stone

March 27, 2025

[1]:
```
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
```

[2]:
```
stones = pd.read_csv('rolling_stones_spotify.csv')
```

[3]:
```
stones.head()
```

[3]:

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

	acousticness	danceability	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	duration_ms
0	0.0824	0.463	0.993	0.996000	0.932	-12.913	0.1100	118.001	0.0302	33	48640
1	0.4370	0.326	0.965	0.233000	0.961	-4.803	0.0759	131.455	0.3180	34	253173
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					

```
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
```

[4]:
```
stones.tail()
```

[4]:
```
      Unnamed: 0                        name                album release_date  \
1605        1605                       Carol  The Rolling Stones   1964-04-16
1606        1606                    Tell Me  The Rolling Stones   1964-04-16
1607        1607          Can I Get A Witness  The Rolling Stones   1964-04-16
1608        1608  You Can Make It If You Try  The Rolling Stones   1964-04-16
1609        1609              Walking The Dog  The Rolling Stones   1964-04-16

      track_number                      id  \
1605             8  0817M5UpRnffGlOFyuRiQZ
1606             9  3JZllQBsTM6WwoJdzFDLhx
1607            10  0t2qvfSBQ3YO8lzRRoVTdb
1608            11  5ivIs5vwSjORChOIvlY3On
1609            12  43SkTJJ2xleDaeiE4TIM70

                                    uri  acousticness  danceability  \
1605  spotify:track:0817M5UpRnffGlOFyuRiQZ        0.1570         0.466
1606  spotify:track:3JZllQBsTM6WwoJdzFDLhx        0.0576         0.509
1607  spotify:track:0t2qvfSBQ3YO8lzRRoVTdb        0.3710         0.790
1608  spotify:track:5ivIs5vwSjORChOIvlY3On        0.2170         0.700
1609  spotify:track:43SkTJJ2xleDaeiE4TIM70        0.3830         0.727

      energy  instrumentalness  liveness  loudness  speechiness    tempo  \
1605   0.932          0.006170    0.3240    -9.214       0.0429  177.340
1606   0.706          0.000002    0.5160    -9.427       0.0843  122.015
1607   0.774          0.000000    0.0669    -7.961       0.0720   97.035
1608   0.546          0.000070    0.1660    -9.567       0.0622  102.634
1609   0.934          0.068500    0.0965    -8.373       0.0359  125.275

      valence  popularity  duration_ms
1605    0.967          39       154080
1606    0.446          36       245266
1607    0.835          30       176080
1608    0.532          27       121680
1609    0.969          35       189186
```

[5]:
```
stones.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
```

[6]:
```
stones.describe()
```

[6]:

	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
```

[7]:
```
stones.dtypes
```

[7]:
```
Unnamed: 0            int64
name                 object
album                object
release_date         object
track_number          int64
id                   object
uri                  object
acousticness        float64
danceability        float64
energy              float64
instrumentalness    float64
liveness            float64
loudness            float64
speechiness         float64
tempo               float64
valence             float64
popularity            int64
duration_ms           int64
dtype: object
```

[8]:
```
# CHECKING MISSING VALUES
```

[9]:
```
missing_values = stones.isnull().sum()
print("Missing Values per Column:")
print(missing_values)
```

```
Missing Values per Column:
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
```

[10]:
```
stones_no_duplicates = stones.drop_duplicates()
```

[11]:
```
stones.drop_duplicates()
```

[11]:

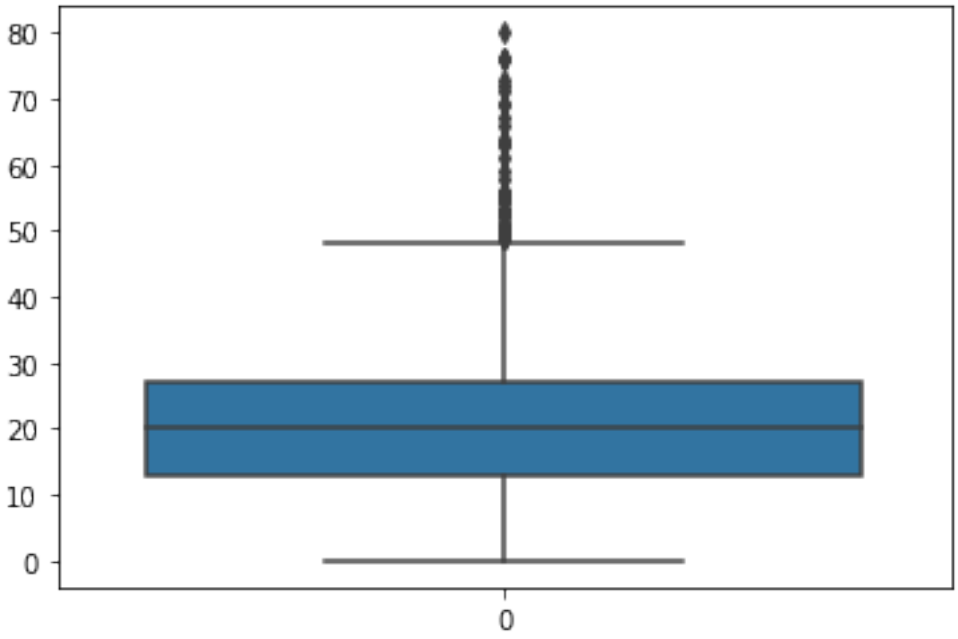
	Unnamed: 0	name	album	\
0	0	Concert Intro Music - Live	Licked Live In NYC	
1	1	Street Fighting Man - Live	Licked Live In NYC	
2	2	Start Me Up - Live	Licked Live In NYC	
3	3	If You Can't Rock Me - Live	Licked Live In NYC	
4	4	Don’t Stop - Live	Licked Live In NYC	
…	…	…	…	
1605	1605	Carol	The Rolling Stones	
1606	1606	Tell Me	The Rolling Stones	
1607	1607	Can I Get A Witness	The Rolling Stones	
1608	1608	You Can Make It If You Try	The Rolling Stones	
1609	1609	Walking The Dog	The Rolling Stones	

	release_date	track_number	id	\
0	2022-06-10	1	2IEkywLJ4ykbhi1yRQvmsT	
1	2022-06-10	2	6GVgVJBKkGJoRfarYRvGTU	
2	2022-06-10	3	1Lu761pZ0dBTGpzxaQoZNW	
3	2022-06-10	4	1agTQzOTUnGNggyckEqiDH	
4	2022-06-10	5	7piGJR8YndQBQWVXv6KtQw	
…	…	…	…	
1605	1964-04-16	8	08l7M5UpRnffGl0FyuRiQZ	
1606	1964-04-16	9	3JZllQBsTM6WwoJdzFDLhx	
1607	1964-04-16	10	0t2qvfSBQ3Y08lzRRoVTdb	
1608	1964-04-16	11	5ivIs5vwSj0RChOIvlY3On	
1609	1964-04-16	12	43SkTJJ2xleDaeiE4TIM70	

	uri	acousticness	danceability	\
0	spotify:track:2IEkywLJ4ykbhi1yRQvmsT	0.0824	0.463	
1	spotify:track:6GVgVJBKkGJoRfarYRvGTU	0.4370	0.326	
2	spotify:track:1Lu761pZ0dBTGpzxaQoZNW	0.4160	0.386	
3	spotify:track:1agTQzOTUnGNggyckEqiDH	0.5670	0.369	
4	spotify:track:7piGJR8YndQBQWVXv6KtQw	0.4000	0.303	
…	…	…	…	
1605	spotify:track:08l7M5UpRnffGl0FyuRiQZ	0.1570	0.466	

