Music Recommendation System

DTSC 710

Selina Narain, Neelam Boywah, Zoya Haq

Professor Wu

Why a Music Recommendation System?

- Can be designed to create personalized music suggestions for the listeners based on various different factors like preferences, listening history, and more.
- Users can discover new songs and artists they may not have otherwise come across.
- Improves listening experience to be enjoyable and engaging.

Methodologies

K Means Clustering

- Unsupervised machine learning algorithm that clusters data points based on similarity

Sklearn Pipeline

- Combining multiple steps to be cross-validated together, including setting different parameters

PCA (Principal Component Analysis)

- Reduce dimensionality for a more narrow analysis and visualization

Standard Scaler

- Used to standardize the features of the dataset

Grid Search CV

- Used for hyperparameter tuning to find the clusters

Spotify Datasets

Visualizations	ML Models	Recommendation System
❖ data_by_year.csv ❖ data_by_genre.csv ❖ data_by_artist.csv	❖ data.csv ❖ data_by_genre.csv ❖ data_by_artist.csv	❖ Spotify Web API

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

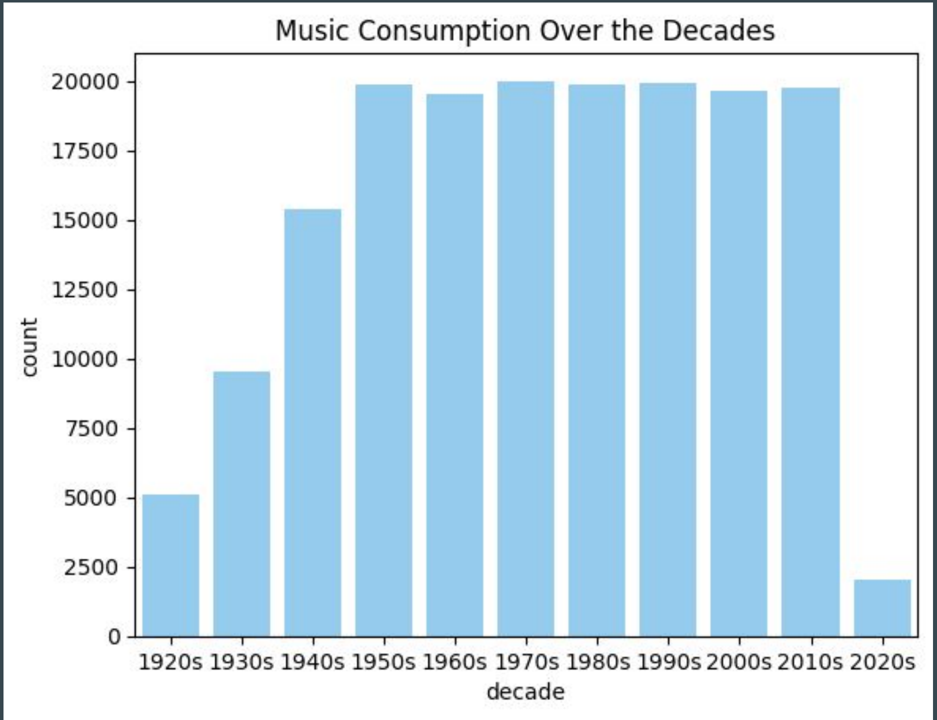
	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness	loudness	mode	name	popularity	release_date	speechiness	tempo
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.665	-20.096	1	Piano Concerto No. 3 in D Minor, Op. 30: III. ...	4	1921	0.0366	80.954
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.160	-12.441	1	Clancy Lowered the Boom	5	1921	0.4150	60.936
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.101	-14.850	1	Gati Bali	5	1921	0.0339	110.339
3	0.1650	1921	0.967	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	0.000028	5	0.381	-9.316	1	Danny Boy	3	1921	0.0354	100.109
4	0.2530	1921	0.957	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	0.000002	3	0.229	-10.096	1	When Irish Eyes Are Smiling	2	1921	0.0380	101.665

Data Preprocessing

- Visualizations
- Feature Selection
  - Hand pick features that best suits the baseline models
- Pipeline used for preprocessing steps
  - Standardizing Data for K-means Clustering for Genres and Songs
- PCA dimension reductionality for K-means Clustering for Songs
  - Reduced to 2 principal components

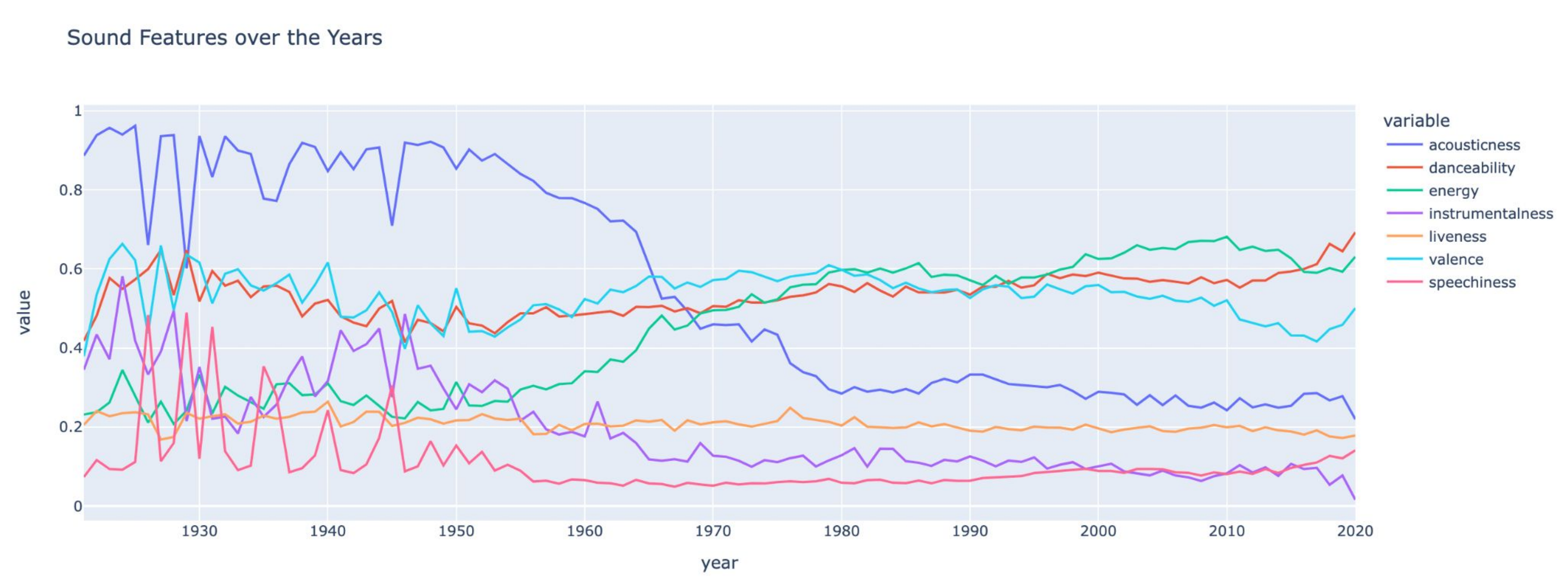
Visualizations

Music Consumption over the Decades

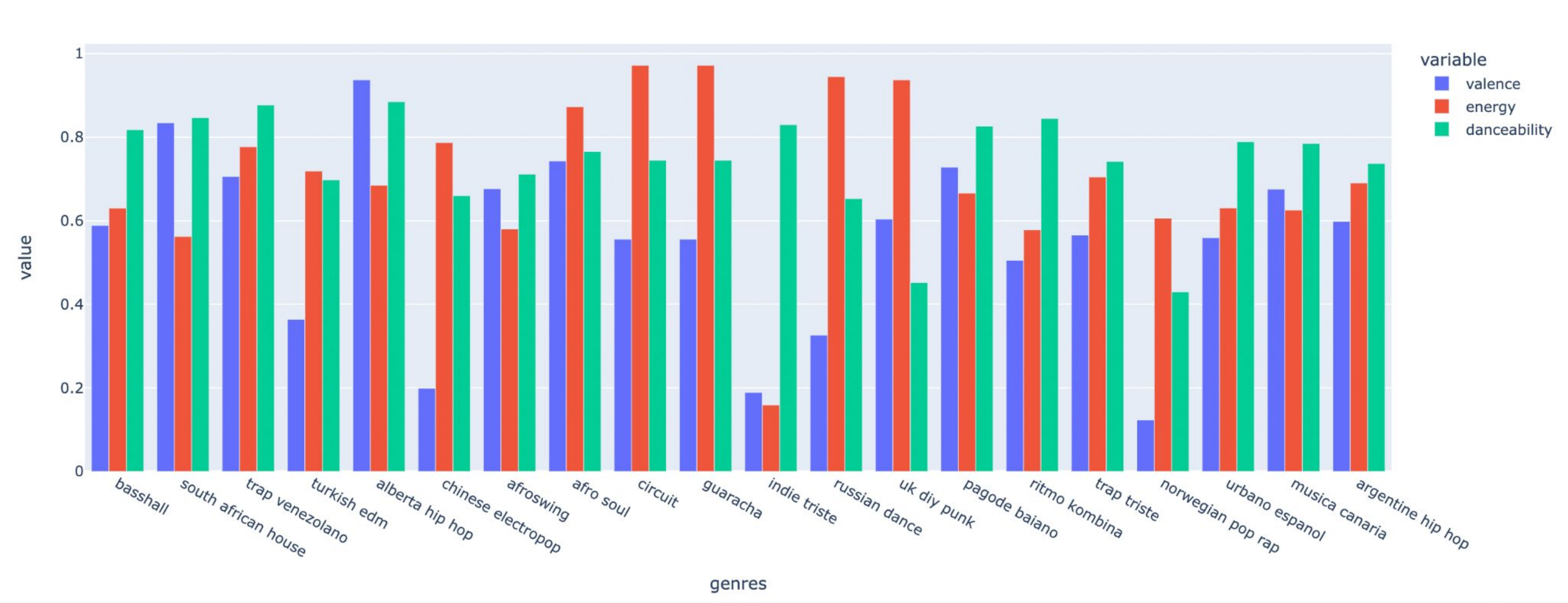


- Steady increase from the 1920s
- Consistent from 1950s to 2010s
- Data for 2020s is skewed as this is the current decade we are in

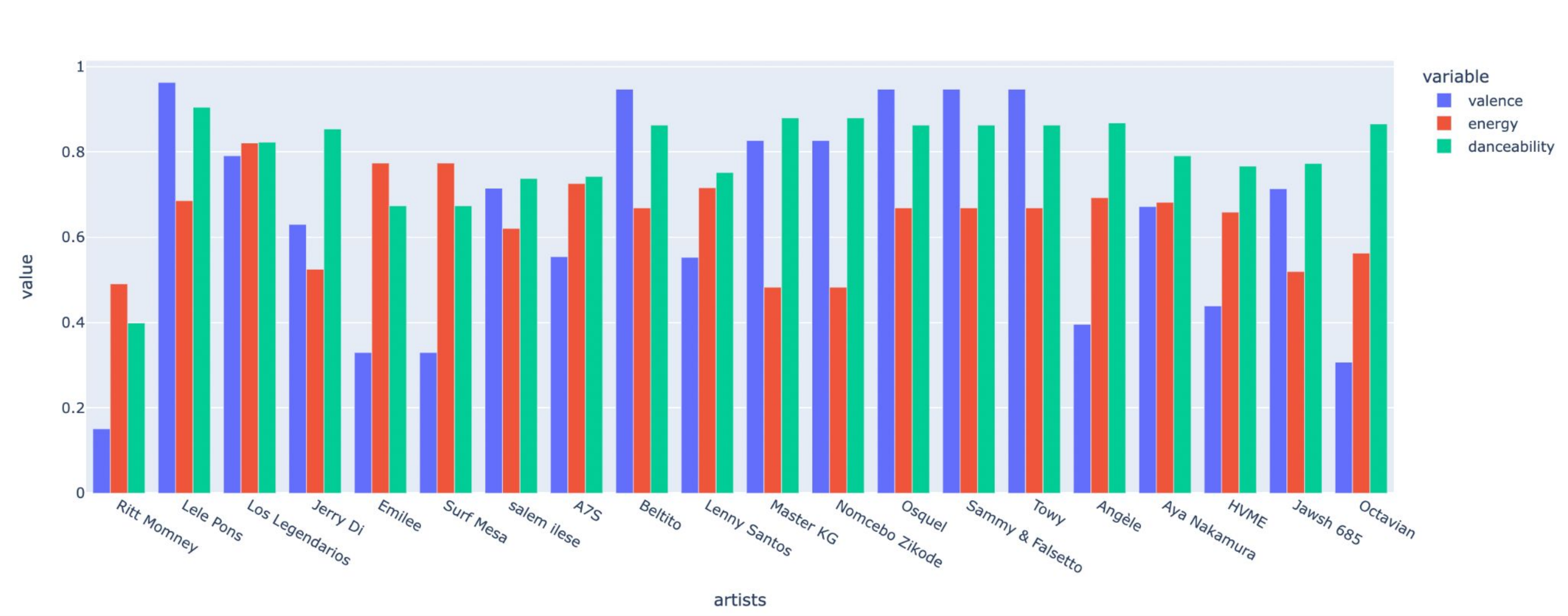
Sound Features over the Years



Top 20 Genres



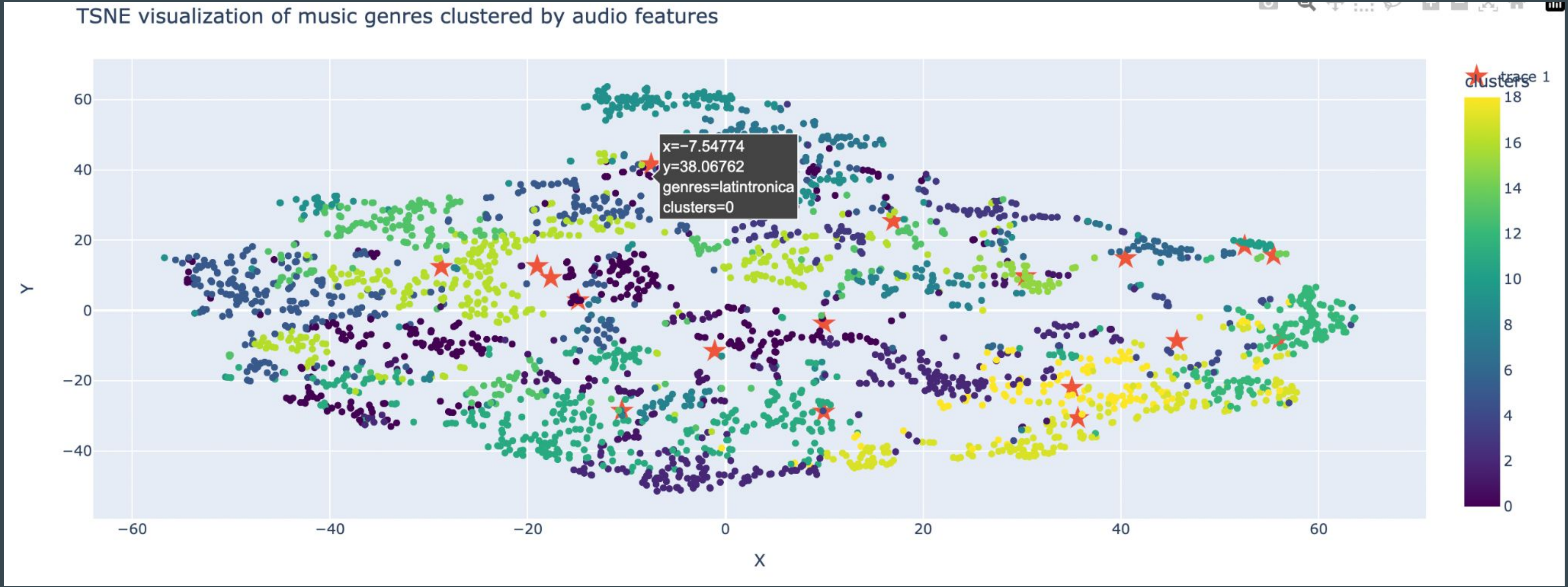
Top 20 Artists



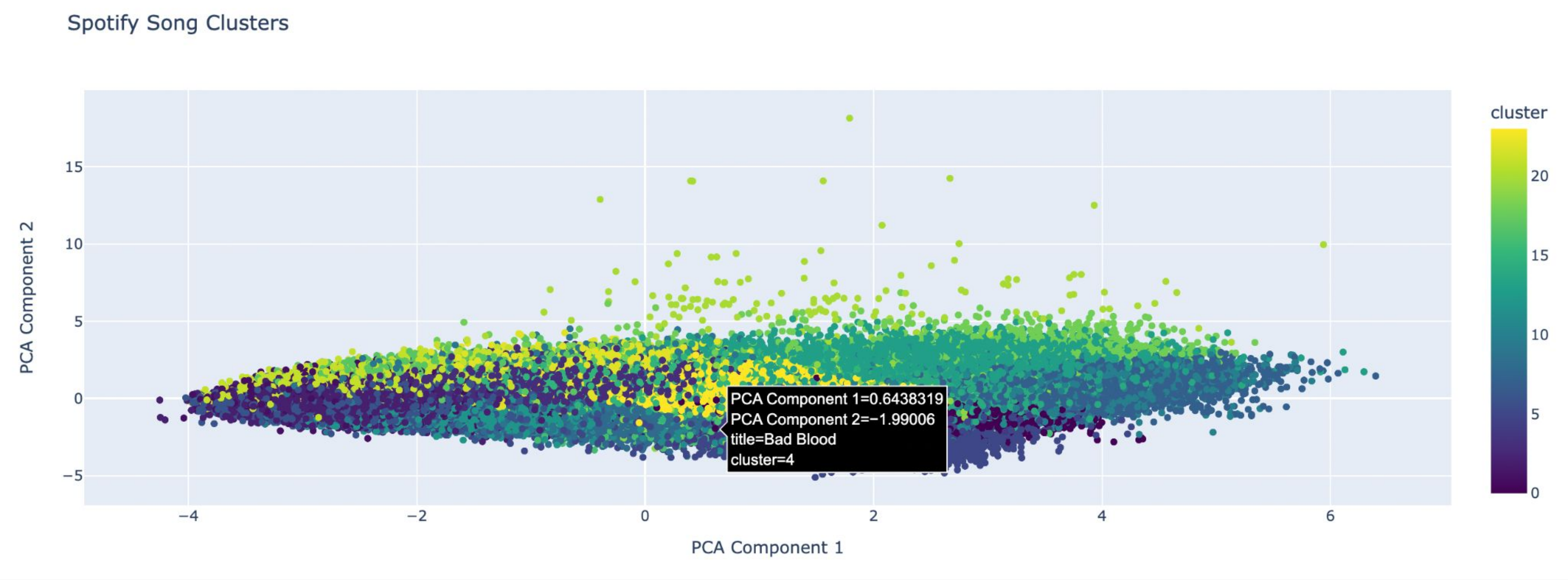
Baseline Models

K-Means for Clustering Genres	K-Means & PCA for Clustering Songs
❖ Pipeline - Scaling data, K-Means Clustering ❖ Grid Search ➢ Hyperparameter tuning for clusters: range 1-20 ➢ Best Parameters: {‘kmeans__n_clusters’: 19} ❖ Pipeline - Scaling data, K-Means Clustering based on Grid Search ❖ Fit K-Means Clustering Model ❖ Visualize the clusters with centroids using TSNE	❖ Pipeline - Scaling data, K-Means Clustering ❖ Grid Search ➢ Hyperparameter tuning for clusters: range 1-25 ➢ Best Parameters: {‘kmeans__n_clusters’: 24} ❖ Pipeline - Scaling data, K-Means Clustering based on Grid Search ❖ Pipeline - Scaling data and performing PCA ❖ Fit K-Means Clustering PCA Model ❖ Visualize Clusters using plotly

Music Genres Clusters



Songs Clusters



K-Means Songs Clustering

K-Means Clustering for Songs

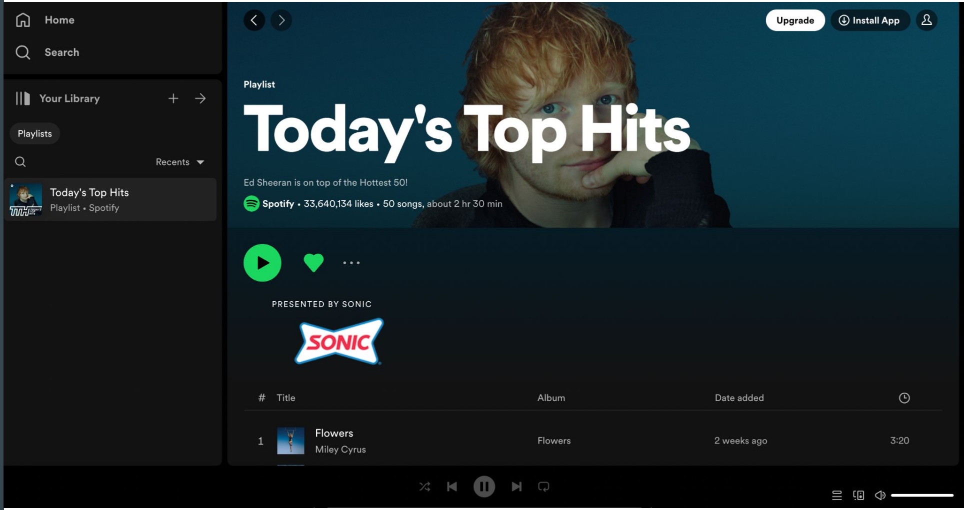
```python
feature_names = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness', 'liveness',
                 'loudness',  'mode', 'popularity', 'speechiness', 'tempo']

X = data[feature_names]

# Create pipeline
songs_cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans())])

# Define parameter grid
param_grid = {'kmeans__n_clusters': range(1, 25)}

# Create grid search object
grid_search = GridSearchCV(songs_cluster_pipeline, param_grid, cv=5)

# Fit grid search
grid_search.fit(X)

# Print best parameters
print("Best parameters:", grid_search.best_params_)
```

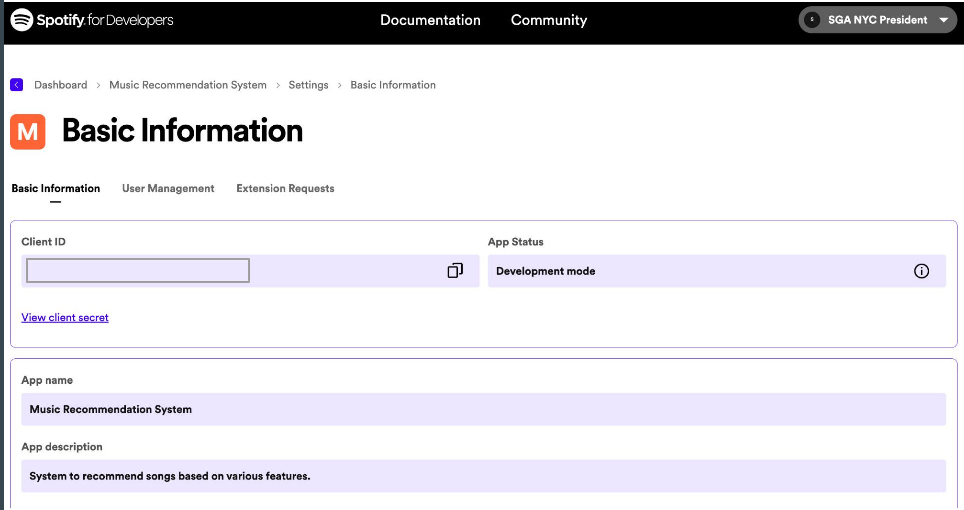
```
Best parameters: {'kmeans__n_clusters': 24}
```

```python
[ ] # Perform clustering on songs data

    #Pipeline - scaling data, performing k means clustering
    song_cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=24, verbose=False))], verbose=False)

    #X Features: valence, year, acousticness, danceability, duration_ms, energy, instrumentalness, liveness, loudness,  mode, popularity, speechiness, tempo

    #Fit clustering model
    song_cluster_pipeline.fit(X)

    #Predict model -> new column 'clusters'
    song_cluster_labels = song_cluster_pipeline.predict(X)

    #Adding cluster_label column to data
    data['cluster_label'] = song_cluster_labels
```

```python
[ ] #Pipeline - scaling data, performing PCA
    pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])

    #Transform X into a 2-dimensional embedded space
    song_transformed = pca_pipeline.fit_transform(X)
```

Spotipy Python Client
Spotify Web API





Implementation

- Using baseline models to develop a recommendation system
- Recommendation system uses Spotify Web API to pull metadata and identify Spotify's catalog for songs
- Specified features to be used to recommend songs
- Functions:
  - Finding song details given its name and release year
  - Retrieve information about a specific song from Spotify API
  - Calculating the mean audio features across a list of songs
    - Retrieving the audio feature data for each song
    - Averaging the feature values across all songs
  - Converting a list of dictionaries with the same keys into a flattened dictionary
  - Recommendations by taking in a list of songs and a Spotify dataset that contains information about various songs, and returns a list of recommended songs based on the input list using cosine distance as a similarity metric

Getting Songs Function

```
[ ]  # Function to find song details given its name and release year
     def get_song(name, year):
         song_details_data = defaultdict() # Create an empty defaultdict to store song details

         # Search for the song on Spotify using track name and release year
         song_results = sp.search(q= 'track: {} year: {}'.format(name,year), limit=1)

         # Check if the search returned any results
         if song_results['tracks']['items'] == []:
             return None

         # Get details of the first search result (which is assumed to be the song we're looking for)
         song_results = song_results['tracks']['items'][0]
         track_id = song_results['id']
         audio_features = sp.audio_features(track_id)[0]

         # Extract relevant song details and store them in the defaultdict
         song_details_data['name'] = [name]
         song_details_data['year'] = [year]
         song_details_data['explicit'] = [int(song_results['explicit'])]
         song_details_data['duration_ms'] = [song_results['duration_ms']]
         song_details_data['popularity'] = [song_results['popularity']]

         for key, value in audio_features.items():
             song_details_data[key] = value

         # Convert the defaultdict to a Pandas DataFrame and return it
         return pd.DataFrame(song_details_data)
```

Song Recommendation Function

```
[ ]  def songs_recommendation(song_list, spotify_data, n_songs=5):
         #List of metadata columns we want to return for the recommended songs
         metadata_cols = ['name', 'year', 'artists', 'popularity']
         #Flattened version of the input song list where each song is represented as a dictionary
         song_dict = flatten_dict_list(song_list)

         #Mean vector of the input song list, calculated using get_mean_val()
         song_mean_vector = get_mean_val(song_list, spotify_data)

         #Scaler used in song_cluster_pipeline
         scaler = song_cluster_pipeline.steps[0][1]

         #Scaled data of all songs in the spotify_data DataFrame
         scaled_data = scaler.transform(spotify_data[features])
         scaled_song_mean_vector = scaler.transform(song_mean_vector.reshape(1, -1))

         #Matrix of cosine distances between scaled_song_mean_vector and scaled_data
         distances = cdist(scaled_song_mean_vector, scaled_data, 'cosine')

         #Top n_songs songs in spotify_data that are closest to song_mean_vector based on cosine distance
         index = list(np.argsort(distances)[:, :n_songs][0])

         #DataFrame of recommended songs based
         song_recs = spotify_data.iloc[index]
         song_recs = song_recs[~song_recs['name'].isin(song_dict['name'])]

         #Returned dictionary with recommended song and its corresponding metadata
         return song_recs[metadata_cols].to_dict(orient='records')
```

Results

```
recommend_songs([{'name': 'Flowers', 'year':2023}], data)

[{'name': 'Circles',
  'year': 2019,
  'artists': "['Post Malone']",
  'popularity': 89},
 {'name': 'Take You Dancing',
  'year': 2020,
  'artists': "['Jason Derulo']",
  'popularity': 92},
 {'name': 'Stay Gold', 'year': 2020, 'artists': "['BTS']", 'popularity': 80},
 {'name': 'Raise Your Glass',
  'year': 2010,
  'artists': "['P!nk']",
  'popularity': 77},
 {'name': 'The Man',
  'year': 2019,
  'artists': "['Taylor Swift']",
  'popularity': 79}]
```

- Input a specific song name and year
- Recommends by returning a list of 5 recommended songs based on the input
- Provides the recommended song's name, release year, arist, and popularity rate

References

Maharshi Pandya. "Spotify Tracks Dataset." Kaggle, 22 Oct. 2022, https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset.

Mavani, Vatsal. "Spotify Dataset." Kaggle, 17 Dec. 2021, https://www.kaggle.com/datasets/vatsalmavani/spotify-dataset.

Samoshyn, Andrii. "Dataset of Songs in Spotify." Kaggle, 6 Dec. 2020, https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify.

Schedl, Markus, et al. "Current Challenges and Visions in Music Recommender Systems Research - International Journal of Multimedia Information Retrieval." SpringerLink, Springer London, 5 Apr. 2018, https://link.springer.com/article/10.1007/s13735-018-0154-2#:~:text=A%20number%20of%20approaches%20have,domain%20recommendation%2C%20and%20active%20learning.

"User Guide: Contents." *Scikit*, scikit-learn.org/stable/user_guide.html.

Thank you!