Spotify Recommendation System Analysis

Hands-On Project

April 8, 2023

0.0 Overview

This project aims to conduct a comparative study of various machine learning algorithms on Spotify recommendation system. It delves into the foundational principles of each model and examines their construction and evaluation using Spotify data. The empirical assessment allows for the observation of their performance and effectiveness in a real-world setting, providing valuable insights into the application of these models in music recommendation system.

Specifically, this project employs content-based filtering method and collaborative filtering method to investigate the underlying algorithms of Spotify recommendation system. For content-based filtering method, it utilizes various machine learning techniques including simple linear regression, linear regression with interactions, stepwise regression, decision tree, random forest, and neural network models to predict the streaming time based on song audio features. For collaborative filtering method, it utilizes Euclidean Distance and Cosine Similarity metrics to identify the most similar songs in terms of audio features based on either one or five target songs.

The results demonstrate that simple linear regression outperforms other models with the least RMSE and has a 30% accuracy in song recommendation. Both Euclidean Distance and Cosine Similarity metrics have a 40% accuracy in song recommendation. Additionally, this project also finds that the choice of test datasets can influence model performance.

1.0 Business Understanding

Task 1.1: Determine Business Objectives
Deliverable: Project Scope Document – Part 1

Background/Problem management wants to address:

Music is an important element of people's mundane lives. As the largest streaming service provider, Spotify is founded with a group of engineers back in 2006 and Spotify differentiates with other competitors (i.e. Apple Music, Amazon Music, etc.) with its heavy investment on research and development.

One of the charms that Spotify offers is the customization of the application based on the user's tastes and preferences. The customized features include recommendation system, user's top mixes or on repeat songs, user's favorite artists and user's preferred genres. Among those, recommendation system is an interesting topic to explore further for two reasons. First, listening to new songs that matched people's taste can certainly delight their days. It is worthwhile to research about this topic to see whether we are able to build a model with stronger performance. Second, compared to user's top mixes or on repeat songs and other topics, the recommendation system topic requires not only the user's individual streaming data but the generalized Spotify music whole dataset, which will probably serve better for the purpose of this course as we want to conduct a project with the goal of providing broader and more generalized benefits instead of only focusing on the individuals.

Business Goals:

The business goals for this project are: 1) explore the behind algorithms of music recommendation systems; and 2) optimize the recommendation system to benefit Spotify users.

Business Success Criteria:

The business success criterion for this project is to measure the accuracy the recommendation system will be based on the performance of models we've built. Based on our optimal model, we will come up with a better algorithm-based recommendation system for Spotify to apply to benefit all users. Details of the quantitative success criteria are elaborated in the "Data-Mining Success Criteria" part.

Constraints:

I personally think there will not be many constraints for this project in terms of staffing, technology, timeframes or money.

Organizational and Business Impact:

If we are able to provide a better recommendation system, it will bring a positive impact on Spotify. More users will subscribe as they enjoy using this app by having new good songs every week. The increasing subscriptions will increase the revenue of organization, which benefit the shareholders and employees.

Task 1.2: Assess the Situation
Deliverable: Project Scope Document – Part 2

Inventory of Resources:

1) Data
1.1) Individual dataset
1.2) Global Weekly Top 100 Songs
1.3) Generalized dataset*

*Previously I found one dataset containing all songs from Spotify with the audio features and popularity from 1921 to 2020 from Kaggle. I was planning to find a more updated dataset (till 2022). There is one dataset (i.e. The Million Playlist Dataset – dataset for music recommendation and automatic music playlist continuation) from Spotify R&D Research page. However, this dataset only has the information of track, artist, album, URI and duration but does not have the audio features of each song. Hence, it would be better for me to utilize the dataset from Kaggle.

2) Hardware
Macbook

3) Software

Python

4) People
Myself – Individual Project

Requirement, Assumptions, Constraints:

Requirement
A well-organized plan should be drafted in advance for this project. I have consolidated a plan table in the "Project Plan" part.

In terms of the definitions of acceptable finished work, I think finally I will submit a package which includes 1) Python codes 2) Word documentation 3) Slide 4) My thoughts/Tips for this project if applicable. I also want to regard this project as my milestone I've completed through this certificate program and put it as an individual piece of demo in my Github page.

Assumptions
I assume that there will not be much issue in terms of the legal and security obligations for the public data I've pulled and its usage as the global weekly top 100 songs dataset and generalized dataset are used among other data enthusiasts previously.
Meanwhile, it should be assumed that three datasets (i.e. individual dataset, global weekly top 100 songs dataset, generalized dataset) are factually based and there will not be much bias on it. There may be slight adjustments for the data but overall it should be justified.

Constraints
I personally think there will not be constraints for this project in terms of staffing, technology, timeframes or money.

Risks, Contingencies:

I do not foresee any risks related to incapability of getting data as I have already downloaded the necessary data for this project.

There may be some risks related to running the models. For instance, I may be unable to run the statistical models due to the code error. Or, owing to the limitation of the datasets, there may be some nuances of the results that we are not able to interpret fully. Those risks are quite normal in data science.

My contingency plans for dealing these risks are that: first, being more patient with the debugging process and doing more research to figure out the errors; second, accepting the results which we may not be able to explain due to the uncontrollable factors.

Terminology:

There will be a lot of terminologies for this project when it comes to the definitions of different statistical models, the criteria of determining the optimal model, explanations of some Python functions and packages and so forth. I think I will list out these terminologies in the documentation. For now, I can give one example of Decision Tree as below.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels.

I used the regression decision tree to solve the numerical data. I used the tree function and put all x variables in the right hand of the equation and y variable in the left hand.

Costs and Benefits:

As this is an individual data science project using the public data, fortunately there will be not much cost. There might be some costs with regard to searching materials which are not free, or some costs regarding outsourcing help as needed if I am unable to figure out the issue. Overall, the costs should not be material.

As for the benefits, if we are able to build a good recommendation system, for our own sake we will be able to enjoy better tailored music. Honestly speaking, nowadays I am depressed with the Discover Weekly as I am only able to have one or two or sometimes none of the songs among 30 songs per week. If I can explore new great songs by myself, that would be fabulous. I plan to run a Github page of myself and post my project there. People who are interested in this may be potentially utilize my models to find the songs they love. It is beneficial to the society. Additionally, as I've mentioned in the "Organizational and Business Impact" section, if we are able to provide a better recommendation system, more users tend to subscribe as they enjoy using this app by having new good songs every week. The increasing subscriptions will increase the revenue of organization, which benefit the shareholders and employees.

Task 1.3: Determine the Data-Mining Goals
Deliverable: Data Mining Scope Document

Data-Mining Goals:

For the content-based filtering method, the data-mining goal is to build a statistical model with the best performance.

For the collaborative filtering method, the data-mining goal is to find the variable with the least distance of the target variable.

Data-Mining Success Criteria:

For the content-based filtering method, the results will be measured by the RMSE (root mean square error).

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE is a frequently used measure of the differences between values predicted by a model or an estimator and the values observed. I will use the above formula of the RMSE as my indicator to determine the optimal prediction model.

For the collaborative filtering method, I will apply neighborhood collaborative filtering to use the similarity metrics method. Calculate the distance using all audio features available in the generalized dataset and find the neighbor songs which have relatively less distance. So the results will be measured by the distance of the neighbor songs with a target song.

Task 1.4: Produce a Project Plan
Deliverable: Data Mining Project/Resource Plan

Project Plan:

Based on my quick glance of the CRISP-DM and also the instructions of the following weeks' tasks, we will have a detailed documentation for each ongoing phase of the plan. Therefore, I only created a table below to show the rough plan of this project.

Tasks	**Start /End Date**	**Required Resources**	**Inputs**	**Outputs**
Data Preparation 1) Understand the data 2) Data transformation 3) Data validation 4) Data cleaning	Jan 29 ~Feb 18	JSON Excel Python	Manual observation and evaluation	Data ready for modeling
Data Modeling 1) Content-based modeling 1.1) Simple regression 1.2) Regression with interactions 1.2) Stepwise regression 1.3) Decision Tree 1.4) Random Forest *(to be finalized)* 2) Collaborative modeling	Feb 19 ~Mar 12	Python	1) Understand the model 2) Multiple modelling techniques 3) Interpret the results	1) Content-based modeling 1.1) Produce the optimal model 2) Collaborative modeling 2.1) Find the variable which has the least distance with the target variable

2.1) Neighborhood collaborative filtering/ Similarity metrics method				
Model Evaluation and Conclusion	Mar 13 ~Apr 1	Word	1) Understand the models 2) Ability to evaluate the performance of each model 3) Ability to interpret the model	Analysis and Conclusion
Documentation, Presentation Preparation and Wrap up	Apr 2 ~Apr 7	Word	1) Technical skills 2) Writing skills 3) Presentation skills	Report and Slide

Initial Assessment of Tools and Techniques

At this point, I think I have grasped a clear understanding of the tools and techniques that are needed for completing my project. However, this is just a high-level overview of my project and I haven't put it down to earth yet. If there are any issues which may cause a change in the tools and resources needed, I will find an alternative way and ask for the professor's approval first.

2.0 Data Understanding

Task 2.1: Gathering Data
Deliverable: Data Collection Report

There are three datasets that I will be using for this project.

- Individual Dataset
- Global Weekly Top 100 Songs
- Generalized Dataset

I verified that I have acquired all the above three datasets, that I tested the data access process, and that the data exists. I loaded the data into Python and verified that the tools are compatible with the data.

Outline Data Requirement

- Individual Dataset
    - From Spotify, simply go to your Account - Privacy Settings - Download Your Data. You will be able to request the dataset there. I requested the dataset on January 15th and received my Account Data on January 18th.
    - The data mining goal is to predict the streaming time based on the audio features.
    - For Streaming History data, the time range is from January 16, 2022 to January 17, 2023. For Your Library data, the time range is up-to-date.

    - The original dataset from Spotify is in JSON format. I transformed into CSV.

- Global Weekly Top 100 Songs
    - This dataset is retrieved from Spotify Charts[1].
    - The data mining goal is to find the tracks with the top 20 predicted streaming time. Those 20 tracks are regarded as the potential songs I will like. I will listen to these and determine whether they really match my taste.
    - Spotify updates the weekly top songs each Thursday. Thus, I downloaded the dataset of Week of January 12 (i.e. from January 6 to January 12) (link).
    - The format is in CSV.
- Generalized Dataset
    - This dataset is retrieved from Kaggle (link).
    - The data mining goal is to use the collaborative filtering method to find the song which has the closest audio features with the target song we've liked in our library.
    - The format is in CSV.

Verify Data Availability
I confirmed that the required data exist, and that they are available to be used.

Define Selection Criteria
- Individual Dataset: The original dataset received from Spotify is a personal dataset including information about user account, identity, payment, search queries, playlist, streaming history and library. For my project, I selected the relevant data, which are Streaming History and Your Library.
- Global Weekly Top 100 Songs: This dataset pulled from Spotify Charts is a dataset including information of rank, uri, artist, track, source, etc. of the global top 100 songs.
- Generalized Dataset: This dataset pulled from Kaggle is a dataset including the information of audio features, artist, name, uri, popularity of all tracks on Spotify from 1920 to 2020.

Task 2.2: Describing Data
Deliverable: Data Description Report

	Individual Dataset		Global Weekly Top 100 Songs	Generalized Dataset
	Streaming History	Your Library		
Source	Account		Spotify Charts	Kaggle
Formats	JSON→CSV		CSV	CSV
Number of Cases	31,989	946	100	170,653
The Number of fields	4	4*	7**	18

*The number of fields are to be updated after pulling the audio features of the track
** The number of fields are to be updated after pulling the audio features and the predicted_msPlayed of the track

[1] https://spotifycharts.com

	Description of the Fields
Audio Features	
danceability	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
energy	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
instrumentalness	Instrumentalness predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks.
key	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C/D, 2 = D, and so on. If no key was detected, the value is -1.
liveness	This value describes the probability that the song was recorded with a live audience. A value above 0.8 provides strong likelihood that the track is live.
loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0. A major key has a brighter and more upbeat sounds, while a minor key has a darker and more melancholic sound.
speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
duration_ms	Duration_ms is the duration of a track in milliseconds.
popularity	Spotify calculated the popularity of a track based on: 1) total streams of a track, 2) how recently a track has been played, 3) the frequency that a track has been played.
General Features	

artist	Artist represents the artist name of the track.
track	Track represents the track name of the song.
album	Album represents the name of the album that a track belongs to.
msPlayed	MsPlayed represents the streaming time of a track in milliseconds.
uri	The Spotify URI for the track. Uniform resource indicator is a link that you can find in the Share menu of any track, album, or Artist Profile on Spotify.
Other Features	
endTime	EndTime is the time when I end listening to the track. For example, 2020-01-16 23:31 means that I ended listening to a certain song at 23:31 on January 16, 2020.
rank	Rank represents the ranking of the tracks by a certain week.
source	Source is the music label of the track.
peak_rank	Peak_rank represents the highest rank of a track.
previous_rank	Previous_rank represents the previous rank of a track.
weeks_on_chart	Weeks_on_chart means how many weeks a track stays on chart.
year	Year indicates which year the track is released on Spotify.
released_date	Released_date indicates which date or year the track is released on Spotify.

From my perspective, the data is suitable for my data-mining goals. The data includes the expected fields and the cases are sufficient based on the number of cases I've shown in the first table.

From Streaming History dataset, I can sum the streaming time of each track I've listened during this year (i.e. January 18, 2022 to January 17, 2023). Then I will pull the data of the streaming time to Your Library dataset. After that, I will run different statistical models based on the audio features of each track and choose the model with the best performance. Then, I will use the optimal model to test in Global Weekly Top 100 Songs dataset. Using the audio features of each track in Global Weekly Top 100 Songs dataset, I can calculate the predicted streaming time of each track and select the top 20 tracks, which are my potential liked songs. By listening, I will be able to tell how many tracks are my actual liked songs.
In Generalized Dataset, they include the audio features of each song, so I will be able to use the neighborhood collaborative filtering to use the similarity metrics method which calculates the distance using audio features in the dataset and find the neighbor songs which have relatively less distance.

Task 2.3: Exploring Data
Deliverable: Data Exploration Report

- Individual Dataset – Streaming History
  - endTime
    - Range: from 2022-01-18 to 2023-01-17 (a whole year)
  - msPlayed
    - Range: from 12 ms to 1,729,654 ms (i.e. 0.0002 min to ~28.9 min)
  - Artist and track are categorical variables, which won't be further analyzed in terms of the ranges/distributions/summaries
- Individual Dataset – Your Library
  - Uri is used to pull the audio features of each track

  - Artist, album and track are categorical variables, which won't be further analyzed in terms of the ranges/distributions/summaries
- Global Weekly Top 100 Songs
  - msPlayed
    - Range: from 8,821,050 ms to 47,288,509 ms (i.e. ~147 mins to ~788 mins)
  - Uri is used to pull the audio features of each track
  - Rank related columns are not necessarily to be analyzed in terms of ranges/distributions/summaries
  - Artist, track and source are categorical variables, which won't be further analyzed in terms of the ranges/distributions/summaries
- Generalized Dataset
  - Year, uri and release_date are not necessarily to be analyzed in terms of ranges/distributions/summaries
  - Artist and track are categorical variables, which won't be further analyzed in terms of the ranges/distributions/summaries

  - Summarized Table

Audio Features	Range	Distribution
Danceability	In theory, from 0 to 1 In our dataset, from 0 to 0.988	N/A
Energy	from 0 to 1	N/A
Explicit	0 or 1	Discrete
Acousticness	In theory, from 0 to 1 In our dataset, 0 to 0.996	N/A
Valence	from 0 to 1	N/A
Instrumentalness	from 0 to 1	N/A
Key	from 0 to 11	Discrete
Liveness	from 0 to 1	N/A
Loudness	In theory, from -60 to 0 In our dataset, -47.046 to -0.671	N/A
Mode	from 0 to 1	Discrete
Speechiness	In theory, from 0 to 1 In our dataset, from 0 to 0.97	N/A
Tempo	from 0 to 215.918	N/A
Duration_ms	From 10,371 ms to 2,525,293 ms (i.e. ~0.173 min to ~42.088 min)	N/A
Popularity	From 0 to 100 (integer)	Discrete

Task 2.4: Verifying Data Quality
Deliverable: Data Quality Report

I have checked these datasets and confirmed that the data quality can support the goals of the project.
In these datasets, as I have roughly examined all the rows and calculated the ranges of the numerical variables. I do not think there are any major quality issues. There may be some minor quality issues which I may encounter during my data modeling phases but I think I will be able to figure them out. Under the worst circumstance where I need to change the project goals or plans, I will consult with the professor in advance if there are any serious quality issues with no adequate solutions.

3.0 Data Preparation

Task 3.1: Selecting Data
Deliverable: Data Rationale Report

There are three datasets that I will be using for this project.

- Individual Dataset
- Global Weekly Top 100 Songs
- Generalized Dataset

In the Data Understanding phase, the three datasets were collected and verified. Here I will specify which portions of the data collected will be used for data mining.

My data mining goal is exploring the behind algorithm of Spotify's recommendation system using two methods:

1. Content-based filtering method – building a statistical model with the best performance
2. Collaborative filtering method – finding the variable with the least distance of the target variable

- Individual Dataset
    - Spotify sent a package of my data as shown below. However, only two datasets are related to my data mining goal.
    - Rational For Inclusion and Exclusion

(Sub) Dataset	Inclusion/Exclusion	Rationale
Follow	Exclusion	This includes information of: 1) The number of followers the account has; 2) The number of other accounts this account is following;

(Sub) Dataset	Inclusion/Exclusion	Rationale
		3) The number of other accounts this account is blocking. The information is not related to my data mining goal.
Identifiers	Exclusion	This includes the identifier type and identifier value information. It shows how I register for Spotify (e.g. email or phone number). The information is not related to my data mining goal.
Identity	Exclusion	This includes my account name, the profile image and whether I am a tastemaker, and whether I am verified. The information is not related to my data mining goal.
Inferences	Exclusion	This includes my interests and preferences based on my usage of the Spotify service. Spotify uses this data to identify a list of market segments with which I am currently associated. Depending on my settings, Spotify may serve interest-based advertising to me within the Spotify service. The information is not related to my data mining goal.
Marquee	Exclusion	In the Spotify app I might periodically receive messages about new releases by artists and creators. The datasets regarding 'marquee' relate to these messages. They are based on my activity in Spotify app to be as relevant to me as possible. The information is not related to my data mining goal.
Payments	Exclusion	This includes the information of my subscription payment, which is not related to my data mining goal.
Playlist	Exclusion	This includes the information about my playlist. It is how I group the similar songs into one playlist, but not related to recommendation system. The information is not related to my data mining goal.
Search Queries	Exclusion	This includes a list of search made, such as the data and time the search was made;

(Sub) Dataset	Inclusion/Exclusion	Rationale
		type of device/platform used (i.e. iOS, desktop). The information can be indirectly related to my music taste and preference. For example, if I like a song, I may search it and save it in my library. However, there may also be a possibility that I search a song because I see its name from an ad or some friend recommends me. I will not necessarily like the song. Thus, I won't include this dataset.
Streaming History 0 **Streaming History 1** **Streaming History 2**	Inclusion (These three datasets are to be merged)	This includes the items (e.g. songs, videos, and podcasts) listened to or watched in the past year, including the data and time when the stream ended in UTC format, name of the creator, name of the item, msPlayed. This information correlates to my music taste and preference, and it is important for me to achieve my data mining goal.
User Address	Exclusion	This includes the information of the user address. The information is not related to my data mining goal.
User Data	Exclusion	This includes the information of username, email, country, whether it is created from Facebook, Facebook user ID, birthdate, gender, postal code, mobile number, mobile operator, mobile brand, creation time. The information is not related to my data mining goal.
Your Library	Inclusion	This includes a summary of the content saved in Your Library (e.g. songs, episodes, shows, etc.), including the entity names, creators, URIs. The songs in my library can represent my music taste and preference. They are related to my data mining goal.

