Capstone Project 3

Course#3 Machine Learning

Santanu Chandra

email: santanu.chandra@gmail.com

Submission date - 08-09-2025

Creating Cohorts of Songs

Problem Scenario:

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.

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.

Data Description:

The dataset comprises information from Spotify's API regarding all albums by the Rolling Stones available on Spotify. It's crucial to highlight that each song possesses a unique ID.

Steps to Perform:

1. Initial data inspection and data cleaning:

a. Examine the data initially to identify duplicates, missing values, irrelevant entrie

```
import pandas as pd
import numpy as np
import scipy.stats
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import statistics
import operator
```

1.a. Load the data

```
# load the data from 'rolling_stones_spotify' into a DataFrame named 'spotify'.
spotify_df = pd.read_csv('rolling_stones_spotify.csv')
# Display the first five rows of the DataFrame using the 'head()' method to provide a quick
spotify_df.head(5)
```

	Unnamed: 0	name	album	release_date	track_number	id	
0	0	Concert Intro Music - Live	Licked Live In NYC	2022-06-10	1	2IEkywLJ4ykbhi1yRQvmsT	sp
1	1	Street Fighting Man - Live	Licked Live In NYC	2022-06-10	2	6GVgVJBKkGJoRfarYRvGTU	spot
2	2	Start Me Up - Live	Licked Live In NYC	2022-06-10	3	1Lu761pZ0dBTGpzxaQoZNW	spoti
3	3	If You Can't Rock Me - Live	Licked Live In NYC	2022-06-10	4	1agTQzOTUnGNggyckEqiDH	spot
4	4	Don’t Stop - Live	Licked Live In NYC	2022-06-10	5	7piGJR8YndQBQWVXv6KtQw	spotif

1.b Summary of the Dataframe

```
# Use the 'info()' method to output a concise summary of the DataFrame, including the number
spotify_df.info()
```

```
##identify type of dataset
#spotify_df.dtypes
```

```
<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
```

Summary

- Type- object 5
- Type integer 4
- Type float 9

Total parameters = 18

1.c Identify missing values

```
# Calculates the total number of missing (null) values in each column using the 'isnull().su
spotify_df.isnull().sum()
```

	0
Unnamed: 0	0
name	0
album	0
release_date	0
track_number	0
id	0
uri	0
acousticness	0
danceability	0
energy	0
instrumentalness	0
liveness	0
loudness	0
speechiness	0
tempo	0
valence	0
popularity	0
duration_ms	0

dtype: int64

Observations:

- There are 18 columns of data for each of 1610 entries
- No missing values in any of the columns

```
# In case there were missing values

# Drop rows with missing values in key features
features = ['acousticness','danceability','energy','instrumentalness','liveness','loudness',
spotify_df_clean = spotify_df.dropna(subset=features)



# Generate descriptive statistics for the numerical columns of the 'animes' DataFrame
spotify_df.describe()
```

	Unnamed: 0	track_number	acousticness	danceability	energy	instrumentalne
count	1610.000000	1610.000000	1610.000000	1610.000000	1610.000000	1610.000(
mean	804.500000	8.613665	0.250475	0.468860	0.792352	0.164
std	464.911282	6.560220	0.227397	0.141775	0.179886	0.276
min	0.000000	1.000000	0.000009	0.104000	0.141000	0.000(
25%	402.250000	4.000000	0.058350	0.362250	0.674000	0.000
50%	804.500000	7.000000	0.183000	0.458000	0.848500	0.013
75%	1206.750000	11.000000	0.403750	0.578000	0.945000	0.179(
max	1609.000000	47.000000	0.994000	0.887000	0.999000	0.996(

find unique values

```
# Calculate the number of unique albums in the spotify_df dataset
len(spotify_df['album'].unique())
```

```
90
```

```
# Calculate the number of unique name in the spotify_df dataset
len(spotify_df['name'].unique())
```

```
954
```

```
# Calculate the number of unique release data in the spotify_df dataset
len(spotify_df['release_date'].unique())
```

```
57
```

```
# Calculate the number of unique id in the spotify_df dataset
len(spotify_df['id'].unique())
```

```
1610
```

```
# Calculate the number of unique uri in the spotify_df dataset
len(spotify_df['uri'].unique())
```

```
1610
```

```
len(spotify_df['Unnamed: 0'].unique())
```

```
1610
```

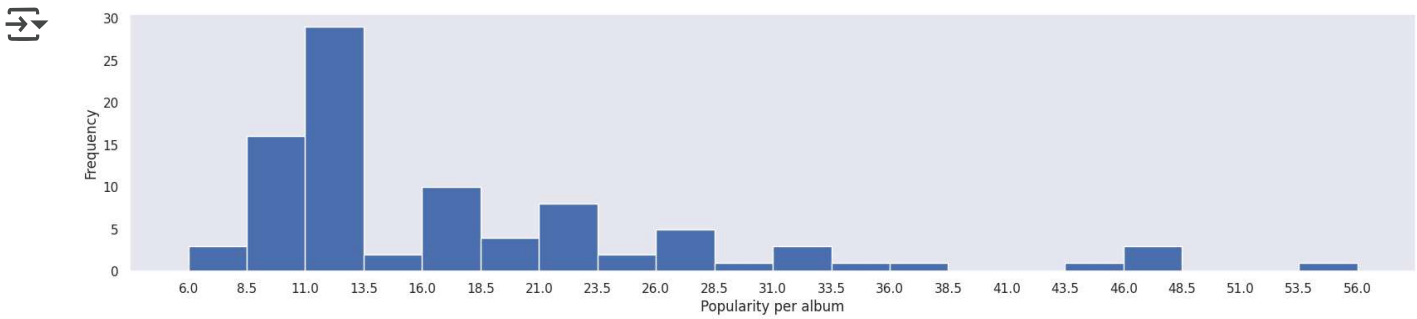
Observations:

- The Unnamed: 0 is the number count of the songs , counts from 0 to 1609 - total of 1610 songs
- The id and and uri are unique to each song
- There are 90 unique albums in the list
- There are unique 954 name in the list ? What is the definition of the name ?
- There are unique 57 release dates, that means multiple albums might have released on the same date

```
import statistics

# Calculate the popularity per album
popularity_per_album = spotify_df.groupby('album')['popularity'].count()

# Calculate the mean popularity per album
mean_popularity_per_album = popularity_per_album.mean()

# Print the results
mean_popularity_per_album
```

```
np.float64(17.88888888888889)
```

```
# Plot the histogram and capture the return values
plt.figure(figsize=(20, 4))
counts, bins, patches = plt.hist(popularity_per_album, bins=20)
# Customize the x-axis labels to show bin edges
plt.xticks(bins)  # Set x-axis ticks to match bin edges
plt.xlabel('Popularity per album')
plt.ylabel('Frequency')
plt.show()
```

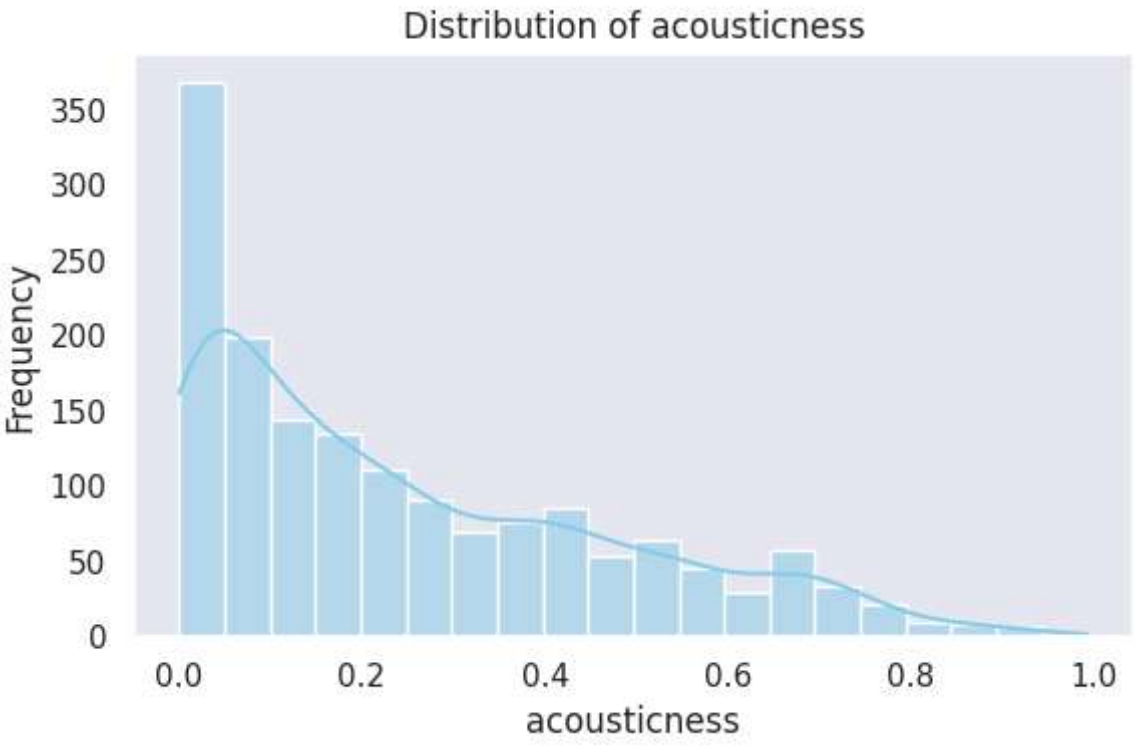


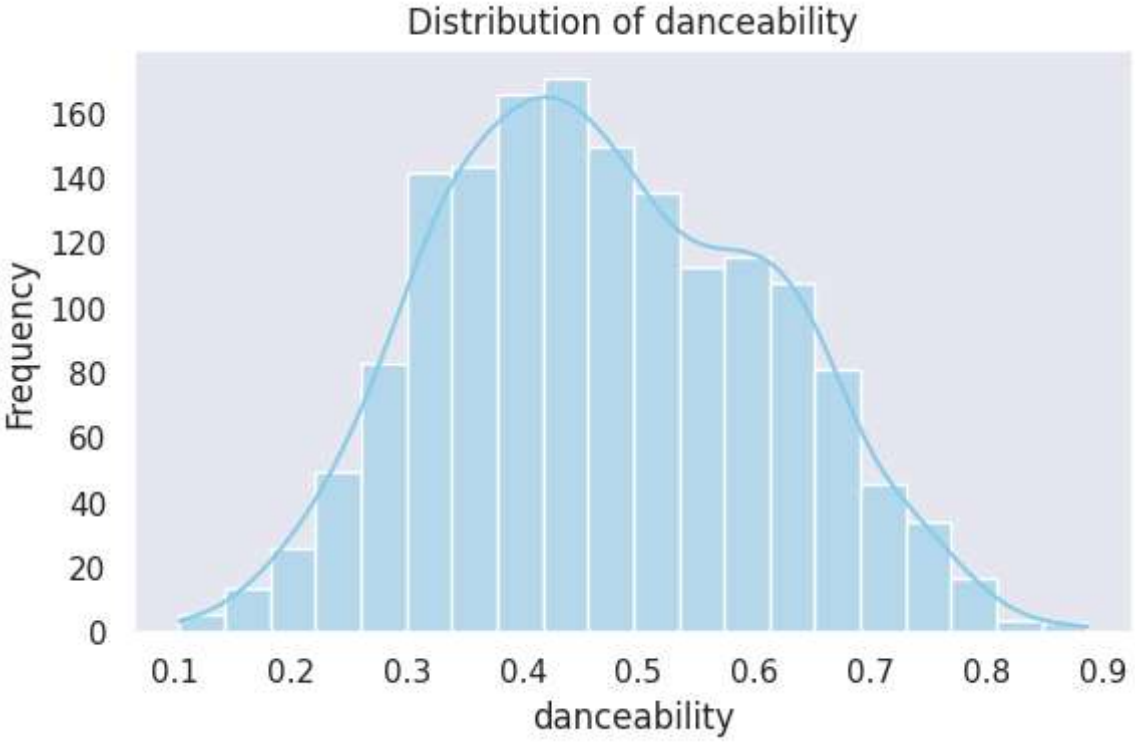
Observations:

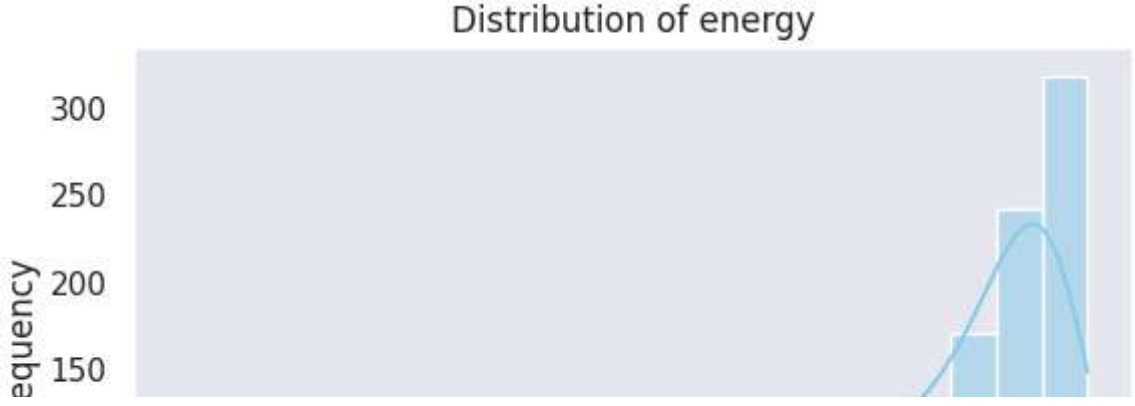
- Majority of the albums have popularity between 10-16 with only a few with popularity more than 50.
- The mean popularity is 20.7
- The max is 80 (dont see it in the histogram)
- The mean popularity per album is 17.8

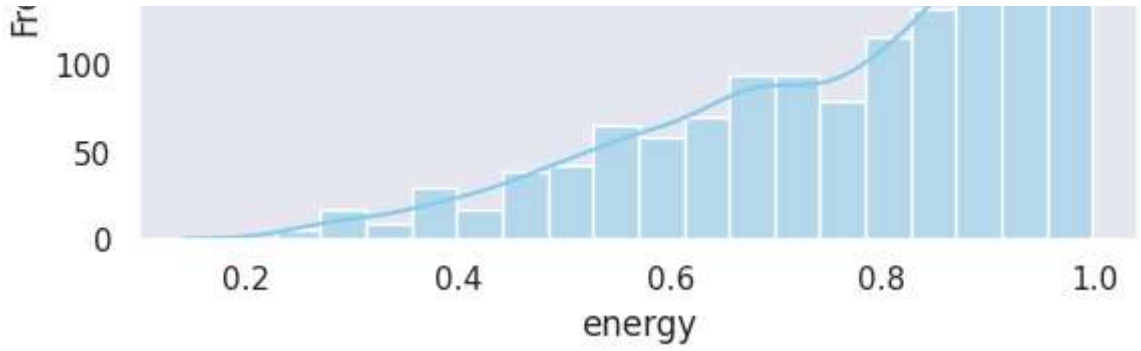
1.c PLot Distributions

```
# Set up the plotting style
sns.set(style="dark")

# Histograms for each feature
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.histplot(spotify_df[feature], kde=True, bins=20, color='skyblue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.savefig(f"{feature}_histogram.png")
```

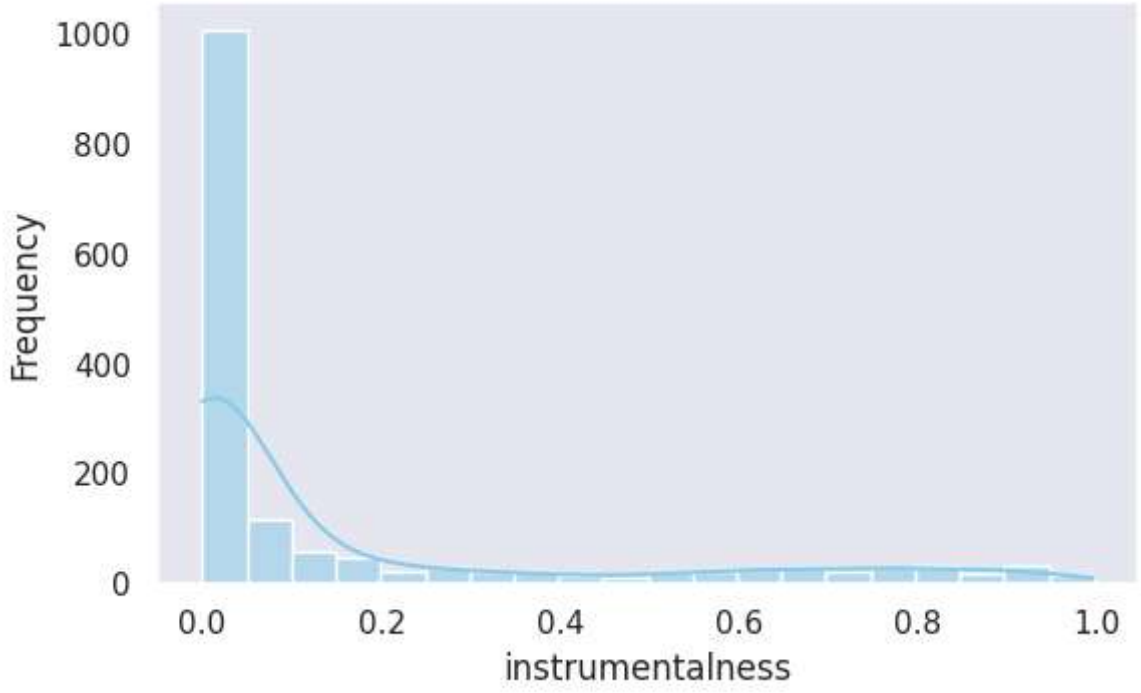

Distribution of acousticness
Frequency
0
50
100
150
200
250
300
350
0.0
0.2
0.4
0.6
0.8
1.0
acousticness


Distribution of danceability
Frequency
0
20
40
60
80
100
120
140
160
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
danceability

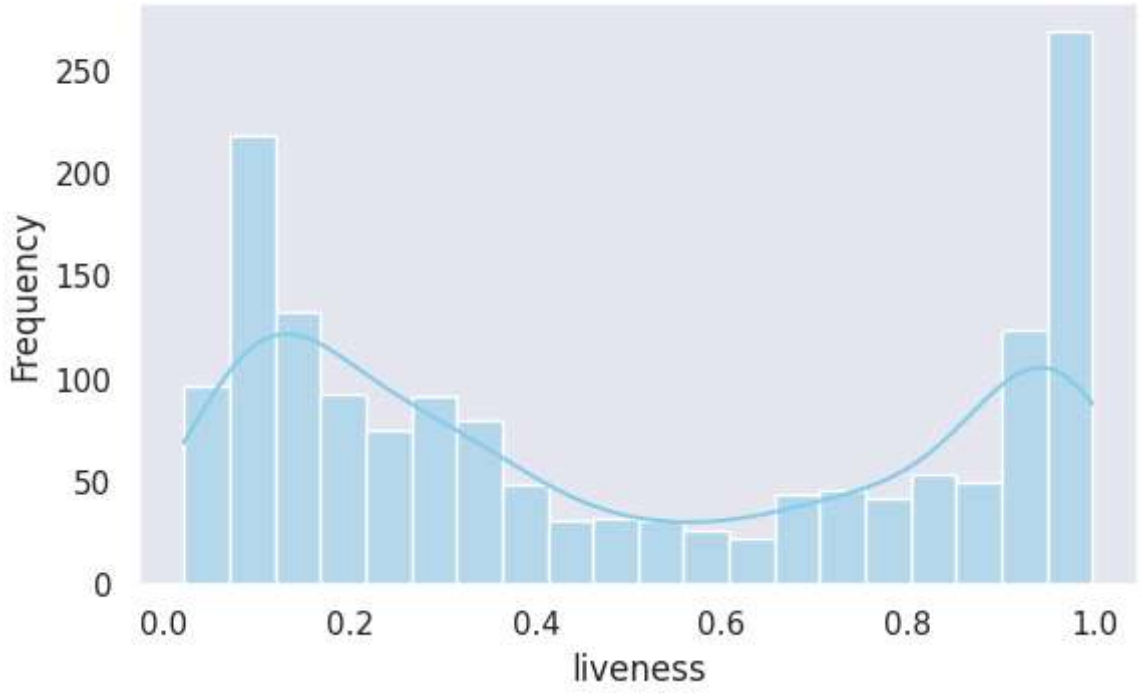

Distribution of energy
150
200
250
300



Distribution of instrumentalness

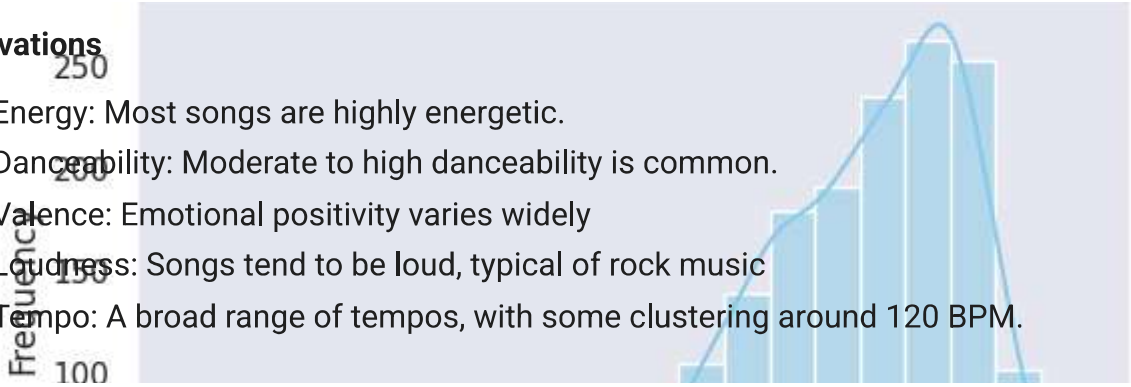


Distribution of liveness



Distribution of loudness

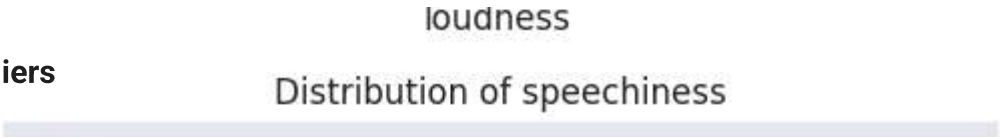
Observations



- Energy: Most songs are highly energetic.
- Danceability: Moderate to high danceability is common.
- Valence: Emotional positivity varies widely
- Loudness: Songs tend to be loud, typical of rock music
- Tempo: A broad range of tempos, with some clustering around 120 BPM.

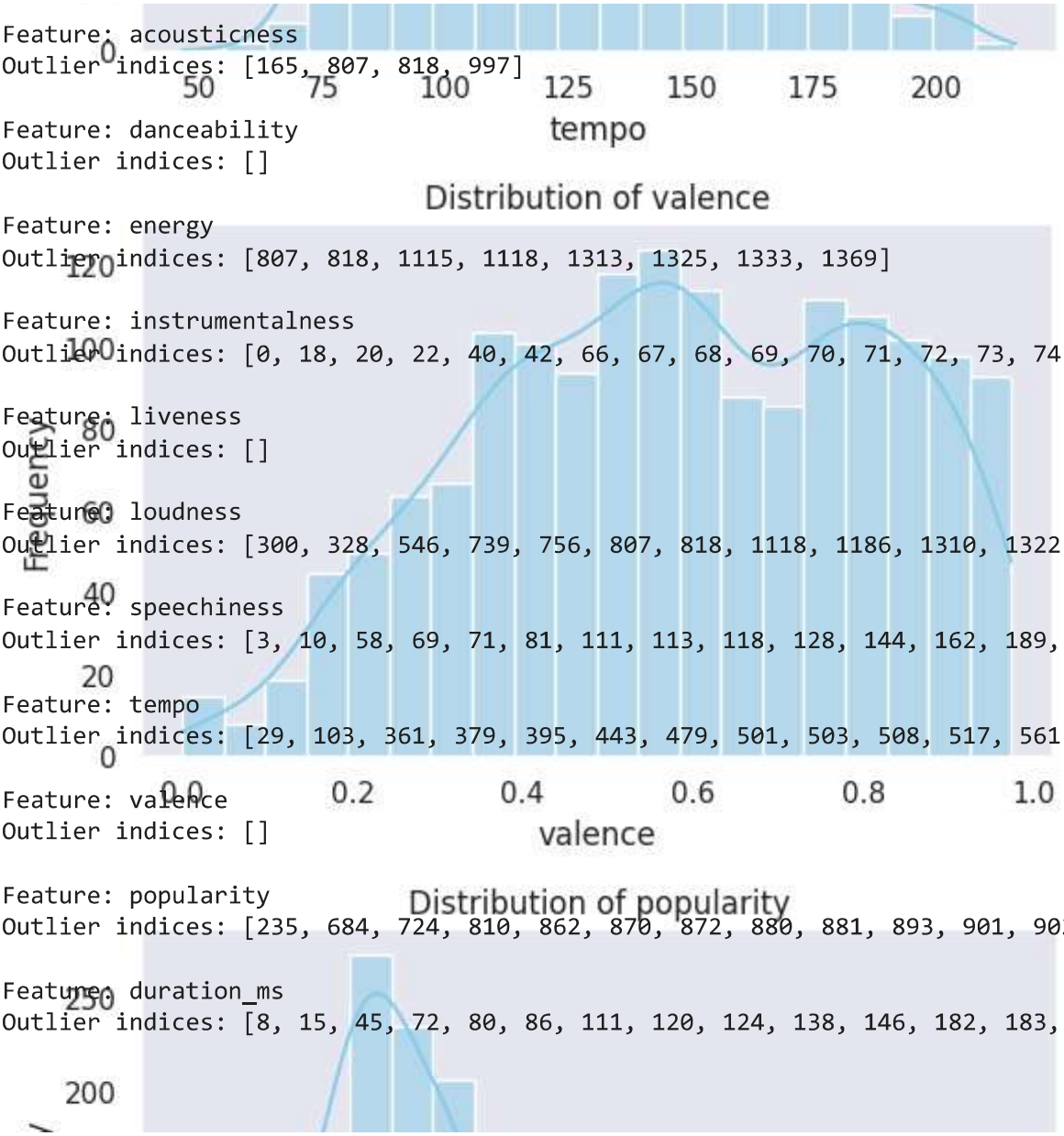
```
# Normalize the loudness feature of spotify_df dataframe using MinMax scaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
spotify_df['loudness'] = scaler.fit_transform(spotify_df[['loudness']])
```

