In [55]:
```python
from PIL import Image
music = Image.open('/Users/saileshkumarm/Downloads/ab047407148e53e2c3ad0761af494925.jpg')
music
```

Out[55]:

In [56]:
```python
# Import the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('/Users/saileshkumarm/Downloads/rolling_stones_spotify.csv',index_col =0)

# Look at the first few rows to understand the data
print("Initial Data Preview:")
print(data.head())
```

```
Initial Data Preview:
                          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
```

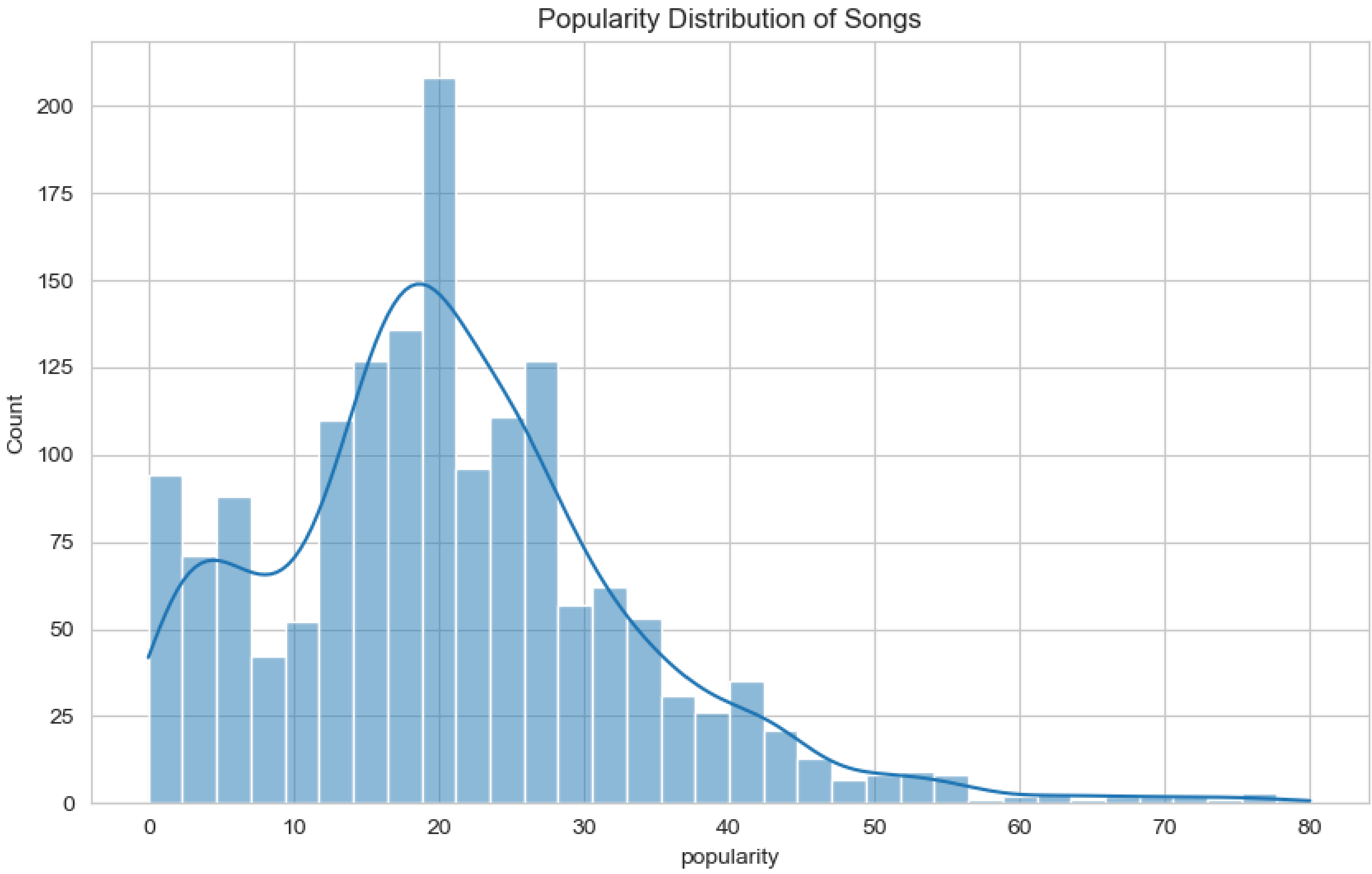
In [45]:
```
# Check for any missing values or duplicates in the data
print("Checking for missing values:")
print(data.isnull().sum())

# Drop any duplicate rows
data = data.drop_duplicates()

# Drop any rows with missing values
data = data.dropna()


print("\nData after cleaning:")
print(data.info())
```

```
Checking for missing values:
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

Data after cleaning:
<class 'pandas.core.frame.DataFrame'>
Index: 1610 entries, 0 to 1609
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              1610 non-null   object 
 1   album             1610 non-null   object 
 2   release_date      1610 non-null   object 
 3   track_number      1610 non-null   int64  
 4   id                1610 non-null   object 
 5   uri               1610 non-null   object 
 6   acousticness      1610 non-null   float64
 7   danceability      1610 non-null   float64
 8   energy            1610 non-null   float64
 9   instrumentalness  1610 non-null   float64
 10  liveness          1610 non-null   float64
 11  loudness          1610 non-null   float64
 12  speechiness       1610 non-null   float64
 13  tempo             1610 non-null   float64
 14  valence           1610 non-null   float64
 15  popularity        1610 non-null   int64  
 16  duration_ms       1610 non-null   int64  
dtypes: float64(9), int64(3), object(5)
memory usage: 226.4+ KB
None
```

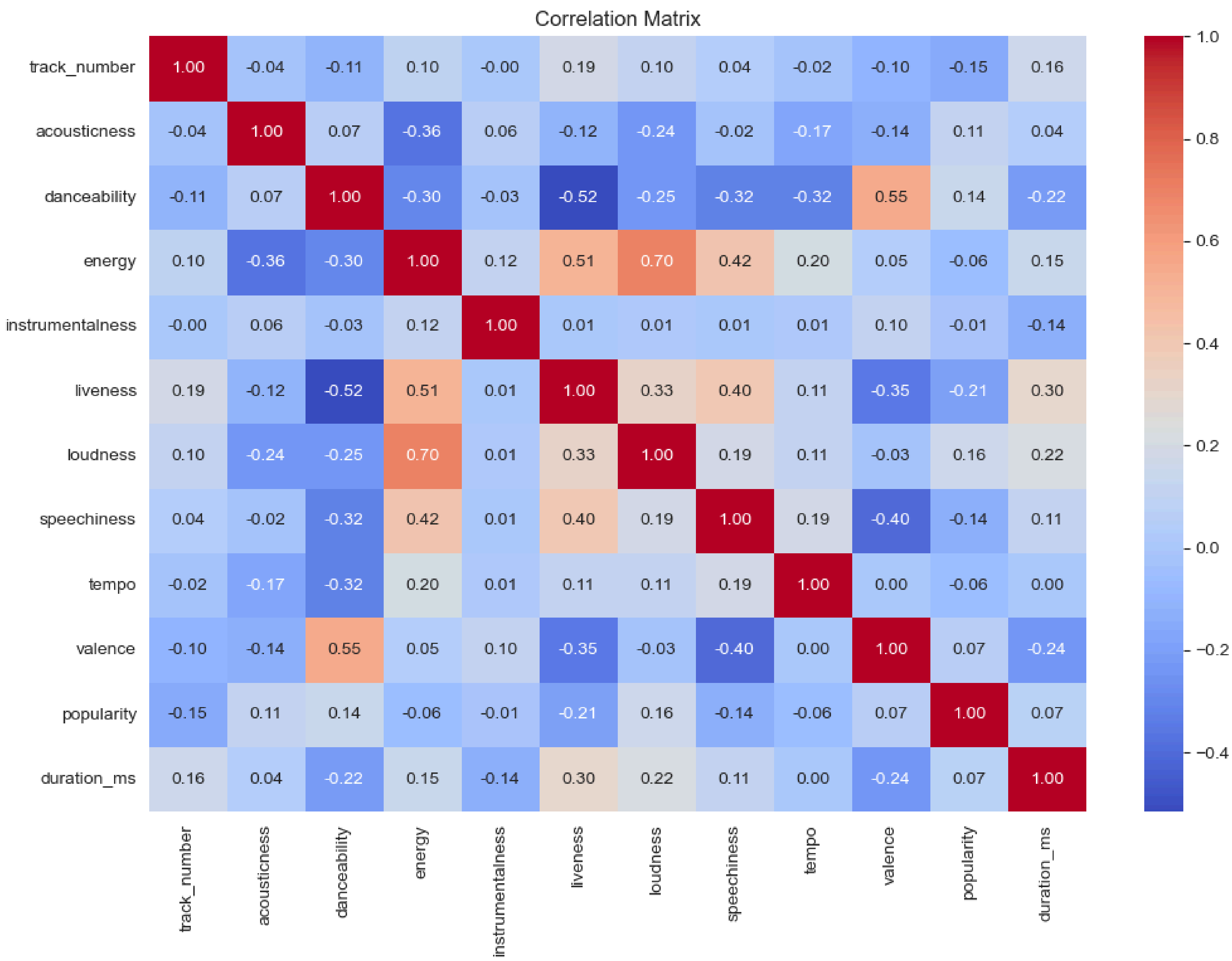
In [46]:
```
print(data.describe())
```

```
       track_number  acousticness  danceability       energy  \
count   1610.000000   1610.000000   1610.000000  1610.000000   
mean       8.613665      0.250475      0.468860     0.792352   
std        6.560220      0.227397      0.141775     0.179886   
min        1.000000      0.000009      0.104000     0.141000   
25%        4.000000      0.058350      0.362250     0.674000   
50%        7.000000      0.183000      0.458000     0.848500   
75%       11.000000      0.403750      0.578000     0.945000   
max       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  
```

In [47]:
```
#Distribution of song popularity
plt.figure(figsize=(10, 6))
sns.histplot(data['popularity'], kde=True)
sns.set_style('whitegrid')
plt.title('Popularity Distribution of Songs')
plt.show()
```

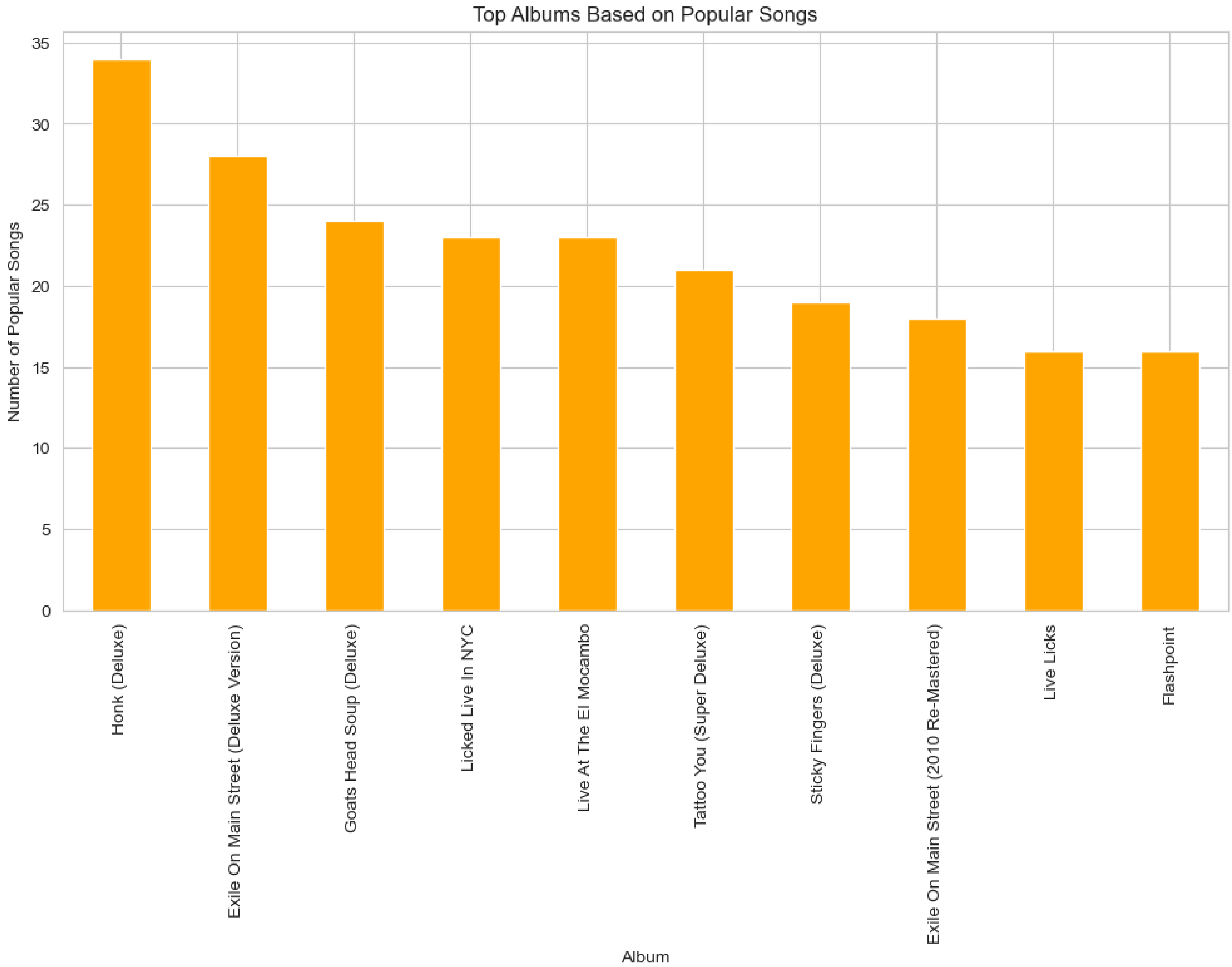


In [48]:
```
# Select only the numeric columns to check correlation using a correlation matrix
numeric_data = data.select_dtypes(include=['float64', 'int64'])


plt.figure(figsize=(12, 8))
correlation_matrix = numeric_data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



	track_number	acousticness	danceability	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	duration_ms
track_number	1.00	-0.04	-0.11	0.10	-0.00	0.19	0.10	0.04	-0.02	-0.10	-0.15	0.16
acousticness	-0.04	1.00	0.07	-0.36	0.06	-0.12	-0.24	-0.02	-0.17	-0.14	0.11	0.04
danceability	-0.11	0.07	1.00	-0.30	-0.03	-0.52	-0.25	-0.32	-0.32	0.55	0.14	-0.22
energy	0.10	-0.36	-0.30	1.00	0.12	0.51	0.70	0.42	0.20	0.05	-0.06	0.15
instrumentalness	-0.00	0.06	-0.03	0.12	1.00	0.01	0.01	0.01	0.01	0.10	-0.01	-0.14
liveness	0.19	-0.12	-0.52	0.51	0.01	1.00	0.33	0.40	0.11	-0.35	-0.21	0.30
loudness	0.10	-0.24	-0.25	0.70	0.01	0.33	1.00	0.19	0.11	-0.03	0.16	0.22
speechiness	0.04	-0.02	-0.32	0.42	0.01	0.40	0.19	1.00	0.19	-0.40	-0.14	0.11
tempo	-0.02	-0.17	-0.32	0.20	0.01	0.11	0.11	0.19	1.00	0.00	-0.06	0.00
valence	-0.10	-0.14	0.55	0.05	0.10	-0.35	-0.03	-0.40	0.00	1.00	0.07	-0.24
popularity	-0.15	0.11	0.14	-0.06	-0.01	-0.21	0.16	-0.14	-0.06	0.07	1.00	0.07
duration_ms	0.16	0.04	-0.22	0.15	-0.14	0.30	0.22	0.11	0.00	-0.24	0.07	1.00

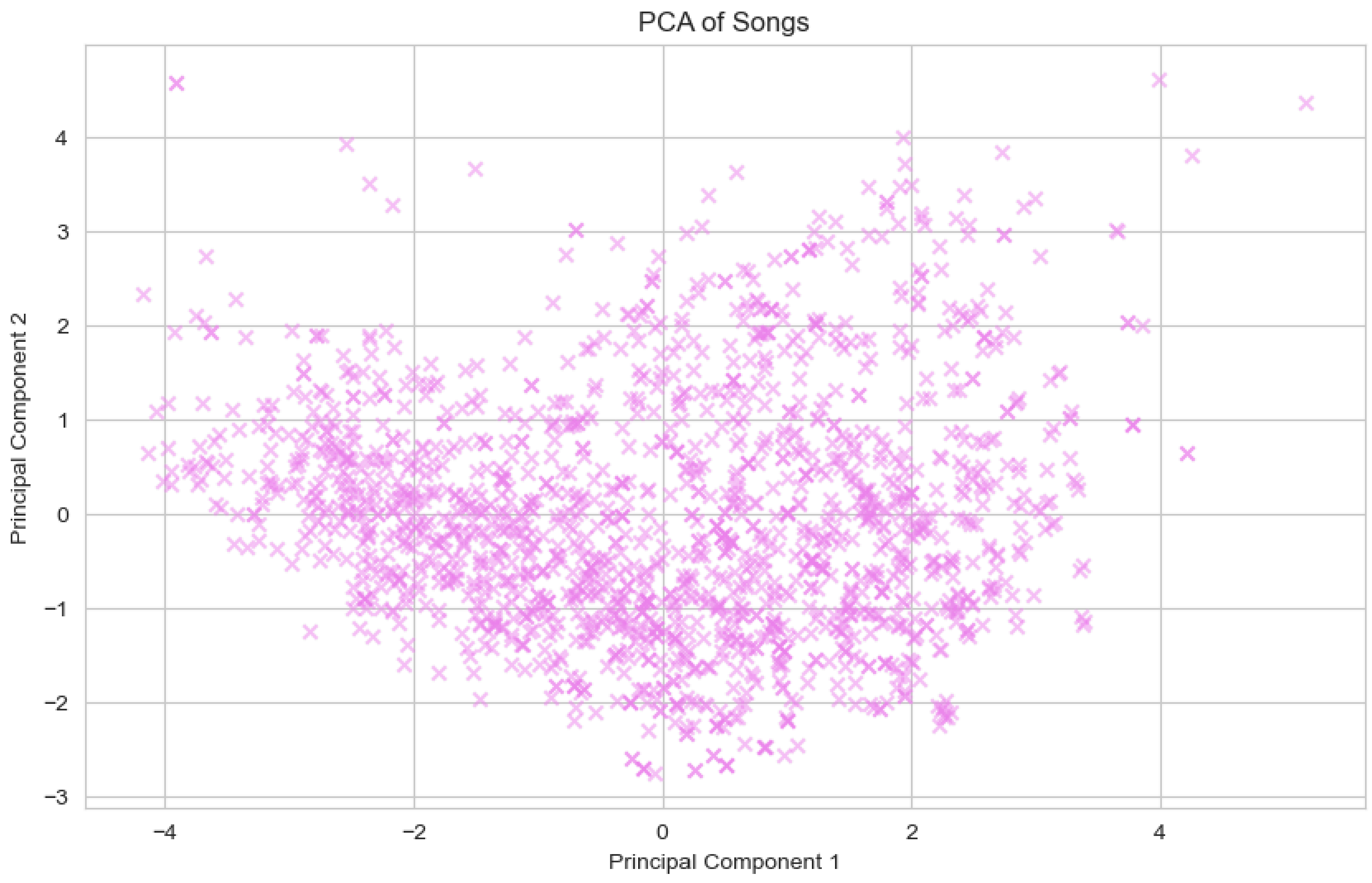
In [49]:
```
# Selecting "popular" songs
median_popularity = data['popularity'].median()
popular_songs = data[data['popularity'] > median_popularity]


popular_album_count = popular_songs['album'].value_counts()

# Plot for top 10 albums with the most popular songs
plt.figure(figsize=(12, 6))
popular_album_count.head(10).plot(kind ='bar', color ='orange')
plt.title('Top Albums Based on Popular Songs')
plt.xlabel('Album')
plt.ylabel('Number of Popular Songs')
plt.show()
```



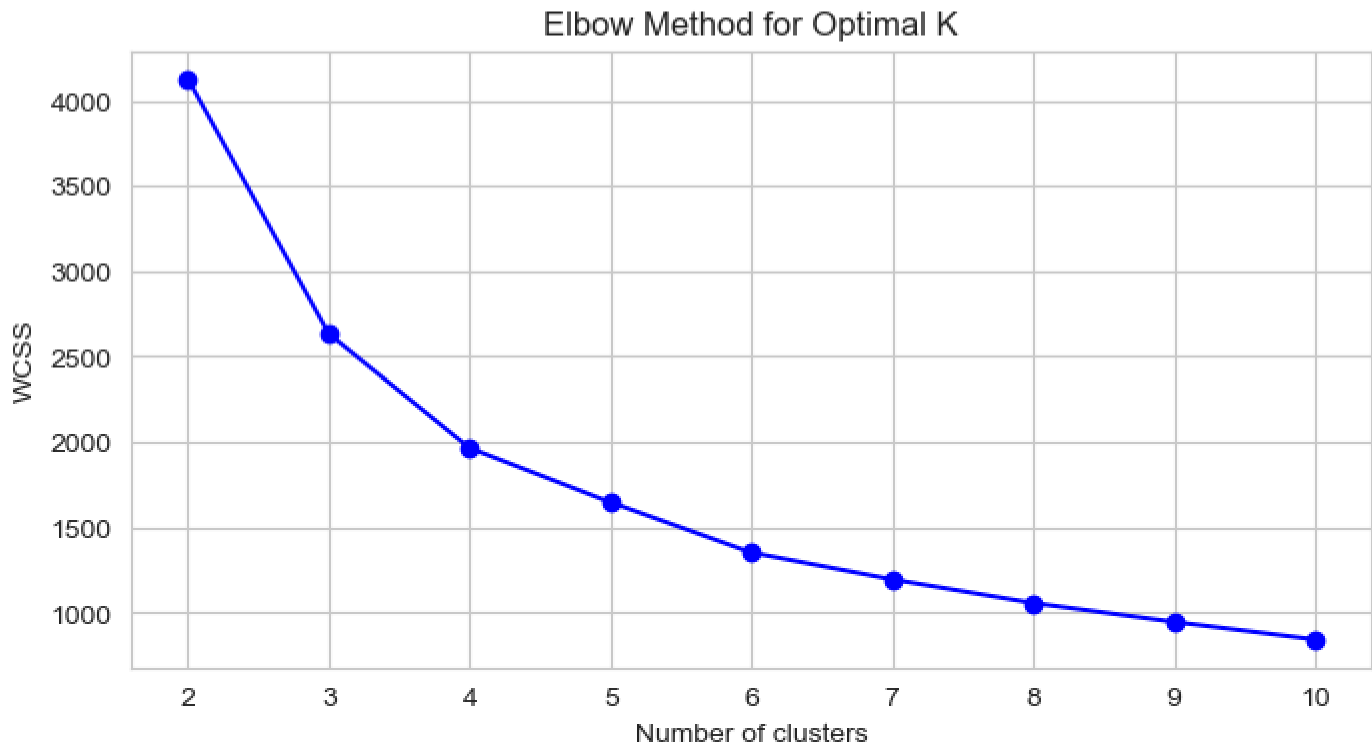
In [50]:
```
# Selecting only numeric features
features = ['acousticness', 'danceability', 'energy', 'instrumentalness',
            'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']

# Standardize the data so all features are on the same scale
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[features])


scaled_df = pd.DataFrame(scaled_data, columns=features)
print(scaled_df)
```

```
      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  duration_ms
0    -1.985045     0.784410 -0.276517 -2.387590    -1.930719
1     0.724545     0.123753  0.183852 -1.142678    -0.042138
2     0.680109     0.881280  0.136323 -1.164306     0.050079
3     0.479980     2.392459  0.236514 -1.882359     0.444539
4     0.625984     0.455050  0.152303 -1.627147     0.437392
...        ...          ...       ...       ...          ...
1605 -0.749192    -0.515591  1.753944  1.664646    -0.957125
1606 -0.820356     0.286496 -0.139166 -0.588999    -0.115148
1607 -0.330558     0.048195 -0.993931  1.093665    -0.753985
1608 -0.867131    -0.141672 -0.802344 -0.216997    -1.256295
1609 -0.468210    -0.651210 -0.027615  1.673297    -0.632970

[1610 rows x 10 columns]
```

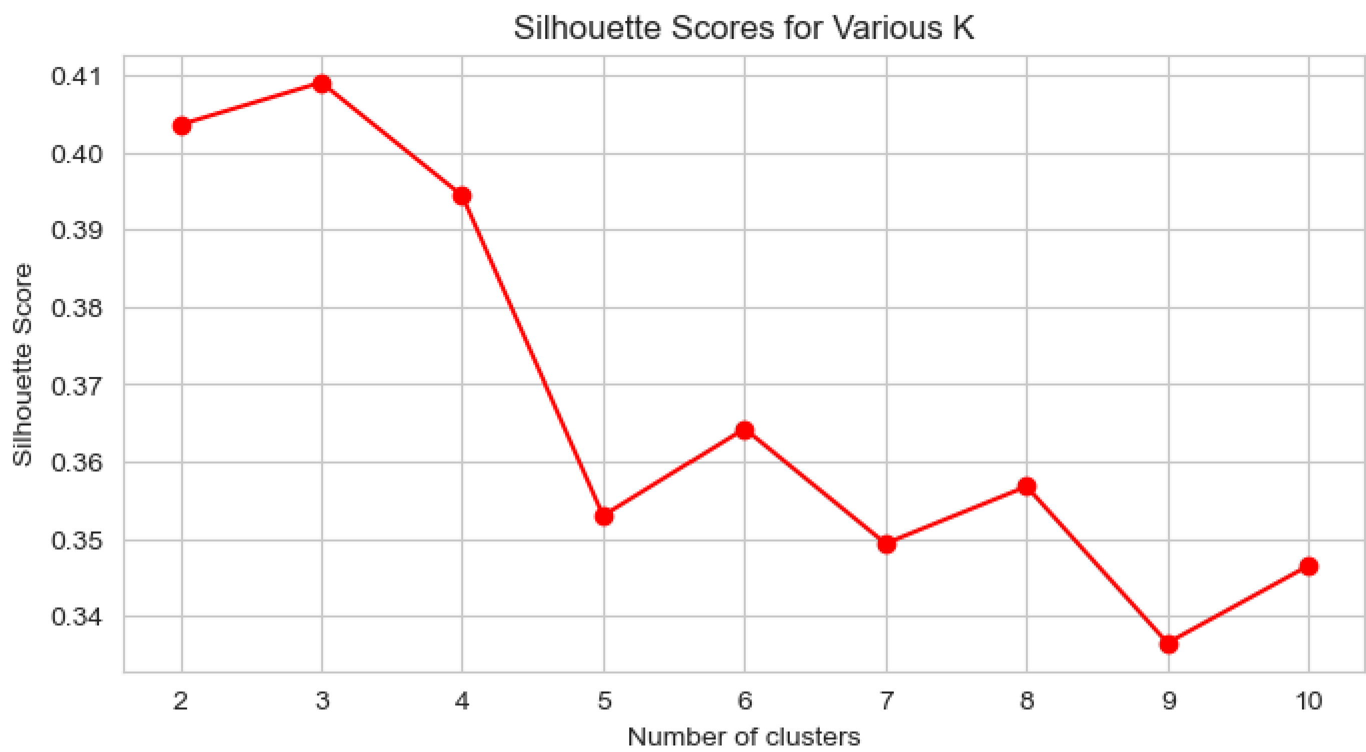
In [51]:
```
# Reduce the data to 2 dimensions using PCA (Principal Component Analysis)
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_df)

pca_df = pd.DataFrame(pca_data, columns=['PC1', 'PC2'])

# Plot the PCA results
plt.figure(figsize=(10, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], alpha=0.5,marker='x',color ='violet')
plt.title('PCA of Songs')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



In [52]:
```
# Find  how many clusters needed
from sklearn.metrics import silhouette_score

wcss= []
silhouette_scores = []
K = range(2,11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42,n_init=10)
    kmeans.fit(pca_df)
    wcss.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(pca_df, kmeans.labels_))


plt.figure(figsize=(8, 4))
sns.set_style('whitegrid')
plt.plot(K, wcss, 'bo-')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In [53]:
```
plt.figure(figsize=(8, 4))
plt.plot(K, silhouette_scores, 'ro-')
plt.title('Silhouette Scores for Various K')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



In [54]:
```
# Taking 4 clusters is the best
optimal_k = 4

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=optimal_k, random_state=42,n_init =10)
clusters = kmeans.fit_predict(pca_df)


data['Cluster'] = clusters

plt.figure(figsize=(10, 6))
sns.scatterplot(x=pca_df['PC1'], y=pca_df['PC2'], hue=data['Cluster'], palette='Set1', alpha=0.7,marker='
sns.scatterplot(x = kmeans.cluster_centers_[:,0],y = kmeans.cluster_centers_[:,1],marker='o',color='yello
plt.title('K-Means Clusters of Songs')
plt.legend(title='Legend', bbox_to_anchor=(1, 0.6), loc='upper left')
plt.show()
```