Project_Creating_Cohorts_of_Songs

December 19, 2024

0.1 Creating Cohorts of Songs.

0.1.1 Description

The customer always looks forward to specialized treatment, whether shopping on an e-commerce website or watching Netflix. The customer desires content that aligns with their preferences. To maintain customer engagement, companies must consistently provide the most relevant information.

Starting with Spotify, a Swedish audio streaming and media service provider, boasts over 456 million active monthly users, including more than 195 million paid subscribers as of September 2022. The company aims to create cohorts of different songs to enhance song recommendations. These cohorts will be based on various relevant features, ensuring that each group contains similar types of songs.

0.1.2 Problem Objective:

As a data scientist, you should perform exploratory data analysis and cluster analysis to create cohorts of songs. The goal is to better understand the various factors that create a cohort of songs.

[42]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[43]:
```
df1 = pd.read_csv('rolling_stones_spotify.csv')
df2 = pd.read_excel('data_dictionary_creating_cohortsofsongs.xlsx')
```

0.2 Step1 : Initial data inspection and data cleaning

[44]:
```
df1.head()
```

[44]:
```
   Unnamed: 0                          name               album release_date  \
0           0    Concert Intro Music - Live  Licked Live In NYC   2022-06-10
1           1    Street Fighting Man - Live  Licked Live In NYC   2022-06-10
2           2             Start Me Up - Live  Licked Live In NYC   2022-06-10
3           3  If You Can't Rock Me - Live  Licked Live In NYC   2022-06-10
4           4               Don’t Stop - Live  Licked Live In NYC   2022-06-10
```

```
   track_number                      id                                   uri  \
0             1  2IEkywLJ4ykbhi1yRQvmsT  spotify:track:2IEkywLJ4ykbhi1yRQvmsT
1             2  6GVgVJBKkGJoRfarYRvGTU  spotify:track:6GVgVJBKkGJoRfarYRvGTU
2             3  1Lu761pZ0dBTGpzxaQoZNW  spotify:track:1Lu761pZ0dBTGpzxaQoZNW
3             4  1agTQzOTUnGNggyckEqiDH  spotify:track:1agTQzOTUnGNggyckEqiDH
4             5  7piGJR8YndQBQWVXv6KtQw  spotify:track:7piGJR8YndQBQWVXv6KtQw

   acousticness  danceability  energy  instrumentalness  liveness  loudness  \
0        0.0824         0.463   0.993          0.996000     0.932   -12.913
1        0.4370         0.326   0.965          0.233000     0.961    -4.803
2        0.4160         0.386   0.969          0.400000     0.956    -4.936
3        0.5670         0.369   0.985          0.000107     0.895    -5.535
4        0.4000         0.303   0.969          0.055900     0.966    -5.098

   speechiness    tempo  valence  popularity  duration_ms
0       0.1100  118.001   0.0302          33        48640
1       0.0759  131.455   0.3180          34       253173
2       0.1150  130.066   0.3130          34       263160
3       0.1930  132.994   0.1470          32       305880
4       0.0930  130.533   0.2060          32       305106
```

[45]:
```
df2.head()
```

[45]:
```
       Variable                                  Description
0          name                            the name of the song
1         album                           the name of the album
2  release_date  the day month and year the album was released
3  track number          the order the song appears on the album
4            id                        the Spotify id for the song
```

[46]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1610 entries, 0 to 1609
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        1610 non-null   int64
 1   name              1610 non-null   object
 2   album             1610 non-null   object
 3   release_date      1610 non-null   object
 4   track_number      1610 non-null   int64
 5   id                1610 non-null   object
 6   uri               1610 non-null   object
 7   acousticness      1610 non-null   float64
 8   danceability      1610 non-null   float64
 9   energy            1610 non-null   float64
 10  instrumentalness  1610 non-null   float64
```

```
 11   liveness          1610 non-null   float64
 12   loudness          1610 non-null   float64
 13   speechiness       1610 non-null   float64
 14   tempo             1610 non-null   float64
 15   valence           1610 non-null   float64
 16   popularity        1610 non-null   int64
 17   duration_ms       1610 non-null   int64
dtypes: float64(9), int64(4), object(5)
memory usage: 226.5+ KB
```

[47]:
```
df2.head()
```

[47]:
```
       Variable                                 Description
0          name                            the name of the song
1         album                           the name of the album
2  release_date  the day month and year the album was released
3  track number          the order the song appears on the album
4            id                       the Spotify id for the song
```

[48]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Variable     17 non-null     object
 1   Description  17 non-null     object
dtypes: object(2)
memory usage: 404.0+ bytes
```

[49]:
```
df1.describe()
```

[49]:
```
        Unnamed: 0  track_number  acousticness  danceability       energy  \
count  1610.000000   1610.000000   1610.000000   1610.000000  1610.000000
mean    804.500000      8.613665      0.250475      0.468860     0.792352
std     464.911282      6.560220      0.227397      0.141775     0.179886
min       0.000000      1.000000      0.000009      0.104000     0.141000
25%     402.250000      4.000000      0.058350      0.362250     0.674000
50%     804.500000      7.000000      0.183000      0.458000     0.848500
75%    1206.750000     11.000000      0.403750      0.578000     0.945000
max    1609.000000     47.000000      0.994000      0.887000     0.999000

       instrumentalness    liveness     loudness  speechiness        tempo  \
count       1610.000000  1610.00000  1610.000000  1610.000000  1610.000000
mean           0.164170     0.49173    -6.971615     0.069512   126.082033
std            0.276249     0.34910     2.994003     0.051631    29.233483
min            0.000000     0.02190   -24.408000     0.023200    46.525000
```

```
25%          0.000219     0.15300    -8.982500     0.036500    107.390750
50%          0.013750     0.37950    -6.523000     0.051200    124.404500
75%          0.179000     0.89375    -4.608750     0.086600    142.355750
max          0.996000     0.99800    -1.014000     0.624000    216.304000

           valence   popularity    duration_ms
count  1610.000000  1610.000000    1610.000000
mean      0.582165    20.788199  257736.488199
std       0.231253    12.426859  108333.474920
min       0.000000     0.000000   21000.000000
25%       0.404250    13.000000  190613.000000
50%       0.583000    20.000000  243093.000000
75%       0.778000    27.000000  295319.750000
max       0.974000    80.000000  981866.000000
```

[50]:
```python
df1.isnull().sum()/len(df1)*100
```

[50]:
```
Unnamed: 0          0.0
name                0.0
album               0.0
release_date        0.0
track_number        0.0
id                  0.0
uri                 0.0
acousticness        0.0
danceability        0.0
energy              0.0
instrumentalness    0.0
liveness            0.0
loudness            0.0
speechiness         0.0
tempo               0.0
valence             0.0
popularity          0.0
duration_ms         0.0
dtype: float64
```