*Further information of each dataset mentioned above can be found here[2].

- Global Weekly Top 100 Songs

[2] Additionally, the information of Identifiers, Identity, Inferences, User Address and User Data are based on my review of the data. The information of Marquee is based on my inquiry of Spotify Support team. The rest are from Understanding my data.

  - This dataset is retrieved from Spotify Charts[3].

  - This is a single dataset which I will include. The rationale for inclusion is that this dataset serves as a test dataset after I determine the optimal prediction model based on my individual dataset. This dataset is crucial for my data mining goal.

- Generalized Dataset

  - This dataset is retrieved from Kaggle (link).

  - This is a single dataset which I will include. The rationale for inclusion is that this dataset is used for the collaborative filtering method. I will apply neighborhood collaborative filtering to use the similarity metrics method by calculating the distance using all audio features available in this dataset and find the neighbor songs which have closet distance. This dataset is crucial for my data mining goal.

As discussed in Task 2.1, I have checked these three datasets and confirmed the data quality can support the data mining goals of the project and there are no major technical issues being identified.

Task 3.2: Cleaning Data
Deliverable: Data Cleansing Report

From my perspective, these three datasets are decently clean as I performed preliminary screens in Data Understanding phase. I did not track down sources to make specific data corrections, replace some data with default values. However, I excluded some cases or individual data or records, chose to use only a subset of the data for all or some the data-mining work as well as other steps.

As discussed in Task 2.1, there should not be much data quality problems for these three datasets. My choices of how I clean the data are consistent with my data mining goals and final outcome of this project.

Below documents in excruciating detail about every detail and action used to clean my data.

- Individual Dataset – Streaming History

  - Merged Streaming History 0, Streaming History 1 and Streaming History 3 into Streaming History

  - Removed rows which are 2022-01-16 and 2022-01-17, so the time range is the exact whole year (i.e. 2022-01-18 ~ 2023-01-17)

[3] https://spotifycharts.com

  - Removed the rows of 0 msPlayed

  - Renamed the columns for consistency throughout three datasets
    - Column1.endTime → endTime
    - Column1.artistName → artist
    - Column1.trackName → track
    - Column1.msPlayed → msPlayed

- Individual Dataset – Your Library

  - Chose a subset of the data - Only selected tracks and removed other rows (i.e. albums, shows, episodes, bannedTracks, artists, bannedArtists)

  - Removed Name column
    - Originally, the dataset includes a column (i.e. Name) which indicates tracker, albums, shows, episodes, etc. As I only kept tracks column, it would not be necessary to keep the Name column as all are tracks.

  - Renamed the columns for consistency throughout three datasets
    - Value.artist → artist
    - Value.album → album
    - Value.track → track
    - Value.uri → URI

- Global Weekly Top 100 Songs

  - Removed rows from 102 to 201
    - Originally, the dataset includes 200 pieces Global Top songs. As I only wanted first 100 songs as a test dataset, I removed rows from 102 to 201 and only kept the first top 100 songs.

  - Renamed the columns for consistency throughout three datasets
    - artist_names → artist
    - track_name → track
    - streams → msPlayed
    - duration → duration_ms

- Generalized Dataset

  - Removed explicit column
    - For the above first two datasets, I need to pull the audio features of each song, which I will discuss further in the Constructing Data section. Spotify Web API[4] allows us to be able to retrieve the audio features of a track based on the track URI. I found out that there is no "explicit" audio feature

[4] https://developer.spotify.com/documentation/web-api/

available in the page of Get Track's Audio Features[5]. However, in generalized dataset, there is "explicit" audio feature for each song. In order to keep consistency throughout three datasets, I decided not to include the "explicit" attribute.

- Renamed the columns for consistency throughout three datasets
  - Name → track
  - Id → URI

Task 3.3: Constructing Data
Deliverable 1: Data Attribute Report

- Individual Dataset – Streaming History
  - Created Sum_msPlayed attribute
    - The Streaming History dataset serves the purpose of building the Your Library dataset. Specifically, I need to create a new field (i.e. Sum_msPlayed column) of each track by using a pivot table. After that, I will use V lookup formula to transfer the Sum_msPlayed data of each track to the Your Library dataset.

- Individual Dataset – Your Library
  - There will be a new field (i.e. Sum_msPlayed column) in the Your Library dataset.
  - The Sum_msPlayed column will be simplified as the msPlayed column moving forward.
  - After transferring the msPlayed data, there are 46 pieces of track which does not have the data (i.e. #N/A) based on the Vlookup formula. I conducted a manual data validation screening and hard code the data. The reasons of #N/A include: 1) There is a bit nuance of the name of the track between Streaming History and Your Library; 2) I did not listen to some track for the period from January 18, 2022 to January 17, 2023.
  - Pulled the audio features data and created these attributes
    - Spotify Web API is an interface that programs can use to retrieve and manage Spotify data over the Internet. In my case, I need to retrieve the audio features of each track. From the page of Get Track's Audio Features, I am able to retrieve these by using the track's URI. However, it would not be ideal if I manually retrieved the audio feature data for all the 946 tracks from my library. Based on public search, I used Python to

[5] https://developer.spotify.com/console/get-audio-features-track/

perform audio feature data scraping from Spotify Web API to pull the data for all 946 tracks.

- So far, there will be a total of 17 attributes with 13 new attributes added in Your Library dataset.

Additionally, for the simplification, the Individual Dataset – Your Library will just be referred as Individual Dataset as we do not need Streaming History moving forward.

- Global Weekly Top 100 Songs
  - Similar as Individual Dataset, I pulled the data of audio features of the top 100 tracks.
  - To create pred_msPlayed attribute
    - Moreover, after determining the optimal prediction model, I will calculate the predicted msPlayed for the top 100 tracks and put them in descending order based on the predicted msPlayed. Then, I will be able to attain the 20 tracks with the top 20 predicted msPlayed. Those 20 tracks are regarded as the potential songs I will like. I will listen to these and determine whether they really match my taste. So, I will add a new field – pred_msPlayed.
  - So far, there will be a total of 22 attributes with 13 new attributes added in this dataset.

- Generalized Dataset
  - This dataset has included all the attributes that serve for my data mining goal. Therefore, there will be no further construction needed.

Deliverable 2: Data Generation Report

Based on my current standpoint, there are no new rows which should be constructed. All the datasets include sufficient number of records.

Task 3.4: Integrating Data
Deliverable: Merged Data Set

- Individual Dataset
  - Previous integration performed.
    - The original streaming history data that Spotify sent are in three separate dataset (i.e. Streaming History 0, Streaming History 1, Streaming History 2). I guess the reason why Spotify did that is due to the size of the dataset as the whole streaming history includes a user's whole year streaming history. These three streaming history datasets are merged as I've

mentioned in the Cleaning Data section. Also, I have performed some data constructing steps and transferred the data to Your Library, so there is no further step needed for integration.

- Global Weekly Top 100 Songs
    - This is a single dataset. No integration needed.
- Generalized Dataset
    - This is a single dataset. No integration needed.

Task 3.5: Formatting Data
Deliverable: Final Formatted Dataset

So far, I have made all three datasets consistent in terms of the attribute naming, data format (i.e. all csv) and so forth. Based on my understanding, there is no further reformatting action needed at this point. These three datasets are ready for modeling.

Below is an overview of what these three datasets will look like.

- Individual Dataset
    - 946 x 17 matrix (946 pieces of tracks, 17 attributes for each track)

Attributes
artist
album
track
URI
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
duration_ms
msPlayed

- Global Weekly Top 100 Songs
    - 100 x 22 matrix (100 pieces of tracks, 17 attributes for each track)

Attributes
rank
URI
artist
track
source
peak_rank
previous_rank
weeks_on_chart
msPlayed
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
duration_ms
pred_msPlayed

- Generalized Dataset
    - 170,653 x 18 matrix (170,653 pieces of tracks, 18 attributes for each track)

Attributes
year
artists
track
URI
release_date
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
duration_ms
popularity

4.0 Data Modeling

Task 4.1: Selecting Modeling Techniques

So far, I have completed the data understanding and preparation phases. These prep work helps ensure that I start the modeling phase exactly where I need to be with minimal omissions and 'do-overs'.

Prior to discussing modeling techniques, I would like to provide an overview of the definitions, descriptions, and assumptions of the different models I intend to utilize. This will help readers gain an understanding of the models, ultimately enhancing the readability of the main part of the document.

Deliverable 1: Modeling Definitions and Descriptions

- Content-based filtering method
    - Prediction Regression Models
        - Simple Linear Regression Model
            - Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine: 1) whether a set of predictor variables do a good job in predicting an outcome variable; 2) which variables in particular are significant predictors of the outcome variable and in what way they indicated by the magnitude and sign of the beta estimates impact the outcome variable.
            - The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.
        - Linear Regression Model with Interactions
            - Apart from what have been described above for linear regression model, there is an interaction between the independent variables. This means that the effect of one independent variable on the dependent variable depends on the level of another independent variable.
        - Stepwise Model
            - Stepwise regression is the step-by-step iterative construction of a regression model that involves the selection of independent variables to be used in a final model. It involves adding or removing potential explanatory variables in succession and testing for statistical significance after each iteration.

- There are three types of stepwise regression models. They are forward selection, backward selection and both selection. Forward selection starts from zero variable and adds up the variables until the results are optimal. Backward selection starts from adding all variables and deletes each variable until the results are optimal. Both selection is a combination of the forward selection and backward selection as it starts somewhere in the middle.

- Decision Tree Model
  - Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.
  - In terms of numerical data, it works by recursively partitioning the input variables into smaller intervals, which can then be treated as categorical variables. It selects a feature and a threshold value for that feature and splits the data into two subsets based on whether the value of that feature is above or below the threshold. In practice, however, decision trees may not perform as well on numerical data as on categorical data. The splitting process can be more complex with numerical data, as there are many possible ways to partition a continuous variable into intervals.

- Random Forest Model
  - Random forest is an extension of the decision tree algorithm, where instead of using a single decision tree to make predictions, multiple decision trees are used to improve the accuracy and stability of the model. In a random forest, each tree is trained on a subset of the training data, and at each split of the decision tree, a random subset of features is considered. This randomness is intended to introduce diversity in the model, which helps to reduce the chance of overfitting and improve generalization performance. Once all the trees are trained, the predictions of the individual trees are combined to produce the final prediction.
  - Random Forest can work well with categorical and numerical data. In terms of numerical data, it partitions the feature space into rectangular regions, making it effective for handling non-linear relationships between the input features and the target variable. Additionally, random forest can handle missing values and is relatively robust to outliers, which can be common issues in numerical data.

- Neural Network Model

- Neural network model is a type of machine learning model which is inspired by the structure and function of a human brain. It consists of interconnected nodes, or neurons that are organized into layers. Each neuron receives input from other neurons and produces an output based on that input, which is then passed on to other neurons in the network. Through a training process, a neural network can learn to recognize patterns and relationships in data, and can be used for classification, regression, and prediction.
- Neural network model can handle numerical data. It can predict a continuous numerical value based on a set of input features. Compared to regression model, the main advantage of neural network model is its ability to learn complex nonlinear relationships in data, thus can capture more intricate patterns and interactions. However, neural networks can be more computationally intensive to train than regression models, and can require more data to avoid overfitting.

- Collaborative filtering method
  - Similarity Metrics
    - Euclidean Distance
      - The Euclidean distance is a popular similarity metric used in machine learning field. It is a measure of the distance between two points in Euclidean space. The formula of calculating the Euclidean space is:
        Point A (a_1, a_2, a_3, …, a_n)
        Point B (b_1, b_2, b_3, …, b_n)
        d (A, B) = $\sqrt{(a1 - b1)^2 + (a2 - b2)^2 + \cdots + (an - bn)^2}$
    - Pearson Correlation Coefficient
      - The Pearson correlation coefficient is a statistical measure that indicates the strength and direction of the linear relationship between two variables. The Pearson correlation coefficient ranges from -1 to 1, with
        -1 indicating a perfect negative correlation, 0 indicating no correlation, and 1 indicating a perfect positive correlation. The formula of calculating Pearson Correlation coefficient is:

$$r = \frac{\Sigma\ （(x - \bar{x}) * (y - \bar{y}))}{\sqrt{(x - \bar{x})^2} * \sqrt{(y - \bar{y})^2}}$$

        where r is the Pearson correlation coefficient,
        x and y are the two variables being correlated,
        $\bar{x}$ and $\bar{y}$ are the mean of x and y

- Cosine Similarity
  - Cosine similarity is a measure of similarity between two non-zero vectors in a high-dimensional space. It measures the cosine of the angle between the two vectors and ranges from -1 to 1, with 1 indicating that the two vectors are pointing in the same direction (i.e. identical),
    0 indicating that the vector are orthogonal (i.e. completely dissimilar), and -1 indicating that the two vectors are pointing in opposite directions (i.e. negatively correlated). The formula of calculating cosine similarity is:

$$cosine\ similarity\ of\ (x,y) = \frac{(x.y)}{||x|| * ||y||}$$

    where (x.y) represents the dot product of the two vectors, which are
    $x_1y_1+x_2y_2+\ldots+x_n*y_n$
    ||x||, ||y|| represent their Euclidean norms (i.e. magnitudes), which are $\sqrt{x1^2 + x2^2 + \cdots + xn^2}$ and $\sqrt{y1^2 + y2^2 + \cdots + yn^2}$

- Jaccard Similarity
  - Jaccard similarity is a measure of similarity between two sets. It is defined as the size of the intersection of the sets divided by the size of the union of the sets. The formula of calculating jaccard similarity is:
    J(X,Y) = |X ∩ Y| / |X ∪ Y|

Deliverable 2: Modeling Assumptions

- Content-based filtering method
  - Prediction Regression Models
    - Simple Linear Regression Model
      - The model assumes that the relationship between dependent variable and independent variables is linear, meaning that a change in the value of one variable is associated with a proportional change in the value of the other variable; Specifically, assumptions are:
      - Linearity: The relationship between the dependent variable and the independent variables is linear.
      - Independence: The observations in the dataset are independent of each other.
      - Homoscedasticity: The variance of the errors (the difference between the predicted values and the actual values) is constant across all levels of the independent variables.
      - Normality: The errors are normally distributed.

- No multicollinearity: There is no perfect linear relationship between any of the independent variables.

- Linear Regression Model with Interactions
  - Apart from what have been described above for linear regression model, there is an interaction between the independent variables. This means that the effect of one independent variable on the dependent variable depends on the level of another independent variable.

- Stepwise Model
  - The stepwise model selects the most important attributes to include in a regression model. The basic idea is to start with a full model containing all potential attributes, then to iteratively remove or add attributes based on their significance or contribution to the model. Stepwise model still includes the assumptions which were covered in the above for the simple regression model.
  - Additionally, stepwise model can be sensitive to the sample size. Consequently, it is important to have a large enough sample size to ensure reliable estimates of the regression coefficients and to avoid overfitting the model.

- Decision Tree Model
  - Feature independence: the features used in the decision tree should be independent of each other. That is, they should not be correlated with each other, as this can lead to problems with overfitting and bias in the resulting model.
  - Data quality: the quality of data used to train the decision tree should be good, with minimal missing or erroneous data. Decision trees can be sensitive to noisy or outlier data, so it is important to preprocess the data appropriately.
  - Splitting criterion: the criterion used to select the best feature to split the data should be appropriate for the problem at hand.
  - Three depth: the depth of the decision tree should be appropriate for the complexity of the problem. If the tree is too deep, it may overfit the data and fail to generalize well to new data. If the tree is too shallow, it may underfit the data and miss important patterns or relationships.
  - Sampling: when working with large datasets, it can be useful to use a subset of the data to train each node of the decision tree. This can help to reduce overfitting and improve the accuracy of the resulting model.

- Random Forest Model

- As mentioned above, random forest is an extension of the decision tree algorithm by using multiple decision trees to improve the accuracy and stability of the model.
- Apart from the assumptions of decision tree model, it has an additional assumption of the number of trees. The number of trees should be sufficient to achieve good performance, but not so large as to be computationally expensive or to overfit the data. There is often a trade-off between model complexity and accuracy, which can be addressed through cross-validation and other methods.

- Neural Network Model
  - Large data sets: neural networks can require a large amount of data to train effectively. This is especially true for deep neural networks, which have many layers of neurons and parameters. It is important to have enough data to avoid overfitting and to achieve good generalization performance.
  - Quality data: the data should be representative of the problem being solved, with minimal missing or erroneous data.
  - Activation functions: neural networks rely on activation functions to introduce nonlinearity into the model. Since my problem involves predicting the streaming time based on audio features, I can use a neural network model with a single output neuron that has a linear activation function in the output layer.
  - Network architecture: this includes the number of layers, the number of neurons in each layer, and the type of connections between the layers. Choosing the right architecture can be a complex process that requires experimentation and tuning.
  - Initialization: the initial values of the weights and biases in the neural network can have a significant impact on the performance of the model. It is important to choose appropriate initialization values to avoid getting stuck in local optima during training.
  - Regularization: neural networks can be prone to overfitting, especially for large and complex models. Regularization techniques, such as dropout, weight decay, or early stopping, can be used to mitigate this problem and improve generalization performance.

- Collaborative filtering method

  - Similarity Metrics

    - Euclidean Distance
      - Euclidean space: the data points lie in a Euclidean space, where the distance between two points is the straight-line distance between them. This is a fundamental assumption of the metric, and if the

data points do not lie in a Euclidean space, the metric may not be appropriate.
    - Continuous variables: the variables used to compute the distance are continuous. If the variables are categorical or ordinal, or if they are measured on different scales, the metric may not be appropriate. In my case, audio features are continuous.
    - Independence: the variables used to compute the distance are independent of each other. If the variables are highly correlated or dependent, the metric may not be appropriate.
    - Linear relationships: the Euclidean distance metric assumes that the relationships between the variables are linear. If the relationships are nonlinear, the metric may not be appropriate.
    - Scale: the Euclidean distance metric is sensitive to the scale of the variables, and if the variables are measured on different scales, the metric may be biased towards the variables with larger scales. Therefore, it is often necessary to standardize the variables before using the Euclidean distance metric.

- Pearson Correlation Coefficient
    - Linearity: the Pearson correlation coefficient assumes that the relationship between the two variables is linear. If the relationship is nonlinear, the correlation coefficient may not be appropriate.
    - Normality: the Pearson correlation coefficient assumes that the two variables are normally distributed. If the variables are not normally distributed, the correlation coefficient may not be appropriate.
    - Homoscedasticity: the Pearson correlation coefficient assumes that the variances of the two variables are equal. If the variances are unequal, the correlation coefficient may be biased.
    - Independence: the Pearson correlation coefficient assumes that the two variables are independent of each other. If the variables are dependent or have a cause-and-effect relationship, the correlation coefficient may not be appropriate.
    - Outliers: the Pearson correlation coefficient is sensitive to outliers, which can have a strong influence on the correlation coefficient. Therefore, it is important to check for outliers and to consider their impact on the correlation coefficient.

- Cosine Similarity
    - Vector representation: two objects being compared are represented as vectors in a high-dimensional space. This is a fundamental assumption of the similarity measure, and if the objects cannot be represented as vectors, the similarity measure may not be appropriate.
    - Positive values: the values in the vectors are non-negative. If the values can be negative, the similarity measure may be biased or inappropriate.

- No zero-norm vectors: the vectors being compared are not zero-norm vectors, that is, they are not vectors with all entries equal to zero. If the vectors are zero-norm vectors, the cosine similarity may not be well-defined.
- Scale invariance: the cosine similarity is scale-invariant, meaning that it is insensitive to the magnitude or scale of the vectors. This can be an advantage in some applications, but it can also be a disadvantage if the magnitude or scale of the vectors is important for the problem being addressed.

