Final Year Project Report

Recommending Song Playlists

Michael Mallon

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science with Data Science

Supervisor: Aonghus Lawlor

UCD School of Computer Science
University College Dublin

April 7, 2019

Project Specification

Project Specification

Subject: Recommender Systems, Playlist Recommendation
Prerequisites: Recommender Systems, Python
Project Type: Design and Implementation
Software Requirements: Python, SQL, Juypter Notebooks
Hardware Requirements: No specific requirements

Description

General Information:

Most approaches to song recommendation are based on either collaborative filtering or content-based methods.

Collaborative filtering aims to determine the users' preferences from historical listening patterns.

A pair of users who listen to mostly the same songs, are considered to have broadly similar tastes. Pure collaborative filtering methods do not care about the songs themselves, only whether users have listened to them.

Content-based approaches, on the other hand, use information gleaned from the songs themselves. There are many different kinds of information associated with the songs that can help our recommender system: tags, artist and album information, lyrics, text mined from the web (reviews, interviews), and the audio signal itself.

This project aims to build a recommender system for song playlists which is personalized to the taste of each user.

The advanced part of this project aims to build a recommender system for song playlists and evaluate the performance for a number of baselines.

Core:

Analysis of Audio Feature Dataset and Spotify Dataset,
Data Mining of datasets

Advanced:

Recommender System for Playlist Continuation. Evaluation of Recommender System

Data:

Spotify Data (RecSys Challenge) 40GB
Track Audio Feature Dataset 4GB

Data provided by Supervisor.

Abstract

Music is very important to many individuals, they spend hours listening to it every day. It creates an ambiance and adds to every experience, it can be listened to at any time, studying, having breakfast and at parties. For most it is used as way of expressing themselves and their feelings. So, of course, there should be an emphasis put on how we find and discover such an influential thing in our lives.

The advances of modern day technology, data storage and software are growing exponentially, this has made finding new music much easier. With streaming applications like Spotify, Apple Music, YouTube and Google Play containing libraries with millions of songs just the click of a button away its hard not to listen to your favorite new song everyday.

Software called recommender systems have been developed and put to the test using large databases of millions of songs to help find the perfect set for each individual.

In recent years an emphasis has been placed in the use of deep audio features when recommending music with companies like Spotify spending a surplus of 100 million euros on music intelligence software.

Using these important audio features this project aims to create a strong unsupervised playlist continuation and recommender system that provides new and fresh music recommendations for individuals to listen to.

Table of Contents

1 Introduction **5**

2 Background Research **6**

2.1 ACM RecSys Challenge 2018 6

2.2 Automatic Continuation of Playlists 6

2.3 Current Recommender Systems 7

2.4 How do you define the popularity of a song? 11

2.5 How many genres are there? 13

2.6 How should we group songs? 14

2.7 Visualization and Dimension Reduction of Clusters: 15

2.8 Measuring the similarity of songs 16

2.9 How do we evaluate recommendations? 17

3 Data and Data Access **18**

3.1 The Dataset 18

4 Core: Analysis and Data Mining **20**

4.1 Analysis of Dataset 20

4.2 Popularity Rating 22

4.3 Clustering Songs 23

4.4 Comparing Artists 24

4.5 Cluster Overlap 25

4.6 Clustering Playlists 25

5 Advanced: Recommender System **27**

5.1 The Recommender System Process 27

5.2 Created Recommender Systems 28

5.3 Evaluation 29

5.3.1 Precision and Recall 29

5.3.2 Self-Evaluation 30

5.3.3 Comparing Against Spotify Recommendations 31
5.3.4 Playlist Continuation System Self Evaluation 32

6 Conclusion **33**

6.1 Project Review 33
6.2 Possible Improvements 34
6.3 Acknowledgments 35
6.4 GitHub 35

Chapter 1: **Introduction**

Music is very important to many individuals, they spend hours listening to it everyday. It creates an ambiance and adds to every experience. It can be listened to at any time thought the day, whilst studying, having breakfast and at parties. For most it is used as way of expressing themselves and their feelings. So, of course, there should be an emphasis put on how we find and discover such an influential thing in our lives.

This project will focus on the area of recommender systems with music, specifically that of playlist continuation. A recommender system is an integral piece of software that seeks to predict and preference a user would have when making a choice, in the case of music this would be a song or playlist. With recent advances in technology, this process has become much easier with data storage and availability costing virtually nothing. Many music streaming applications like Spotify, Apple Music, YouTube and Google Play possess vast music libraries containing millions of songs, albums and playlists. The idea of playlist continuation is a system that aims to allow a user to keep listening to music once their playlist has finished. This continuation should contain songs that they like, whether it be songs in their library, songs they've listened to before or completely new songs.

There are many ways to undertake the problem of playlist recommendation. Large corporations like Spotify use an ensemble of these methods fine tuned to produce the best results. In recent years one method has stood out from the rest, deep audio features. This method involves using the individual music features of songs like tempo and energy and grouping them through their audible similarity rather than the generic genre method. The main benefit of this method is that newly released music can be utilized for recommendations, which is both beneficial to users and also small artists. The music recommending and streaming industry has made considerable moves to exploit this market with companies like Spotify spending a surplus of 100 million euro on a music intelligence company.

The aim of this project is to create an unsupervised recommender system that will focus mainly on deep audio analysis when making recommendations. This recommender system will be given a playlist of a set length and create a continuation of songs similar to that of the playlist. There is an abundance of ways to create this system and after extensive researching a solution has been drawn. First cleaning the data removing any abnormalities that will cause in-accuracy. Develop a rating to represent the popularity of a song. Clustering (grouping) the data using clustering methods in python libraries. Perform an extensive analysis on the data and any created clusters to find insight to help recommendations. Build a recommender system using all of the information learned that will have the ability to continue a playlist with a good accuracy. Finally the recommender system will be evaluated using set challenges with the dataset.
The dataset used in this project is provided by Spotify the music streaming platform. In early 2018 Spotify set a challenge to the world of programmers to create a playlist continuation recommender system to whom they provided a dataset. The dataset contains one million playlists and all their relevant information, inside these playlists was every single song and all the information about the song. This dataset is the content I will use in my project.

A recent statistic provided by Spotify to me about my profile was that I had listened to 46,000 minutes of Music in 2018, this is over 11% (2+ hours) of each day. Music is something I am very passionate about, as an avid user of Spotify I struggle to go a day without using the service and its recommender system. My personal goal in this project is to create a recommender system that I can use to my own benefit.

Chapter 2: **Background Research**

2.1 ACM RecSys Challenge 2018

ACM RecSys Challenge 2018 was organized by Spotify, The University of Massachusetts, Amherst, and Johannes Kepler University. Spotify is one of the most popular music streaming services in the world. One of its popular features is the ability to create playlists, and the service currently hosts over 2 billion. The challenge set out was to create a music recommendation system for automatic playlist continuation. Spotify released a public dataset of 1 million playlists containing all the info of each playlist including all of the songs inside.The competition was to be given a number of tracks outside the dataset and predict what tracks were missing from their playlist.The data was released January 2018 and the final submission late June, a conference and workshop held in Vancouver October 7th. A detailed description of the challenge can be found on the challenge site[1].

2.2 Automatic Continuation of Playlists

Playlists are usually positioned carefully in an ordered sequence of songs chosen to be listened together by the playlist creator. The problem to be solved automatic continuation of playlists is to create a further number arbitrary songs to a playlist that a user would like. It is a useful feature in streaming services and increases user engagement encouraging the creation of more playlists. Users surprisingly didn't expect a recommended list in a particular order or find it that important.Whereas the songs in a playlist and the direct song-song relationship transitions where more important (high tempo song straight to a slow tempo song). Kamehkhosh et al. [2018]
There are many music streaming companies that use automatic playlist continuation in their software. Spotify is the company that comes to mind when we talk of this, this playlist driven company prides itself in its playlist creation and continuation. It has many different ways available to implement this feature. The main method of employing automatic playlist continuation Spotify uses is that of its "Radio",every track,artist,album and playlists have their own individual "Radio Station".The station is comprised of similar songs, artists and albums. In most cases the continuation contains songs from your library or similar songs to what you have listened to before, in some cases the continuation of songs contain brand new and unheard of songs to the user which is why the service has such a large following. There are many ways to go about automatic continuation, the main factor to consider is the knowledge you have. Track information (known as metadata) like tracks audio and content information can be helpful in the process. A cold start can be a position you are put in for automatic playlist continuation, this means you have no knowledge of the user or their listening habits making any recommendations purely track based. User knowledge that could exist could be a users information and their music libraries. With a cold start every additional element aside from metadata will be vital in creating a better automatic continuation system. Using metadata popularity rankings can be formed and used in the system.Approaches made before include:Target Characteristics: of a playlist with musical attributes and metadata.A circular playlist: comprises all tracks given and consecutive songs as similar as possible.

[1] https://recsys-challenge.spotify.com

2.3 Current Recommender Systems

There are many different platforms that use playlist recommendation. The most popular services are Spotify and Apple Music. Spotify renowned to have one of the best recommender systems. Most platforms ask what music and artists you like when signing up to give recommendations from the get-go.

Spotify uses an ensemble of 3 main methods Collaborative Filtering, Natural Language Processing (metadata) and Raw Audio Features (the focus of this project). Spotify also uses hand-curated recommendations from employees, these recommendations normally come in the form of carefully created playlists. This ensemble of recommendation methods takes into account new music when recommending which is very important to artists and users alike. Using this ensemble all the flaws the exist with each method are avoided and the best recommendation is given. Spotify also uses user information like what songs you skip, what songs you listen to all of etc to help with recommendations.

Apple Music, uses a Love and Dislike feature for songs you play to help with recommendations. Like Spotify's 'Made For You' playlists it also has a 'For You' section that contains many different playlist mixes like chill mix,friends mix etc. Apple Music also incorporates the use of Siri its native system AI to help with recommendations, you can tell the AI to 'never play that song again' and it takes this into account.

Throughout this project I will mainly focus on the recommendations created by Spotify as they have one of the best recommender systems they provide new recommendations with the utilization of Audio Features, the dataset was also provided by Spotify. Spotify delivers many interesting different ways of recommending songs and artist to users. This is mainly contained in their 'Made For You' section. Each day they create a set for four 'Daily Mixes', these mixes cover a broad spectrum of music you listen to and contains music you like and new music for each mix. Each mix is categorized into different moods to give you as variation in you re listening as possible i.e: Acoustic, Rap, Dance.

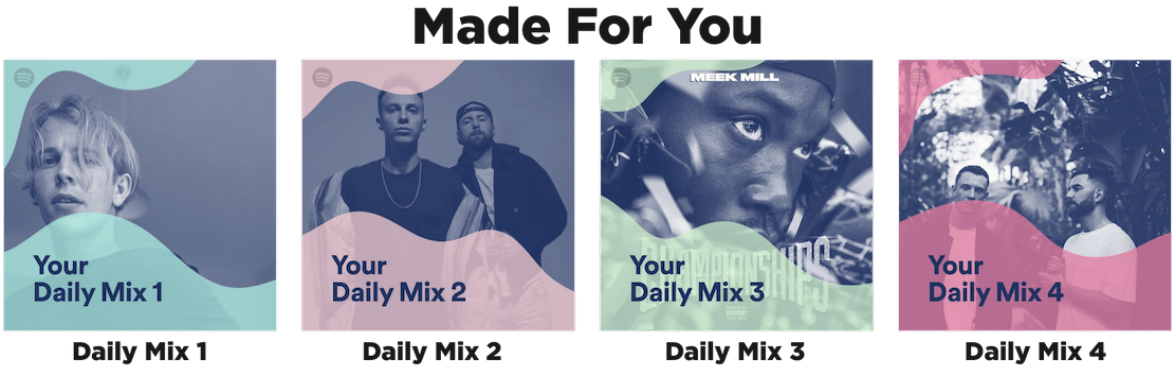


Figure 2.1: "Made for You" playlist, generated automatically by Spotify.