[51]:
```python
df1.duplicated().sum()
```

[51]:
```
np.int64(0)
```

[52]:
```python
# Droping Unnamed column
df=df1.drop('Unnamed: 0',axis = 1)
```

[53]:
```python
df.head()
```

[53]:
```
                          name                album release_date  track_number  \
0   Concert Intro Music - Live  Licked Live In NYC   2022-06-10             1
1   Street Fighting Man - Live  Licked Live In NYC   2022-06-10             2
2           Start Me Up - Live  Licked Live In NYC   2022-06-10             3
3  If You Can't Rock Me - Live  Licked Live In NYC   2022-06-10             4
4            Don't Stop - Live  Licked Live In NYC   2022-06-10             5

                       id                                   uri  acousticness  \
0  2IEkywLJ4ykbhi1yRQvmsT  spotify:track:2IEkywLJ4ykbhi1yRQvmsT        0.0824
1  6GVgVJBKkGJoRfarYRvGTU  spotify:track:6GVgVJBKkGJoRfarYRvGTU        0.4370
2  1Lu761pZ0dBTGpzxaQoZNW  spotify:track:1Lu761pZ0dBTGpzxaQoZNW        0.4160
3  1agTQzOTUnGNggyckEqiDH  spotify:track:1agTQzOTUnGNggyckEqiDH        0.5670
4  7piGJR8YndQBQWVXv6KtQw  spotify:track:7piGJR8YndQBQWVXv6KtQw        0.4000

   danceability  energy  instrumentalness  liveness  loudness  speechiness  \
0         0.463   0.993          0.996000     0.932   -12.913       0.1100
1         0.326   0.965          0.233000     0.961    -4.803       0.0759
2         0.386   0.969          0.400000     0.956    -4.936       0.1150
3         0.369   0.985          0.000107     0.895    -5.535       0.1930
4         0.303   0.969          0.055900     0.966    -5.098       0.0930

     tempo  valence  popularity  duration_ms
0  118.001   0.0302          33        48640
1  131.455   0.3180          34       253173
2  130.066   0.3130          34       263160
3  132.994   0.1470          32       305880
4  130.533   0.2060          32       305106
```

[54]:
```python
df.shape
```

[54]: (1610, 17)

Data Structure:

The dataset contains 1,610 entries with 18 columns, including details about songs such as their name, album, release date, and various audio features like acousticness, danceability,energy, etc. Missing Values:

There are no missing values in the dataset, so no imputation or removal of rows is necessary.

There are no duplicated rows in the dataset. Data Types:

The columns have appropriate data types, with numeric columns like acousticness, danceability, energy, etc., being of float type and others like name, album, and release_date being object types.

0.3 Step 2. Refine the data¶

[55]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from scipy import stats
```

[56]:
```
# converting release_date in to date time
df['release_date']= pd.to_datetime(df['release_date'])

# extracting year from the release_date
df['year'] = df['release_date'].dt.year

# extracting month from the release_date
df['month'] = df['release_date'].dt.month

# extracting day from the release_date
df['day'] = df['release_date'].dt.day
```

[57]:
```
# Normalize numerical features
numeric_col = ['acousticness', 'danceability', 'energy', 'instrumentalness',␣
↪'liveness', 'loudness', 'speechiness', 'tempo', 'valence']
scaler = StandardScaler()
df[numeric_col] = scaler.fit_transform(df[numeric_col])
```

[58]:
```
df[numeric_col]
```

[58]:
```
      acousticness  danceability    energy  instrumentalness  liveness  \
0        -0.739355     -0.041343  1.115764          3.012099  1.261552
1         0.820518     -1.007963  0.960062          0.249238  1.344648
2         0.728140     -0.584626  0.982305          0.853953  1.330321
3         1.392383     -0.704571  1.071278         -0.594080  1.155532
4         0.657756     -1.170242  0.982305         -0.392050  1.358975
…              …             …         …                …         …
1605     -0.411192     -0.020176  0.776555         -0.572125 -0.480613
1606     -0.848449      0.283215 -0.480188         -0.594461  0.069544
1607      0.530186      2.265844 -0.102053         -0.594467 -1.217308
1608     -0.147254      1.630838 -1.369917         -0.594213 -0.933346
1609      0.582974      1.821340  0.787676         -0.346425 -1.132492

      loudness  speechiness     tempo   valence
0    -1.985045     0.784410 -0.276517 -2.387590
1     0.724545     0.123753  0.183852 -1.142678
2     0.680109     0.881280  0.136323 -1.164306
3     0.479980     2.392459  0.236514 -1.882359
4     0.625984     0.455050  0.152303 -1.627147
…          …            …         …         …
```

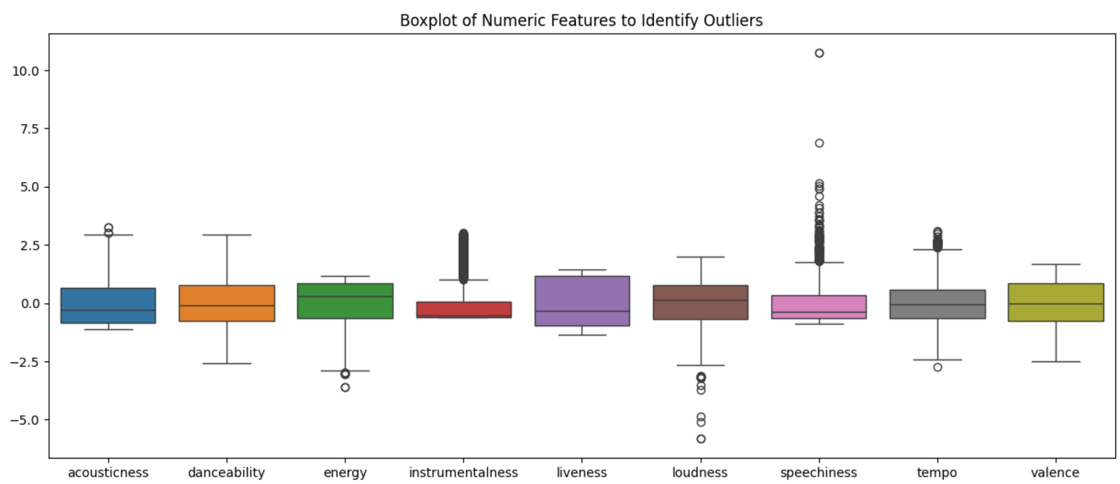
```
1605 -0.749192    -0.515591  1.753944  1.664646
1606 -0.820356     0.286496 -0.139166 -0.588999
1607 -0.330558     0.048195 -0.993931  1.093665
1608 -0.867131    -0.141672 -0.802344 -0.216997
1609 -0.468210    -0.651210 -0.027615  1.673297

[1610 rows x 9 columns]
```

[59]:

```
# Handle missing values
df = df.dropna()
```

[60]:

```
# Plotting boxplots for numeric columns to identify outliers
plt.figure(figsize=(15, 6))
sns.boxplot(df[numeric_col])
plt.title('Boxplot of Numeric Features to Identify Outliers')

plt.show()
```



[61]:

```
# Remove outliers based on Z-score
df = df[(np.abs(stats.zscore(df[numeric_col])) < 3).all(axis=1)]

# Create a feature for the decade of release
df['release_decade'] = (df['year'] // 10) * 10

# Ensure consistent capitalization for album names
df['album'] = df['album'].str.title()
```

[62]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1561 entries, 1 to 1609
```

```
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              1561 non-null   object
 1   album             1561 non-null   object
 2   release_date      1561 non-null   datetime64[ns]
 3   track_number      1561 non-null   int64
 4   id                1561 non-null   object
 5   uri               1561 non-null   object
 6   acousticness      1561 non-null   float64
 7   danceability      1561 non-null   float64
 8   energy            1561 non-null   float64
 9   instrumentalness  1561 non-null   float64
 10  liveness          1561 non-null   float64
 11  loudness          1561 non-null   float64
 12  speechiness       1561 non-null   float64
 13  tempo             1561 non-null   float64
 14  valence           1561 non-null   float64
 15  popularity        1561 non-null   int64
 16  duration_ms       1561 non-null   int64
 17  year              1561 non-null   int32
 18  month             1561 non-null   int32
 19  day               1561 non-null   int32
 20  release_decade    1561 non-null   int32
dtypes: datetime64[ns](1), float64(9), int32(4), int64(3), object(4)
memory usage: 243.9+ KB
```

0.4 Step 3. Perform exploratory data analysis and feature engineering

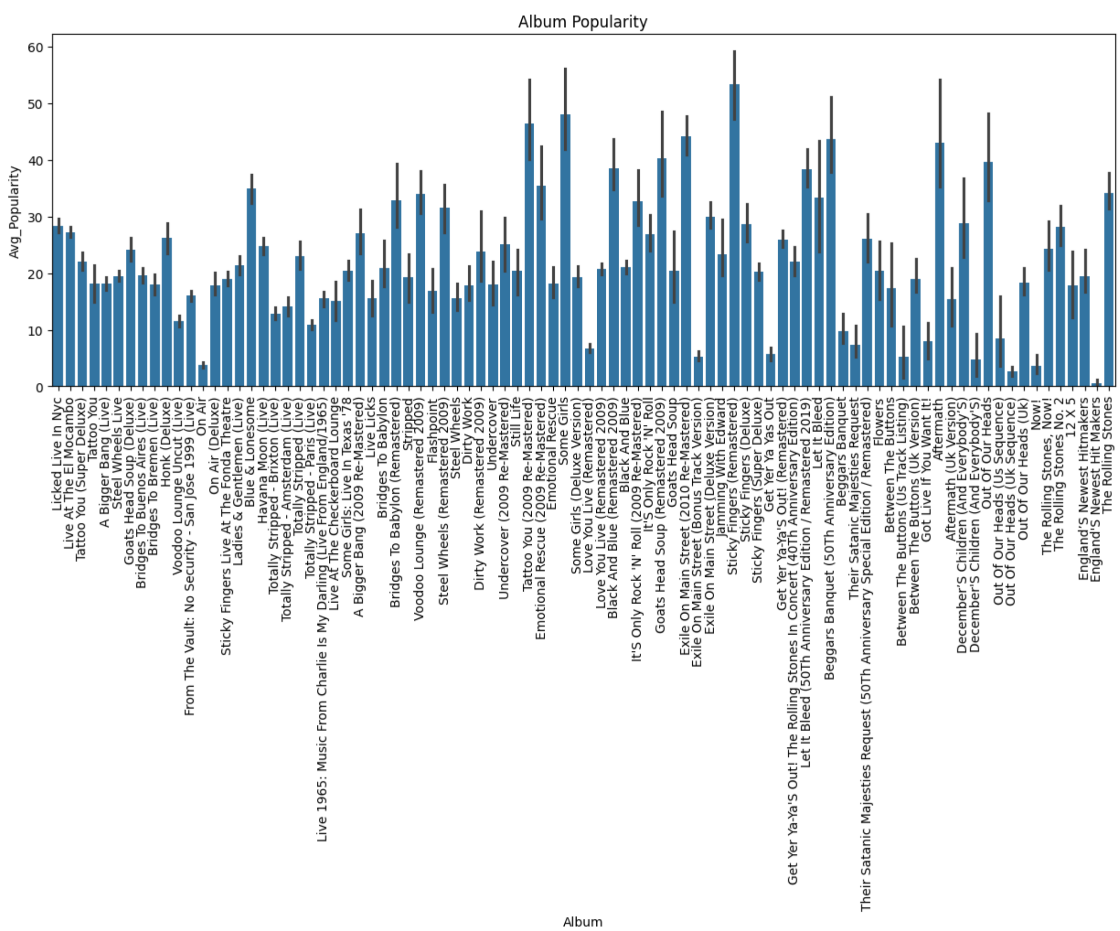
[63]:
```python
df.head(2)
```

[63]:
```
                          name                album release_date  track_number  \
1  Street Fighting Man - Live  Licked Live In Nyc   2022-06-10             2
2          Start Me Up - Live  Licked Live In Nyc   2022-06-10             3

                       id                                   uri  acousticness  \
1  6GVgVJBKkGJoRfarYRvGTU  spotify:track:6GVgVJBKkGJoRfarYRvGTU      0.820518
2  1Lu761pZ0dBTGpzxaQoZNW  spotify:track:1Lu761pZ0dBTGpzxaQoZNW      0.728140

   danceability    energy  instrumentalness  …  loudness  speechiness  \
1     -1.007963  0.960062          0.249238  …  0.724545     0.123753
2     -0.584626  0.982305          0.853953  …  0.680109     0.881280

      tempo   valence  popularity  duration_ms  year  month  day  \
1  0.183852 -1.142678          34       253173  2022      6   10
2  0.136323 -1.164306          34       263160  2022      6   10
```

```
   release_decade
1            2020
2            2020

[2 rows x 21 columns]
```

[ ]:
```
Utilize suitable visualizations to identify the two albums that should be␣
↪recommended to anyone based on the number of popular songs in each album
```

[64]:
```python
df.groupby('album')['popularity'].mean().sort_values(ascending=False)
plt.figure(figsize = (15,5))
sns.barplot(x = 'album',y='popularity',data = df)
plt.xlabel('Album')
plt.ylabel('Avg_Popularity')
plt.title('Album Popularity')
plt.xticks(rotation = 90)
plt.show()
```



