Using Machine Learning to Categorise Spotify Playlists

I have been using Spotify for 11 years and while I love making playlists, it can get a little messy. So, most of the times I just like a song and that's that. This has resulted in my saved songs library containing almost 4000 songs! Occasionally, I get a sudden burst of motivation to re-organise my entire Spotify. Of course, this usually happens when I have an assignment I don't want to do.

I decided to make my procrastination worthwhile and organise my Spotify library for my final project. I also would love to work on the data visualisation team at Spotify, so I am taking this opportunity to get familiar with the Spotify API.

I will be taking two approaches: unsupervised and supervised approach. In the unsupervised version, the tracks are clustered using the song features. In the supervised version, I will use my existing playlists to train a model to categorise the liked songs into playlists after understanding the relation between the songs and the playlists.

Here is a quick overview of my steps:

1. Data Acquisition and Preparation
    - Get my saved tracks
    - Generate audio features
    - Save them to a csv file
2. Unsupervised Approach
    - Use KNN to cluster saved songs
    - Generate playlists based on the clusters
    - Add tracks to the playlists
3. Supervised Approach
    - Get existing playlists
    - Produce training data using the existing playlists
    - Train a random forest model for track classification
    - Use the trained model to categorise the saved songs

I. Data Preparation

Spotipy is a Python library for the Spotify Web API. With *Spotipy* I can get full access to the music data provided by the Spotify platform. All methods require user authorisation. I got my necessary credentials (`client_id`, `client_secret`, and `redirect_URI`) for authorisation from the Spotify Developer Dashboard.

The Spotify API has an `audio_feature()` feature which returns 11 song attributes, such as energy, instrumentalness, tempo, etc. I will be using these song attributes to create clusters and playlists.

```
import pandas as pd
import spotipy
import numpy as np
from spotipy.oauth2 import SpotifyClientCredentials
from spotipy.oauth2 import SpotifyOAuth
from time import time

cid ="2a9c4a27a7434e66b5f054fe97a9662a"
secret = "69fca95506cc46e180f929f784fb446c"
redirect_uri = 'http://localhost:8888/callback'

FEATURE_KEYS = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'a
cousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

OFFSET=0
SAVED_TRACKS_LIMIT=50
FEATURE_LIMIT = 100

sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id=cid,
                                               client_secret=secret,
                                               redirect_uri=redirect_uri,
                                               scope="user-library-read"))

liked_tracks = list()
print(liked_tracks)

while(True):
    paged_tracks = sp.current_user_saved_tracks(offset=OFFSET, limit=SAVED_TRACKS_LIMI
T)
    liked_tracks.extend([{'name':el['track']['name'], 'id':el['track']['id']} for el i
n paged_tracks['items']])
    print(f'Fetched {len(liked_tracks)} tracks')
    OFFSET+=SAVED_TRACKS_LIMIT
    if paged_tracks['next'] is None:
        break

def get_windowed_track_ids(liked_tracks, limit):
    for i in range(0, len(liked_tracks), limit):
        track_window = liked_tracks[i:i + limit]
        yield track_window, [t['id'] for t in track_window]
```

```
track_feature_list = list()
print('')

for track_window, track_window_ids in get_windowed_track_ids(liked_tracks, FEATURE_LIM
IT):
    track_features = sp.audio_features(tracks=track_window_ids)
    for index, _track in enumerate(track_window):
        _track.update({k:v for k,v in track_features[index].items() if k in FEATURE_KE
YS})
        track_feature_list.append(_track)
    print(f'Fetched features for {len(track_feature_list)} tracks')

df = pd.DataFrame.from_dict(track_feature_list)
mysavedsongs = f'liked_tracks_{int(time())}.csv'
df.to_csv(mysavedsongs, index=False)
print('')
print(f'Saved features to {mysavedsongs}')
```

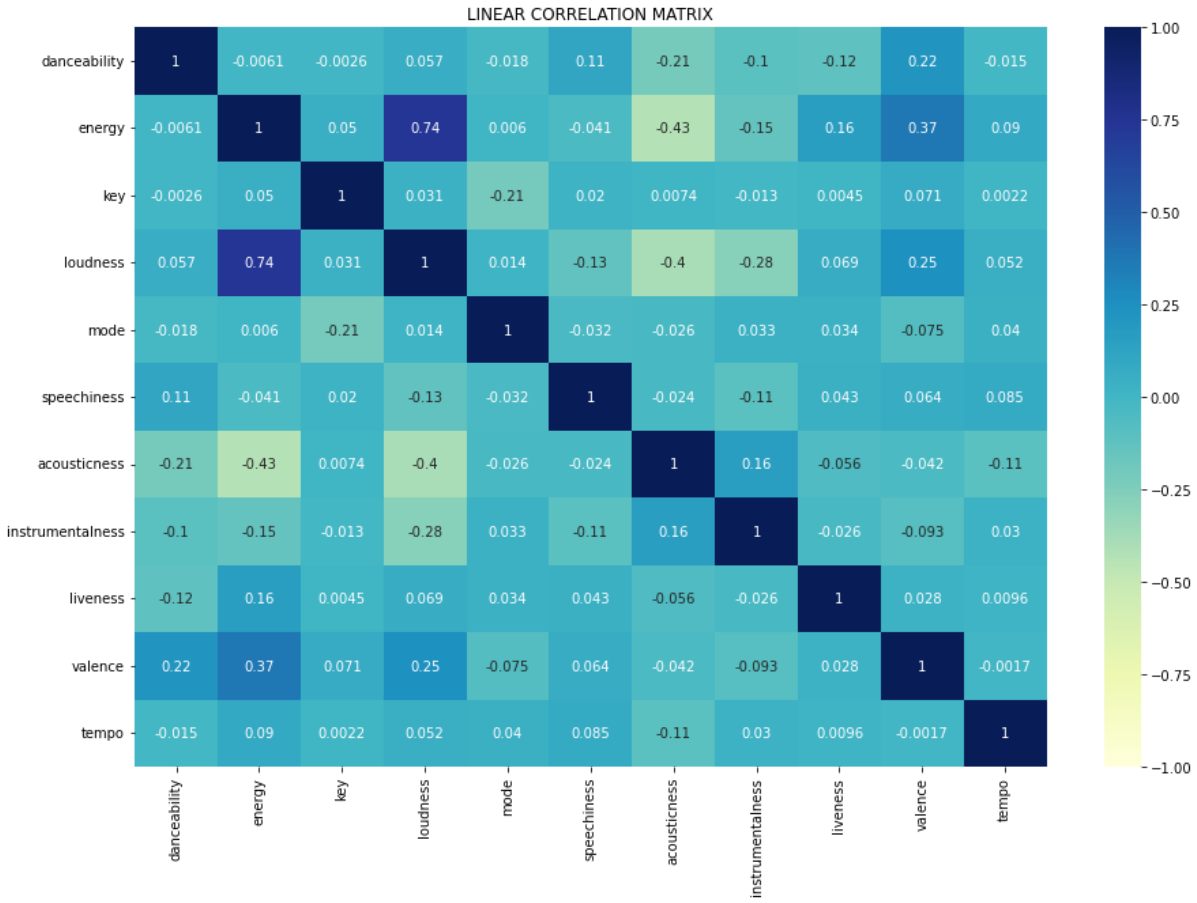
Below is a sample of the generated dataset.

	name	id	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0	Law Of Attraction (feat. Snoh Aalegra)	25tXNghelZKdGZZVoSL9Yg	0.774	0.5130	0	-8.159	1	0.3420	0.3200	0.000105	0.0866	0.5100	102.780
1	Interference	5Bngj85lUf1HrAWGhMAwRn	0.614	0.0908	3	-13.388	1	0.0503	0.9380	0.000000	0.3230	0.2990	126.217
2	Ice T	4vcUf8YqLi1mDeT9c0Wo6f	0.679	0.4810	0	-8.791	0	0.1150	0.1010	0.000000	0.0844	0.6650	134.168
3	Free Mind	2mzM4Y0Rnx2BDZqRnhQ5Q6	0.562	0.4630	6	-7.478	1	0.0787	0.4420	0.000145	0.1570	0.4160	125.150
4	Higher	2QdSb68BzZGMgCbsrFmSLc	0.609	0.5810	2	-10.854	1	0.2770	0.5090	0.000000	0.2830	0.5280	102.964
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3995	Fine Line	6VzcQuzTNTMFnJ6rBSaLH9	0.306	0.3470	2	-8.500	1	0.0334	0.1720	0.000130	0.0485	0.0511	120.996
3996	The Love Club	2yrJ1jWo3HLksJFUqUsZE4	0.794	0.4930	5	-7.257	1	0.0563	0.2120	0.013300	0.1300	0.6260	92.026

I.I Correlation

Before we start with the machine learning, I wanted to do a little bit of exploratory analysis to visually see the interactions between the different song features. Using `matplotlib` and `seaborn`, I plot a correlation matrix of the 11 different features.

```
import matplotlib.pyplot as plt
import seaborn as sns

fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(df.corr(), annot=True, cmap='YlGnBu', vmin=-1, vmax=1, center=0, ax=ax)
plt.title('LINEAR CORRELATION MATRIX')
plt.show()
```



We see that energy and loudness have a strong positive correlation suggesting songs that are 'loud' are probably also 'energetic'. This reinforces a fairly intuitive observation. There also seems to be a noticeable negative correlation between 'energy' and 'acousticness', which is also not very surprising.

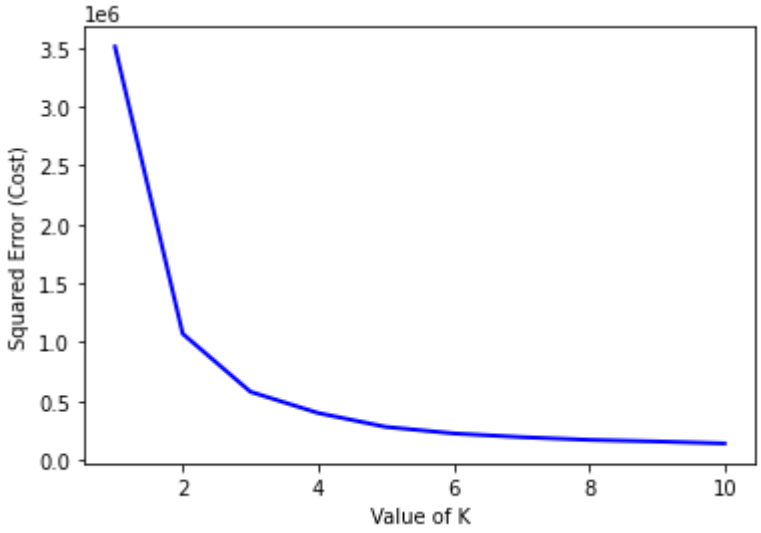
With added metadata such as language, genre and artist, we can further our analysis. We can look into the genre makeup of my entire music library. In the future, I would like the split my analysis based on the language of the music since my library consists of music in multiple different languages.

II. Unsupervised Approach

Now that our data is prepared , we can start generating clusters. The first step is to determine how many clusters must be created. I do this using the Elbow Method, which is a common method of determining the number of clusters (K) in a dataset. The Value of K is plotted against the squared error, and the point where the curve kinks is considered to be the optimal number of clusters.

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
from pandas import read_csv

FEATURE_KEYS = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'a
cousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']
K_MAX = 11

df=read_csv('liked_tracks_1639144651.csv')

cost = list()
for i in range(1, K_MAX):
    KM = KMeans(n_clusters = i, max_iter = 500)
    KM.fit(df[FEATURE_KEYS])
    cost.append(KM.inertia_)

plt.plot(range(1, K_MAX), cost, color ='b', linewidth ='2')
plt.xlabel("Value of K")
plt.ylabel("Squared Error (Cost)")
plt.show()
```

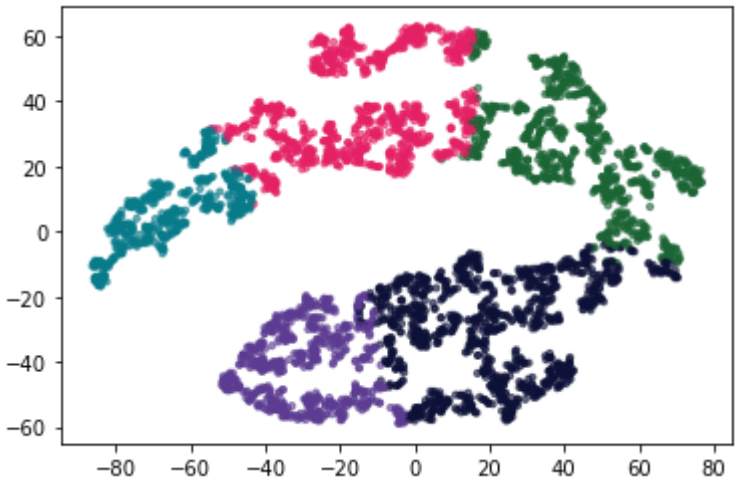


In the graph above we can see that curve kinks at a value of around 3. However, I will pick my K=5 since that's when the slop really starts to decrease. With K=5, all of my music will be split into 5 playlists.

Once the tracks are organised into clusters, it would be good to see distribution visually. However, since we are using 11 different attributes, there are 11 dimensions to each track. It would be difficult to visualise it in a 2D format. To solve this I used TSNE, which is a method of dimensionality reduction.

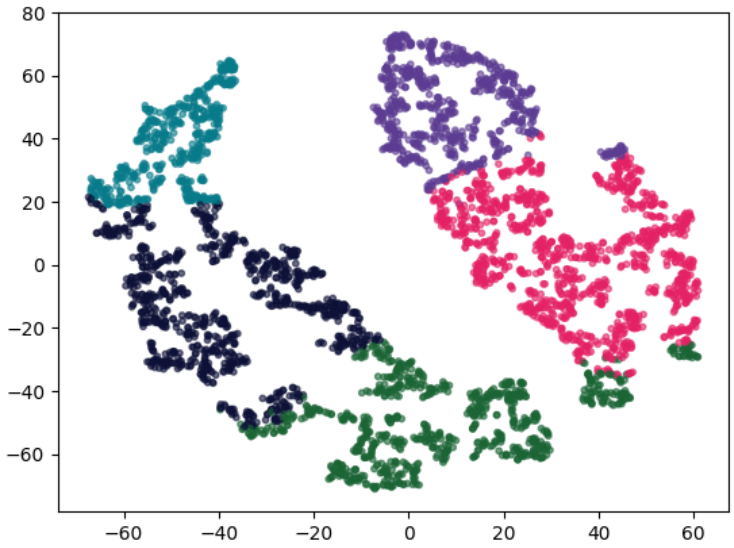
```
from pandas import read_csv

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
```

```
import numpy as np

NUM_CLUSTERS = 5
TRACK_ADD_LIMIT = 100
COLORS = ['#e52165', '#0d1137', '#077b8a', '#5c3c92', '#1b6535']
COLOR_MAP = {0:COLORS[0], 1:COLORS[1], 2:COLORS[2], 3:COLORS[3], 4:COLORS[4]}
FEATURE_KEYS = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'a
cousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

df=read_csv('liked_tracks_1639144651.csv')

kmeans = KMeans(n_clusters=NUM_CLUSTERS, random_state=0)
df['cluster'] = kmeans.fit_predict(df[FEATURE_KEYS])
df['color'] = df.cluster.map(COLOR_MAP)

tsne = TSNE(n_components=2, random_state=0)
np.set_printoptions(suppress=True)
Y = tsne.fit_transform(df[FEATURE_KEYS].values)
df['x_coords'] = Y[:, 0]
df['y_coords'] = Y[:, 1]
plt.scatter(df.x_coords, df.y_coords, c=df.color, alpha = 0.6, s=10)
plt.show()
```



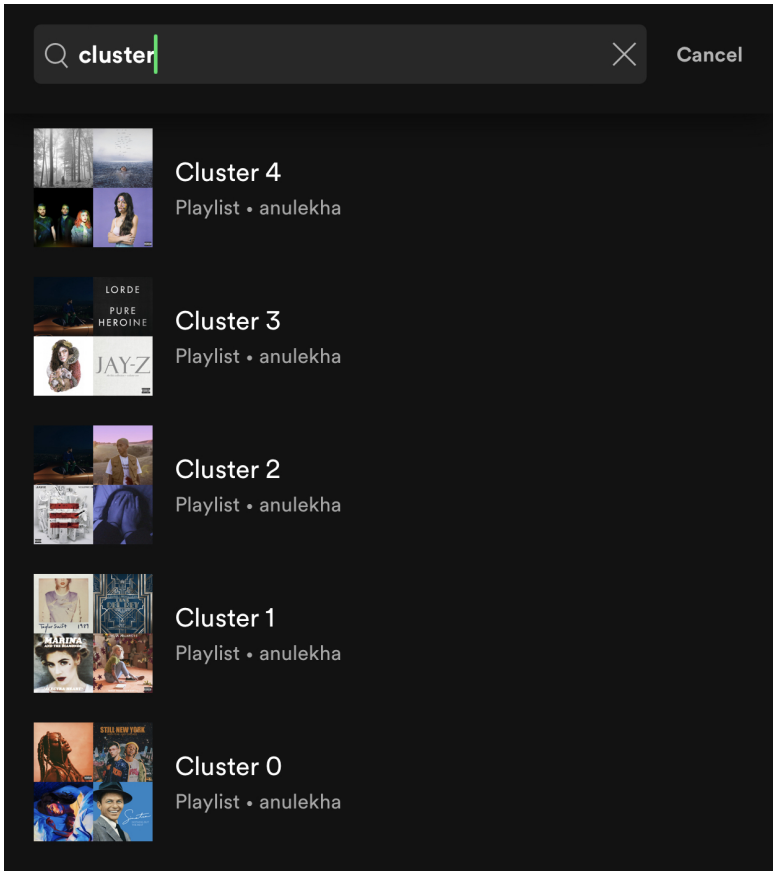
In the plot above, each dot represents a track, while each color represents a cluster. As you can see, there are five colors, corresponding to each of the five clusters.

Surprisingly, when I had run this code previously, I got a different looking graph, provided below. It would be interesting to further research the mathematical workings of clustering. However, that is beyond the scope of this paper.



Finally, I used the Spotify APIs to create new playlists for each of the clusters and add tracks to those playlists.

```
sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id=cid,
                                               client_secret=secret,
                                               redirect_uri=redirect_uri,
                                               scope="playlist-modify-public"))
user_id = sp.current_user()['id']

g=df.groupby('cluster')

def get_windowed_track_ids(track_ids, limit):
    for i in range(0, len(track_ids), limit):
        track_window = track_ids[i:i + limit]
        yield track_window

for cluster in range(NUM_CLUSTERS):
    _cluster_name = f'Cluster {cluster}'
    playlist_id = sp.user_playlist_create(user_id, _cluster_name)['id']
    print(f'Created playlist {_cluster_name}')
    for _tracks in get_windowed_track_ids(list(g.get_group(cluster)['id']), TRACK_ADD_
LIMIT):
        sp.playlist_add_items(playlist_id, _tracks)
        print(f'Added {len(_tracks)} tracks to playlist {_cluster_name}')
```



The screenshot above shows the five playlists that were generated from our algorithm.

III. Supervised Approach

In the supervised approach, we train a model to study the relationship between a playlist and its included songs. Doing so will help in clustering that is more suited to "our tastes". In the unsupervised KNN approach, all attributes are weighed equally. In reality, we may prefer certain attributes to be grouped together. Through the supervised approach, our model can understand these correlations better.

The first step is to prepare training data that we will use to train our model. I have created six playlists with the number of songs per playlist ranging from 600-1000.

```
TRACK_LIMIT = 100

FEATURE_KEYS = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'a
cousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

TRAIN_PLAYLISTS =   [
                    {'playlist_name': 'pl_1', 'playlist_id':'3TRzj5KZUrBbfm5Zsn1NEc'},
                    {'playlist_name': 'pl_2', 'playlist_id':'5LlX21oVHEh2TqQWBNYInG'},
                    {'playlist_name': 'pl_3', 'playlist_id':'6u6jBpu1Lnfq01y2PSrie2'},
```

```
                    {'playlist_name': 'pl_4', 'playlist_id':'0tHNRm3aMhSbu9hrbE08st'},
                    {'playlist_name': 'pl_5', 'playlist_id':'6gcG4gfLwNcY3zENW2sUVU'},
                    {'playlist_name': 'pl_6', 'playlist_id':'51P7SA9tYI19Usar8TsryX'}
                    ]

TRACKS = [[] for i in range(len(TRAIN_PLAYLISTS))]


sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id=cid,
                                               client_secret=secret,
                                               redirect_uri=redirect_uri,
                                               scope="playlist-read-private"))

for idx, train_playlist in enumerate(TRAIN_PLAYLISTS):
    print(f'Fetching tracks from playlist {train_playlist["playlist_name"]}')
    offset=0
    while True:
        paged_tracks = sp.playlist_items(train_playlist['playlist_id'], limit=TRACK_LI
MIT, offset=offset)
        TRACKS[idx].extend([{'name':el['track']['name'], 'id':el['track']['id']} for e
l in paged_tracks['items']])
        print(f'Fetched {len(TRACKS[idx])} tracks')
        offset+=TRACK_LIMIT
        if paged_tracks['next'] is None:
            break

TRAIN_DATA = []

def get_windowed_track_ids(_tracks, limit):
    for i in range(0, len(_tracks), limit):
        track_window = _tracks[i:i + limit]
        yield track_window, [t['id'] for t in track_window]

for idx, train_playlist in enumerate(TRAIN_PLAYLISTS):
    for track_window, track_window_ids in get_windowed_track_ids(TRACKS[idx], TRACK_LI
MIT):
        track_features = sp.audio_features(tracks=track_window_ids)
        for index, _track in enumerate(track_window):
            _track.update({k:v for k,v in track_features[index].items() if k in FEATUR
E_KEYS})
            _track.update(train_playlist)
            TRAIN_DATA.append(_track)
        print(f'Fetched {len(TRAIN_DATA)} features')

df=pd.DataFrame.from_dict(TRAIN_DATA)
filename = f'playlist_features_{int(time())}.csv'
df.to_csv(filename, index=False)
print(f'Saved features to {filename}')
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from collections import Counter
from pandas import read_csv
```

```
FEATURE_KEYS = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'a
cousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

TRAIN_DATA=read_csv('playlist_features_1639145050.csv')
PREDICT_DATA=read_csv('liked_tracks_1639144651.csv')

X_train, X_test, y_train, y_test = train_test_split(TRAIN_DATA[FEATURE_KEYS], TRAIN_DA
TA['playlist_name'],test_size=0.3)

model = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=7)
model.fit(X_train, y_train)

y_predicted = model.predict(X_test)
print(f'\nModel accuracy is {accuracy_score(y_test, y_predicted)}', end='\n\n')

_predicted_playlist = model.predict(PREDICT_DATA[FEATURE_KEYS])
PREDICT_DATA['assigned_playlist'] = _predicted_playlist

print(f'Prediction completed. Target distribution : {dict(Counter(_predicted_playlis
t))}', end='\n\n')

print(PREDICT_DATA[['name', 'assigned_playlist']].sample(10))
```

```
Model accuracy is 0.6053811659192825

Prediction completed. Target distribution : {'pl_6': 1393, 'pl_3': 948, 'pl_4': 454, 'pl_5': 980, 'pl_2': 221}

                             name assigned_playlist
3926                     Trophies              pl_3
3822                Feeling Myself              pl_5
667                           Lied              pl_5
2865                Reminiscence.              pl_5
345                     Good Form              pl_6
257                     EVERYTIME              pl_6
245       Changed (feat. Big Sean)              pl_5
437       So Good (& Metro Boomin)              pl_6
1453                Nowhere To Run              pl_6
938   Gum Body (feat. Jorja Smith)              pl_6
```

The model accuracy is around 60% which fairly good considering the fact that we didn't have a high number of training data.

The supervised version can still be refined, with a larger and better training data. The `max_depth` can also be increased for greater precision.