```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('dataset.csv')
```

```
df.sample(5)
```

	Unnamed: 0	track_id	artists	album_name	track_name	populai
69866	69866	3hW6C2zvNurb9gIR3tckrV	K. G. Markose	Marian	Jeevanekidum	
83100	83100	0b6wdul3A5sQNpIOv03OxP	Duke Dumont	Ocean Drive	Ocean Drive	
60645	60645	29gFw3PNupQzPv0XiYrAXm	Intence	Leave If Uh Waah Leave	Leave If Uh Waah Leave	
44710	44710	5jsDxDkJ1PqyYUWhDMr86B	Stone Sour	Hydrograd	Song #3	
4160	4160	7gbSDq9luQx6yVl7HyJGIW	Peter Sandberg	Dismantle	Dismantle	

5 rows × 21 columns

```
df.shape
```

```
(114000, 21)
```

```
df.isnull().value_counts()
```

Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explici
False	**False**	**False**	**False**	**False**	**False**	**False**	**False**
		True	**True**	**True**	**False**	**False**	**False**

dtype: int64

```
df.columns
```

```
Index(['Unnamed: 0', 'track_id', 'artists', 'album_name', 'track_name',
       'popularity', 'duration_ms', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature',
       'track_genre'],
      dtype='object')
```

```
df.drop(['Unnamed: 0'],axis = 1,inplace = True)
df.sample(5)
```

	track_id	artists	album_name	track_name	popularity
19251	57buRfUBYm7fFFoIM78qbs	Kacey Musgraves	Best Alternative Pop Tunes	easier said	0
55359	4YwGnKgtq7V0EI1hYNOirm	Pritam;Shreya Ghoshal	Action Replayy	O Bekhabar	52
104445	1d6B9SauXraTqsUULNIGxD	Ana Torroja;Sentidos Opuestos	90's Pop Tour 4 (En Vivo Desde la Arena Ciudad...	Hijo de la Luna (En Vivo)	31
47201	225xvV8r1yKMHErSWivnow	Aerosmith	Armageddon - The Album	I Don't Want to Miss a Thing - From "Armageddo...	74
75651	4r48ds35jyBVxzK4gS1NLW	Himekami	Sennen Kairou	Sennen no Inori	29

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 20 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   track_id          114000 non-null  object
 1   artists           113999 non-null  object
 2   album_name        113999 non-null  object
 3   track_name        113999 non-null  object
 4   popularity        114000 non-null  int64
 5   duration_ms       114000 non-null  int64
 6   explicit          114000 non-null  bool
 7   danceability      114000 non-null  float64
 8   energy            114000 non-null  float64
 9   key               114000 non-null  int64
```

```
 10  loudness          114000 non-null  float64
 11  mode              114000 non-null  int64
 12  speechiness       114000 non-null  float64
 13  acousticness      114000 non-null  float64
 14  instrumentalness  114000 non-null  float64
 15  liveness          114000 non-null  float64
 16  valence           114000 non-null  float64
 17  tempo             114000 non-null  float64
 18  time_signature    114000 non-null  int64
 19  track_genre       114000 non-null  object
dtypes: bool(1), float64(9), int64(5), object(5)
memory usage: 16.6+ MB
```

```
df.describe()
```

	popularity	duration_ms	danceability	energy	key	lo
count	114000.000000	1.140000e+05	114000.000000	114000.000000	114000.000000	114000.
mean	33.238535	2.280292e+05	0.566800	0.641383	5.309140	-8.
std	22.305078	1.072977e+05	0.173542	0.251529	3.559987	5.
min	0.000000	0.000000e+00	0.000000	0.000000	0.000000	-49.
25%	17.000000	1.740660e+05	0.456000	0.472000	2.000000	-10.
50%	35.000000	2.129060e+05	0.580000	0.685000	5.000000	-7.
75%	50.000000	2.615060e+05	0.695000	0.854000	8.000000	-5.
max	100.000000	5.237295e+06	0.985000	1.000000	11.000000	4.

Minimum value of duration is 0, as seen from df.describe().

This is very peculiar => let's have a look at such tracks, they are probably disposable.

```
df.loc[df['duration_ms'] == 0]
```

	track_id	artists	album_name	track_name	popularity	duration_n
65900	1kR4gIb7nGxHPI3D2ifs59	NaN	NaN	NaN	0	

The above track is the only one that has any missing values => we will remove it from from the dataset.

```
df.drop(df.loc[df['duration_ms'] == 0].index, inplace = True)
df.isnull().sum()
```

```
df.isnull().sum()
```

	0
track_id	0
artists	0
album_name	0
track_name	0
popularity	0
duration_ms	0
explicit	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
track_genre	0

dtype: int64

Creating a subset df containing features that pertain to the technical aspects of a song, suspect it might come in handy later.

```
technicalities = df[['track_id','acousticness','danceability','energy','key', 'loudness',
technicalities.sample(5)
```

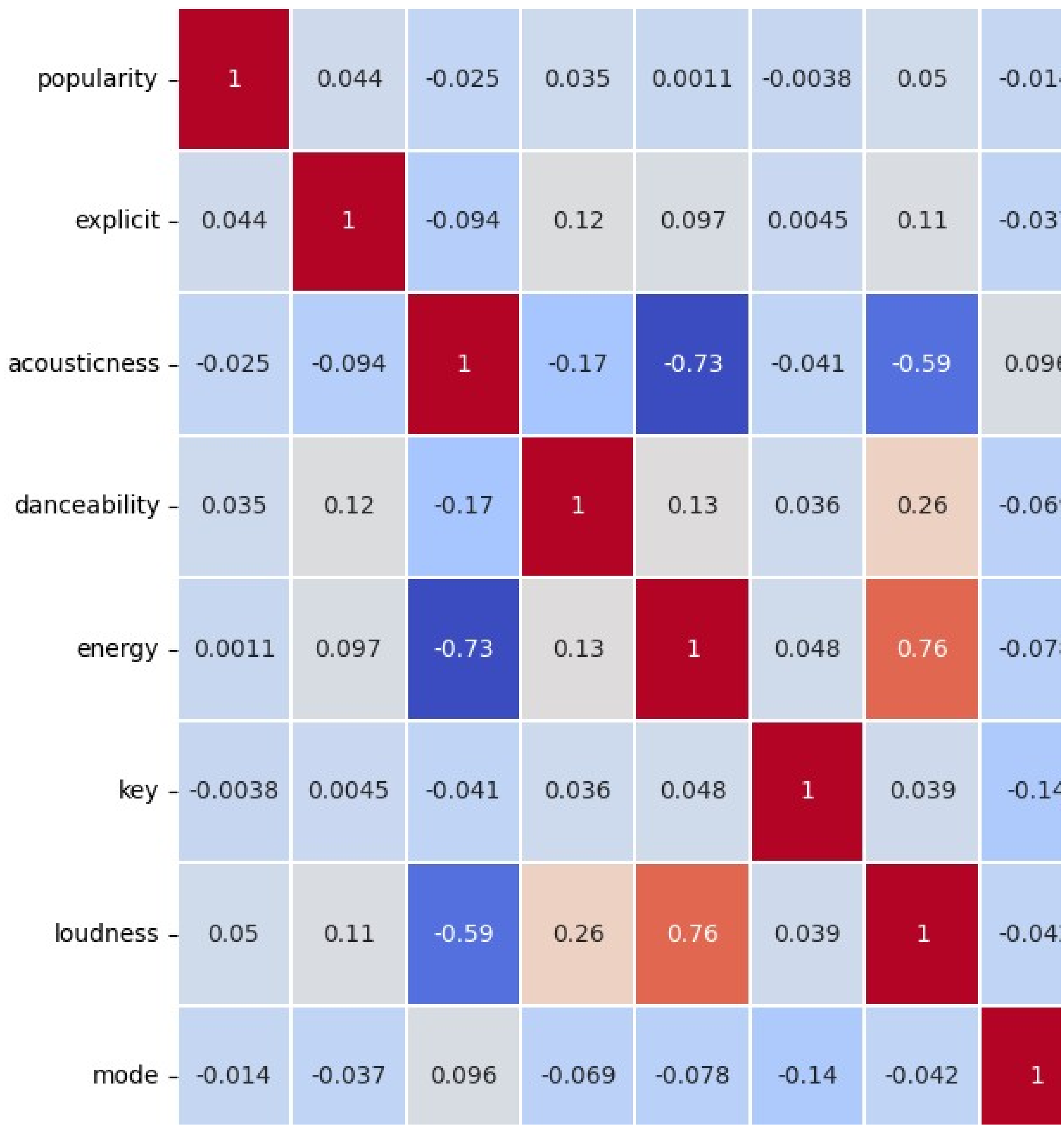
	track_id	acousticness	danceability	energy	key	loudness
62279	0YAMQSmHk6BSUGTYpaoqTJ	0.0286	0.379	0.690	4	-4.790

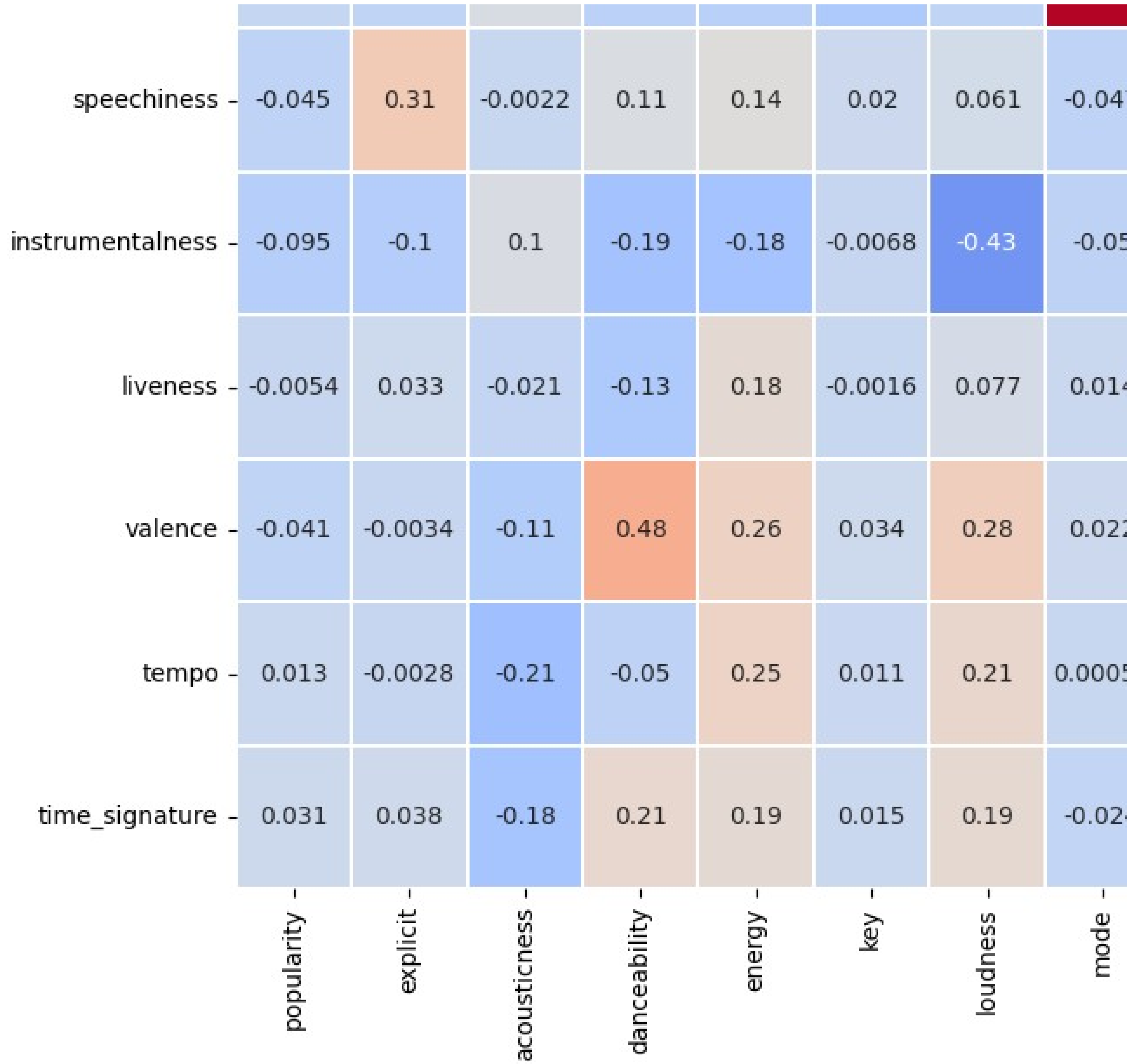
57548	7M3bjqyNXPKL3iHzvjRNdU	0.8550	0.314	0.309	2	-3.809
67733	1ObXAJokXug3fmwrVtlj7v	0.1940	0.737	0.607	0	-3.599
59477	4j6qfCtWcSVUls9zmWHuNI	0.0909	0.658	0.549	1	-17.113
113934	73EIpn5AcedCWPOmxWMxH9	0.0925	0.317	0.562	11	-8.719

Correlation Matrix

