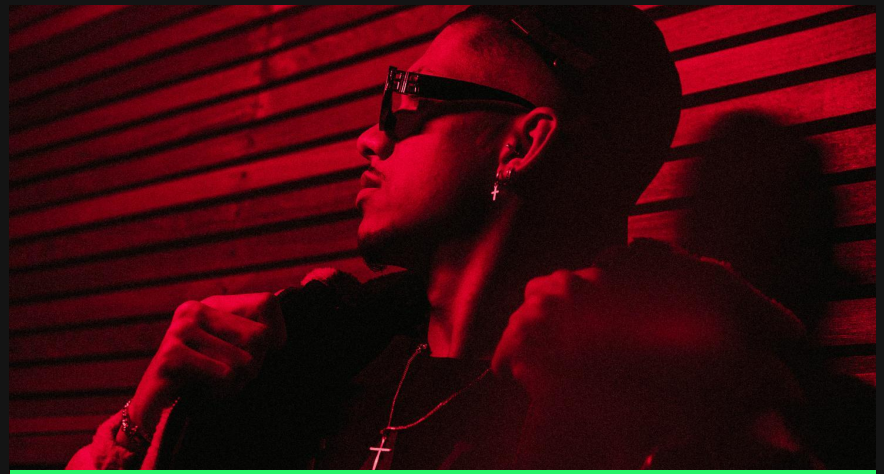
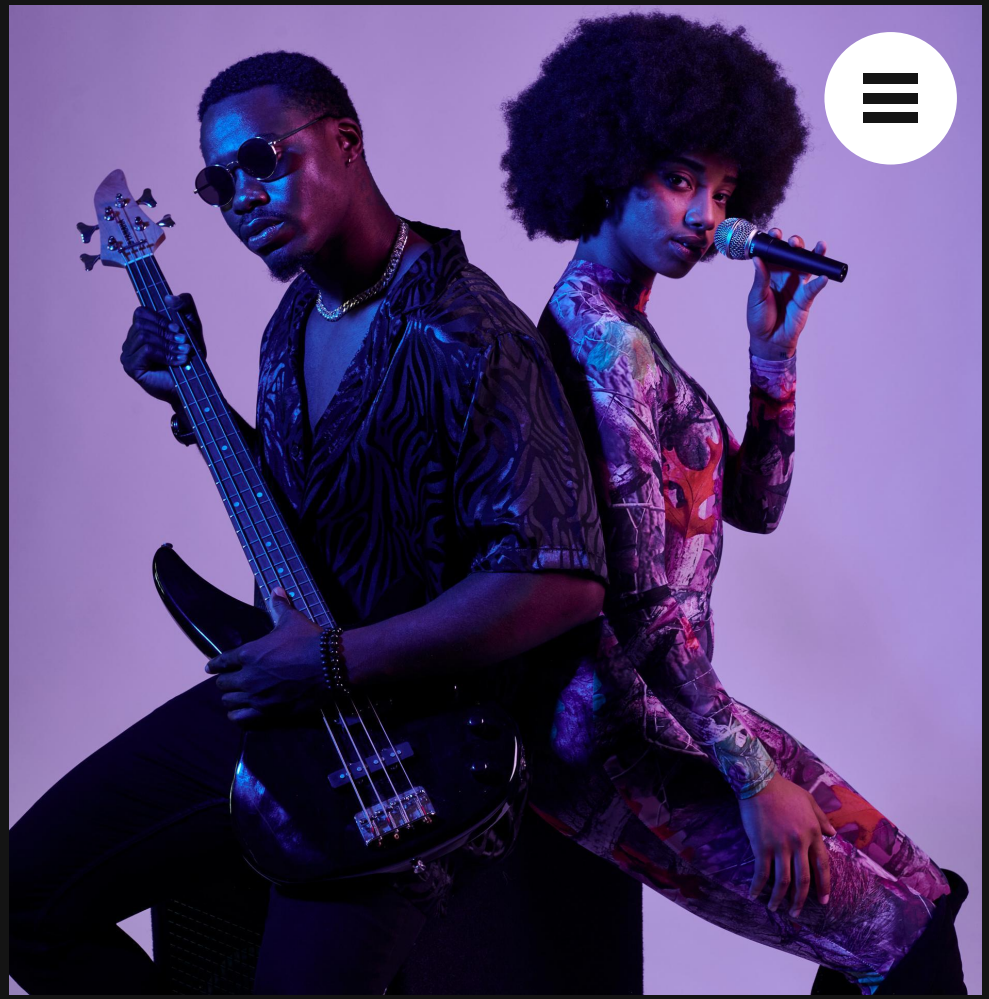


ENHANCING USER EXPERIENCE

SPOTIFY MUSIC

Made by: Janhavi T, Rushabh B,
Tanishk D, Snigdha S

NEXT



WELCOME

THE JOURNEY TO PERSONALIZED MUSIC

The project aims to design a robust music recommendation system using Spotify datasets and machine learning techniques to enhance the personalization of music suggestions for users.

Key Highlights:

- Explored Spotify audio features and metadata.
- Built a recommendation system based on content-based filtering.
- Integrated Spotify API to fetch real-time song data.