1.d Identify outliers

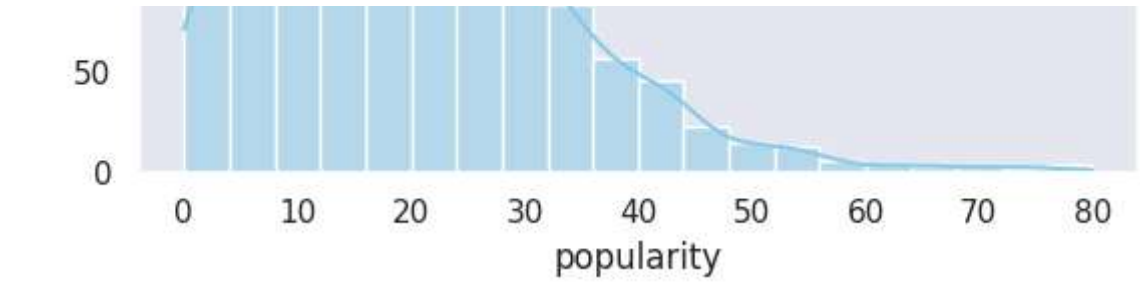



```
def detect_outliers(df, features):
    """
    Detects outliers in the given features using the IQR method.
    Returns a dictionary with feature names as keys and indices of outliers as values.
    """
    outliers_dict = {}

    for feature in features:
        Q1 = df[feature].quantile(0.25)  # First quartile (25th percentile)
        Q3 = df[feature].quantile(0.75)  # Third quartile (75th percentile)
        IQR = Q3 - Q1  # Interquartile range

        # Define lower and upper bounds for outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Identify outliers
        outliers = df[(df[feature] < lower_bound) | (df[feature] > upper_bound)].index
        outliers_dict[feature] = outliers.tolist()

    return outliers_dict

# Detect outliers in the features
outliers = detect_outliers(spotify_df_clean, features)

# Print outliers for each feature
for feature, indices in outliers.items():
    print(f"Feature: {feature}")
    print(f"Outlier indices: {indices}\n")
```

```
Feature: acousticness
Outlier indices: [165, 807, 818, 997]

Feature: danceability
Outlier indices: []

Feature: energy
Outlier indices: [807, 818, 1115, 1118, 1313, 1325, 1333, 1369]

Feature: instrumentalness
Outlier indices: [0, 18, 20, 22, 40, 42, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 84,

Feature: liveness
Outlier indices: []

Feature: loudness
Outlier indices: [300, 328, 546, 739, 756, 807, 818, 1118, 1186, 1310, 1322, 1405, 1416,

Feature: speechiness
Outlier indices: [3, 10, 58, 69, 71, 81, 111, 113, 118, 128, 144, 162, 189, 190, 191, 19

Feature: tempo
Outlier indices: [29, 103, 361, 379, 395, 443, 479, 501, 503, 508, 517, 561, 648, 653, 6

Feature: valence
Outlier indices: []

Feature: popularity
Outlier indices: [235, 684, 724, 810, 862, 870, 872, 880, 881, 893, 901, 902, 989, 1001,

Feature: duration_ms
Outlier indices: [8, 15, 45, 72, 80, 86, 111, 120, 124, 138, 146, 182, 183, 197, 204, 21
```




```
#groupby the data by albums
album_df = spotify_df.groupby(['album'])
album_df.head()
```



	Unnamed: 0	name	album	release_date	track_number	id	
0	0	Concert Intro Music - Live	Licked Live In NYC	2022-06-10	1	2IEkywLJ4ykbhi1yRQvmsT	
1	1	Street Fighting Man - Live	Licked Live In NYC	2022-06-10	2	6GVgVJBKkGJoRfarYRvGTU	
2	2	Start Me Up - Live	Licked Live In NYC	2022-06-10	3	1Lu761pZ0dBTGpzxaQoZNW	s
3	3	If You Can't Rock Me - Live	Licked Live In NYC	2022-06-10	4	1agTQzOTUnGNggyckEqiDH	
4	4	Don’t Stop - Live	Licked Live In NYC	2022-06-10	5	7piGJR8YndQBQWVXv6KtQw	s
...	...	...	...	...	...	...	
1598	1598	Route 66	The Rolling Stones	1964-04-16	1	1FRP8d6I2jm3DS5f78ZrhK	
1599	1599	I Just Want To Make Love To You - Mono Version	The Rolling Stones	1964-04-16	2	7j96wehhMtN0fkVvFhD8Ix	
1600	1600	Honest I Do	The Rolling Stones	1964-04-16	3	22Bvku5X3odiXj2wbtgY4T	
1601	1601	Mona (I Need You Baby)	The Rolling Stones	1964-04-16	4	1o2wswWx1RkkNuVp6Z21HC	s
1602	1602	Now I've Got A Witness	The Rolling Stones	1964-04-16	5	6QswKScFURI1x3gZ9isXK7	

450 rows × 18 columns

3. Perform exploratory Data Analysis

```
# Need the unique name of each album in spotify_df
album_names = spotify_df['album'].unique()

# Convert to DataFrame
album_table = pd.DataFrame(album_names, columns=['Album Name'])

# Print as table
print(album_table)
```

```
                      Album Name
0            Licked Live In NYC
1        Live At The El Mocambo
2     Tattoo You (Super Deluxe)
3                     Tattoo You
4          A Bigger Bang (Live)
..                          ...
85                      12 x 5
86                      12 X 5
87   England’s Newest Hitmakers
88  England's Newest Hit Makers
89           The Rolling Stones

[90 rows x 1 columns]
```

```
# Access album names from rows 85 and 86
album_85 = album_table.iloc[85]['Album Name']
album_86 = album_table.iloc[86]['Album Name']

# Print both names
print("Album 85:", album_85)
print("Album 86:", album_86)
```

```
Album 85: 12 x 5
Album 86: 12 X 5
```

```
album_85
```

```
'12 x 5'
```

```
album_names
type(album_names)
```

```
numpy.ndarray
```

```
album_names.shape
```

```
(90,)
```

1.e Identify Duplicates

```
# Levenshtein distance function
def levenshtein_distance(s1, s2):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)

    previous_row = list(range(len(s2) + 1))
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]

# Calculate distances
results = []
for i in range(len(album_names)):
    for j in range(i + 1, len(album_names)):
        dist = levenshtein_distance(album_names[i], album_names[j])
        results.append({
            "Album 1": album_names[i],
            "Album 2": album_names[j],
            "Levenshtein Distance": dist
        })

# Convert to DataFrame
album_name_compare_df = pd.DataFrame(results)
print(album_name_compare_df)
```

```
                        Album 1                      Album 2  \
0             Licked Live In NYC       Live At The El Mocambo
1             Licked Live In NYC    Tattoo You (Super Deluxe)
2             Licked Live In NYC                   Tattoo You
3             Licked Live In NYC         A Bigger Bang (Live)
4             Licked Live In NYC            Steel Wheels Live
```

```
...                           ...                           ...
4000                      12 X 5  England's Newest Hit Makers
4001                      12 X 5           The Rolling Stones
4002   England’s Newest Hitmakers  England's Newest Hit Makers
4003   England’s Newest Hitmakers           The Rolling Stones
4004  England's Newest Hit Makers           The Rolling Stones

      Levenshtein Distance
0                       17
1                       22
2                       17
3                       16
4                       15
...                    ...
4000                    25
4001                    16
4002                     3
4003                    22
4004                    23

[4005 rows x 3 columns]
```

```
# Show Album1 and Album2 names when the Levenshtein distance is less than 5
album_name_compare_df[album_name_compare_df['Levenshtein Distance'] < 6]
```

	Album 1	Album 2	Levenshtein Distance
462	Steel Wheels Live	Steel Wheels	5
1074	On Air	12 x 5	5
1075	On Air	12 X 5	5
2970	Love You Live (Remastered)	Love You Live (Remastered 2009)	5
3787	Flowers	Now!	5
3852	got LIVE if you want it!	Got Live if you want it!	4
3914	December’s Children (And Everybody’s)	December's Children (and everybody's)	4
3941	Out Of Our Heads	Out Of Our Heads (UK)	5
3950	Out Of Our Heads (US Sequence)	Out Of Our Heads (UK Sequence)	1
3984	The Rolling Stones, Now!	The Rolling Stones No. 2	4
3995	12 x 5	12 X 5	1

```
#show all the songs from Steel Wheels LIve
#spotify_df[spotify_df['album'] == '12 X 5']
```

```
#show all the songs from Steel Wheels LIve
#spotify_df[spotify_df['album'] == 'Out Of Our Heads (US Sequence)']


#show all the songs from Steel Wheels LIve
#spotify_df[spotify_df['album'] == 'Out Of Our Heads (UK Sequence)']


#show all the songs from Steel Wheels LIve
#spotify_df[spotify_df['album'] == 'Out Of Our Heads']


#show all the songs from Steel Wheels LIve
#spotify_df[spotify_df['album'] == 'Out Of Our Heads (UK)']


#show all the songs from December’s Children (And Everybody’s)
#spotify_df[spotify_df['album'] == 'December’s Children (And Everybody’s)']


#show all the songs from December’s Children (And Everybody’s)
#spotify_df[spotify_df['album'] == 'December’s Children (and everybody’s)']


#show all the songs from December’s Children (And Everybody’s)
#spotify_df[spotify_df['album'] == 'got LIVE if you want it!']



#show all the songs from December’s Children (And Everybody’s)
#spotify_df[spotify_df['album'] == 'Got Live if you want it!']


#show all the songs from December’s Children (And Everybody’s)
#spotify_df[spotify_df['album'] == 'Love You Live (Remastered)']



#show all the songs from December’s Children (And Everybody’s)
#spotify_df[spotify_df['album'] == 'Love You Live (Remastered 2009)']



#show all the songs from Steel Wheels LIve
#spotify_df[spotify_df['album'] == 'Steel Wheels Live']


#show all the songs from Steel Wheels
#spotify_df[spotify_df['album'] == 'Steel Wheels']
```

2. Refine data

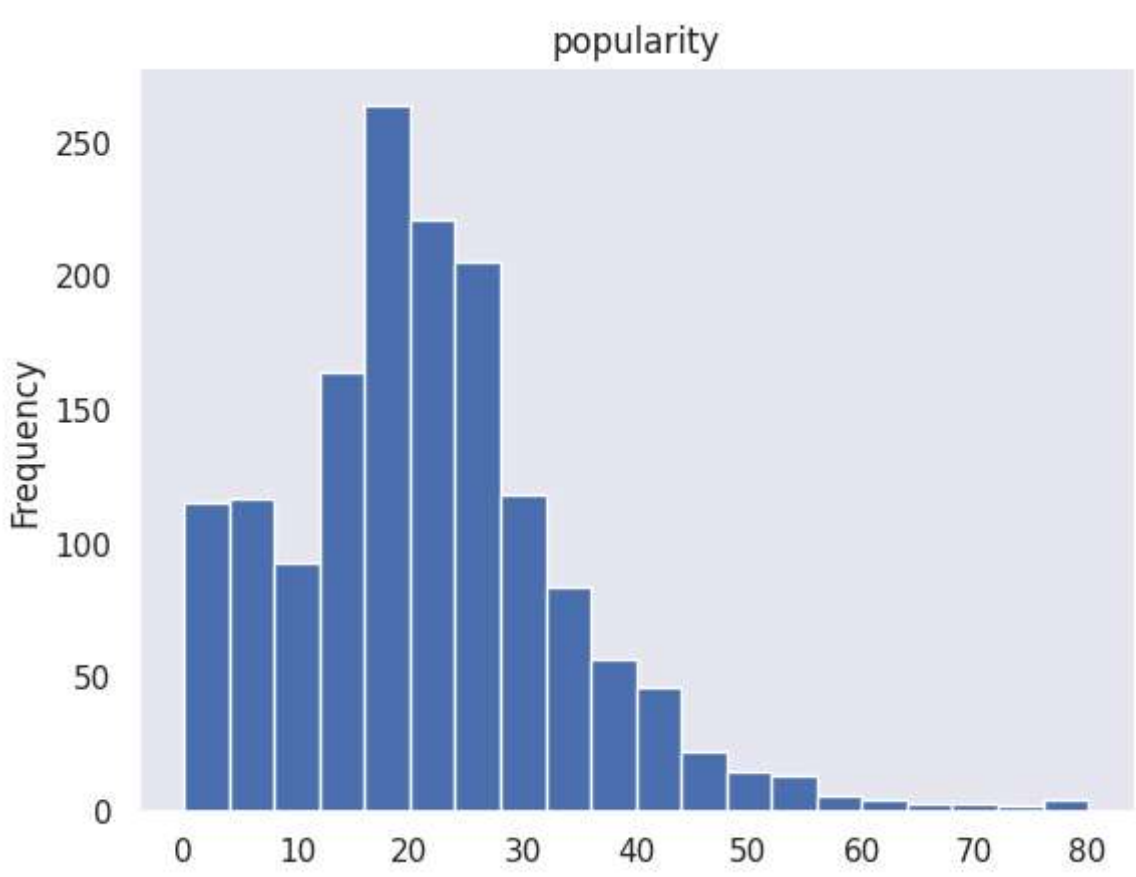
```
# drop all the songs of the album 'Steel Wheel'
spotify_df = spotify_df[spotify_df['album'] != 'Out Of Our Heads (UK)']
spotify_df = spotify_df[spotify_df['album'] != 'Out Of Our Heads (UK Sequence)']
spotify_df = spotify_df[spotify_df['album'] != 'Out Of Our Heads (US Sequence)']
spotify_df = spotify_df[spotify_df['album'] != 'Love You Live (Remastered 2009)']
```

```
# Calculate the number of unique albums in the spotify_df dataset
len(spotify_df['album'].unique())
```

```
86
```