Although these mixes are very interesting Spotify's most renowned recommender system is called 'Discover Weekly' this service offers a playlist of 30 brand new songs a user have never heard before based on their listening habits in the past week. This is a feature that I personally use every week finding new songs to add to my library. They also have a playlist Release Radar' which contains newly released songs from artists you've listened to before and that are stored in you're library.

Figure 2.2: Spotify's renowned Discover Weekly and Release Radar.

1. **Collaborative Filtering**

   Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences of many users with a similar taste and using historical usage data.

   Can be very useful with music. As normally people that listen to the same music have similar music taste. This method can be flawed, relies on usage data, popular songs playlists much easier to recommend. No new or unpopular recommended even if you would like it. Results boring and predictable.

   Music specific problem, albums can contain intro, outro, remixes etc. This could be atypical of a users taste but be in an album of a favorite artist, skews results. What most basic services use. Spotify uses this technique in an ensemble comparing each of its 140 million users against each other. It looks at stream counts, if a user has saved a track to a playlist/library, visited the artist page songs before or if they like similar artists. If they skip songs etc. This data not provided for my specific project [2]

[2] http://benanne.github.io/2014/08/05/spotify-cnns.html

2. **Content-Based Recommendation**

   Content-based recommendation uses a users profile and item descriptions to make recommendations.

   Music companies recommender systems take into account song metadata: tags artist,album info, lyrics, text mined from web (reviews, interviews) and audio signal. There is a possibility of using Sentiment analysis with this information also to help recommendations. Use of text analytic methods like word clouds on playlists/playlist clusters/audio clusters could show a lot of information. Spotify recently acquired the company 'The Echo Nest' a music intelligence company for 49.7 million euro, this shows how important these systems are to large companies. Using audio signal can be difficult, its hard to identify genres sometimes as songs can sound similar. Differences like where the song comes from the mood and year of release can be near impossible It is assumed that when Spotify scrape all this data about songs they create a top terms matrices for artists and songs, this would include the adjacency to each term from the artist/song to determine if songs or artists are similar. The sound of a song plays a big roll in determining if you like a song, hence it cannot be ignored.

3. **Raw Audio Features**[Raw Audio Features]

   The main advantage of using Raw Audio Features to recommend music is because it takes into account new songs. Songs could be released one day and not be popular enough for a content or collaborative based approach to work but raw audio can produce this. In recent time there has been a huge emphasis put on deep audio analysis in recommender systems. Many companies just like 'Echo Nest' who focused on deep audio analysis have now been purchased for millions to be utilized by companies in the music streaming and recommendation business. There are many different types of features that can be extracted from audio files. The Spotify API[3] allows you to scrape these values (not included in the given Spotify dataset).

   They include:

   track_uri - the tracks unique identifier,

   acousticness - 0 - 1 confidence measure of the acousticness of a track,

   danceability - 0 - 1 confidence measure of the danceability of a track,

   energy - 0 - 1 confidence measure of the energy of a track (fast,loud and noisy),

   instrumentalness - 0 - 1 confidence measure of the instrumentalness of a track,

   key - Detected key of the song i.e: C/C#/D. If none found -1 assigned,

   liveness - 0 - 1 confidence measure of the liveness of a track,

   loudness - 0 - 1 confidence measure of the loudness of a track,

   mode - detects the modality(scale of melodic content) of a track Major-1,Minor-0,

   speechiness - 0 - 1 confidence measure of the speechiness of a track,

   tempo - int value representing the beats per minute (BPM) of a track,

   valence - 0 - 1 confidence measure of the valence (positivity) of a track,

   time_signature - estimated time signature of a track (how many beats are in each bar)

[3] https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

When using audio features the recommender system will attempt to find similar songs to make recommendations. If the audio features are rich enough, similarity can be computed over their audio features. Clustering by audio features will help identify new and interesting groupings ('audio clusters') that are not captured by genre labels. Applying this methodology to playlists rather than just songs will allow us to find similar playlists which could also help with making new recommendations.
The use of dimension reduction like PCA or t-SNE using these features could possibly be a method of clustering and categorizing songs using their Audio Features. These clusters could later be used for recommendations of similar songs. Another use of Audio Features could be used in automatic continuation. Every song has a tempo etc which dictates the speed of a song, this may rise and fall during an ordered playlist as the creator typically won't just have all upbeat songs or slow songs consecutively but rather create a pattern of high and lows. If a playlists features are compiled separately for every song in the playlist into a single format this could be closely mimicked in a recommender system for automatic continuation of a playlist.In the example graph below we can see the Tempo and Energy of the playlist 'Throwbacks', we can see how they rise and fall at different positions throughout the playlist. This pattern is created on purpose by the playlist creator and it is something that must be considered when making playlist recommendations.

In the two figures 2.3 below there two different types of playlists, a Throwbacks a playlist that contains older songs and Workout a playlist for the gym. There are many differences that can be seen between the two, the playlist Workout seems much more erratic than Throwbacks with its line chart fluctuating up and down frequently. It can be seen that the energy always seems to be larger than the tempo in Workout but this difference is less severe in Throwbacks. Workout playlists normally contain dance and fast paced music. Using this logic can we identify fast paced and upbeat songs if the energy replicates this format with other songs? Many comparisons like this can be made using the multiple columns available from the Spotify API. Creating new logical observations like this will be the foundation of the recommender system being created.

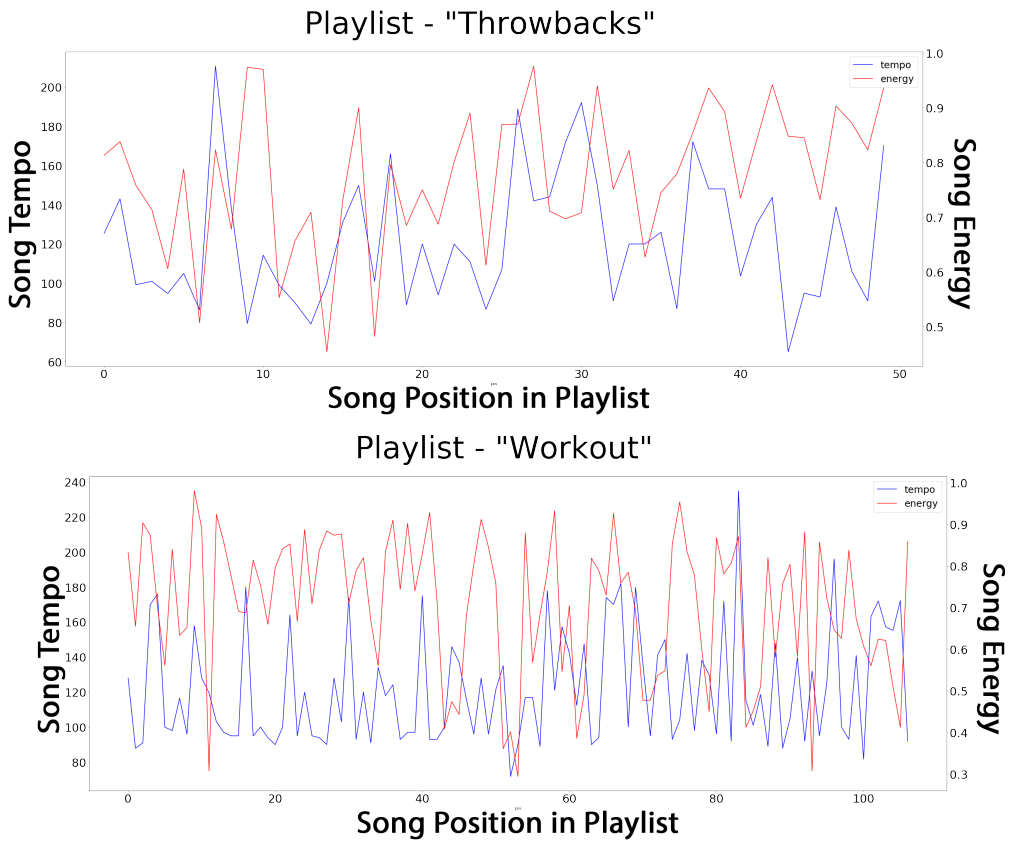


Figure 2.3: 'Throwbacks' Playlist vs 'Workout' Playlist

2.4 How do you define the popularity of a song?

The popularity of a song is very important when it comes to music recommendation especially on a cold start where you have no prior knowledge of a user.Schedl et al. [2017] You can use a popularity rating as an additional factor when creating a recommender system. This would be especially helpful in a content-based approach as it is an additional layer to consider. The downside of this method could be that the results seem a lot more predictable than with other approaches and mean less new music will be recommended to user. Although the majority of users don't mind listening to popular songs, they're popular for a reason. Taking this factor into account when creating a popularity formula could eradicate/lessen the problem.

The official definition of a popular song according to Merriam-Webster is :
"a song written and marketed with the intention of achieving mass distribution and sales principally in the form of recordings"[4]

Popularity of Songs and Popularity of Playlists are they the same?

When creating a measure of song and playlist popularity there are some considerations that need to be taken into account. One of the most important is that a Playlists popularity cannot purely be defined by the popularity of the songs as it contains. For example if a playlist contains only ten songs but they happen to be the top 10 songs in a dataset this doesn't mean the playlist is popular.When taking song popularity into consideration there are many factors that could be taken into account, the popularity of the playlists its contained in, the number of playlists its contained in, its similarity audibly to previous popular songs (according to charts).

A very interesting project called 'Song Popularity Predictor'[5] looks to compare the importance of audio features in popular songs. The researcher uses a dataset of a million songs and a scraped dataset of all the songs that have appeared on the Billboard charts since 1958. Using the billboard charts they created a "song hotness" rating which they compared to the given audio features. The project found the audio features tempo, mode and loudness to be very important with time signature being the least. As a result, using classifiers they were able to predict a songs hotness when compared to Spotify in a 60 percent confidence interval.

The Spotify Dataset

Using only the data provided in the dataset by Spotify there are a few good ways to go about acquiring a song and playlist popularity rating. There are two main points of interest that can be taken into account the number of followers owned by a playlist, the amount of playlists a song is included in. Other attributes like the position of a song (averaged by the number of tracks in a playlist) could also be taken into account as it is common people put their favorite songs and the start/end of a playlist, this although could lead to inaccuracy. As we already have a popularity rating for playlists (number of followers) the main task is to measure a song's popularity.

[4] https://www.merriam-webster.com/dictionary/popular%20song
[5] https://towardsdatascience.com/song-popularity-predictor-1ef69735e380

A popular song can be defined as popular when:

- The song appears in popular playlists.
- The song appears in many unique playlists.

