Spotify Music Clustering and Recommender System

https://github.com/sfs-projects/ml

Objective

- The objective of this analysis is to use clustering algorithms to partition songs based on their audio features and develop a recommender system to suggest similar songs to the user based on the selected clusters.
- The benefits of this analysis include enabling music streaming services to provide personalized recommendations to users, which could lead to increased user engagement and customer satisfaction.

Description

- Dataset source: Spotify API
- The dataset used in this project was obtained through the Spotify API and consists of song features such as tempo, loudness, danceability, and energy. The dataset's attributes are described in detail, including information on the artist, track name, album name, popularity, release date, and musical characteristics.

Columns and description:

- ➔ **artist** 'string' – Name of the artist
- ➔ **track_name** 'string' – Name of the song
- ➔ **album_name** 'string' – Name of the album
- ➔ **popularity** 'integer' **–**
- ➔ **release_date** 'date' –
- ➔ **acousticness** 'float' -- A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
- ➔ **danceability** 'float' -- Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
- ➔ **duration_ms** 'integer' -- The duration of the track in milliseconds.
- ➔ **energy** 'float' -- Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
- ➔ **instrumentalness** 'float' -- Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
- ➔ **key '**integer' -- The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
- ➔ **liveness** 'float' -- Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
- ➔ **loudness** 'float' -- The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.
- ➔ **mode** 'integer' -- Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
- ➔ **speechiness** 'float' -- Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
- ➔ **tempo** 'float' -- The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
- ➔ **time_signature** 'integer' -- An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4".
- ➔ **valence** 'float' -- A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Data exploration, data cleaning

Steps included:

- Renaming columns
- Removing null values duplicate rows
- Numerical scaling
- Plotting histograms for numerical columns
- Plotting correlations heatmap
- Plotting Silhouette score, David Bouldin index, Elbow method, dendrograms to understand the optimal nr of clusters
- Plotting bars or radar graphs to better view the differences between clusters

```
df_no_dupl = dataset.drop_duplicates(["artist", "track_name"]).reset_index(drop=True)
df_no_dupl.dropna(inplace=True, how="any")
df_no_dupl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1589 entries, 0 to 1589
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   artist            1589 non-null   object
 1   track_name        1589 non-null   object
 2   album_name        1589 non-null   object
 3   popularity        1589 non-null   int64
 4   release_date      1589 non-null   object
 5   duration_ms       1589 non-null   int64
 6   acousticness      1589 non-null   float64
 7   danceability      1589 non-null   float64
 8   energy            1589 non-null   float64
 9   instrumentalness  1589 non-null   float64
 10  key               1589 non-null   int64
 11  liveness          1589 non-null   float64
 12  loudness          1589 non-null   float64
 13  mode              1589 non-null   int64
 14  speechiness       1589 non-null   float64
 15  tempo             1589 non-null   float64
 16  time_signature    1589 non-null   int64
 17  valence           1589 non-null   float64
dtypes: float64(9), int64(5), object(4)
memory usage: 235.9+ KB
```

- Excluding columns like 'popularity' or 'duration_ms' which would be irrelevant; also categorical columns like 'time_signature, 'key' and 'mode' would be excluded.

```
final_cols = ["artist", "track_name", "album_name"]
audio_features_cols = [
    "acousticness",
    "danceability",
    "energy",
    "instrumentalness",
    "liveness",
    "loudness",
    "speechiness",
    "tempo",
    "valence",
]

print(len(final_cols), len(audio_features_cols), len(df_no_dupl.columns))
print("Columns used in the ML models:", audio_features_cols)
print(
    "Columns not used in the ML models:",
    df_no_dupl.columns.difference(audio_features_cols).to_list(),
)
```

