Enhancing the Song Recommender Engine

By: Jonathan Chan, Bryan Chen, Geoffrey Liu

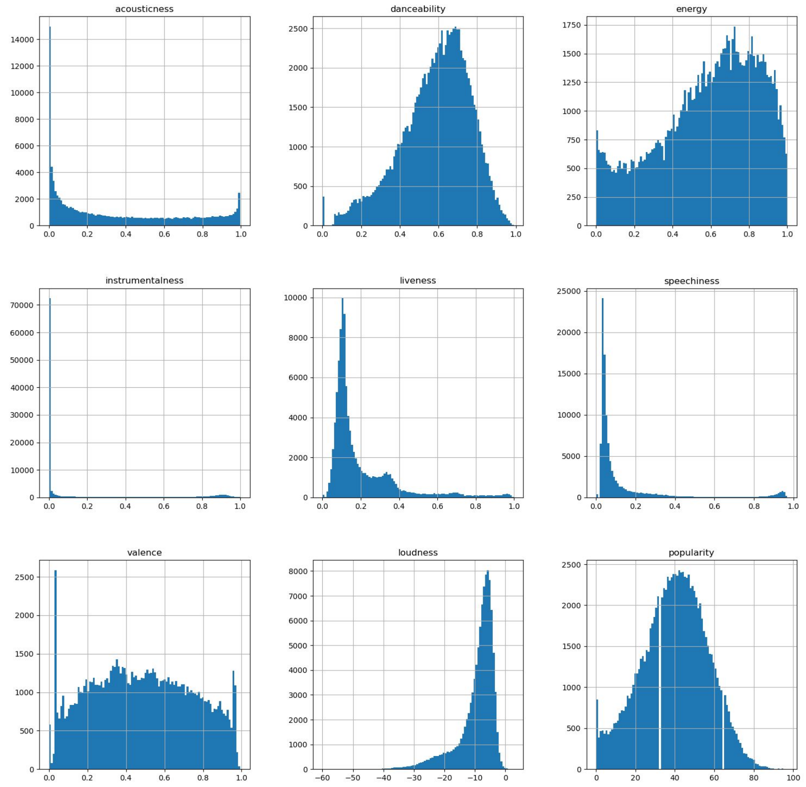
Original Model Overview

The original song recommender engine uses a K-Means clustering algorithm that creates 5 clusters. The model predicts which cluster the user is most likely to prefer based on their favorite songs, and recommends the first 5 songs in the cluster.

When a user inputs the IDs of their favorite songs, the song recommender engine filters the DataFrame to retrieve the user's favorite songs. The engine then identifies the cluster that most of the user's favorite songs belong to by counting the occurrences of each cluster number in the 'type' column of the user's favorite songs. The cluster with the highest frequency becomes the user's preferred cluster.

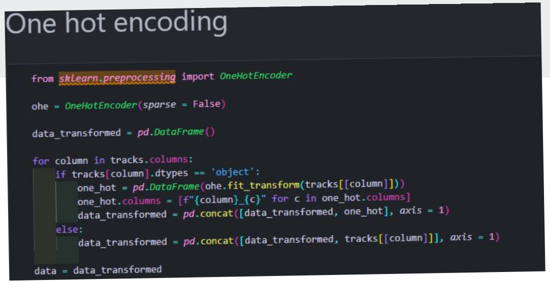
Data Exploration

Visualizations - Histogram



Some observations :