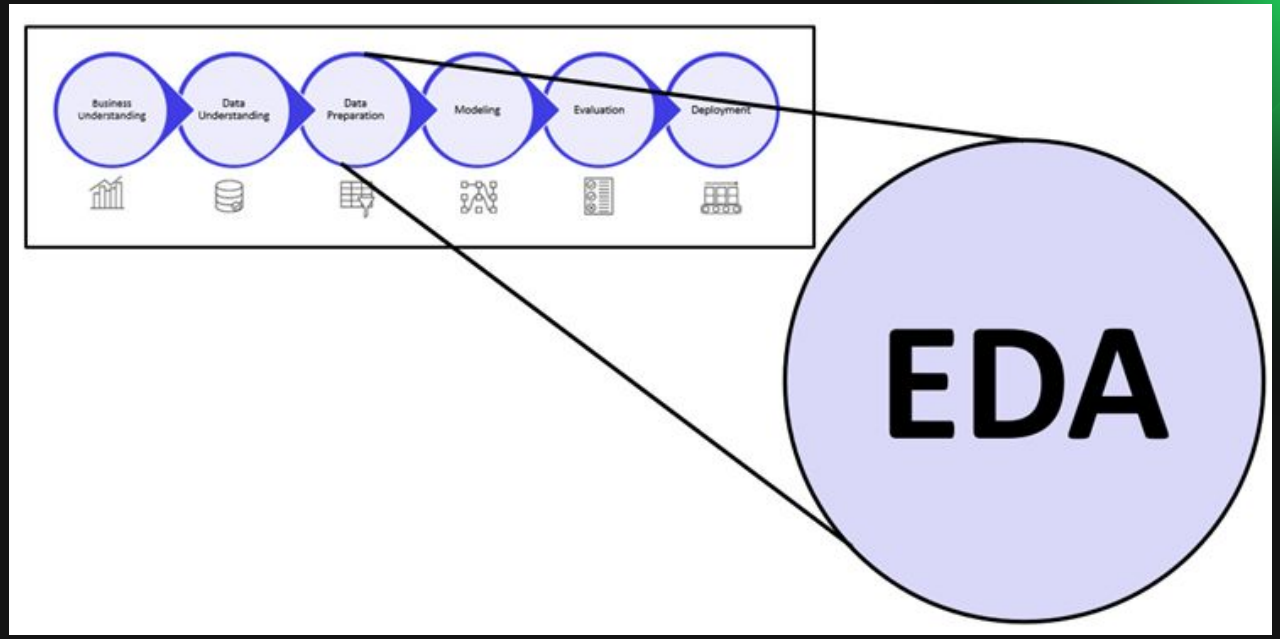
EXPLORING THE SPOTIFY DATASET

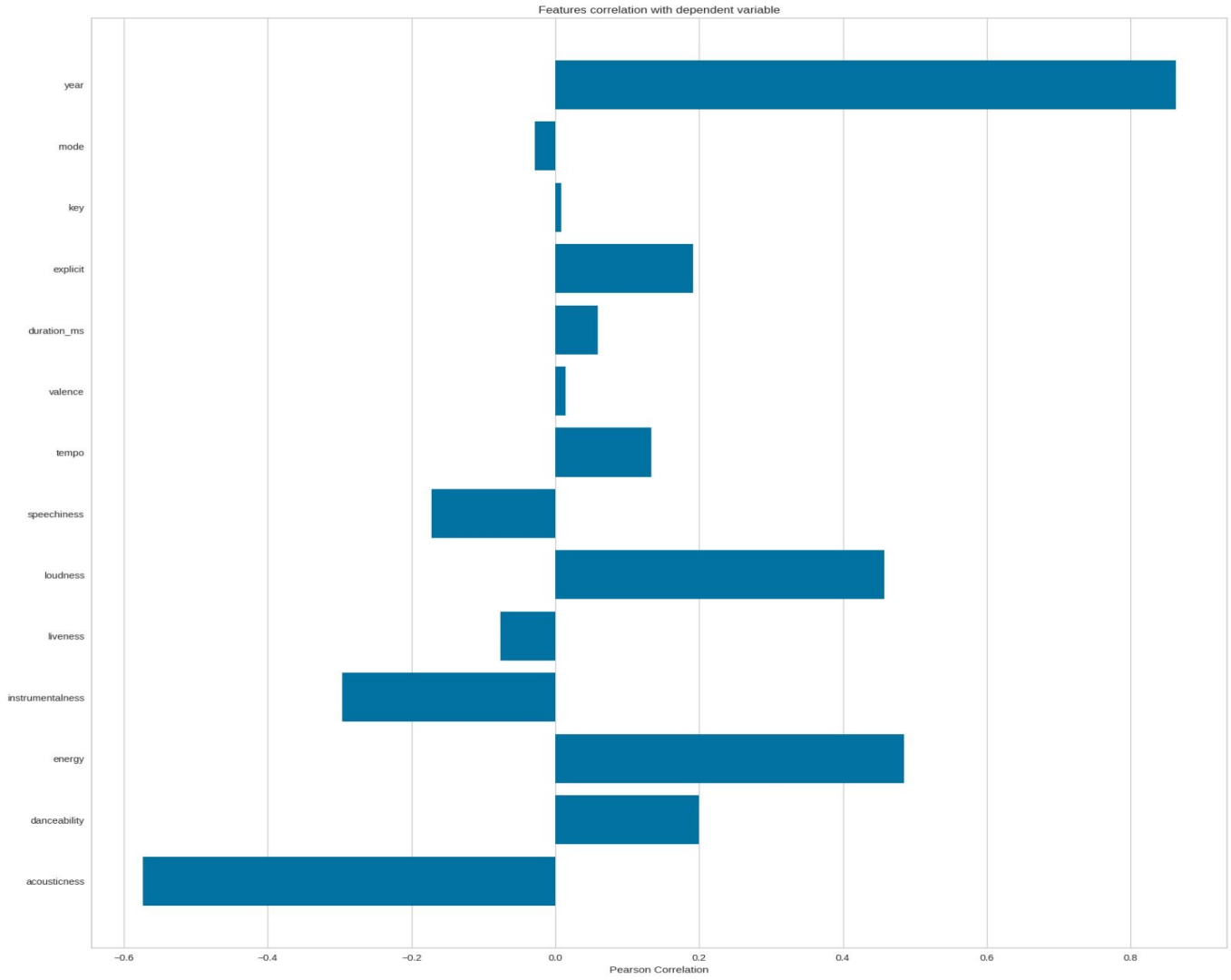
1. Data.csv: 170,653 entries with track-level details (e.g., valence, danceability, energy).
2. Data_by_genres.csv: 2,973 entries, summarizing audio features across genres.
3. Data_by_year.csv: 100 entries tracking audio trends over decades.