```
1606  spotify:track:3JZllQBsTM6WwoJdzFDLhx        0.0576         0.509
1607  spotify:track:0t2qvfSBQ3YO8lzRRoVTdb        0.3710         0.790
1608  spotify:track:5ivIs5vwSjORChOIvlY3On        0.2170         0.700
1609  spotify:track:43SkTJJ2xleDaeiE4TIM70        0.3830         0.727

      energy  instrumentalness  liveness  loudness  speechiness    tempo  \
0      0.993          0.996000    0.9320   -12.913       0.1100  118.001
1      0.965          0.233000    0.9610    -4.803       0.0759  131.455
2      0.969          0.400000    0.9560    -4.936       0.1150  130.066
3      0.985          0.000107    0.8950    -5.535       0.1930  132.994
4      0.969          0.055900    0.9660    -5.098       0.0930  130.533
…      …                …       …         …          …       …
1605   0.932          0.006170    0.3240    -9.214       0.0429  177.340
1606   0.706          0.000002    0.5160    -9.427       0.0843  122.015
1607   0.774          0.000000    0.0669    -7.961       0.0720   97.035
1608   0.546          0.000070    0.1660    -9.567       0.0622  102.634
1609   0.934          0.068500    0.0965    -8.373       0.0359  125.275

      valence  popularity  duration_ms
0      0.0302          33        48640
1      0.3180          34       253173
2      0.3130          34       263160
3      0.1470          32       305880
4      0.2060          32       305106
…       …          …           …
1605   0.9670          39       154080
1606   0.4460          36       245266
1607   0.8350          30       176080
1608   0.5320          27       121680
1609   0.9690          35       189186

[1610 rows x 18 columns]
```

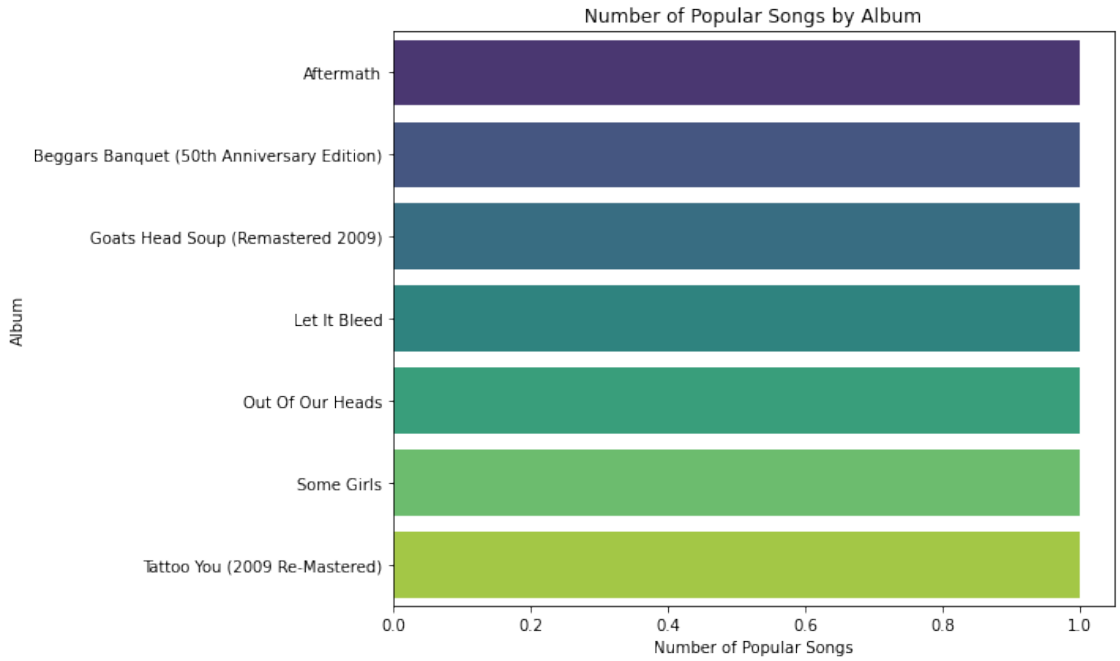
[12]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(stones['popularity'])
plt.show()
```



[ ]:

[13]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define popularity threshold (e.g., 70)
popularity_threshold = 70

# Filter dataset to keep only popular songs
stones_popular_songs = stones[stones['popularity'] >= popularity_threshold]

# Group by album and count the number of popular songs
album_popularity = stones_popular_songs.groupby('album')['track_number'].
 ↪count().reset_index()

# Sort albums by number of popular songs in descending order
album_popularity = album_popularity.sort_values(by='track_number',␣
 ↪ascending=False)

# Plot a bar chart to visualize the top albums based on popular songs
plt.figure(figsize=(10, 6))
sns.barplot(x='track_number', y='album', data=album_popularity,␣
 ↪palette='viridis')
```

```
plt.title('Number of Popular Songs by Album')
plt.xlabel('Number of Popular Songs')
plt.ylabel('Album')

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



[ ]:

[14]:
```
data = {
    'album': ['Licked live in NYC', 'Tattoo You', 'A Bigger Bang-live ', 'Steel␣
 ↪Wheels Live' , 'Licked live in NYC', 'Tattoo You', 'A Bigger Bang-live'],
    'song': ['Dont stop-live', 'Start me up', 'Happy-live', 'Terrifing-live',␣
 ↪'Let it Bleed-Live', 'No Use in Crying ' , 'Sympathy for the devil-live'],
    'duration_ms': [305106, 214173, 234733, 299373, 313586, 206533, 481720],
    'popularity': [32, 12, 15, 18, 30, 9, 24],
    'release_year': [2022, 2021, 2021, 2020, 2022, 2021, 2021]
}
stones = pd.DataFrame(data)

# Basic information about the dataset
stones.info()

# Show summary statistics
stones.describe()
```