popularity

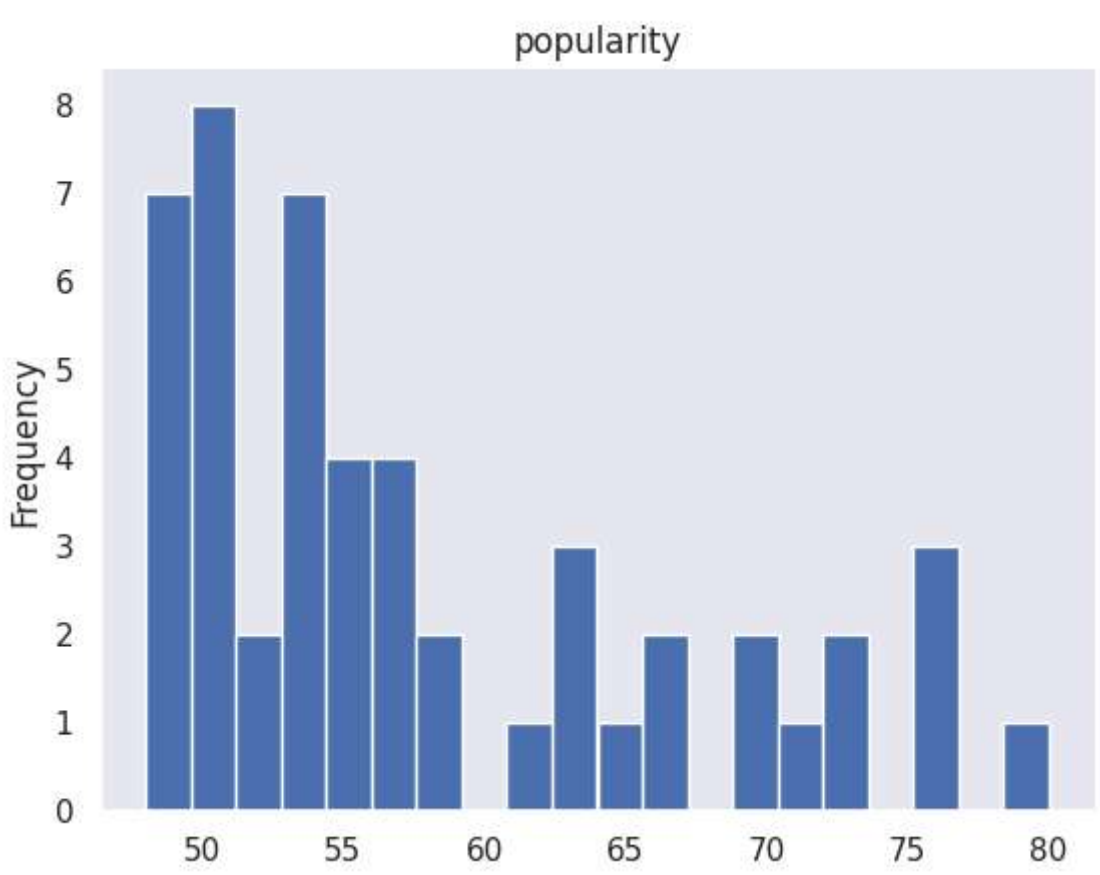
```
# @title popularity

spotify_df['popularity'].plot(kind='hist', bins=20, title='popularity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# create group by popularity ascending order and take the top 50
top_50_popularity = spotify_df.sort_values(by='popularity', ascending=False).head(50)
top_50_popularity.shape
```

```
(50, 18)
```

```
top_50_popularity['popularity'].plot(kind='hist', bins=20, title='popularity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



3.a. Utilize suitable visualizations to identify the two albums that should be recommended to anyone based on the number of popular songs in each album

```
# FInd out which album has the most songs in the list
top_50_popularity_album =top_50_popularity['album'].value_counts()

# Print the top 2 albums with highest popularity
top_50_popularity_album.head(2)
```

album	count
Sticky Fingers (Remastered)	6
Let It Bleed	5

dtype: int64

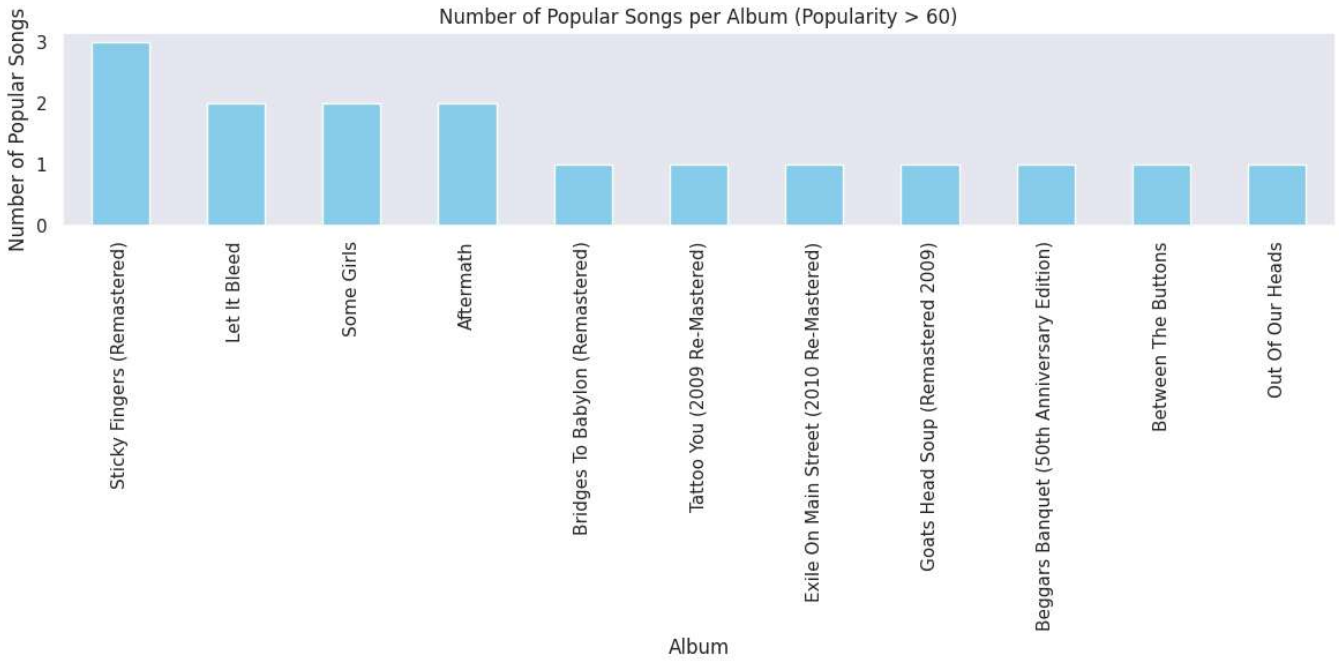
```
top_50_popularity.shape
```

```
(50, 18)
```

3.c Examine the relationship between songs popularity and various factors

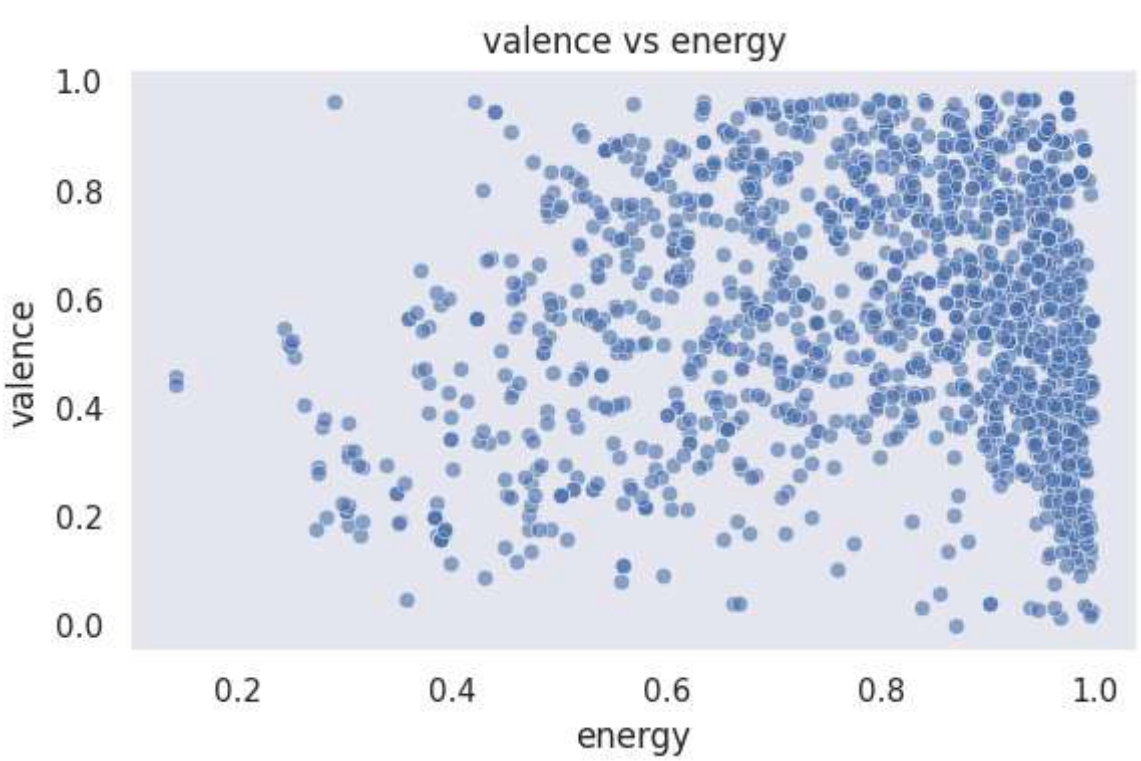
```
# Define popular songs as those with a popularity score above 80
popular_songs = spotify_df[spotify_df['popularity'] > 60]

# Count the number of popular songs per album
popular_counts = popular_songs['album'].value_counts()

# Identify the top 2 albums with the most popular songs
top_2_albums = popular_counts.head(2)

# Plot the number of popular songs per album
plt.figure(figsize=(12, 6))
popular_counts.plot(kind='bar', color='skyblue')
plt.title('Number of Popular Songs per Album (Popularity > 60)')
plt.xlabel('Album')
plt.ylabel('Number of Popular Songs')
plt.xticks(rotation=90)
plt.tight_layout()
plt.savefig("popular_songs_per_album.png")