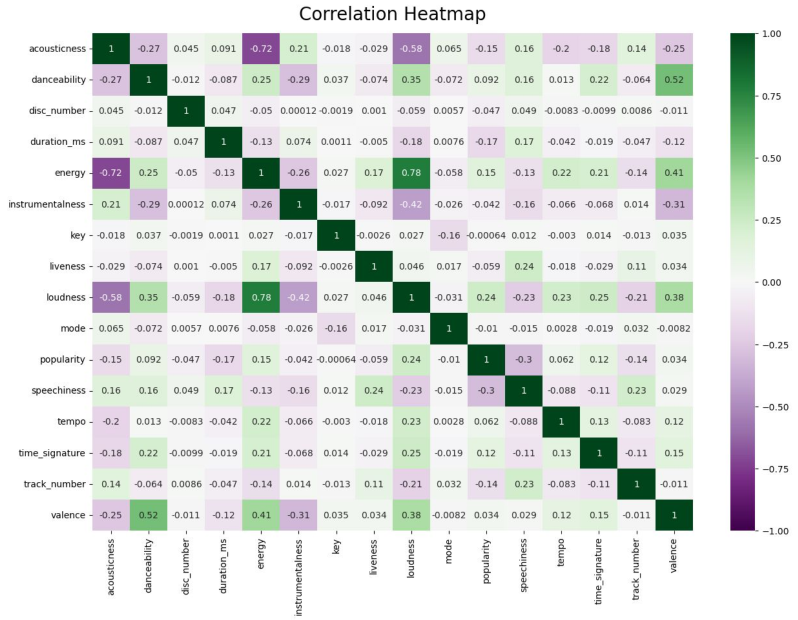
- High frequency of songs have low ‘Instrumentalness’, meaning if the user is looking for songs with a heavy focus on the instrumental, the choices may be more limited.
- Valence has two obvious clusters on either end of the graph, which show that there are many very sad and very happy songs.



One hot encoding

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse = False)

data_transformed = pd.DataFrame()

for column in tracks.columns:
    if tracks[column].dtypes == 'object':
        one_hot = pd.DataFrame(ohe.fit_transform(tracks[[column]]))
        one_hot.columns = [f"{column}_{c}" for c in one_hot.columns]
        data_transformed = pd.concat([data_transformed, one_hot], axis = 1)
    else:
        data_transformed = pd.concat([data_transformed, tracks[[column]]], axis = 1)