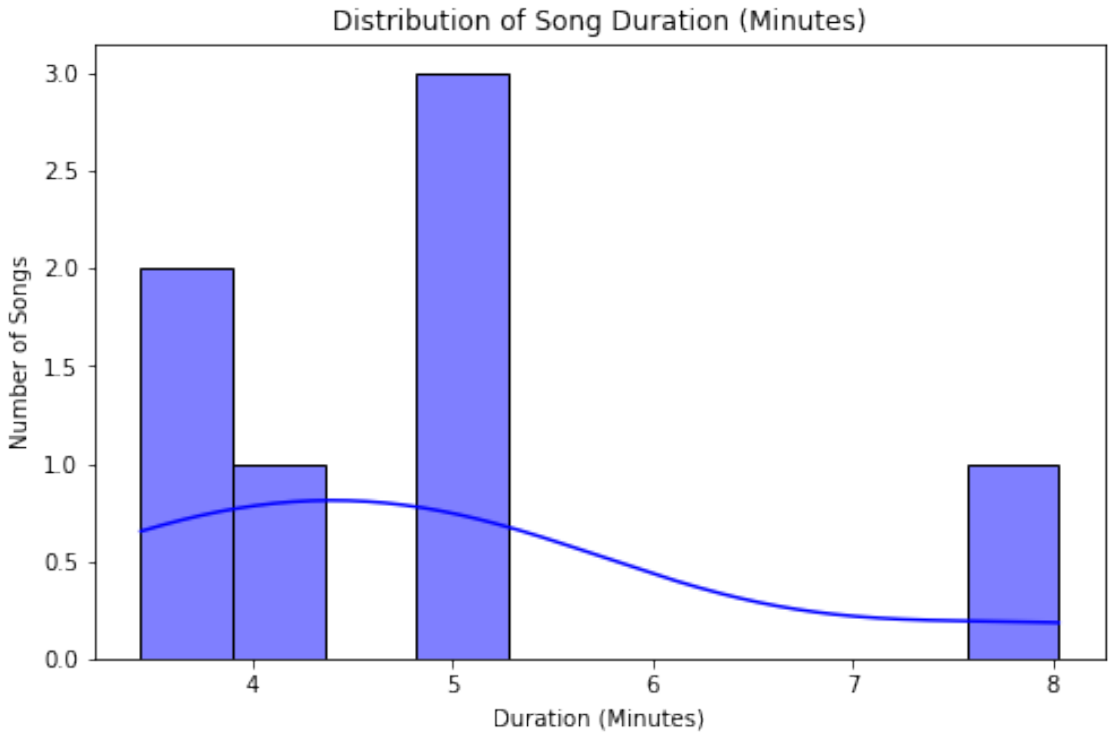
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   album         7 non-null      object
 1   song          7 non-null      object
 2   duration_ms   7 non-null      int64 
 3   popularity    7 non-null      int64 
 4   release_year  7 non-null      int64 
dtypes: int64(3), object(2)
memory usage: 408.0+ bytes
```

[14]:

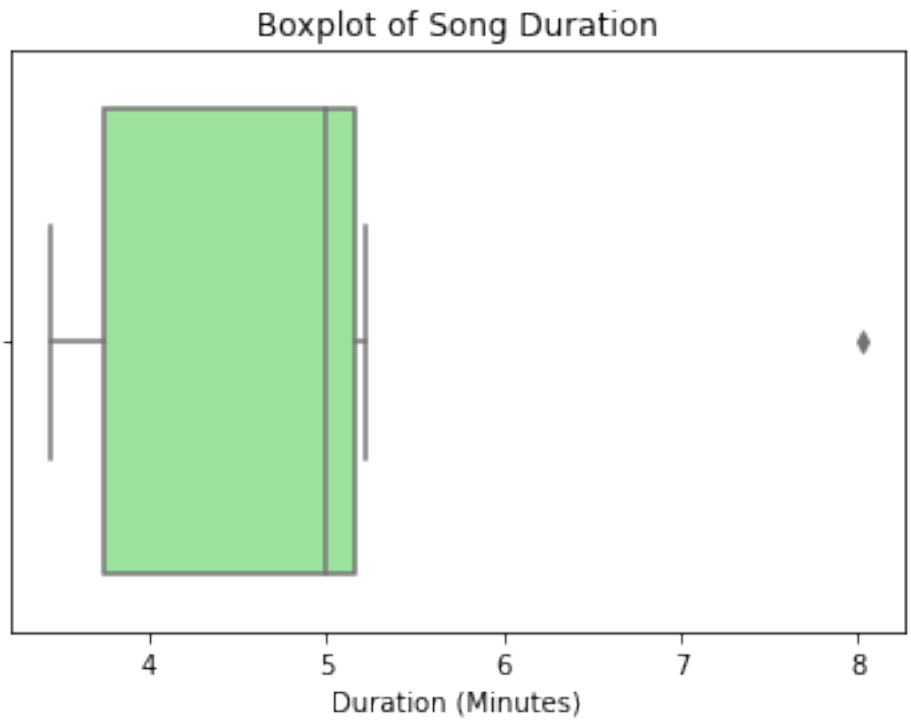
	duration_ms	popularity	release_year
count	7.000000	7.000000	7.000000
mean	293603.428571	20.000000	2021.142857
std	94259.327200	8.888194	0.690066
min	206533.000000	9.000000	2020.000000
25%	224453.000000	13.500000	2021.000000
50%	299373.000000	18.000000	2021.000000
75%	309346.000000	27.000000	2021.500000
max	481720.000000	32.000000	2022.000000

[15]:
```
# Song Duration Analysis
```

[16]:
```
# Convert duration from milliseconds to minutes
stones['duration_min'] = stones['duration_ms'] / 60000