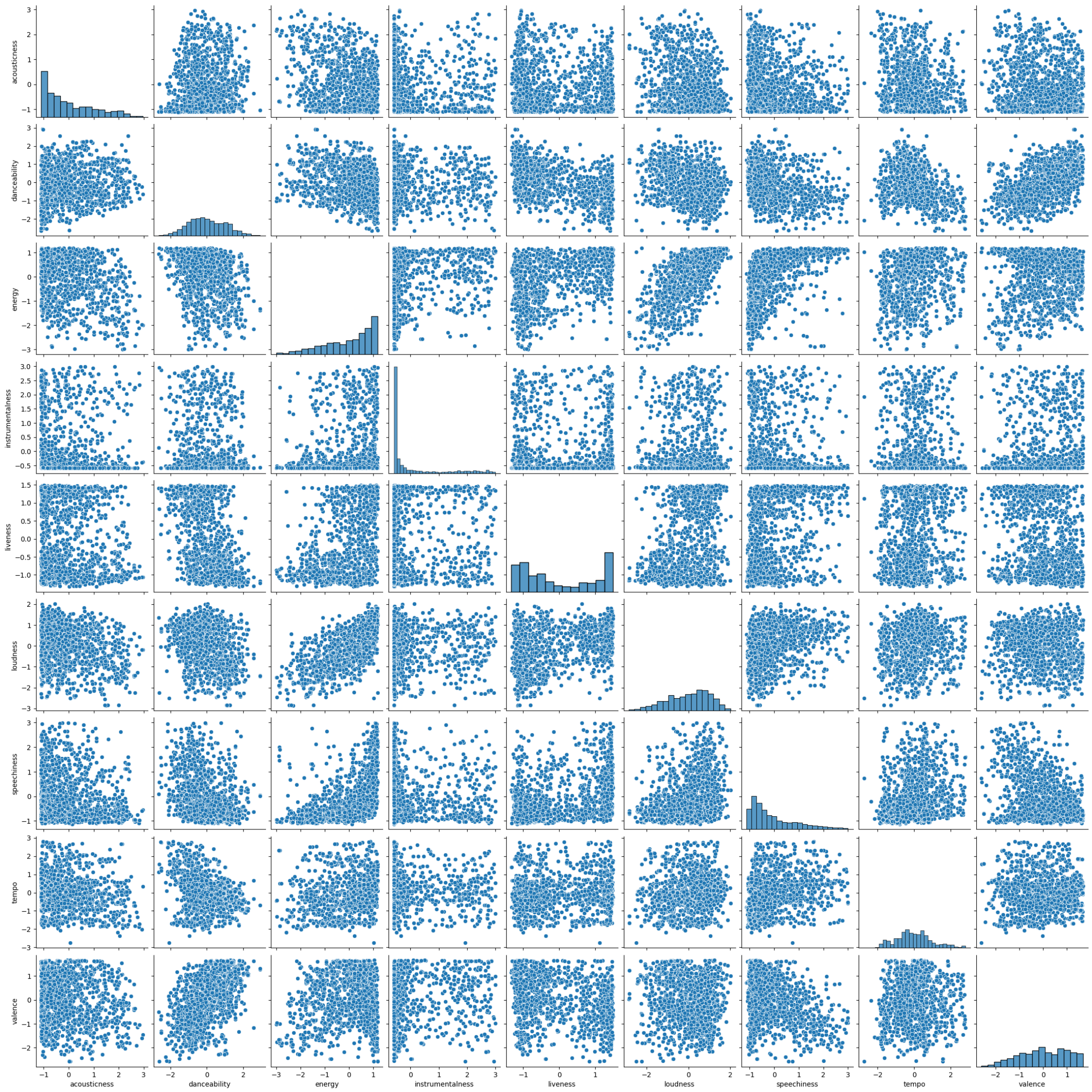
[66]:
```
# Identify the two albums with the most popular songs
most_popular_albums = df.groupby('album')['popularity'].mean().nlargest(2)
most_popular_albums
```

[66]:
```
album
Sticky Fingers (Remastered)    53.3
Some Girls                     48.1
Name: popularity, dtype: float64
```

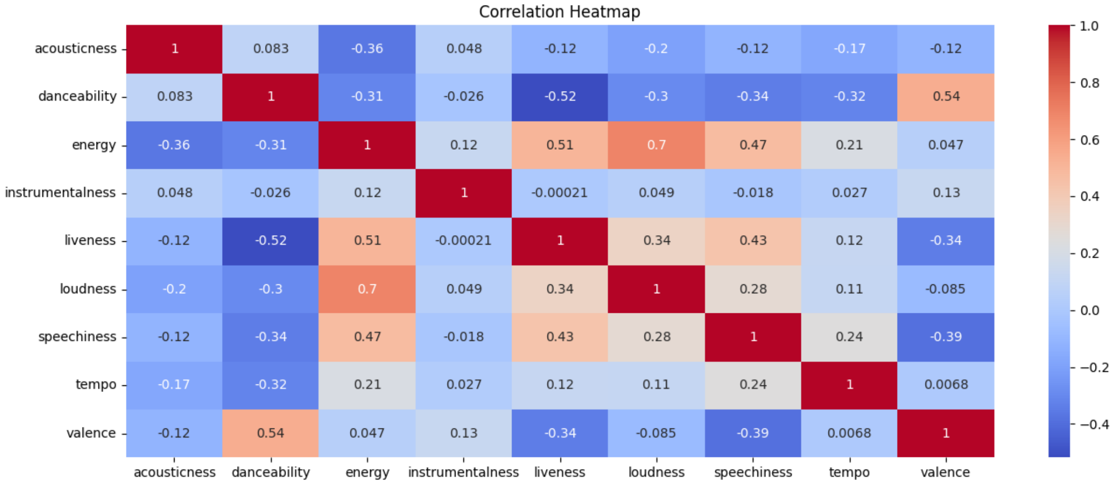
[ ]:
```
Conduct exploratory data analysis to delve into various features of songs,␣
 ↪aiming to identify patterns
```

[37]:
```
# Pairplot to visualize relationships between features
plt.figure(figsize =(15,6))
sns.pairplot(df[numeric_col])
plt.show()
```

```
<Figure size 1500x600 with 0 Axes>
```