data = data_transformed
```

Visualizations – Correlation Heatmap



Correlation Heatmap

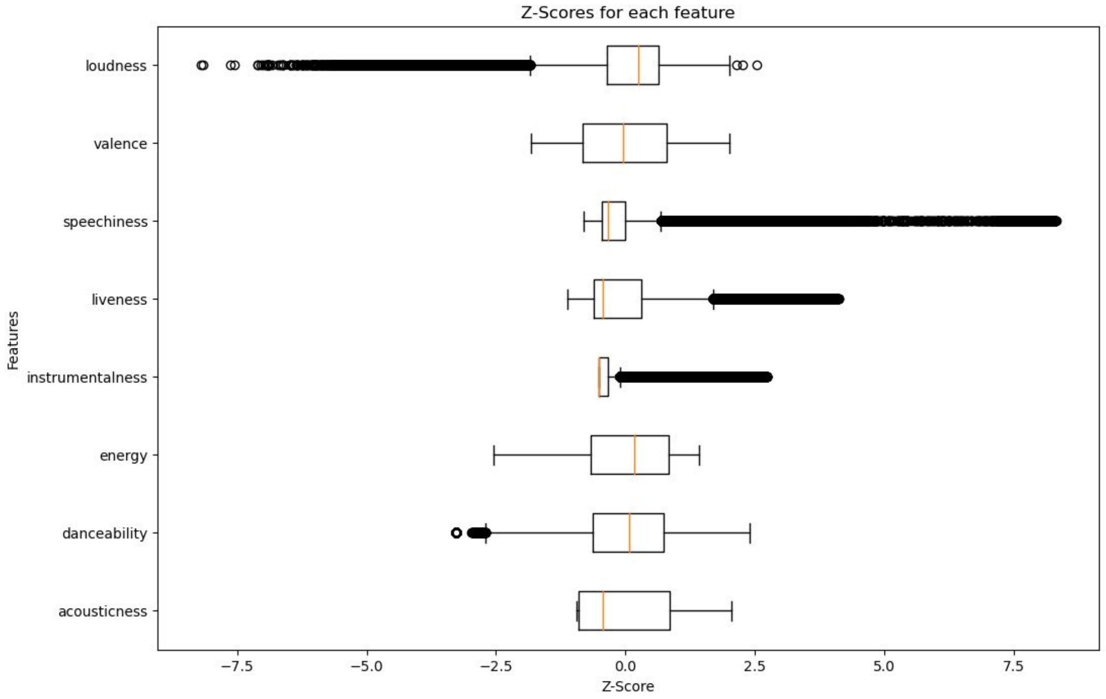
	acousticness	danceability	disc_number	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	time_signature	track_number	valence
acousticness	1	-0.27	0.045	0.091	-0.72	0.21	-0.018	-0.029	-0.58	0.065	-0.15	0.16	-0.2	-0.18	0.14	-0.25
danceability	-0.27	1	-0.012	-0.087	0.25	-0.29	0.037	-0.074	0.35	-0.072	0.092	0.16	0.013	0.22	-0.064	0.52
disc_number	0.045	-0.012	1	0.047	-0.05	0.00012	-0.0019	0.001	-0.059	0.0057	-0.047	0.049	-0.0083	-0.0099	0.0086	-0.011
duration_ms	0.091	-0.087	0.047	1	-0.13	0.074	0.0011	-0.005	-0.18	0.0076	-0.17	0.17	-0.042	-0.019	-0.047	-0.12
energy	-0.72	0.25	-0.05	-0.13	1	-0.26	0.027	0.17	0.78	-0.058	0.15	-0.13	0.22	0.21	-0.14	0.41
instrumentalness	0.21	-0.29	0.00012	0.074	-0.26	1	-0.017	-0.092	-0.42	-0.026	-0.042	-0.16	-0.066	-0.068	0.014	-0.31
key	-0.018	0.037	-0.0019	0.0011	0.027	-0.017	1	-0.0026	0.027	-0.16	-0.00064	0.012	-0.003	0.014	-0.013	0.035
liveness	-0.029	-0.074	0.001	-0.005	0.17	-0.092	-0.0026	1	0.046	0.017	-0.059	0.24	-0.018	-0.029	0.11	0.034
loudness	-0.58	0.35	-0.059	-0.18	0.78	-0.42	0.027	0.046	1	-0.031	0.24	-0.23	0.23	0.25	-0.21	0.38
mode	0.065	-0.072	0.0057	0.0076	-0.058	-0.026	-0.16	0.017	-0.031	1	-0.01	-0.015	0.0028	-0.019	0.032	-0.0082
popularity	-0.15	0.092	-0.047	-0.17	0.15	-0.042	-0.00064	-0.059	0.24	-0.01	1	-0.3	0.062	0.12	-0.14	0.034
speechiness	0.16	0.16	0.049	0.17	-0.13	-0.16	0.012	0.24	-0.23	-0.015	-0.3	1	-0.088	-0.11	0.23	0.029
tempo	-0.2	0.013	-0.0083	-0.042	0.22	-0.066	-0.003	-0.018	0.23	0.0028	0.062	-0.088	1	0.13	-0.083	0.12
time_signature	-0.18	0.22	-0.0099	-0.019	0.21	-0.068	0.014	-0.029	0.25	-0.019	0.12	-0.11	0.13	1	-0.11	0.15
track_number	0.14	-0.064	0.0086	-0.047	-0.14	0.014	-0.013	0.11	-0.21	0.032	-0.14	0.23	-0.083	-0.11	1	-0.011
valence	-0.25	0.52	-0.011	-0.12	0.41	-0.31	0.035	0.034	0.38	-0.0082	0.034	0.029	0.12	0.15	-0.011	1

Acousticness has a strong negative correlation with energy. This shows that acoustic songs tend to have lower energy.

Valence is positively correlated with danceability, energy, and loudness. This shows that happier songs tend to be easier to dance to, more high-energy, and louder.

Popularity is not closely correlated with any of the variables, which show that Spotify has a diverse user base with people enjoying several types of music and that all types of music can be popular.

Visualizations 3 - Z-Score Distribution



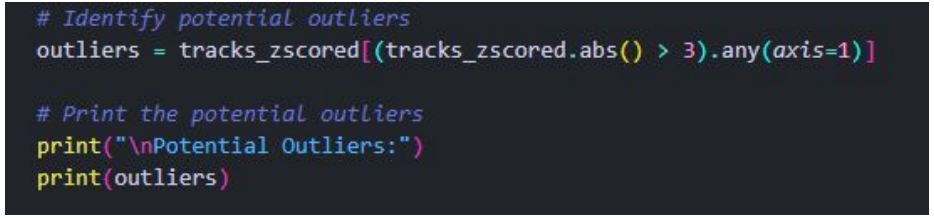
Some observations :

- The boxplot for ‘Loudness’ has many low outliers. This shows that while songs tend to be at a certain loudness, many songs are below that loudness.
- The boxplot for ‘Energy’ has no outliers. This shows that energy is rather evenly distributed in Spotify songs.

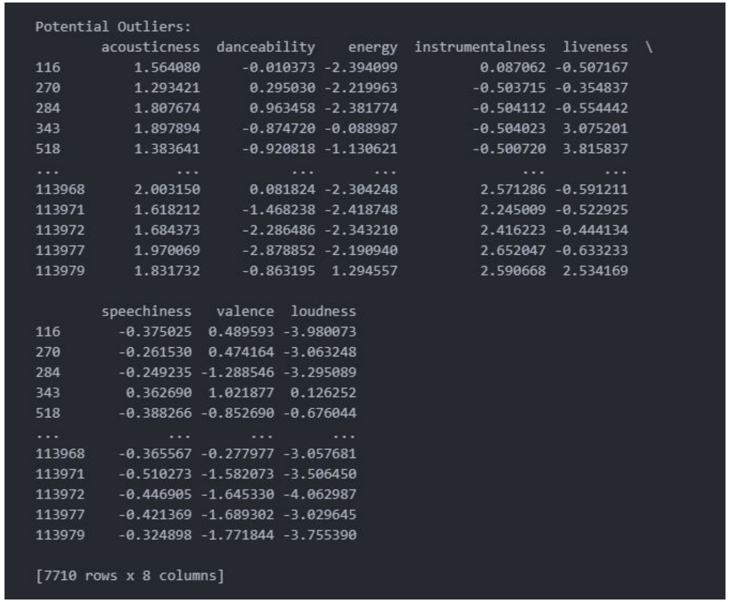
Model Enhancement

Handling Outliers

In order to handle the outliers, we implemented z-score normalization as a preprocessing step before applying the K-Means clustering algorithm. By standardizing the data through z-score, we ensure that all the features have a mean of 0 and a standard deviation of 1. See Visualization 3 for the z-score distribution.



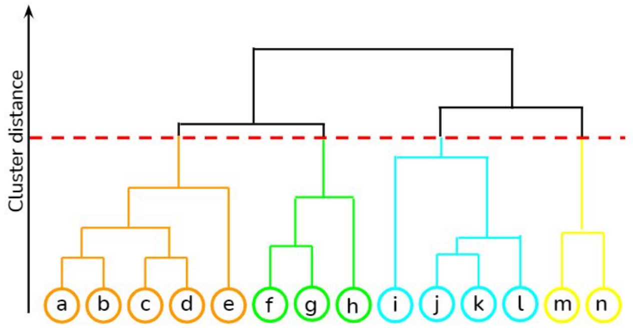
```python
# Identify potential outliers
outliers = tracks_zscored[(tracks_zscored.abs() > 3).any(axis=1)]