# Histogram of song durations
plt.figure(figsize=(8, 5))
sns.histplot(stones['duration_min'], bins=10, kde=True, color='blue')
plt.title('Distribution of Song Duration (Minutes)')
plt.xlabel('Duration (Minutes)')
plt.ylabel('Number of Songs')
plt.show()
```

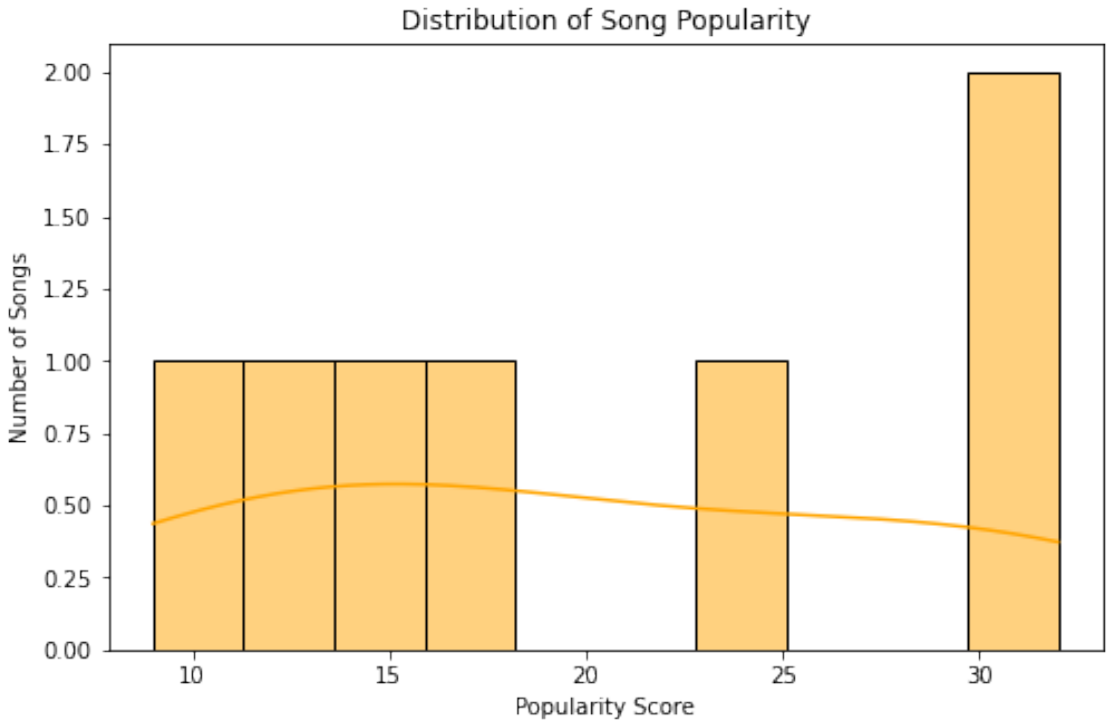


[17]:
```
# Boxplot to check for outliers
plt.figure(figsize=(6, 4))
sns.boxplot(x=stones['duration_min'], color='lightgreen')
plt.title('Boxplot of Song Duration')
plt.xlabel('Duration (Minutes)')
plt.show()
```



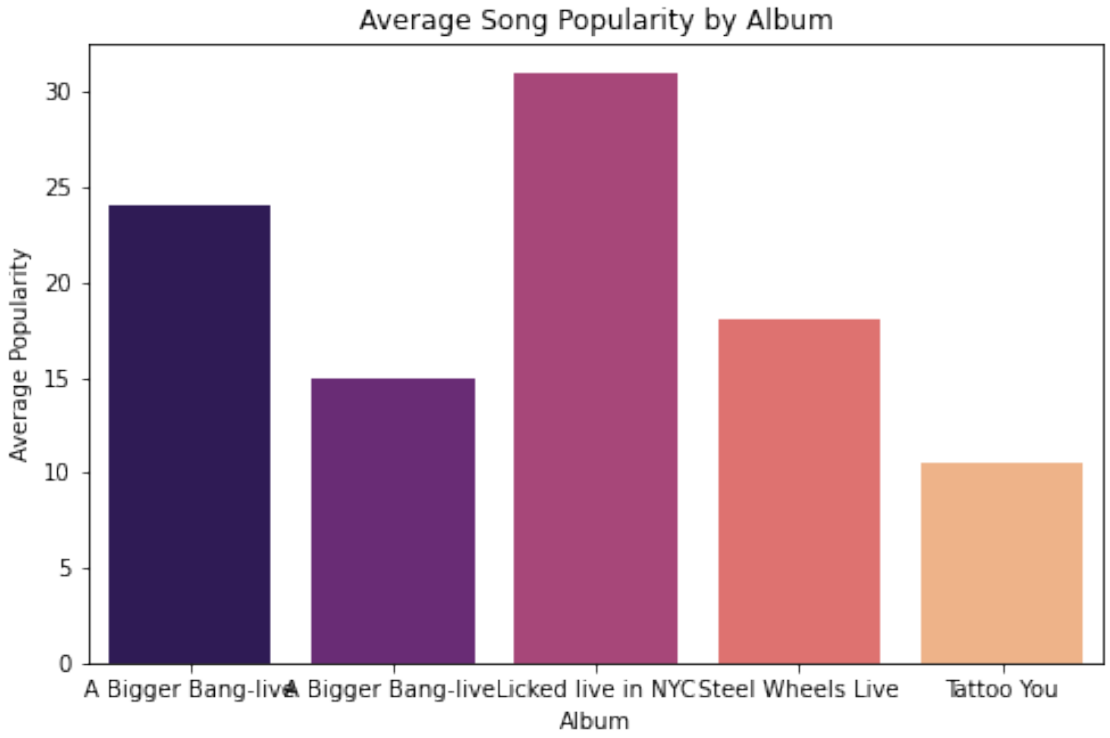
[18]:
```
#  Popularity Analysis
```

[19]:
```
# Histogram of song popularity
plt.figure(figsize=(8, 5))
sns.histplot(stones['popularity'], bins=10, kde=True, color='orange')
plt.title('Distribution of Song Popularity')
plt.xlabel('Popularity Score')
plt.ylabel('Number of Songs')
plt.show()
```



[20]:
```
# Average popularity per album
avg_popularity_by_album = stones.groupby('album')['popularity'].mean().
 ↪reset_index()
```

[21]:
```
# Bar chart for average popularity per album
plt.figure(figsize=(8, 5))
sns.barplot(x='album', y='popularity', data=avg_popularity_by_album,␣