To generate a song popularity rating it was important to create an original function that would compute the popularity of a song dependant on its use in playlists. The first step in this process was to create a non-weighted representation of popularity. To create this the total number of followers a song had (aggregated from the number of followers of all the playlist it is contained in) was then divided by the COUNT(*) (the number of playlists it appeared in). This would give the song an average popularity rating of all the playlists it was contained in.

$$Popularity = \frac{number\_followers}{COUNT(*)} \tag{2.1}$$

Using the rating for above the following weighted formula for calculating song popularity was derived. It was created by both using mathematical measures and experimental weighted values, the resulting formula was what felt the most correct (knowledge of other music charts and rankings were considered):
weight = 0.4, N = Number of Playlists

$$SongPopularity = \frac{weight * power(Popularity/PopularityMax), 0.05)}{(1 - weight) * power(COUNT(*)/N), 0.1)} \tag{2.2}$$

Why use this weighted formula?: The reason behind the weighted formula seen above is very simple. If a song was included in a very small amount of unique playlists yet the follow count on the playlists was very high according to the original *Popularity* formula it would be deemed very popular. In the figure 4.3 below we see the most popular songs when this formula was applied the problem is very evident.

track_uri	num_followers	num_tracks	COUNT(*)	popularity
spotify:track:669rlJ42fC74kLaNKXJNaF	22102	56	1	22102.0
spotify:track:3EAnWUBXPUDDmMJ8GBrqKt	22102	56	1	22102.0
spotify:track:7J57Z9jti07gqBLBrXkAn4	22102	56	1	22102.0
spotify:track:3OcnMblYvWBN4UHeET4bJ0	22102	56	1	22102.0
spotify:track:2Dqgu96bZ9fRLlo7ACYcXT	23502	29	2	11751.0

Figure 2.4: Popularity Rating

After applying a weighting scale that considered the number of playlists a song was contained in the results were much more fruitful and realistic as seen in the figure 2.5. Songs that had high follow counts but also high playlist inclusion counts were now ranked higher even though their *Popularity* isn't that high.

	track_uri	num_followers	COUNT(*)	popularity	Popularity_Rating_Normalized
0	spotify:track:7KXjTSCq5nL1LoYtL7XAwS	110286	46574	2.367974	1.000000
1	spotify:track:1xznGGDReH1oQq0xzbwXa3	93456	43447	2.151035	0.985832
2	spotify:track:7BKLCZ1jbUBVqRi2FVlTVw	89506	41079	2.178875	0.978242
3	spotify:track:7yyRTcZmCiyzzJlNzGC9Ol	83430	41309	2.019657	0.975882
4	spotify:track:3a1lNhkSLSkpJE4MSHpDu9	87291	39987	2.182984	0.974429

Figure 2.5: Weighted Popularity Rating

2.5 How many genres are there?

The number of genres that exist is very subjective and controversial. There exists many sub-genres for every genre ie: dance music contains progressive house which itself contains sub-genres like deep progressive house. The top 10 genres in current times could be said to be Electronic Dance, Rock, Jazz, Rap/Hip-hop, Rhythm and Blues, Techno, Country, Electro, Indie and of course Pop (Popular) music.

Genres in music in theory exist to assist labeling and marketing rather than a true description of the music. It could be said it is not a useful way to think about music as it narrows a persons exposure and taste in music. There tends to be a lot of overlap in music where different songs may actually be part of multiple genres.

A different approach entirely to genres would be that would be the use of **audio clusters**.Clustering songs into groups using just their audio features like tempo,valence and acousticness. This gives a completely different outlook on music, grouping them by their audible similarity rather than their hand labeled genres or their artists typical genre. This removes the problem of controversy of pinning a song to just one genre. With methods like this a slow rap-song could be grouped with an indie song if they are audibly similar. This helps for better recommender systems because the system is not limited to subjective genre labels rather audio clusters. This recommender system method can produce very interesting and fruitful results, Spotify's Discover Weekly is a great implementation of this.
When it comes to deciding how many genres/audio groupings to use it is very subjective and the best practice to find the correct number is to test results before finalizing a set number of groupings.

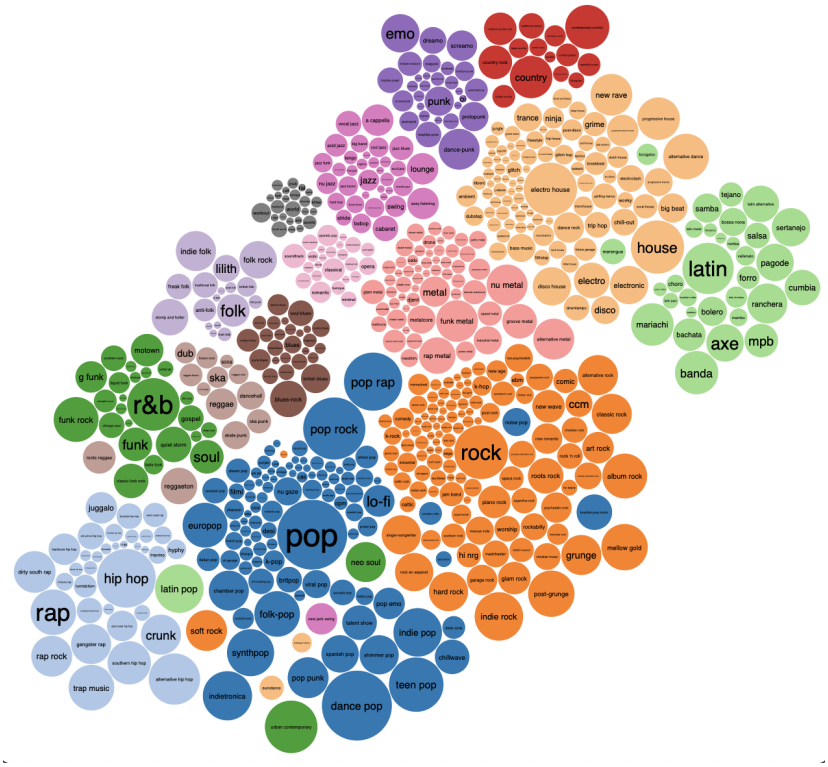


Figure 2.6: This interactive bubble cluster graph shows a website ran by the music intelligence company 'The Echo Nest' we spoke of above called Music Popcorn. It groups all the genres Echo Nest feel are identifiable and clusters them around a center genre. The interactive graph changes every time you pick a new center genre. It can be found at Echo Nest Popcorn.

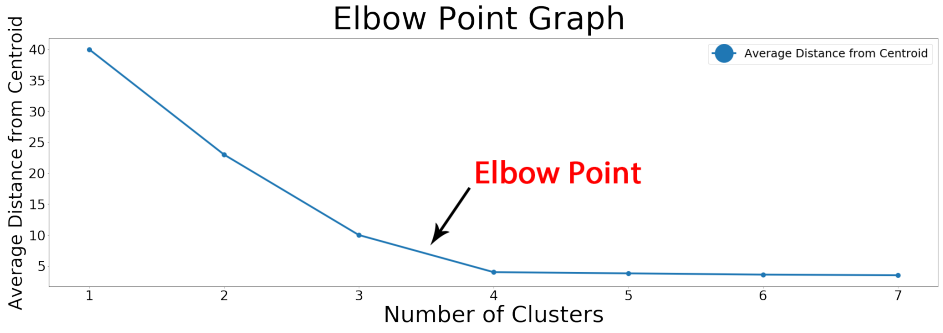
2.6 How should we group songs?

There are many different ways to go about grouping songs. Using metadata (Content-based), audio features or user data (Collaborative-Filtering) are all very valid and reliable methods. Using an ensemble of these methods could further increase accuracy. With the unique dataset provided by Spotify there are two main approaches that could be formed.

1. **Content-based Approach:** Using all the meta-data that can be accessed through the dataset and also though the Spotify API like artist names, song names, playlist names, playlist id's and all of the tracks included in each playlist a well formed structure can be formed. An example of how this could be useful would be if two of the same songs are contained in ten playlists together we can assume that if the first song is in a playlist it is likely that the user would also like the second song. This can be achieved by generating similarity matrices. Creating a similarity matrix for each song and it's distance to the next would an accurate representation of how much a song is suited to the next. Applying this to an entire dataset although time and memory consuming would give great results. When it is said that two songs are the most similar they could then be placed in a group, eventually each song would be then grouped.
2. **Audio Features Approach:** There are many ways to group songs using audio features. Clustering methods the most popular being K-Means and Hierarchical Clustering. When it comes to big data Hierarchical Clustering doesn't handle information as well whereas K-Means does. Hence K-Means is the more effective and better choice for a dataset with such magnitude as the Spotify playlist.

K-Means: K-Means clustering is an unsupervised learning algorithm type that uses unlabeled (ungrouped) data. The goal of the algorithm is to find K groups in the data provided, in the case of the Spotify dataset the audio features. It works iteratively to assign each data point in one of the K groups with the data given. Each data point is then clustered based on feature similarity. The algorithm then produces a centroid to each of the K clusters and labels all of the data to its identified cluster. The centroid of each cluster is the 'center point' to all of the items in each cluster and is the best representative of the cluster.
Each centroid defines a cluster, each created data point is assigned to the nearest centroid using a Euclidean distance function. The centroids are then recomputed iteratively by taking the mean of all the data points in each cluster until a stopping point.

Choosing the value of K: When deciding what the value of K is there are many factors that should be taken into account. In the case of the Spotify dataset, how many audio clusters do we want. Another thing to be considered is the "elbow-point". Increasing the value of K will always reduce the distance of data points to it's centroid. The "elbow-point" is when there is a sharp decrease in the decline of distance between the data point and it's centroid, at this stage we should use the value of K.



2.7 Visualization and Dimension Reduction of Clusters:

When using large datasets there tends to be multiple variables. Visualizing said multi-dimensional results can be difficult but tend to give good results once envisioned. This can be achieved by using techniques called dimensionality reduction. The main methods are Principal Component Analysis (PCA) and T-Distributed Stochastic Neighbor Embedding (t-SNE). In most cases PCA performs much better than t-SNE, PCA is deterministic which means its graphical representation is true until t-SNE which aims to create a better looking graph.

- **Principal Component Analysis (PCA)**
  The PCA technique is the best way of reducing the number of dimensions to 2/3 which humans can visually see whilst retaining the most information. It uses the correlation between dimensions and provides the minimum number of variables that keep the maximum amount of information of how the original data is distributed. Using a package provided by Python called sk-learn PCA is easily implemented. It is first imperative that all values being taken into PCA are normalized. The values are then passed through a function placing them in a matrix of distances to each other. PCA's fit transform function then converts the values into two separate components. These values can then be plotted in a graph. The example below shows a dataset that has been PCA reduced.