# Print the potential outliers
print("\nPotential Outliers:")
print(outliers)
```



```
Potential Outliers:
        acousticness  danceability    energy  instrumentalness  liveness  \
116         1.564080     -0.010373 -2.394099          0.087062 -0.507167
270         1.293421      0.295030 -2.219963         -0.503715 -0.354837
284         1.807674      0.963458 -2.381774         -0.504112 -0.554442
343         1.897894     -0.874720 -0.088987         -0.504023  3.075201
518         1.383641     -0.920818 -1.130621         -0.500720  3.815837
...              ...           ...       ...               ...       ...
113968      2.003150      0.081824 -2.304248          2.571286 -0.591211
113971      1.618212     -1.468238 -2.418748          2.245009 -0.522925
113972      1.684373     -2.286486 -2.343210          2.416223 -0.444134
113977      1.970069     -2.878852 -2.190940          2.652047 -0.633233
113979      1.831732     -0.863195  1.294557          2.590668  2.534169

        speechiness   valence  loudness
116       -0.375025  0.489593 -3.980073
270       -0.261530  0.474164 -3.063248
284       -0.249235 -1.288546 -3.295089
343        0.362690  1.021877  0.126252
518       -0.388266 -0.852690 -0.676044
...             ...       ...       ...
113968    -0.365567 -0.277977 -3.057681
113971    -0.510273 -1.582073 -3.506450
113972    -0.446905 -1.645330 -4.062987
113977    -0.421369 -1.689302 -3.029645
113979    -0.324898 -1.771844 -3.755390

[7710 rows x 8 columns]
```

Other Clustering Algorithms

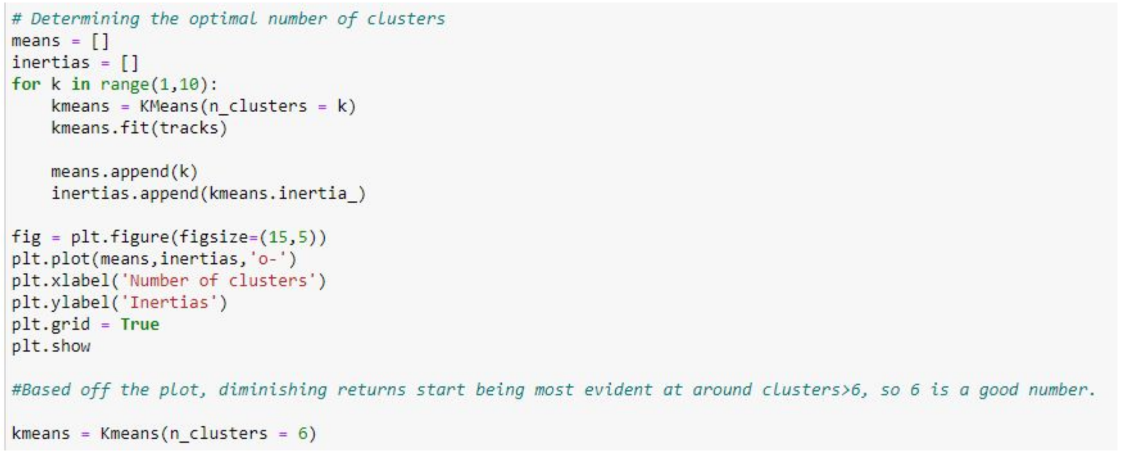
- Hierarchical Clustering



When comparing to KMeans, Hierarchical clustering offers a more detailed representation of data relationships by creating nested clusters. It splits song types based on their traits which is appropriate for different music genres. However, we found that its computational power limits its real-time applicability due to the high RAM it was needed to run for a large database like this one. Overall, KMeans was better.

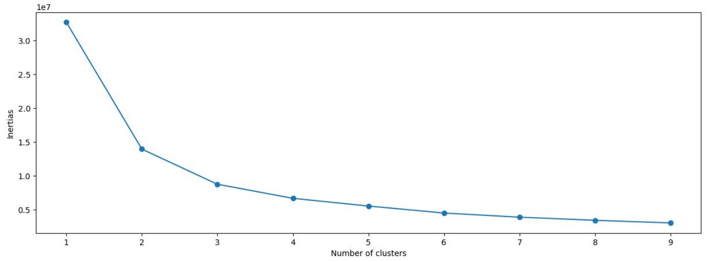
Number of Clusters

The optimal number of clusters was found through adding this section of code to plot the inertias:



```
# Determining the optimal number of clusters
means = []
inertias = []
for k in range(1,10):
    kmeans = KMeans(n_clusters = k)
    kmeans.fit(tracks)

    means.append(k)
    inertias.append(kmeans.inertia_)

fig = plt.figure(figsize=(15,5))
plt.plot(means,inertias,'o-')
plt.xlabel('Number of clusters')
plt.ylabel('Inertias')
plt.grid = True
plt.show

#Based off the plot, diminishing returns start being most evident at around clusters>6, so 6 is a good number.

