Build a Recommendation System Music

https://www.kaggle.com/datasets/vatsalmavani/spotify-dataset

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np
import pandas as pd

import seaborn as sns
import plotly.express as px
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from wordcloud import WordCloud

from collections import defaultdict
from scipy.spatial.distance import cdist
from sklearn.preprocessing import MinMaxScaler, StandardScaler

import warnings
warnings.filterwarnings("ignore")

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input direc

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you c
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Importing Data

In [2]:
```
# Saving data from csv to pandas dataframe
data = pd.read_csv("data.csv")
genre_data = pd.read_csv('data_by_genres.csv')
year_data = pd.read_csv('data_by_year.csv')
artist_data = pd.read_csv('data_by_artist.csv')
```

In [3]:
```
data.sample(5)
```

Out[3]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness
70595	0.8310	1997	0.247000	['La Makina']	0.717	276573	0.6780	0	7lCrsNdipwOICCRHFNtOoy	0.000286
22308	0.7300	1937	0.996000	['Naseem Bano']	0.659	206638	0.0352	0	3yyR7MXpeELxM4OvQhqfDX	0.792000
24028	0.0921	1946	0.994000	['Johannes Brahms', 'Eugene Istomin']	0.404	52373	0.1130	0	0fV3WmqXp2YZ6LcO84M7nQ	0.921000
156587	0.7000	1950	0.996000	['Geeta Dutt', 'Lata Mangeshkar']	0.573	197633	0.1600	0	1p4VyPnPQiNdallpF42fkt	0.906000
139041	0.0811	2011	0.000417	['Trent Reznor and Atticus Ross', 'Karen O', '...	0.586	167509	0.9240	0	3g5kQgKEllNBUklsmARGg8	0.956000

In [4]:
```
genre_data.sample(5)
```

Out[4]:

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo
496	1	chaotic hardcore	0.078030	0.427750	169739.083333	0.756000	0.003201	0.206750	-6.156083	0.118783	123.718250
1468	0	italo house	0.017879	0.717769	301557.846154	0.811077	0.219245	0.077008	-7.473846	0.045346	115.476846
2523	1	souldies	0.591308	0.530998	201635.484024	0.474346	0.060641	0.259583	-10.337703	0.040823	116.216000
657	1	cleveland metal	0.001609	0.440714	231443.250000	0.914143	0.164065	0.131957	-5.270464	0.103361	120.111179
2082	0	oriental classical	0.961000	0.199000	181547.000000	0.384000	0.002430	0.151000	-10.759000	0.035000	80.370000

In [5]:
```python
year_data.sample(5)
```

Out[5]:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	vale
79	1	2000	0.289323	0.590918	242724.642638	0.625413	0.101168	0.197686	-8.247766	0.089205	118.999323	0.559
22	1	1943	0.902752	0.455146	240119.909859	0.279990	0.409897	0.239211	-13.602125	0.105720	106.196792	0.495
6	1	1927	0.936179	0.648268	184993.598374	0.264321	0.391328	0.168450	-14.422374	0.113610	114.846524	0.659
95	1	2016	0.284171	0.600202	221396.510295	0.592855	0.093984	0.181170	-8.061056	0.104313	118.652630	0.431
29	1	1950	0.853941	0.504253	215073.125500	0.314071	0.245001	0.216958	-13.863834	0.153453	111.749725	0.551

In [6]:
```python
artist_data.sample(5)
```

Out[6]:

	mode	count	acousticness	artists	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	temp
1116	1	8	0.31785	Andrew Belle	0.5085	279876.75	0.54175	0.005719	0.123725	-6.841	0.031975	126.1667
15814	0	2	0.66400	Makiko	0.7690	231386.00	0.45700	0.005910	0.109000	-9.291	0.033000	101.0170
21406	0	2	0.36500	Roy Ayres	0.7240	348893.00	0.41600	0.000000	0.096900	-8.964	0.084100	151.1810
13335	1	2	0.47900	Kelly Badon	0.5230	264413.00	0.53400	0.000029	0.111000	-8.940	0.029600	75.9880
21928	1	2	0.10200	Sandy & Papo	0.9010	289427.00	0.73200	0.000794	0.028900	-12.316	0.044400	131.8700

In [7]:
```python
# combine for usability
datasets = [("data", data), ("genre_data", genre_data), ("year_data", year_data), ("artist_data", artist_data)]
```