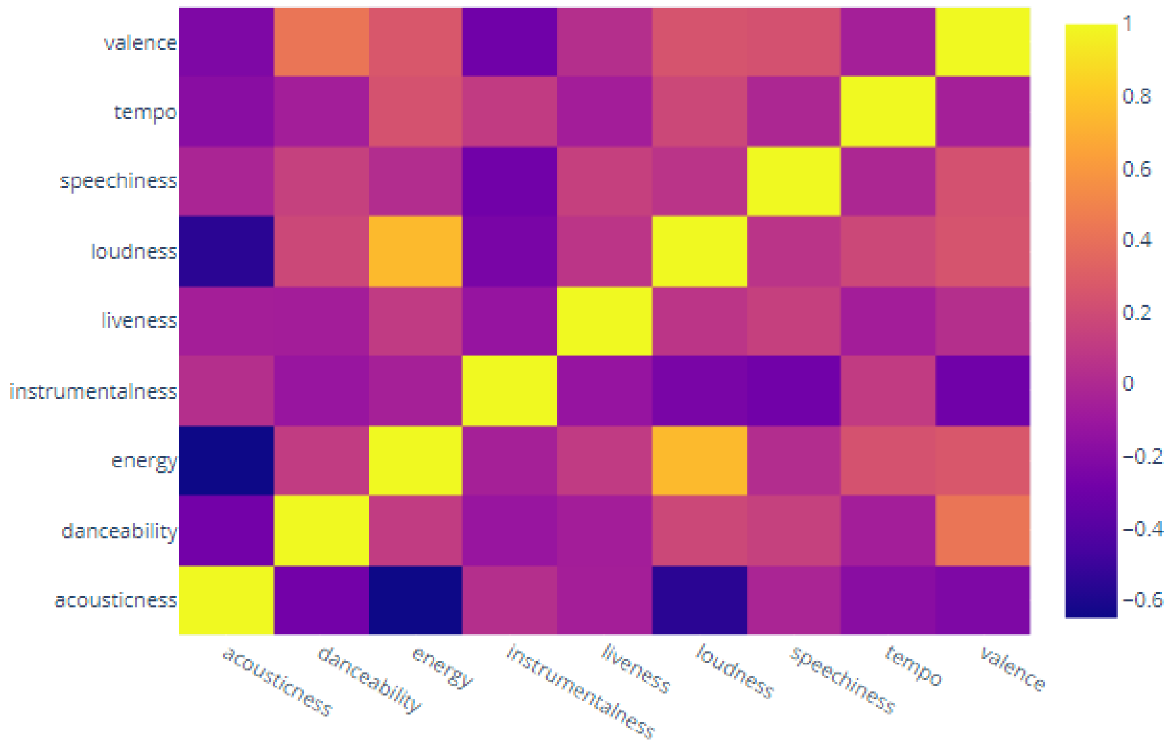
```
3 9 18
Columns used in the ML models: ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'valence']
Columns not used in the ML models: ['album_name', 'artist', 'duration_ms', 'key', 'mode', 'popularity', 'release_date', 'time_signature', 'track_name']
```

Correlations

- The x-axis and y-axis represent the same set of audio features, with each square in the heatmap representing the correlation between a pair of features.
- A high positive correlation (close to 1) between two features means that they tend to increase or decrease together, while a high negative correlation (close to -1) means that they tend to have opposite effects.
- A low correlation (close to 0) means that the two features are not strongly related to each other.
- energy vs loudness, high positive correlation = 0.74, meaning if one increases the other one changes in the same direction
- acousticness vs loudness, high negative correlation = -0.55, quite normal as loudness is smaller the higher the chance the track is acoustic
- danceability vs valence, medium positive correlation = 0.42, a higher valence value means a more positive track which makes it more danceable, rather than a depressing, sad one

Heatmap for feature correlations

```
fig_heatmap = go.Figure(
    data=go.Heatmap(
        z=corr_matrix,
        x=audio_features_cols,
        y=audio_features_cols,
    )
)

white_margins = 15
size_ = 700
fig_heatmap.update_layout(
    margin=dict(t=white_margins, r=white_margins, b=white_margins, l=white_margins),
    width=size_,
    height=size_ * 0.65,
    autosize=True,
)

fig_heatmap.show()
```



Scaling the numerical variables for better use on clustering algorithms. This way all features will have the same weight.