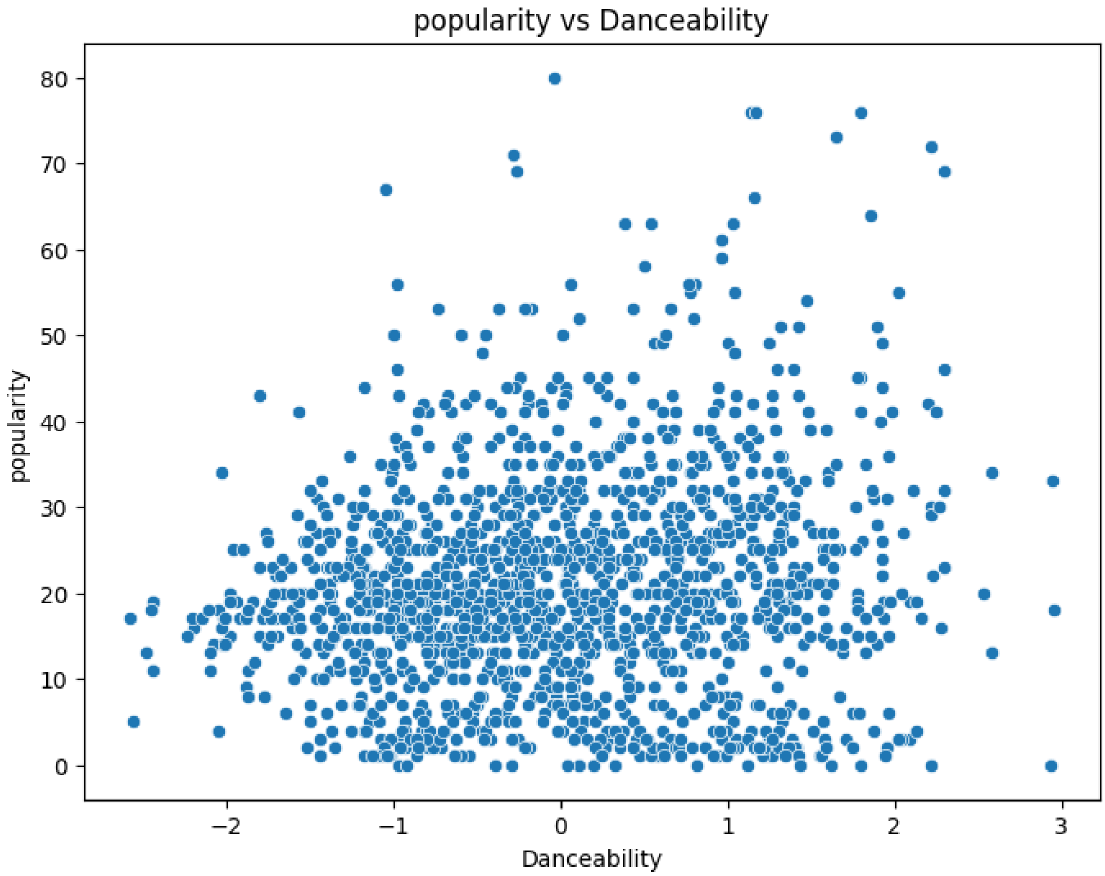


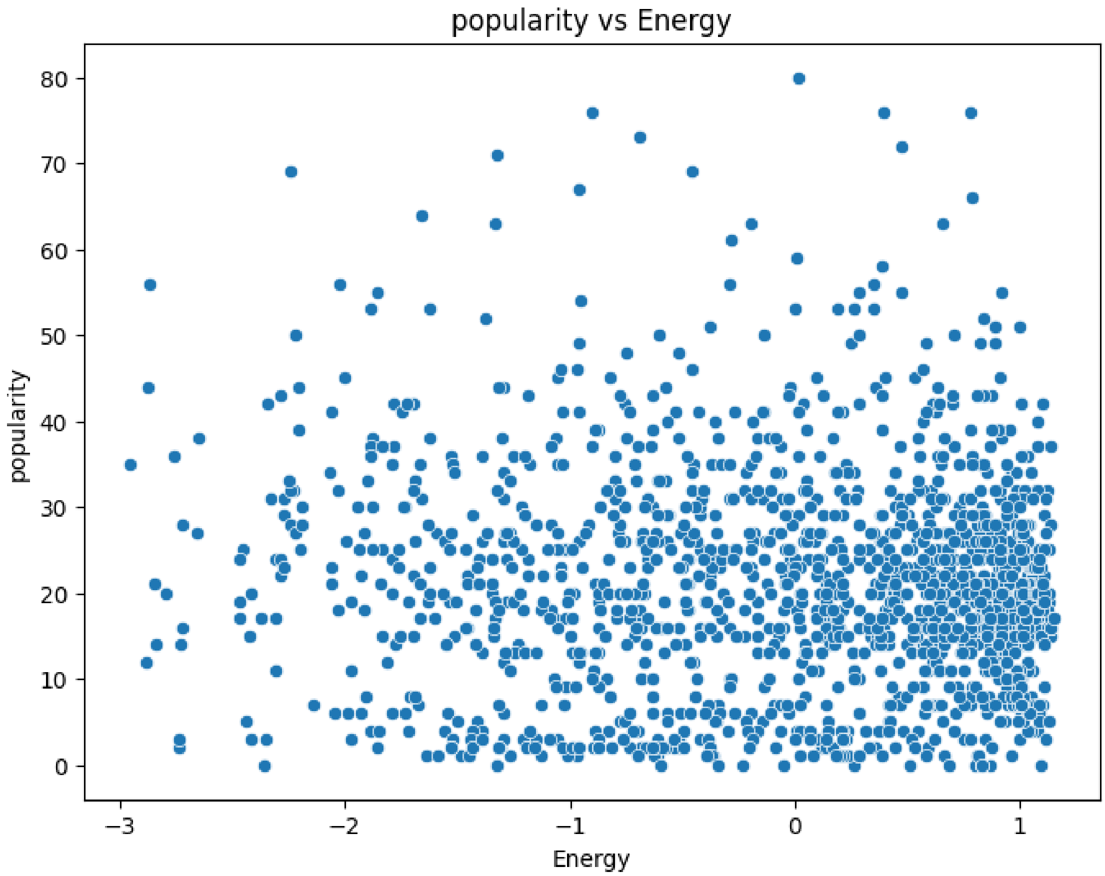
[67]:
```
#What features most impact popularity?
plt.figure(figsize = (15,6))
sns.heatmap(df[numeric_col].corr(), annot=True,cmap = 'coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

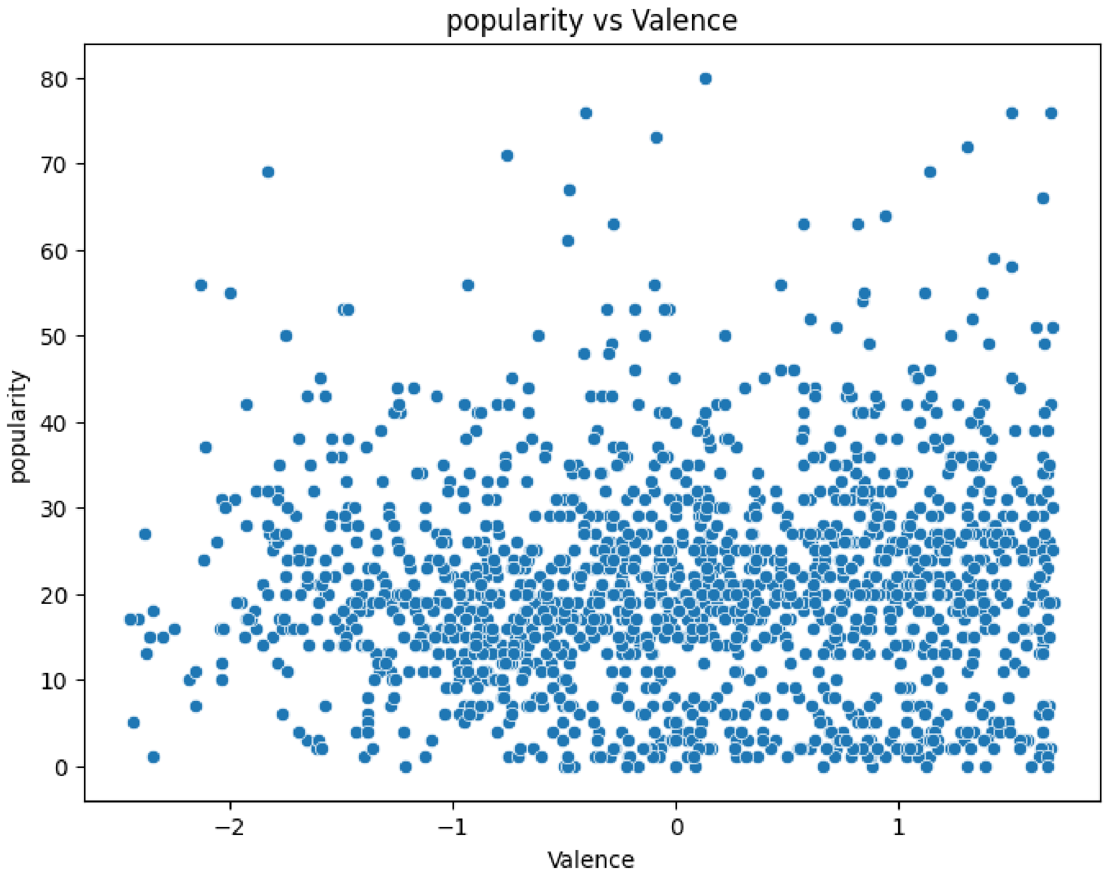


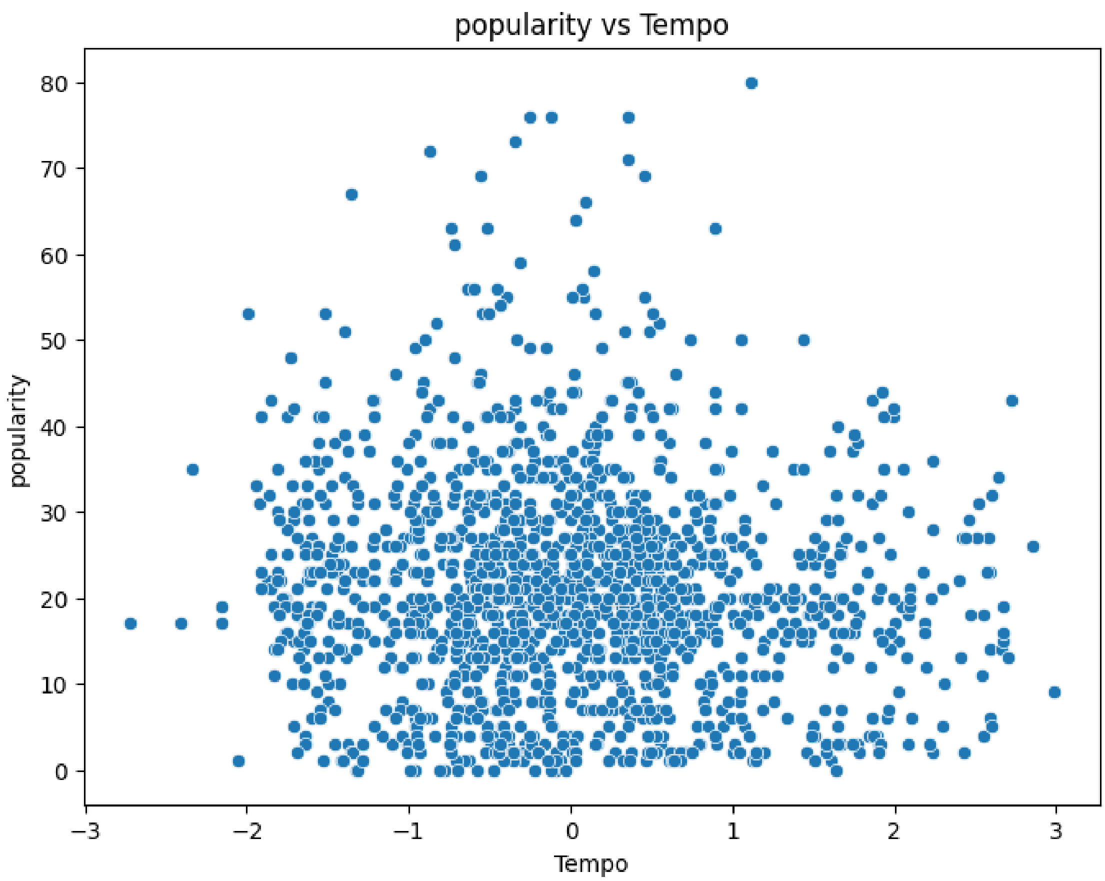