 ↪palette='magma')
plt.title('Average Song Popularity by Album')
plt.xlabel('Album')
plt.ylabel('Average Popularity')
plt.show()
```



[22]:
```
# Top 10 most popular songs
top_10_songs = stones.sort_values(by='popularity', ascending=False).head(10)
print("Top 10 Most Popular Songs:\n", top_10_songs[['song', 'popularity']])
```

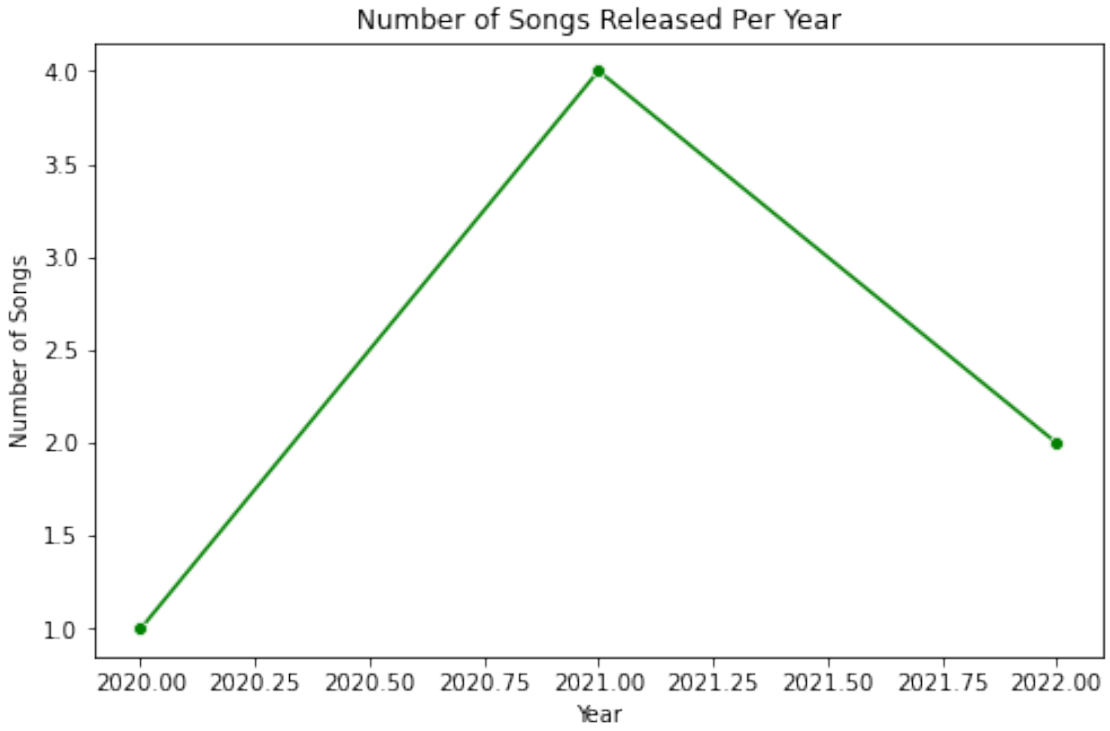
```
Top 10 Most Popular Songs:
                          song  popularity
0               Dont stop-live          32
4           Let it Bleed-Live           30
6  Sympathy for the devil-live          24
3               Terrifing-live          18
2                   Happy-live          15
1                  Start me up          12
5           No Use in Crying             9
```

[23]:
```
# Release Year Analysis
```

[24]:
```
# Count of songs released by year
songs_per_year = stones.groupby('release_year')['song'].count().reset_index()
```

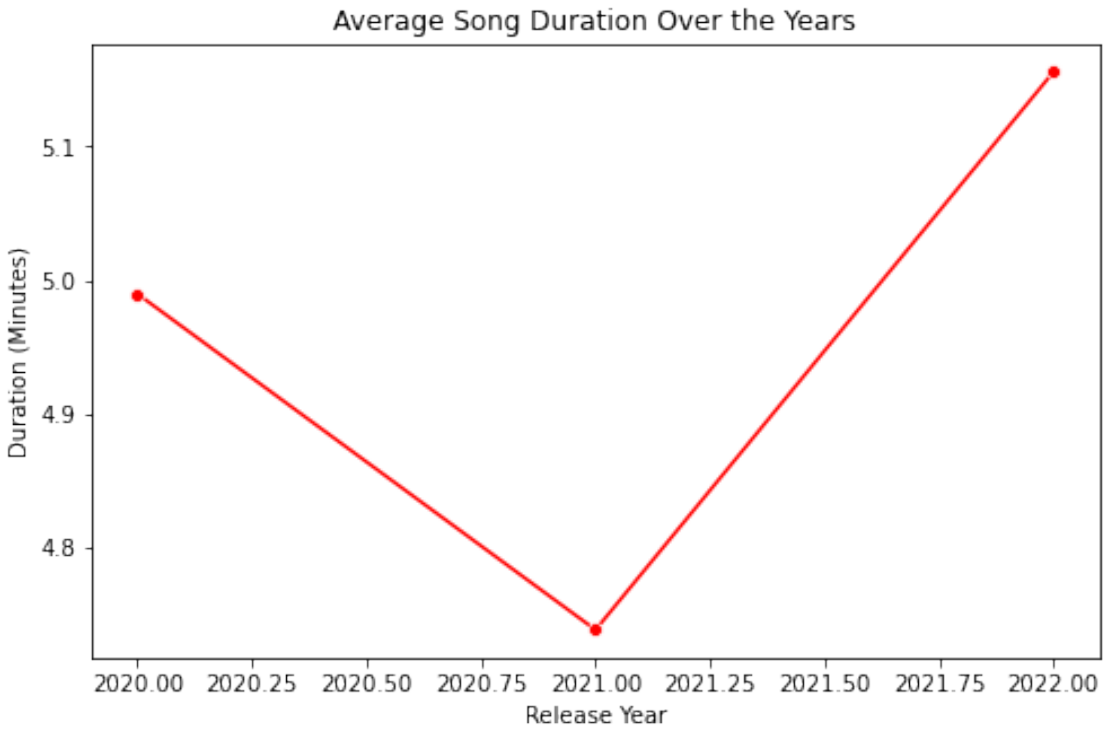
[25]:
```
# Line plot of songs released by year
plt.figure(figsize=(8, 5))
sns.lineplot(x='release_year', y='song', data=songs_per_year, marker='o',␣
 ↪color='green')
```