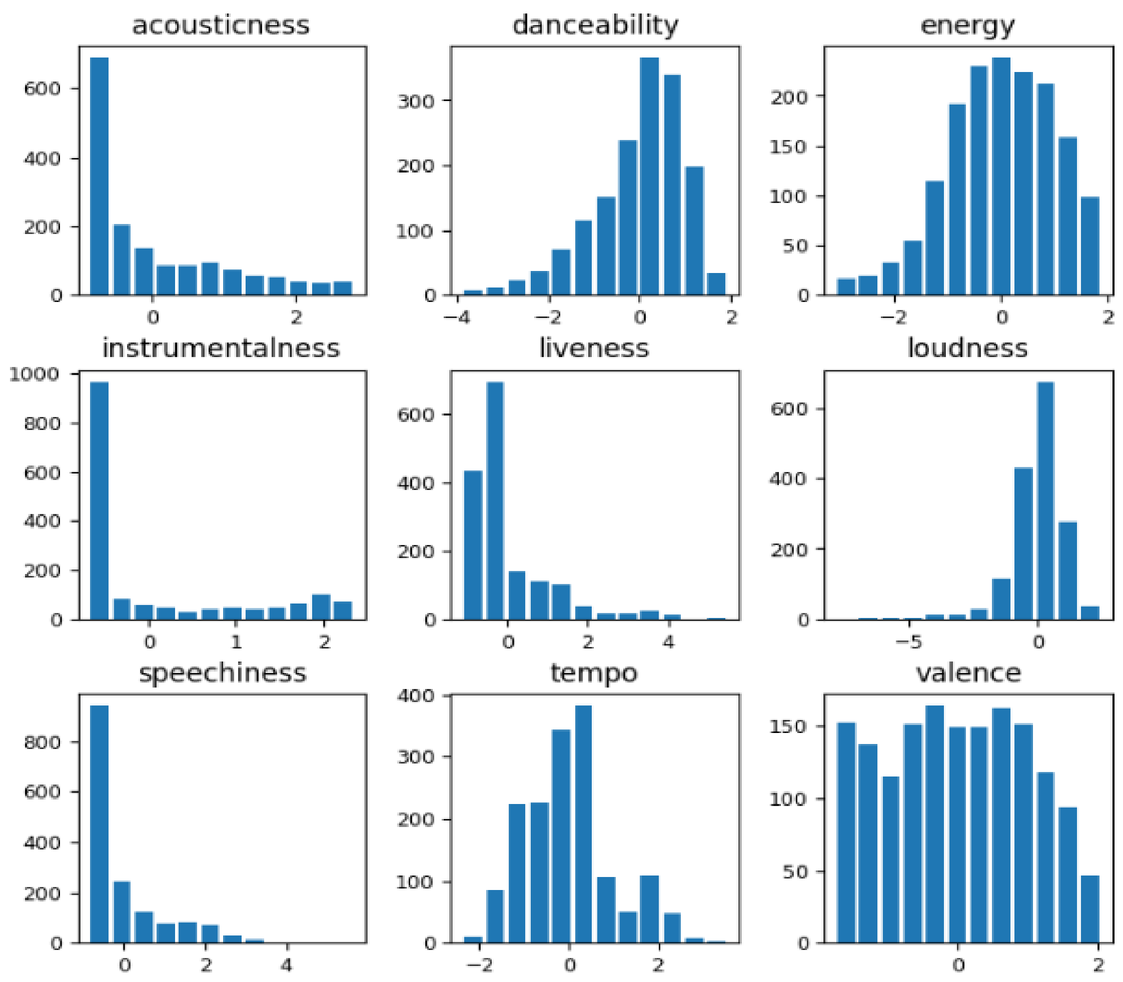
```
scaler = StandardScaler()
X_s = scaler.fit_transform(X_array)
X_s.shape
```

```
(1589, 9)
```

```
X_scaled_df = pd.DataFrame(X_s, columns=audio_features_cols)
round(X_scaled_df.describe(), 4)
```

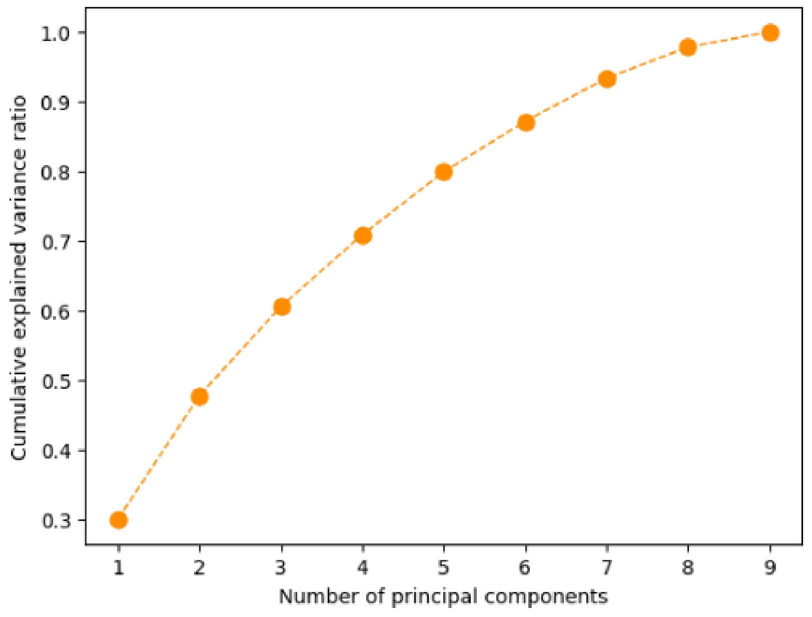
	acousticness	danceability	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence
count	1589.0000	1589.0000	1589.0000	1589.0000	1589.0000	1589.0000	1589.0000	1589.0000	1589.0000
mean	-0.0000	-0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000	-0.0000	0.0000
std	1.0003	1.0003	1.0003	1.0003	1.0003	1.0003	1.0003	1.0003	1.0003
min	-0.8904	-3.8881	-3.1049	-0.6895	-1.1031	-7.8686	-0.8481	-2.3903	-1.7260
25%	-0.8130	-0.5135	-0.6805	-0.6894	-0.5731	-0.4566	-0.6651	-0.7589	-0.8015
50%	-0.4401	0.1975	0.0535	-0.6460	-0.4066	0.1336	-0.4530	0.0110	-0.0012
75%	0.6092	0.7275	0.7925	0.6548	0.1581	0.6094	0.2555	0.3389	0.8070
max	2.8122	1.9426	1.8835	2.3556	5.4721	2.4457	5.6608	3.5575	2.0412