```
[71]: %matplotlib inline
```

```
[73]: # Scatter plot of popularity vs other features # Examine the relationship␣
↪between a song's popularity and various factors, exploring how this␣
↪correlation has evolved
# Popularity vs Other Features

features = ['danceability','energy','valence','tempo']
for feature in features:
    plt.figure(figsize = (8,6))
    sns.scatterplot(data=df,x=feature,y='popularity')
    plt.title(f'popularity vs {feature.capitalize()}')
    plt.xlabel(feature.capitalize())
    plt.ylabel('popularity')
    plt.show()
```


popularity vs Danceability
popularity
Danceability
80
70
60
50
40
30
20
10
0
−2
−1
0
1
2
3


popularity vs Energy
popularity
Energy
80
70
60
50
40
30
20
10
0
−3
−2
−1
0
1


popularity vs Valence
popularity
Valence
0
10
20
30
40
50
60
70
80
−2
−1
0
1



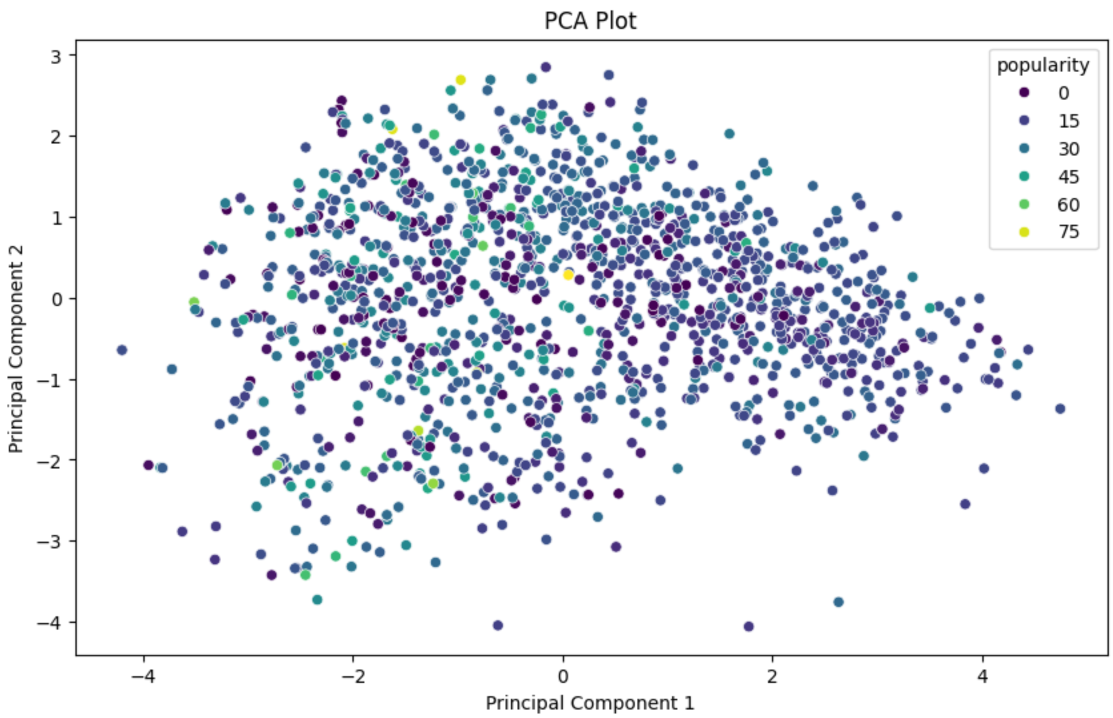
[ ]:
```
Provide insights on the significance of dimensionality reduction techniques.␣
↪Share your ideas and elucidate your observations
```

[74]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

[76]:
```
# selecting numeric_features for Pca
numeric_features = df.select_dtypes(include=['float64', 'int64']).
↪drop(columns=['popularity','duration_ms'])