Deliverable 3: Defined Modeling Techniques

There are many different modeling techniques, from which I need to determine which ones are the best for my project business goal – exploring the behind algorithm of Spotify recommendation system and optimizing the recommendation system to benefit Spotify users.

- Content-based filtering method
    - For this method, my data mining goal is to find the optimal prediction model based on the audio features of the tracks.
    - This should be based on supervised learning algorithm. All the 12 audio features (i.e. danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo) are the independent variables (i.e. x variables) and msPlayed is the dependent variable (i.e. y variable). I will build several prediction regression models and find the one with the least Root-mean-square-deviation ("RMSE"), which is a metric for evaluating the performance of regression models by measuring the difference between predicted values by a model and the actual observed values.
    - The following is a summarized table indicating my intentions to use the modeling technique based on each model's definitions, descriptions and assumptions.

Content-based filtering method	
Simple Linear Regression Model	Simple Linear Regression Model is a fundamental and widely used model. The assumptions of this model align with my intuition, as there should be a linear relationship between the streaming time of a song and its audio features. I plan using this modeling technique. It directly correlates to my data mining goal.
	Apart from the points made in the simple linear regression model, some of the audio features will

Content-based filtering method	
Linear Regression Model with Interactions	be correlated. For instance, in most of the cases, a song with higher energy will have higher liveness. I plan using this modeling technique.
Stepwise Model	I plan using this modeling technique as I think it may be beneficial to use machine learning algorithms to determine which audio features and which interactions of audio features I should include instead of me manually figuring it out. The assumptions of this model also align with my above point.
Decision Tree Model	I plan using this modeling technique to solve the numerical data. I would like to see how the decision tree model can help me predict the streams by creating a tree-like flowchart based on the audio features data. However, I will keep in mind that this model technique may not be the best option compared to regression model. For example, I think some of the assumptions may not be aligned. The audio features are not independent of each other.
Random Forest Model	As random forest model is an extension of the decision tree models, this may also not be the best choice. I still plan using this modeling technique to solve the numerical data. While it is true that traditional decision tree models may not perform as well on numerical data compared to categorical data, random forest can handle both types of data effectively.

Content-based filtering method	
Neural Network Model	I plan using this modeling technique. My intuition is that there probably will not be a linear relationship between the audio features and my preference/how much I like a song. I use the stream as the metric to measure my preference of a song as generally I will stream more if I like a song more. While it might be a subjunctive matter to tell what kinds of music I like, I agree that using machine learning algorithm of predicting streams through audio features is a good approach. Based on the Neural Network Model assumptions, it includes a number of layers where a number of neurons embedded within. Neural network model may be a good technique as it is a type of machine learning model that mimics the structure and function of a human brain. I have never had an opportunity to know more about this modeling technique in my past data science projects. I would love to have a try this time.
Collaborative filtering method	
Euclidean Distance	Based on my understanding, in my case, first, audio features of a track can are identified by a unique set of coordinates; thus they are in the Euclidean space. Second, audio features (apart from mode is binary) are continuous. Third, the target track and all tracks in Spotify are not highly correlated. Fourth, the relationships between variables should be linear. Audio features are quantified between 0 to 1 in general, but there are a few audio features (i.e. key, loudness, tempo) that are not the same scale as others, I might need to adjust. Overall, Euclidean space is an appropriate metric to use.
Pearson Correlation Coefficient	I plan using this modeling technique. Based on my understanding, in my case, first, the relationships between two variables is linear. Second, in terms of the homoscedasticity, the variances of audio features should be similar as they are quantified between 0 to 1 after I adjusted some of the audio features. Third, the target track and all tracks in Spotify are not highly correlated. Fourth, there should not be any outliers in my dataset as I've conducted the data preparation

Content-based filtering method	
	phase. Overall, Pearson correlation coefficient is an appropriate metric to use.
Cosine Similarity	I plan using this modeling technique. Based on my understanding, the first assumption works perfectly for my case. Each track has several audio features, which can be regarded as a vector in a high-dimensional space. As for the second and third assumptions, while the values of the vectors are generally non-negative, the attribute representing loudness is negative. To address this, I may consider adjusting the loudness attribute by adding a positive value to ensure that all values are non-negative. Finally, the fact that Cosine similarity is scale-invariant is a great advantage, as it means there is no need for me to normalize audio features that do not have a scale between 0 and 1. Overall, Cosine similarity coefficient is an appropriate metric to use.
Jaccard Similarity	I suppose this modeling technique does not apply for my dataset. Spotify launched a playlist initiative, Blend, where any user can invite any user to generate a playlist wherein the two users' tastes are combined into one shared playlist[6]. Spotify may use the Jaccard Similarity to find the user B song set which is the most similar to the user A as Spotify is able to retrieve all sets from subscribers. It would be great if I have the datasets of other users; however, since I do not, I won't be able to utilize Jaccard similarity technique.

Overall, apart from the Jaccard Similarity, I plan using all the above modeling techniques for my following analysis.

Task 4.2: Designing Tests
Deliverable: Test Design Document

When designing tests, it is crucial to carefully consider my training, testing, and validation strategy in order to avoid introducing bias into the data.

Training data enables the model to learn from the patterns and relationships in the data, and to make accurate predictions on new, unseen data. The testing data is used to evaluate the performance of the model after it has been trained and tuned on the training data. The testing data is typically held out from the training process, and the model is evaluated on this dataset to

[6] https://engineering.atspotify.com/2021/12/a-look-behind-blend-the-personalized-playlist-for-youand-you/

see how well it generalizes to new, unseen data. Apart from training and testing split, in order to fine-tune the model's hyperparameters during training and ensure the model is not overfitting to the training data, we usually have a validation data.

I will split 80% of my data for training and the remaining 20% for testing. Of my 80% training data, I will further split 20% of my training data into validation data and kept the 80% as training data.
So, 64% training data, 16% validation data and 20% testing data.

In summary,

First, I will train the model on the training dataset.
Then, I will fine-tune the model and its hyperparameters using the validation dataset.
Finally, I will evaluate the final model on the testing set to estimate its performance on new, unseen data.
Overall, this can help avoid introducing bias into the data as much as possible.

Task 4.3: Building Models
Deliverable 1: Parameter Definitions/Settings

Regression models use a numerical optimization algorithm to find the best coefficients that minimize the prediction error, and these algorithms can be sensitive to the scale of the input features. Therefore, it is generally a good idea to scale the variables in a regression model to the same range. Moreover, as mentioned in modeling assumptions section, the Euclidean distance metric is sensitive to the scale of variables, Pearson correlation coefficient requires that the variables are normally distributed and is sensitive to outliers, and Cosine similarity metric requires positive values of the variables, I decided to perform scaling the some of the input features for across all three datasets. Based on my research, there are several methods with regard to rescaling data. They are min-max scaling, standardization, robust scaling, log transformation, power transformation and so forth. Min-max scaling method scales the data to a fixed range (usually 0 to 1), which satisfies the assumptions of all similarity metrics.
The below is a summary of how I adjusted the parameters settings. I mostly adjusted three audio features, which are key, loudness and tempo using the min-max scaling method.

	Description of the Fields	Scaling the data
Audio Features		
danceability	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.	This attribute ranges from 0 to 1. No further scaling action needed.
energy	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.	This attribute ranges from 0 to 1. No further scaling action needed.
acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.	This attribute ranges from 0 to 1. No further scaling action needed.
valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).	This attribute ranges from 0 to 1. No further scaling action needed.
instrumentalness	Instrumentalness predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks.	This attribute ranges from 0 to 1. No further scaling action needed.

key	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C/D, 2 = D, and so on.	I used min-max scaling to scale the values to the range [0,1] by subtracting the min value and dividing by the range.
liveness	This value describes the probability that the song was recorded with a live audience. A value above 0.8 provides strong likelihood that the track is live.	This attribute ranges from 0 to 1. No further scaling action needed.
loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.	I used min-max scaling to scale the values to the range [0,1] by subtracting the min value and dividing by the range.
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.	No further scaling action needed.
speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.	This attribute ranges from 0 to 1. No further scaling action needed.
tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.	I used min-max scaling to scale the values to the range [0,1] by subtracting the min value and dividing by the range.
duration_ms	Duration_ms is the duration of a track in milliseconds.	Duration of a track is irrelevant for predicting the streams. Thus, I won't include this attribute in my model.
popularity	Spotify calculated the popularity of a track based on: 1) total streams of a track, 2) how recently a track has been played, 3) the frequency that a track has been played.	As msPlayed (i.e. streaming time) has been determined as the dependent variable, I excluded the popularity.
General Features		
artist	Artist represents the artist name of the track.	N/A
track	Track represents the track name of the song.	N/A
album	Album represents the name of the album that a track belongs to.	N/A
msPlayed	MsPlayed represents the streaming time of a track in milliseconds.	N/A
uri	The Spotify URI for the track. Uniform resource indicator is a link that you can find in the Share menu of any track, album, or Artist Profile on Spotify.	N/A
Other Features		
endTime	EndTime is the time when I end listening to the track. For example, 2020-01-16 23:31 means that I ended listening to a certain song at 23:31 on January 16, 2020.	N/A
rank	Rank represents the ranking of the tracks by a certain week.	N/A
source	Source is the music label of the track.	N/A
peak_rank	Peak_rank represents the highest rank of a track.	N/A
previous_rank	Previous_rank represents the previous rank of a track.	N/A
weeks_on_chart	Weeks_on_chart means how many weeks a track stays on chart.	N/A
year	Year indicates which year the track is released on Spotify.	N/A
released_date	Released_date indicates which date or year the track is released on Spotify.	N/A

Code

```
#Impoart Datasets
import pandas as pd
indiv=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da

gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da

genl=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Dat
```

```
#Scale some of the attributes
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Scale the key, loudness, tempo attributes using min-max scaling in individual dataset
key_scaler = MinMaxScaler()
scaled_key = key_scaler.fit_transform(indiv[['key']])

loudness_scaler = MinMaxScaler()
scaled_loudness = loudness_scaler.fit_transform(indiv[['loudness']])

tempo_scaler = MinMaxScaler()
scaled_tempo = tempo_scaler.fit_transform(indiv[['tempo']])

indiv['scaled_key']=scaled_key
indiv['scaled_loudness']=scaled_loudness
indiv['scaled_tempo']=scaled_tempo

indiv.head()

# Scale the key, loudness, tempo attributes using min-max scaling in global weekly top 100 songs dataset
key_scaler = MinMaxScaler()
scaled_key = key_scaler.fit_transform(gt100[['key']])

loudness_scaler = MinMaxScaler()
scaled_loudness = loudness_scaler.fit_transform(gt100[['loudness']])

tempo_scaler = MinMaxScaler()
scaled_tempo = tempo_scaler.fit_transform(gt100[['tempo']])

gt100['scaled_key']=scaled_key
gt100['scaled_loudness']=scaled_loudness
gt100['scaled_tempo']=scaled_tempo

gt100.head()

# Scale the key, loudness, tempo attributes using min-max scaling in generalized dataset
key_scaler = MinMaxScaler()
scaled_key = key_scaler.fit_transform(genl[['key']])

loudness_scaler = MinMaxScaler()
scaled_loudness = loudness_scaler.fit_transform(genl[['loudness']])

tempo_scaler = MinMaxScaler()
scaled_tempo = tempo_scaler.fit_transform(genl[['tempo']])

genl['scaled_key']=scaled_key
genl['scaled_loudness']=scaled_loudness
genl['scaled_tempo']=scaled_tempo

genl.head()
```

Output

```
print(indiv[['scaled_key', 'scaled_loudness', 'scaled_tempo']].describe())
print(gt100[['scaled_key', 'scaled_loudness', 'scaled_tempo']].describe())
print(genl[['scaled_key', 'scaled_loudness', 'scaled_tempo']].describe())
```

```
       scaled_key  scaled_loudness  scaled_tempo
count  946.000000       946.000000    946.000000
mean     0.494234         0.851035      0.409319
std      0.317224         0.080194      0.198654
min      0.000000         0.000000      0.000000
25%      0.181818         0.823856      0.247772
50%      0.545455         0.863354      0.411580
75%      0.727273         0.894990      0.539610
max      1.000000         1.000000      1.000000
       scaled_key  scaled_loudness  scaled_tempo
count  100.000000       100.000000    100.000000
mean     0.487273         0.588379      0.416351
std      0.344908         0.165273      0.208721
min      0.000000         0.000000      0.000000
25%      0.090909         0.506426      0.240796
50%      0.545455         0.622233      0.370492
75%      0.818182         0.683993      0.537682
max      1.000000         1.000000      1.000000
         scaled_key  scaled_loudness   scaled_tempo
count  170653.000000    170653.000000  170653.000000
mean        0.472713         0.760035       0.479911
std         0.319554         0.089233       0.126109
min         0.000000         0.000000       0.000000
25%         0.181818         0.710751       0.383648
50%         0.454545         0.773941       0.471153
75%         0.727273         0.827140       0.556604
max         1.000000         1.000000       1.000000
```

As demonstrated in the above output screenshot, the key, loudness, and tempo attributes have been rescaled from 0 to 1.

Additionally, I performed the train/validation/test data split before modeling for Individual Dataset. Note that there is no need for train/validation/test data split for Global Weekly Top 100 Songs Dataset as I will use this dataset purely as the test dataset for my optimal stream prediction model. Also, since Generalized Dataset is used to apply unsupervised learning method, I can use the entire dataset to find the similar songs without the split.

Code

```
from sklearn.model_selection import train_test_split

# Split the indiv dataset into training and test sets (80% training data, 20% test data)
indiv_train, indiv_test, msplayed_train, msplayed_test = train_test_split(indiv.drop('msPlayed', axis=1),
                                                                          indiv['msPlayed'],
                                                                          test_size=0.2,
                                                                          random_state=69)

# Split the training data further into training and validation sets (80% training data, 20% validation data)
indiv_train, indiv_val, msplayed_train, msplayed_val = train_test_split(indiv_train,
                                                                        msplayed_train,
                                                                        test_size=0.2,
                                                                        random_state=69)


# Print the shapes of the resulting datasets
print("Indiv Training data shape:", indiv_train.shape)
print("Indiv Training msPlayed shape:", msplayed_train.shape)
print("Indiv Validation data shape:", indiv_val.shape)
print("Indiv Validation msPlayed shape:", msplayed_val.shape)
print("Indiv Test data shape:", indiv_test.shape)
print("Indiv Test msPlayed shape:", msplayed_test.shape)
```

*Random seed is used to ensure that the same samples are selected each time the algorithm is run, which in turn ensures the results are reproducible. The value of random seed is random but it should stay consistent among different models.

Output

```
Indiv Training data shape: (604, 19)
Indiv Training msPlayed shape: (604,)
Indiv Validation data shape: (152, 19)
Indiv Validation msPlayed shape: (152,)
Indiv Test data shape: (190, 19)
Indiv Test msPlayed shape: (190,)
```

Based on the output screenshot, we can see that the Individual Dataset has been split into three sub datasets, which are individual training dataset, individual validation dataset, and individual test dataset. For each sub dataset, I separated the msPlayed variable as a target variable array because this is the dependent variable which I need to predict based on the audio feature variables. The individual training data shape (604, 19) means that this data frame has 604 observations (i.e. rows) and 19 attributes (i.e. columns). The individual training msPlayed shape (604,) means that there are 604 values in the msPlayed array, one of each observation in the training dataset. Same interpretations for the rest four data shapes.

At this point, the data should be ready for modeling.

Deliverable 2: Model Descriptions

To my understanding, I have documented the type of model will be used and why I choose it in the Section Selecting Modeling Techniques, in this section, I would like to elaborate the explanations of inclusion of variables and why I choose these variables to be included in my model, together with how the model should be interpreted, as well as the difficulties encountered in the modeling process, or any important items of note if applicable.

- Content-based filtering method
    - Prediction Regression Models
        - Simple Linear Regression Model
            - As this is a simple linear regression and my first model, I decided to include all 11 audio features. They are danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo. MsPlayed is the dependent variable.
            - In terms of how to interpret the model, let's use a simple linear regression model as an illustration.

              $y = b_0 + b_1 * x_1 + b_2 * x_2$

              where:
              y is the MsPlayed
              x is one of the audio features (e.g. energy, speechiness)
              b_0 is the intercept (the value of MsPlayed when energy=0 and speechiness = 0)
              b_1, b_2 are the slopes (the change in MsPlayed for a one-unit increase in energy/speechiness)

In my case, it means that I will stream around b_0 miliseconds even the tracks with 0 value of energy and speechiness. Holding the speechiness feature constant, one-unit increase in energy of a track will lead me to stream b_1 milliseconds more. Same for energy feature.

- Linear Regression Model with Interactions
  - Some audio features will have correlation with each other. Generally, songs with high danceability would have high energy, liveness, tempo. Songs with high instrumentalness tend to have fewer or no vocals while songs with high speechiness typically feature more vocals or spoken words. Therefore, instrumentalness and speechiness are often negatively correlated. Songs with major modes (i.e. mode = 1) have brighter and more upbeat sounds, which may have high valence and high energy; while songs with minor modes (i.e. mode = 0) have darker and more melancholic sounds, which may have low valence and low energy.
  - It is hard for me to figure out all interactions for all audio features. I decided to include a few interactions based on my intuition.
    - Danceability ~ Energy ~ Liveness ~ Tempo
    - Instrumentalness ~ Speechiness
    - Mode ~ Valence ~ Energy
  - Apart from interpretations in simple linear regression model, the interpretation with variables with interactions is as below.

    $y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * (x_1 * x_2)$

    where b_3 is the coefficient for the interaction term $(x_1 * x_2)$

    In my case, it means that the change in the effect of energy on streaming time for a one-unit increase in speechiness will be b_3 milliseconds.

- Stepwise Model
  - I do not need to decide which variables to be included, as the stepwise model will either start from zero variable and adds up the variables or start from adding all variables and delete each variable or will start somewhere in the middle, until the results are optimal.
  - The interpretation of a stepwise model is similar to that of a standard linear regression model apart from that the variables included in the model are selected automatically by the algorithm rather than being chosen on my own.

- Decision Tree Model
  - I do not need to decide which variables to be included, as the decision tree model can automatically select the most informative

variables to create a tree-like structure that can be used to make predictions. I did some research for how to select variables for the decision tree. Generally, I can use feature selection techniques, such as information gain or Gini impurity to help identify the most informative audio features to include in the model.

- In terms of how to interpret the model, I will start looking at the root node of the tree, which represents the first decision based on one of the input variables. From there, I will follow the path down the tree based on the values of the input variables until I reach a leaf node, which represents a predicted value of the target variable.

- Random Forest Model
  - As random forest model is an ensemble learning method that combines multiple decision tree models to make predictions. When making a prediction with a random forest model, each decision tree in the forest independently predicts the target variable based on the input variables. The final prediction is then made by aggregating the predictions from all the decision trees.
  - The interpretations of a random forest model is similar to that of a decision tree model apart from that it additionally involves understanding how the individual decision trees combined to make the final prediction. This will involve examining the importance of each input variable for making predictions, as well as visualizing the structure of the decision trees in the forest.

- Neural Network Model
  - The neural network model collects interconnected artificial neurons that can learn from data and make predictions. It consists of several layers (i.e. input layer, hidden layer and output layer) of neurons that are connected by weights. As this is my first time to try neural network model, I would like to keep it simple – I will include all audio features without interactions for this model.
  - In terms of how to interpret the model, let's use a simple neural network model as an illustration.

    $h = f(Wx + e_1)$
    $y = Uh + e_2$

    where:
    x is a vector of input audio feature (i.e. danceability)
    h is a vector of hidden units
    y is a vector of output unit (i.e. MsPlayed)
    f is an activation function applied to the weighted sum of inputs to each hidden unit

W and U are weight matrices for the connections between the input layer and the hidden layer, and the hidden layer and the output layer, respectively
e_1 and e_2 are bias vectors for the hidden layer and the output layer, respectively

In my case, first, the neural network model will try to learn how the energy of a song relates to my streaming by adjusting the weights (i.e. W) of the connections between the energy and the hidden layer, and the weights (i.e. U) of the connections between the hidden layer and the output layer, as well as the biases, to minimize the difference between the predicted MsPlayed value and the actual MsPlayed value for a given input value of energy.

- Collaborative filtering method
    - Similarity Metrics
        - Euclidean Distance
        - Pearson Correlation Coefficient
        - Cosine Similarity

I do not need to decide which variables to be included as these are all unsupervised learning methods.

The above three metrics are all similarity metrics that are used to measure similarity between two vectors. They differ in calculation methods, range of values and interpretation. In general, the interpretation is that the smaller distances between two tracks, more similarities are shared between two tracks. As for Euclidean Distance, it represents the straight-line distance between two points in a multidimensional space. A smaller Euclidean distance indicates that two tracks are closer to each other, which a larger distance indicates that they are further apart. As for Pearson correlation coefficient, a positive correlation closer to 1 indicates that two tracks tend to be more similar. As for Cosine Similarity, it measures the angle between two vectors; a positive cosine similarity closer to 1 indicates that two tracks tend to be more similar.

As far as I am concerned, I encountered some difficulties about the interpretations of the random forest model and neural network model. However, based on my research, I put down my understandings of how to interpret these two models as shown above. Additionally, there are not any important items need to be addressed for now.

Deliverable 3: Data Models

In this data models deliverable, I will present the code and the output with explanations. I will provide the further assessments and interpretations in the Assessing Models section.

- Content-based filtering method

- Prediction Regression Models
  - Simple Linear Regression Model

Code

```
#Content-based filtering methods - Supervised Learning
#Data Modeling 1
#Simple Linear Regression Model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd


# Include all audio features in the matrix X for the training set
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include all audio features in the matrix X for the validation set
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include all audio features in the matrix X for the test set
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Convert X and y to numpy arrays for use with scikit-learn
X_train = np.array(X_train)
y_train = np.array(y_train)
X_valid = np.array(X_valid)
y_valid = np.array(y_valid)
X_test = np.array(X_test)
y_test = np.array(y_test)

# Fit a linear regression model to the training set
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = regressor.predict(X_valid)

# Calculate the RMSE between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = regressor.predict(X_test)

# Calculate the RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the model coefficients and RMSE on the validation and test sets
audioFeature_names = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
coeffs = pd.DataFrame({'Audio Feature': audioFeature_names, 'Coefficient': regressor.coef_})
print(coeffs)
print('Intercept:', regressor.intercept_)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

Output

```
        Audio Feature   Coefficient
0        danceability  4.131844e+06
1              energy  1.053620e+07
2                 key  2.980138e+05
3            loudness -2.343507e+05
4                mode -5.480630e+05
5         speechiness -1.047539e+07
6        acousticness -9.046304e+05
7   instrumentalness -2.069824e+06
8            liveness  8.198829e+05
9             valence -5.897379e+06
10              tempo  9.138906e+03
Intercept: -3775278.715154171
RMSE (validation): 7392924.665124627
RMSE (test): 5053163.91013996
```

- As shown in the above two screenshots, I firstly imported some Python libraries that are needed for my modeling. Sklearn (i.e. Scikit-learn) is a library used for machine learning. It provides a range of tools for building predictive models. It supports algorithms including linear regression, decision trees and random forests. Numpy and Pandas are also Python libraries that are used for numerical operations and data analysis.

- After that, I included all audio features to fit the linear regression model into my train dataset. Then I performed a RMSE calculation of both validation dataset and test dataset. Finally, I print the coefficient of each audio feature and the RMSEs for validation and test datasets.

- Linear Regression Model with Interactions

Code

```
#Data Modeling 2
#Linear Regression Model with Interactions
#Danceability ~ Energy ~ Liveness ~ Tempo
#Instrumentalness ~ Speechiness
#Mode ~ Valence ~ Energy

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Include audio features and manually determined interaction terms in the matrix X for the training set
indiv_train['danceability*energy*liveness*tempo'] = indiv_train['danceability'] * indiv_train['energy']* indiv_train['liveness
indiv_train['instrumentalness*speechiness'] = indiv_train['instrumentalness'] * indiv_train['speechiness']
indiv_train['mode*valence*energy'] = indiv_train['mode'] * indiv_train['valence'] * indiv_train['energy']
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include audio features and manually determined interaction terms in the matrix X for the validation set
indiv_val['danceability*energy*liveness*tempo'] = indiv_val['danceability'] * indiv_val['energy']* indiv_val['liveness']* inc
indiv_val['instrumentalness*speechiness'] = indiv_val['instrumentalness'] * indiv_val['speechiness']
indiv_val['mode*valence*energy'] = indiv_val['mode'] * indiv_val['valence'] * indiv_val['energy']
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include audio features and manually determined interaction terms in the matrix X for the test set
indiv_test['danceability*energy*liveness*tempo'] = indiv_test['danceability'] * indiv_test['energy']* indiv_test['liveness']*
indiv_test['instrumentalness*speechiness'] = indiv_test['instrumentalness'] * indiv_test['speechiness']
indiv_test['mode*valence*energy'] = indiv_test['mode'] * indiv_test['valence'] * indiv_test['energy']
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Fit a linear regression model to the training set
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = regressor.predict(X_valid)

# Calculate the RMSE between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = regressor.predict(X_test)

# Calculate the RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the model coefficients and RMSE on the validation and test sets
coeffs = pd.DataFrame({'Audio Feature': X_train.columns, 'Coefficient': regressor.coef_})
print(coeffs)
print('Intercept:', regressor.intercept_)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

Output

```
                        Audio Feature   Coefficient
0                        danceability  5.542521e+06
1                              energy  1.106656e+07
2                                 key  3.052543e+05
3                            loudness -2.619426e+05
4                                mode -1.459335e+06
5                         speechiness -9.457672e+06
6                        acousticness -8.637662e+05
7                    instrumentalness  1.787383e+06
8                            liveness  7.251294e+06
9                             valence -7.099418e+06
10                              tempo  1.572947e+04
11  danceability*energy*liveness*tempo -1.201629e+05
12        instrumentalness*speechiness -6.422697e+07
13                  mode*valence*energy  3.229060e+06
Intercept: -5623034.231415641
RMSE (validation): 7475304.374043499
RMSE (test): 5138853.409688507
```

- As shown in the above two screenshots, apart from including all audio features, I added some interactions among audio features based on my intuition. Other steps are the same as the simple linear regression model.

- Stepwise Model

Code

```
#Data Modeling 3
#Stepwise Model

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Include all audio features in the matrix X for the training set
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include all audio features in the matrix X for the validation set
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include all audio features in the matrix X for the test set
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Create interaction terms for the training set
poly = PolynomialFeatures(interaction_only=True)
X_train_interact = poly.fit_transform(X_train)

# Create interaction terms for the validation set
X_valid_interact = poly.transform(X_valid)

# Create interaction terms for the test set
X_test_interact = poly.transform(X_test)

# Fit a linear regression model to the training set
regressor = LinearRegression()
regressor.fit(X_train_interact, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = regressor.predict(X_valid_interact)

# Calculate the RMSE between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = regressor.predict(X_test_interact)

# Calculate the RSME between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the model coefficients and RMSE on the validation and test sets
audioFeature_names = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
audioFeature_interact_names = poly.get_feature_names_out(audioFeature_names)
coeffs = pd.DataFrame({'Audio Feature': audioFeature_interact_names, 'Coefficient': regressor.coef_})
print(coeffs)
print('Intercept:', regressor.intercept_)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

Output

```
              Audio Feature   Coefficient
0                         1 -5.741632e-04
1              danceability -2.750454e+07
2                    energy  5.864829e+07
3                       key -2.439979e+06
4                  loudness -2.404026e+06
..                       ...           ...
62  instrumentalness valence  1.929080e+07
63    instrumentalness tempo -5.764145e+04
64         liveness valence -1.165593e+07
65           liveness tempo  3.149011e+04
66            valence tempo -1.133302e+05

[67 rows x 2 columns]
Intercept: -23050013.987113398
RMSE (validation): 7822212.655658523
RMSE (test): 5557021.414021865
```

- As shown in the above two screenshots, I firstly imported a Polynomial Features library. As the default stepwise model only include audio features without interactions, I used this library so that stepwise model will have the capability to include higher-

order interactions without myself having to manually add them. After that, I included all audio features in the matrix as a pool for stepwise model to select. Other steps are the same as the simple linear regression model.

- Decision Tree Model

Code

```
#Data Modeling 4
#Decision Tree Model
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Include audio features in the matrix X for the training set
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include audio features in the matrix X for the validation set
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include audio features in the matrix X for the test set
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Fit a decision tree model to the training set
tree = DecisionTreeRegressor(max_depth=15, random_state=0)
tree.fit(X_train, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = tree.predict(X_valid)

# Calculate RMSE between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = tree.predict(X_test)

# Calculate RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the feature importances and RMSE on the validation and validation and test sets
importances = tree.feature_importances_
feature_names = X_train.columns
importances_df = pd.DataFrame({'Audio Feature': feature_names, 'Importance': importances})
print(importances_df)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

*Random state is used to ensure that the random number generator produces the same sequence of numbers each time the code is run. The value of random seed is random but it should stay consistent among different models.

Output

```
       Audio Feature  Importance
0       danceability    0.044955
1             energy    0.090306
2                key    0.038821
3           loudness    0.060877
4               mode    0.026226
5        speechiness    0.088814
6       acousticness    0.323193
7   instrumentalness    0.018705
8           liveness    0.018914
9            valence    0.193849
10             tempo    0.095341
RMSE (validation): 13798540.815978158
RMSE (test): 15059149.18537464
```

- As shown in the above two screenshots, I firstly included all audio features to fit the tree model. I set the max depth to be 15 as there are 11 audio features of each track and I leave some space for the model to add some interaction terms. In tree model, we do not have the coefficient parameters like linear regression models. Instead, decision trees use a set of rules to recursively split the data into

smaller and more homogeneous groups based on the features in the data. Therefore, I used the feature importance to indicate the significance of audio features for streaming determined by the tree model. Other steps are the same as the simple linear regression model.

- Random Forest Model

Code

```
#Data Modeling 5
#Random Forest Model

from sklearn.ensemble import RandomForestRegressor

# Include audio features in the matrix X for the training set
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include audio features in the matrix X for the validation set
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include audio features in the matrix X for the test set
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Fit a random forest model to the training set
forest = RandomForestRegressor(n_estimators=100, max_depth=15, random_state=0) #n_estimators is a parameter that represents t
forest.fit(X_train, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = tree.predict(X_valid)

# Calculate the mean squared error between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = forest.predict(X_test)

# Calculate the RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the feature importances and RMSE on the validation and test sets
importances = forest.feature_importances_
feature_names = X_train.columns
importances_df = pd.DataFrame({'Audio Feature': feature_names, 'Importance': importances})
print(importances_df)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

*n estimators is a parameter that represents the number of decision trees

Output

```
       Audio Feature  Importance
0       danceability    0.060186
1             energy    0.123244
2                key    0.025050
3           loudness    0.047716
4               mode    0.006843
5        speechiness    0.085702
6       acousticness    0.298100
7   instrumentalness    0.025404
8           liveness    0.084965
9            valence    0.140324
10             tempo    0.102464
RMSE (validation): 13798540.815978158
RMSE (test): 9811745.734805068
```

- As shown in the above two screenshots, apart from the same steps as decision tree model, I built 100 decision trees with the same depth and combined them to improve the accuracy and reduce overfitting for this random forest model. The feature importance calculation is based on the average of all 100 decision trees in the forest.

- Neural Network Model

Code

```
!python -m ensurepip --default-pip
```

```
!pip install kera
!pip install tensorflow
```

```
#Data Modeling 6
#Neural Network Model

from keras.models import Sequential
from keras.layers import Dense

# Include audio features in the matrix X for the training set
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include audio features in the matrix X for the validation set
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include audio features in the matrix X for the test set
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Define the neural network model
model = Sequential()
model.add(Dense(12, input_dim=11, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile the model
model.compile(loss='mse', optimizer='adam')

# Train the model on the training set
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_valid, y_valid))

# Use the model to predict the target variable for the test set
y_test_pred = model.predict(X_test)

# Calculate the RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the RMSE on the validation and test sets
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

Output

```
Epoch 93/100
19/19 [==============================] - 0s 4ms/step - loss: 134307905536000.0000 - val_loss: 64748993380352.0000
Epoch 94/100
19/19 [==============================] - 0s 4ms/step - loss: 134302150950912.0000 - val_loss: 64744794882048.0000
Epoch 95/100
19/19 [==============================] - 0s 3ms/step - loss: 134296203427840.0000 - val_loss: 64740613160960.0000
Epoch 96/100
19/19 [==============================] - 0s 3ms/step - loss: 134290088132608.0000 - val_loss: 64736536297472.0000
Epoch 97/100
19/19 [==============================] - 0s 3ms/step - loss: 134284258050048.0000 - val_loss: 64732081946624.0000
Epoch 98/100
19/19 [==============================] - 0s 3ms/step - loss: 134278000148480.0000 - val_loss: 64727686316032.0000
Epoch 99/100
19/19 [==============================] - 0s 3ms/step - loss: 134271700303872.0000 - val_loss: 64723265519616.0000
Epoch 100/100
19/19 [==============================] - 0s 4ms/step - loss: 134265425625088.0000 - val_loss: 64718685339648.0000
6/6 [==============================] - 0s 2ms/step
RMSE (test): 5969918.578732859
```

*Due to the length of the complete output, only results from Epoch 93 to 100 are shown in this screenshot.

- As shown in the above two screenshots, I firstly installed the Kera and Tensorflow packages using pip for neural network models. I included the single 11 audio features to fit the neural network model. After that, I defined the neural network model. The model consists of three fully connected layers (i.e. dense layers) stacked on top of each other, with the first layer having 12 neurons, the second layer having 8 neurons, and the output layer having single neuron. The input layer has 11 audio features that are passed as

inputs to the first layer. The activation function used for the first and second layer is relu (i.e. rectified linear unit) which is commonly used in neural networks to introduce nonlinearity. The output layer uses a linear activation function, which means that the output of the model is a continuous value, in this case representing msPlayed.

- The output shown is the training process of the neural network model. The loss is a measure of how well the model is performing at minimizing the difference between its predictions and the actual values. The output shows the loss for each epoch from 1 to 100. The loss should decrease over time as the model learns to make better predictions. Finally, it shows the RMSE of the test dataset.

- Collaborative filtering method
  - Similarity Metrics
    - Euclidean Distance

Code

```
#Collaborative filtering method - Unsupervised Learning
#Similarity Metric 1
#Euclidean Distance

# Input the name of the track I like
track = 'Soft'

# Filter the DataFrame based on the track name
filtered_indiv = indiv[indiv['track'] == 'Soft']

# Display the audio features for the specified track
print(filtered_indiv[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
```

```
# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_row = indiv.loc[indiv['track'] == track]
track_features = track_row.iloc[:, 4:15].values[0]
track_artist = track_row.iloc[0, 2]

# Load the generalized dataset and extract the relevant audio features
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:16].values
genl_artists = genl.iloc[:, 1].values

# Calculate the Euclidean distances between the track features and all other tracks
distances = np.sqrt(np.sum(np.square(genl_features - track_features), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Output

```
     danceability  energy  key  loudness  mode  speechiness  acousticness  \
468         0.534   0.325   11    -7.612     0       0.0362         0.757

     instrumentalness  liveness  valence   tempo
468               0.0    0.0699    0.394  148.12

The 5 most similar tracks to "Soft" are:
```

```
Hong Kong by ['Gorillaz']:
danceability: 0.513
energy: 0.582
key: 11
loudness: -7.394
mode: 0
speechiness: 0.0285
acousticness: 0.712
instrumentalness: 0.0369
liveness: 0.0771
valence: 0.241
tempo: 148.128

Mongoloid - 2009 Remaster by ['DEVO']:
danceability: 0.573
energy: 0.616
key: 11
loudness: -7.806
mode: 0
speechiness: 0.0445
acousticness: 0.0743
instrumentalness: 0.0437
liveness: 0.0996
valence: 0.588
tempo: 148.148

Bad Habits - uncut by ['Maxwell']:
danceability: 0.661
energy: 0.67
key: 11
loudness: -7.645
mode: 0
speechiness: 0.258
acousticness: 0.0873
instrumentalness: 2.01e-06
liveness: 0.0794
valence: 0.661
tempo: 148.02

Stepping Into Tomorrow by ['Donald Byrd']:
danceability: 0.675
energy: 0.919
key: 11
loudness: -7.772
mode: 0
speechiness: 0.0406
acousticness: 0.247
instrumentalness: 0.0658
liveness: 0.299
valence: 0.872
tempo: 148.241

This Town (feat. Sasha Sloan) by ['Kygo', 'Sasha Sloan']:
danceability: 0.736
energy: 0.449
key: 10
loudness: -7.956
mode: 0
speechiness: 0.0633
acousticness: 0.506
instrumentalness: 6.63e-05
liveness: 0.117
valence: 0.487
tempo: 147.971
```

- As shown in the above screenshots, I picked one of my favorite track *Soft* from my individual dataset and pulled its audio features. After that, I loaded the generalized dataset and extract the audio features columns. Then, I calculated the Euclidean distances between *Soft* are all other tracks. I found the 5 tracks with the smallest distances of *Soft* and pulled their audio features. They are *Hong Kong*, *Mongoloid – 2009 Remaster*, *Bad Habits – uncut*, *Stepping Into Tomorrow*, and *This Town (feat. Sasha Sloan)*.

- Pearson Correlation Coefficient

Code

```
# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_features = indiv.loc[indiv['track'] == track].iloc[:, 4:15].values[0]

# Load the generalized dataset and extract the relevant audio features
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:16].values

# Calculate the Pearson correlation coefficients between the track features and all other tracks
correlations = np.corrcoef(genl_features, track_features)

# Get the column index of the track features
track_index = indiv.columns.get_loc('track') + 1

# Find the indices of 5 tracks with the highest correlation (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(correlations[:, track_index])[::-1][:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, [1, 2]]

# Rename the columns to include artist and track names
most_similar_track_names.columns = ['artist', 'track']

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Create a comparison table of the most similar tracks and the original track
table = pd.concat([most_similar_track_names.reset_index(drop=True), most_similar_features.reset_index(drop=True).T, track_fea
table.columns = ['artist', 'track'] + most_similar_track_names.tolist() + [track]

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" are:')
print(table.to_string(index=False))
```

Output

```
MemoryError: Unable to allocate 217. GiB for an array with shape (170654, 170654) and data type float64
```

- As shown in the above screenshots, I was unable to use Pearson correlation coefficient metric to calculate the distance between *Soft* and all other tracks. There is not enough memory available to perform such a request, which is to create a distance matrix of size (170654, 170654) as there are 170,654 rows in generalized dataset. The matrix requires a very large amount of memory to store, and

may not be feasible to calculate on a single machine with limited memory. We may be able to use Hadoop Framework to solve this problem. I do not have this framework installed in my computer. Also, during my Big Data course last semester, we were unable to complete Hadoop assignment due to hardware issues. As a result, we ended up submitting documentation instead of competing practical exercises with Hadoop. Considering the practical limitations, I decided to pause at this point as continuing further would be too time-consuming.

- Cosine Similarity

Code

```
#Similarity Metric 3
#Cosine Similarity

from sklearn.metrics.pairwise import cosine_similarity

# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_features = indiv.loc[indiv['track'] == track].iloc[:, 4:15].values[0]

# Load the generalized dataset and extract the relevant audio features and artist names
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:16].values
genl_artists = genl.iloc[:, 1].values

# Calculate the cosine similarities between the track features and all other tracks
similarities = cosine_similarity(track_features.reshape(1, -1), genl_features)

# Find the indices of 5 tracks with the highest cosine similarity
most_similar_indices = np.argsort(similarities[0])[-6:-1]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl_artists[most_similar_indices]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Output

```
The 5 most similar tracks to "Soft" are:
Citgo by ['Chief Keef']:
danceability: 0.675
energy: 0.486
key: 10
loudness: -7.397
mode: 0
speechiness: 0.0482
acousticness: 0.595
instrumentalness: 0.0
liveness: 0.1
valence: 0.414
tempo: 140.073

Hello, Dolly! - From the Musical Production, "Hello Dolly" by ['Paul Anka']:
danceability: 0.372
energy: 0.547
key: 11
loudness: -7.646
mode: 0
speechiness: 0.0925
acousticness: 0.72
instrumentalness: 0.0
liveness: 0.502
valence: 0.552
tempo: 149.514
```

```
Love, Life And Money by ['Little Willie John']:
danceability: 0.446
energy: 0.499
key: 10
loudness: -7.299
mode: 0
speechiness: 0.0333
acousticness: 0.886
instrumentalness: 1.5e-05
liveness: 0.205
valence: 0.567
tempo: 137.922
```

```
Tadow by ['Masego', 'FKJ']:
danceability: 0.704
energy: 0.487
key: 9
loudness: -6.407
mode: 0
speechiness: 0.0604
acousticness: 0.669
instrumentalness: 8.64e-05
liveness: 0.111
valence: 0.313
tempo: 121.726
```

```
Good Vibrations - Remastered by ['The Beach Boys']:
danceability: 0.403
energy: 0.495
key: 10
loudness: -6.888
mode: 0
speechiness: 0.0312
acousticness: 0.387
instrumentalness: 8.03e-06
liveness: 0.147
valence: 0.38
tempo: 132.912
```

- As shown in the above screenshots, I imported the Cosine similarity package. Similarly as what I did for Euclidean Distance metric, I picked one of my favorite track *Soft* and loaded the generalized dataset. Then, I calculated the Cosine similarity between *Soft* are all other tracks. I found the 5 tracks with the smallest distances of *Soft* and pulled their audio features. Note that the np.argsort function returns an array of indices that would sort the Cosine similarity values in ascending order. [-6:-1] means the 5 highest Cosine similarity values from the sorted array were selected. So, the 5 most similar tracks are *Citgo*, *Hello, Dolly!*, *Love, Life and Money*, *Tadow*, *Good Vibrations – Remastered*. I was pretty excited to see that *Tadow* was selected because I love this song and it is already in my library actually.

Task 4.4: Assessing Models
Deliverable 1: Model Assessment

So far, I have finished building all the models. In this section, I would like to give a summary of the information developed in the models and provide technical assessments of the models.

- Content-based filtering method

Summary of information developed in the models

	SLR	LR with Interactions	Stepwise	Decision Tree	Random Forest	Neural Network
RMSE (test)	**5,053,164**	**5,138,853**	**5,557,021**	**15,059,149**	**9,811,746**	**5,890,962**
RMSE (validation)	7,392,925	7,475,304	7,822,213	13,798,541	13,798,541	7,392,925
Intercept	-3,775,279	-5,623,034	-23,050,014	N/A	N/A	N/A
Coefficients/Importance						
Danceability	4,131,844	5,542,521	-27,504,540	0.045	0.060	N/A
Energy	105,326,200	11,066,560	58,648,290	0.090	0.123	N/A
Key	298,014	305,254	-2,439,979	0.039	0.025	N/A
Loudness	-2,343,501	-261,943	-2,404,026	0.061	0.048	N/A
Mode	-548,063	-1,459,335	N/A	0.026	0.007	N/A
Speechiness	-10,475,390	-9,457,672	N/A	0.089	0.086	N/A

Acousticness	-904,630	-863,766	N/A	0.323	0.298	N/A
Instrumentalness	-2,069,824	1,787,383	N/A	0.019	0.025	N/A
Liveness	819,883	7,251,294	N/A	0.019	0.085	N/A
Valence	-5,897,379	-7,099,418	N/A	0.194	0.140	N/A
Tempo	9,139	15,729	N/A	0.095	0.102	N/A
Danceability* Energy* Liveness* Tempo	N/A	-120,163	N/A	N/A	N/A	N/A
Instrumentalness* Speechiness	N/A	-64,226,970	N/A	N/A	N/A	N/A
Mode* Valence* Energy	N/A	3,229,060	N/A	N/A	N/A	N/A
Instrumental* Valence	N/A	N/A	19,290,800	N/A	N/A	N/A
Instrumentalness* Tempo	N/A	N/A	-57,641	N/A	N/A	N/A
Liveness* Valence	N/A	N/A	-11,655,930	N/A	N/A	N/A
Liveness* Tempo	N/A	N/A	31,490	N/A	N/A	N/A
Valence* Tempo	N/A	N/A	113330	N/A	N/A	N/A

Technical assessments of the models

- Based on the RMSE on the test dataset, the simple linear regression model has the relatively lowest RMSE among all six prediction regression models, followed by linear regression model with interactions, stepwise model, neural network model. The decision tree model and random forest model do not perform well compared to others in my case.
    - In my opinion, due to the small sample size of my individual dataset which contains only 946 rows with 11 audio feature attributes, a simple regression model is adequate for developing a prediction model. This is also why a simple regression model is considered as the most commonly used and popular method. The decision tree and random forest models may be overfitting to the training data, which means that they capture the noise in the data and do not generalize well to the test data. Moreover, it appears that adding interactions between the audio features did not improve the performance of the linear regression and stepwise models. Therefore, it suggests that the relationships between the audio features are not complex and can be adequately captured by a simple linear regression model.
    - Additionally, note that the table above illustrates that the RMSE values on the validation dataset are generally in agreement with those on the test dataset, except for some differences observed in the neural network model. When the RMSE is lower on the test dataset, it tends to be lower on the validation dataset as well.
- Based on the simple linear regression, we can see that energy plays the most significant positive role in my streaming of a track, followed by liveness, and danceability. However, speechiness plays a significantly negative role in my streaming of a track, followed by valence, loudness, and instrumentalness. Key and tempo play positive roles in my streaming time, but the impacts are pretty slight compared to other audio features.

Mode plays negative role in my streaming time, but the impact is pretty slight compared to other audio features.

- Electronic dance music ("EDM") consists of the majority of songs in my library. They tend to have high energy, liveness and danceability. Most EDMs do not have much vocals and I believe I do not listen to Raps much so it makes sense that speechiness plays a negative role. I do not listen to instrumental music much so it makes sense that instrumentalness plays a negative role.

- I am surprised to find that valence and loudness play a negative role in my streaming. I like cheerful songs which should have high valence and sound more positive. If I like EDM, it should be the case that loudness play a positive role in my streaming. It might be some noise or randomness in the data that is causing the negative relationship between valence or loudness and my streaming preferences.

- As the linear regression model has a slightly higher RMSE compared to the simple linear regression model. It is still worthwhile for me to interpret some interaction terms.

    - As shown in the table, the interaction between instrumentalness and speechiness is negative, which aligns my initial intuition. Usually, a track with a higher instrumentalness has less vocals/spoken words/raps. The interactions among mode, valence and energy are positive, which also align my initial intuition. Most of the case, a track with high energy should have more positiveness, meaning a higher valence; and have a brighter and more upbeat sounds, meaning having a major key. However, the interactions among danceability, energy, liveness and tempo are negative. I could not be able to explain this, unfortunately.

- Collaborative filtering method

Summary of information developed in the models

	Euclidean Distance	Cosine Similarity
The 5 most similar tracks to Soft		
1	Hong Kong	Citgo
2	Mongoloid – 2009 Remaster	Hello, Dolly!
3	Bad Habits – uncut	Love, Life and Money
4	Stepping Into Tomorrow	Tadow
5	This Town (feat. Sasha Sloan)	Good Vibrations – Remastered

Technical assessments of the models

Unsupervised learning models cannot be evaluated using a single metric, unlike supervised learning models that use RMSE as a measure of performance. This is because the goal of unsupervised learning is to uncover patterns, structures, or relationships within the data.

In my case, I am using the similarity metrics to uncover the relationships within the data. From my perspective, the approach to measure the performance should be just manually inspect the results. In other words, I will listen to those 10 tracks and determine whether I like them or not.

Deliverable 2: Revised Parameter Settings

So far, I haven't performed the step of manually inspecting the results. This manual inspection will apply to both content-based filtering method and collaborative filtering method. For content-based filtering method, I will fit the optimal prediction model to the global weekly top 100 songs dataset and add one new column pred_msPlayed. I will choose the top 20 songs with the largest predicted streaming time and listen to them and determine how many songs I want to add to my library. For collaborative filtering method, as discussed above, I will listen to those 10 songs which are attained based on two similarity metrics and determine how many songs I want to add to my library.

Based on my review of the CRISP-DM instructions, we will have the Phase 5.0 – Evaluation where it looks more broadly at which model best meets the business needs and this phase will include evaluating results, reviewing the process and determining the next steps. In contrast to the technical focus of the data modeling phase, I would like to reserve the manual inspection for the subsequent phase.

- Content-based filtering method
    - Prediction Regression Models
        - Simple Linear Regression Model
            - As for revising parameter settings, my understanding is that there should not be many adjustments in terms of the parameter settings for the simple linear regression model. Most of the time, we just leave the default parameter settings as it is for a simple linear regression model. Nevertheless, I tried two models with revise on parameter settings by added a few interaction terms on the top of the simple linear regression model to see if there is any space for lowering the RMSE.
            - For the first model, I removed the interaction terms of danceability*energy*liveness*tempo for our original linear regression model with interactions as I was unable to interpret the negative coefficient. I just kept the interactions between instrumentalness and speechiness and the interactions among mode, valence and energy.

Code

```
#Revised Parameter Settings
#Revised Regression Model 1

#Instrumentalness ~ Speechiness
#Mode ~ Valence ~ Energy

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Include audio features and manually determined interaction terms in the matrix X for the training set
indiv_train['instrumentalness*speechiness'] = indiv_train['instrumentalness'] * indiv_train['speechiness']
indiv_train['mode*valence*energy'] = indiv_train['mode'] * indiv_train['valence'] * indiv_train['energy']
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include audio features and manually determined interaction terms in the matrix X for the validation set
indiv_val['instrumentalness*speechiness'] = indiv_val['instrumentalness'] * indiv_val['speechiness']
indiv_val['mode*valence*energy'] = indiv_val['mode'] * indiv_val['valence'] * indiv_val['energy']
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include audio features and manually determined interaction terms in the matrix X for the test set
indiv_test['instrumentalness*speechiness'] = indiv_test['instrumentalness'] * indiv_test['speechiness']
indiv_test['mode*valence*energy'] = indiv_test['mode'] * indiv_test['valence'] * indiv_test['energy']
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Fit a linear regression model to the training set
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = regressor.predict(X_valid)

# Calculate the RMSE between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = regressor.predict(X_test)

# Calculate the RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the model coefficients and RMSE on the validation and test sets
coeffs = pd.DataFrame({'Audio Feature': X_train.columns, 'Coefficient': regressor.coef_})
print(coeffs)
print('Intercept:', regressor.intercept_)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

Output

```
                    Audio Feature   Coefficient
0                    danceability  4.419394e+06
1                          energy  9.512113e+06
2                             key  3.014784e+05
3                        loudness -2.532981e+05
4                            mode -1.470128e+06
5                     speechiness -9.648437e+06
6                    acousticness -9.186839e+05
7                instrumentalness  1.833977e+06
8                        liveness  1.081713e+06
9                         valence -7.086316e+06
10                          tempo  9.581889e+03
11  instrumentalness*speechiness -6.808614e+07
12            mode*valence*energy  3.305357e+06
Intercept: -3051501.7424576553
RMSE (validation): 7477293.322066186
RMSE (test): 5144825.893391904
```

- For the second model, I included the interactions between danceability and energy, and interactions between instrumentalness and speechiness.

Code

```
#Revised Parameter Settings
#Revised Regression Model 2

#Danceability ~ Energy
#Instrumentalness ~ Speechiness

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Include audio features and manually determined interaction terms in the matrix X for the training set
indiv_train['danceability*energy'] = indiv_train['danceability'] * indiv_train['energy']
indiv_train['instrumentalness*speechiness'] = indiv_train['instrumentalness'] * indiv_train['speechiness']
X_train = indiv_train[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness'
y_train = msPlayed_train

# Include audio features and manually determined interaction terms in the matrix X for the validation set
indiv_val['danceability*energy'] = indiv_val['danceability'] * indiv_val['energy']
indiv_val['instrumentalness*speechiness'] = indiv_val['mode'] * indiv_val['valence'] * indiv_val['energy']
X_valid = indiv_val[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_valid = msPlayed_val

# Include audio features and manually determined interaction terms in the matrix X for the test set
indiv_test['danceability*energy'] = indiv_test['danceability'] * indiv_test['energy']
indiv_test['instrumentalness*speechiness'] = indiv_test['mode'] * indiv_test['valence'] * indiv_test['energy']
X_test = indiv_test[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
y_test = msPlayed_test

# Fit a linear regression model to the training set
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Use the model to predict the target variable for the validation set
y_valid_pred = regressor.predict(X_valid)

# Calculate the RMSE between the predicted and actual values on the validation set
mse_valid = mean_squared_error(y_valid, y_valid_pred)
rmse_valid = np.sqrt(mse_valid)

# Use the model to predict the target variable for the test set
y_test_pred = regressor.predict(X_test)

# Calculate the RMSE between the predicted and actual values on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)

# Print the model coefficients and RMSE on the validation and test sets
coeffs = pd.DataFrame({'Audio Feature': X_train.columns, 'Coefficient': regressor.coef_})
print(coeffs)
print('Intercept:', regressor.intercept_)
print('RMSE (validation):', rmse_valid)
print('RMSE (test):', rmse_test)
```

Output

```
                  Audio Feature   Coefficient
0                  danceability  1.019582e+07
1                        energy  1.810585e+07
2                           key  3.020612e+05
3                      loudness -2.809619e+05
4                          mode -4.911389e+05
5                   speechiness -1.007410e+07
6                  acousticness -8.785976e+05
7              instrumentalness  1.639679e+06
8                      liveness  7.512530e+05
9                       valence -5.893945e+06
10                        tempo  9.548918e+03
11          danceability*energy -1.099299e+07
12  instrumentalness*speechiness -6.479854e+07
Intercept: -8364076.85887859
RMSE (validation): 17060056.796461314
RMSE (test): 15518902.639484894
```

- Unfortunately, as presented in the above two output screenshots, neither of the revised model have a lower RMSE compared to the simple linear regression model. I will stick to the simple linear regression model.

- Collaborative filtering method
    - Similarity Metrics
        - Euclidean Distance

- In terms of the similarity metrics, for Euclidean Distance, the parameter that can be adjusted is the "p" value. By default, p=2, which means that the standard Euclidean Distance is used. However, we can set p=1 to use Manhattan Distance. I revised my code and produced the following output.

Code

```
#Revised Parameter Settings
#Revised Regression Model 3

# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_row = indiv.loc[indiv['track'] == track]
track_features = track_row.iloc[:, 4:15].values[0]
track_artist = track_row.iloc[0, 2]

# Load the generalized dataset and extract the relevant audio features
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:16].values
genl_artists = genl.iloc[:, 1].values

# Calculate the Manhattan distances between the track features and all other tracks
distances = np.sum(np.abs(genl_features - track_features), axis=1)

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Output

```
The 5 most similar tracks to "Soft" are:
Hong Kong by ['Gorillaz']:
danceability: 0.513
energy: 0.582
key: 11
loudness: -7.394
mode: 0
speechiness: 0.0285
acousticness: 0.712
instrumentalness: 0.0369
liveness: 0.0771
valence: 0.241
tempo: 148.128

Mongoloid - 2009 Remaster by ['DEVO']:
danceability: 0.573
energy: 0.616
key: 11
loudness: -7.806
mode: 0
speechiness: 0.0445
acousticness: 0.0743
instrumentalness: 0.0437
liveness: 0.0996
valence: 0.588
tempo: 148.148

Bad Habits - uncut by ['Maxwell']:
danceability: 0.661
energy: 0.67
key: 11
loudness: -7.645
mode: 0
speechiness: 0.258
acousticness: 0.0873
instrumentalness: 2.01e-06
liveness: 0.0794
valence: 0.661
tempo: 148.02

When I Say I Do by ['Matthew West']:
danceability: 0.478
energy: 0.433
key: 11
loudness: -7.16
mode: 1
speechiness: 0.0328
acousticness: 0.77
instrumentalness: 0.0
liveness: 0.219
valence: 0.31
tempo: 147.919

This Town (feat. Sasha Sloan) by ['Kygo', 'Sasha Sloan']:
danceability: 0.736
energy: 0.449
key: 10
loudness: -7.956
mode: 0
speechiness: 0.0633
acousticness: 0.506
instrumentalness: 6.63e-05
liveness: 0.117
valence: 0.487
tempo: 147.971
```

- *Stepping Into Tomorrow* has been replaced by *When I say I do* when I used Manhattan distance. Apart from that, all other four songs remain the same. However, I am hesitant to revise the parameter settings in this way, as it seems to involve changing the metric itself rather than adjusting the parameters of the current metric.

- Cosine Similarity
  - For Cosine Similarity, there are no specific parameters to adjust. However, the similarity measure can be affected by preprocessing

steps such as scaling or normalization of the features. I believe I have already scaled the audio features so there should be no more adjustment in my case.

As this assignment involves a significant amount of data modeling, I included the code script for your reference in submission.

5.0 Modeling Evaluation

Task 5.1: Evaluating Results
Deliverable 1: Result Assessment

As I've mentioned in the 4.0 Data Modeling assignment, I would like to reserve the step of manual inspection in the 5.0 Modeling Evaluation assignment.

- Content-based filtering method
    - For content-based filtering method, I will fit the optimal prediction model to the global weekly top 100 songs dataset and add one new column pred_msPlayed. I will choose the top 20 songs with the largest predicted streaming time and listen to them and determine how many songs I want to add to my library.
    - The optimal prediction model is the simple linear regression model.

$$y = -3{,}775{,}279 + 4{,}131{,}844 * danceability + 105{,}326{,}200 * energy + 298{,}014 * key + -2{,}343{,}501 * loudness - 548{,}063 * mode - 10{,}475{,}390 * speechiness - 904{,}630 * acousticness - 2{,}069{,}824 * instrumentalness + 819{,}883 * liveness + -5{,}897{,}379 * valence + 9{,}139 * tempo + e$$

I used the above formula and calculated the pred_msPlayed for each track. Then, I sorted in descending order and showed the top 20 songs. Those songs can be regarded as my potential liked songs.

Code

```
#Impoart Datasets
import pandas as pd
gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da
```

```
# define the variables used in the formula
a = 4131844
b = 105326200
c = 298014
d = -2343501
e = -548063
f = -10475390
g = -904630
h = -2069824
i = 819883
j = -5897379
k = 9139
l = -3775279

# calculate the predicted msPlayed using the formula and variables
gt100 ['pred_msPlayed'] = l + (a * gt100['danceability']) + (b * gt100['energy']) + (c * gt100['key']) + (d * gt100['loudness

# print the updated DataFrame to verify the new column and values
print(gt100)
```

```
# sort the DataFrame by pred_msPlayed in descending order
gt100_sorted = gt100.sort_values ('pred_msPlayed', ascending = False)

# show the top 20 songs by pred_msPlayed
top20_songs = gt100_sorted.head(20)

# print the top 20 songs
print(top20_songs[['track', 'artist']])
```

Output

```
                                track                                artist
                    I'm Good (Blue)                 David Guetta, Bebe Rexha
                      Murder In My Mind                              Kordhell
                     Marisola - Remix  Cris Mj, Duki, Nicki Nicole, Standly, Stars Mu...
                                  505                        Arctic Monkeys
                             Hype Boy                               NewJeans
                                Leão                       Marília Mendonça
                        Save Your Tears                           The Weeknd
                             Calm Down                                 Rema
                        SPIT IN MY FACE!                              ThxSoMch
          Calm Down (with Selena Gomez)                  Rema, Selena Gomez
                                10:35                        Tiësto, Tate McRae
                             Miss You                 Oliver Tree, Robin Schulz
                  Cold Heart - PNAU Remix               Elton John, Dua Lipa, PNAU
                                LOKERA          Rauw Alejandro, Lyanno, Brray
                       Shinunoga E-Wa                           Fujii Kaze
                       I Ain't Worried                           OneRepublic
Quevedo: Bzrp Music Sessions, Vol. 52                    Bizarrap, Quevedo
                       Sweater Weather                   The Neighbourhood
                  STAY (with Justin Bieber)          The Kid LAROI, Justin Bieber
                      Escapism. - Sped Up                    RAYE, 070 Shake
                           ANTIFRAGILE                          LE SSERAFIM
```

*I added an additional song because *Calm Down* appeared twice.