```
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(df[['popularity','explicit','acousticness','danceability','energy','key', 'lo
```

```
<Axes: >
```





OBSERVATIONS FROM CORRELATION MATRIX

1. **Acousticness vs Energy**

   Relatively Strong Negative correlation (-0.73) between "Acousticness" and "Energy" => Acoustic music is generally more laid back and less energetic.

2. **Acousticness vs Loudness**

   Moderate Negative (-0.59) => Acoustic music is generally quieter.

3. **Danceability and Valence**

   Moderarte Positive Correlation (0.48)

4. **Loudness and Energy**

Moderate to Strong Positive correlation (0.78), louder music is more energtic.

5. **Speechiness and Explicit**

   Weak to Moderate correlation (0.31) => Music containing explicit verses tends to have more verbose lyrics.

Mode column

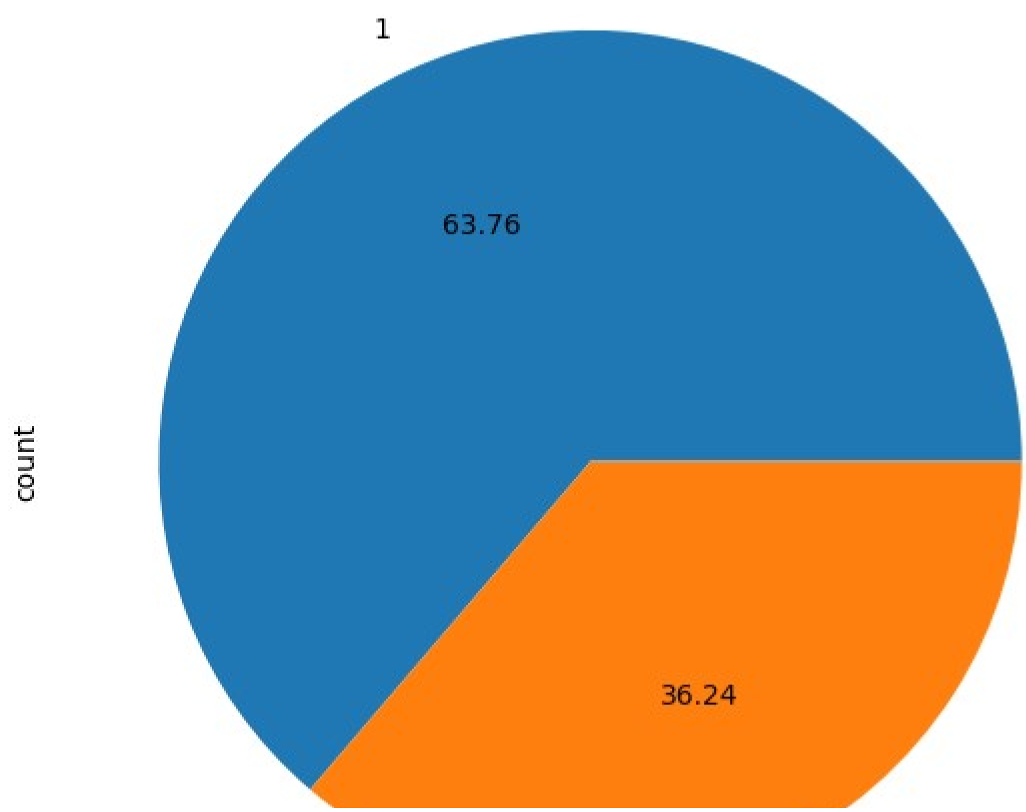
```
df['mode'].values   #only 1s and 0s
df['mode'].nunique()
```

```
2
```

Mode has only two types of values 1s and 0s (likely major and minor modes)

```
df['mode'].value_counts().plot(kind="pie",autopct = "%.2f", figsize=(7,7))
```

```
<Axes: ylabel='count'>
```



Duration

```
df['duration_m'] = df['duration_ms']/60000    #converting ms to min and eventually droppi
```

```
df.drop(['duration_ms'],axis = 1,inplace = True)
```

```
df['duration_m']
```

	duration_m
0	3.844433
1	2.493500
2	3.513767
3	3.365550
4	3.314217
...	...
113995	6.416650
113996	6.416667
113997	4.524433
113998	4.731550
113999	4.030433

113999 rows × 1 columns

dtype: float64