Key Features:

- Audio Features: Acousticness, energy, danceability, loudness, etc.
- Metadata: Song name, artist, popularity, release date.

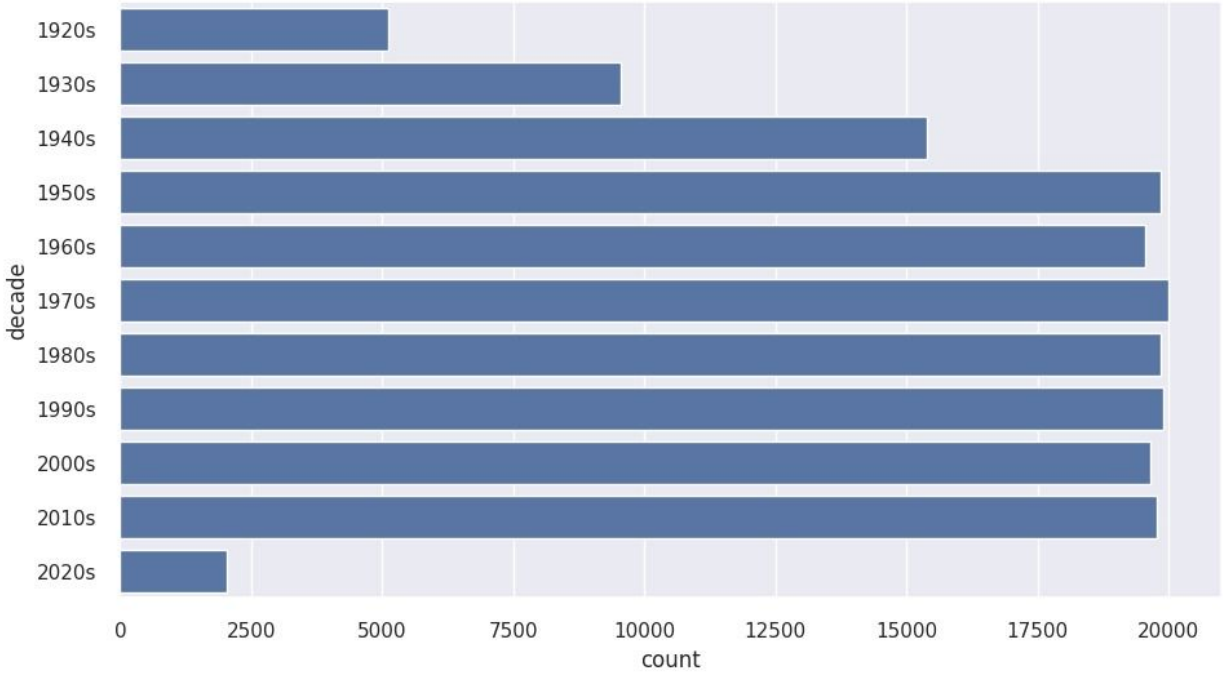
Exploratory data analysis- EDA





Pearson correlation coefficients of various features with the dependent variable

Song Counts by Decade



NYU

Popularity Trend Over Time

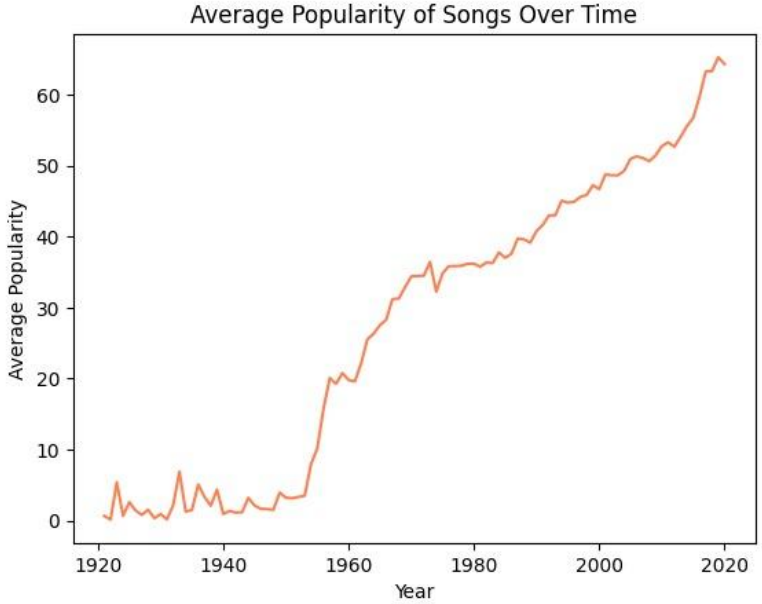
This line chart plots average song popularity by release year to observe trends over time.