- Based on the simple linear regression model, the 20 songs shown above are identified to be my potential liked songs. To determine whether each song is a good fit for my music library, I listened to each one, rated it on a scale from 1 to 10 (where 1 means I do not like it at all, 5 means I am okay with it, 10 means I really like it), and indicated whether I would like to add it to my library.

	Track	Artist	Scale	Added to my library
1	I'm Good (Blue)	David Guetta, Bebe Rexha	6	No
2	Murder In My Mind	Kordhell	9	Yes
3	Marisola – Remix	Cris Mj, Duki, Nicki Nicole, etc.	4	No
4	505	Arctic Monkeys	3	No
5	Hype Boy	NewJeans	3	No
6	Leao	Marilia Mendonca	1	No
7	Save Your Tears	The Weekend	5	No
8	Calm Down	Rema	3	No
9	SPIT IN MY FACE!	ThxSoMch	3	No
10	Calm Down (with Selena Gomez)	Rema, Selena Gomez	N/A	N/A (duplicate as #8)
11	10:35	Tiesto, Tate McRae	8	Already in my library
12	Miss You	Oliver Tree, Robin Schulz	7	Yes
13	Cold Heart – PNAU Remix	Elton John, Dua Lipa, RNAU	10	Yes!
14	LOKERA	Rauw Alejandro, Lyanno, Brray	6	No
15	Shinunoga E-Wa	Fujii Kaze	4	No
16	I Ain't Worried	OneRepublic	4	No
17	Quevedo: Bzrp Music Sessions, Vol. 52	Bizarrap, Quevedo	6	No
18	Sweater Weather	The Neighbourhood	2	No
19	Stay	The Kid LAROI, Justin Bieber	9	Already in my library
20	Escapism. – Sped Up	RAYE, 070 Shake	7	Yes
21	ANTIFRAGILE	LE SSERAFIM	5	No

I determined a total of six out of 20 songs are my liked songs. Among these six songs, two are already in my library, four more songs are newly added in my library.

I have a high standard when adding songs to my library. As shown in the scale column, I set a threshold of 7 for determining whether a song is worthy of being

added to my library. Only songs that I rate 7 or higher are added. I gave 10 to *Cold Heart*, because I really enjoy listening to this song and it indeed delights my day. I gave 9 to *Murder In My Mind*. It is great but not quite on the same level as *Cold Heart*. I gave 7 to *Miss You* and *Escapism. – Sped Up*. I think they are decent satisfactory and can pass my threshold. I gave 6 to *I'm Good (Blue)* and *Quevedo*. I think they did not meet the criteria for being added to my library. While they may be enjoyable in small listening doses, I do not have a strong desire to listen to them frequently. For the rest of the songs, I honestly do not quite enjoy that much, so I gave the ratings 5 or below. I am glad that out of 20 songs selected by the model, two of them were already existed in my library. This is a strong evidence that the model is pretty effective and accurate in capturing my musical preference.

Additionally, while I was listening to the songs, I noticed that the majority of the songs selected by the model were from EDM genre, known for their high energy and rhythm. Furthermore, all the songs were American pop music, with only one K-pop song appearing as the final entry. While Global Weekly Top 100 songs from Spotify Chart proved to be a good dataset for testing my model, but there might be some limitations which I will address in the Reviewing the Process - Process Evaluation Report part.

- Overall, in my opinion, a recommendation system model with 30% accuracy is quite satisfactory. Based on my personal experience, out of 30 songs that Spotify suggests in its Discover Weekly section every Monday, I am only able to add a maximum of three songs to my library. Therefore, I think that the prediction model that I built has a better performance than Spotify's current algorithm.

- Collaborative filtering method

  - For collaborative filtering method, as discussed above, I will listen to those 10 songs which were attained based on two similarity metrics and determine how many songs I want to add to my library.

	Euclidean Distance	Cosine Similarity
The 5 most similar tracks to Soft		
1	Hong Kong	Citgo
2	Mongoloid – 2009 Remaster	Hello, Dolly!
3	Bad Habits – uncut	Love, Life and Money
4	Stepping Into Tomorrow	Tadow
5	This Town (feat. Sasha Sloan)	Good Vibrations – Remastered

  - Similarly, I scaled each song from 1 to 10 and determined whether they should be added to my library in the below table.

	Track	Artist	Scale	Added to my library
Euclidean Distance				
1	Hong Kong	Gorillaz	1	No
2	Mongoloid – 2009 Remaster	DEVO	1	No
3	Bad Habits – uncut	Maxwell	5	No
4	Stepping Into Tomorrow	Donald Byrd	1	No
5	This Town (feat. Sasha Sloan)	Kygo, Sasha Sloan	5	No

	Track	Artist	Scale	Added to my library
Cosine Similarity				
1	Citgo	Chief Keef	2	No
2	Hello, Dolly!	Paul Anka	1	No
3	Love, Life And Money	Little Willie John	1	No
4	Tadow	Masego FKJ	8	Already in my library
5	Good Vibrations – Remastered	The Beach Boys	3	No

I felt disappointed that none of the songs met the criteria of being added to my library based on either Euclidean Distance or Cosine Similarity metrics, except that one song *Tadow* which was already in my library. I am unsure why the collaborative filtering method is performing poorly. My initial thought is due to the nature of the generalized dataset. This dataset consists of 170,654 pieces songs stored in Spotify database, spanning from year 1921 to 2020. Given that I was born in 1990s, it is possible that I may not fully appreciate songs from the 1990s or earlier. Therefore, I retrieved the year information for these 10 songs as shown in the below two screenshots. As expected, six out of 10 songs are from earlier decades. Another issue, from my perspective, may be that using a single song as a target and then finding the most similar songs based on two similarity metrics could not provide a comprehensive view of the algorithm's performance. Hence, I am considering testing the algorithm with multiple target songs to evaluate its robustness and reliability. This approach can provide the algorithm with more comprehensive information of the audio features, which may improve the performance. I will address this approach in the Reviewing the Process – Process Evaluation Report part.

Code

```
#Retrieve the year information of the 10 songs
genl=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Dat

songs = ['Hong Kong', 'Mongoloid - 2009 Remaster', 'Bad Habits - uncut', 'Stepping Into Tomorrow', 'This Town (feat. Sasha Sl

filtered_genl = genl.loc[genl['track'].isin(songs)]

track_and_year = filtered_genl.groupby('track')['year'].min().reset_index()

print(track_and_year)
```

Output

```
                         track  year
            Bad Habits - uncut  2009
                          Citgo  2012
  Good Vibrations - Remastered  1967
                 Hello, Dolly!  1964
                     Hong Kong  1958
          Love, Life And Money  1958
     Mongoloid - 2009 Remaster  1978
        Stepping Into Tomorrow  1975
                         Tadow  2018
This Town (feat. Sasha Sloan)  2017
```

- The business goals for this project are: 1) explore the behind algorithms of music recommendation systems; and 2) optimize the recommendation system to benefit Spotify users.
  From the current standpoint, only the content-based filtering method meets my business goals. With regard to the collaborative filtering method, I will revise and see if it can

meet the business goals. As content-based filtering method outperforms Spotify Discover Weekly, I think that I optimized the recommendation system algorithm. By sharing my work on the public platform, I can benefit Spotify users by providing them with access to my code.

Deliverable 2: Model Approval

In conclusion, the simple linear regression model worked pretty well. It has 30% accuracy, outperforming the Spotify Discover Weekly. The model meets the project business criteria.

At present, the two similarity metrics unfortunately did not perform as well as the simple linear regression model. However, I am still in the process of making revisions, and the model approval is still pending.

Task 5.2: Reviewing the Process
Deliverable: Process Evaluation Report

At this stage, it is time to evaluate the process and the steps taken for this project. I have completed the first five phases of the project, and the Modeling Evaluation phase is currently ongoing.

- Let's Get Started
    - This assignment provides an overview of my proposed project for this course. It includes a brief description of the project, the reasons for my choice of the topic, the datasets that I will be working with, and the methods that I plan to use.
    - As this is the very preliminary phase, I was able to quickly come up with the idea of working on a cool data project about Spotify, which has always been my passion. I think that nothing was overlooked, the steps were properly executed, sufficient time was given to this assignment, and no issues or problems that could impact the final product be corrected before deployment.
- 1.0 Business Understanding
    - This assignment provides a comprehensive overview of my project from the business perspective. It includes a detailed project scope, which encompasses the background/program management, business goals, business success criteria, constraints, and organizational and business impact. Moreover, it covers the assessment of the project's situation, which includes the inventory of resources, requirement, assumptions and constraints, risks, contingencies, terminology, costs and benefits. Finally, it includes the data mining scope that details the data mining goals, success criteria, and resource plan that details project plan and initial assessment of tools and techniques.

- I remembered that the due date of this assignment is at the end of January, when my workload was particularly heavy. However, I was still able to complete the assignment on time. There was nothing being overlooked, the steps were properly executed, and no issues or problems that could impact the final product be corrected before deployment. However, not sufficient time was given to this assignment. I conducted some research in terms of the Spotify recommendation system algorithms and searched for appropriate datasets for statistical modeling. While I was satisfied with the datasets I found, I wish I could delve deeper into public perceptions of the Spotify recommendation system by reading more research papers.

- 2.0 Data Understanding

  - This assignment is about understanding the data for my project. It includes how I gathered the data, the descriptions of the data, explorations of the data, and the verification of the data quality. In my case, I have three datasets that will be used for my project. They are individual dataset, global weekly top 100 songs dataset and generalized dataset. For each dataset, I provided the source and the format. Moreover, I described the fields of the data, including the audio features, general features and other features. Additionally, I summarized a table to show the range or distribution of the quantifiable variables. At last, I checked the data quality of all these three datasets.

  - I requested the extension of the due date of this assignment as I was swamped with my work at that time. Thanks to the professor's approval, I had an extra day to work on it, so I was able to complete it without rushing. I ensured nothing was overlooked, all steps were properly executed, and no issues or problems that could impact the final product be corrected before deployment.

- 3.0 Data Preparation

  - This assignment is about preparation of the data for my project. It includes the steps of selecting data, cleaning data, constructing data, integrating data and formatting data. For selecting the data, I elaborated the rationales of the inclusion and exclusion of the datasets. For cleaning data, I listed out the detail and action used to clean my data. For example, I removed unnecessary rows, removed zero value, renamed some columns, etc. For constructing data, I created some new variables, transferred the data across from the datasets, pulled the audio feature data from Spotify Web API. For integrating data, I merged some sub datasets. Finally, for formatting data, I ensured all three datasets were consistent in terms of the attribute naming and data format.

  - Data preparation is a very crucial phase in the data analysis project. This process is significantly necessary because the raw data may contain errors, inconsistencies, and missing values that can affect the accuracy and validity of the analysis. Without proper data preparation, the results of the analysis can be inaccurate and misleading. We do not want garbage in, garbage out. Therefore, it is essential to invest sufficient

time and effort in the data preparation phase to ensure the accuracy and reliability before the data modeling.

- I dedicated a significant amount of time to this phase. I ensured the datasets were ready for the modeling. As I did not encounter any issues with the data quality during the modeling phase, I am confident that I did not overlook anything, executed all steps properly, and there were no issues or problems that could impact the final project to be corrected before deployment.

- 4.0 Data Modeling

  - This assignment is about modeling of the data for my project. It includes selecting modeling techniques, designing tests, building models, and assessing models. For selecting modeling techniques, which was more theory level based, I introduced different types of models including simple linear regression model, linear regression model with interactions, stepwise model, decision tree model, random forest model, and neural network model as well as four similarity metrics, which are Euclidean distance, Pearson Correlation coefficient, Cosine similarity, and Jaccard similarity. For designing tests, I split the data into training, testing and validation sets. For building models, I firstly used the min-max method to scale my audio features data from 0 to 1, and then I conducted the train/validation/test data split. After that, I provided the model descriptions, including the reasons of the inclusions of the variables of each model, as well as how the model should be interpreted in my case. Next, I built the models and documented the rationales of the codes of each model. Finally, for assessing models, in terms of content-based filtering method, I presented a summary table of the performance of each model and interpreted the results of the optimal model (i.e. simple linear regression model) based on the RMSE. However, for the collaborative filtering method, as it is unsupervised learning method, the assessment approach involves manually inspection of the results, and I reserved this evaluation for the next phase. Moreover, for revising parameter settings, I performed some revisions of the linear regression models by adding or removing some interactions. Unfortunately, they did not have a smaller RMSE compared to the simple linear regression model. I also tried the Manhattan distance instead of Euclidean distance by adjusting the p value to one, four out of five songs remain the same except *Stepping Into Tomorrow* was replaced by *When I say I do*.

  - Data modeling involves selecting the appropriate modeling techniques, preprocessing the data, building the model, tuning the hyperparameters, and evaluating the performance of the model. It requires a deep understanding of the data and the business problem, as well as expertise in selecting and implementing the appropriate modeling techniques.

  - From my perspective, data modeling phase is the core of a data project as it involves creating quantitative models that help in predicting or classifying the target variable. The effectiveness and success of a data project heavily rely on the accuracy and efficiency of the data modeling phase. To ensure that I can have ample time for this

phase, I started working on it well in advance and devoted my majority of my time to it. Although I encountered several issues when coding, I was able to resolve them through research. However, one issue that I could not resolve - when I used Pearson correlation coefficient similarity metric to find the similar songs of my target song, there was not enough memory available to perform the request of creating a distance matrix of size (170654, 170654). I attributed this issue to the nature of the dataset and made a decision not to use this metric. Apart from that, I followed the CRISP-PM guide and executed all steps correctly. There were no issues or problems that could impact the final project to be corrected before deployment.

- 5.0 Modeling Evaluation

  - This assignment focuses on the post phase of data modeling. When it comes to modeling evaluation, it looks more broadly at which model best meets the business needs. In my data modeling assignment, I've addressed that I would reserve the manual inspection for the Modeling Evaluation phase since it is not technically focused.

  - For content-based filtering method, I fit the optimal model to the global weekly top 100 songs dataset and selected the top 20 songs which have the largest predicted streaming time. I listened to these 20 songs and determined six songs that I like. For collaborative filtering method, I used two similarity metrics and found 10 songs similar to *Soft*. I listened to these 10 songs and determined only one song that I like. Nothing has been overlooked and all steps have been properly executed so far. Sufficient time was given to this assignment. However, as I mentioned earlier, the content-based filtering model has some limitations, and the two similarity metrics used showed poor performance. To address this, I would like to make the following adjustments to see whether I can make any improvement.

  - Content-based filtering method

	Track	Artist	Scale	Added to my library
1	I'm Good (Blue)	David Guetta, Bebe Rexha	6	No
2	Murder In My Mind	Kordhell	9	Yes
3	Marisola – Remix	Cris Mj, Duki, Nicki Nicole, etc.	4	No
4	505	Arctic Monkeys	3	No
5	Hype Boy	NewJeans	3	No
6	Leao	Marilia Mendonca	1	No
7	Save Your Tears	The Weekend	5	No
8	Calm Down	Rema	3	No
9	SPIT IN MY FACE!	ThxSoMch	3	No
10	Calm Down (with Selena Gomez)	Rema, Selena Gomez	N/A	N/A (duplicate as #8)
11	10:35	Tiesto, Tate McRae	8	Already in my library
12	Miss You	Oliver Tree, Robin Schulz	7	Yes
13	Cold Heart – PNAU Remix	Elton John, Dua Lipa, RNAU	10	Yes!
14	LOKERA	Rauw Alejandro, Lyanno, Brray	6	No
15	Shinunoga E-Wa	Fujii Kaze	4	No
16	I Ain't Worried	OneRepublic	4	No
17	Quevedo: Bzrp Music Sessions, Vol. 52	Bizarrap, Quevedo	6	No
18	Sweater Weather	The Neighbourhood	2	No
19	Stay	The Kid LAROI, Justin Bieber	9	Already in my library
20	Escapism. – Sped Up	RAYE, 070 Shake	7	Yes
21	ANTIFRAGILE	LE SSERAFIM	5	No

- The above are the top 20 songs with the largest streaming time based on the optimal model. While I was trying listening, I noticed the majority of the songs selected by the model were from EDM genre, known for their high energy and rhythm. Furthermore, all the songs were American pop music, with only one K-pop song appearing as the final entry. While Global Weekly Top 100 songs from Spotify Chart proved to be a good dataset for testing my model, but there might be some selection bias if we used this dataset. To elaborate further, I am self-aware that my music preference may not be fully aligned with that of most people. The Global Weekly Top 100 songs primarily includes pop music in English lyrics. My music tastes lean towards genres such as EDM, K-Indie, R&B and Soul music. As it is true that EDM is generally liked by many people, it makes sense that the top 20 songs selected by my model were predominantly from EDM genre. To mitigate this bias, I plan to try the revised generalized dataset. As I've mentioned earlier in the collaborative filtering method section, songs from the earlier ages may not align my music taste. To address this, I retrieved a subset of songs from the year 2011 to 2020. I am curious to know if the model can be improved by switching to a dataset with a broader range of genres.

Code

```
#Review the process

#Content-based filtering method

import pandas as pd

#try using generalized dataset for the optimal model
genl=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Dat

# Create a boolean mask to filter the rows based on the "year" variable
mask = (genl["release_date"] >= '2011') & (genl["release_date"] <= '2020')

# Apply the boolean mask to retrieve the subset of the dataset
genl_sub = genl.loc[mask]

print()

# define the variables used in the formula
a = 4131844
b = 105326200
c = 298014
d = -2343501
e = -548063
f = -10475390
g = -904630
h = -2069824
i = 819883
j = -5897379
k = 9139
l = -3775279

# calculate the predicted msPlayed using the formula and variables
genl_sub['pred_msPlayed'] = l + (a * genl_sub['danceability']) + (b * genl_sub['energy']) + (c * genl_sub['key']) + (d * genl

# sort the DataFrame by pred_msPlayed in descending order
genl_sub_sorted = genl_sub.sort_values ('pred_msPlayed', ascending = False)

# show the top 20 songs by pred_msPlayed
top20_songs = genl_sub_sorted.head(20)

# print the top 20 songs
print(top20_songs[['track', 'artist']])

#print the observations of the sub generalized dataset
print(genl_sub.shape[0])
```

Output

track	artist
Jiyuu no Tsubasa	['Linked Horizon']
Aww Sh*t - Instrumental Mix	['Jason Rivas', 'Supersonic Lizards']
Greece 2000 - Extended Mix	['Jason Rivas', 'Cosmic Phosphate']
Me!Me!Me!, Pt. 1	['TeddyLoid', 'DAOKO']
PAW Patrol on a Roll	['PAW Patrol']
Kill Your Heroes	['AWOLNATION']
Guren no Yumiya	['Linked Horizon']
Soda	['Kartel']
Gita	['Jason Rivas', 'Positive Feeling']
Shinzo wo Sasageyo!	['Linked Horizon']
Sex	['The 1975']
Promises	['NERO']
Runaway	['Mat Kearney']
Boom Boom Boom Boom Boom Boom Boom Boom Boom B...	['Dan Bull']
Elevate	['St. Lucia']
Hell	['Disturbed']
I Got	['Young the Giant']
Some Time Alone, Alone	["Melody's Echo Chamber"]
My Body	['Young the Giant']
Timber (feat. Ke$ha)	['Pitbull', 'Kesha']

Similarly as what I did when I used the Global Weekly Top 100 Songs dataset, I listened to each one, rated it on a scale from 1 to 10, and indicated whether I would like to add it to my library.