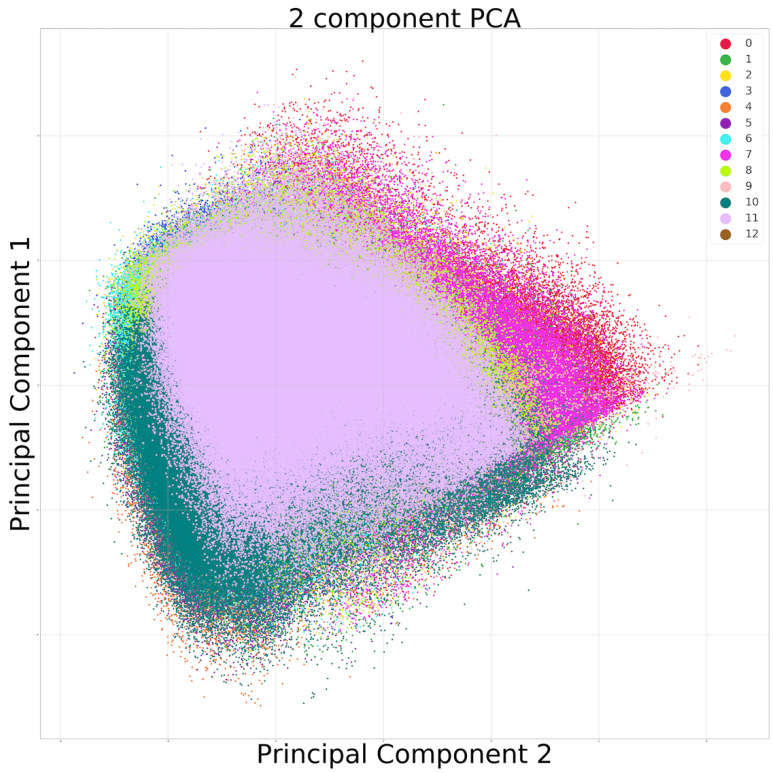


Figure 2.7: PCA Reduced Audio Features Visualization

- **T-Distributed Stochastic Neighbor Embedding (t-SNE)**
  t-SNE is another method of dimensionality reduction especially suited to datasets with high-dimensions. Rather than taking the mathematical approach using the Euclidean distance like PCA, t-SNE is a probabilistic approach to the problem. It aims to use the original data and represent it in the best way using fewer dimensions by matching both distributions. The method is computationally heavy which has severe consequences when using large datasets. Hence with that Spotify dataset this may not be the best option. Although a common compromise for this is to first reduce the dimensions to a lower amount and perform t-SNE then or to perform t-SNE on smaller random samples of the data.

2.8 Measuring the similarity of songs

Similarity measures in python measure the likeness of two data objects. They calculates the distance between different features of N dimensions and give back a number representation of objects in a range of 0 to 1. The smaller the distance the higher the degree of similarity and vice versa when the distance is larger the objects are more dissimilar. Similarity between objects can be very pliable to the information it is given so when inputting features to compare great care should be taken. One way of implementing this would be to normalize all features before similarity is computed or removing features that have little variation ie: Mode is a musical feature that is either a 0 or 1, it indicates the modality of track (major or minor).

Using the audio features and data mined features of each song it is possible to calculate their similarity. There are many different methods that exist to generate these calculations the most popular in Python are Euclidean Distance, Manhattan Distance and Cosine Similarity.[6]

Euclidean Distance

Euclidean Distance is the most commonly used measure of distance for similarity. When data is dense or continuous it is the best proximity measure. It uses the Pythagorean theorem to find the distance between two points, in the case of this project two songs and their respective features.

$$D(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{2.3}$$

Manhattan Distance

Manhattan distance is a metric that calculates sum of absolute differences of each objects co-ordinates as its distance. Given two objects it sums the absolute x and y variation and their differences.

$$D(M,P) \equiv |M_x - P_x| + |M_y - P_y| \tag{2.4}$$

Cosine Similarity

Cosine similarity finds the normalized dot product of the inputted features. Finding the cosine similarity of two objects finds the angle between two objects, it is very effective with multiple features. A cosine value of 0° indicates 1, a cosine of 90° indicates two objects that have a similarity of 0. Using an imported library in python **(sklearn.metrics.pairwise.cosine_similarity)** it can be calculated efficiently and with ease. In this project this similarity measure will be the most practical.

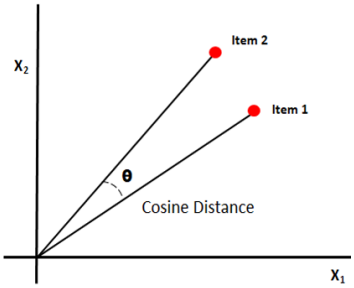


Figure 2.8: The cosine similarity $cos(x,y) = \frac{x \cdot y}{||x|| \cdot ||y||}$

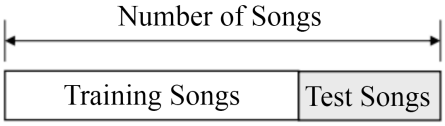
[6] http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/

2.9 How do we evaluate recommendations?

The two ways music recommendations can be evaluated. The first is by taking a sample set of songs from a given playlist and attempt to predict the remaining songs in the playlist. The second is to use a self evaluation of recommended songs to check if they are strong.

1. **Predicting Songs in a Playlist**

   This method of evaluation is a good means of testing that recommendations are strong. It could also be automated and used on a large scale of playlists with a simple loop in python. The most common way of using this means of evaluation is to take a train and test set from a given playlist. The training set consists of a larger number of songs (commonly 70%) than the test set (commonly 30%). Given the training set predictions are made the same size of the test set.

   To measure the success of these predictions we can use Precision and Recall. Precision and Recall are the most common and best means of model evaluation.

   - The **Precision** measure shows the correct predictions over all predictions.

   $$precision = \frac{\mid relevant\ songs \cap retrieved\ songs \mid}{\mid retrieved\ songs \mid} \tag{2.5}$$

   - The **Recall** measure shows correct predictions intersect all predictions over total correct predictions.

   $$recall = \frac{\mid relevant\ songs \cap retrieved\ songs \mid}{\mid retrieved\ songs \mid} \tag{2.6}$$

   This method of evaluation does come with its own flaws. The main reason for this is the inconsistency in playlists. When creating a playlist a user will add songs they like or songs they feel fit in the playlist. These songs may not always be predictable even if they are similar. Using very small playlists will also be hard to make predictions as the training set will be very small and may not be a true representation of the playlist.

2. **Self-Evaluation**

   A self evaluation is also a good way of evaluating predictions. By inputting a playlist of songs and checking the predictions you can assess the results to see if they are strong. For example if you predict a test set from a playlist containing Rap songs if the returned predictions are Rap songs and contain similar artists to that of the training set it could be said to be a good recommendation. In the case of this project a self evaluation is the better measure as the goal of this project is to create a **Playlist Continuation System** rather than simply predicting a test set of a given playlist. If predictions made are similar to that of the training set it will work well as a playlist continuation system. One problem with this method is that it will be very time consuming manually evaluating each prediction.

Chapter 3: **Data and Data Access**

3.1 The Dataset

The first step once I had all the data in dataframes was to analyze it, the results were very fruitful. There were four separate datasets provided at the begging of this project all in a .sql format which was later read into four separate dataframes.

1. **Playlists** reduce them all down, use example of pid and a real song
   This dataset provided by Spotify contained 1,000,000 playlists with 10 relevant columns.
   pid - the playlists unique id number,
   name - the name of the playlist,
   num_tracks - number of tracks in the playlist,
   num_followers - number of tracks for each playlist,
   num_artists - number of artists in the playlists,
   num_albums - number of albums in the playlist,
   modified_at - the time of which the playlist was last edited,
   duration_ms - duration of the playlist in milliseconds,
   collaborative - whether the playlist was a collaboration or not.
   Each playlist on average contained 66 tracks, 38 artists, 50 albums, 3 followers and was just over 2 and half hours long.

2. **Tracks**
   This dataset provided by Spotify contained the 2.26 million songs unique songs and all of the metadata for each track. The figure 3.1 shows three sample tracks from the playlist 'Throwbacks'
   track_name - the name of the track,
   track_uri - the tracks unique identifier,
   artist_name - the name of the artist who made the track,
   artist_uri - the unique identifier of the artist,
   album_name - the name of the album the track is contained in,
   album_uri - the unique identifier for the album,
   duration_ms - the length of the track in milliseconds.

3. **Track_elems**
   This dataset provided by Spotify contained 66.34 million songs with 3 columns. Track_elems contained every single song in each individual one of the million playlists and was the key to join each of the dataframes when needed.The columns where **pid** - playlist id, **pos** - the position of the song in the playlist, **track_uri** - track id. The dataset contained 2,262,292 unique tracks. The average song was 4 minutes and 12 seconds long.

	track_name	track_uri	artist_name	artist_uri
995	Heart Attack	spotify:track:1V6glisPpYqgFeWbMLl0bA	Demi Lovato	spotify:artist:6S2OmqARrzebs0tKUEyXyp
996	Who Says	spotify:track:3TcL0dyCMyr0kyTTc4NLgl	Selena Gomez & The Scene	spotify:artist:6dJeKm76NjfXBNTpHmOhfO
997	It Girl	spotify:track:4fINc8dnfcz7AdhFYVA4i7	Jason Derulo	spotify:artist:07YZf4WDAMNwqr4jfgOZ8y

Figure 3.1: Throwbacks

4. **Audio Features**
   This dataset was obtained by my supervisor using the Spotify API taking every unique song in the million playlist datasets audio features. It contained the columns :track_uri,acousticness, danceability,energy,instrumentalness,key,liveness,loudness,mode,speechiness,tempo,valence, time_signature which are detailed above [3 - Raw Audio Features].

Accessing the Data

To place the dataframes in a workable format there were many steps that had to be taken.The dataset was extremely large around 40GB in total, hence it was imperative that a server at the university was used to store and process and commands for speed and functionality. First the dataframes where all provided in a raw JSON file. Next it was processed into a relational database in SQLite and saved as a .sql file, this reduced the file size to 16GB making it easier to work with. Using this file you could then access each dataframe by querying the .sql file in Python Pandas and Juypter Notebooks using "pd.read_sql". Once read into a dataframe they were each saved into **pickle format** for quick and easy reading into new Juypter Notebooks. The files were in the correct format so implementing any calculations, graphs and clustering methods was much easier as I had access to python packages like pandas, matplotlib, sk-learn PCA(dimension reduction) and K-Means(clustering). The audio_features file scraped by my supervisor was also in a JSON format so I implemented the same methodology to place it in a dataframe.