Key Insights:

Increasing Popularity: From around the 2000s onward, there's a significant increase in the average popularity of songs. This could be due to changes in music production, digital accessibility, or shifts in listener preferences.

Peak in 2020s: The highest average popularity is seen in recent years, suggesting that users tend to engage more with newer music.

Relevance for Model: This trend suggests that recent songs should potentially be prioritized in recommendations, as users may have a preference for newer, popular music.



Top 10 Most Frequent Artists

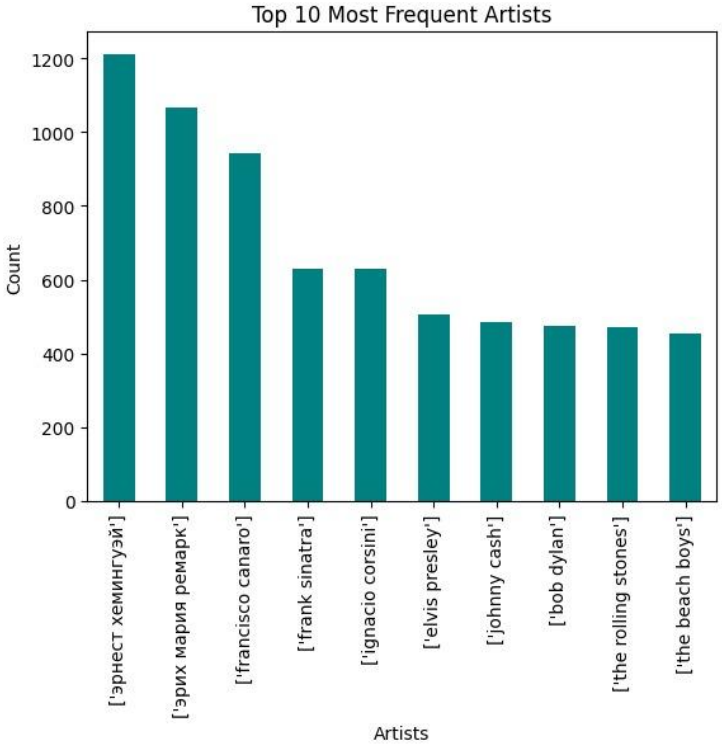
This bar chart shows the top 10 artists with the most songs in the dataset, indicating which artists appear frequently.

Key Insights:

Artists like Frank Sinatra and Elvis Presley are among the top entries, showing they have extensive catalogs on Spotify.

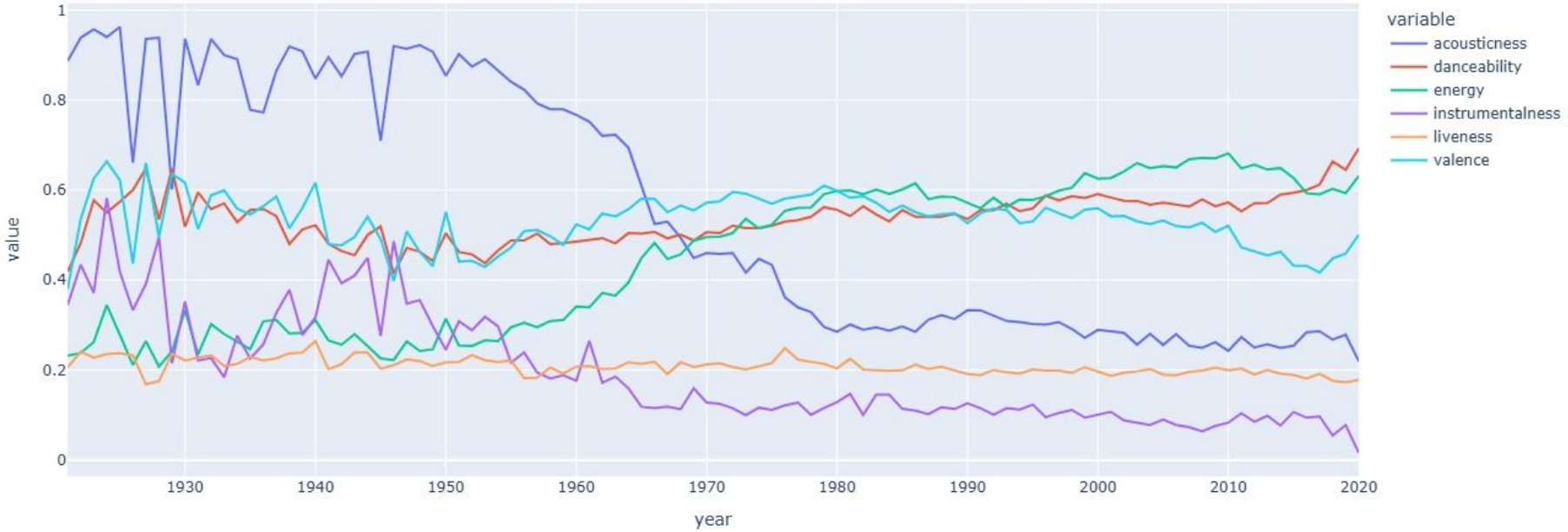
Overrepresentation Risk: Frequent artists could dominate recommendations if not balanced, potentially leading to bias toward popular or classic artists.