kmeans = Kmeans(n_clusters = 6)
```

Result :

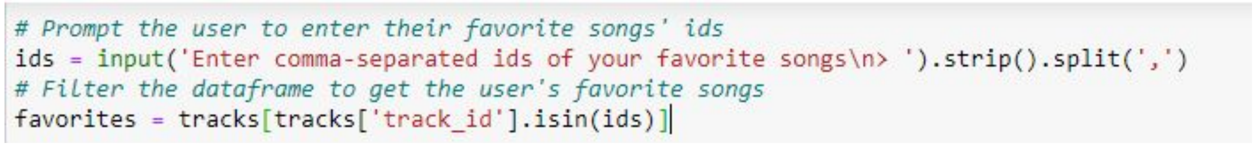


Inertias = sums of distance between plots and cluster (generally, the lower inertia the better, but if the inertia is too low the model may be overfitted.)

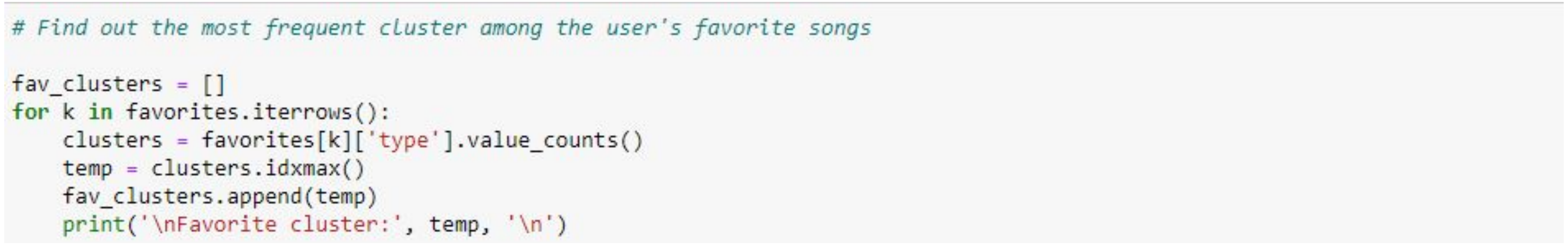
User Interaction & Feedback

Improving Inputs & Recommendations

- The user is able to input multiple song IDs at once, each ID being separated by a comma.
- These IDs are stored within a list which we would call ‘favorites’
- Our code identifies the user’s favorite cluster for each song by iterating through ‘favorites’, giving us multiple clusters to search through.
- The code then outputs multiple songs from each identified cluster as ‘recommendations’.

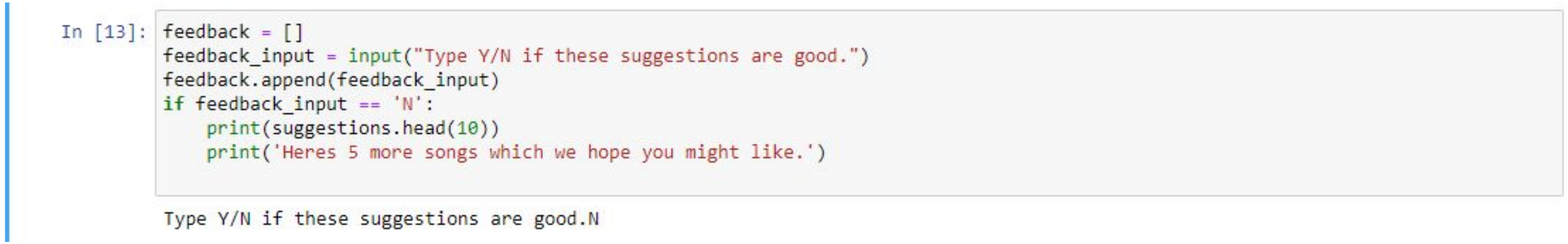


```
# Prompt the user to enter their favorite songs' ids
ids = input('Enter comma-separated ids of your favorite songs\n> ').strip().split(',')
# Filter the dataframe to get the user's favorite songs
favorites = tracks[tracks['track_id'].isin(ids)]
```



```
# Find out the most frequent cluster among the user's favorite songs

fav_clusters = []
for k in favorites.iterrows():
    clusters = favorites[k]['type'].value_counts()
    temp = clusters.idxmax()
    fav_clusters.append(temp)
    print('\nFavorite cluster:', temp, '\n')
```

Future Uses for User Feedback



```
In [13]: feedback = []
         feedback_input = input("Type Y/N if these suggestions are good.")
         feedback.append(feedback_input)
         if feedback_input == 'N':
             print(suggestions.head(10))
             print('Heres 5 more songs which we hope you might like.')

         Type Y/N if these suggestions are good.N
```

- User feedback is stored in a list for future purposes (To see ratio of Y/N and improve model)
- In the case that the user is unhappy with the model, we can suggest 5 more songs.
- If N>Y, then we can improve model through a variety of ways such as: Changing number of clusters, changing parameters in ‘tracks’, etc.

Future Improvements

Future Improvements

- When selecting relevant features, features such as ‘lyrics’ are difficult to analyze and be used as it is not numerical and are long string values. A future improvement could be to somehow implement this into our code as well. For example, analyzing the patterns of the words used in the lyrics can provide valuable information into user’s favourite songs.
- Genre tags on songs can also be used as data in the future to help in identifying clusters. By using genre information, the engine can better understand the context of each song, which will lead to more accurate clustering.
- Creating a UI using elements such as HTML for a more comprehensible and user-friendly format will be much more appealing and effective to use for the average user.
- Improve runtime