	Track	Artist	Scale	Added to my library
1	Jiyuu no Tsubasa	Linked Horizon	2	No (J-pop)
2	Aww Sh*t – Instrumental Mix	Jason Rivas Supersonic Lizards	2	No
3	Greece 2000 – Extended Mix	Jason Rivas Cosmic Phosphate	2	No
4	Me!Me!Me!, Pt. 1	TeddyLoid DAOKO	4	No (J-pop)
5	PAW Patrol on a Roll	PAW Patrol	3	No
6	Kill Your Heroes	AWOLNATION	3	No
7	Guren no Yumiya	Linked Horizon	2	No (J-pop)
8	Soda	Kartel	3	No
9	Gita	Jason Rivas Positive Feeling	6	No
10	Shinzo wo Sasageyo!	Linked Horizon	3	No (J-pop)
11	Sex	The 1975	5	No
12	Promises	NERO	4	No
13	Runaway	Mat Kearney	3	No
14	Boom Boom Boom Boom Boom B…	Dan Bull	7	Yes
15	Elevate	St. Lucia	3	No
16	Hell	Disturbed	2	No
17	I Got	Young the Giant	3	No
18	Some Time Alone, Alone	Melody’s Echo Chamber	2	No
19	My Body	Young the Giant	2	No
20	Timber (feat. Ke$ha)	Pitbull Kesha	6	No

After listening to these 20 songs, I found that they are mostly electronic and rock music, with some diversity in language, including four Japanese songs. Interestingly, the algorithm predicted that I would like Linked Horizon, Jason Rivas, and Young the Giant, as it suggests three songs by Linked Horizon, two songs by Jason Rivas, and two songs by Young the Giant out of 20 songs. Unfortunately, I am not a fan of any of these three artists. Out of the 20 songs, only one song kind of caught my eye. I am not sure why the optimal model did not perform as well as in adjusted generalized dataset compared to Global Weekly Top 100 songs. One potential reason is that generalized dataset is too large, which it may become difficult for me to find meaningful patterns and relationships between variables. Larger datasets often have a lot of noise or irrelevant data that may make it more challenging to identify the patterns. However, as I checked the size of the adjusted generalized dataset, it only includes 440 observations which should not be too large.

I will still use Global Weekly Top 100 songs as the test dataset in this case.

- Collaborative filtering method

	Track	Artist	Scale	Added to my library
Euclidean Distance				
1	Hong Kong	Gorillaz	1	No
2	Mongoloid – 2009 Remaster	DEVO	1	No
3	Bad Habits – uncut	Maxwell	5	No
4	Stepping Into Tomorrow	Donald Byrd	1	No
5	This Town (feat. Sasha Sloan)	Kygo, Sasha Sloan	5	No
Cosine Similarity				
1	Citgo	Chief Keef	2	No
2	Hello, Dolly!	Paul Anka	1	No
3	Love, Life And Money	Little Willie John	1	No
4	Tadow	Masego FKJ	8	Already in my library
5	Good Vibrations – Remastered	The Beach Boys	3	No

  - The above 10 songs were selected as five most similar songs as *Soft* based on two similarity metrics, respectively. As mentioned earlier, I plan to test the algorithm with multiple target songs instead of using one single target song. I used five K-indie songs as the target songs. They are *Soft*, *Polaroid*, *Vacance in September*, *Soothe (feat. Jumbo)*, and *bath*. In the below codes, these five target songs were defined as a list of strings. The individual dataset then was filtered to only include rows corresponding to those target songs. After that, I calculated the average value of audio features and used these values to find the five most similar songs in the generalized dataset using two similarity metrics. Note that I did not use the adjusted generalized dataset as it did not provide better results in the content-based filtering method section.

    Code

```
#Review the process

#Collaborative filtering method

#Euclidean Distance

import pandas as pd
import numpy as np

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the generalized dataset
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:15].mean()

# Extract the relevant audio features from the generalized dataset
genl_features = genl.iloc[:, 5:16].values

# Calculate the Euclidean distances between the target features and all other tracks
distances = np.sqrt(np.sum(np.square(genl_features - target_features.values), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:6]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(6):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Output

```
Across The Great Divide by ['The Band']:
danceability: 0.73
energy: 0.539
key: 7
loudness: -7.323
mode: 1
speechiness: 0.0556
acousticness: 0.419
instrumentalness: 0.00188
liveness: 0.222
valence: 0.636
tempo: 110.581
```

```
El Sastre by ['Los Tigres Del Norte']:
danceability: 0.826
energy: 0.625
key: 7
loudness: -6.906
mode: 0
speechiness: 0.0325
acousticness: 0.447
instrumentalness: 0.0
liveness: 0.36
valence: 0.939
tempo: 110.326
```

```
Johnny Boy's Bones by ['Colter Wall', 'The Dead South']:
danceability: 0.443
energy: 0.616
key: 7
loudness: -7.193
mode: 0
speechiness: 0.0543
acousticness: 0.112
instrumentalness: 0.00219
liveness: 0.0834
valence: 0.574
tempo: 110.028
```

```
Nice, Nice, Very Nice by ['Ambrosia', 'Alan Parsons']:
danceability: 0.468
energy: 0.462
key: 7
loudness: -7.102
mode: 1
speechiness: 0.035
acousticness: 0.0881
instrumentalness: 0.00162
liveness: 0.106
valence: 0.585
tempo: 110.354
```

```
Mine (Glee Cast Version) by ['Glee Cast']:
danceability: 0.787
energy: 0.344
key: 7
loudness: -7.331
mode: 1
speechiness: 0.031
acousticness: 0.706
instrumentalness: 0.0
liveness: 0.0643
valence: 0.361
tempo: 110.036
```

```
I'll Never Love Again - Film Version by ['Lady Gaga', 'Bradley Cooper']:
danceability: 0.451
energy: 0.34
key: 7
loudness: -7.466
mode: 1
speechiness: 0.031
acousticness: 0.763
instrumentalness: 2.49e-06
liveness: 0.135
valence: 0.221
tempo: 110.502
```

Code

```
import pandas as pd
import numpy as np

#Cosine Similarity

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the generalized dataset
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:15].mean()

# Extract the relevant audio features from the generalized dataset
genl_features = genl.iloc[:, 5:16].values

# Calculate the Cosine Similarities between the target features and all other tracks
similarities = np.dot(genl_features, target_features.values) / (np.sqrt(np.sum(np.square(genl_features), axis=1)) * np.sqrt(r

# Find the indices of 5 tracks with the highest similarity (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(similarities)[::-1][:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Output

```
And The Green Grass Grows All Around by ['Barney']:
danceability: 0.704
energy: 0.517
key: 11
loudness: -10.651
mode: 1
speechiness: 0.122
acousticness: 0.647
instrumentalness: 0.0
liveness: 0.199
valence: 0.899
tempo: 171.264

Men of Good Fortune by ['Lou Reed']:
danceability: 0.549
energy: 0.464
key: 10
loudness: -9.4
mode: 1
speechiness: 0.0545
acousticness: 0.708
instrumentalness: 0.00016
liveness: 0.182
valence: 0.328
tempo: 150.125

Pink Skies (Demo) by ['Wiley from Atlanta']:
danceability: 0.565
energy: 0.566
key: 9
loudness: -8.737
mode: 0
speechiness: 0.141
acousticness: 0.485
instrumentalness: 6.55e-06
liveness: 0.104
valence: 0.435
tempo: 138.836

Born to Love You by ['George Duke']:
danceability: 0.485
energy: 0.63
key: 9
loudness: -8.841
mode: 1
speechiness: 0.0325
acousticness: 0.431
instrumentalness: 0.0
liveness: 0.235
valence: 0.39
tempo: 137.335

I Saw Three Ships by ['Sting']:
danceability: 0.775
energy: 0.396
key: 10
loudness: -9.61
mode: 1
speechiness: 0.0583
acousticness: 0.316
instrumentalness: 1.14e-06
liveness: 0.172
valence: 0.646
tempo: 154.571
```

	Track	Artist	Scale	Added to my library
Euclidean Distance				
1	Across The Great Divide	The Band	3	No
2	El Sastre	Los Tigres Del Norte	4	No
3	Johnny Boy's Bones	Colter Wall The Dead South	4	No
4	Nice, Nice, Very Nice	Ambrosia Alan Parsons	5	No
5	Mine (Glee Cast Version)	Glee Cast	8	Yes
Cosine Similarity				
1	And The Green Grass Grows All Around	Barney	1	No
2	Men of Good Fortune	Lou Reed	2	No
3	Pink Skies (Demo)	Wiley from Atlanta	8	Yes
4	Born to Love You	George Duke	6	No
5	I Saw Three Ships	Sting	2	No

I was able to find one out of five songs in both Euclidean Distance and Cosine Similarity metrics. Also, there is one song *Born to Love You* which I gave a rating of 6, which almost met my criteria for adding it to my library. Using multiple target songs slightly improved the accuracy of the similarity metric compared to the previous result where only one song matched my taste out of the total 10 recommended songs. By using more than one target song, the algorithm had more information to identify and patterns and similarities in the data, which improved its ability to recommend similar songs.

Task 5.3: Determining the Next Steps

Deliverable 1: Possible Actions

I attempted to enhance the performance of the models for both content-based filtering method and collaborative filtering method; however, I faced difficulty in improving the results when I switched to the generalized dataset as the test dataset for the content-based filtering method. Fortunately, for the collaborative filtering method, the result improved slightly when I adjusted the size of target songs. As the simple regression model has 30% accuracy in song recommendations, which outperforms the Spotify Discover Weekly, I believe this model is ready for deployment.

There should be certain steps repeated to improve the results, and I have done those in the Reviewing the Process section and tried to improve. While the results did not achieve my expectations, they were also not disappointing. There is no need to undertake a new data-mining project, as I have at least one model that demonstrates excellent performance, despite not all models performing well.

Regarding the possible actions to be taken, given that we will have the deployment plan assignment in the subsequent phase, my first step is to conduct a thorough inspection of my models and the generalized dataset to eliminate any possible errors or mistakes that may have been made during the modeling/evaluation processes. Following that, I will assess whether there is any further room for me to update the models based on the model performance or changes in available data.

Deliverable 2: Final Decision

From the current perspective, my current final decision is using the simple linear regression model and the global weekly top 100 songs dataset as the test dataset base for our recommendation system algorithm.

6.0 Data Deployment

Task 6.1: Planning Deployment
Deliverable: Deployment Plan

Before coming up with a strategy for putting the models to work in my business, my first step is to conduct a thorough inspection of my models and generalized dataset to eliminate any possible errors or mistakes that may have been made during the modeling/evaluation processes.

Below is a recap of my outcome.

Model	Test Dataset	Target Song	Outcome	Accuracy
Content-based filtering method				
Original				
Simple Linear Regression Model	Global Weekly Top 100 Songs Dataset	N/A	Six out of 20 songs were added to my library.	30%
Revised				
Simple Linear Regression Model	Adjusted Generalized Dataset (Songs from year 2011 to 2020)	N/A	One out of 20 songs was added to my library.	5%
Collaborative filtering method				
Original				
Euclidean Distance	Generalized Dataset	One Song	None of five songs were added to my library.	0%
Cosine Similarity	Generalized Dataset	One Song	One of five songs was selected. This song was already in my library.	20%
Revised				
Euclidean Distance	Generalized Dataset	Five Songs	One of five songs were added to my library.	20%
Cosine Similarity	Generalized Dataset	Five Songs	One of five songs was selected. This song was already in my library.	20%

- Content-based filtering method
    - Based on my review of the codes, I did not find any issues.
- Collaborative filtering method
    - Based on my review of the codes, in terms of the collaborative filtering method, I found there was an overlook in terms of the audio features across from individual dataset and generalized dataset. I accidently included the duration column when I retrieved the audio features. I revised my codes accordingly. There should be 11 columns of audio features included instead of 12 columns.
    - One Target Song
        - Euclidean Distance

Old Code

```
#Collaborative filtering method - Unsupervised Learning
#Similarity Metric 1
#Euclidean Distance
import pandas as pd
import numpy as np

indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Input the name of the track I like
track = 'Soft'

# Filter the DataFrame based on the track name
filtered_indiv = indiv[indiv['track'] == 'Soft']

# Display the audio features for the specified track
print(filtered_indiv[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',

# Load the individual dataset and filter for the desired track
track_row = indiv.loc[indiv['track'] == track]
track_features = track_row.iloc[:, 4:15].values[0]
track_artist = track_row.iloc[0, 2]

# Load the generalized dataset and extract the relevant audio features
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:16].values
genl_artists = genl.iloc[:, 1].values

# Calculate the Euclidean distances between the track features and all other tracks
distances = np.sqrt(np.sum(np.square(genl_features - track_features), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Revised Code

```
#Collaborative filtering method - Unsupervised Learning
#Similarity Metric 1
#Euclidean Distance
import pandas as pd
import numpy as np

indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Input the name of the track I like
track = 'Soft'

# Filter the DataFrame based on the track name
filtered_indiv = indiv[indiv['track'] == 'Soft']

# Display the audio features for the specified track
print(filtered_indiv[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',

# Load the individual dataset and filter for the desired track
track_row = indiv.loc[indiv['track'] == track]
track_features = track_row.iloc[:, 4:14].values[0]
track_artist = track_row.iloc[0, 0]

# Load the generalized dataset and extract the relevant audio features
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:15].values
genl_artists = genl.iloc[:, 1].values

# Calculate the Euclidean distances between the track features and all other tracks
distances = np.sqrt(np.sum(np.square(genl_features - track_features), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:15]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:15])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Revised Output

```
The 5 most similar tracks to "Soft" by J_ust are:

Gravity by ['EDEN']:
danceability: 0.507
energy: 0.436
key: 11
loudness: -7.756
mode: 0
speechiness: 0.044
acousticness: 0.713
instrumentalness: 0.000169
liveness: 0.114
valence: 0.246

Romeo And Juliet by ['Indigo Girls']:
danceability: 0.424
energy: 0.327
key: 11
loudness: -7.758
mode: 0
speechiness: 0.048
acousticness: 0.788
instrumentalness: 0.0
liveness: 0.283
valence: 0.453

Who Hurt You? by ['Daniel Caesar']:
danceability: 0.649
energy: 0.493
key: 11
loudness: -7.645
mode: 0
speechiness: 0.11
acousticness: 0.586
instrumentalness: 0.000167
liveness: 0.0667
valence: 0.277

Paint A Lady by ['Susan Christie']:
danceability: 0.392
energy: 0.446
key: 11
loudness: -7.481
mode: 0
speechiness: 0.0711
acousticness: 0.543
instrumentalness: 6.03e-06
liveness: 0.13
valence: 0.382

Al Gany Baad Youmen by ['Samira Said']:
danceability: 0.299
energy: 0.44
key: 11
loudness: -7.624
mode: 0
speechiness: 0.0372
acousticness: 0.853
instrumentalness: 0.0
liveness: 0.22
valence: 0.461
```

- Cosine Similarity

Old Code

```
#Similarity Metric 3
#Cosine Similarity

from sklearn.metrics.pairwise import cosine_similarity

# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_features = indiv.loc[indiv['track'] == track].iloc[:, 4:15].values[0]

# Load the generalized dataset and extract the relevant audio features and artist names
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:16].values
genl_artists = genl.iloc[:, 1].values

# Calculate the cosine similarities between the track features and all other tracks
similarities = cosine_similarity(track_features.reshape(1, -1), genl_features)

# Find the indices of 5 tracks with the highest cosine similarity
most_similar_indices = np.argsort(similarities[0])[-6:-1]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl_artists[most_similar_indices]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Revised Code

```
#Similarity Metric 3
#Cosine Similarity

from sklearn.metrics.pairwise import cosine_similarity

# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_features = indiv.loc[indiv['track'] == track].iloc[:, 4:14].values[0]
track_artist = track_row.iloc[0, 0]

# Load the generalized dataset and extract the relevant audio features and artist names
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D
genl_features = genl.iloc[:, 5:15].values
genl_artists = genl.iloc[:, 1].values

# Calculate the cosine similarities between the track features and all other tracks
similarities = cosine_similarity(track_features.reshape(1, -1), genl_features)

# Find the indices of 5 tracks with the highest cosine similarity
most_similar_indices = np.argsort(similarities[0])[-6:-1]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:15]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl_artists[most_similar_indices]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:15])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Revised Output

```
The 5 most similar tracks to "Soft" by J_ust are:

Mal de Ausencia - Remasterizado by ['Francisco Canaro', 'Roberto Maida']:
danceability: 0.655
energy: 0.319
key: 11
loudness: -7.66
mode: 0
speechiness: 0.0588
acousticness: 0.993
instrumentalness: 0.000302
liveness: 0.102
valence: 0.576

Paint A Lady by ['Susan Christie']:
danceability: 0.392
energy: 0.446
key: 11
loudness: -7.481
mode: 0
speechiness: 0.0711
acousticness: 0.543
instrumentalness: 6.03e-06
liveness: 0.13
valence: 0.382
```

```
Who Hurt You? by ['Daniel Caesar']:
danceability: 0.649
energy: 0.493
key: 11
loudness: -7.645
mode: 0
speechiness: 0.11
acousticness: 0.586
instrumentalness: 0.000167
liveness: 0.0667
valence: 0.277
```

```
Romeo And Juliet by ['Indigo Girls']:
danceability: 0.424
energy: 0.327
key: 11
loudness: -7.758
mode: 0
speechiness: 0.048
acousticness: 0.788
instrumentalness: 0.0
liveness: 0.283
valence: 0.453
```

```
Alone by ['Bazzi']:
danceability: 0.516
energy: 0.414
key: 9
loudness: -6.335
mode: 0
speechiness: 0.0351
acousticness: 0.614
instrumentalness: 0.0
liveness: 0.0851
valence: 0.358
```

- Five Target Songs
  - Euclidean Distance

Old Code

```
#Collaborative filtering method

#Euclidean Distance

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project'

# Load the generalized dataset
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:15].mean()

# Extract the relevant audio features from the generalized dataset
genl_features = genl.iloc[:, 5:16].values

# Calculate the Euclidean distances between the target features and all other tracks
distances = np.sqrt(np.sum(np.square(genl_features - target_features.values), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:6]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Revised Code

```python
#Collaborative filtering method

#Euclidean Distance

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the generalized dataset
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:14].mean()

# Extract the relevant audio features from the generalized dataset
genl_features = genl.iloc[:, 5:15].values

# Calculate the Euclidean distances between the target features and all other tracks
distances = np.sqrt(np.sum(np.square(genl_features - target_features.values), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:6]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:15]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:15])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Revised Output

```
The 5 most similar tracks to the target set ['Soft', 'Polaroid', 'Vacance in September', 'Soothe
(feat. Junmo)', 'bath'] are:

It Took Me By Surprise by ['Maria Mena']:
danceability: 0.614
energy: 0.532
key: 7
loudness: -7.095
mode: 0
speechiness: 0.0505
acousticness: 0.395
instrumentalness: 0.0
liveness: 0.0718
valence: 0.299

The Lonesome Road by ['Sammy Davis Jr.']:
danceability: 0.574
energy: 0.542
key: 7
loudness: -6.89
mode: 0
speechiness: 0.048
acousticness: 0.701
instrumentalness: 0.0
liveness: 0.0798
valence: 0.329

Diana by ['Comus']:
danceability: 0.418
energy: 0.592
key: 7
loudness: -7.045
mode: 0
speechiness: 0.043
acousticness: 0.57
instrumentalness: 2.3e-05
liveness: 0.223
valence: 0.457

Beaver Junction by ['Count Basie']:
danceability: 0.519
energy: 0.532
key: 7
loudness: -6.899
mode: 0
speechiness: 0.0384
acousticness: 0.67
instrumentalness: 0.00703
liveness: 0.0998
valence: 0.539

Pretender - Acoustic by ['AJR']:
danceability: 0.811
energy: 0.365
key: 7
loudness: -6.943
mode: 0
speechiness: 0.0291
acousticness: 0.639
instrumentalness: 0.0
liveness: 0.224
valence: 0.539
```