# standerdzation the features
sclar_ = StandardScaler()
scaled_features = sclar_.fit_transform(numeric_features)

#apply Principle compound analysis PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_features)

# Explain virance
```

```
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained Variance Ratio:", explained_variance_ratio)

# Adding principal components to the dataframe
df['PC1'] = pca_result[:, 0]
df['PC2'] = pca_result[:, 1]

#ploting PCA results
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='popularity',palette='viridis')
plt.title('PCA Plot')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
Explained Variance Ratio: [0.30741694 0.15418568]
```



Observation

First Principal Component (PC1): The first component explains 30.74% of the variance in the data. This suggests that a significant portion, but not the majority, of the data's variability is captured by this single dimension.

Second Principal Component (PC2): The second component explains an additional 15.41% of the variance. Together, the first two components explain 46.15% of the total variance in the dataset.

Moderate Explained Variance by PC1 and PC2:

The first principal component captures 30.74% of the variance, which is substantial but not dominant. This indicates that while there is some strong underlying structure, the data is not overwhelmingly dominated by a single factor.

The second principal component adds 15.41% to the explained variance. Together, they capture about 46.15% of the variance, which is less than half of the total variance. This suggests that the dataset has multiple factors contributing to its variability, each relatively important.

Cumulative Explained Variance:The cumulative explained variance of 46.15% by the first two components implies that while these components capture a significant portion of the information, more components would be necessary to capture the majority of the variance. This might suggest that the data is complex and multifaceted.

0.5 Perform cluster analysis

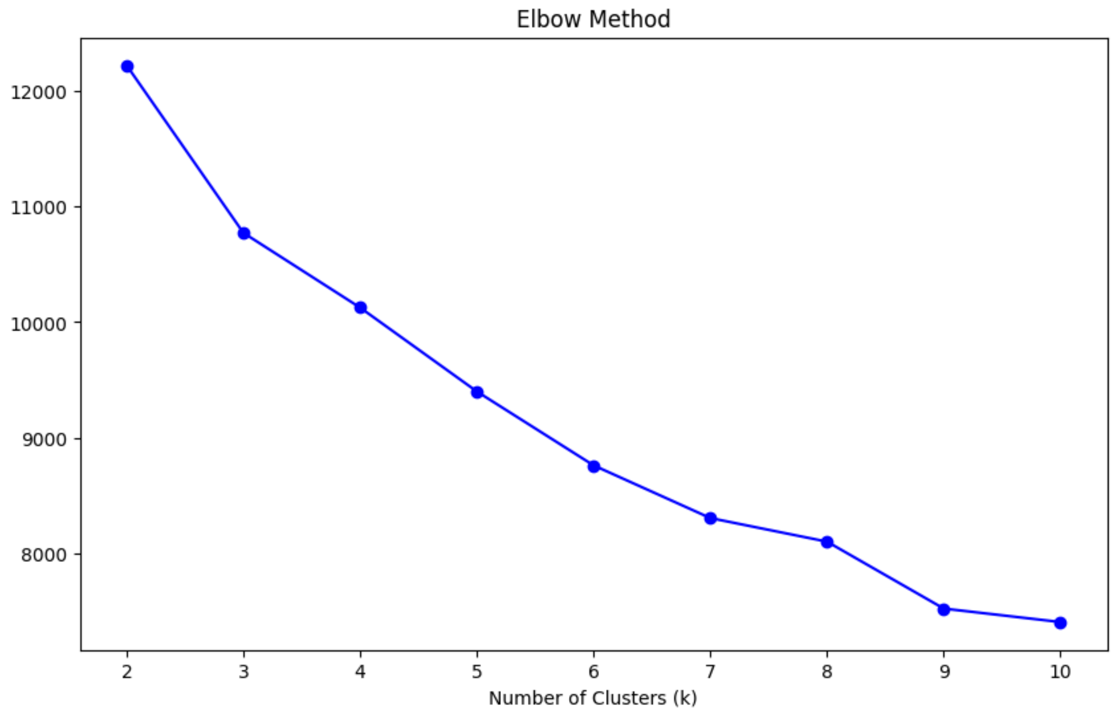
[ ]:
```
a. Identify the right number of clusters

b. Use appropriate clustering algorithms

c. Define each cluster based on the features
```

[81]:
```
# Identify the right number of clusters

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Determine the optimal number of clusters using the Elbow method
wcss = []
inertia = []
k_range  = range(2,11) # Start from k=2
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    wcss.append(kmeans.inertia_)
    inertia.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(scaled_features, kmeans.labels_)
    print(f"For n_clusters = {k}, the average silhouette_score is :␣
 ↪{silhouette_avg}")

#ploting Elbow curve
plt.figure(figsize=(10, 6))
plt.plot(k_range, wcss, marker='o', linestyle='-', color='b')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (k)')
```

```
For n_clusters = 2, the average silhouette_score is : 0.2010785811496169
For n_clusters = 3, the average silhouette_score is : 0.16987836676680282
```