```
df['duration_m'].skew()
```

```
np.float64(11.195825995953085)
```

```
df['duration_m'].median()
```

```
3.5484333333333336
```

```
df['duration_m'].mode()
```

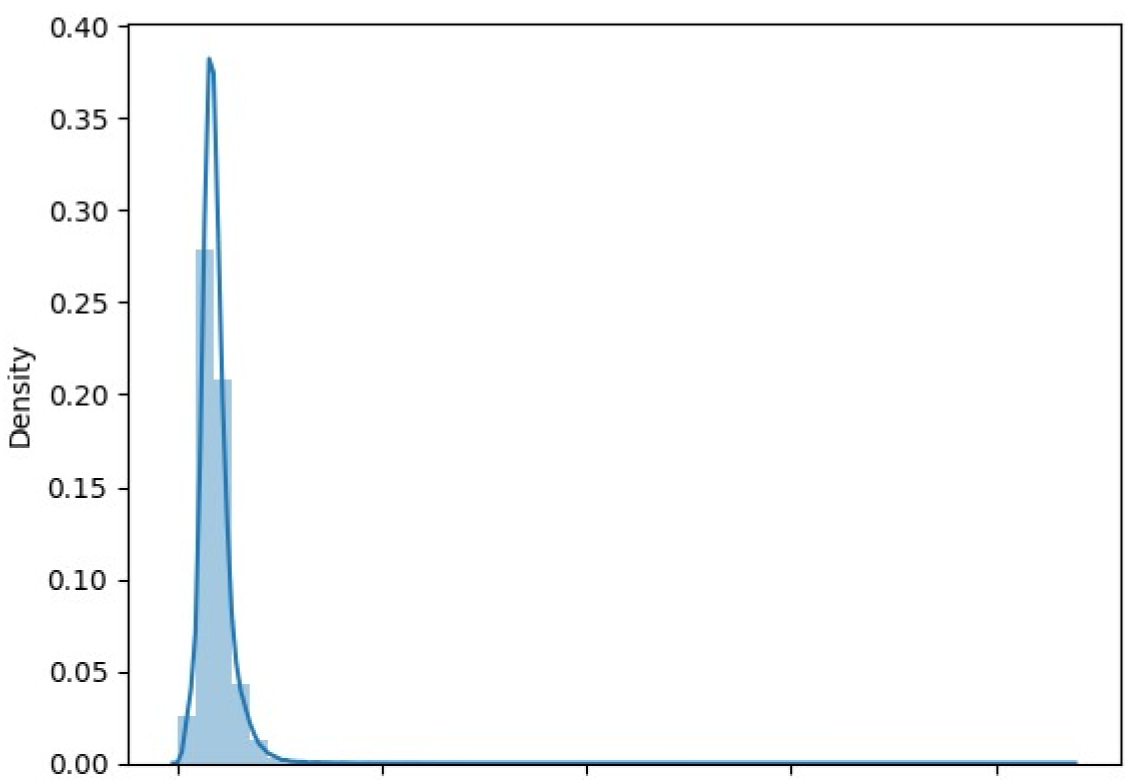
	duration_m
0	2.71495

dtype: float64

We see that the median is greater than the mode => this data is **positively skewed** (this is also exident form the following graphs)

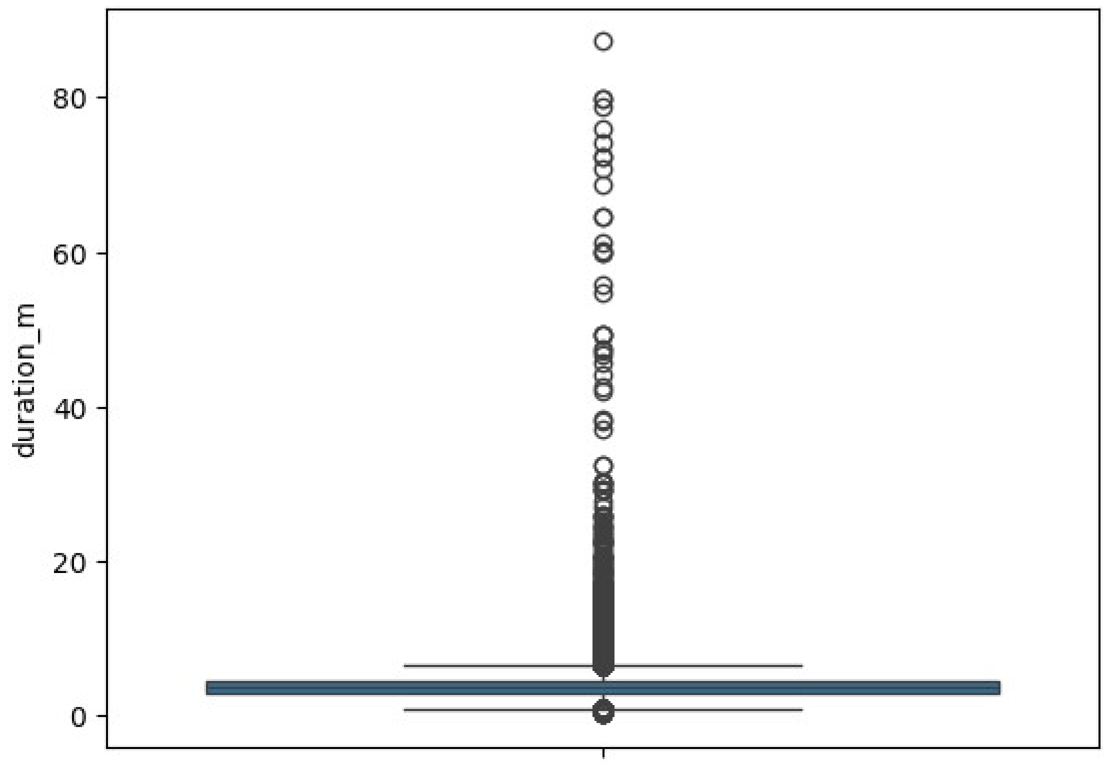
```
sns.distplot(df['duration_m'])
```

```
<ipython-input-21-add6c3771088>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['duration_m'])
<Axes: xlabel='duration_m', ylabel='Density'>
```




```
sns.boxplot(df['duration_m'])
```

```
<Axes: ylabel='duration_m'>
```



Both the distplot and boxplot tell us that MOST songs have a short duration (median = 3.54 min). However there are also several outliers. We have used the IQR method to classify points as outliers, so any duration longer than (1.5*IQR + Q3) has been branded as outlier. We can see that 5344 entries satisfy this condition => 4.6877% of total entries can be considered as outliers.

```
df['duration_m'].describe()
```

	duration_m
count	113999.000000
mean	3.800519
std	1.788268
min	0.143100
25%	2.901100
50%	3.548433
75%	4.358433

75%	4.358433
max	87.288250

dtype: float64

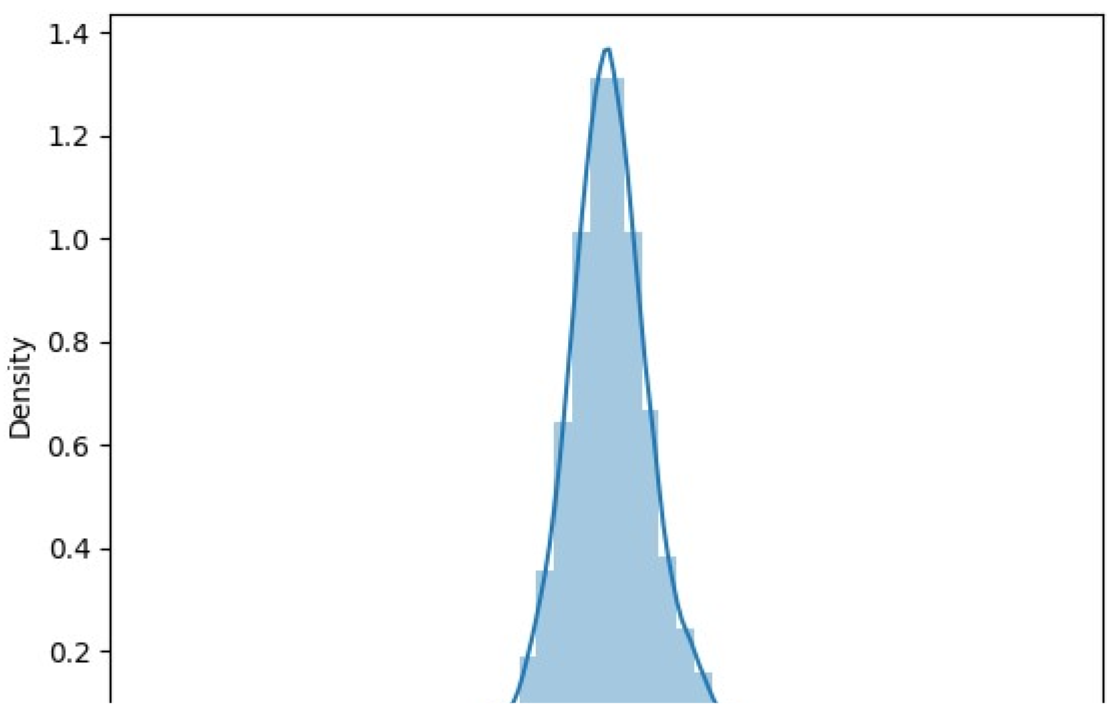
```
durationIQR = (4.358433 - 2.901100)
outliersCounts = df.loc[df['duration_m']>= (1.5*(durationIQR) + 4.358433)].value_counts()

print(f"Percentage of Outliers = {100 * outliersCounts/df.shape[0]}")
```

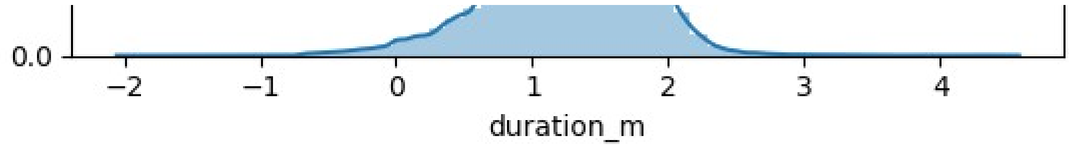
```
Percentage of Outliers = 4.687760418951044
```

The duration data is heavily right skewed as determined earlier. This means that we can apply the *LOG TRANSFORM* during the feaure engineering phase on the Duration column.

```
transfDuration = np.log(df['duration_m'])
sns.distplot(transfDuration)
```

```
<ipython-input-25-aec7c8f53997>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(transfDuration)
<Axes: xlabel='duration_m', ylabel='Density'>
```






```
transfDuration.skew()
```

```
np.float64(-0.31831854307791757)
```

```
df['duration_m'] = transfDuration
```

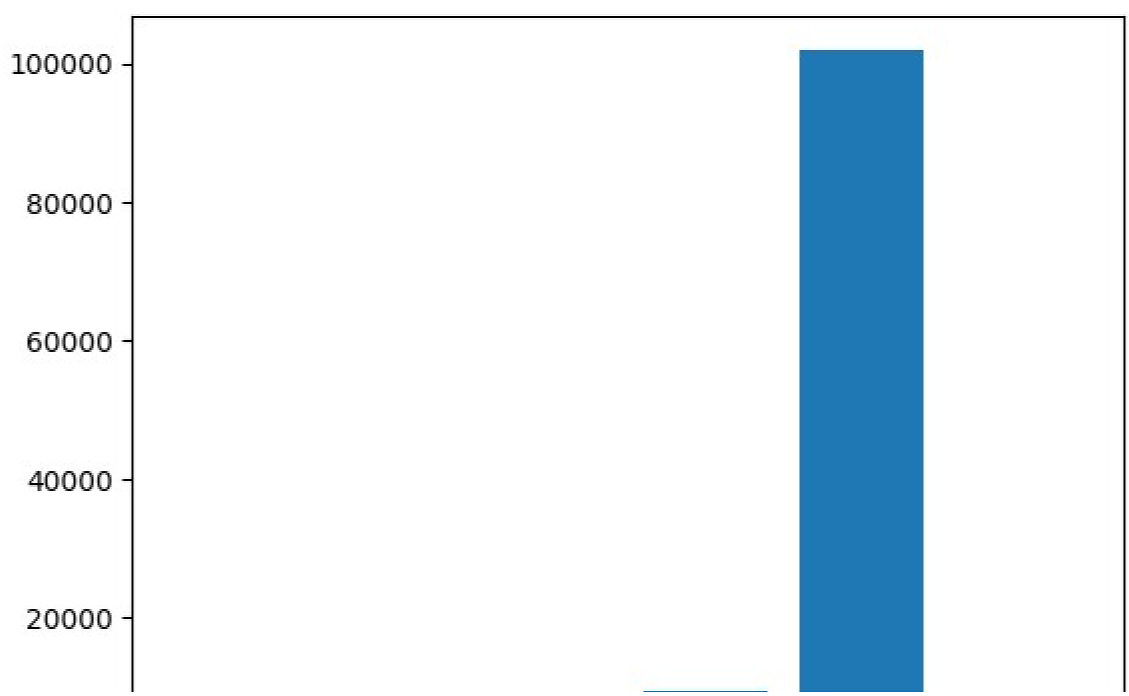
We can see that the LOG Transform has brought the data to closer to being a normal distribution. This is ocnfirmed visually thru the graph as well as mathemtically since the skew value of the transformed data is now much closer to 0.

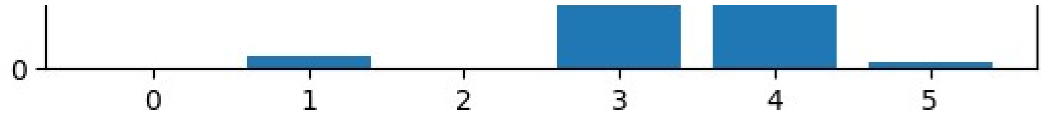
Time Signature

Anlaysing common Time Signatures, we see from the following graphs that 4/4 is by far the most commonly used time signature.

```
plt.bar(df['time_signature'].unique(),df['time_signature'].value_counts())
```

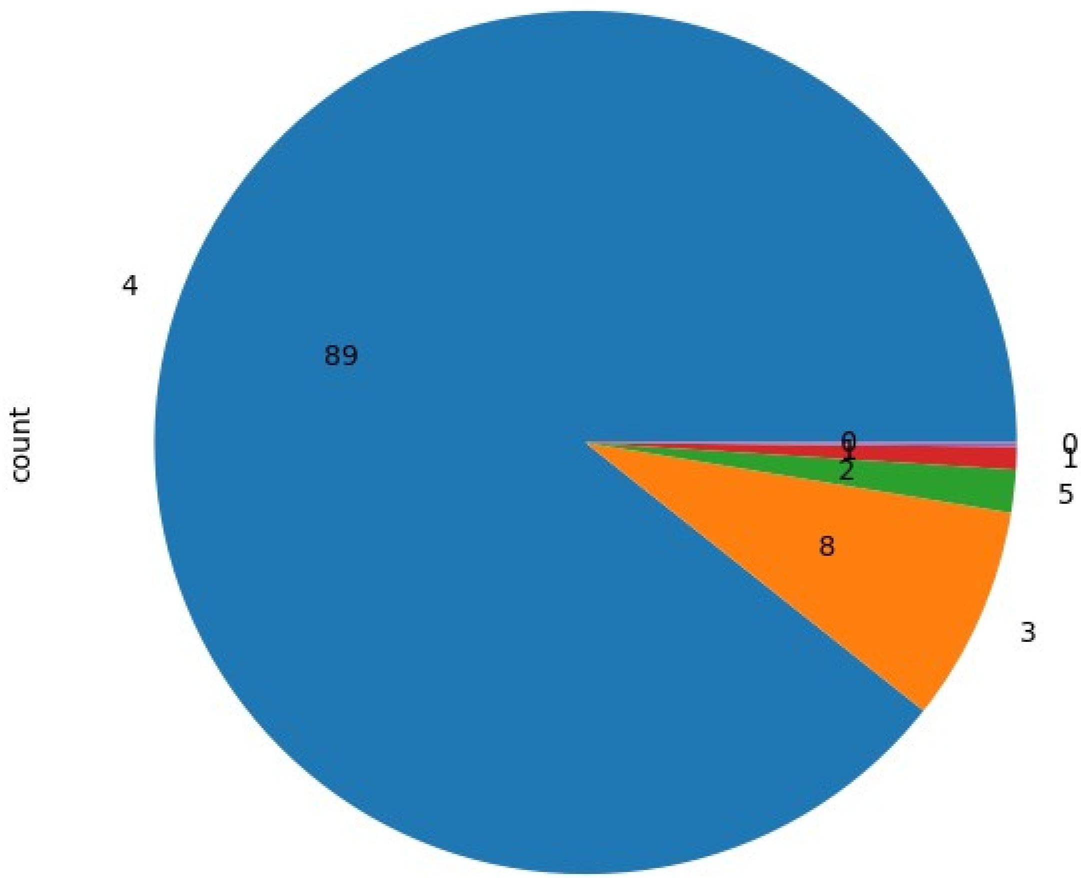
```
<BarContainer object of 5 artists>
```






```
df['time_signature'].value_counts().plot(kind = 'pie', autopct = "%2.f", figsize = (7,7))
```

```
<Axes: ylabel='count'>
```



A Hunch about Time Signatures and Subsequent Testing

From the above results we can see that "4" vastly outnumbers the other time signatures. I feel it would make sense to turn this feature into a binary feature of "4 vs Not-4".

At this point, I have a hunch that songs with **time signature 4** may be **more popular** than songs with other time signatures or vice-versa. We will test this hunch now.

```
binaryTimeSig = df['time_signature'].apply(lambda x: 1 if x == 4 else 0)

df2 = pd.DataFrame({'Popularity':df['popularity'],'binaryTimeSig':binaryTimeSig})

df2.corr()
```

	Popularity	binaryTimeSig
Popularity	1.000000	0.056315
binaryTimeSig	0.056315	1.000000

A correlation of **0.056315** is considered **very weak.** *Therefore my hunch was* **incorrect**.

We can conclude that in this datset, there is little to no correlation between whether a song is in 44 time signature or not.

```
df2[df2['binaryTimeSig']==1]['Popularity'].describe()
```

	Popularity
count	101842.000000
mean	33.672807
std	22.378415
min	0.000000
25%	17.000000
50%	35.000000
75%	50.000000
max	100.000000

dtype: float64

TEMPO

Analysing Tempo Column