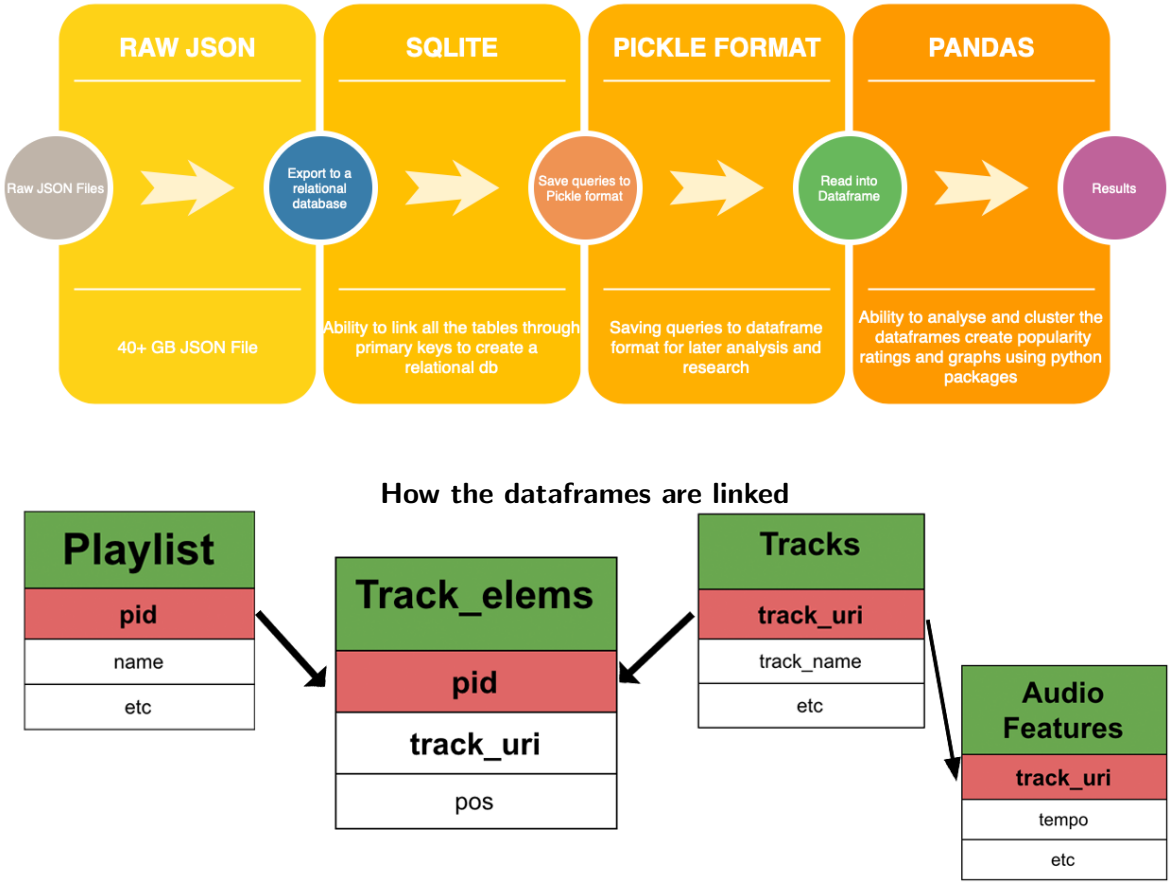


Figure 3.2: Data Access Process

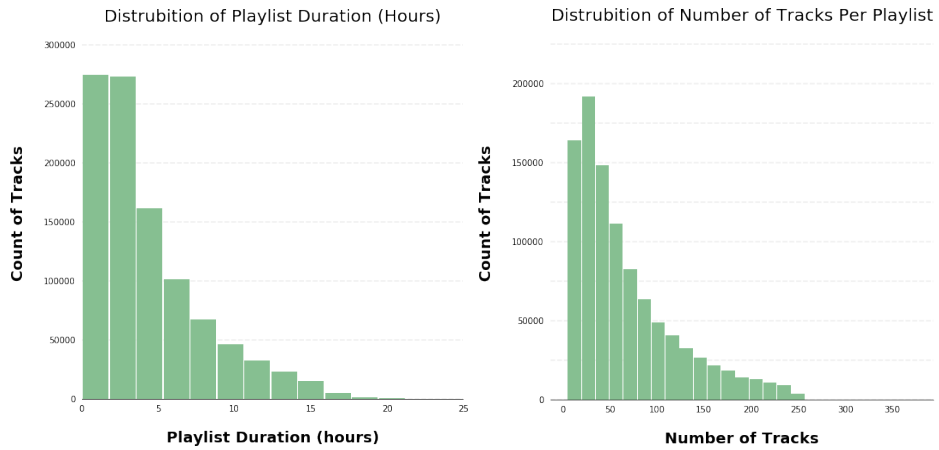
Chapter 4: **Core: Analysis and Data Mining**

4.1 Analysis of Dataset

It is key in a project like this using such a large dataset to have a good understanding of what it contains. Over the first few weeks once of having access to the dataset I performed analysis on each dataframe, merged dataframes and culminated results . Although the results were all small and not fruitful and as visually appealing as the PCA visualization of clusters it was still important to have this knowledge of the dataset.

Playlist Analysis The figures 4.1 below show two different distributions in the playlist dataset.

- The figure on the left 4.1a shows the distribution of the length of playlists in hours in the dataset. The max x value in the graph is 25 hours, this value was obtained after removing 113 other playlists with the longest playlist in the dataset being 176.6 hours which is over 7 days of listening time! The average length of a playlist was 4.3 hours long.
- The figure on the right 4.1b shows the distribution of the number of tracks in each of the playlists. The largest number of tracks in a playlist was 376 and the minimum was 5 which was a preset in the provided dataset. The average number of tracks in a playlist was 66.



(a) Distribution of Playlist Length **(b)** Number of Tracks Distribution

Figure 4.1

- The most popular playlist names were Country (10347), Chill (10196), Rap (8588), Workout (8581) and Christmas (8375).
  As this dataset is taken from Spotify's US customers Country being the most popular playlist name makes sense. Christmas is also a playlist name has a self explanatory amount of creations.
- 97.7% of playlists in the dataset were not Collaborative playlists meaning only one person had created it.
- Another very interesting insight is that 98% percentage of playlists had less than 5 playlist followers.

Track Analysis

- The average song length was 4.14 minutes in the dataset with the longest song being over 5.7 hours long and the shortest 30 seconds long (after removal of a small number of songs less then 30 seconds in data cleaning).
- The most occurring artists were Johann Sebastian Bach (5406) ,Wolfgang Mozart (5251), Ludwig van Beethoven (4308) who are all classical artists. The reason for these artists having so many songs is edits and remixes that use their melody accredit them the song. Some other honourable mentions include Frank Sinatra (1931) and Elvis Presley (1376).
- The most occurring album names were Greatest Hits (2167), Live (1273), Spotify Sessions (1159). Spotify Sessions is a recording studio owned by Spotify that release songs weekly of artists who come in and record their songs and covers of other songs.
- The most occurring song names were Intro (1358), Silent Night (945), Home (842), White Christmas (617), O Holy Night (554), Jingle Bells (537). With the song name Intro being attributed to artists placing the song at the start of their album a clear Christmas theme can be seen. This is due to so many covers of the same famous Christmas songs being recorded.

Track Elements Analysis

- Using this dataset the data feature **COUNT(*)** was data mined. This feature showed the number of occurrences of each song across all of the playlists. The most occurring song was the song HUMBLE. by Kendrick Lamar. This song was actually placed 8th in the USA Billboard charts in 2017 the year the dataset was taken from.

Audio Feature Analysis

As there is more advanced analysis of audio features as it was the main focus of my project they are detailed more in depth later in the section's Clustering (4.3,4.4,and 4.5). This analysis is a very basic interpretation of the data.

- A big insight gained when analyzing the audio feature dataset was that a small percentage of songs had 0 in some of their features. At first it was put down to incorrect data entry, but after delving deeper they were correct.It was mainly attributed to tracks being very short in length (less 20 seconds).
- Detailed in my notebooks **200_Average_Artist** and **210_Average_Playlist** are deeper insights into which artists and playlists contain the highest and lowest values for set audio features which gives some very interesting insights.
  For example I was able to identify many drum and bass playlists by finding playlists with the highest Tempo (BPM) on average. I also identified playlists containing white noise like rain in a forest etc by finding playlists with the highest instrumentalness on average

4.2 Popularity Rating

After I utilized the weighted formula for a Popularity Rating it was important to perform an external evaluation on the results. To do this I accessed the most popular songs from 2017 (The dataset provided finished in 2017) from **'chart2000.com'** and compared their rankings to my own. As you can see in the table below the rating generated from the weighted formula 'Popularity Rank' has similar figures to that of the 'Billboard Number' column. As there are over 2.2million tracks in the entire library this shows that the accuracy of the rating is very good as the lowest ranking with the weighted formula is 470 when compared to the Billboards. This is a tiny figure when you put it into this perspective of the 2.2million tracks, placing all the top 22 billboard numbers within **less than .0022% of the entire Spotify dataset**. This shows how accurate the rating system is and why it was applicable. The reasons why the ratings are not even more accurate could be attributed to a couple of things:

1. The dataset has playlists that were created in early 2017 hence popular songs from late 2016 would still be added into playlists which could skew the total accuracy of the songs.
2. Very Popular songs from the past could skew the accuracy i.e: Michael Jackson's songs.
3. Popular songs from particular seasons could change the results as they have a big following i.e: Christmas Playlists.
4. The ranking is purely follower based when the billboards are created other factors are taken into account that are not fully represented in the dataset.

The results from this ranking are very promising and we can now be confident to use this Popular Song rating when creating the recommender system.

Billboard_Number	Popularity Rank	artist_name	track_name
1	13	Ed Sheeran	Shape Of You
2	19	Luis Fonsi & Daddy Yankee	Despacito - Remix
3	10	Bruno Mars	That's What I Like
4	103	The Chainsmokers & Coldplay	Something Just Like This
5	43	French Montana & Swae Lee	Unforgettable
6	90	Imagine Dragons	Believer
7	4	Post Malone & Quavo	Congratulations
8	1	Kendrick Lamar	Humble
9	470	Charlie Puth	Attention
10	65	James Arthur	Say You Won't Let Go
11	386	Shawn Mendes	There's Nothing Holdin' Me Back
12	334	Ed Sheeran	Castle On The Hill
13	76	Kygo & Selena Gomez	It Ain't Me
14	187	Imagine Dragons	Thunder
15	16	DJ Khaled, Justin Bieber, Quavo, Chance The Ra...	I'm The One
16	106	Sam Hunt	Body Like A Back Road
17	118	DJ Khaled, Rihanna & Bryson Tiller	Wild Thoughts
18	5	Migos & Lil Uzi Vert	Bad And Boujee
19	2	The Chainsmokers & Halsey	Closer
21	15	Future	Mask Off
22	6	Lil Uzi Vert	XO TOUR Llif3

Figure 4.2: Popularity Ranking vs US Billboards 2017

4.3 Clustering Songs

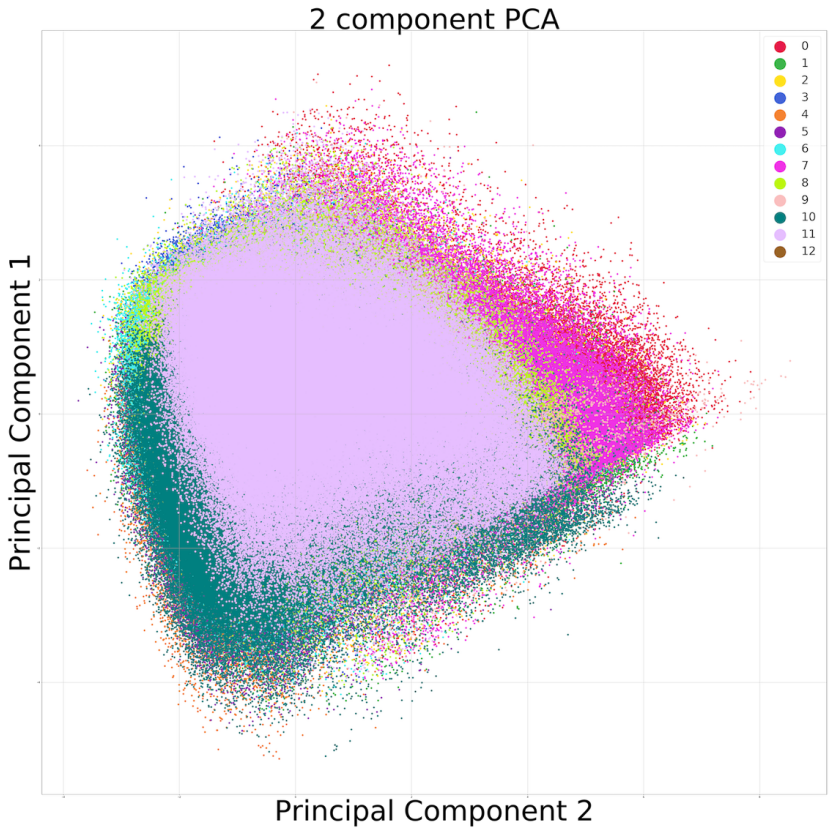


Figure 4.3: PCA Reduced Visualization of Clusters

To cluster the 2.2+ million songs in the dataset first the audio features for each one were extracted into a dataframe. It was important to remove audio features from the dataframe that had little impact as they skewing the clustering results. These features were mode which indicates if the song is in a major or minor key and the time signature (number of beats in each bar). After consideration the features were not normalized as the resulting clusters were inaccurate representations. Each song and their features were passed through pythons sk-learn K-Means package using 12 clusters. After experimentation 12 clusters gave the best results preventing overlapping audio clusters with similar and dissimilar songs in each. The largest cluster Cluster 6 contained 350,000 songs and the smallest Cluster 9 contained 3,000 songs. After achieving authentic clusters analysis was then performed.
The cluster centres of each cluster was then saved and PCA reduced for later addition to the recommender system.