# Display the top 2 albums
print("Top 2 albums with the most popular songs:")
print(top_2_albums)
```

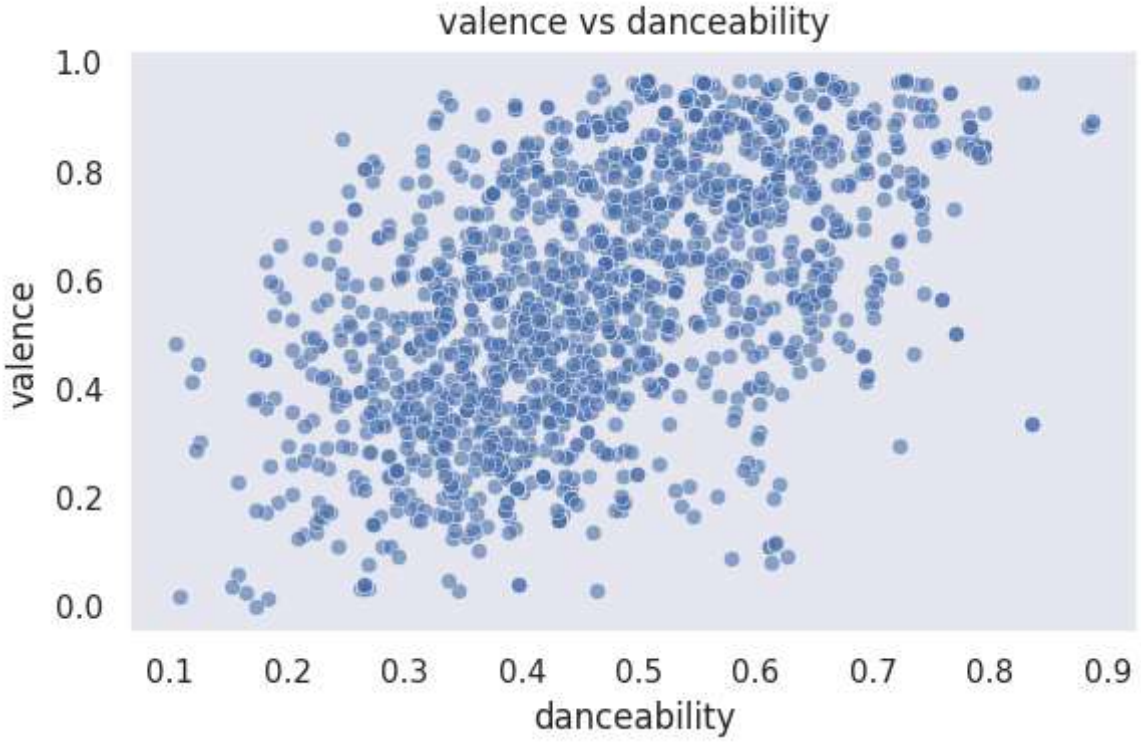
```
Top 2 albums with the most popular songs:
album
Sticky Fingers (Remastered)    3
Let It Bleed                   2
Name: count, dtype: int64
```

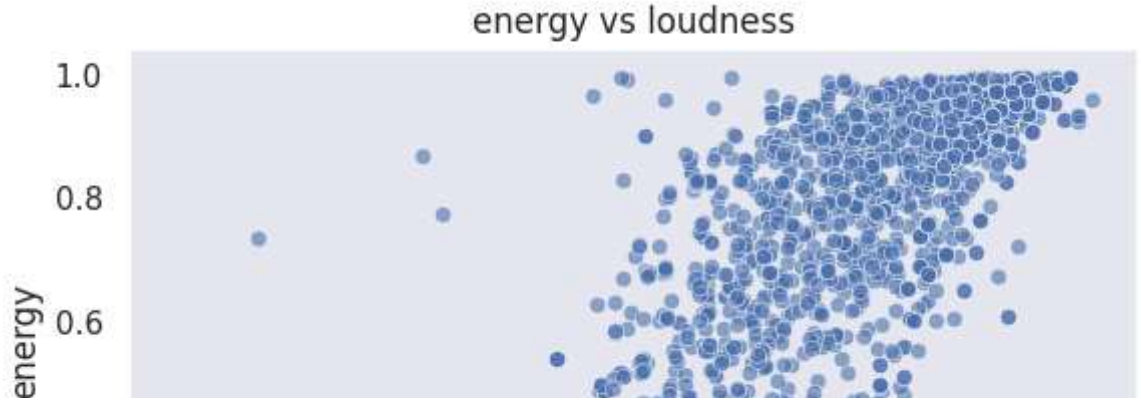


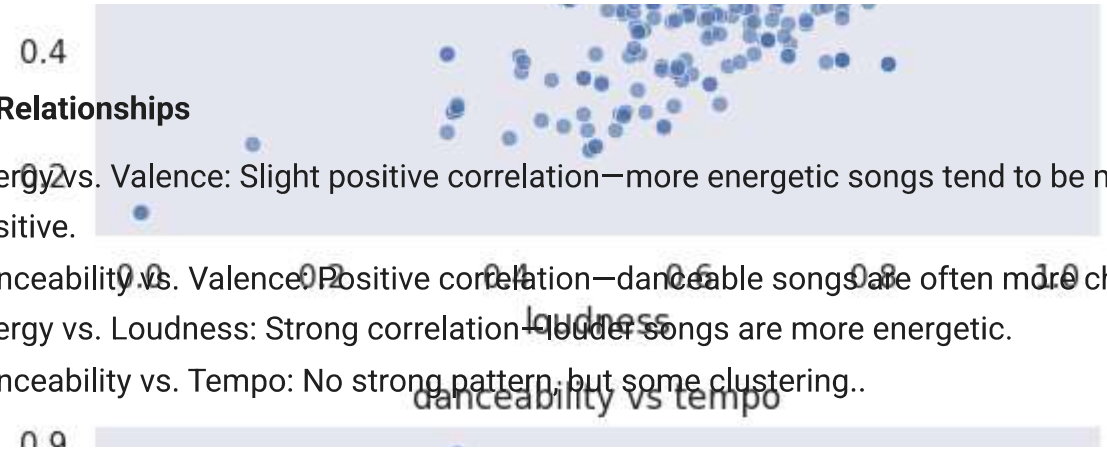

```
#popular_songs


# Scatter plots to explore relationships
scatter_pairs = [
    ('energy', 'valence'),
    ('danceability', 'valence'),
    ('loudness', 'energy'),
    ('tempo', 'danceability')
]
for x, y in scatter_pairs:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(data=spotify_df, x=x, y=y, alpha=0.6)
    plt.title(f'{y} vs {x}')
```

```
        plt.xlabel(x)
        plt.ylabel(y)
        plt.tight_layout()
        plt.savefig(f"{y}_vs_{x}_scatter.png")
```


valence vs energy
valence
energy
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


valence vs danceability
valence
danceability
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

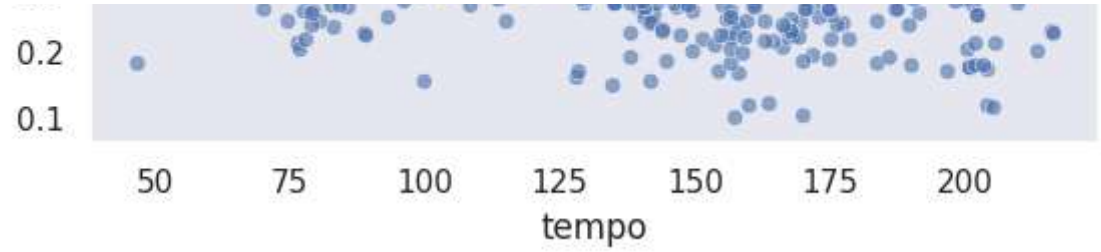

energy vs loudness
energy
1.0
0.8
0.6



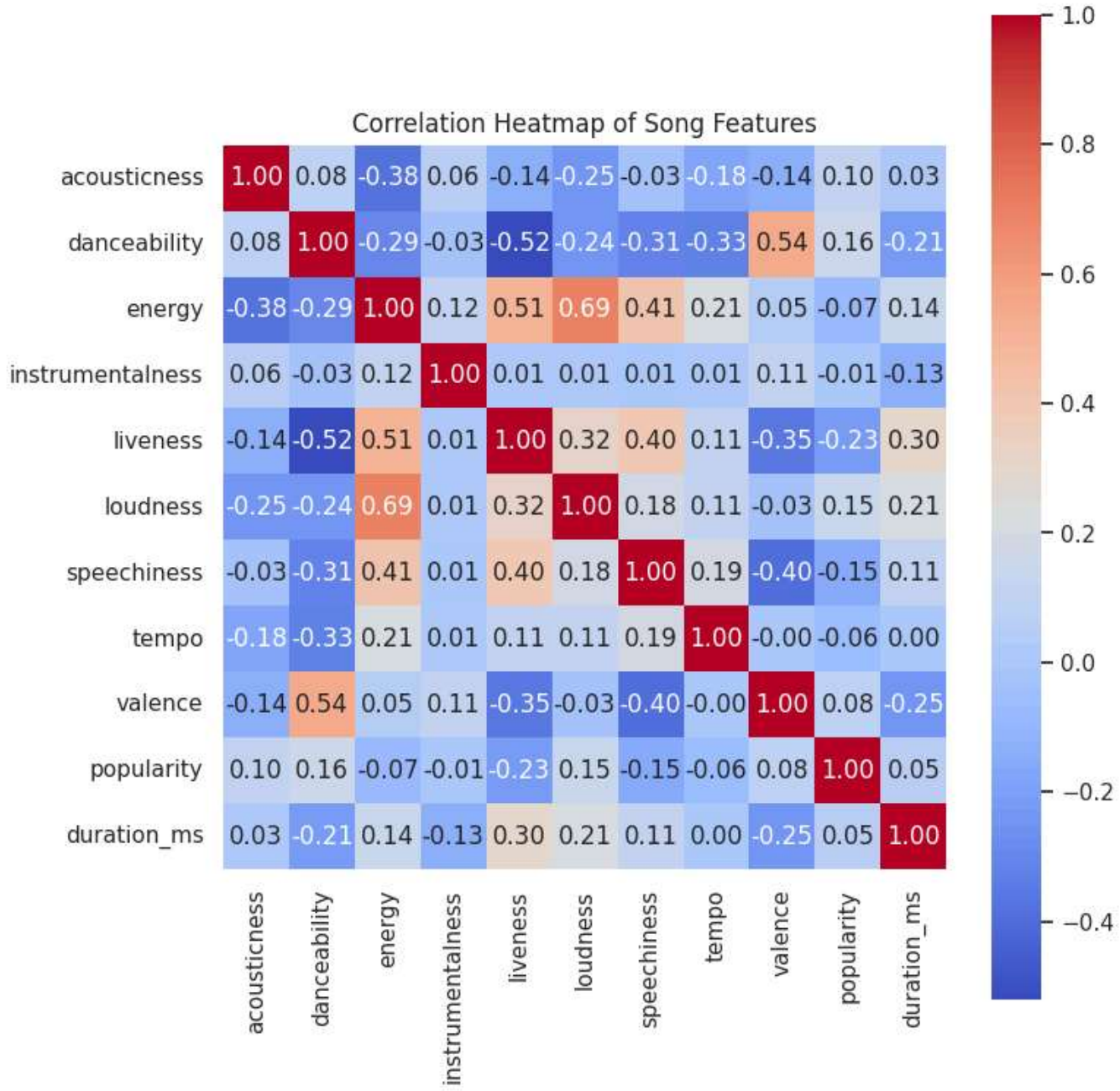
Feature Relationships

- Energy vs. Valence: Slight positive correlation—more energetic songs tend to be more positive.
- Danceability vs. Valence: Positive correlation—danceable songs are often more cheerful.
- Energy vs. Loudness: Strong correlation—louder songs are more energetic.
- Danceability vs. Tempo: No strong pattern, but some clustering..