Histograms



Perform PCA

- PCA can help reduce the dimensionality of the data by identifying the most important features that contribute the most to the variance in the data.



```
def get_num_components(cmp_evr, variance_threshold=0.90):
    cumulative_evr = np.cumsum([e[1] for e in cmp_evr])
    num_components = np.where(cumulative_evr >= variance_threshold)[0][0] + 1
    actual_explained = round(cumulative_evr[num_components - 1] * 100, 2)

    print(
        f"{num_components} components explain {actual_explained}% of the variance. {len(cumulative_evr)} original componenents."
    )
    return num_components


optimal_n_components = get_num_components(cmp_evr)
```

```
7 components explain 93.35% of the variance. 9 original componenents.
```

- Decided to use 7 components, which would be more than enough to use for modeling. These explain more than 90% of the variance.

Models

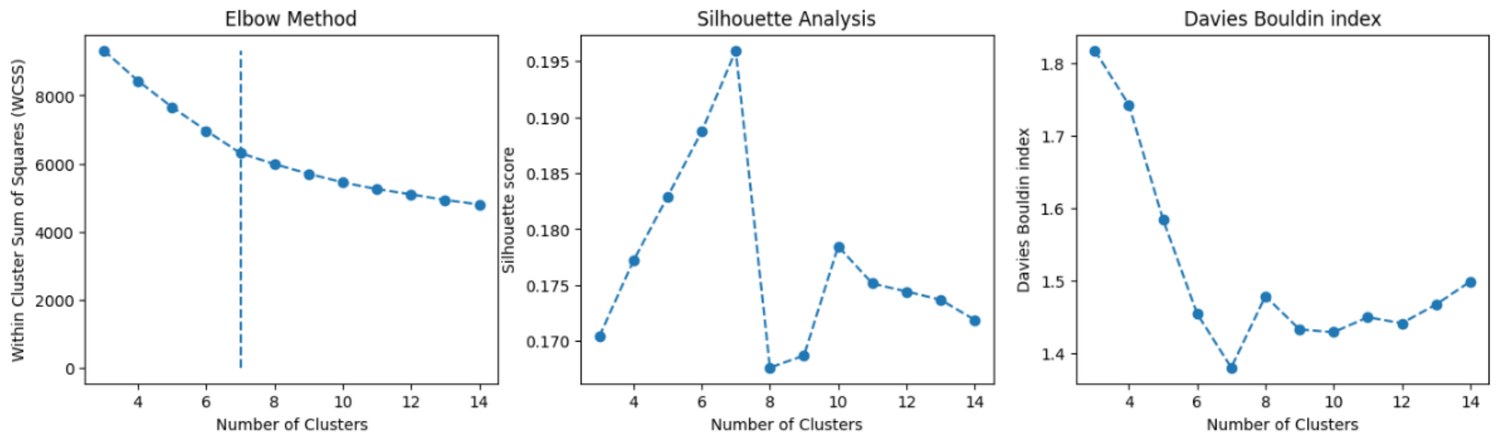
K-Means

Silhouette score, David Bouldin index and the Elbow method

Started by getting the optimal number of clusters for this model and we’ve used 3 different methods to acquire it. Silhouette score, David Bouldin index and the Elbow method.

```
elbow_clusters, silhouette_clusters, davies_bouldin_clusters = find_optimal_clusters(X_pca, min_clusters, max_clusters, (16, 4))
```

```
Recommended number of clusters from the elbow method: 7
Recommended number of clusters from the silhouette score: 7
Recommended number of clusters from the Davies Bouldin index: 7
```