To visualize these clusters each songs features where normalized and PCA reduced to two components using sk-learns StandardScaler and PCA packages.
The figure below shows the final representation of the PCA reduced song features, each colour represents a different cluster.

Very distinctive groupings can be seen between each cluster when viewed individually as seen in Fig: 4.4 the clustering technique was a success.

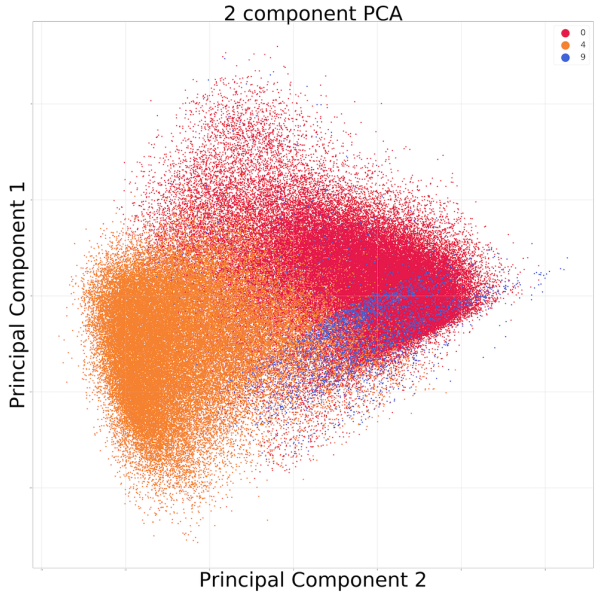


Figure 4.4: Principal component analysis of clusters 0, 4, 9

4.4 Comparing Artists

Here we compare different two artists, **deadmau5** a Progressive House DJ and **Ludwig Van Beethoven** a classical artist. Fig: 4.5 shows all of the songs from each of artist (deadmau5 in 'Yellow' and Beethoven in 'Blue'). It can be said that Beethoven has many more songs in the dataset compared to deadmau5, this is due to many artists accrediting remakes and remixes of his music to him. Each artist cluster is very distinctive but an overlap can be seen, this is due to some of deadmau5's songs containing very little lyrics and of a slower tempo.

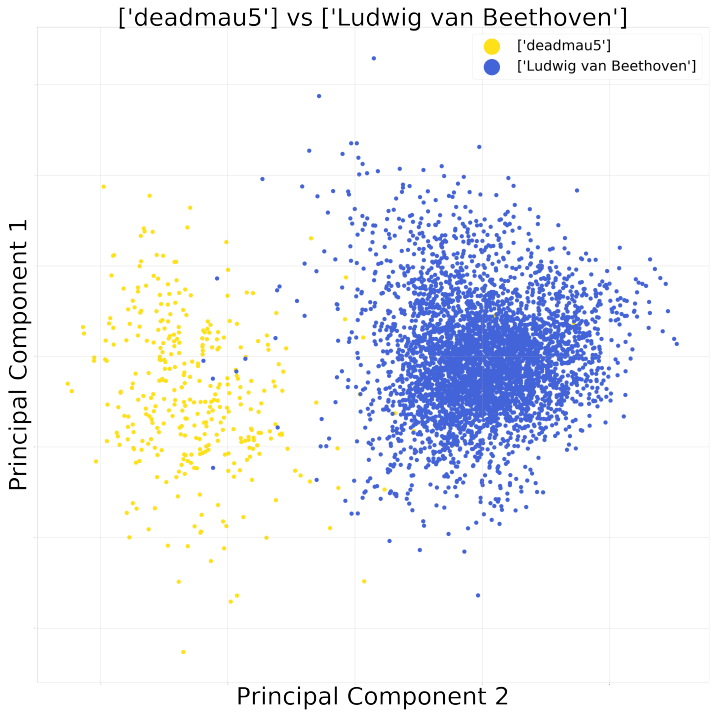


Figure 4.5: deadmau5 vs Ludwig van Beethoven

4.5 Cluster Overlap

In the below graph we can see a PCA reduced visualization of two clusters, the red cluster Cluster 3 who's top artists are all DJ's and the green cluster Cluster 11 who's top artists are classical musicians. AS clear difference between the two clusters with Cluster 3 located more to the top left and Cluster 11 to the bottom right. Although this difference is clear an overlap can be seen. After investigating into this overlap it can be seen that the top artists are deadmau5 and Porter Robinson. Both of these artists are Progressive House DJ's, progressive house music tends to contain songs that build up over time and contain very little lyrics. Hence this representation in the overlap makes sense as the artist's music could be said to be in both clusters.

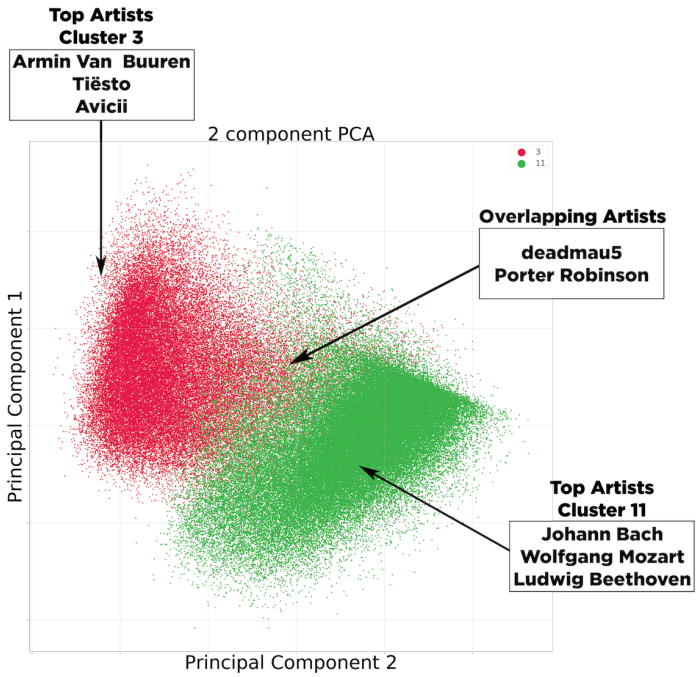


Figure 4.6: Overlapping Cluster Analysis

4.6 Clustering Playlists

Once again clustering was performed but instead using the average of each audio feature for every song in each playlist. Again K-Means clustering via sk-learn's K-Means package was applied. Every audio feature was PCA reduced and represented in the graph below with each colour representing a cluster. Although this method of clustering visually seems more realistic, averaging audio features from every playlist could be said to be a very crude measure of getting the playlist average. This can be attributed to the fact that not all playlists will have songs with similar audio features / genres which could skew results.

In the the figure 4.7 we can see the graphical representation of these results in a PCA reduced form. Although not the best means of representing each playlist using averages the visualization is very interesting.

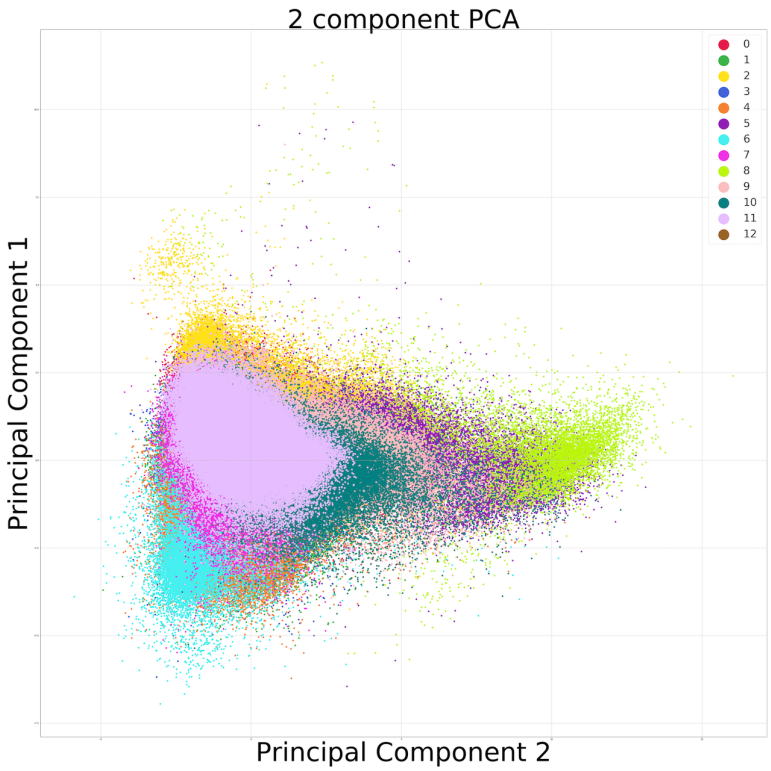


Figure 4.7: Clustered Average Playlist PCA Visualization

Next I identified individual clusters and created Word Clouds using the playlist names. These created fantastic results and showed a good representation of what each playlist contained. Fig: 4.8 shows two different playlist cluster Word Clouds, one classical and the other chill.

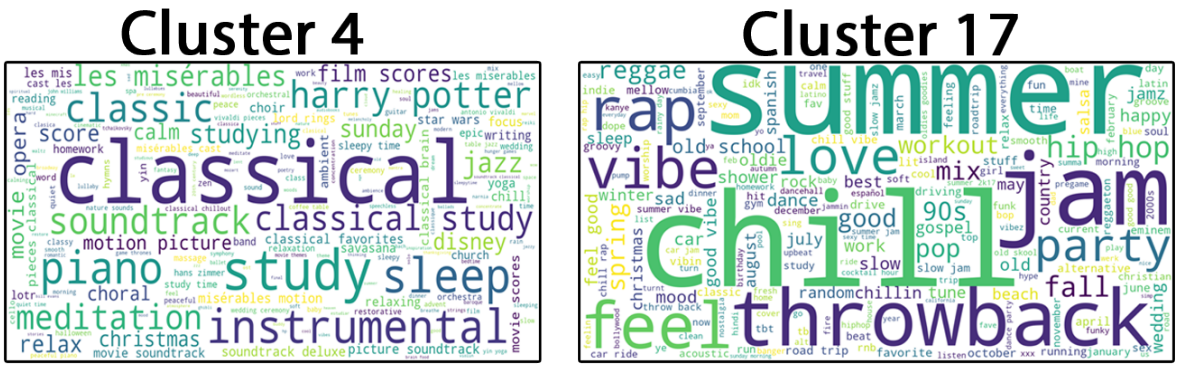


Figure 4.8: Wordclouds built from the text of the playlist names

Chapter 5: **Advanced: Recommender System**

5.1 The Recommender System Process

For each recommender system the same process is performed. First both the cosine and track information dataframes are ordered by the track_uri as they both will have the same index value. A dataframe of N songs is then provided and all of the features to be inputted are normalized using sklearn's MinMax Scalar. Next the normalized dataframe is passed into the function "Recommender_System". The function takes the dataframe and gives the correct index value to each song for later use. It is then split into train and test dataframes (70/30), and the training dataframe index values are passed into a Top N function. This Top N function calculates the cosine similarity each individual song every song in the dataset and finds the smallest cosine value using it as the recommendation. The function recommends a number of songs the length of the train-set and then from this a random sample the length of the test-set is taken as a final prediction.This is visually represented in the figure 5.1 below displaying each step in the recommendation process.

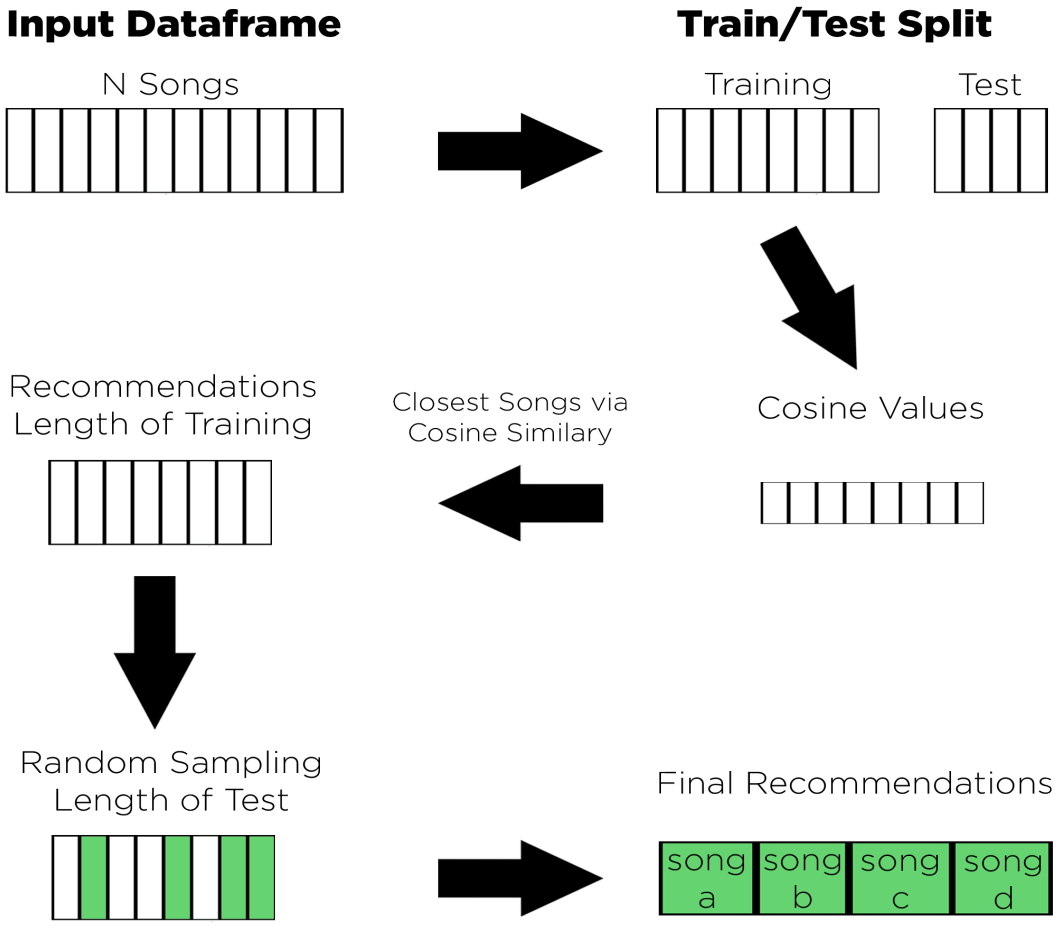


Figure 5.1: Recommender System Process

5.2 Created Recommender Systems

After setting a process to be used to gain recommendations the next set was to decide what features to include to test which would have the best results. Using the audio features of each song and their respective Cluster and Popularity Rating five recommender systems were created. The first using only audio features, the second using audio features and popularity ,the third using audio features popularity and the PCA reduced cluster values and the fourth is a song recommender that would recommend the best 10 songs for an entry and the final is a playlist continuation system.

The five recommender systems created where as follows:

1. **Audio Features for playlist continuation:**

   This is the most basic recommender system. This recommender system stays true to the audio features of a song and recommends the closest sounding songs to that of the playlist. Using only these features of each song the recommender system finds the cosine similarity of each song in the playlist and samples from these predictions a number of tracks the length of test set to get its final predictions.

2. **Audio Features and the Popularity for playlist continuation:**

   This recommender system takes into account audio features and the popularity of a song also. This recommender system is more likely to recommend a song of a similar popularity to that of the songs in the playlist. Using these features the recommender system finds the cosine similarity of each song in the playlist and samples from these predictions a number of tracks the length of test set to get its final predictions.

3. **Audio Features, Popularity and Cluster Centers for playlist continuation:**

   This recommender system uses audio features, popularity and a PCA reduced value for each of the data mined cluster values. It is more likely to recommend songs that are contained in the same cluster. Using these features the recommender system finds the cosine similarity of each song in the playlist and samples from these predictions a number of tracks the length of test set to get its final predictions.

4. **Song Recommender using all 3 features:**

   This recommender system takes only one song. Using all of the provided and mined features it generates songs of the closest cosine similarity to the song, these are the top 10 recommendations.

5. **Playlist Continuation System:**

   This Playlist Continuation System takes a random playlist in the dataset and continues it for the next N songs. It uses all of the features available and gets a set amount of recommendations for each song in the playlist and samples and recommends N of these songs as the playlist continuation.

5.3 Evaluation

5.3.1 Precision and Recall

Using the recommender process detailed in figure 5.1 this was then repeated in a python loop for 100 sample playlists. Each of the previous recommender systems described where used. Each time the precision and recall value was obtained it was appended to a list. After completion the average value for each was calculated as the performance of each recommender system.

As seen in the figure 5.2 below the first recommender system using only audio features has very poor performance and struggled to obtain any correct recommendations.
The second recommender system including Popularity ranking sees a large increase in the precision and recall by over .06.
Finally once the PCA reduced cluster centers for each song was added the precision and recall averages increased once more.
This proves that the two data mined features Popularity ranking and Clusters created stronger predictions. Popularity rankings gave the highest increase in precision and recall, it could be argued that this is due to the majority of playlist creators picking songs in general of a higher popularity.

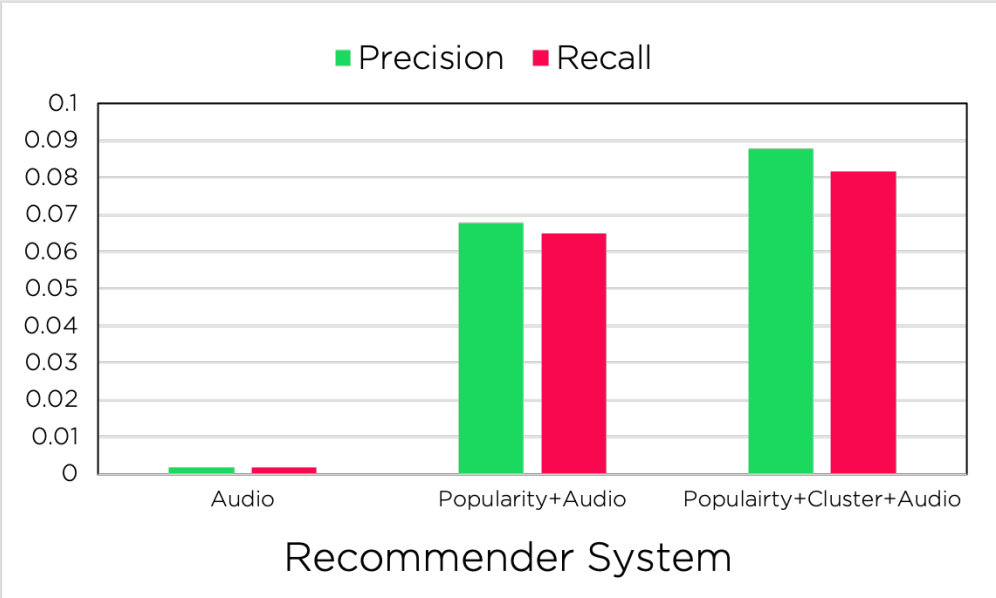


Figure 5.2: Precision and Recall Evaluation of Recommender Systems

Although, these results should be taken with a degree of skepticism. Despite the precision and recall values being very low predicting a small amount of the test set, as I have already mentioned these recommendations may not actually be weak but have fallen victim to that of the artist curator. The curator may have included songs in a given playlist with all their favorite songs, they may not have been of a similar audio cluster or popularity rating. This is a common understanding that I have learned since undertaking this project.
Hence, the next option would be self evaluate recommendations.

5.3.2 Self-Evaluation

There are many different ways to perform a self-evaluation. Recommending using songs that I personally have knowledge of and like is the most obvious method of performing this.
The first attempt was to create recommendations for only one song in a various array of genres. As seen in figure 5.3 below I have created 5 recommendations using my song recommender system that takes into account all audio features and data mined features for the song **Take Care** by **Drake** a famous rapper. The recommendations are in a descending order with the first song being that to be recommended.
The most apparent observation that can be made in these recommendations is that all of the artists are also popular rappers like Childish Gambino and Kendrick Lamar.
The 2nd recommendation made to the song is actually another song by the artist Drake, this recommendation has been made with no knowledge of artists yet has the ability to identify them using their audio features.
Another observation that can be made is that all of the songs recommended range between 74-91 in their respective Popularity ratings. This is very similar to that of the 82 of Drake's song.

track_name	artist_name	album_name	Popularity_Rating_Normalized
Take Care	Drake	Take Care	0.824830
Heartbeat	Childish Gambino	Camp	0.795680
Make Me Proud	Drake	Take Care	0.800857
Really Really	Kevin Gates	Islah	0.869861
Check	Young Thug	Barter 6	0.749550
DNA.	Kendrick Lamar	DAMN.	0.929789

Figure 5.3: Drake - Take Care Song Recommendations

The second attempt takes 20 random songs from the famous and popular DJ's Avicii, Calvin Harris, Swedish House Mafia, Tiesto and deadmau5 and puts them in a playlist dataframe. It then recommended 5 songs using all the tracks in the playlist. As seen in figure 5.4 the results are very strong.
Two of the five recommended songs are owned one of the 5 DJ's listed deadmau5.
The other three artists included Steve Aoki, Gryffin and Phantogram are also all other popular DJ's hence these 5 songs are good recommendations.

track_name	artist_name
ILYSM	Steve Aoki
Strobe - Club Edit	deadmau5
Turning Into Stone	Phantogram
King - Gryffin Remix	Years & Years
Animal Rights	deadmau5

Figure 5.4: Five Songs Recommendations for Popular DJ's

5.3.3 Comparing Against Spotify Recommendations

Another method of evaluation is to compare my recommendations against that of the recommender system used by Spotify. The song I chose to perform this test on was **Strobe** by **deadmau5** the Progressive House DJ who has featured throughout this report.