In [8]:
```python
data['year'] = pd.to_datetime(data['year'], format='%Y')
data['release_date'] = pd.to_datetime(data['release_date'])
year_data['year'] = pd.to_datetime(year_data['year'], format='%Y')
```

Checking the Data

In [9]:
```python
#for name, df in datasets:
    # print some info about the datasets
   # print(f"Info about the dataset: {name}")
   # print("-"*30)
    # print(df.info())
    # print()
```

In [10]:
```python
for name, df in datasets:
    # Check for missing values in the datasets
    print(f"Missing Values in: {name}")
    print("-"*30)
    print(df.isnull().sum())
    print()
```

```
Missing Values in: data
------------------------------
valence             0
year                0
acousticness        0
artists             0
danceability        0
duration_ms         0
energy              0
explicit            0
id                  0
instrumentalness    0
key                 0
liveness            0
loudness            0
mode                0
name                0
popularity          0
release_date        0
speechiness         0
tempo               0
dtype: int64

Missing Values in: genre_data
------------------------------
mode                0
genres              0
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
valence             0
popularity          0
key                 0
dtype: int64

Missing Values in: year_data
------------------------------
mode                0
year                0
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
valence             0
popularity          0
key                 0
dtype: int64

Missing Values in: artist_data
------------------------------
mode                0
count               0
acousticness        0
artists             0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
valence             0
popularity          0
key                 0
dtype: int64
```

In [11]:

```
for name, df in datasets:
    # check for duplicates in the datasets
    print(f"Duplicates in the dataset: {name}")
    print("-"*30)
    print(df.duplicated(keep=False).sum())
    print()
```

```
Duplicates in the dataset: data
------------------------------
0

Duplicates in the dataset: genre_data
------------------------------
0

Duplicates in the dataset: year_data
------------------------------
0

Duplicates in the dataset: artist_data
------------------------------
0
```

In [12]:

```python
for name, df in datasets:
    # Check the unique values in the dataset
    print(f"Unique Values in: {name}")
    print("-"*30)
    print(df.nunique())
    print()
```

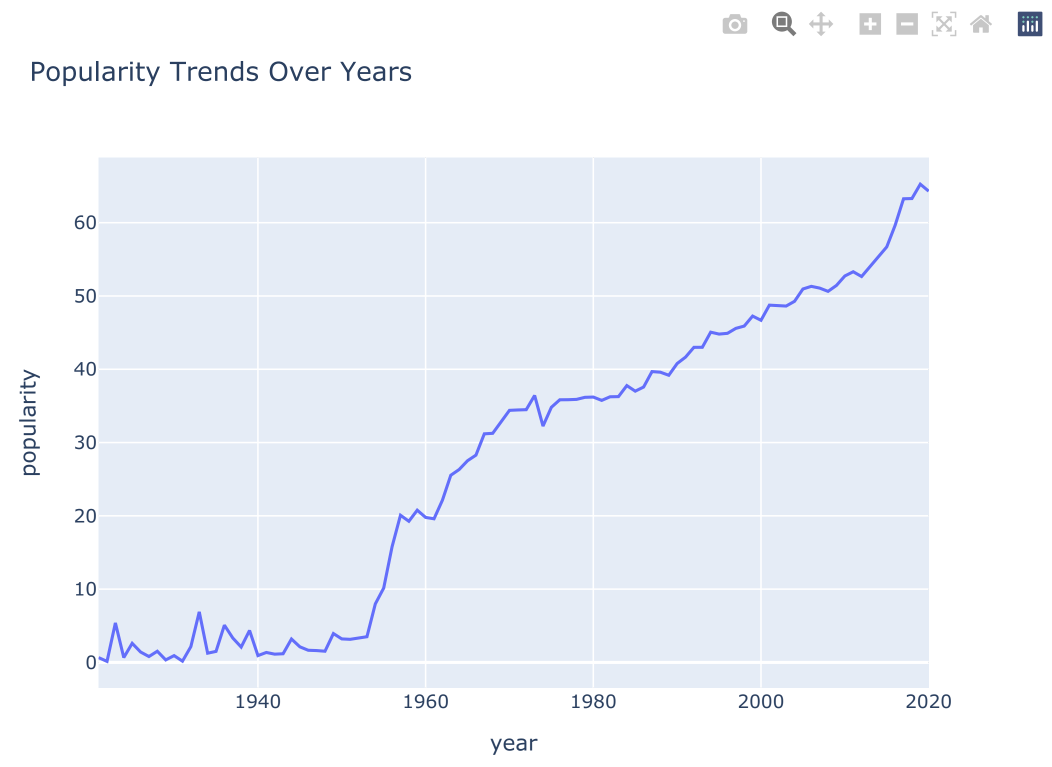
```
Unique Values in: data
------------------------------
valence               1733
year                   100
acousticness          4689
artists              34088
danceability          1240
duration_ms          51755
energy                2332
explicit                 2
id                  170653
instrumentalness      5401
key                     12
liveness              1740
loudness             25410
mode                     2
name                133638
popularity             100
release_date         10968
speechiness           1626
tempo                84694
dtype: int64

Unique Values in: genre_data
------------------------------
mode                   2
genres              2973
acousticness        2798
danceability        2725
duration_ms         2872
energy              2778
instrumentalness    2731
liveness            2709
loudness            2873
speechiness         2707
tempo               2872
valence             2745
popularity          2188
key                   12
dtype: int64

Unique Values in: year_data
------------------------------
mode                  1
year                100
acousticness        100
danceability        100
duration_ms         100
energy              100
instrumentalness    100
liveness            100
loudness            100
speechiness         100
tempo               100
valence             100
popularity          100
key                   7
dtype: int64

Unique Values in: artist_data
------------------------------
mode                    2
count                 379
acousticness        14127
artists             28680
danceability        10650
duration_ms         23960
energy              12126
instrumentalness    15517
liveness            12156
loudness            21862
speechiness         10950
tempo               24801
valence             11882
popularity           4663
key                    12
dtype: int64
```

Data Visualization

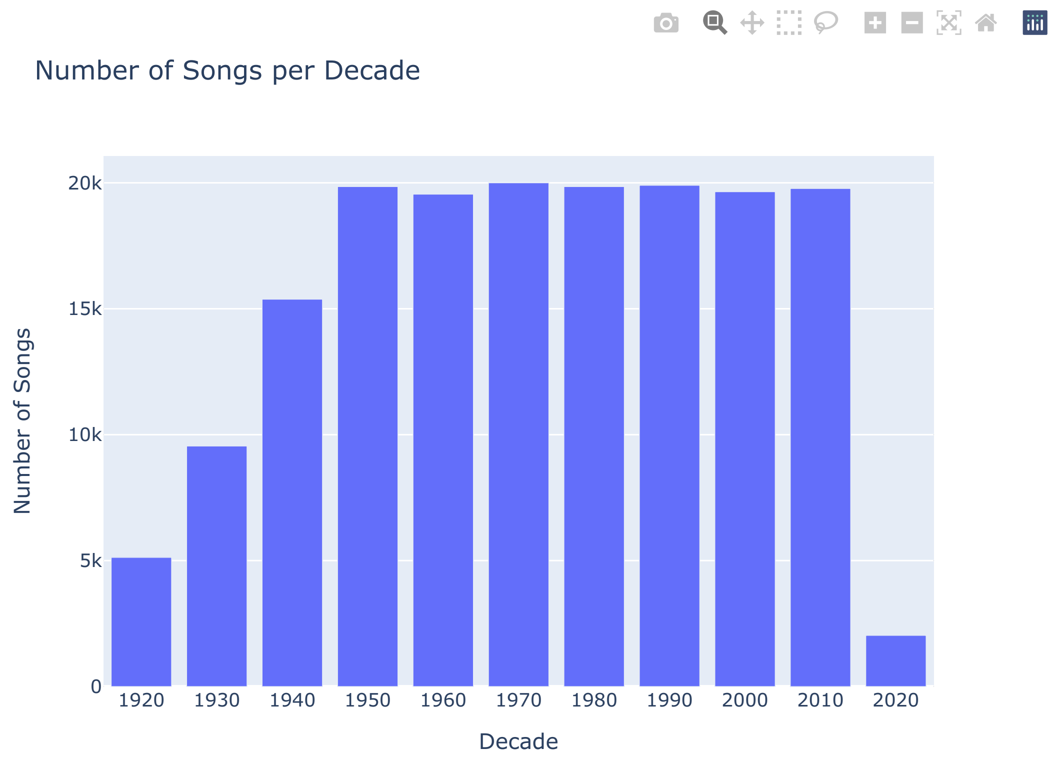
Music overtime

In [13]:
```
# Popularity Trends Over Years
fig = px.line(year_data, x='year', y='popularity', title='Popularity Trends Over Years')
```

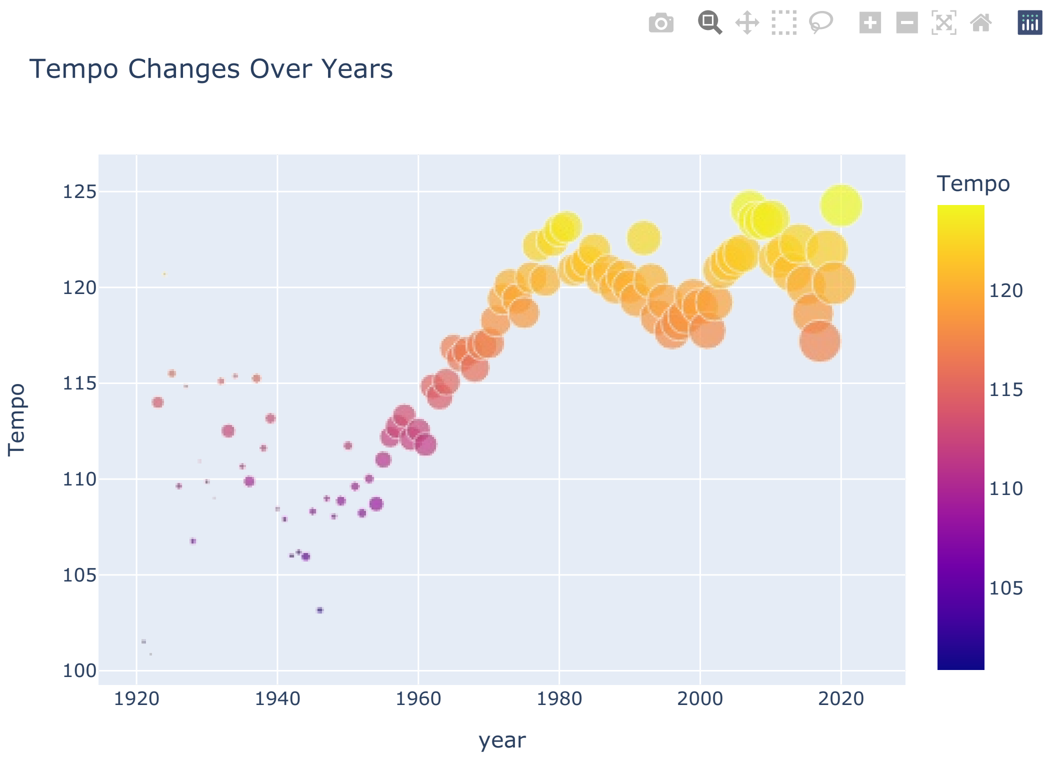
```
fig.show()
```



In [14]:
```
# Convert release_date to datetime and extract decade
data['release_decade'] = (data['release_date'].dt.year // 10) * 10

# Count the number of songs per decade
decade_counts = data['release_decade'].value_counts().sort_index()

# Create a bar chart for songs per decade
fig = px.bar(x=decade_counts.index, y=decade_counts.values, labels={'x': 'Decade', 'y': 'Number of Songs'},
             title='Number of Songs per Decade')
fig.update_layout(xaxis_type='category')
fig.show()
```



In [15]:
```
# Tempo Changes Over Years
fig = px.scatter(year_data, x='year', y='tempo', color='tempo', size='popularity',
                 title='Tempo Changes Over Years', labels={'tempo': 'Tempo'})
fig.show()
```



In [16]:
```python
# Average Danceability Over Years
fig = px.line(year_data, x='year', y='danceability', title='Average Danceability Over Years')
fig.show()
```

In [17]:
```python
# Danceability and Energy Over Years
fig = go.Figure()

fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['danceability'], mode='lines', name='Danceability'))
fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['energy'], mode='lines', name='Energy'))

fig.update_layout(title='Danceability and Energy Over Years', xaxis_title='Year', yaxis_title='Value')
fig.show()
```

In [18]:
```python
# Energy and Acousticness Over Years
fig = go.Figure()

fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['energy'], mode='lines', name='Energy'))
fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['acousticness'], mode='lines', name='Acousticness'))

fig.update_layout(title='Energy and Acousticness Over Years', xaxis_title='Year', yaxis_title='Value')
fig.show()
```

In [19]:
```python
# Speechiness and Instrumentalness Over Years
fig = go.Figure()

fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['speechiness'], mode='lines', name='Speechiness'))
fig.add_trace(go.Scatter(x=year_data['year'], y=year_data['instrumentalness'], mode='lines', name='Instrumental

fig.update_layout(title='Speechiness and Instrumentalness Over Years', xaxis_title='Year', yaxis_title='Value')
fig.show()
```

In [20]:
```python
# Valence Distribution by Release Year
fig = px.box(data, x=data['release_date'].dt.year, y='valence', title='Valence Distribution by Release Year')
fig.show()
```

In [21]:
```python
# Release Frequency Over Years
release_counts = data['release_date'].dt.year.value_counts().reset_index()
release_counts.columns = ['Year', 'Count']

fig = px.bar(release_counts, x='Year', y='Count', title='Release Frequency Over Years')
fig.show()
```

Genres

In [22]:
```
# Genre Analysis: Top Genres by Popularity
top_10_genre_data = genre_data.nlargest(10, 'popularity')

fig = px.bar(top_10_genre_data, x='popularity', y='genres', orientation='h',
             title='Top Genres by Popularity', color='genres')
fig.show()
```

In [23]:
```
# Genre Analysis: Danceability Distribution for Top 10 Popular Genres
fig = px.bar(top_10_genre_data, x='genres', y='danceability', color='genres',
             title='Danceability Distribution for Top 10 Popular Genres')
fig.show()
```

In [24]:
```
# Genre Analysis: Energy Distribution for Top 10 Popular Genres
fig = px.bar(top_10_genre_data, x='genres', y='energy', color='genres',
             title='Energy Distribution for Top 10 Popular Genres')
fig.show()
```

In [25]:
```
# Genre Analysis: Valence Distribution for Top 10 Popular Genres
fig = px.bar(top_10_genre_data, x='genres', y='valence', color='genres',
             title='Valence Distribution for Top 10 Popular Genres')
fig.show()
```

```
In [26]: # Genre Analysis: Acousticness Distribution for Top 10 Popular Genres
         fig = px.bar(top_10_genre_data, x='genres', y='acousticness', color='genres',
                      title='Acousticness Distribution for Top 10 Popular Genres')
         fig.show()
```

```
In [27]: # Genre Analysis: Instrumentalness Distribution for Top 10 Popular Genres
         fig = px.bar(top_10_genre_data, x='genres', y='instrumentalness', color='genres',
                      title='Instrumentalness Distribution for Top 10 Popular Genres')
         fig.show()
```

Artists

```
In [28]: # Artist Analysis: Average Attributes for Top 10 Popular Artists
         top_10_artist_data = artist_data.nlargest(10, 'popularity')

         fig = px.bar(top_10_artist_data, x='popularity', y='artists', orientation='h', color='artists',
                      title='Top Artists by Popularity')
         fig.show()
```

```
In [29]: fig = px.scatter(top_10_artist_data, x='speechiness', y='instrumentalness', color='artists',
                          size='popularity', hover_name='artists',
                          title='Speechiness vs. Instrumentalness for Top Artists')
         fig.show()
```

In [30]:
```
# Artist Analysis: Danceability vs. Energy for Top 10 Popular Artists
fig = px.scatter(top_10_artist_data, x='danceability', y='energy', color='artists',
                 size='popularity', hover_name='artists',
                 title='Danceability vs. Energy for Top 10 Popular Artists')
fig.show()
```

Songs

In [31]:
```
# Song Analysis: Top Songs by Popularity
top_songs = data.nlargest(10, 'popularity')

fig = px.bar(top_songs, x='popularity', y='name', orientation='h',
             title='Top Songs by Popularity', color='name')
fig.show()
```

In [32]:
```
fig = px.scatter(top_songs, x='danceability', y='energy', color='popularity',
                 size='popularity', hover_name='name',
                 title='Danceability vs. Energy for Top Songs')
fig.show()
```

In [33]:
```
fig = px.scatter(top_songs, x='speechiness', y='instrumentalness', color='popularity',
                 size='popularity', hover_name='name',
                 title='Speechiness vs. Instrumentalness for Top Songs')
fig.show()
```

Building the recommender system

In [34]:
```python
# Convert year column back
data['year'] = data['year'].dt.year
```

In [35]:
```python
# List of numerical columns to consider for similarity calculations
number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit', 'year',
               'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']
```

In [36]:
```python
# Function to retrieve song data for a given song name
def get_song_data(name, data):
    try:
        return data[data['name'].str.lower() == name].iloc[0]
        return song_data
    except IndexError:
        return None
```