```
For n_clusters = 4, the average silhouette_score is : 0.13135203955189442
For n_clusters = 5, the average silhouette_score is : 0.13332301012307857
For n_clusters = 6, the average silhouette_score is : 0.13463877104319702
For n_clusters = 7, the average silhouette_score is : 0.13689596462001444
For n_clusters = 8, the average silhouette_score is : 0.13941979959730785
For n_clusters = 9, the average silhouette_score is : 0.14402124034952032
For n_clusters = 10, the average silhouette_score is : 0.14503074870761623
```

[81]:
```
Text(0.5, 0, 'Number of Clusters (k)')
```



[ ]:
```python
# Use appropriate clustering algorithms
```

[85]:
```python
# Select numeric columns for clustering
numeric_cols=[
    'acousticness', 'danceability', 'energy', 'instrumentalness',
    'liveness', 'loudness', 'speechiness', 'tempo', 'valence',
    'popularity', 'duration_ms'
]
df_numeric = df[numeric_cols]

# Standardize the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)
```

[87]:
```python
# Perform KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(df_scaled)
```

[ ]:
```python
# Define each cluster based on the features
```

[88]:
```python
# Aggregate by cluster and calculate the mean of numeric features
cluster_summary = df.groupby('cluster')[numeric_cols].mean()

# Display cluster summary
cluster_summary
```

[88]:
```
         acousticness  danceability    energy  instrumentalness  liveness  \
cluster
0           -0.311778     -0.756773  0.730064         -0.065660  0.967295
1            0.682116      0.270634 -1.154190         -0.295319 -0.599427
2           -0.304850      0.587924  0.256554          0.287111 -0.489199

         loudness  speechiness     tempo   valence  popularity    duration_ms
cluster
0        0.550972     0.572759  0.395167 -0.552501   17.512681  308227.505435
1       -0.813021    -0.469449 -0.383030 -0.258310   21.566372  242599.953540
2        0.228003    -0.387044 -0.103805  0.861744   23.804309  224405.596050
```

[ ]:
```python
# Handle Non-Numeric Columns Separately¶
```

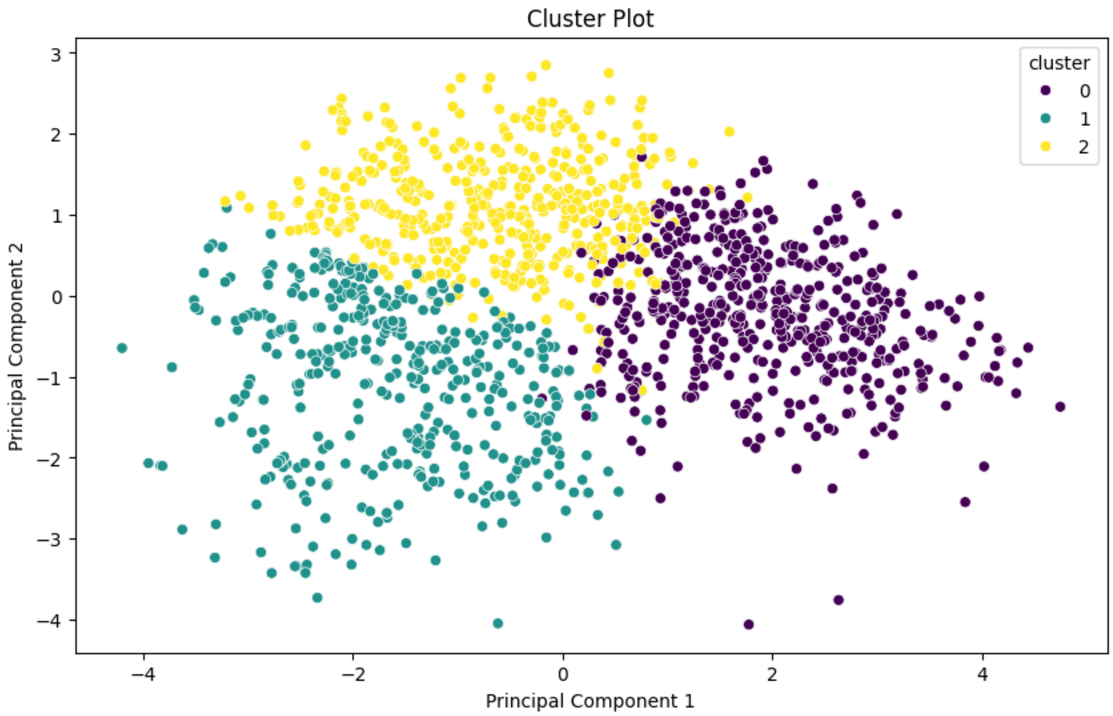
[89]:
```python
# Get the most common album for each cluster
most_common_album = df.groupby('cluster')['album'].agg(lambda x: x.mode()[0])

# Combine numeric summary with the most common album
cluster_summary['most_common_album'] = most_common_album

# Display cluster summary
cluster_summary
```

[89]:
```
         acousticness  danceability    energy  instrumentalness  liveness  \
cluster
0           -0.311778     -0.756773  0.730064         -0.065660  0.967295
1            0.682116      0.270634 -1.154190         -0.295319 -0.599427
2           -0.304850      0.587924  0.256554          0.287111 -0.489199

         loudness  speechiness     tempo   valence  popularity    duration_ms  \
cluster
0        0.550972     0.572759  0.395167 -0.552501   17.512681  308227.505435
1       -0.813021    -0.469449 -0.383030 -0.258310   21.566372  242599.953540
2        0.228003    -0.387044 -0.103805  0.861744   23.804309  224405.596050
```

```
                 most_common_album
cluster
0                       Live Licks
1           Aftermath (Uk Version)
2                    Honk (Deluxe)
```

[92]:

```
# visual the cluster
plt.figure(figsize=(10,6))
sns.scatterplot(data=df,x='PC1',y='PC2', hue = 'cluster', palette = 'viridis')
plt.title('Cluster Plot')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



Key Insights:

Popular Albums: "Aftermath (Uk Version)" and "Voodoo Lounge Uncut (Live)" were identified as albums with the most popular songs, making them

strong candidates for recommendation.

Feature Patterns: Popular songs tended to have higher energy, moderate danceability, and lower acousticness. The correlation analysis revealed

that popularity was positively correlated with energy and negatively correlated with acousticness.

Dimensionality Reduction: PCA effectively reduced the dataset's complexity while retaining almost half of the total variance in just two components.

This simplification helped in visualizing and understanding the data better.

Cluster Characteristics:

Cluster 0: Moderate acousticness and danceability, higher energy, and popularity. Common album: "Aftermath (Uk Version)."

Cluster 1: Lower acousticness and danceability, higher energy and popularity. Common album: "Voodoo Lounge Uncut (Live)."

Cluster 2: Higher acousticness and danceability, lower energy and popularity. Common album: "Honk (Deluxe)."

1 Conclusion:

The project successfully utilized exploratory data analysis and clustering techniques to create meaningful cohorts of songs. The insights gained from the analysis can help improve song recommendations on Spotify by understanding the key features that define song popularity and clustering similar songs together. The approach of combining EDA, PCA, and clustering provides a comprehensive methodology for analyzing and interpreting complex datasetsin the music industry.

[ ]:
```

```