```
# Correlation heatmap
plt.figure(figsize=(8, 8))
corr_matrix = spotify_df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation Heatmap of Song Features')
plt.tight_layout()
plt.savefig("correlation_heatmap.png")

print("Exploratory data analysis completed. Visualizations saved as PNG files.")
```



```
Exploratory data analysis completed. Visualizations saved as PNG files.
```

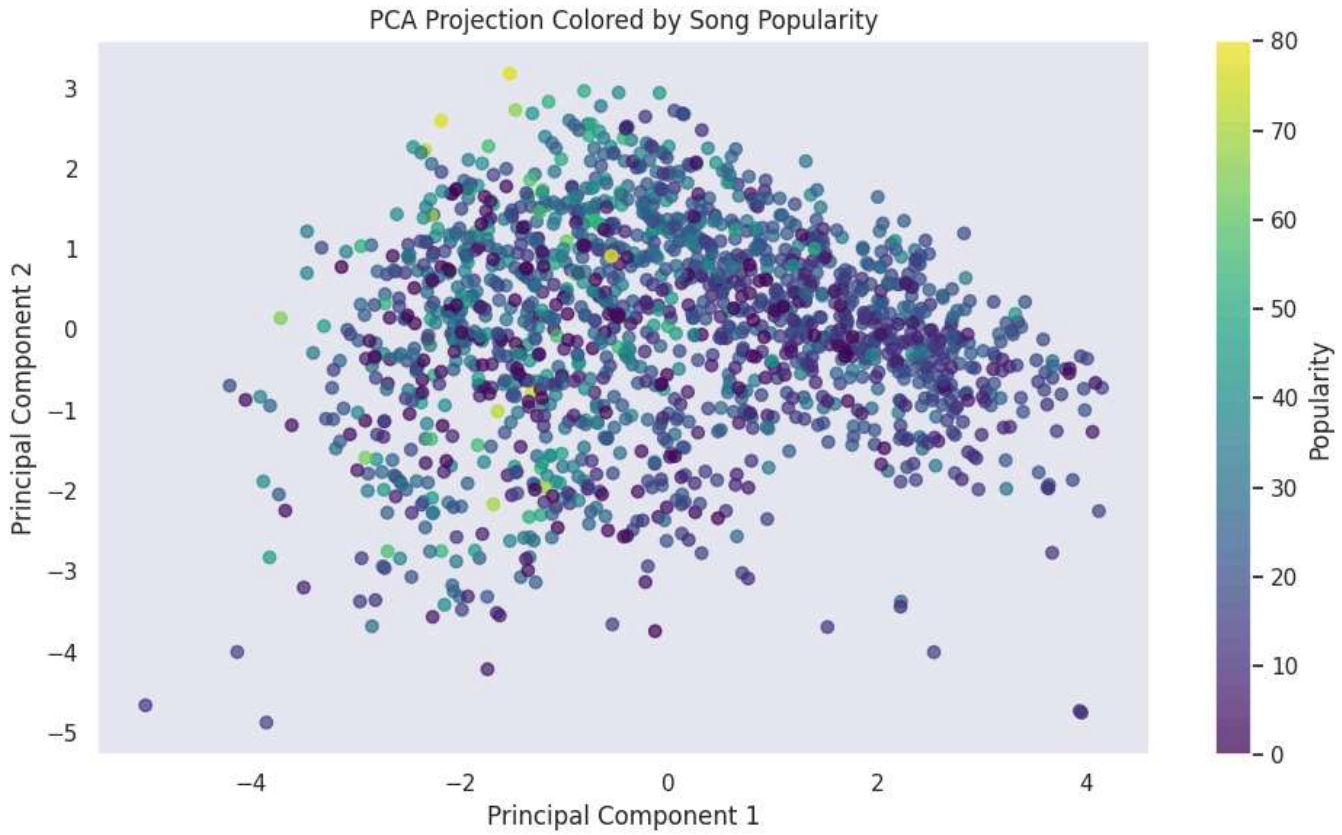


Observations

- The maximum positive correlation coeeficient is 0.69 (loudness vs energy)
- The maximum negative correlation is -0.52 (liveness vs danceability)
- None of the factors a super correlated - CANNOT Drop any factor

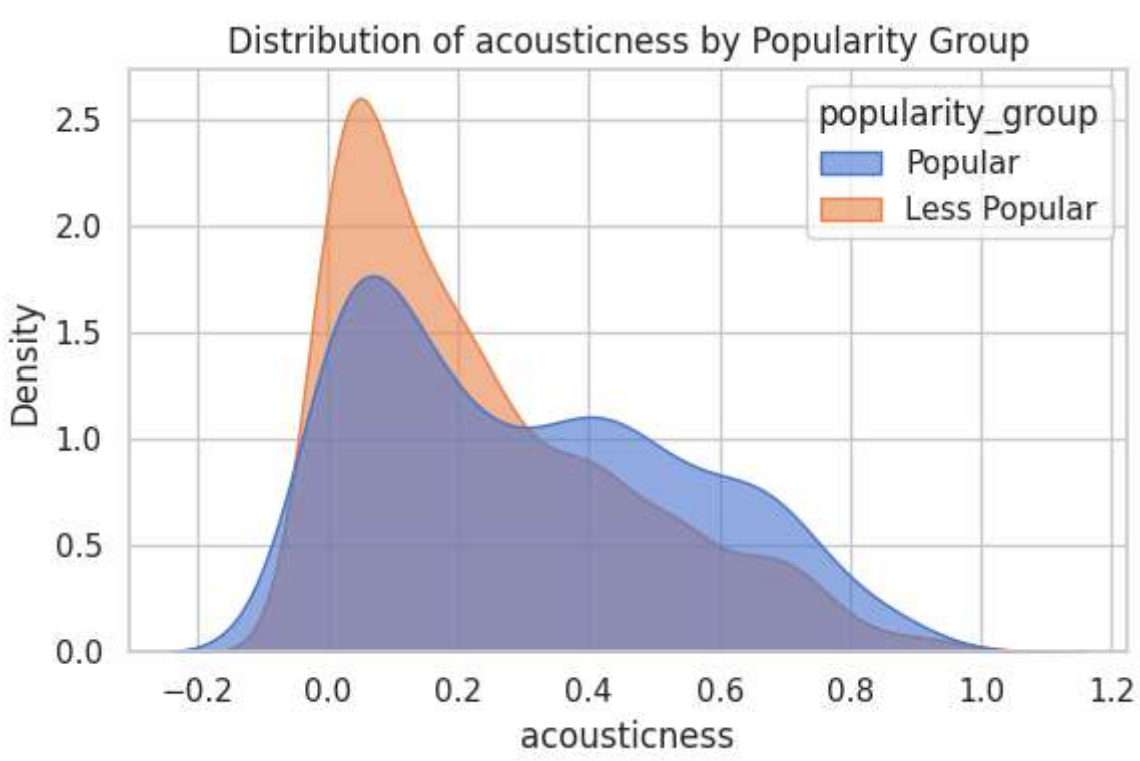
3.d Perform Principal COmponent Analysis

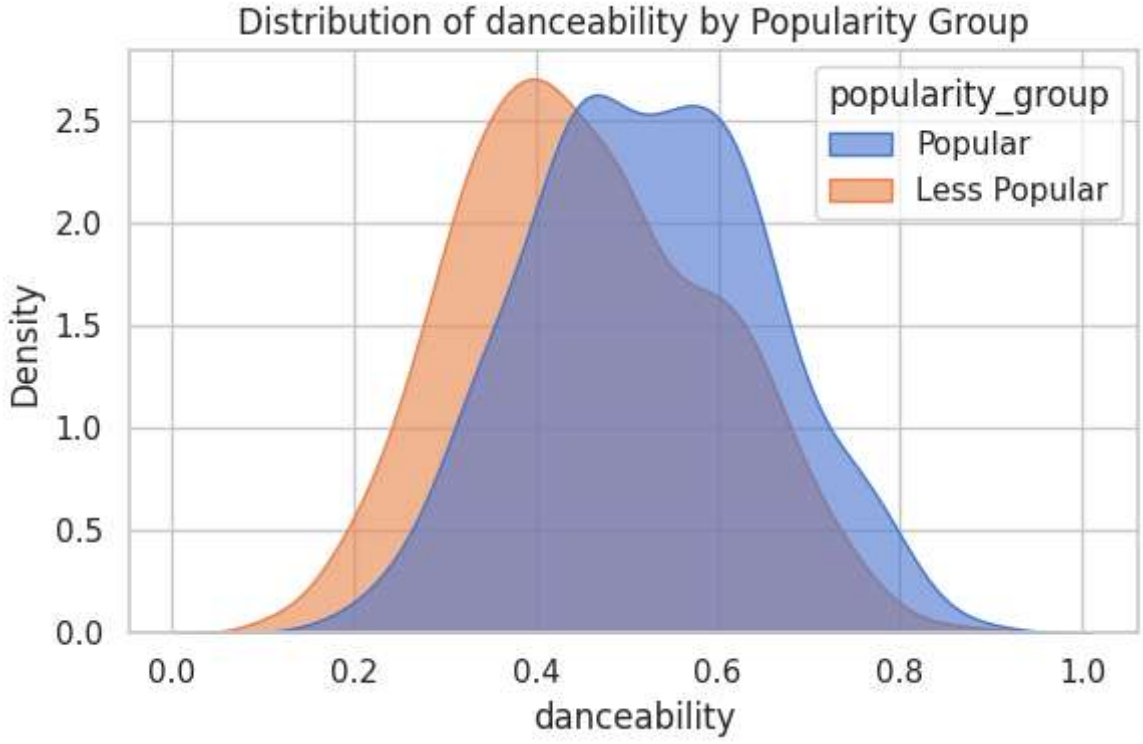
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(spotify_df[features])

# Apply PCA
pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_features)

# Create a DataFrame with PCA components and popularity
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])
pca_df['popularity'] = spotify_df['popularity'].values

# Plot PCA components colored by popularity
plt.figure(figsize=(10, 6))
scatter = plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['popularity'], cmap='viridis',
plt.colorbar(scatter, label='Popularity')
plt.title('PCA Projection Colored by Song Popularity')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.tight_layout()
plt.savefig("pca_popularity_projection.png")
```

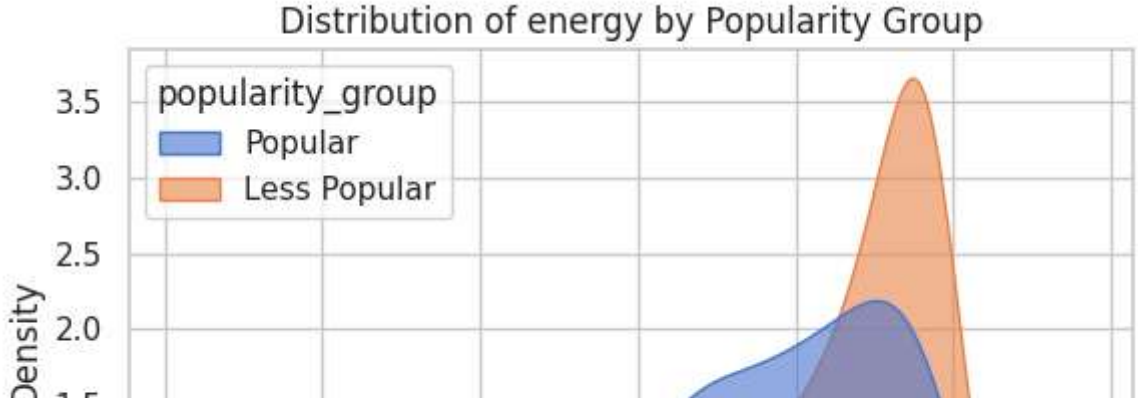


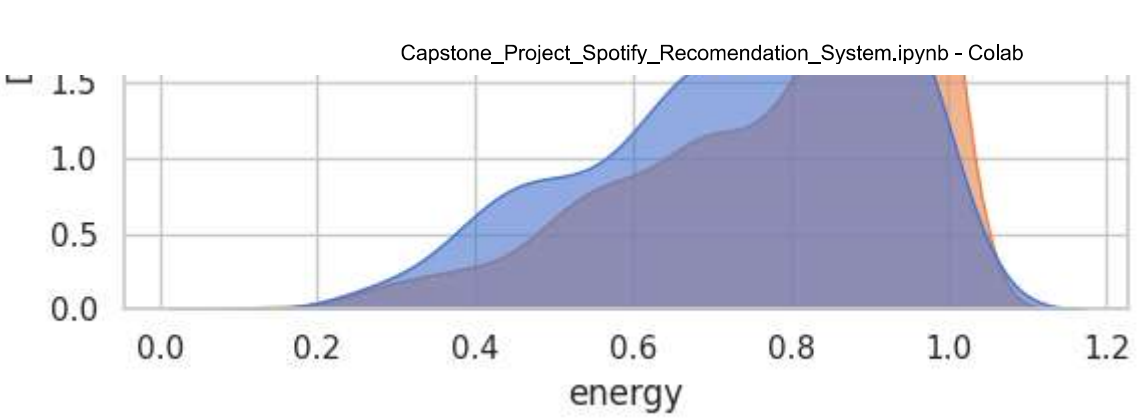

```
# Define popularity threshold
popularity_threshold = 30

# Create a new column to categorize songs
spotify_df['popularity_group'] = spotify_df['popularity'].apply(lambda x: 'Popular' if x >=

# Set seaborn style
sns.set(style="whitegrid")

# Generate KDE plots for each feature
for feature in features:
    plt.figure(figsize=(6, 4))
    sns.kdeplot(data=spotify_df, x=feature, hue='popularity_group', fill=True, common_norm=F
    plt.title(f'Distribution of {feature} by Popularity Group')
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.tight_layout()
```