In [37]:
```python
# Function to calculate the mean vector of a list of songs
def get_mean_vector(song_list, data):
    song_vectors = []
    for song in song_list:
        song_data = get_song_data(song['name'], data)
        if song_data is None:
            print('Warning: {} does not exist in the dataset'.format(song['name']))
            return None
        song_vector = song_data[number_cols].values
        song_vectors.append(song_vector)
    song_matrix = np.array(list(song_vectors))
    return np.mean(song_matrix, axis=0)
```

In [38]:
```python
# Function to flatten a list of dictionaries into a single dictionary
def flatten_dict_list(dict_list):
    flattened_dict = defaultdict()
    for key in dict_list[0].keys():
        flattened_dict[key] = []
    for dictionary in dict_list:
        for key, value in dictionary.items():
            flattened_dict[key].append(value)
    return flattened_dict
```

In [39]:
```python
# Normalize the song data using Min-Max Scaler
min_max_scaler = MinMaxScaler()
normalized_data = min_max_scaler.fit_transform(data[number_cols])

# Standardize the normalized data using Standard Scaler
standard_scaler = StandardScaler()
scaled_normalized_data = standard_scaler.fit_transform(normalized_data)
```

In [40]:
```python
# Function to recommend songs based on a list of seed songs
def recommend_songs(seed_songs, data, n_recommendations=10):
    metadata_cols = ['name', 'artists', 'year']
    song_center = get_mean_vector(seed_songs, data)

    # Return an empty list if song_center is missing
    if song_center is None:
        return []

    # Normalize the song center
    normalized_song_center = min_max_scaler.transform([song_center])

    # Standardize the normalized song center
    scaled_normalized_song_center = standard_scaler.transform(normalized_song_center)

    # Calculate Euclidean distances and get recommendations
    distances = cdist(scaled_normalized_song_center, scaled_normalized_data, 'euclidean')
    index = np.argsort(distances)[0]

    # Filter out seed songs and duplicates, then get the top n_recommendations
    rec_songs = []
    for i in index:
        song_name = data.iloc[i]['name']
        if song_name not in [song['name'] for song in seed_songs] and song_name not in [song['name'] for song i
            rec_songs.append(data.iloc[i])
            if len(rec_songs) == n_recommendations:
                break

    return pd.DataFrame(rec_songs)[metadata_cols].to_dict(orient='records')
```

In [41]:

```python
# List of seed songs (replace with your own seed songs)
seed_songs = [
    {'name': 'Paranoid'},
    {'name': 'Blinding Lights'},
    # Add more seed songs as needed
]
seed_songs = [{'name': name['name'].lower()} for name in seed_songs]

# Number of recommended songs
n_recommendations = 15

# Call the recommend_songs function
recommended_songs = recommend_songs(seed_songs, data, n_recommendations)

# Convert the recommended songs to a DataFrame
recommended_df = pd.DataFrame(recommended_songs)

# Print the recommended songs
for idx, song in enumerate(recommended_songs, start=1):
    print(f"{idx}. {song['name']} by {song['artists']} ({song['year']})")
```

```
1. Infinity by ['One Direction'] (2015)
2. Secrets by ['OneRepublic'] (2009)
3. In My Blood by ['Shawn Mendes'] (2018)
4. Head Above Water by ['Avril Lavigne'] (2019)
5. Green Light by ['Lorde'] (2017)
6. My Wish by ['Rascal Flatts'] (2006)
7. Magic Shop by ['BTS'] (2018)
8. Good Things Fall Apart (with Jon Bellion) by ['ILLENIUM', 'Jon Bellion'] (2019)
9. Inside Out (feat. Griff) by ['Zedd', 'Griff'] (2020)
10. A.M. by ['One Direction'] (2015)
11. Love You Goodbye by ['One Direction'] (2015)
12. Story of My Life by ['One Direction'] (2013)
13. Perfect by ['Simple Plan'] (2018)
14. arms by ['Christina Perri'] (2011)
15. Breezeblocks by ['alt-J'] (2012)
```

In [42]:

```python
# Create a bar plot of recommended songs by name
recommended_df['text'] = recommended_df.apply(lambda row: f"{row.name + 1}. {row['name']} by {row['artists']} (
fig = px.bar(recommended_df, y='name', x=range(n_recommendations, 0, -1), title='Recommended Songs', orientatio
fig.update_layout(xaxis_title='Recommendation Rank', yaxis_title='Songs', showlegend=False, uniformtext_minsize
fig.update_traces(width=1)
fig.show()
```