```
df['tempo'].describe()
```

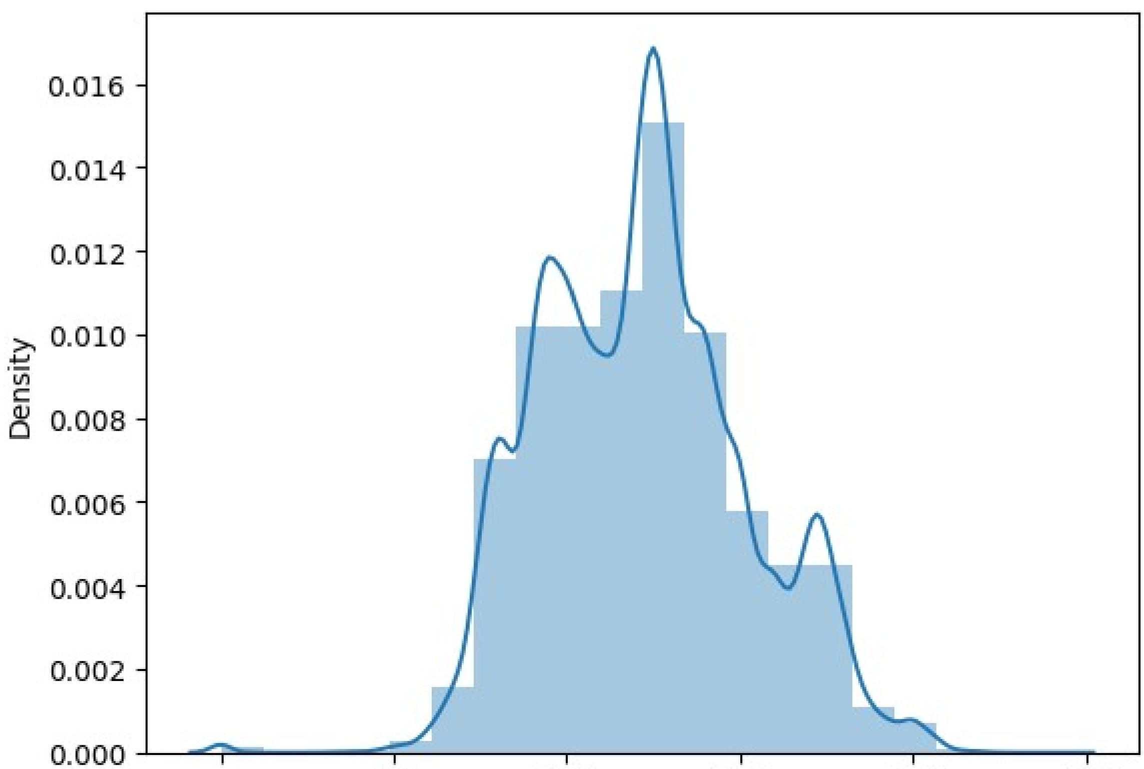
	tempo

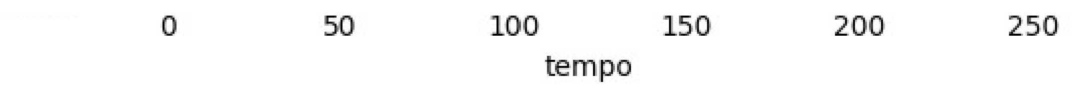
count	113999.000000
mean	122.147695
std	29.978290
min	0.000000
25%	99.218500
50%	122.017000
75%	140.071000
max	243.372000

dtype: float64

```
sns.distplot(df['tempo'],bins = 20)
```

```
<ipython-input-33-06e52430cf90>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['tempo'],bins = 20)
<Axes: xlabel='tempo', ylabel='Density'>
```





Popularity

Analysing the Popularity Column

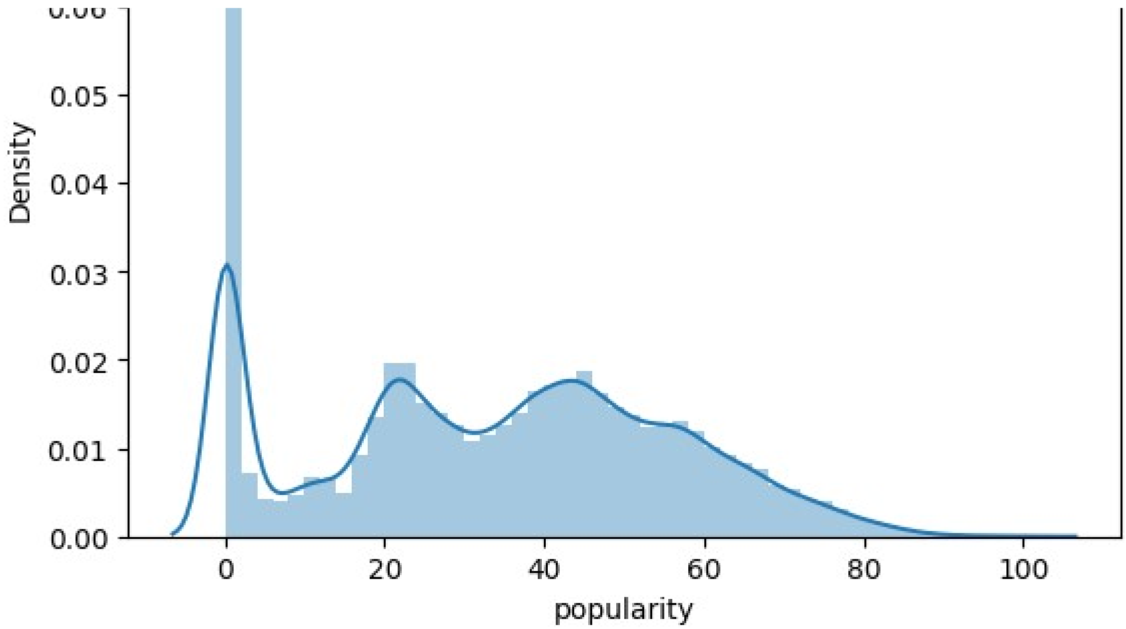
```
df['popularity'].describe()
```

	popularity
count	113999.000000
mean	33.238827
std	22.304959
min	0.000000
25%	17.000000
50%	35.000000
75%	50.000000
max	100.000000

dtype: float64

```
sns.distplot(df['popularity'])
```

```
<ipython-input-35-93de2ca8ddf5>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['popularity'])
<Axes: xlabel='popularity', ylabel='Density'>
```



Just for Reference

```
df.columns
```

```
Index(['track_id', 'artists', 'album_name', 'track_name', 'popularity',
       'explicit', 'danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'time_signature', 'track_genre', 'duration_m'],
      dtype='object')
```

```
df['track_genre'].value_counts()
```

	count
track_genre	
acoustic	1000
afrobeat	1000
alt-rock	1000
alternative	1000
ambient	1000
...	...
techno	1000

turkish	1000
trip-hop	1000
world-music	1000
k-pop	999

114 rows × 1 columns

dtype: int64

Groupby Genre

```
genreWise = pd.DataFrame(df.groupby("track_genre"))
```

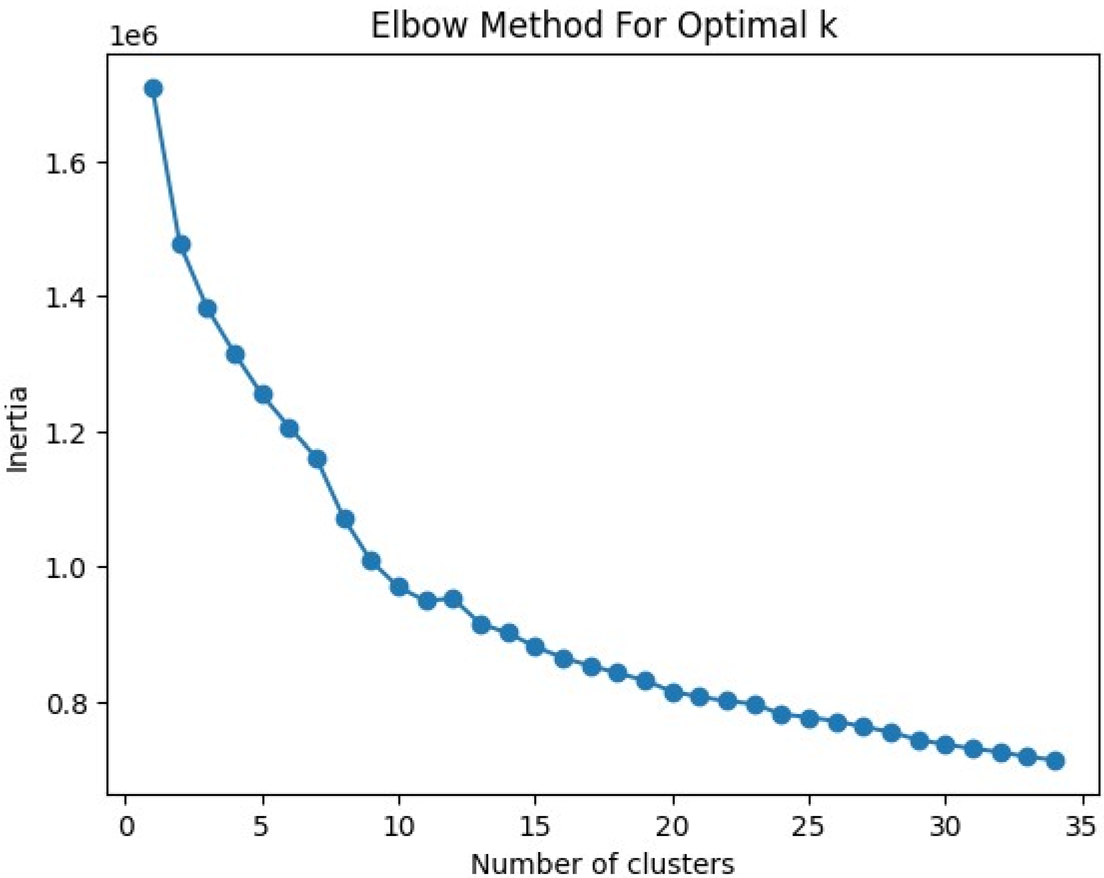
Attempting Clustering of Using Kmeans

This dataset has 114 genres in it, most of which are very granular and specific. We will attempt to group together songs with similar features into clusters using kmeans algorithm.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```
features = df.drop(['track_genre','track_id', 'artists', 'album_name', 'track_name'], axi

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