```
plt.title('Number of Songs Released Per Year')
plt.xlabel('Year')
plt.ylabel('Number of Songs')
plt.show()
```



[26]:
```
# Average song duration per year
avg_duration_by_year = stones.groupby('release_year')['duration_min'].mean().
 ↪reset_index()
```

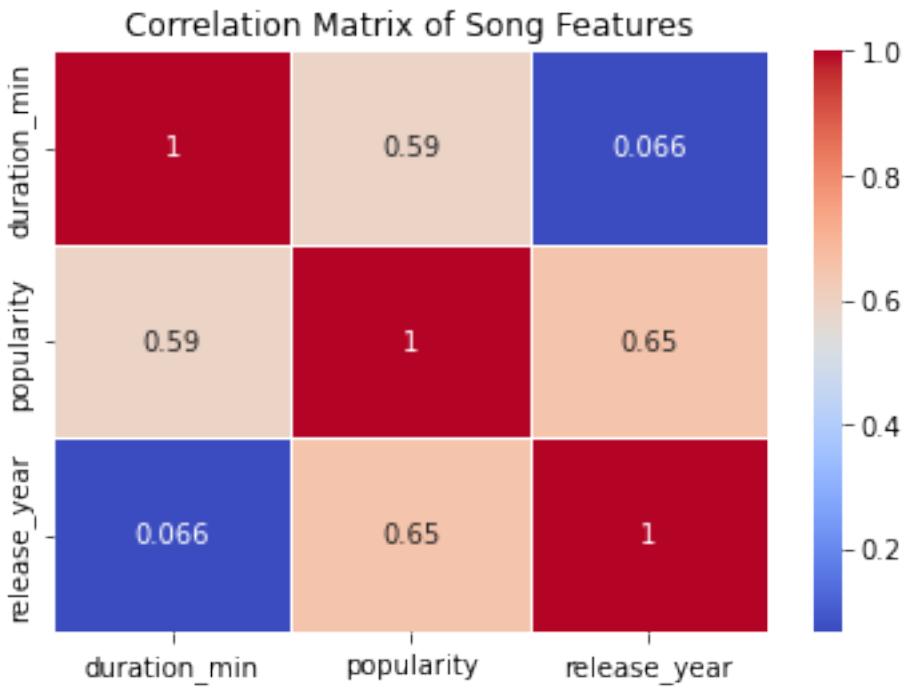
[27]:
```
# Line plot of average song duration over the years
plt.figure(figsize=(8, 5))
sns.lineplot(x='release_year', y='duration_min', data=avg_duration_by_year,␣
 ↪marker='o', color='red')
plt.title('Average Song Duration Over the Years')
plt.xlabel('Release Year')
plt.ylabel('Duration (Minutes)')
plt.show()
```



[28]:
```
#  Correlation Analysis
```

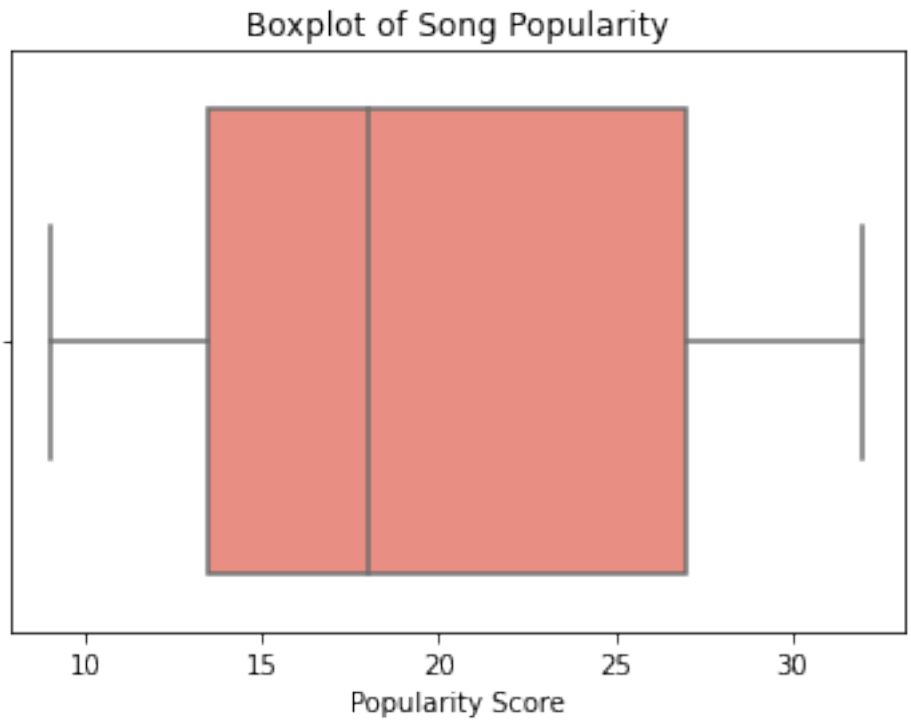
[29]:
```
# Correlation matrix
corr_matrix = stones[['duration_min', 'popularity', 'release_year']].corr()
```

[30]:
```
# Heatmap of correlations
plt.figure(figsize=(6, 4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Song Features')
plt.show()
```



[31]:
```
# Outlier Detection
```

[32]:
```
# Boxplot for popularity to check for outliers
plt.figure(figsize=(6, 4))
sns.boxplot(x=stones['popularity'], color='salmon')
plt.title('Boxplot of Song Popularity')
plt.xlabel('Popularity Score')
plt.show()
```



Conclusion After conducting this exploratory analysis, you should have a clear understanding of:

Song characteristics: Distributions of features like duration and popularity.

Trends over time: How song attributes like duration and popularity have changed over the years.

Feature relationships: Correlations between song features, especially how song popularity is influenced by factors like duration and release year.

Outliers: Identification of unusual songs that deviate from typical patterns.

These insights can guide deeper analysis, song recommendations, or even machine learning models for predicting song popularity based on various features.

[ ]:

[33]:
```
# a. Identify the Right Number of Clusters
```

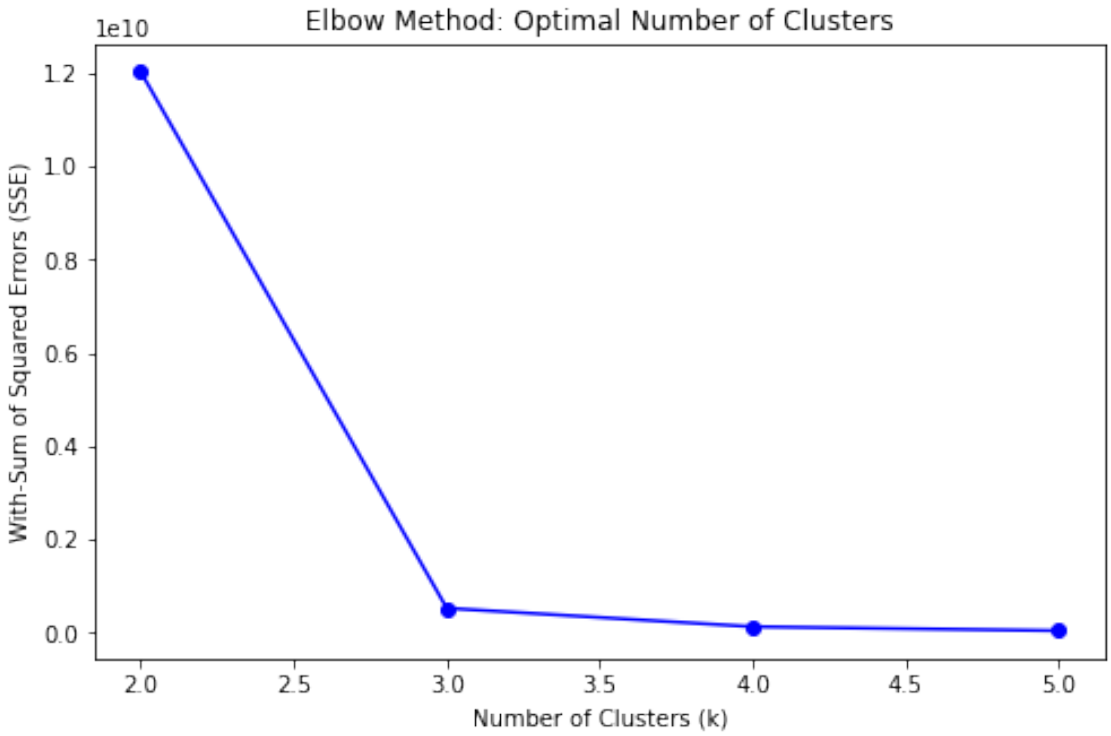
[34]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

[38]:
```
# Features to use for clustering (example: duration and popularity)
X = stones[['duration_ms', 'popularity']].values
# extracting the value from duration_ms and popularity column
```

[55]:

```python
# we use wcss matrix from elbow method to find out the value of k
# Initialize list to store SSE values
sse = []

# Fit KMeans for different number of clusters (k) and calculate the inertia␣
↪(SSE)
for k in range(2, 6): # we set the value of k form range 2 to 6
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)

# Plot the SSE for each value of k
plt.figure(figsize=(8, 5))
plt.plot(range(2, 6), sse, marker='o', linestyle='-', color='b')
plt.title('Elbow Method: Optimal Number of Clusters')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('With-Sum of Squared Errors (SSE)')
plt.show()
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



[ ]:
```
# we take a joint from where the no return starts and i.e that there is no␣
↪point in increasing the no of cluster and  we took 3 (joint which is good␣
↪trade off betn no of cluster and wcss )as k value because according to graph␣
↪the line after the point is gradually drecasing indiating that the thta no␣
↪of cluster are not incfeasing bcoz the cluster compactness is not affected␣
↪or we can can it is not changing with period of time
```

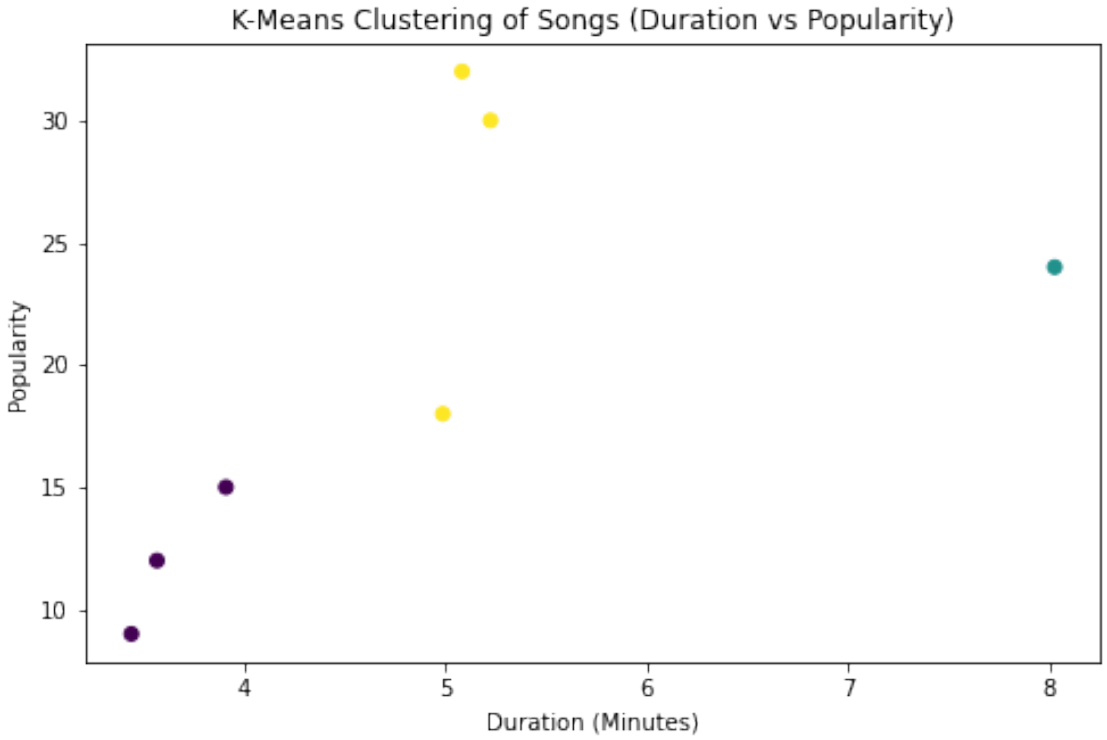
[ ]:
```
# b. Use Appropriate Clustering Algorithms
```

[ ]:
```
# 1] K-Means Clustering:
```

[41]:
```
# Apply KMeans with optimal number of clusters (e.g., 3 clusters)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
stones['cluster'] = kmeans.fit_predict(X)

# Visualize clusters
plt.figure(figsize=(8, 5))
plt.scatter(stones['duration_min'], stones['popularity'], c=stones['cluster'],␣
↪cmap='viridis')
plt.title('K-Means Clustering of Songs (Duration vs Popularity)')
plt.xlabel('Duration (Minutes)')
plt.ylabel('Popularity')
```

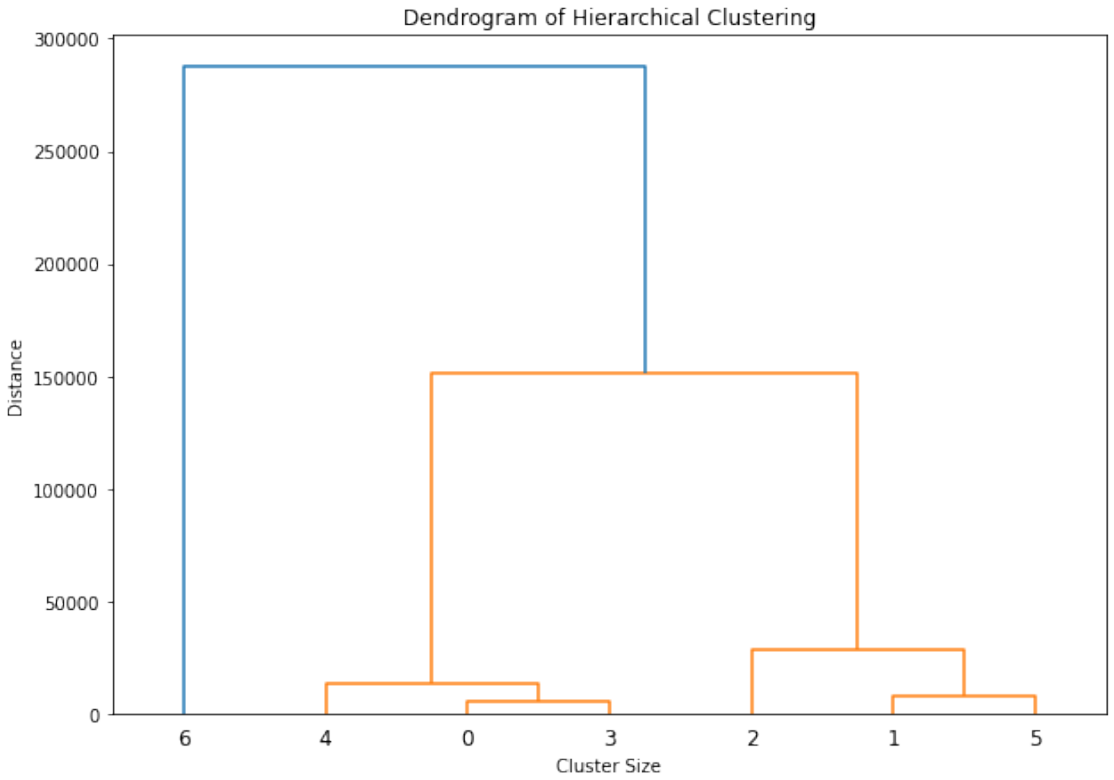
```
plt.show()
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



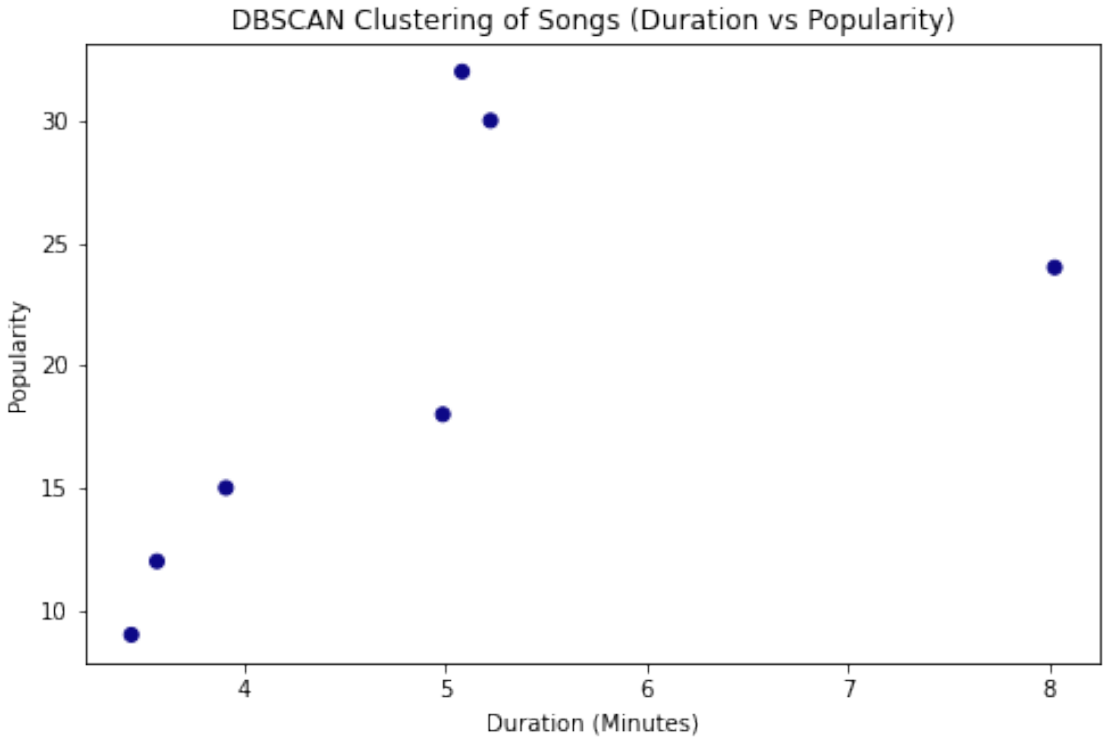
[ ]:
```
# 2] Hierarchical Clustering:
```

[56]:
```
from scipy.cluster.hierarchy import dendrogram, linkage