In the figure below 5.5 we can see Spotify's recommendations for the song Strobe. Its top recommendation for the song is another song by deadmau5 (underlined in red) so it is clear that Spotify clearly use artist based recommendations. Spotify's 5th recommendation is the song Scary Monsters and Nice Sprites by Skrillex, this is a bad recommendation as this song is very upbeat unlike Strobe and artist is known for his Dubstep music.

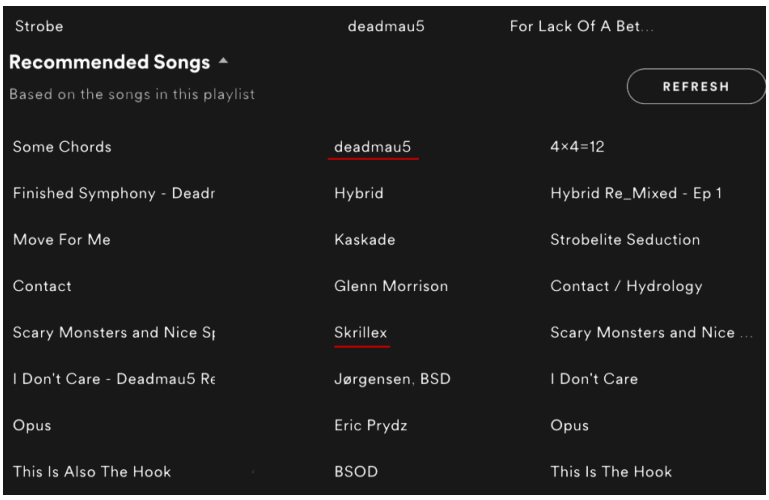


Figure 5.5: Spotify's Recommendations For deadmau5 - Strobe

In the figure 5.6 we can see the recommendations made by my system. The first recommendation for the song is the exact same song but the Club Edit which is slightly more upbeat but still very audibly similar, this shows the recommendations are staying true to its features. The recommender system has also recommended two other songs by deadmau5, this has been achieved with no knowledge of the artist showing it is able to identify an artist using only its audio features. Other popular DJ's like Martin Garrix and Tiesto have also been recommended. When we look at the cluster of the songs recommended it can be seen that only nine of ten are in the same audio cluster as the song.

track_name	cluster	Popularity_Rating_Normalized	artist_name
Strobe - Radio Edit	2	0.584544	deadmau5
Strobe - Club Edit	2	0.543267	deadmau5
See - Beacon Remix	2	0.359319	Tycho
Cole's Memories - Original Mix	2	0.388374	Pyramid
Need You Now	10	0.461531	Hot Chip
Paradise	2	0.489216	Tiësto
Super Skunk - Bart B More Remix	2	0.461389	Noir
Project T - Martin Garrix Remix	2	0.400204	Dimitri Vegas & Like Mike
Bad Wings	2	0.492216	The Glitch Mob
Terrors In My Head	2	0.483570	deadmau5
deadmau5 Remix-Cubrik Re-Edit	2	0.459790	James Talk

Figure 5.6: My Recommendations For deadmau5 - Strobe

5.3.4 Playlist Continuation System Self Evaluation

The main task of this entire Final Year Project was to create a playlist continuation system. The system would take a set playlist and my task was to recommend the next N songs using the track information and audio features of each song as seen clearly in figure 5.7 below.

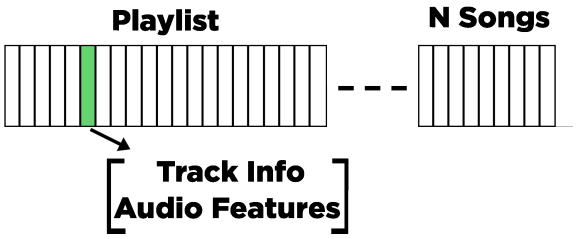


Figure 5.7: Playlist Continuation Visualization

As it is impossible to run a precision and recall test on the playlist extension system the only option was to perform a self evaluation.
To perform this I followed the same structure as the previous recommender system with some slight changes. First a random playlist was chosen from the entire dataset. Next a value for N was set and a number of recommendations where made for each song in the playlist. The system would then choose at random a number of songs the length of the value N. This then produced a continuation dataframe of the playlist of length N.

Random Playlist

track_name	artist_name
My Everything	Ariana Grande
The Gift	Pia Mia
Humble	Kendrick Lamar
good kid, m.A.A.d city	Kendrick Lamar
x	Ed Sheeran
Trilogy	The Weeknd
	Ed Sheeran
My Everything	Ariana Grande
The New Classic	Iggy Azalea
Yours Truly	Ariana Grande
Trilogy	The Weeknd
BEYONCÉ [Platinum Edition]	Beyoncé
Nothing Was The Same	Drake

Figure 5.8: Random Playlist

Playlist Continuation

artist_name	track_name
Migos	Call Casting
Kendrick Lamar	Ronald Reagan Era
Lil Wayne	A Milli
Justin Bieber	Eenie Meenie
Eminem	Berzerk
Carly Rae Jepsen	Run Away With Me
Lil Wayne	Mr. Carter
Chris Brown	Don't Wake Me Up
Lorde	Team
Travis Scott	through the late night
The Weeknd	Rockin'
Katy Perry	Unconditionally
U2	I Still Haven't Found What I'm Looking For
Post Malone	Feel

Figure 5.9: Playlist continuation

In the figure 5.8 we can see a sample from the random playlist containing artists like Kendrick Lamar, Ed Sheeran and The Weeknd. In the right figure 5.9 we see a sample of the continuation of the playlist containing similar artists like Kendrick Lamar, Travis Scott, Post Malone and The Weeknd. From looking at the sample of the playlist it can be said it is a Rap/Pop playlist, this has been replicated in the playlist continuation system recommending the same and similar artists. The full results for this experiment is contained in the Juypter Notebook **500_Playlist_Continuation_System**

Chapter 6: **Conclusion**

It is clear that music is very important in the daily lives of a multitude of people. They have a strong passion for music. Music Discovery is an essential part of keeping the passion alive. Music recommender systems have become an everyday part of any everyday music lovers life and are very important for artists in the music industry to grow. The recent emphasis on recommending systems using deep audio features like Spotify's Discovery Weekly to recommend fresh music is paramount for the growth of new artists who don't own the same fame as popular artists.

6.1 Project Review

Over the course of this project many different technical methodologies have been applied, they included Data Analysis, Data Mining and the creation of Recommender System Algorithm.

Analyzing the Spotify dataset returned many interesting results and gave great insights that were later used and implemented in the recommender system for example Information Gain of audio features in the dataset.

Data Mining produced two very strong features that significantly improved recommendations. Popularity Rankings [4.2] which were proven to be very robust and comparable to that of the US Billboard Rankings in 2017 [4.2]. K-Means Clustering [4.3] produced excellent audio clusters that once inspected produced interesting insights to the data, some of which can be seen in figures [4.4] and [4.5].

After creating five different recommender systems each more fine tuned then the last using audio and data mined features predictions became superior. The final implementation achieved the goal of creating a Playlist Continuation System for a given playlist providing N recommendations.
After Precision/Recall [5.3.1] and Self-Evaluation [5.3.2] noteworthy recommendations were produced further solidifying the strength of the recommender system.

This project **Recommending Song Playlists** aimed to create a **Playlist Continuation System** which was a success. The recommender system produced strong automated recommendations with no knowledge of artist or track data but only using audio and data mined features which was the goal set at the beginning of this project. Using these features the recommender system now stands up and is comparable to that of recommendations by global corporations like Spotify and Apple Music.

Music is extremely important in my life, I listen every day and use of Spotify's Discover Weekly recommender system frequently. I am thrilled to have created a recommender system mimicking that of Spotify's with a limited dataset and time. Having a vested interest in this project has helped me maintain focus and keeping me engrossed in my work constantly thinking of new ways to improve the system.

6.2 Possible Improvements

Although the recommender system I created performs well and gives satisfactory results there is always possibility of betterment to create even more accurate recommendations. As time was limited in my final year project there are still more improvements that could have made to obtain superior results.

These improvements include:

Artist Based Similarity
To perform this similarity metric you find the similarity between different artists. Common methods of this metric would be to evaluate using a collaborative filtering method finding similar musics that like similar artists. You would then recommend artists that one of the similar users artists they haven't heard before that the other similar user likes. A very interesting paper written in National Taiwan University looks into this metric in depth, Lin et al. [2014].

As there are limitations in my dataset with no user information a method of performing this metric would to look for artists that occur in the same playlists, the more often the artists occur in the same playlist the higher their similarity rating.

Playlist Inclusion Similarity
This similarity metric would take into account the amount of times two songs occur in the same playlist together. For example if the song HUMBLE. by Kendrick Lamar occurs in the same playlist as Drake's song God's Plan four hundred times it could be said they are more similar than two songs who never occur in the same playlist. I believe that this would be a very good way to fine-tune recommendations for the Precision and Recall Evaluations.

Weighting and Data Mining New Features
Another simple but beneficial way of getting stronger would be by weighting data mined features more strongly. For example increasing how similar Popularity rankings must be when making recommendations, this could intern remove bad recommendations of songs with lesser Popularity rankings.

Natural Language Processing (NLP)
Another method of creating stronger recommendations would be the use of NLP and Word2Vec. These methods take into account the track information to find similarity's in track and album names etc. This is a very interesting means of obtaining recommendations, an interesting article by Ramzi Karam of Towards Data Science looks into this in depth and produces compelling results, *Using Word2vec for Music Recommendations*.

Sentiment Analysis
A final method of creating better would be that of using Sentiment Analysis, sentiment measures the positivity of an input. Using playlists and track names to generate these ratings would be very influential generating a rating. When recommending songs you would recommend songs of similar sentiment values. A huge factor that could be taken into account in this would be the use of the audio feature "Valence". This feature created by Spotify also measures the positivity of a song, using all of these values in an ensemble would give better recommendations.

6.3 Acknowledgments

A special thanks goes to that of **Dr.Aonghus Lawlor** my Project Supervisor. Aonghus was a tremendous help in the completion of my final year project providing guidance and was always available when help was needed.

6.4 GitHub

This project in its entirety is on GitHub at the link : **Recommending Song Playlists**
or at url: `https://github.com/michaelmallon/Recommending-Song-Playlists`

This GitHub link contains a project README file, all Python Juypter Notebooks, a System Requirements File, Project Presentation and this Final Report.

Bibliography

Iman Kamehkhosh, Dietmar Jannach, and Geoffray Bonnin. How automated recommendations affect the playlist creation behavior of users. In *Joint Proceedings of the ACM IUI 2018 Workshops co-located with the 23rd ACM Conference on Intelligent User Interfaces (ACM IUI 2018), Tokyo, Japan, March 11, 2018.*, 2018. URL `http://ceur-ws.org/Vol-2068/milc1.pdf`.

N. Lin, P. Tsai, Y. Chen, and H. H. Chen. Music recommendation based on artist novelty and similarity. In *2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, Sep. 2014. doi: 10.1109/MMSP.2014.6958801.

Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. Current challenges and visions in music recommender systems research. *CoRR*, abs/1710.03208, 2017. URL `http://arxiv.org/abs/1710.03208`.