```
inertia = []
K = range(1, 35)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=45)
    kmeans.fit(features_scaled)
    inertia.append(kmeans.inertia_)

plt.plot(K, inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```




```
numClust = 7

kmeans = KMeans(n_clusters=numClust, random_state=42, init='k-means++')
df['cluster'] = kmeans.fit_predict(features_scaled)

genre_cluster_ct = pd.crosstab(df['cluster'], df['track_genre'])
print(genre_cluster_ct)
```

```
track_genre  acoustic  afrobeat  alt-rock  alternative  ambient  anime  \
cluster
0                  10        58        63           31       10     33
1                  32         6         1            1      651    157
2                  60       314       286          299       11    261
3                 608        89        89           97      181     56
4                 234       282       463          382       40    311
5                  50        19        56          164        4     55
6                   6       232        42           26      103    127

track_genre  black-metal  bluegrass  blues  brazil  ...  spanish  study  \
cluster                                             ...
0                     57         89     37     201  ...       94     22
1                     18         34      8       2  ...        1    254
2                    162         75    200     178  ...      264     40
3                     11        422    341     205  ...      145    174
4                    154        304    385     321  ...      445     22
5                    118          5     14      86  ...       37      0
```

```
5                     118          5     14      86  ...       37     0
6                     480         71     15       7  ...       14   488

track_genre  swedish  synth-pop  tango  techno  trance  trip-hop  turkish  \
cluster
0                 34         38     73      22      59        47       30
1                  6          1     54       4       0        18        8
2                194        308     61     160     228       214      394
3                258        100    791      22       2       115      184
4                401        456      9     150     192       169      145
5                 95         44      0      25      24        45      218
6                 12         53     12     617     495       392       21

track_genre  world-music
cluster
0                    152
1                    104
2                     56
3                    269
4                    418
5                      0
6                      1

[7 rows x 114 columns]
```

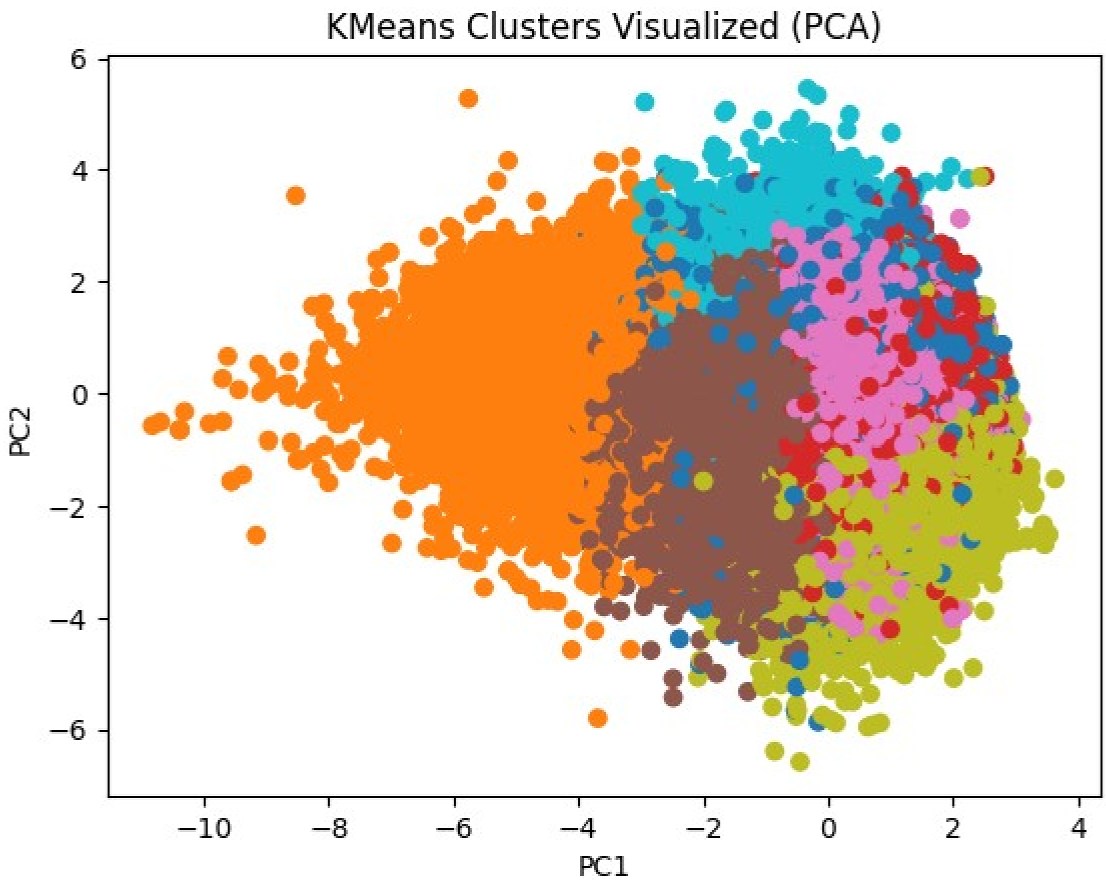
```
'''
from sklearn.cluster import DBSCAN

# Scale your data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)

# Run DBSCAN
dbscan = DBSCAN(eps=3, min_samples=10)
df['cluster'] = dbscan.fit_predict(X_scaled)
'''
```

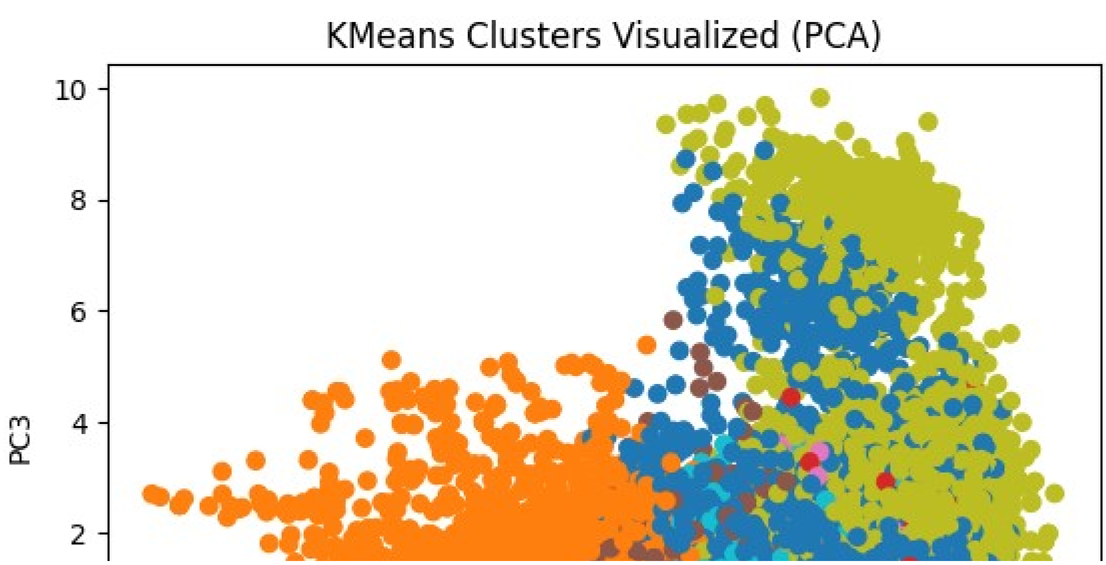
```
'\nfrom sklearn.cluster import DBSCAN\n\n# Scale your data\nscaler = StandardScaler
()\nX_scaled = scaler.fit_transform(features)\n\n# Run DBSCAN\ndbscan = DBSCAN(eps=
3, min_samples=10)\ndf['cluster'] = dbscan.fit_predict(X_scaled)\n'
```

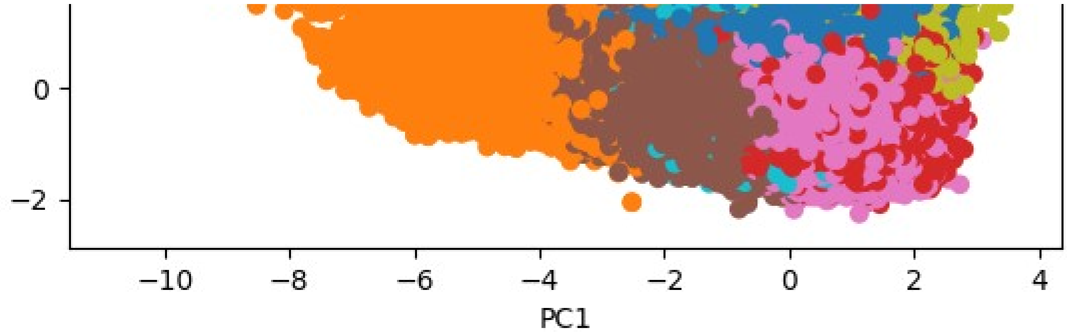
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
components = pca.fit_transform(features_scaled)
```

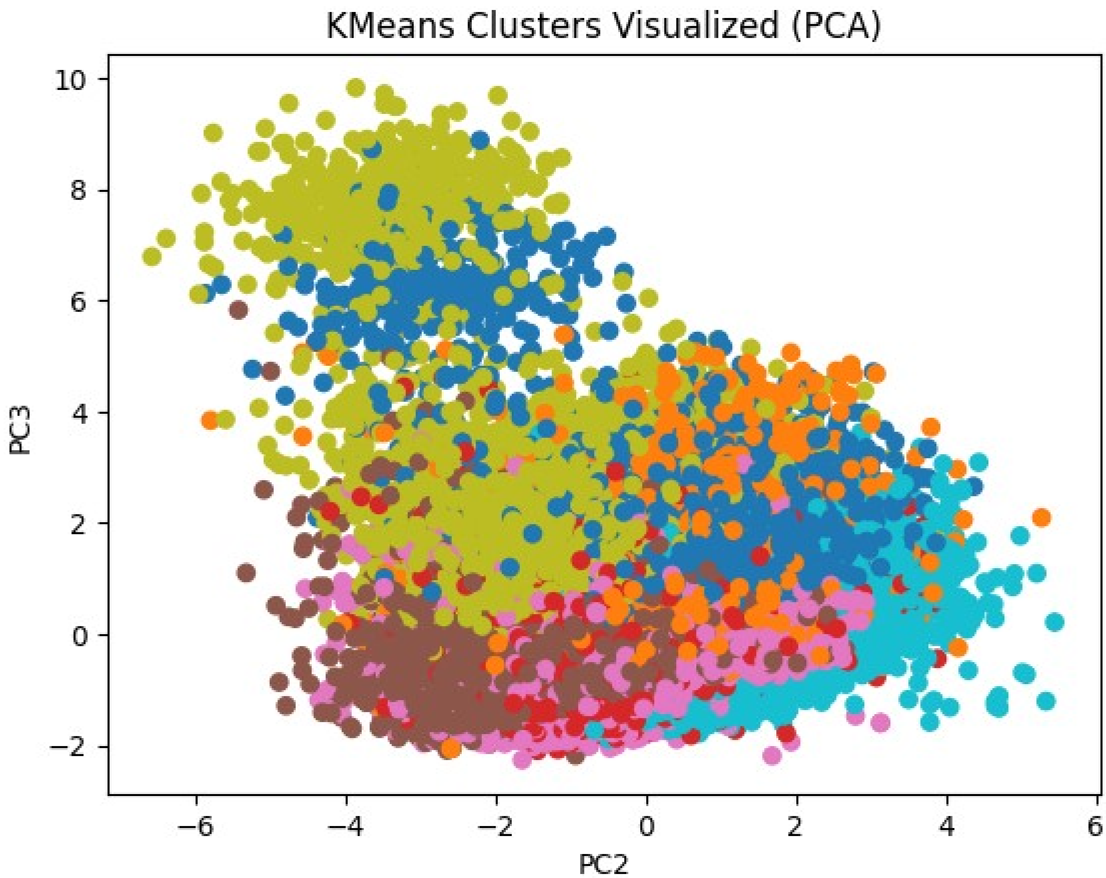
```
plt.scatter(components[:, 0], components[:, 1], c=df['cluster'], cmap='tab10')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('KMeans Clusters Visualized (PCA)')
plt.show()
```

```
plt.scatter(components[:, 0], components[:, 2], c=df['cluster'], cmap='tab10')
plt.xlabel('PC1')
plt.ylabel('PC3')
plt.title('KMeans Clusters Visualized (PCA)')
plt.show()

plt.scatter(components[:, 1], components[:, 2], c=df['cluster'], cmap='tab10')
plt.xlabel('PC2')
plt.ylabel('PC3')
plt.title('KMeans Clusters Visualized (PCA)')
plt.show()
```



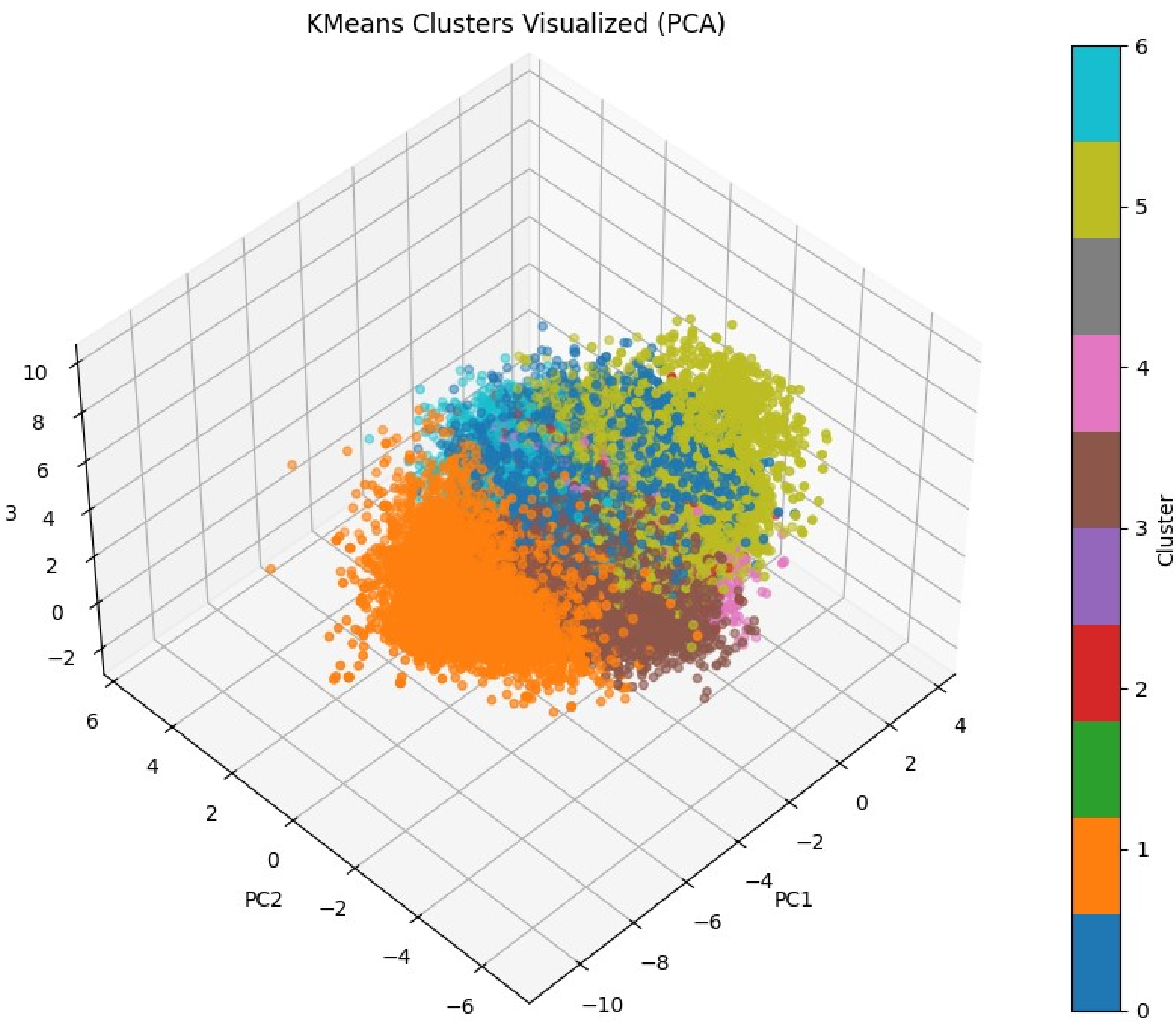






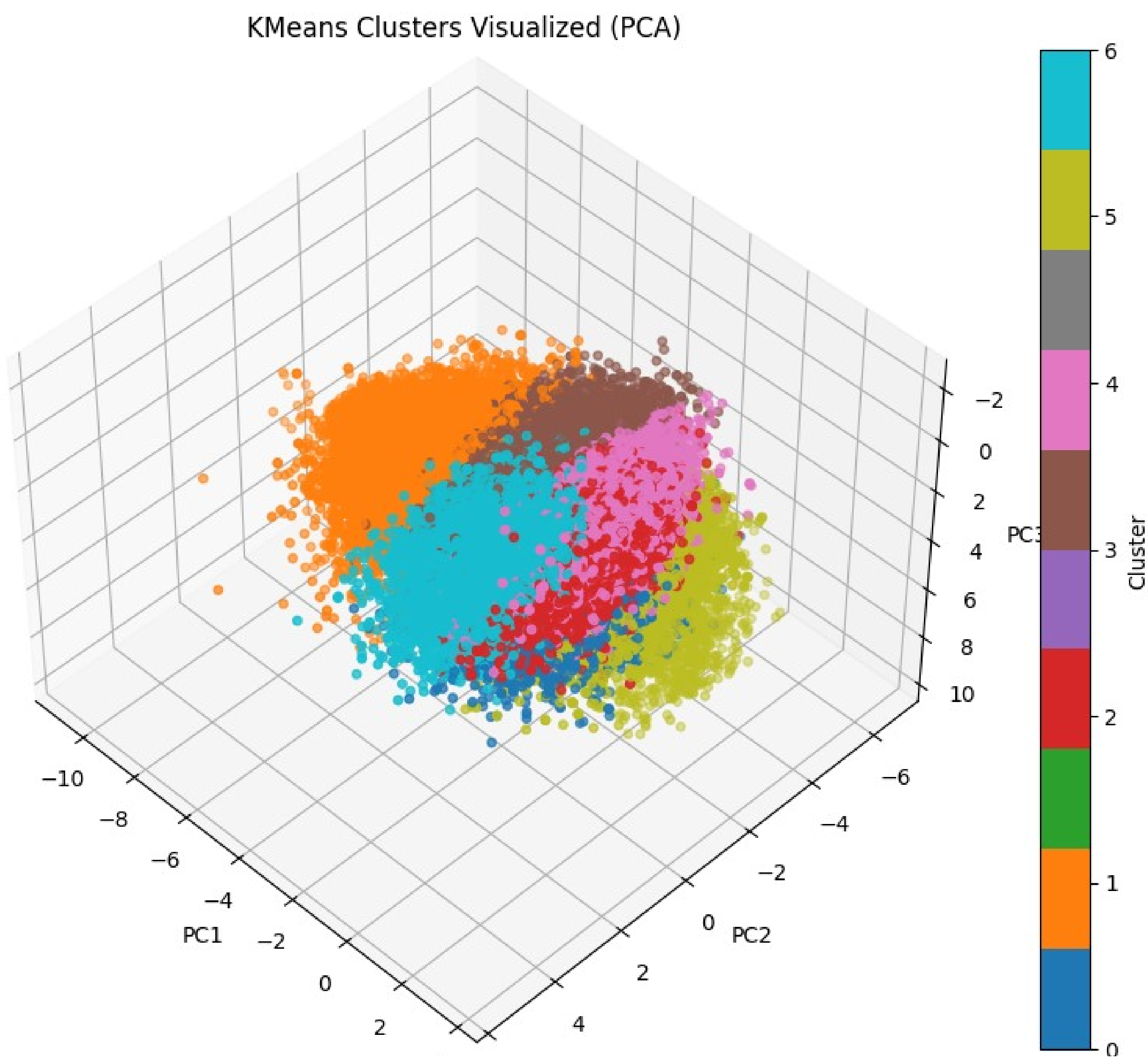

```
# Create 3D scatter plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(
    components[:, 0], components[:, 1], components[:, 2],
    c=df['cluster'], cmap='tab10', s=15
)

# Labeling axes and title
ax.set_title("KMeans Clusters Visualized (PCA)", fontsize=12)
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
```

```
ax.view_init(45,225)

# Add colorbar
fig.colorbar(scatter, ax=ax, label='Cluster')


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




```
fig = plt.figure(figsize=(10, 7))
```

```
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(
    components[:, 0], components[:, 1], components[:, 2],
    c=df['cluster'], cmap='tab10', s=15
)

ax.set_title("KMeans Clusters Visualized (PCA)", fontsize=12)
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")

ax.view_init(225, 225)

# Add colorbar
fig.colorbar(scatter, ax=ax, label='Cluster')


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




```
print(pca.explained_variance_)
```

```
[3.00256424 1.5671262  1.38153061]
```

```
print(pca.explained_variance_ratio_)
```

```
[0.20016919 0.10447416 0.09210123]
```

One can see a fair amount of patterns forming after performing clustering. It is not perfect, we do have some overlap in the clusters, but we must bear in mind that we have only considered 3 Principal Compnenents.

As we can see, the first 3 principal compnents account for ~20%, ~10%, and ~9.2% respectively, which is not a lot, so it is remarkable that we are still able to see patterns form within the clusters

CLUSTERING OBSERVATIONS

> Add blockquote

```
for i in range(numClust):
    print(f"Cluster {i+1}")
    print(df[df['cluster'] == i]['track_genre'].value_counts().head(35))
    print()
```

```
Cluster 1
track_genre
pagode          487
sertanejo       415
samba           409
comedy          276
mpb             271
gospel          222
brazil          201
forro           176
world-music     152
```

```
r-n-b            148
ska              143
heavy-metal      141
sleep            119
happy            109
grindcore        104
groove            99
party             95
spanish           94
hardstyle         93
bluegrass         89
punk-rock         84
hard-rock         83
goth              81
grunge            77
tango             73
psych-rock        71
show-tunes        70
opera             70
rock-n-roll       67
metalcore         64
alt-rock          63
power-pop         62
drum-and-bass     62
punk              61
trance            59
Name: count, dtype: int64

Cluster 2
track_genre
sleep                792
new-age              772
classical            761
ambient              651
piano                560
guitar               448
disney               440
iranian              437
opera                351
german               299
idm                  272
study                254
romance              181
anime                157
british              148
jazz                 117
world-music          104
```

```
**POSITIVES**

* This clustering certainly isn't perfect, but there are several eye-catching
groupings
```

groupings.

* Cluster 1 has a strong Brazilian theme — Pagode, Sertanejo, Samba, MPB, Forró, and Brazil all show up. RNB and Spanish are also present, which fits, considering musical influences.

* However heavy Metal and Sleep are also present in CLuster 1, which is rather odd and amusing.

* Cluster 2 is full of Ambient, Sleep, New-Age, Piano, Opera, and Study. It also includes Anime, Disney, and Jazz, which makes this one of the most coherent clusters.

* Cluster 3 contains House, EDM, Dance, Hip-Hop, Reggaeton, K-pop, Salsa, and Dubstep. A very energetic mix — lots of rhythm-heavy genres.

* Cluster 4 includes Jazz, Folk, Acoustic, Country, Singer-Songwriter, and Bluegrass. Pretty strong grouping of organic and melodic genres that have shared history, which is remarkable (although "Jazz" is a massive genre and has several sub genres which may or may not be similar to the others in this group).

* Cluster 5 features Punk, Rock, Grunge, J-Rock, J-Pop, Kids, Party, Power-Pop, and Ska. Feels chaotic, but all very youth-focused and upbeat.

* Cluster 6 is kind of aggressive and emotional — Comedy, Emo, Sad, Hardcore, Funk, Grindcore, and Metalcore all show up. A rather peculiar group, perhaps the most inconsistent cluster.

* Cluster 7 is packed with electronic genres — Minimal Techno, Detroit Techno, Techno, Trance, IDM, Trip-Hop, Drum-and-Bass. Also includes Black Metal, Death Metal, and Industrial, which is somewhat odd, but does makes some sense sonically (unlike sleep and heavy metal being in grouped together in cluster 1).

NEGATIVES

* 0.15 silhouette score, which indicates very poor cluster spearation.

FINAL CONCLUSIONS

Despite a relatively low silhouette score of 0.15, the clustering results reveal meaningful structure in the audio feature space. Several clusters group together genres with shared musical characteristics — such as:

    Techno subgenres and house styles

    Ambient, study, and sleep tracks

    Folk, jazz, and acoustic genres

    Overlaps between Latin, Spanish, and RnB

```
This indicates that even without explicit genre labels, a simple algorithm like
KMeans was able to capture some degree of stylistic similarities in the data.
However, the presence of genres like "Kids" or "Sleep" in multiple, unrelated
clusters highlights the limitations of using purely audio-based features for
genre classification.

Ultimately, the 0.15 silhouette score reflects the complexity and fuzziness of
real-world music genre boundaries. While the clusters aren't cleanly separable,
the model does uncover some valuable patterns — suggesting that unsupervised
learning, when combined with domain insight, can still yield practically useful
groupings.
```

POSITIVES

- This clustering certainly isn't perfect, but there are several eye-catching groupings.
- Cluster 1 has a strong Brazilian theme — Pagode, Sertanejo, Samba, MPB, Forró, and Brazil all show up. RNB and Spanish are also present, which fits, considering musical influences.
- However heavy Metal and Sleep are also present in CLuster 1, which is rather odd and amusing.
- Cluster 2 is full of Ambient, Sleep, New-Age, Piano, Opera, and Study. It also includes Anime, Disney, and Jazz, which makes this one of the most coherent clusters.
- Cluster 3 contains House, EDM, Dance, Hip-Hop, Reggaeton, K-pop, Salsa, and Dubstep. A very energetic mix — lots of rhythm-heavy genres.
- Cluster 4 includes Jazz, Folk, Acoustic, Country, Singer-Songwriter, and Bluegrass. Pretty strong grouping of organic and melodic genres that have shared history, which is remarkable (although "Jazz" is a massive genre and has several sub genres which may or may not be similar to the others in this group).
- Cluster 5 features Punk, Rock, Grunge, J-Rock, J-Pop, Kids, Party, Power-Pop, and Ska. Feels chaotic, but all very youth-focused and upbeat.
- Cluster 6 is kind of aggressive and emotional — Comedy, Emo, Sad, Hardcore, Funk, Grindcore, and Metalcore all show up. A rather peculiar group, perhaps the most inconsistent cluster.
- Cluster 7 is packed with electronic genres — Minimal Techno, Detroit Techno, Techno, Trance, IDM, Trip-Hop, Drum-and-Bass. Also includes Black Metal, Death Metal, and

industrial, which is somewhat odd, but does makes some sense sonically (unlike sleep and heavy metal being in grouped together in cluster 1).

NEGATIVES

- 0.15 silhouette score, which indicates very poor cluster spearation.

FINAL CONCLUSIONS

Despite a relatively low silhouette score of 0.15, the clustering results reveal meaningful structure in the audio feature space. Several clusters group together genres with shared musical characteristics — such as:

```
Techno subgenres and house styles

Ambient, study, and sleep tracks

Folk, jazz, and acoustic genres

Overlaps between Latin, Spanish, and RnB
```

This indicates that even without explicit genre labels, a simple algorithm like KMeans was able to capture some degree of stylistic similarities in the data. However, the presence of genres like "Kids" or "Sleep" in multiple, unrelated clusters highlights the limitations of using purely audio-based features for genre classification.

Ultimately, the 0.15 silhouette score reflects the complexity and fuzziness of real-world music genre boundaries. While the clusters aren't cleanly separable, the model does uncover some valuable patterns — suggesting that unsupervised learning, when combined with domain insight, can still yield practically useful groupings.

```
Start coding or generate with AI.
```