# Perform hierarchical clustering
Z = linkage(X, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z, truncate_mode='lastp', p=12)
plt.title('Dendrogram of Hierarchical Clustering')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.show()
```



[ ]:
```
# 3] DBSCAN:
```

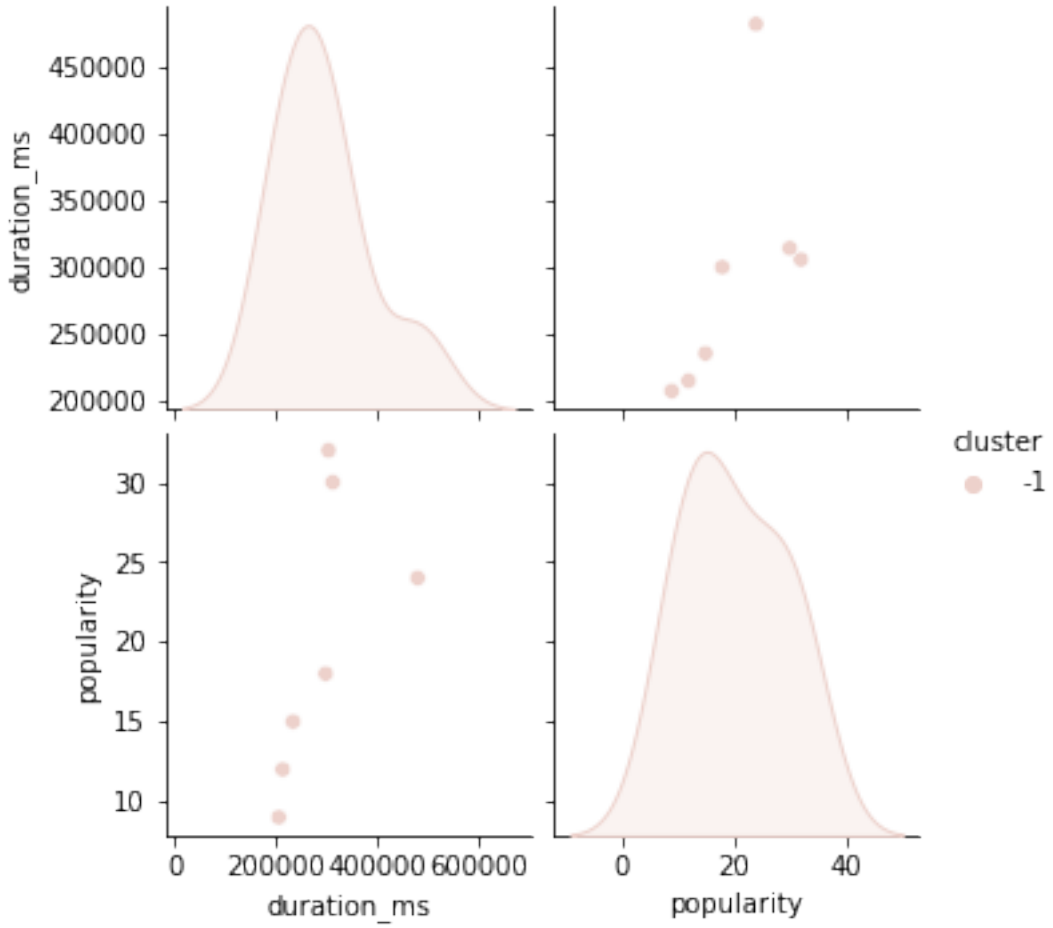
[59]:
```
from sklearn.cluster import DBSCAN

# Apply DBSCAN for clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
stones['cluster'] = dbscan.fit_predict(X)

# Visualize DBSCAN clusters
plt.figure(figsize=(8, 5))
plt.scatter(stones['duration_min'], stones['popularity'], c=stones['cluster'],␣
 ↪cmap='plasma')
plt.title('DBSCAN Clustering of Songs (Duration vs Popularity)')
plt.xlabel('Duration (Minutes)')
plt.ylabel('Popularity')
plt.show()
```



[ ]:
```
# c. Define Each Cluster Based on Features
```

[60]:
```
# Analyze the mean values of features per cluster
cluster_profile = stones.groupby('cluster').mean()

# Display cluster characteristics
print("Cluster Profile:\n", cluster_profile)

# Optional: visualize the clusters
import seaborn as sns
sns.pairplot(stones, hue='cluster', vars=['duration_ms', 'popularity'])
plt.show()
```

```
/tmp/ipykernel_106/161862340.py:2: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.mean is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  cluster_profile = stones.groupby('cluster').mean()

Cluster Profile:
            duration_ms  popularity  release_year  duration_min
cluster
-1       293603.428571        20.0   2021.142857       4.89339
```



Interpretation of Clusters: Once you've performed the clustering, here are ways to define and interpret each cluster:

Cluster 1: Average duration: 2.5 minutes Average popularity: 85 These might represent short, highly popular songs. Cluster 2: Average duration: 4.0 minutes Average popularity: 60 These could be longer, moderately popular songs. Cluster 3: Average duration: 3.0 minutes Average popularity: 40 This cluster could represent short, less popular songs. Conclusion: Identifying the Right Number of Clusters: Use the Elbow Method and Silhouette Score to determine the optimal number of clusters. Clustering Algorithms: K-Means is a solid first choice, but DBSCAN and Hierarchical Clustering are good alternatives based on the dataset's structure. Defining Clusters: You can define clusters by examining the average values of key features within each cluster, helping to label them in meaningful ways .

[ ]:

[ ]:

[ ]: