Spotify Recommendation System

Datasets used:

- Data:

```
[7] data.head()
```

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz

instrumentalness	key	liveness	loudness	mode	name	popularity	release_date	speechiness	tempo
0.878000	10	0.665	-20.096	1.0	Piano Concerto No. 3 in D Minor, Op. 30: III. ...	4.0	1921	0.0366	80.954
0.000000	7	0.160	-12.441	1.0	Clancy Lowered the Boom	5.0	1921	0.4150	60.936

- Data by genres:

```
genre_data.head()
```

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness
0	1	21st century classical	0.979333	0.162883	1.602977e+05	0.071317	0.606834
1	1	432hz	0.494780	0.299333	1.048887e+06	0.450678	0.477762

s	liveness	loudness	speechiness	tempo	valence	popularity	key
4	0.361600	-31.514333	0.040567	75.336500	0.103783	27.833333	6

- Data by year

```
year_data.head()
```

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878

liveness	loudness	speechiness	tempo	valence	popularity	key
0.205710	-17.048667	0.073662	101.531493	0.379327	0.653333	2

- Data by artist:

```
artist_data.head()
```

	mode	count	acousticness	artists	danceability	duration_ms	energy
0	1	9	0.590111	"Cats" 1981 Original London Cast	0.467222	250318.555556	0.394003

energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	key
0.394003	0.011400	0.290833	-14.448000	0.210389	117.518111	0.389500	38.333333	5

Data exploration and EDA:

- Identified the data types and non-null values:

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   valence          170653 non-null  float64
 1   year             170653 non-null  int64
 2   acousticness     170653 non-null  float64
 3   artists          170653 non-null  object
 4   danceability     170653 non-null  float64
 5   duration_ms      170653 non-null  int64
 6   energy           170653 non-null  float64
 7   explicit         170653 non-null  int64
 8   id               170653 non-null  object
 9   instrumentalness 170653 non-null  float64
 10  key              170653 non-null  int64
 11  liveness         170653 non-null  float64
 12  loudness         170653 non-null  float64
 13  mode             170653 non-null  int64
 14  name             170653 non-null  object
 15  popularity       170653 non-null  int64
 16  release_date     170653 non-null  object
 17  speechiness      170653 non-null  float64
 18  tempo            170653 non-null  float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB
None
```

```
[ ] print(genre_data.info())
```

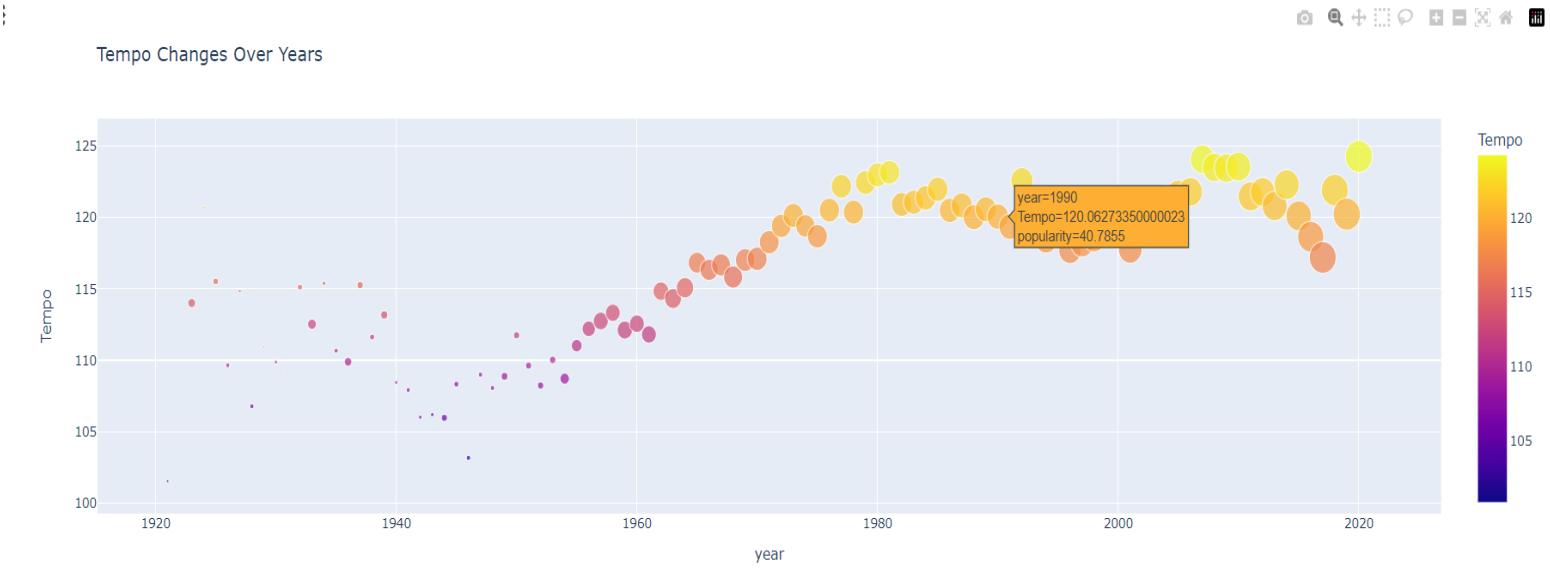
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   mode             2973 non-null   int64
 1   genres           2973 non-null   object
 2   acousticness     2973 non-null   float64
 3   danceability     2973 non-null   float64
 4   duration_ms      2973 non-null   float64
 5   energy           2973 non-null   float64
 6   instrumentalness 2973 non-null   float64
 7   liveness         2973 non-null   float64
 8   loudness         2973 non-null   float64
 9   speechiness      2973 non-null   float64
 10  tempo            2973 non-null   float64
 11  valence          2973 non-null   float64
 12  popularity       2973 non-null   float64
 13  key              2973 non-null   int64
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KB
None
```

```
print(year_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   mode             100 non-null    int64
 1   year             100 non-null    int64
 2   acousticness     100 non-null    float64
 3   danceability     100 non-null    float64
 4   duration_ms      100 non-null    float64
 5   energy           100 non-null    float64
 6   instrumentalness 100 non-null    float64
 7   liveness         100 non-null    float64
 8   loudness         100 non-null    float64
 9   speechiness      100 non-null    float64
 10  tempo            100 non-null    float64
 11  valence          100 non-null    float64
 12  popularity       100 non-null    float64
 13  key              100 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 11.1 KB
None
```

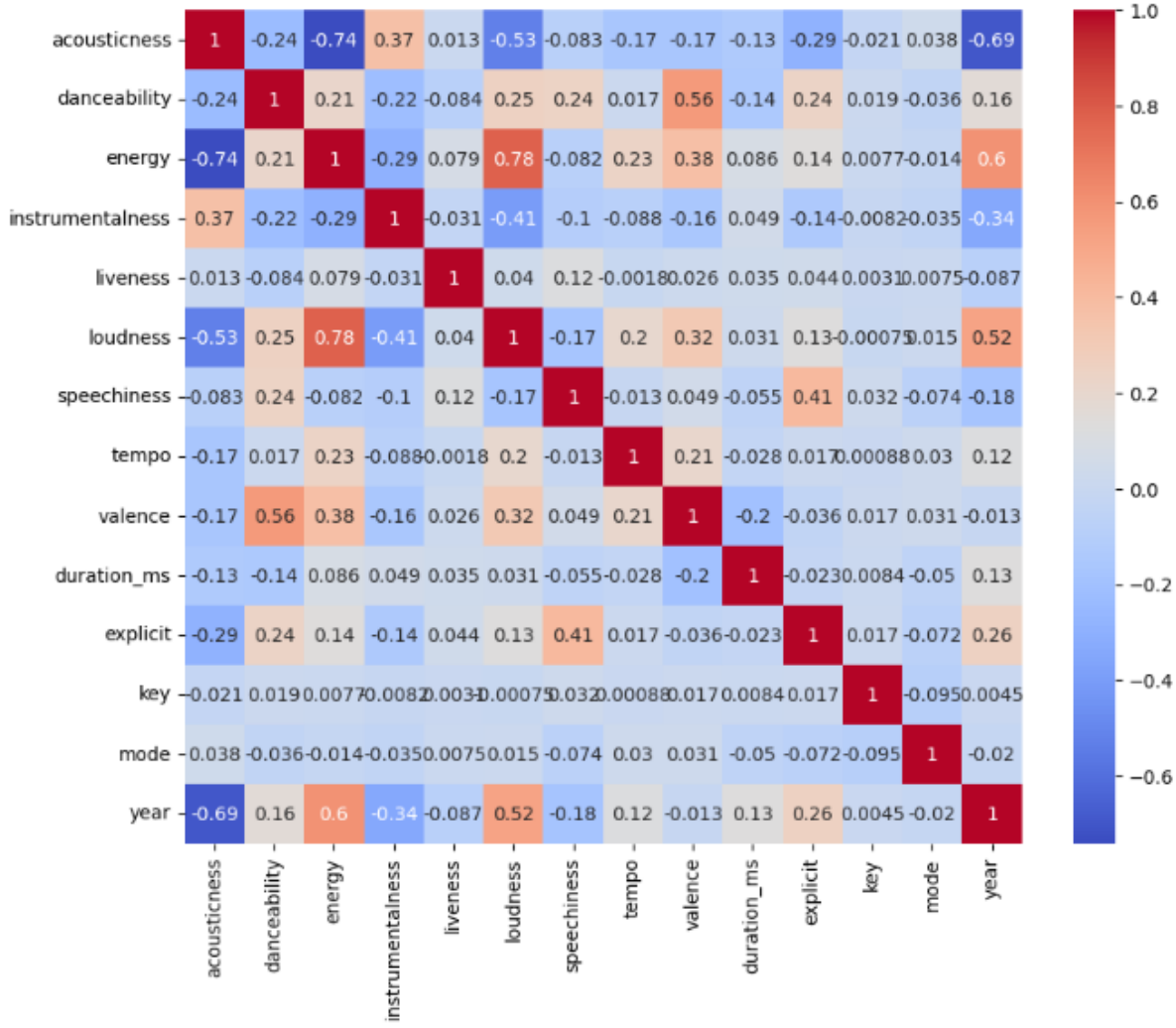
- Using the data grouped by year, we can understand how the overall sound of music has changed from 1921 to 2020.

```
# 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()
```



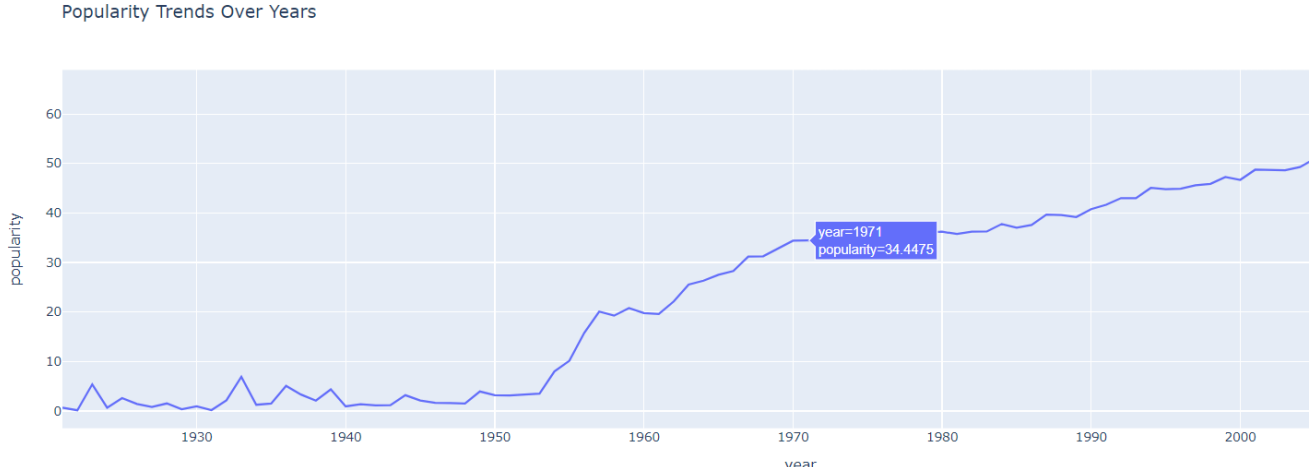
- Correlation matrix:

```
plt.figure(figsize=(10,8))
sns.heatmap(data[feature_names].corr(), annot=True, cmap='coolwarm')
plt.show()
```



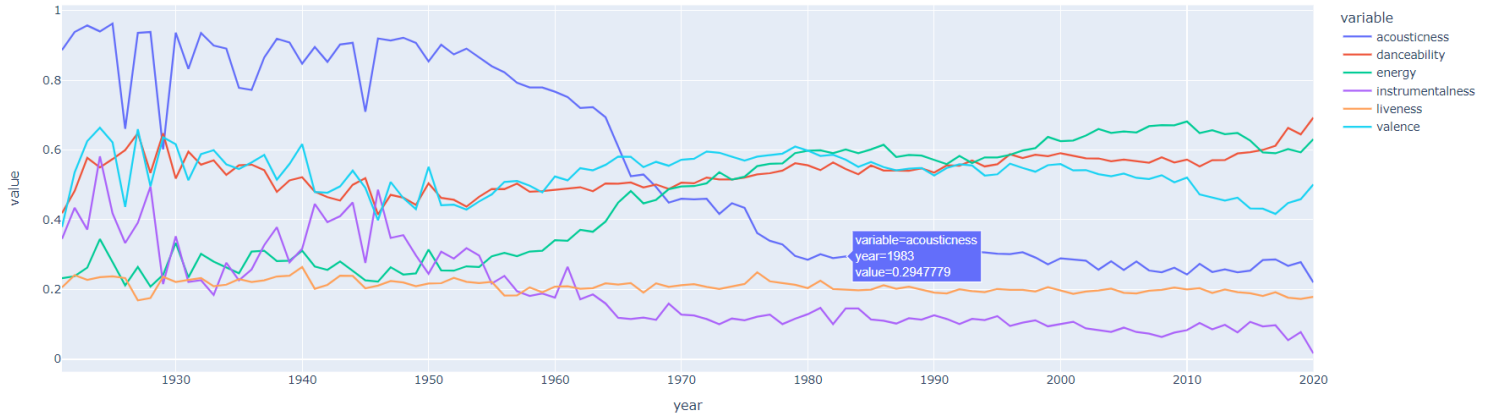
- Popularity:

```
fig = px.line(year_data, x='year', y='popularity', title='Popularity Trends Over Years')
fig.show()
```



- Graph of change in variables with respect to time :

```
sound_features = ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'valence']
fig = px.line(year_data, x='year', y=sound_features)
fig.show()
```

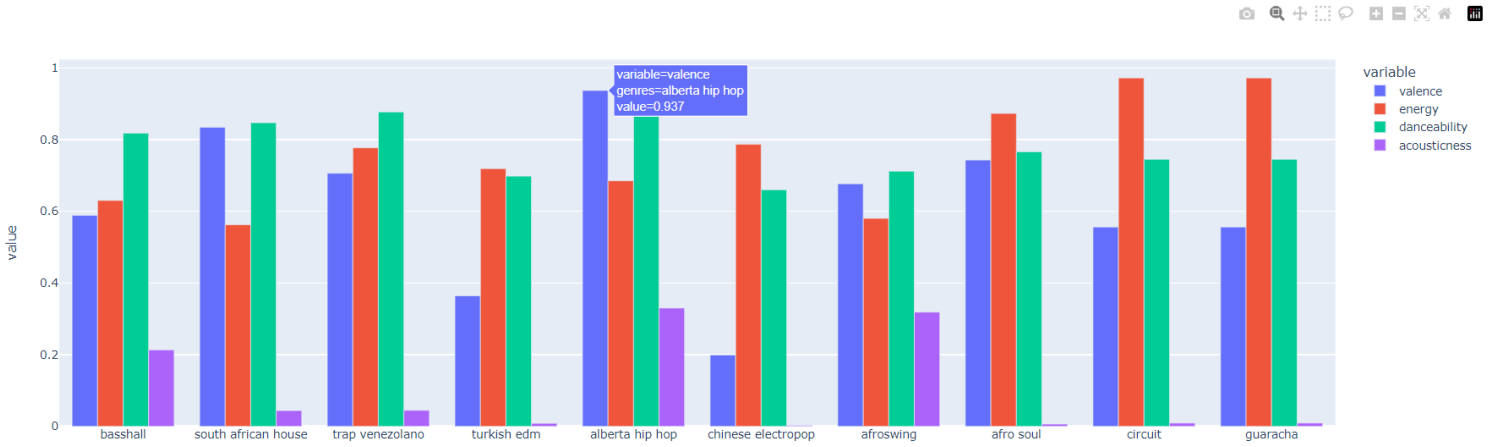


Characteristics of Different Genres

- This dataset contains the audio features for different songs along with the audio features for different genres. We can use this information to compare different genres and understand their unique differences in sound.

```
top10_genres = genre_data.nlargest(10, 'popularity')

fig = px.bar(top10_genres, x='genres', y=['valence', 'energy', 'danceability', 'acousticness'], barmode='group')
fig.show()
```



Clustering Genres with K-Means

- Here, the simple K-means clustering algorithm is used to divide the genres in this dataset into ten clusters based on the numerical audio features of each genres.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=10))])
X = genre_data.select_dtypes(np.number)
cluster_pipeline.fit(X)
genre_data['cluster'] = cluster_pipeline.predict(X)
```

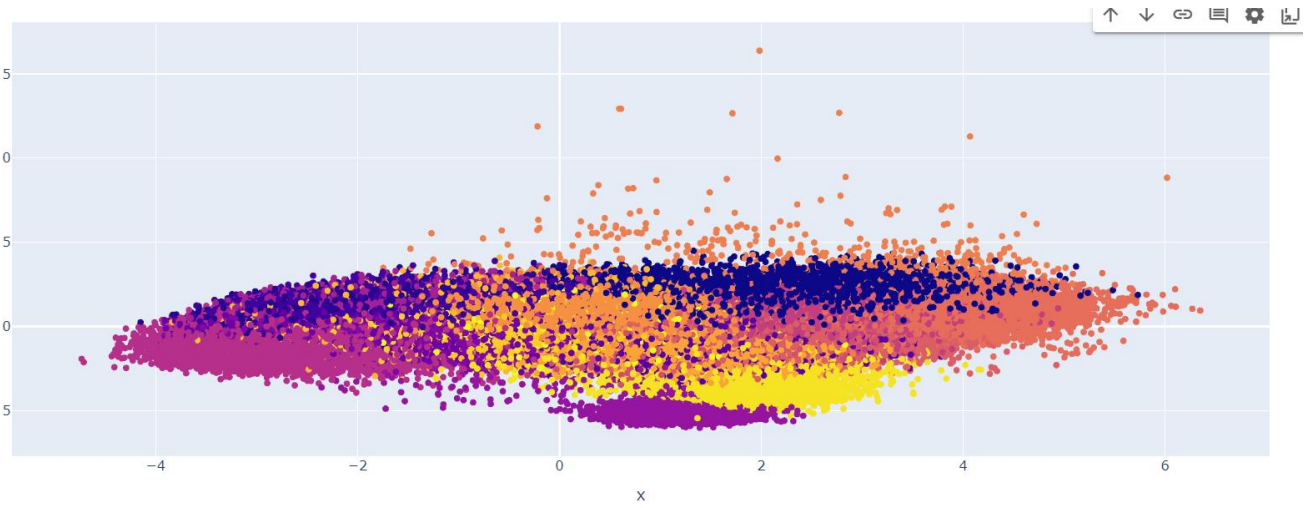
To speed up the computation process of t-SNE for high dimensional data, we apply PCA before t-SNE and combine both methods as in the following diagram.

PCA

```
from sklearn.decomposition import PCA

pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])
song_embedding = pca_pipeline.fit_transform(X)
projection = pd.DataFrame(columns=['x', 'y'], data=song_embedding)
projection['title'] = data['name']
projection['cluster'] = data['cluster_label']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'title'])
fig.show()
```



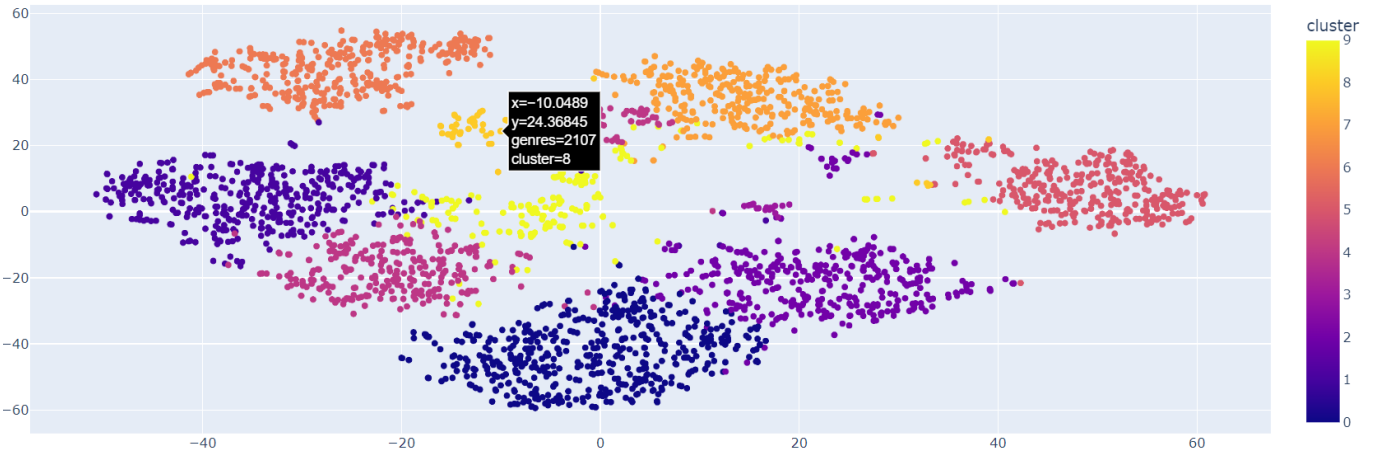
t-SNE

```
# Visualizing the Clusters with t-SNE

from sklearn.manifold import TSNE

tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2, verbose=1))])
genre_embedding = tsne_pipeline.fit_transform(X)
projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
projection['genres'] = genre_data['genres']
projection['cluster'] = genre_data['cluster']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'])
fig.show()
```



Building Recommender System

```
number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit', 'year',
               'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']
```

```
def get_song_data(name, data):
    try:
        return data[data['name'].str.lower() == name].iloc[0]

    except IndexError:
        return None
```

It takes a song name and a DataFrame as input, then attempts to find the song data in the DataFrame based on the provided name.

```
[175] # 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)
```

This code effectively calculates the mean vector of a list of songs based on their data in the provided DataFrame. This code plays a crucial role in representing the collective features of a list of songs to generate personalized song recommendations.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
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
)
```

By applying both MinMax scaling and standardization, the data undergoes a two-step transformation process:

- It is scaled to a predefined range using MinMax scaling.
- It is then standardized to have a mean of 0 and a standard deviation of 1 using standardization.

```
def recommend_songs( song_list, spotify_data, n_song=10):

    metadata_cols = ['name', 'year', 'artists']
    song_dict = flatten_dict_list(song_list)

    song_center = get_mean_vector(song_list, spotify_data)
    scaler = song_cluster_pipeline.steps[0][1]
    scaled_data = scaler.transform(spotify_data[number_cols])
    scaled_song_center = scaler.transform(song_center.reshape(1, -1))
    distances = cdist(scaled_song_center, scaled_data, 'cosine')
    index = list(np.argsort(distances)[:, :n_songs][0])

    rec_songs = spotify_data.iloc[index]
    rec_songs = rec_songs[~rec_songs['name'].isin(song_dict['name'])]
    return rec_songs[metadata_cols].to_dict(orient='records')
```

This function uses the average characteristics of the user-selected songs to find similar songs from the Spotify dataset and returns metadata for recommended song.

```
[176] 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
```

Flatten_dict_list :This function essentially combines the information from multiple dictionaries into a single dictionary, where each key aggregates the values from all dictionaries in the list. This is useful for various data transformation task.

FINAL OUTPUT:

Iteration 1:

```
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)
```

Iteration 2:

```
seed_songs = [
    {'name': 'Shape of You'},
    {'name': 'Despacito'},
    {'name': 'Perfect'}
]
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. She'll Leave You With A Smile - 2001 Version by ['George Strait'] (2001)
2. Te Mentiría by ['Gian Marco'] (1998)
3. This Christmas by ['Donny Hathaway'] (1990)
4. Don't Wanna Lose You Now by ['Backstreet Boys'] (1999)
5. Nebraska by ['moe.'] (1998)
6. Por Que Te Conocí by ['Los Temerarios'] (1998)
7. Seasons in the Sun by ['Westlife'] (1999)
8. Riding With Private Malone by ['David Ball'] (2001)
9. Always Be My Baby by ['Mariah Carey'] (1995)
10. It Ain't Me, Babe (with June Carter Cash) by ['Johnny Cash', 'June Carter Cash'] (2002)
11. To Be With You by ['Mr. Big'] (1996)
12. Lamberto Quintero by ['Antonio Aguilar'] (1996)
13. Como Decirte Adios by ['Los Yaguarú'] (1999)
14. Te Metiste En Mi Cama by ['Palomo'] (2002)
15. She'll Leave You With A Smile - Edit by ['George Strait'] (2004)
```

Iteration 3:

```
seed_songs = [
    {'name': 'Numb'},
    {'name': 'In the end'},
]
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. Savior by ['Rise Against'] (2008)
2. Stillborn by ['Black Label Society'] (2003)
3. Nothing Breaks Like a Heart (feat. Miley Cyrus) by ['Mark Ronson', 'Miley Cyrus'] (2018)
4. Crawling by ['Linkin Park'] (2000)
5. Infinity 2008 - Klaas Vocal Edit by ['Guru Josh Project', 'Klaas'] (2008)
6. Pushing Me Away by ['Linkin Park'] (2000)
7. Testify by ['Rage Against The Machine'] (1999)
8. Hayloft by ['Mother Mother'] (2008)
9. Heroes (we could be) by ['Alesso', 'Tove Lo'] (2015)
10. Nice To Meet Ya by ['Niall Horan'] (2019)
11. Call Me When You're Sober by ['Evanescence'] (2006)
12. Youth of the Nation by ['P.O.D.'] (2001)
13. Don't Stop Believin' by ['Journey'] (2001)
14. Happy Pills by ['Weathers'] (2016)
15. I Will Not Bow by ['Breaking Benjamin'] (2009)
```

Future Scope:

- Multimodal Recommendations: With the increasing availability of multimedia content e.g., images, videos & audio, recommendation systems can evolve to provide multimodal recommendations.

- Music Recommendation App: Application can be made and deployed for various platforms including mobile, web, smart speakers, APIs for third party developers.

---Thank you---