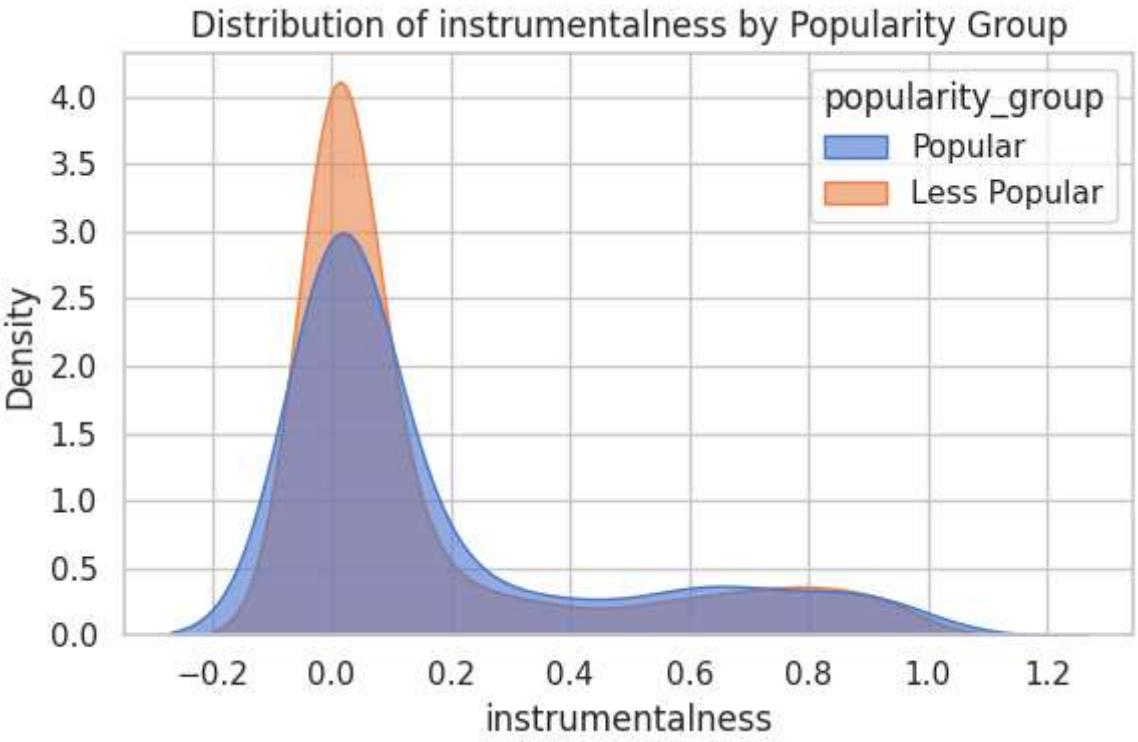
```
        plt.savefig(f"{feature}_popularity_kde.png")
```

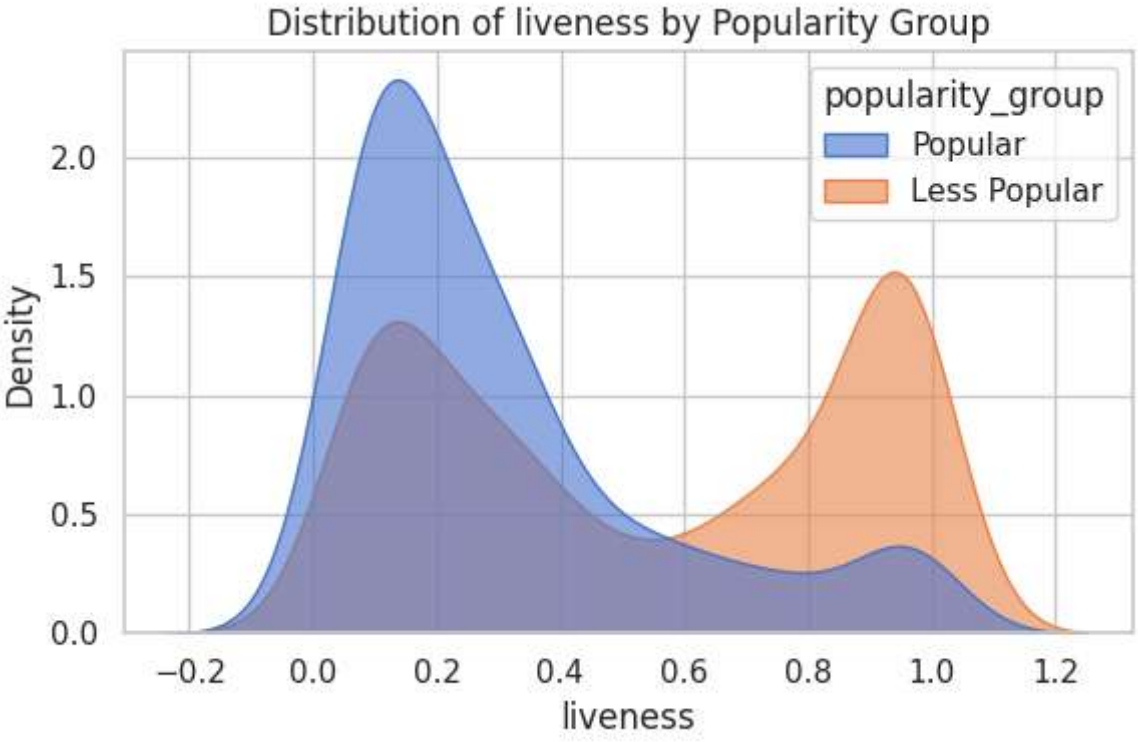

Distribution of acousticness by Popularity Group
popularity_group
Popular
Less Popular
Density
acousticness


Distribution of danceability by Popularity Group
popularity_group
Popular
Less Popular
Density
danceability

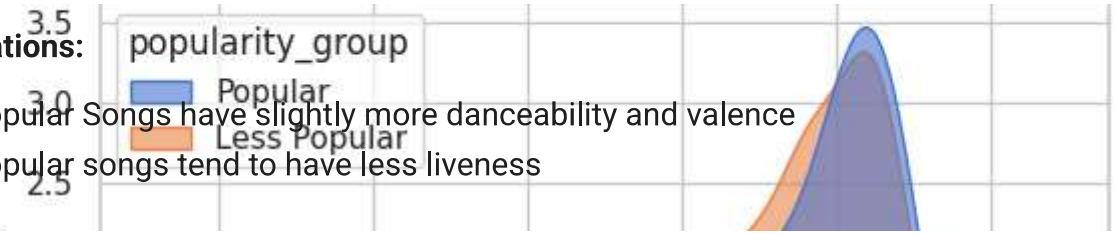

Distribution of energy by Popularity Group
popularity_group
Popular
Less Popular


1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
energy


Distribution of instrumentalness by Popularity Group
popularity_group
Popular
Less Popular
Density
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
instrumentalness


Distribution of liveness by Popularity Group
popularity_group
Popular
Less Popular
Density
2.0
1.5
1.0
0.5
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
liveness

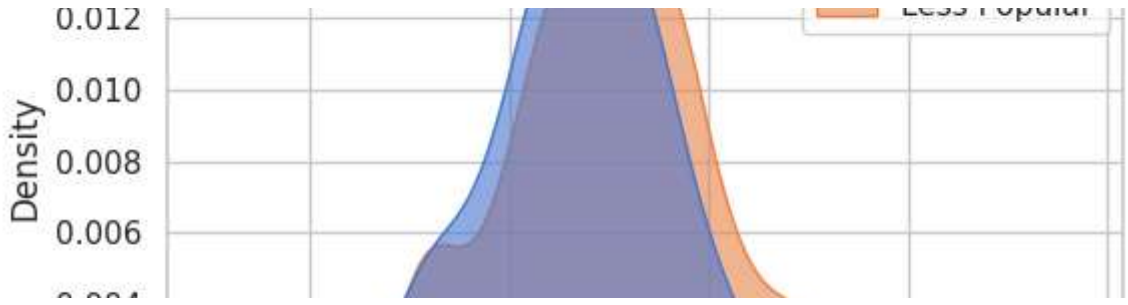

Distribution of loudness by Popularity Group



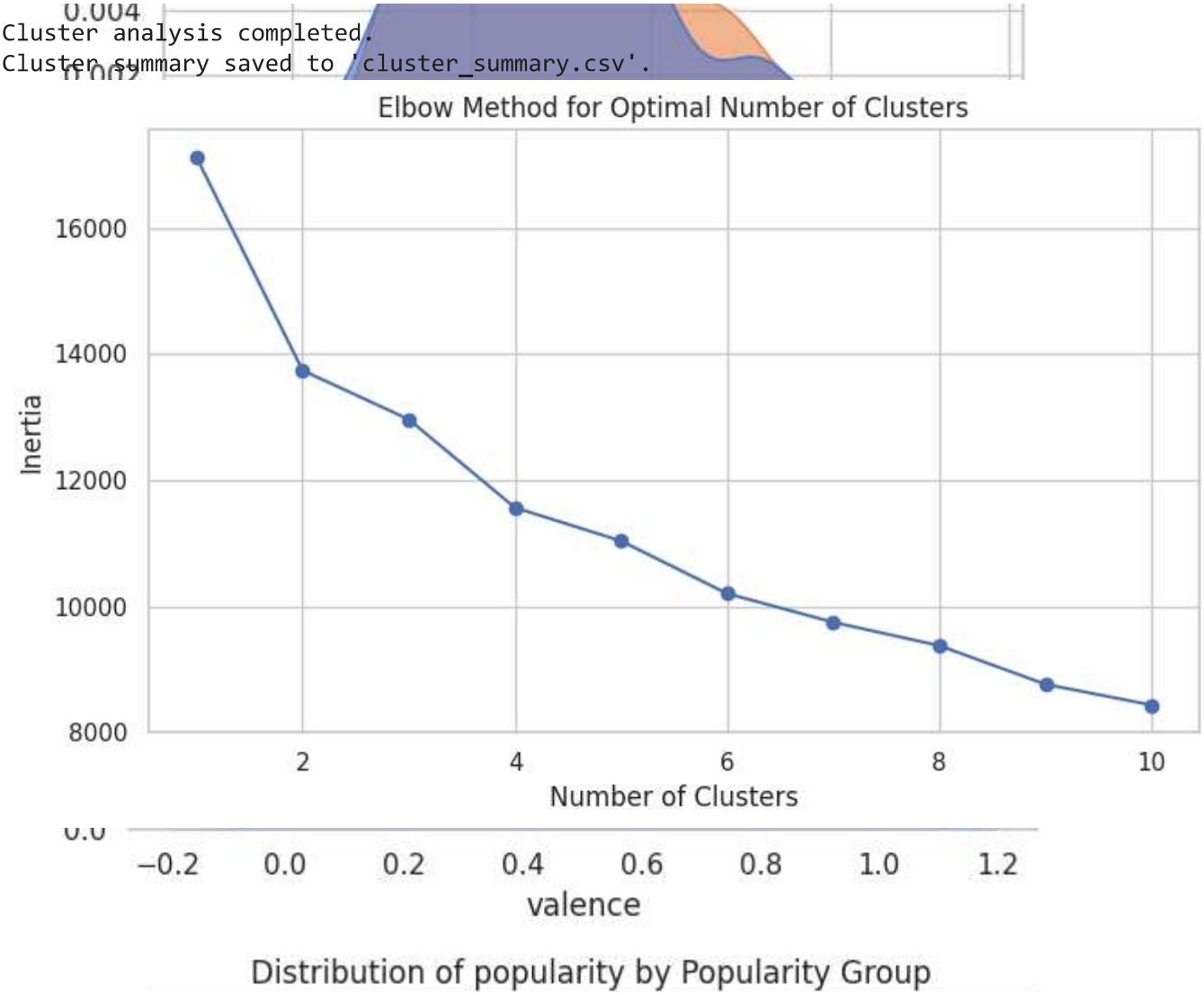
Observations:

- Popular Songs have slightly more danceability and valence
- Popular songs tend to have less liveness

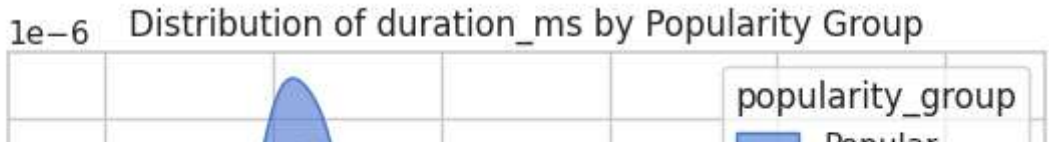
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(spotify_df[features])

# Identify the optimal number of clusters using the elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.tight_layout()
plt.savefig("elbow_curve.png")

# Apply KMeans clustering with optimal number of clusters (e.g., 3 based on elbow curve)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
spotify_df['cluster'] = kmeans.fit_predict(scaled_features)