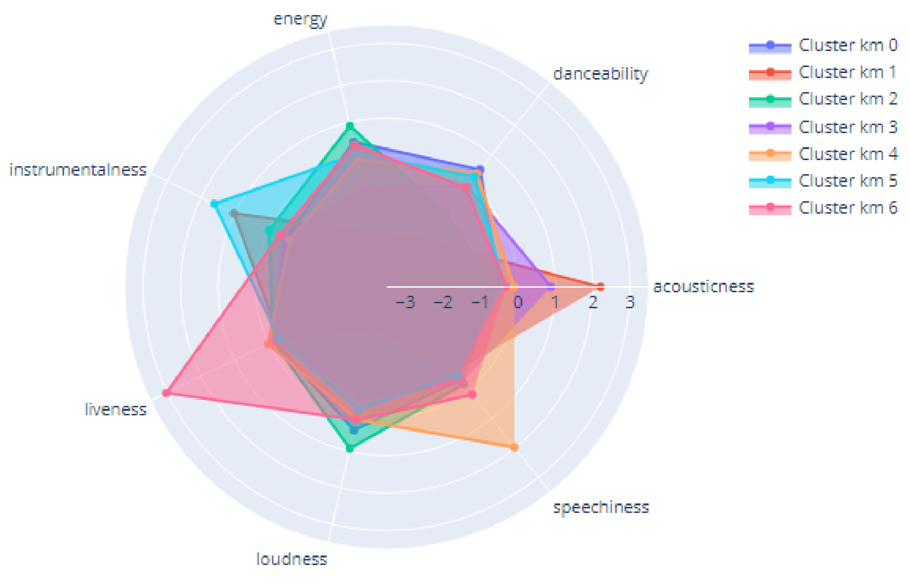
- Fitting the model once decided on 7 clusters.

```python
kmeans = KMeans(
    n_clusters=best_km_clusters,
    init="k-means++",
    random_state=random_state,
    n_init=10,
    max_iter=300,
)
kmeans.fit(X_pca)

X_scaled_df_pca_km = X_scaled_df[retained_features]
X_scaled_df_pca_km["clusters_km"] = kmeans.labels_
X_scaled_df_pca_km.head()
```

	acousticness	danceability	energy	instrumentalness	liveness	loudness	speechiness	clusters_km
0	-0.672724	-0.823773	1.265117	2.147364	-0.699076	0.642478	-0.745796	2
1	1.126501	-1.780373	0.194297	-0.676840	-0.442789	0.385026	-0.695121	3
2	2.127518	-2.866242	-1.892041	-0.689522	-0.609303	-0.414374	-0.750488	1
3	0.084551	-2.297453	-1.052479	-0.689409	-0.442789	-0.228114	-0.731720	3
4	-0.168494	-0.985361	-0.172697	-0.628307	-0.582516	0.118467	-0.705444	3

Visualizing the clusters using a radar plot

```python
# Create the Radar chart
fig = go.Figure(data=traces)
range_ = 3.5
fig.update_layout(
    polar=dict(radialaxis=dict(visible=True, range=[-range_, range_])),
    showlegend=True,
    width=700,
    height=550,
)

fig.show()
```



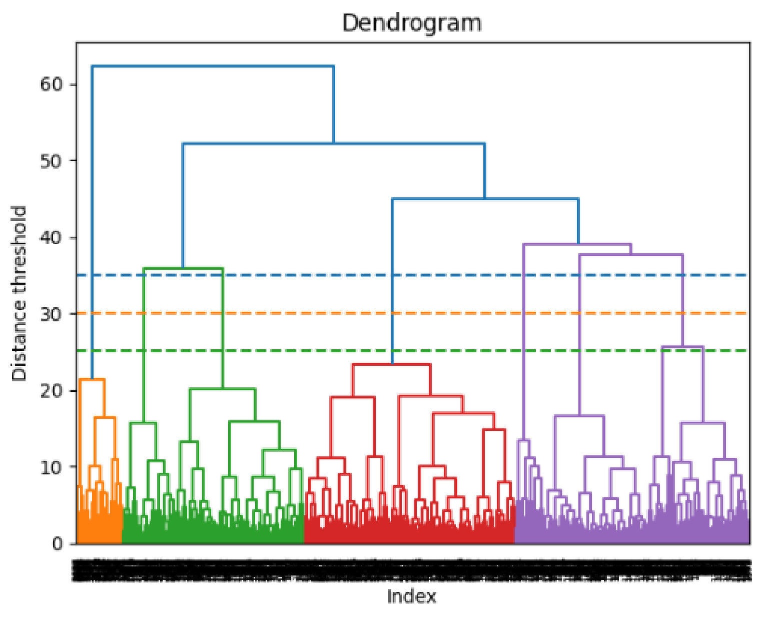
We can already notice distinct features for some clusters:

- Cluster 6: Tracks with a live feeling
- Cluster 4: Vocal, focused on speech tracks
- Cluster 1: Very acoustic feeling
- Cluster 0: Highest danceability from the 7 clusters found

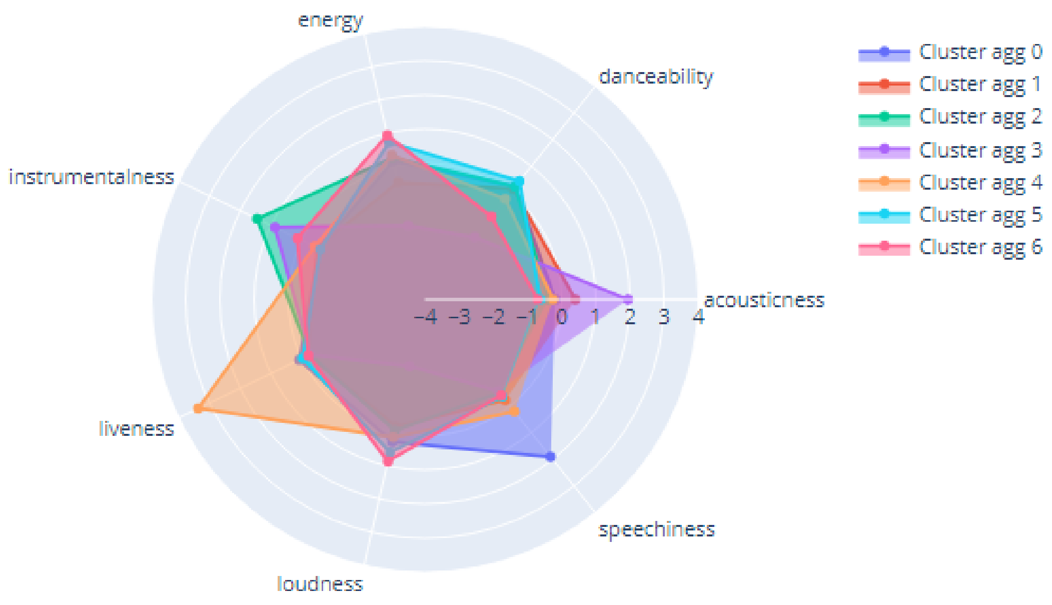
Agglomerative Clustering

Using a dendrogram and a 'ward' linkage for this model to assess the optimal nr of clusters. Using a distance threshold of 30 between clusters we can select 7 clusters as with the previous model and compare. Any cluster link made above that threshold will be ignored.

Dendrogram



Visualizing the clusters using a radar plot



The plot looks very similar to the KMeans one so we will compare the recommended songs later on.

Mean Shift

Estimate_bandwidth - using this we can get the bandwidth param

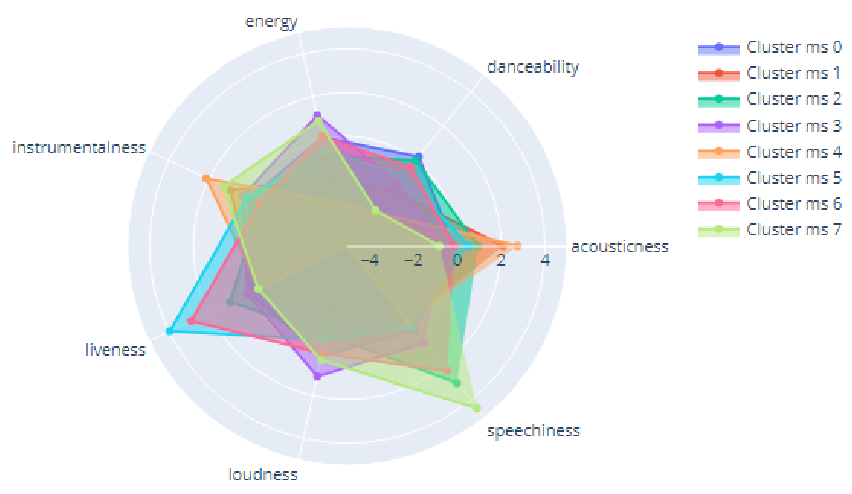
```
# Estimate bandwidth
bandwidth = estimate_bandwidth(X_pca, quantile=0.1, n_jobs=-1)

ms = MeanShift(bandwidth=bandwidth)
ms.fit(X_pca)
```

```
MeanShift
MeanShift(bandwidth=2.4669764093501394)
```

```
labels = ms.labels_

best_ms_clusters = len(np.unique(labels))
print("Number of clusters:", best_ms_clusters)
```

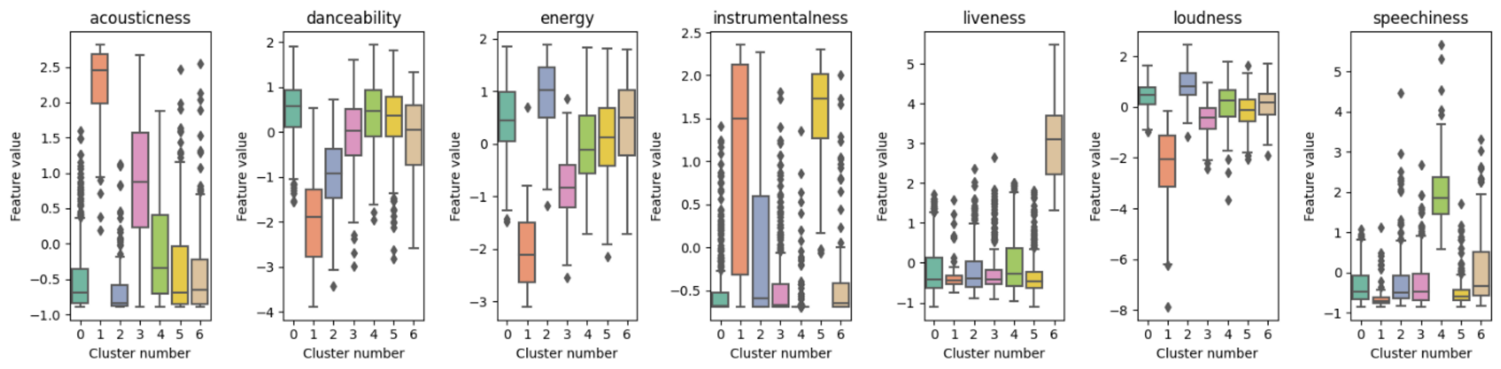
```
Number of clusters: 8
```

We got a similar number of clusters, however, there's a very large group and some clusters as low as 1-2 songs. This is not going to be used in the recommender system.



Differences between clusters, quite different from the previous two and it looks like more clusters are similar between them.

Cluster interpretation - KMeans



- Cluster 0: Highly danceable, energetic and loud tracks.
- Cluster 1: Acoustic and instrumental tracks with a lower energy.
- Cluster 2: High-energy and loud tracks with an intense sound.
- Cluster 3: Danceable and but with an acoustic feeling, less energy than 0 or 2 clusters.
- Cluster 4: Vocal/speech focused and danceable songs.
- Cluster 5: Instrumental, danceable tracks with low acousticness and liveness.
- Cluster 6: Songs with a live feel.

Checking clusters songs

```
Cluster 0: 424 tracks
```

	artist	track_name
1149	FKJ	Better Give U Up
1470	OFFAIAH	Love Me
110	Max Romeo	One Step Forward
311	Jungle	Casio
626	Gorgon City	Nobody
340	Chronixx	Never Give Up
456	ZHU	Secret Weapon
810	L'Entourloop	Shoefiti
406	Jack Swoon	Somebody
353	Night Tales	Friends

```
Cluster 1: 84 tracks
```

	artist	track_name
1531	Pola & Bryson	Southbank
1074	Coastal	Dusk
1221	RY X	Bound
1246	The Orioles	Tell Me So
1240	Gia Margaret	Smoke
1232	Goldmund	Sometimes
1144	ford.	Bedford Falls
150	Gregory Isaacs	Don't Play With Fire
1405	Galimatias	Intro
1541	Noah Slee	Ngata'anga (Outro)

```
Cluster 2: 180 tracks
```

	artist	track_name
1522	Pola & Bryson	Night Dawns
1333	grandson	Dirty
299	Document One	Technology
327	Mefjus	Disclosed
247	Mefjus	Break Away
359	Culture Shock	Recombine
337	ODESZA	Love Letter (feat. The Knocks)
414	Benny Page	Turn Down the Lights
17	Inhaler	It Won't Always Be Like This
530	Protoje	Really Like You

```
Cluster 3: 318 tracks
```

	artist	track_name
1568	Cigarettes After Sex	Nothing's Gonna Hurt You Baby
908	Subcarpați	Dă-l Foale
103	Vibrações	Ela Partiu (Superstar)
556	Nu	Gold
395	Ed.1t	Breathe
781	KOLIDESCOPES	Foundations
1129	zodivk	feelittoo (+)
1337	Mansionair	We Could Leave
1084	ford.	The Feeling
255	Koji.	Honest

Looking at some random songs from a couple of clusters and they seem to fit our previous descriptions.

Cluster 0: With songs from ZHU, Gorgon City, Jungle, FKJ, very danceable and energetic tracks.

Cluster 1: RY X, Galimatias, Noah Slee - slower songs, with no strong beat, acoustic feeling.

Cluster 2: Noticing some DNB artists like Mefjus, Culture Shock, Document One with very high energy and aggressive sounds, or grandson with an alternative rock song.

Cluster 3: Danceable songs here, mostly with vocals, but a more chill vibe.

Recommender system

For this I kept both the KMeans and Agglomerative Clustering models and the recommender system is based on Euclidean distance and cosine similarity.

The function used here takes a model and an artist as input and either a selected song or a random one from that artist and cluster.

Examples below bring similar results as previously noted for Cluster 2, energetic, loud, DNB songs for the most part..

```python
models = [kmeans_loaded, agg_loaded]
get_recommendations_loop(models, song_selection='random')
```

```
Enter artist name:  Mefjus
Selected song for Mefjus: Sientelo
```

```
cluster_num: 2
KMeans(n_clusters=7, n_init=10, random_state=420)

 Here are 5 similar tracks to: Mefjus - Sientelo
```

	Euclidean Distance_artist	Euclidean Distance_track	Cosine Similarity_artist	Cosine Similarity_track
474	Fox Stevenson	Good Time	Fox Stevenson	Good Time
1442	MUZZ	The Warehouse	MUZZ	The Warehouse
225	Mefjus	Zenith (feat. flowanastasia)	Mefjus	Zenith (feat. flowanastasia)
1082	Flume	The Difference	Flume	The Difference
342	Document One	Flute Ting	-	-
421	-	-	Krafty Kuts	The Remedy

```
cluster_num: 6
AgglomerativeClustering(n_clusters=7)

 Here are 5 similar tracks to: Mefjus - Sientelo
```

	Euclidean Distance_artist	Euclidean Distance_track	Cosine Similarity_artist	Cosine Similarity_track
474	Fox Stevenson	Good Time	Fox Stevenson	Good Time
1442	MUZZ	The Warehouse	MUZZ	The Warehouse
342	Document One	Flute Ting	Document One	Flute Ting
1510	Secondcity	I Wanna Feel - Radio Edit	-	-
377	Flume	Say Nothing	-	-
421	-	-	Krafty Kuts	The Remedy
1289	-	-	ZHU	I Admit It (feat. 24kGoldn) [NGHTMRE Remix]

Key findings and insights

- Since there is no ground truth, the clusters cannot be evaluated. Instead, we should determine if the data and the findings meet our requirements.
- Models group songs based on selected features and generate useful insights, but no model can perfectly match our subjective preferences.
- To simplify the model, PCA was used to reduce the number of components to 7, explaining over 90% of the variance.

- After testing the K-Means algorithm using the Elbow method, Silhouette score, and Davies Bouldin index, the optimal number of clusters was found to be 7 from all three methods.
- For Agglomerative Clustering, a dendrogram was used to test the number of clusters, and 7 clusters seemed good based on the selected distance and cluster formation.
- Using the estimate_bandwidth, the Mean Shift model grouped the songs into 8 clusters.

- The K-Means algorithm partitions songs into properly sized groups, and manual checks show that similar songs are grouped together.
- Since Agglomerative Clustering is quite similar, both can be kept for the recommender system as a comparison.
- The Mean Shift model generated a similar number of clusters, but most groups were small, with one large group.

- A recommender system was built based on the K-Means cluster using Euclidean distance and cosine similarity.
- This is a small project, probably not comparable to Spotify's models, but these features served as a base for clustering music. The recommender system works.

Suggestions

- Get a much larger song database, as these songs are gathered based on personal preferences, so some songs might be outliers in my playlist as a style.
- Create a web app, where you can input your song and get a number of recommendations.
- Create a function which can automatically create playlists in your Spotify account based on the clusters / model of choice.