Relevance for Model: Knowing which artists are overrepresented helps in adjusting the recommendation model to avoid bias and ensure variety in suggestions.

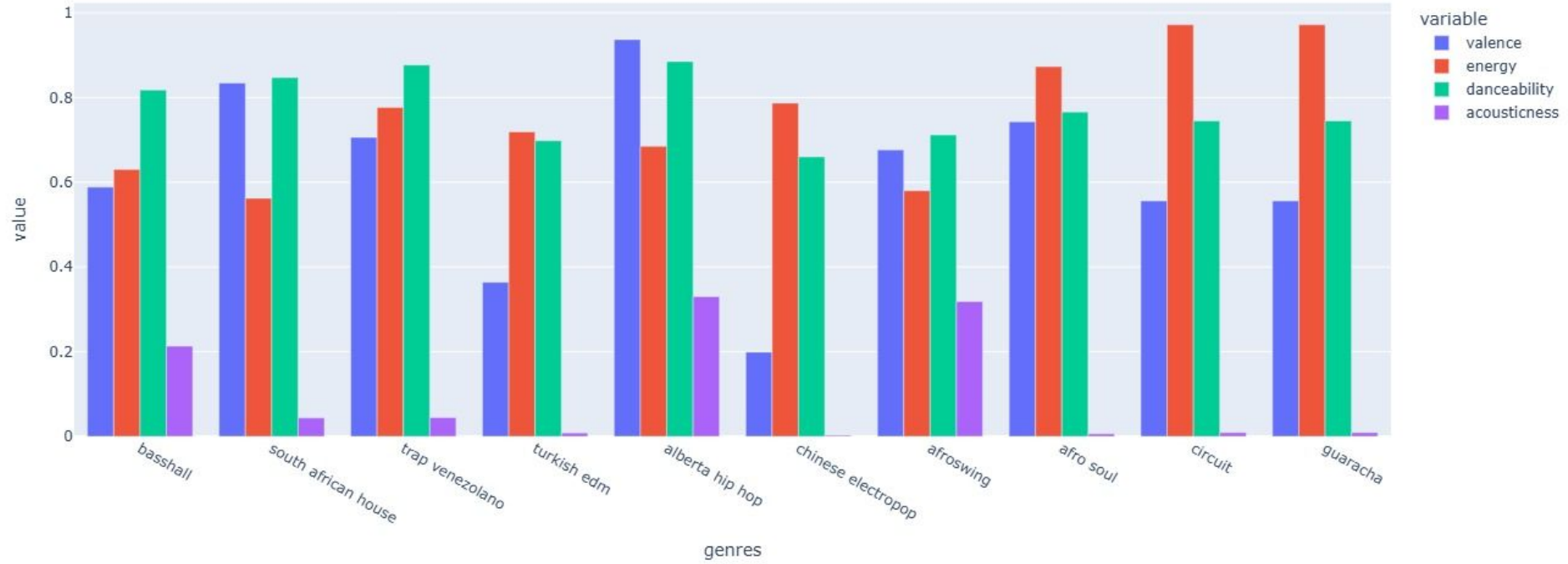


NYU

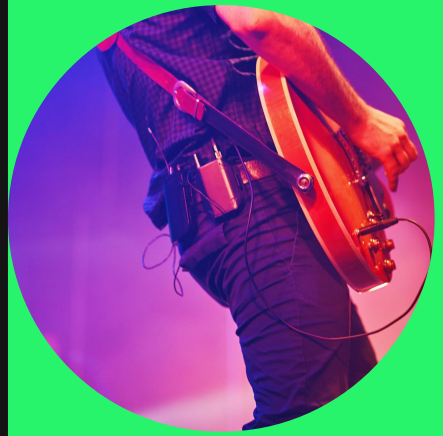
Temporal Trends of Features



Comparing the valence, energy, danceability, and Acousticness of various music genres



Our Modelling Approach



TOP 10 RECOMMENDED SONGS

Approach:
Content-based filtering using audio features.
Clustering songs based on feature similarity.
Steps:
Normalize features.
Cluster songs with K-Means.
Visualize clusters using PCA and t-SNE.
Tools: Scikit-learn, Spotify API.
Visualization:
PCA-based cluster scatter plot for songs.

Model Implementation

This code demonstrates clustering and visualization techniques applied to two datasets: genre data and song data.

It uses pipelines for streamlined preprocessing, clustering, and dimensionality reduction, allowing for efficient and reproducible workflows.

Clustering and Visualization of Genre Data

Clustering with K-Means:
Purpose: Group numerical features in the genre_data dataset into 10 clusters.

Result: Adds a new column, cluster, to the genre_data dataframe, representing the cluster label for each entry.

Visualizing Clusters with t-SNE:
Purpose: Reduce high-dimensional data to 2D for visualization while preserving cluster relationships.

Result: A 2D visualization that highlights clusters and their corresponding genres.

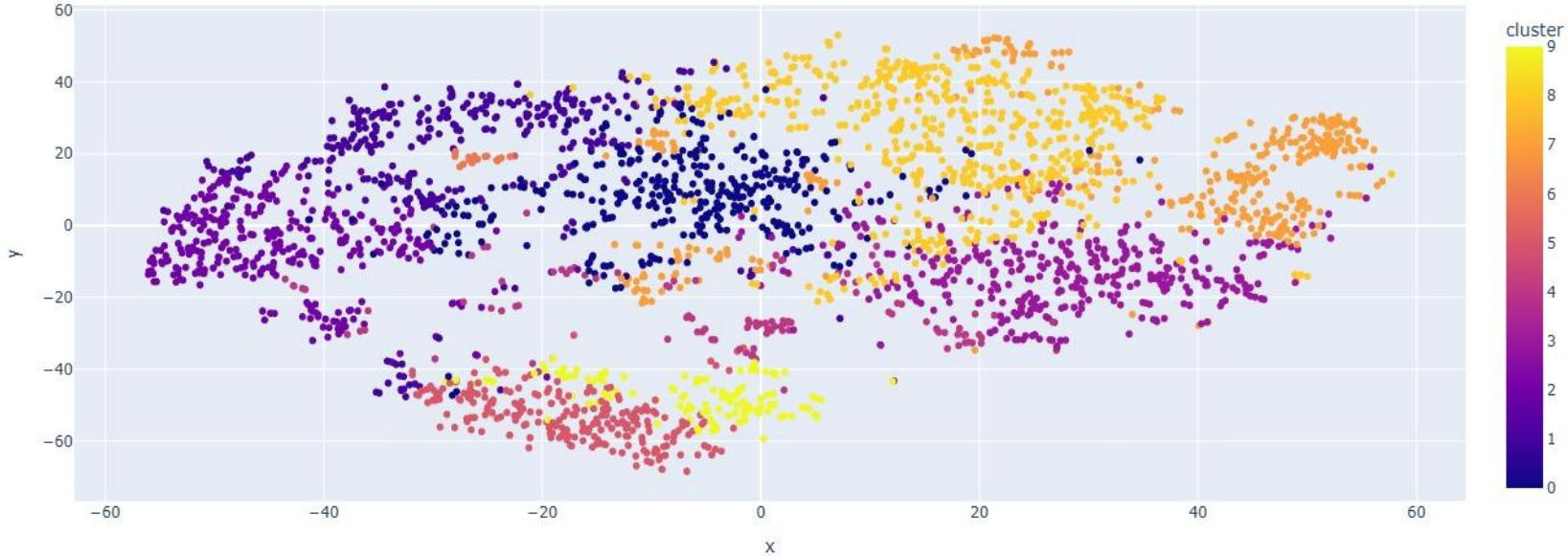
Clustering Genres with K-Means

```
[ ] 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)
```

```
# 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()
```

NYU

Visualizing the clusters with T-SNE



NYU

Clustering and Visualization of Song Data (Data.csv)

Clustering with K-Means:

Purpose: Group numerical features in the data.csv dataset into 20 clusters

Result: Adds a new column, cluster_label, to the data dataframe, representing the cluster label for each song.

```
#Clustering of Song Data
song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),('kmeans', KMeans(n_clusters=20,verbose=False))], verbose=False)
X = data.select_dtypes(np.number)
number_cols = list(X.columns)
song_cluster_pipeline.fit(X)
song_cluster_labels = song_cluster_pipeline.predict(X)
data['cluster_label'] = song_cluster_labels
```

Visualizing Clusters with PCA:

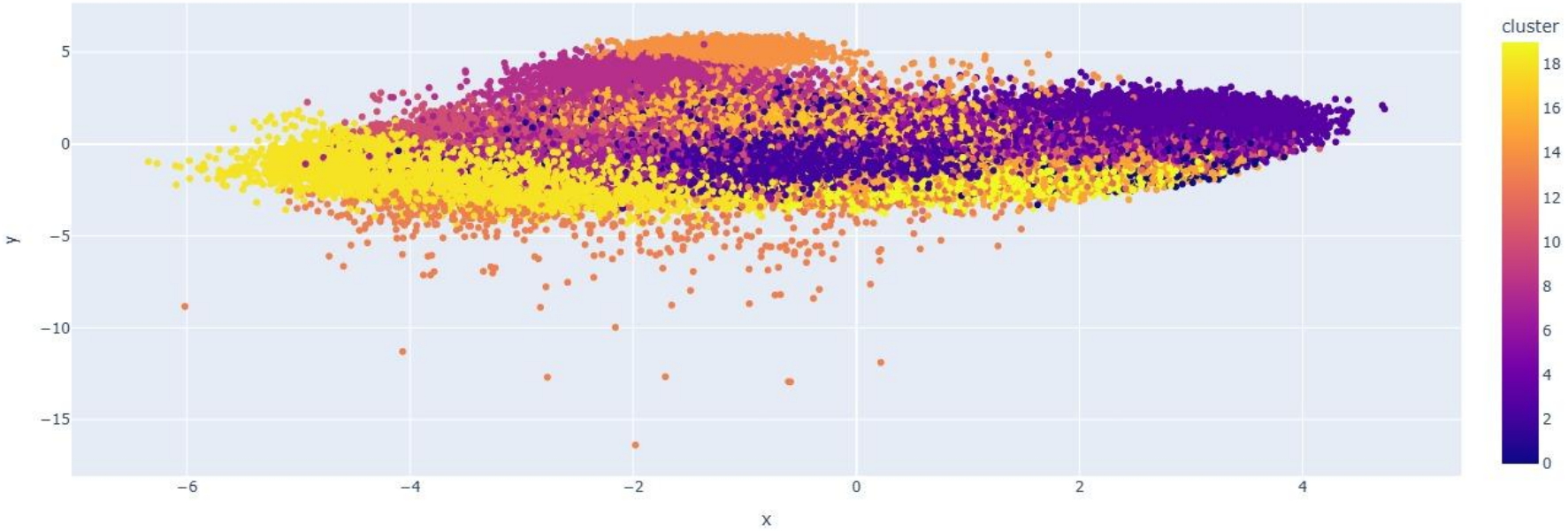
Purpose: Reduce high-dimensional data to 2D for visualization using PCA.

Result: A 2D visualization that shows clusters and their corresponding song titles.

```
# Visualizing the Clusters with 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()
```

NYU

Visualizing Clusters with PCA



NYU

Building Recommender System

Based on the analysis and visualizations, it's clear that similar genres tend to have data points that are located close to each other while similar types of songs are also clustered together.

This observation makes perfect sense. Similar genres will sound similar and will come from similar time periods while the same can be said for songs within those genres. We can use this idea to build a recommendation system by taking the data points of the songs a user has listened to and recommending songs corresponding to nearby data points.

Recommend Songs Based on Similarity:
This code takes a list of songs, calculates a feature-based average (mean vector), and recommends songs similar to the provided ones.

Purpose:
Recommends similar songs based on audio features and metadata using cosine similarity in the scaled feature space.

Recommending Songs Based on Similarity:

```python
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


def recommend_songs( song_list, spotify_data, n_songs=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')
```

```python
from collections import defaultdict
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist
import difflib

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


def get_song_data(song, spotify_data):

    try:
        song_data = spotify_data[(spotify_data['name'] == song['name'])
                                 & (spotify_data['year'] == song['year'])].iloc[0]
        return song_data

    except IndexError:
        return find_song(song['name'], song['year'])


def get_mean_vector(song_list, spotify_data):

    song_vectors = []

    for song in song_list:
        song_data = get_song_data(song, spotify_data)
        if song_data is None:
            print('Warning: {} does not exist in Spotify or in database'.format(song['name']))
            continue
        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)
```

Retry Mechanism for Fetching Song Data:

This code adds a try except mechanism to the get_song_data function for better error handling when a song is not immediately found in the dataset or the Spotify database.

Purpose:
To improve reliability by handling potential API delays or missing data in the dataset.

```python
import time

def get_song_data(song, spotify_data):
    try:
        song_data = spotify_data[(spotify_data['name'] == song['name']) & (spotify_data['year'] == song['year'])].iloc[0]
    except IndexError:
        try:
            # Retry the request after a brief delay
            time.sleep(2)
            song_data = spotify_data[(spotify_data['name'] == song['name']) & (spotify_data['year'] == song['year'])].iloc[0]
        except IndexError:
            print(f"Error: Could not find data for song '{song['name']}' ({song['year']})")
            song_data = None
    return song_data
```

Inputs and Outputs

```
recommend_songs([
                {'name': 'Antidote', 'year': 2015},
                {'name': 'See You Again (feat. Charlie Puth)', 'year': 2015}
                ],  data)
```

```
[{'name': 'Spicy (feat. Post Malone)',
  'year': 2020,
  'artists': "['Ty Dolla $ign', 'Post Malone']"},
 {'name': 'do re mi', 'year': 2017, 'artists': "['blackbear']"},
 {'name': 'Topanga', 'year': 2018, 'artists': "['Trippie Redd']"},
 {'name': 'Modern Loneliness', 'year': 2020, 'artists': "['Lauv']"},
 {'name': '...And To Those I Love, Thanks For Sticking Around',
  'year': 2020,
  'artists': "['$uicideBoy$']"},
 {'name': "Ballin' (with Roddy Ricch)",
  'year': 2019,
  'artists': "['Mustard', 'Roddy Ricch']"},
 {'name': 'Dead Eyes',
  'year': 2019,
  'artists': "['Promoting Sounds', 'Powfu', 'Ouse']"},
 {'name': 'GREECE (feat. Drake)',
  'year': 2020,
  'artists': "['DJ Khaled', 'Drake']"},
 {'name': 'Go Flex', 'year': 2016, 'artists': "['Post Malone']"},
 {'name': 'Pray For Me (with Kendrick Lamar)',
  'year': 2018,
  'artists': "['The Weeknd', 'Kendrick Lamar']"}]
```

input/output 1

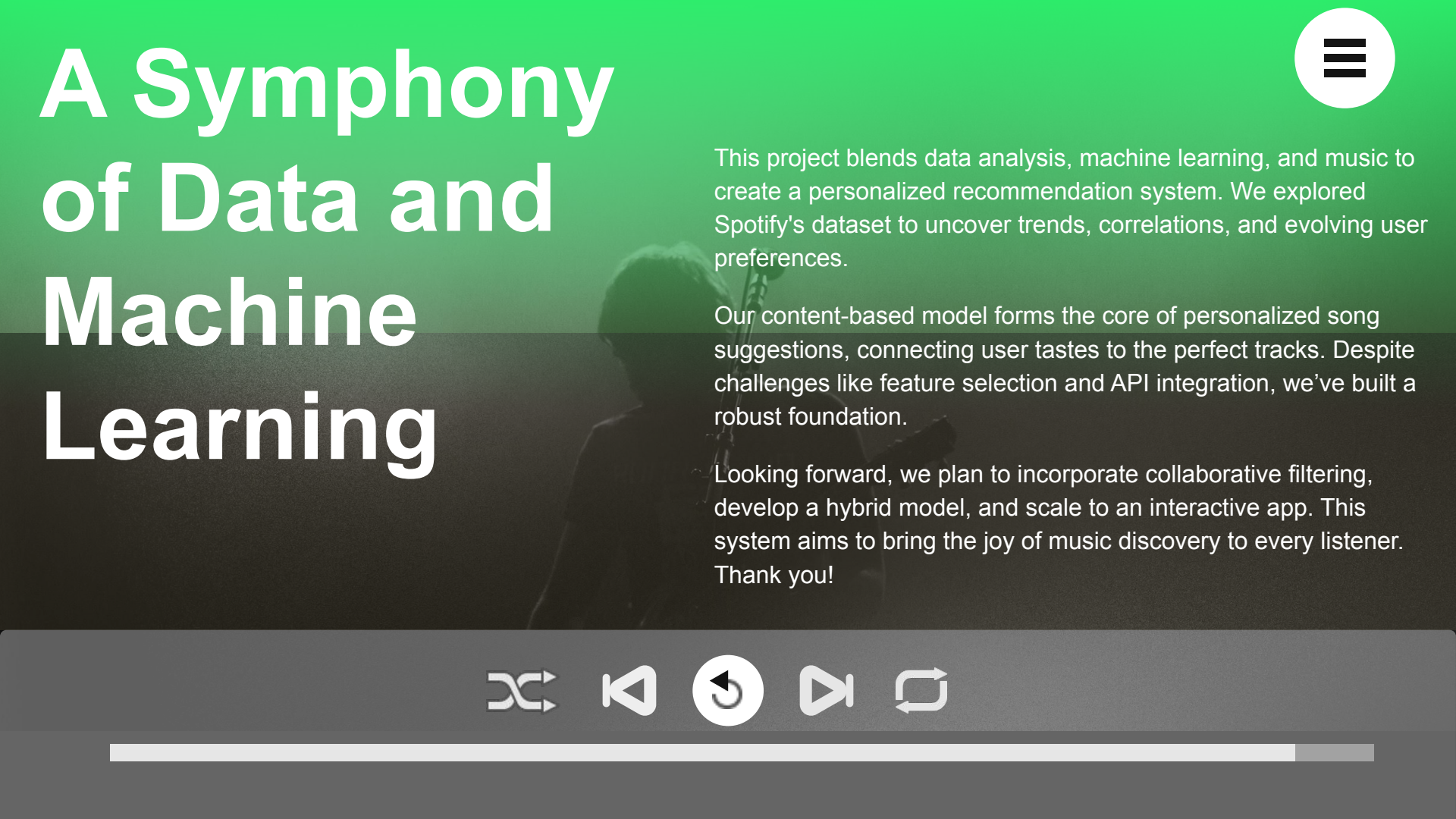
```
recommend_songs([
                {'name': 'Come As You Are', 'year':1991},
                {'name': 'Smells Like Teen Spirit', 'year': 1991},
                {'name': 'Lithium', 'year': 1992},
                {'name': 'All Apologies', 'year': 1993},
                ],  data)
```

```
[{'name': 'Hanging By A Moment', 'year': 2000, 'artists': "['Lifehouse']"},
 {'name': 'Kiss Me', 'year': 1997, 'artists': "['Sixpence None The Richer']"},
 {'name': "Breakfast At Tiffany's",
  'year': 1995,
  'artists': "['Deep Blue Something']"},
 {'name': 'Otherside', 'year': 1999, 'artists': "['Red Hot Chili Peppers']"},
 {'name': "It's Not Living (If It's Not With You)",
  'year': 2018,
  'artists': "['The 1975']"},
 {'name': 'No Excuses', 'year': 1994, 'artists': "['Alice In Chains']"},
 {'name': 'Wherever You Will Go', 'year': 2001, 'artists': "['The Calling']"},
 {'name': 'Ballbreaker', 'year': 1995, 'artists': "['AC/DC']"},
 {'name': 'Runaway (U & I)', 'year': 2015, 'artists': "['Galantis']"},
 {'name': "Club Can't Handle Me (feat. David Guetta)",
  'year': 2010,
  'artists': "['Flo Rida', 'David Guetta']"}]
```

input/output 2

NYU

A Symphony of Data and Machine Learning



This project blends data analysis, machine learning, and music to create a personalized recommendation system. We explored Spotify's dataset to uncover trends, correlations, and evolving user preferences.

Our content-based model forms the core of personalized song suggestions, connecting user tastes to the perfect tracks. Despite challenges like feature selection and API integration, we've built a robust foundation.

Looking forward, we plan to incorporate collaborative filtering, develop a hybrid model, and scale to an interactive app. This system aims to bring the joy of music discovery to every listener. Thank you!