# Define each cluster based on feature averages
cluster_summary = spotify_df.groupby('cluster')[features].mean()
cluster_summary.to_csv("cluster_summary.csv")

print("Cluster analysis completed.")
print("Cluster summary saved to 'cluster_summary.csv'.")
```

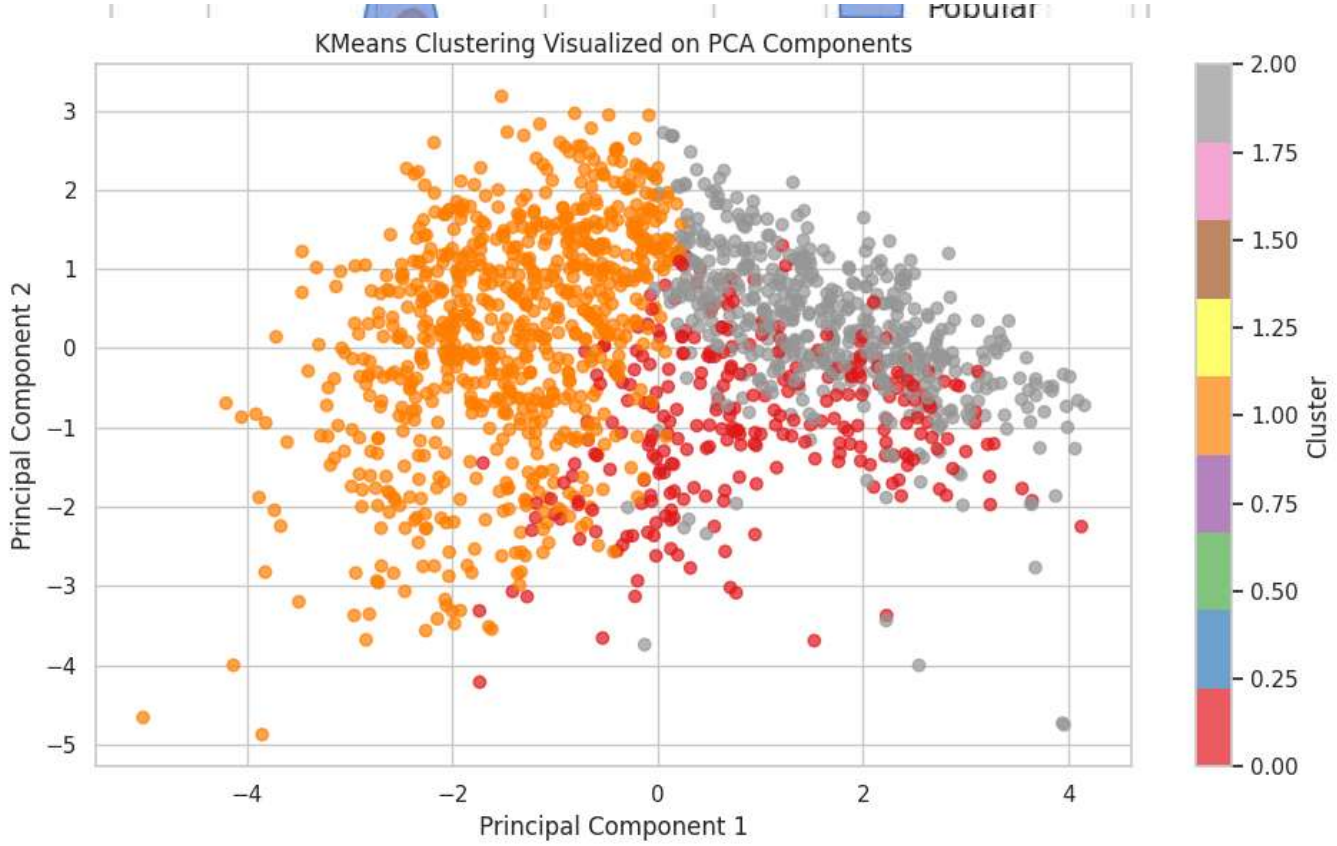


```
Cluster analysis completed.
Cluster summary saved to 'cluster_summary.csv'.
```




```
# Apply KMeans clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(scaled_features)

# Create a DataFrame for visualization
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters

# Plot the clusters
plt.figure(figsize=(10, 6))
scatter = plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['cluster'], cmap='Set1', alpha=
plt.title('KMeans Clustering Visualized on PCA Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(scatter, label='Cluster')
plt.tight_layout()
plt.savefig("cluster_pca_visualization_3.png")
```

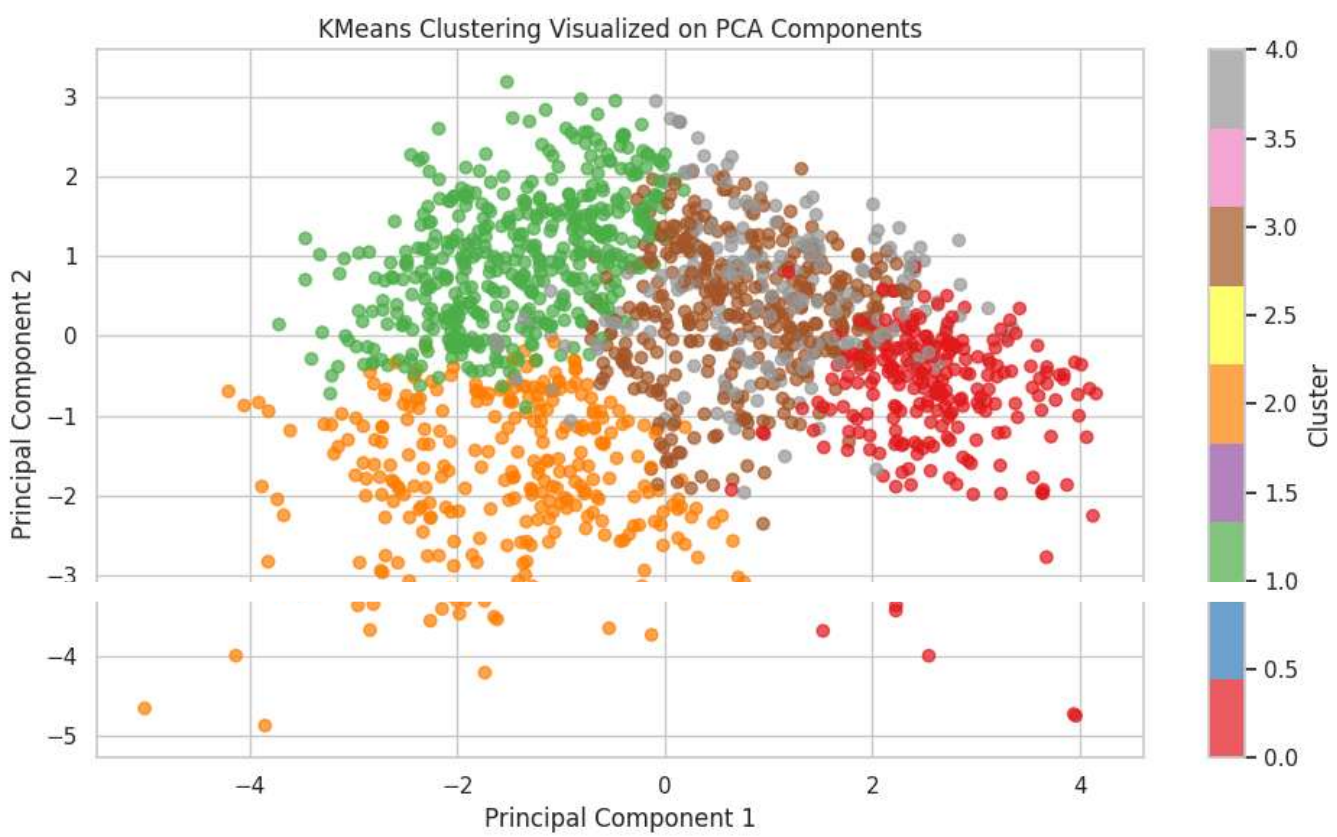




Start coding or generate with AI.

```
# Apply KMeans clustering with 3 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(scaled_features)

# Create a DataFrame for visualization
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])
pca_df['cluster'] = clusters

# Plot the clusters
plt.figure(figsize=(10, 6))
scatter = plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['cluster'], cmap='Set1', alpha=
plt.title('KMeans Clustering Visualized on PCA Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(scatter, label='Cluster')
plt.tight_layout()
```

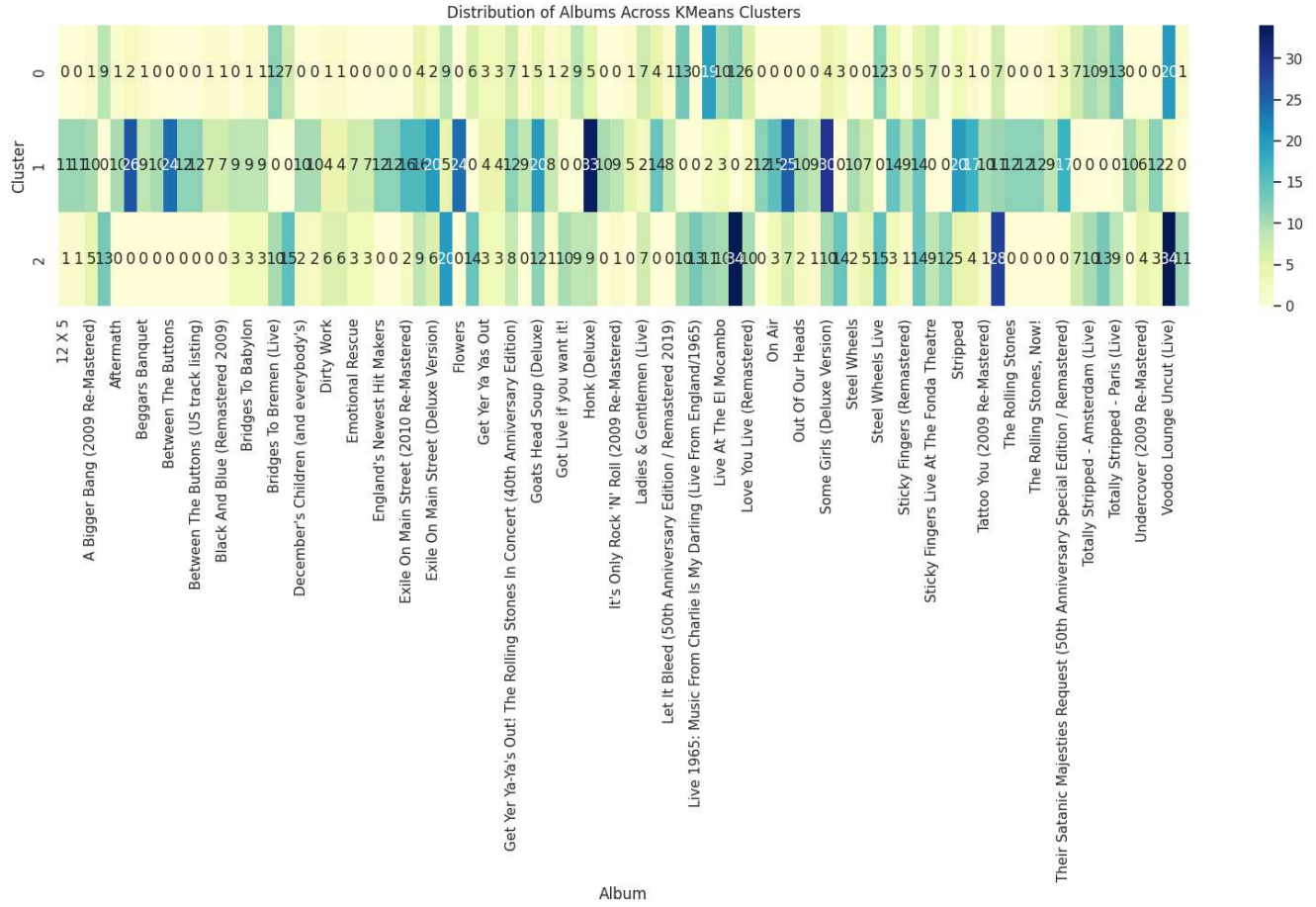
```
plt.savefig("cluster_pca_visualization_5.png")
```




```
# Apply KMeans clustering (assuming 3 clusters based on previous analysis)
kmeans = KMeans(n_clusters=3, random_state=42)
spotify_df['cluster'] = kmeans.fit_predict(scaled_features)

# Group by cluster and album to count songs
cluster_album_counts = spotify_df.groupby(['cluster', 'album']).size().unstack(fill_value=0)

# Plot heatmap of album distribution across clusters
plt.figure(figsize=(20, 4))
sns.heatmap(cluster_album_counts, cmap='YlGnBu', annot=True, fmt='d')
plt.title('Distribution of Albums Across KMeans Clusters')
plt.xlabel('Album')
plt.ylabel('Cluster')
plt.xticks(rotation=90)
plt.tight_layout()
```

```
    plt.savefig("album_cluster_heatmap.png")
```

```
/tmp/ipython-input-3466032068.py:15: UserWarning: Tight layout not applied. The bottom
  plt.tight_layout()
```



```
# Apply KMeans clustering (assuming 3 clusters based on previous analysis)
kmeans = KMeans(n_clusters=3, random_state=42)
spotify_df['cluster'] = kmeans.fit_predict(scaled_features)

# Compute average popularity per cluster
popularity_by_cluster = spotify_df.groupby('cluster')['popularity'].mean()

# Plot the average popularity per cluster
plt.figure(figsize=(8, 5))
popularity_by_cluster.plot(kind='bar', color='coral')
plt.title('Average Popularity per KMeans Cluster')
plt.xlabel('Cluster')
plt.ylabel('Average Popularity')
plt.xticks(rotation=0)
plt.tight_layout()
plt.savefig("cluster_popularity_bar.png")

# Display the average popularity values
print("Average popularity per cluster:")
print(popularity_by_cluster)
```

```
Average popularity per cluster:
cluster
0    18.766187
```