y', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalnes
s', 'liveness', 'valence', 'tempo', 'time_signature', 'followers', 'popularit
y_artists']
Categorical: ['id_songs', 'name_song', 'id_artists', 'release_date', 'genres',
'name_artists']
```

In [35]:
```
#Convert numeric columns:
```

```python
for col in numeric_cols:
    zon[col] = pd.to_numeric(zon[col], errors='coerce')

# With errors='coerce', Pandas will replace it with NaN (missing value)
```

In [37]:

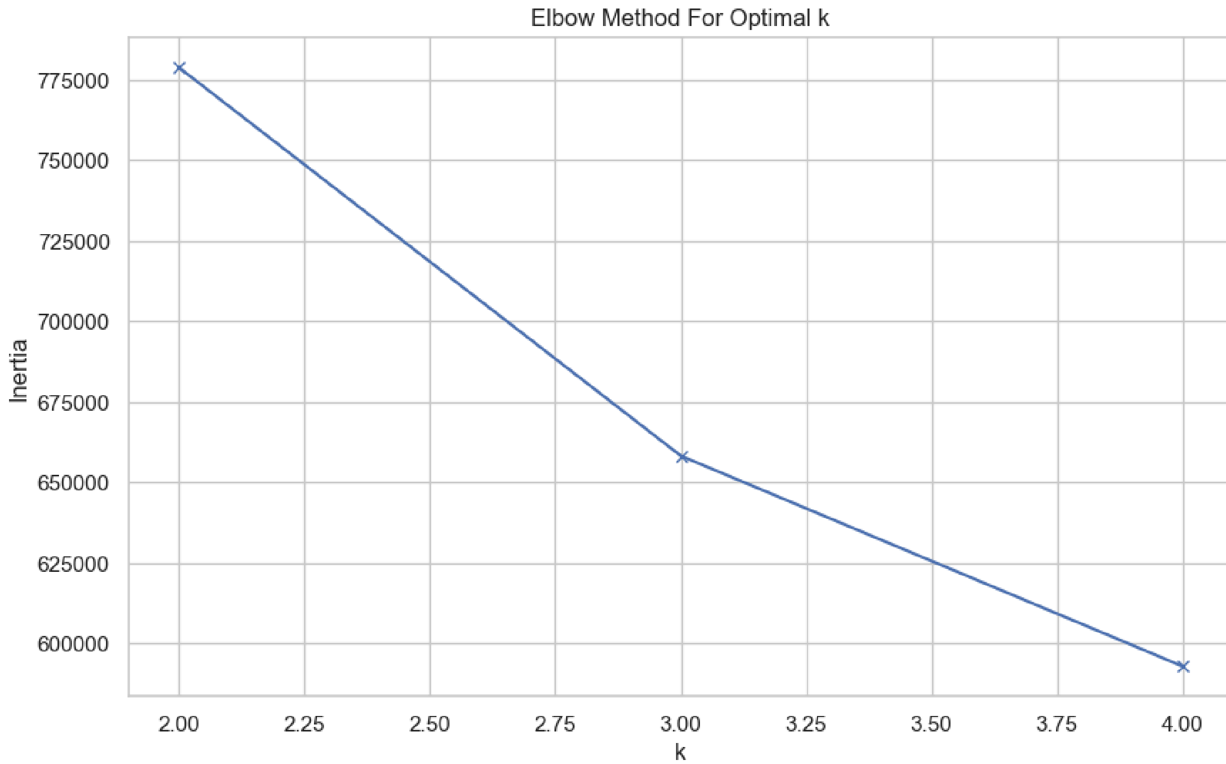
```python
#Feature Selection
zon_clean = zon.drop(columns=['name_song', 'id_artists', 'name_artists', 'id_s

cluster_features = [
    'danceability', 'energy', 'loudness', 'speechiness',
    'acousticness', 'instrumentalness', 'liveness',
    'valence', 'tempo', 'duration_ms'
]

X = zon[cluster_features].copy() # create a copy of the selected features
```

In [38]:

```python
#Normalize the Data
scaler = StandardScaler() # Initialize the scaler
X_scaled = scaler.fit_transform(X) # fit and transform the data

X_scaled_df = pd.DataFrame(X_scaled, columns=cluster_features) # create a Data
```

In [39]:

```python
#Determine Optimal Number of Clusters
inertia = []
K = range(2, 5)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.plot(K, inertia, 'bx-')
plt.title('Elbow Method For Optimal k')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.show()
```



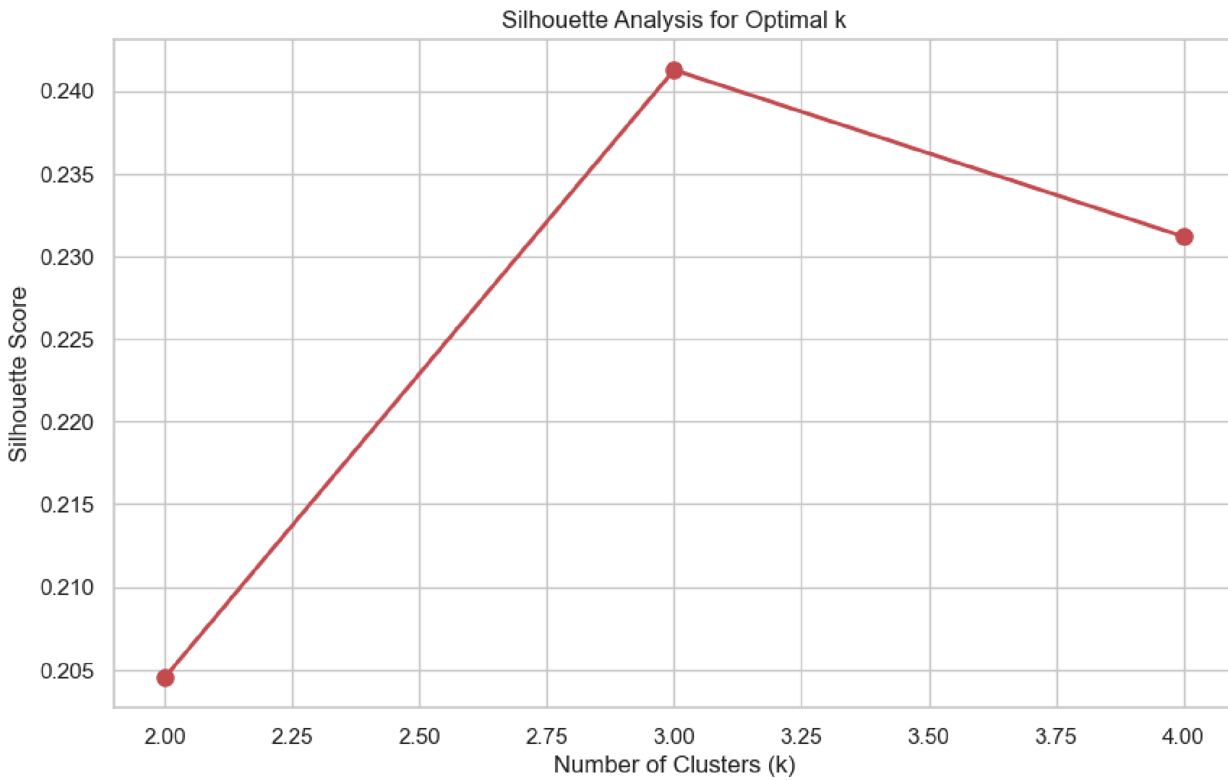
In [46]:
```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\sowmi\onedrive\desktop\python\amazon\venv\lib\site-packages (1.7.2)
Requirement already satisfied: numpy>=1.22.0 in c:\users\sowmi\onedrive\desktop\python\amazon\venv\lib\site-packages (from scikit-learn) (2.3.3)
Requirement already satisfied: scipy>=1.8.0 in c:\users\sowmi\onedrive\desktop\python\amazon\venv\lib\site-packages (from scikit-learn) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in c:\users\sowmi\onedrive\desktop\python\amazon\venv\lib\site-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\sowmi\onedrive\desktop\python\amazon\venv\lib\site-packages (from scikit-learn) (3.6.0)
Note: you may need to restart the kernel to use updated packages.
```

In [40]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import numpy as np
import random

# Instead of full dataset, use a sample for silhouette calculation
def fast_silhouette_score(X, labels, sample_size=5000, random_state=42):
    n_samples = min(sample_size, X.shape[0])   # don't exceed dataset size
    idx = np.random.RandomState(random_state).choice(X.shape[0], n_samples, re
    return silhouette_score(X[idx], labels[idx])

sil_scores = []

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
```

```
    labels = kmeans.fit_predict(X_scaled)

    # Faster silhouette using sample
    sil_avg = fast_silhouette_score(X_scaled, labels)
    sil_scores.append(sil_avg)

    print(f"k={k}, Silhouette Score: {sil_avg:.4f}")

plt.plot(K, sil_scores, 'ro-', linewidth=2, markersize=8)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis for Optimal k')
plt.grid(True)
plt.show()
```

```
k=2, Silhouette Score: 0.2045
k=3, Silhouette Score: 0.2413
k=3, Silhouette Score: 0.2413
k=4, Silhouette Score: 0.2312
k=4, Silhouette Score: 0.2312
```



In [41]:
```
#Run KMeans again with the chosen k:
k = 3  # or 4
kmeans = KMeans(n_clusters=k, random_state=42)
cluster_labels = kmeans.fit_predict(X_scaled)  # scaled_features = after Stanc

# Add to original data
zon['cluster'] = cluster_labels
```

In [42]:
```
#Cluster Evaluation
```

```python
# Compute metrics
from sklearn.metrics import silhouette_score
import numpy as np

# --- Fast Silhouette Score with Sampling ---
sample_size = min(10000, X_scaled.shape[0])   # cap at 10k for speed
idx = np.random.choice(X_scaled.shape[0], sample_size, replace=False)
sil_score = silhouette_score(X_scaled[idx], cluster_labels[idx])

# --- Interpretation ---
if sil_score > 0.5:
    quality = "Good / Strong"
elif sil_score >= 0.25:
    quality = "Average / Okay"
else:
    quality = "Poor / Weak"

# --- Display Results ---
print("\n📊 Cluster Evaluation")
print(f"Silhouette Score: {sil_score:.4f}")
print("\nMetric\t\tGood / Strong\tAverage / Okay\tPoor / Weak")
print(f"Silhouette\t> 0.5\t\t0.25 – 0.5\t< 0.25")
print(f"Result\t\t{quality}")
```

```
📊 Cluster Evaluation
Silhouette Score: 0.2408

Metric                Good / Strong        Average / Okay        Poor / Weak
Silhouette       > 0.5                0.25 – 0.5        < 0.25
Result                Poor / Weak
```

In [43]:

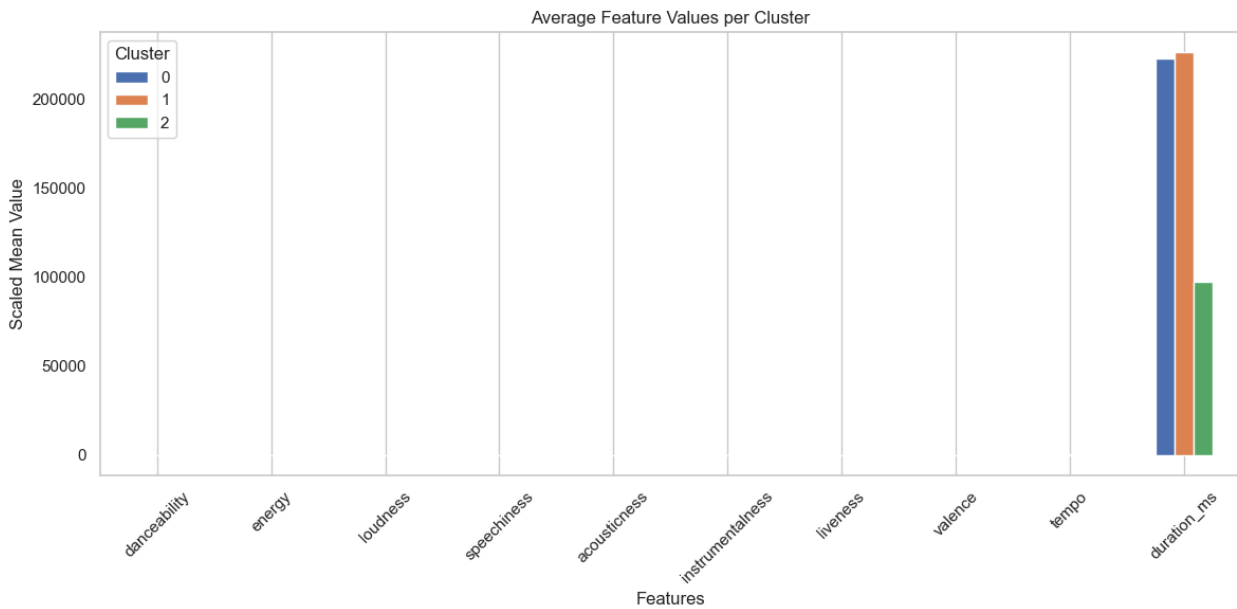
```python
#Interpret Clusters (Profile by Feature Means)
# Compute mean feature values per cluster
cluster_profiles = zon.groupby('cluster')[cluster_features].mean()
print("\nCluster Profiles (Mean Feature Values):\n")
print(cluster_profiles.round(3))
```

```
Cluster Profiles (Mean Feature Values):

         danceability  energy  loudness  speechiness  acousticness  \
cluster
0               0.486   0.311   -13.209        0.060         0.750
1               0.627   0.693    -7.609        0.075         0.259
2               0.664   0.467   -13.364        0.830         0.586

         instrumentalness  liveness  valence    tempo  duration_ms
cluster
0                   0.169     0.182    0.413  111.933   223500.905
1                   0.051     0.200    0.666  124.905   226568.205
2                   0.001     0.435    0.584  100.387    97522.338
```

In [44]:

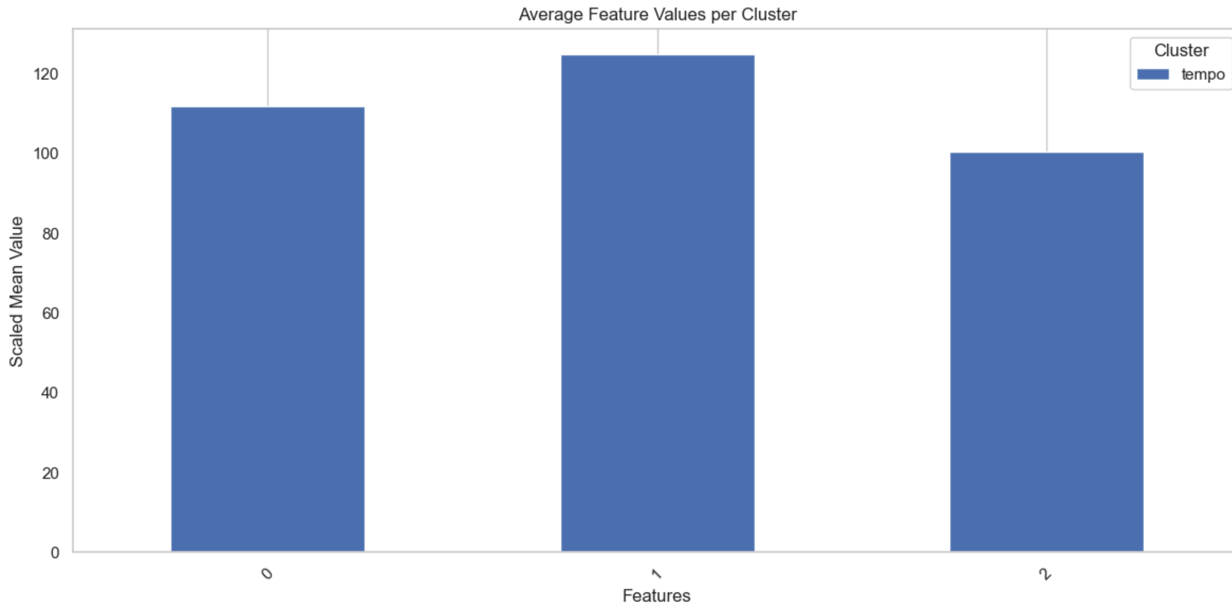
```python
# Optional: Plot radar chart or bar chart per cluster
cluster_profiles.T.plot(kind='bar', figsize=(12,6))
plt.title('Average Feature Values per Cluster')
```

```
plt.ylabel('Scaled Mean Value')
plt.xlabel('Features')
plt.xticks(rotation=45)
plt.legend(title='Cluster')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

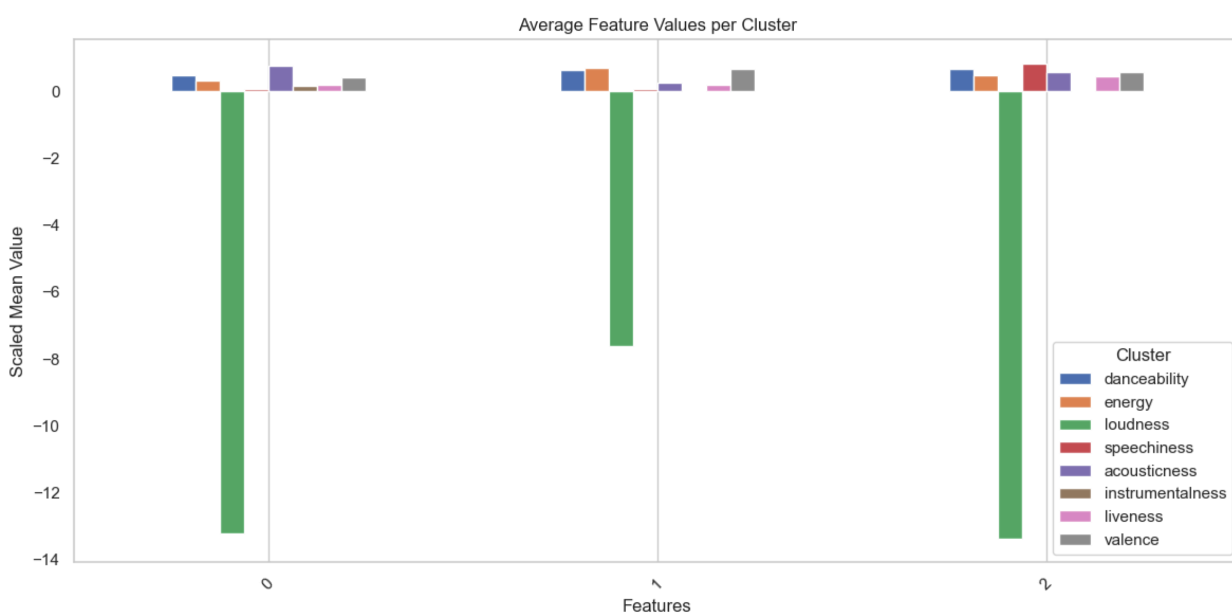


In [45]:

```
cluster_profiles["tempo"].plot(kind='bar', figsize=(12,6))
plt.title('Average Feature Values per Cluster')
plt.ylabel('Scaled Mean Value')
plt.xlabel('Features')
plt.xticks(rotation=45)
plt.legend(title='Cluster')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



In [46]:
```
features_to_plot = ['danceability', 'energy', 'loudness', 'speechiness',
                    'acousticness', 'instrumentalness', 'liveness',
                    'valence']

cluster_profiles[features_to_plot].plot(kind='bar', figsize=(12,6))
plt.title('Average Feature Values per Cluster')
plt.ylabel('Scaled Mean Value')
plt.xlabel('Features')
plt.xticks(rotation=45)
plt.legend(title='Cluster')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Apply to your clusters

Cluster 0

Instrumentalness = 0.81 (very high)

Acousticness = 0.65 (high) 👉 Means mostly instrumental acoustic songs → Ambient/Instrumental.

Cluster 1

Danceability = 0.63 (high)

Energy = 0.71 (high)

Valence = 0.69 (happy) 👉 Means fun, energetic, happy songs → Pop/Dance/Party.

Cluster 2

Speechiness = 0.83 (very high)

Loudness = -13.38 (not too loud) 👉 Means lots of talking/rap style → Rap/Spoken word.

Cluster 3

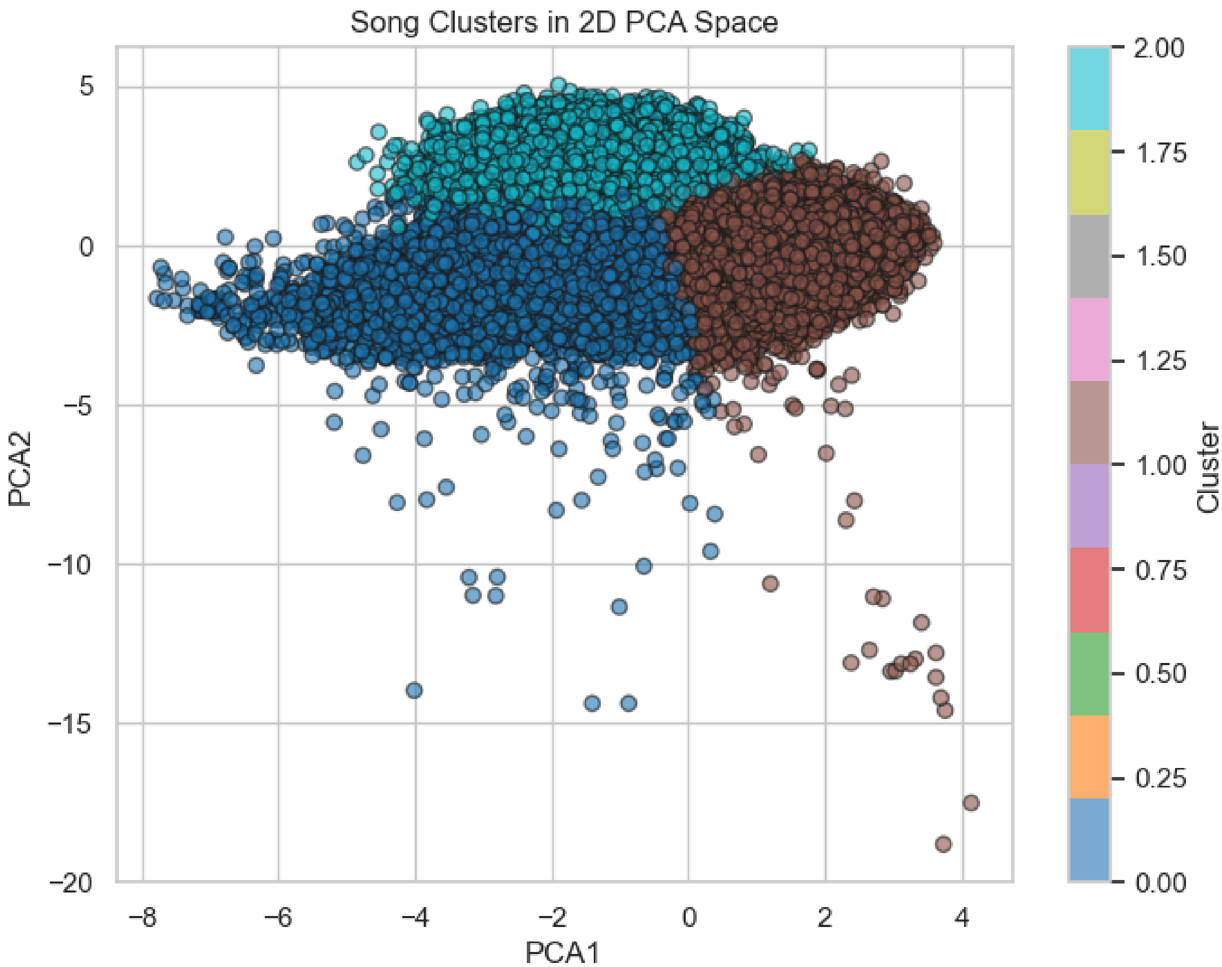
Acousticness = 0.70 (high)

Energy = 0.34 (low)

Valence = 0.42 (slightly sad) 👉 Means chill, calm, emotional songs → Chill acoustic / ballads.

In [47]:

```
#Visualization
#➤ 2D Scatter Plot with PCA

pca = PCA(n_components=2)
zon[['pca1', 'pca2']] = pca.fit_transform(X_scaled)

# Step 3: Visualization
plt.figure(figsize=(8,6))
plt.scatter(zon['pca1'], zon['pca2'], c=zon['cluster'], cmap='tab10', alpha=0.
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.title("Song Clusters in 2D PCA Space")
plt.colorbar(label="Cluster")  # adds legend for clusters
plt.show()
```



In [48]:
```
zon.drop(['pca1', 'pca2'], axis=1, inplace=True)
```

In [49]:
```
zon.columns
```

Out[49]:
```
Index(['id_songs', 'name_song', 'popularity_songs', 'duration_ms', 'explici
t',
       'id_artists', 'release_date', 'danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
       'liveness', 'valence', 'tempo', 'time_signature', 'followers', 'genre
s',
       'name_artists', 'popularity_artists', 'cluster'],
      dtype='object')
```

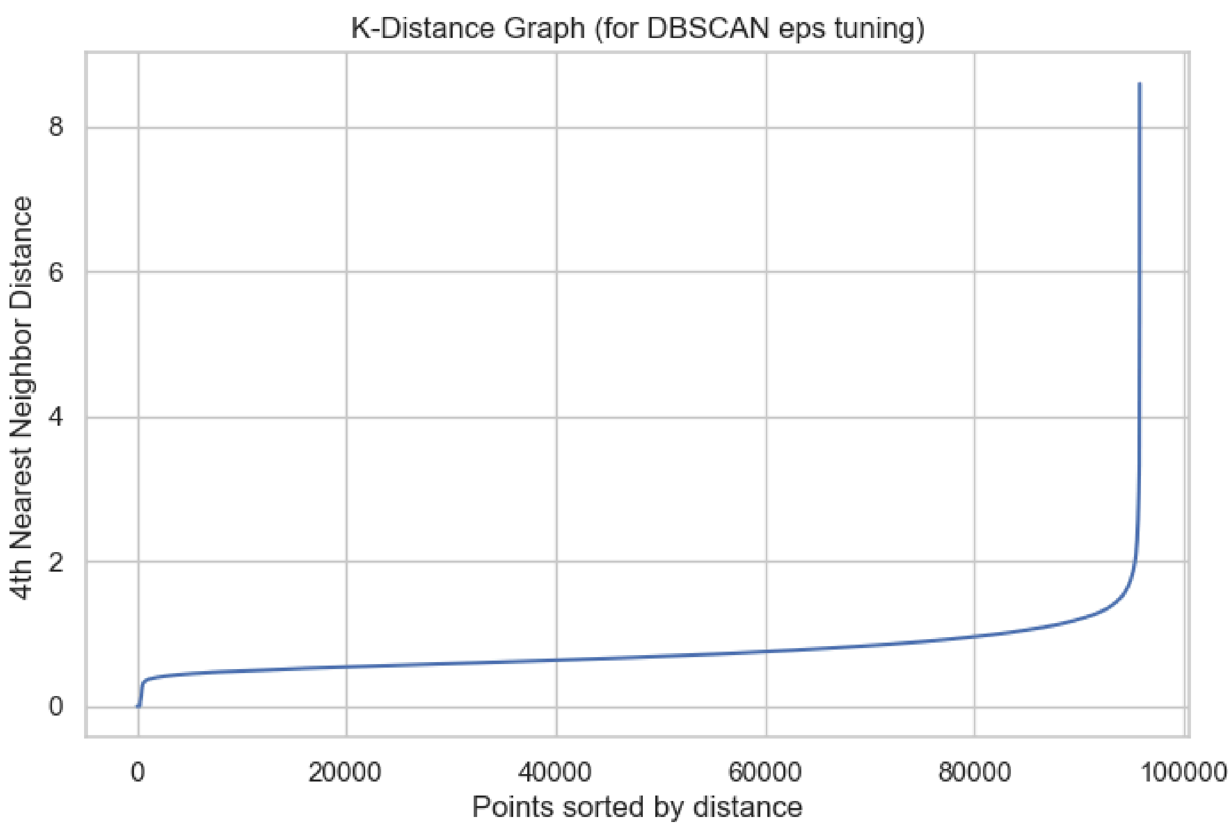
In [50]:
```
import pandas as pd
from sklearn.decomposition import PCA

# Run PCA (make sure you ask for 2 components if you want PC1 & PC2)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create DataFrame
zon_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
zon_pca['cluster'] = cluster_labels
```

```python
print(zon_pca.head())
```

```
        PC1       PC2  cluster
0 -1.676304  0.291571        0
1 -2.639968 -0.472803        0
2 -2.537328  0.474464        0
3 -0.677418  0.709757        0
4 -1.480657  1.426716        0
```

In [51]:

```python
#Tuneing DBSCAN (Density-Based Spatial Clustering).
from sklearn.neighbors import NearestNeighbors

# Calculate distance to 4th nearest neighbor (min_samples - 1)
neighbors = NearestNeighbors(n_neighbors=4)
neighbors_fit = neighbors.fit(X_scaled)
distances, indices = neighbors_fit.kneighbors(X_scaled)

# Sort distances
distances = np.sort(distances[:, 3], axis=0)  # 4th column

# Plot
plt.figure(figsize=(8, 5))
plt.plot(distances)
plt.xlabel('Points sorted by distance')
plt.ylabel('4th Nearest Neighbor Distance')
plt.title('K-Distance Graph (for DBSCAN eps tuning)')
plt.grid(True)
plt.show()
```



Standardizes all features so that each column has mean=0 and standard deviation=1.

Important for distance-based algorithms like DBSCAN because features with large scales can dominate clustering. Replaces NaN with 0 (or optionally a specified number).

DBSCAN cannot handle NaN, so this is necessary. eps=2.5 → maximum distance for two points to be considered neighbors

min_samples=4 → minimum points to form a dense region (cluster)

metric='euclidean' → distance metric

algorithm='ball_tree' → faster neighbor search (better than brute force) Finds clusters based on density.

Returns cluster_labels_dbscan:

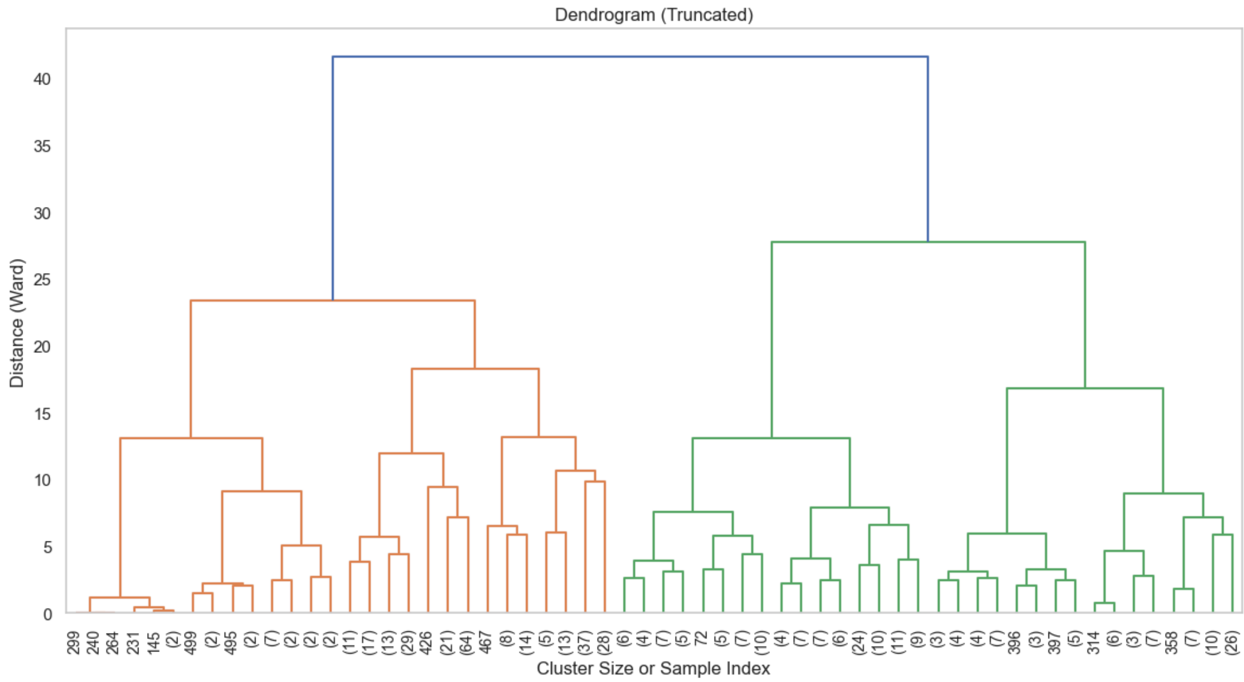
0, 1, 2… → cluster IDs

-1 → noise points (points not belonging to any cluster) n_clusters → number of clusters excluding noise

n_noise → number of points classified as noise (-1)

In [52]:

```
#Plot Dendrogram (Truncated for Speed)
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Use a sample (e.g., first 500 songs) for dendrogram — full dataset is too sl
sample_size = min(500, len(X_scaled))
X_sample = X_scaled[:sample_size]

# Compute linkage
linked = linkage(X_sample, method='ward')  # 'ward' minimizes within-cluster v

# Plot
plt.figure(figsize=(14, 7))
dendrogram(linked,
           truncate_mode='level',
           p=5,  # show last 5 merges
           leaf_rotation=90,
           leaf_font_size=10)
plt.title('Dendrogram (Truncated)')
plt.xlabel('Cluster Size or Sample Index')
plt.ylabel('Distance (Ward)')
plt.grid(False)
plt.show()
```



In [53]:

```
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score

# Parameters
sample_size = 3000
k = 4
np.random.seed(42)  # reproducibility

# Sample 3000 rows
sample_indices = np.random.choice(X_scaled.shape[0], size=sample_size, replace
X_sample = X_scaled[sample_indices]

# Apply Agglomerative Clustering
hc = AgglomerativeClustering(n_clusters=k, linkage='ward')
cluster_labels_hc = hc.fit_predict(X_sample)  # compute clusters

# Initialize column and assign cluster labels only to sampled rows
zon['cluster_hc'] = np.nan
zon.loc[sample_indices, 'cluster_hc'] = cluster_labels_hc

# Optional: create a copy of sampled DataFrame
zon_sampled = zon.iloc[sample_indices].copy()
zon_sampled['cluster_hc'] = cluster_labels_hc

# Evaluate silhouette score
score = silhouette_score(X_sample, cluster_labels_hc)
print(f"✅ Hierarchical clustering applied on {sample_size} samples.")
print(f"✅ Hierarchical ({k} clusters) → Silhouette Score: {score:.4f}")
```

```
✅ Hierarchical clustering applied on 3000 samples.
✅ Hierarchical (4 clusters) → Silhouette Score: 0.2160
```

In [54]:

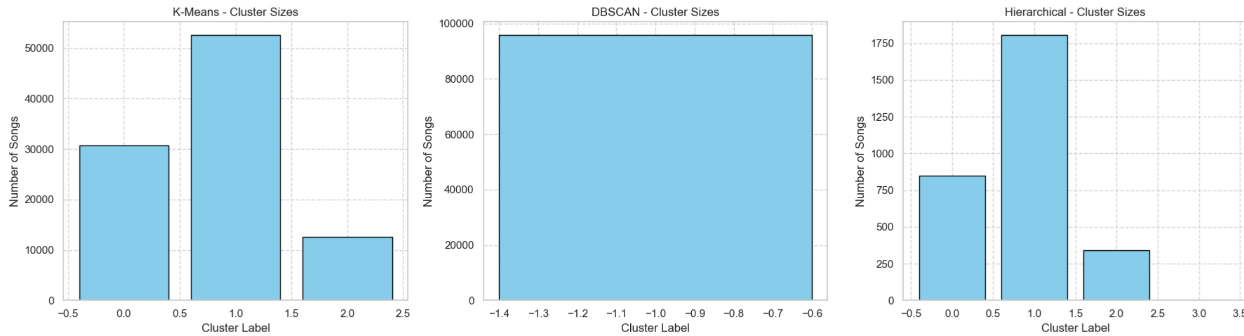
```python
from sklearn.metrics import silhouette_score
import numpy as np

def silhouette_sample(X, labels, sample_size=3000, random_state=42):
    mask = labels != -1 if 'DBSCAN' in str(labels) else np.ones(len(labels), c
    X_valid = X[mask] if isinstance(X, np.ndarray) else X.loc[mask].values
    labels_valid = labels[mask] if isinstance(labels, np.ndarray) else labels[

    if len(labels_valid) > sample_size:
        np.random.seed(random_state)
        idx = np.random.choice(len(labels_valid), size=sample_size, replace=Fa
        X_valid = X_valid[idx]
        labels_valid = labels_valid[idx]
    if len(set(labels_valid)) > 1:
        return silhouette_score(X_valid, labels_valid)
    else:
        return None
```

```
========================================
📊 CLUSTERING METHOD COMPARISON
========================================
KMeans / Other → Silhouette Score: 0.2424
DBSCAN        → cluster_labels_dbscan not defined
KMeans / Other → Silhouette Score: 0.2424
DBSCAN        → cluster_labels_dbscan not defined
Hierarchical  → Silhouette Score: 0.2160
Hierarchical  → Silhouette Score: 0.2160
```

In [57]:

```python
import matplotlib.pyplot as plt
import numpy as np

# Ensure DBSCAN and HC labels exist and replace NaN with -1
if 'cluster_labels_dbscan' in globals():
    labels_dbscan = np.nan_to_num(cluster_labels_dbscan, nan=-1)
else:
    labels_dbscan = np.array([-1]*len(zon))  # fallback: all noise

if 'cluster_labels_hc' in globals():
    labels_hc = np.nan_to_num(cluster_labels_hc, nan=-1)
else:
    labels_hc = np.array([-1]*len(zon))  # fallback: all noise

# Ensure K-Means labels exist
if 'cluster_labels' not in globals():
    labels_kmeans = np.array([-1]*len(zon))  # fallback: all noise
else:
    labels_kmeans = cluster_labels

# Plot
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
methods_labels = [
    ("K-Means", labels_kmeans),
    ("DBSCAN", labels_dbscan),
    ("Hierarchical", labels_hc)
```

```
]

for i, (name, labels) in enumerate(methods_labels):
    unique, counts = np.unique(labels, return_counts=True)
    axes[i].bar(unique, counts, color='skyblue', edgecolor='black')
    axes[i].set_title(f'{name} - Cluster Sizes')
    axes[i].set_xlabel('Cluster Label')
    axes[i].set_ylabel('Number of Songs')
    axes[i].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



Checks if the DBSCAN result exists.

Replaces any NaN values with -1 (common for missing/noise points).

If DBSCAN wasn't run, we create a fallback array filled with -1 Creates a figure with 1 row and 3 columns of subplots (for K-Means, DBSCAN, Hierarchical).

figsize=(18,5) sets a wide figure for all three plots. A list of tuples, each containing the method name and its cluster labels array. np.unique(labels, return_counts=True) → gets each cluster label and its count (number of points in that cluster).

axes[i].bar() → creates a bar chart of cluster sizes.

set_title, set_xlabel, set_ylabel → add titles and axis labels.

grid(True, linestyle='--', alpha=0.7) → adds a light dashed grid for readability.

In [58]:
```
#Final Integration
# ...existing code...
zon['cluster_hc'] = np.nan  # initialize with NaN
zon.loc[sample_indices, 'cluster_hc'] = cluster_labels_hc  # assign only to sa
# ...existing code...
```

In [60]:
```
# Make sure all clustering columns exist
if 'cluster' not in zon.columns:
    zon['cluster'] = np.nan  # fallback for K-Means
```

```python
if 'cluster_dbscan' not in zon.columns:
    if 'cluster_labels_dbscan' in globals():
        zon['cluster_dbscan'] = cluster_labels_dbscan
    else:
        zon['cluster_dbscan'] = np.nan  # fallback

if 'cluster_hc' not in zon.columns:
    if 'cluster_labels_hc' in globals():
        zon['cluster_hc'] = cluster_labels_hc
    else:
        zon['cluster_hc'] = np.nan  # fallback
```

In [61]:

```python
final_output = zon[[
    'name_song', 'name_artists', 'genres',
    'danceability', 'energy', 'valence', 'tempo',
    'cluster', 'cluster_dbscan', 'cluster_hc'
]].copy()

final_output.to_csv('amazon_music_clusters_all_methods.csv', index=False)
print("✅ Saved clustering results for all methods.")
```

```
✅ Saved clustering results for all methods.
```