Old Code

```python
#Cosine Similarity

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the generalized dataset
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:15].mean()

# Extract the relevant audio features from the generalized dataset
genl_features = genl.iloc[:, 5:16].values

# Calculate the Cosine Similarities between the target features and all other tracks
similarities = np.dot(genl_features, target_features.values) / (np.sqrt(np.sum(np.square(genl_features), axis=1)) * np.sqrt(n

# Find the indices of 5 tracks with the highest similarity (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(similarities)[::-1][:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:16]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:16])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Revised Code

```python
#Cosine Similarity

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the generalized dataset
genl = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\D

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:14].mean()

# Extract the relevant audio features from the generalized dataset
genl_features = genl.iloc[:, 5:15].values

# Calculate the Cosine Similarities between the target features and all other tracks
similarities = np.dot(genl_features, target_features.values) / (np.sqrt(np.sum(np.square(genl_features), axis=1)) * np.sqrt(n

# Find the indices of 5 tracks with the highest similarity (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(similarities)[::-1][:5]

# Extract the information of the most similar tracks
most_similar_features = genl.iloc[most_similar_indices, 5:15]
most_similar_track_names = genl.iloc[most_similar_indices, 2]
most_similar_artists = genl.iloc[most_similar_indices, 1]

# Get the names of the audio feature coefficients
feature_names = list(genl.columns[5:15])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Revised Output

```
The 5 most similar tracks to the target set ['Soft', 'Polaroid', 'Vacance in September', 'Soothe
(feat. Junmo)', 'bath'] are:
```

```
Trooper's Lament by ['Sgt. Barry Sadler']:
danceability: 0.608
energy: 0.477
key: 11
loudness: -10.573
mode: 1
speechiness: 0.036
acousticness: 0.82
instrumentalness: 6.13e-05
liveness: 0.401
valence: 0.565
```

```
Stfu by ['Pink Guy']:
danceability: 0.827
energy: 0.524
key: 10
loudness: -9.819
mode: 1
speechiness: 0.0536
acousticness: 0.552
instrumentalness: 0.000112
liveness: 0.198
valence: 0.356
```

```
Remember the Alamo by ['Johnny Cash']:
danceability: 0.619
energy: 0.553
key: 10
loudness: -9.604
mode: 1
speechiness: 0.0568
acousticness: 0.828
instrumentalness: 0.00235
liveness: 0.132
valence: 0.447
```

```
Media Vuelta by ['Eydie Gormé', 'Los Panchos']:
danceability: 0.664
energy: 0.48
key: 10
loudness: -9.536
mode: 1
speechiness: 0.0437
acousticness: 0.619
instrumentalness: 0.0
liveness: 0.365
valence: 0.584
```

```
It's Alright To Cry by ['Rosy Grier']:
danceability: 0.734
energy: 0.464
key: 10
loudness: -10.018
mode: 1
speechiness: 0.0366
acousticness: 0.719
instrumentalness: 7.64e-06
liveness: 0.0989
valence: 0.53
```

- Summary of the outcomes based on the revised codes

	Track	Artist	Scale	Added to my library
One Target Song				
Euclidean Distance				
1	Gravity	EDEN	3	No
2	Romeo And Juliet	Indigo Girls	3	No
3	Who Hurt You?	Daniel Caesar	5	No
4	Paint A Lady	Susan Christie	1	No
5	Al Gany Baad Youmen	Samira Said	2	No
Cosine Similarity				
1	Mal de Ausencia – Remasterizado	Francisco Canaro Roberto Maida	1	No
2	Paint A Lady	Susan Christie	1	No
3	Who Hurt You?	Daniel Caesar	5	No
4	Romeo And Juliet	Indigo Girls	3	No
5	Alone	Bazzi	4	No
Five Target Songs				
Euclidean Distance				
1	It Took Me By Surprise	Maria Mena	6	No
2	The Lonesome Road	Sammy Davis Jr.	1	No
3	Diana	Comus	1	No
4	Beaver Junction	Count Basie	4	No
5	Pretender – Acoustic	AJR	10	Yes (already in my library)
Cosine Similarity				
1	Trooper's Lament	Barry Sadler	2	No
2	Stfu	Pink Guy	6	No
3	Remember the Alamo	Johnny Cash	3	No
4	Media Vuelta	Eydie Gorme Los Panchos	3	No
5	It's Alright To Cry	Rosy Grier	4	No

- According to the table shown above, it appears that the revisions made to the code did not result in any improvement. When using a single target song, none of the recommended songs were added to my library. When using five target songs, only one of the recommended songs, *Pretender – Acoustic*, which I gave a 10 rating, was already in my library based on Euclidean Distance metric. It is interesting to note that the five target songs I picked are all K-Indie songs, which typically have a R&B or Soul vibe. However, the *Pretender – Acoustic* is actually an EDM track. I am unsure how the algorithm was able to suggest such a seemingly ironic result. One good thing I observed that, two songs – *Who Hurt You?* and *Paint A Lady*, were both selected according to two different similarity metrics, which somewhat proved that the revised codes were accurate.

- I decided to have one last try by using the Global Weekly Top 100 Songs dataset as my selection pool.

- One Target Song
  - Euclidean Distance

Revised Code – gt100

```
#Revise codes - gt100

#Collaborative filtering method - Unsupervised Learning
#Similarity Metric 1
#Euclidean Distance
import pandas as pd
import numpy as np

indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Input the name of the track I like
track = 'Soft'

# Filter the DataFrame based on the track name
filtered_indiv = indiv[indiv['track'] == 'Soft']

# Load the individual dataset and filter for the desired track
track_row = indiv.loc[indiv['track'] == track]
track_features = track_row.iloc[:, 4:14].values[0]
track_artist = track_row.iloc[0, 0]

# Load the global weekly top 100 songs dataset and extract the relevant audio features
gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da
gt100_features = gt100.iloc[:, 9:19].values
gt100_artists = gt100.iloc[:, 2].values

# Calculate the Euclidean distances between the track features and all other tracks
distances = np.sqrt(np.sum(np.square(gt100_features - track_features), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:6]

# Extract the information of the most similar tracks
most_similar_features = gt100.iloc[most_similar_indices, 9:19]
most_similar_track_names = gt100.iloc[most_similar_indices, 3]
most_similar_artists = gt100.iloc[most_similar_indices, 2]

# Get the names of the audio feature coefficients
feature_names = list(gt100.columns[9:19])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Revised Output – gt100

```
The 5 most similar tracks to "Soft" by J_ust are:

Sure Thing by Miguel:
danceability: 0.684
energy: 0.607
key: 11
loudness: -8.127
mode: 0
speechiness: 0.1
acousticness: 0.0267
instrumentalness: 0.000307
liveness: 0.191
valence: 0.498

Heat Waves by Glass Animals:
danceability: 0.761
energy: 0.525
key: 11
loudness: -6.9
mode: 1
speechiness: 0.0944
acousticness: 0.44
instrumentalness: 6.7e-06
liveness: 0.0921
valence: 0.531

Just Wanna Rock by Lil Uzi Vert:
danceability: 0.486
energy: 0.545
key: 11
loudness: -7.924
mode: 1
speechiness: 0.0336
acousticness: 0.0652
instrumentalness: 0.00474
liveness: 0.0642
valence: 0.0385
```

```
Yellow by Coldplay:
danceability: 0.429
energy: 0.661
key: 11
loudness: -7.227
mode: 1
speechiness: 0.0281
acousticness: 0.00239
instrumentalness: 0.000121
liveness: 0.234
valence: 0.285
```

```
No Role Modelz by J. Cole:
danceability: 0.69
energy: 0.521
key: 10
loudness: -8.492
mode: 0
speechiness: 0.339
acousticness: 0.324
instrumentalness: 0.0
liveness: 0.0534
valence: 0.494
```

- Cosine Similarity

Revised Code – gt100

```
#Revise codes - gt100

#Similarity Metric 3
#Cosine Similarity

from sklearn.metrics.pairwise import cosine_similarity

# Load the individual dataset and filter for the desired track
track = 'Soft'
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\
track_features = indiv.loc[indiv['track'] == track].iloc[:, 4:14].values[0]
track_artist = track_row.iloc[0, 0]

# Load the global weekly top 100 songs dataset and extract the relevant audio features and artist names
gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da
gt100_features = gt100.iloc[:, 9:19].values
gt100_artists = gt100.iloc[:, 2].values

# Calculate the cosine similarities between the track features and all other tracks
similarities = cosine_similarity(track_features.reshape(1, -1), gt100_features)

# Find the indices of 5 tracks with the highest cosine similarity
most_similar_indices = np.argsort(similarities[0])[-6:-1]

# Extract the information of the most similar tracks
most_similar_features = gt100.iloc[most_similar_indices, 9:19]
most_similar_track_names = gt100.iloc[most_similar_indices, 3]
most_similar_artists = gt100_artists[most_similar_indices]

# Get the names of the audio feature coefficients
feature_names = list(gt100.columns[9:19])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to "{track}" by {track_artist} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()  # print an empty line for spacing between each track
```

Revised Output – gt100

```
The 5 most similar tracks to "Soft" by J_ust are:
Lift Me Up - From Black Panther: Wakanda Forever by Rihanna:
danceability: 0.247
energy: 0.299
key: 9
loudness: -6.083
mode: 1
speechiness: 0.0315
acousticness: 0.899
instrumentalness: 0.0
liveness: 0.131
valence: 0.172

Just Wanna Rock by Lil Uzi Vert:
danceability: 0.486
energy: 0.545
key: 11
loudness: -7.924
mode: 1
speechiness: 0.0336
acousticness: 0.0652
instrumentalness: 0.00474
liveness: 0.0642
valence: 0.0385

Heat Waves by Glass Animals:
danceability: 0.761
energy: 0.525
key: 11
loudness: -6.9
mode: 1
speechiness: 0.0944
acousticness: 0.44
instrumentalness: 6.7e-06
liveness: 0.0921
valence: 0.531

Under The Influence by Chris Brown:
danceability: 0.733
energy: 0.69
key: 9
loudness: -5.529
mode: 0
speechiness: 0.0427
acousticness: 0.0635
instrumentalness: 1.18e-06
liveness: 0.105
valence: 0.31

Bloody Mary by Lady Gaga:
danceability: 0.591
energy: 0.637
key: 9
loudness: -6.365
mode: 0
speechiness: 0.03
acousticness: 0.0107
instrumentalness: 1.95e-06
liveness: 0.113
valence: 0.432
```

- Five Target Songs

- Euclidean Distance

Revised Code – gt100

```
#Revise codes - gt100

#Collaborative filtering method

#Euclidean Distance

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the global weekly top 100 songs dataset
gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:14].mean()

# Extract the relevant audio features from the global weekly top 100 songs dataset
gt100_features = gt100.iloc[:, 9:19].values

# Calculate the Euclidean distances between the target features and all other tracks
distances = np.sqrt(np.sum(np.square(gt100_features - target_features.values), axis=1))

# Find the indices of 5 tracks with the smallest distance (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(distances)[:6]

# Extract the information of the most similar tracks
most_similar_features = gt100.iloc[most_similar_indices, 5:15]
most_similar_track_names = gt100.iloc[most_similar_indices, 3]
most_similar_artists = gt100.iloc[most_similar_indices, 2]

# Get the names of the audio feature coefficients
feature_names = list(gt100.columns[9:19])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Revised Output – gt100

```
The 5 most similar tracks to the target set ['Soft', 'Polaroid', 'Vacance in September', 'Soothe
(feat. Junmo)', 'bath'] are:

Starboy by The Weeknd, Daft Punk:
danceability: 1
energy: 50
key: 119
loudness: 13540082
mode: 0.679
speechiness: 0.587
acousticness: 7
instrumentalness: -7.015
liveness: 1
valence: 0.276

DESPECHÁ by ROSALÍA:
danceability: 5
energy: 29
key: 24
loudness: 14534374
mode: 0.914
speechiness: 0.622
acousticness: 7
instrumentalness: -6.538
liveness: 1
valence: 0.0892

SPIT IN MY FACE! by ThxSoMch:
danceability: 72
energy: 93
key: 9
loudness: 10009994
mode: 0.73
speechiness: 0.785
acousticness: 8
instrumentalness: -6.512
liveness: 1
valence: 0.0554

Miss You by Oliver Tree, Robin Schulz:
danceability: 11
energy: 24
key: 14
loudness: 17020232
mode: 0.587
speechiness: 0.742
acousticness: 6
instrumentalness: -6.64
liveness: 0
valence: 0.0529

Nobody Gets Me by SZA:
danceability: 12
energy: 43
key: 5
loudness: 13664312
mode: 0.358
speechiness: 0.284
acousticness: 7
instrumentalness: -8.285
liveness: 1
valence: 0.0285
```

- Cosine Similarity

Revised Code – gt100

```
#Revise codes - gt100

#Cosine Similarity

# Load the individual dataset
indiv = pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\

# Load the global weekly top 100 songs dataset
gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da

# Define the set of target songs
target_songs = ['Soft', 'Polaroid', 'Vacance in September', 'Soothe (feat. Junmo)', 'bath']

# Filter the individual dataset based on the target songs
target_indiv = indiv[indiv['track'].isin(target_songs)]

# Compute the average feature values for the target songs
target_features = target_indiv.iloc[:, 4:14].mean()

# Extract the relevant audio features from the generalized dataset
gt100_features = gt100.iloc[:, 9:19].values

# Calculate the Cosine Similarities between the target features and all other tracks
similarities = np.dot(gt100_features, target_features.values) / (np.sqrt(np.sum(np.square(gt100_features), axis=1)) * np.sqrt

# Find the indices of 5 tracks with the highest similarity (i.e., the 5 most similar tracks)
most_similar_indices = np.argsort(similarities)[::-1][:6]

# Extract the information of the most similar tracks
most_similar_features = gt100.iloc[most_similar_indices, 9:19]
most_similar_track_names = gt100.iloc[most_similar_indices, 3]
most_similar_artists = gt100.iloc[most_similar_indices, 2]

# Get the names of the audio feature coefficients
feature_names = list(gen1.columns[5:15])

# Display the most similar tracks and their features
print(f'The 5 most similar tracks to the target set {target_songs} are:')
for i in range(5):
    print(f'{most_similar_track_names.iloc[i]} by {most_similar_artists.iloc[i]}:')
    for j in range(len(feature_names)):
        print(f'{feature_names[j]}: {most_similar_features.iloc[i, j]}')
    print()
```

Revised Output – gt100

```
The 5 most similar tracks to the target set ['Soft', 'Polaroid', 'Vacance in September', 'Soothe
(feat. Junmo)', 'bath'] are:

Lavender Haze by Taylor Swift:
danceability: 0.733
energy: 0.436
key: 10
loudness: -10.489
mode: 1
speechiness: 0.08
acousticness: 0.258
instrumentalness: 0.000573
liveness: 0.157
valence: 0.0976

Shinunoga E-Wa by Fujii Kaze:
danceability: 0.6
energy: 0.76
key: 6
loudness: -6.124
mode: 0
speechiness: 0.0452
acousticness: 0.166
instrumentalness: 4.09e-05
liveness: 0.189
valence: 0.519

Starboy by The Weeknd, Daft Punk:
danceability: 1
energy: 50
key: 119
loudness: 13540082
mode: 0.679
speechiness: 0.587
acousticness: 7
instrumentalness: -7.015
liveness: 1
valence: 0.276

Ditto by NewJeans:
danceability: 0.814
energy: 0.641
key: 6
loudness: -5.957
mode: 0
speechiness: 0.111
acousticness: 0.027
instrumentalness: 0.0
liveness: 0.0993
valence: 0.183

No Role Modelz by J. Cole:
danceability: 0.69
energy: 0.521
key: 10
loudness: -8.492
mode: 0
speechiness: 0.339
acousticness: 0.324
instrumentalness: 0.0
liveness: 0.0534
valence: 0.494
```

- Summary of the outcomes tested on the Global Weekly Top 100 Songs

	Track	Artist	Scale	Added to my library
One Target Song				
Euclidean Distance				
1	Sure Thing	Miguel	7	Yes
2	Heat Waves	Glass Animals	6	No
3	Just Wanna Rock	Lil Uzi Vert	4	No
4	Yellow	Coldplay	3	No
5	No Role Modelz	J. Cole	6	No
Cosine Similarity				
1	Lift Me Up	Rihanna	5	No
2	Just Wanna Rock	Lil Uzi Vert	4	No

	Track	Artist	Scale	Added to my library
3	Heat Waves	Glass Animals	6	No
4	Under The Influence	Chris Brown	5	No
5	Bloody Mary	Lady Gaga	4	No
Five Target Songs				
Euclidean Distance				
1	Starboy	The Weeknd Daft Punk	9	Yes (already in my library)
2	DESPECHA	ROSALIA	6	No (Latin Pop)
3	SPIT IN MY FACE!	ThxSoMch	3	No
4	Miss You	Oliver Tree Robin Schulz	8	Yes (already in my library)
5	Nobody Gets Me	SZA	3	No
Cosine Similarity				
1	Lavender Haze	Taylor Swift	5	No
2	Shinunoga E-Wa	Fujii Kaze	4	No (J-pop)
3	Starboy	The Weekend Daft Punk	9	Yes (already in my library)
4	Ditto	NewJeans	8	Yes (K-pop)
5	No Role Modelz	J.Cole	6	No

- According to the table shown above, when using a single target song, one song was added to my library based on the Euclidean Distance metric while no song was added to my library based on the Cosine Similarity metric. When using five target songs, two out of five songs were selected and they were already in my library based on the Euclidean Distance metric; two out of five songs were selected and one song was already in my library. What is more, the Euclidean Distance and Cosine Similarity metrics enabled me to discover Latin Pop, K-pop, and J-pop songs, in addition to American pop songs with English lyrics. This brought me joy as it has added diversity to the recommended songs. Additionally, it is pretty satisfactory that Euclidean Distance metric recommends me a song by ROSALIA, who is one of my favorite artists.

- Below is a summarized table outlining the improvements after I changed the test dataset to Global Weekly Top 100 Songs Dataset. The accuracy of two similarity metrics improved from 20% to 40% if I used five target songs. The accuracy of Euclidean distance's accuracy increased from 0% to 20% while the Cosine similarity drops from 20% to 0% if I used one target song.

Model	Test Dataset	Target Song	Outcome	Accuracy
Content-based filtering method				
Original				
Simple Linear Regression Model	Global Weekly Top 100 Songs Dataset	N/A	Six out of 20 songs were added to my library.	30%
Revised				
Simple Linear Regression Model	Adjusted Generalized Dataset (Songs from year 2011 to 2020)	N/A	One out of 20 songs was added to my library.	5%
Collaborative filtering method				
Original				
Euclidean Distance	Generalized Dataset Global Weekly Top 100 Songs Dataset	One Song	None of five songs were added to my library. One of five songs was added to my library.	0% 20%
Cosine Similarity	Generalized Dataset Global Weekly Top 100 Songs Dataset	One Song	One of five songs was selected. This song was already in my library. None of five songs were added to my library.	20% 0%
Revised				
Euclidean Distance	Generalized Dataset Global Weekly Top 100 Songs Dataset	Five Songs	One of five songs were added to my library. Two of five songs were selected. These two songs were already in my library.	20% 40%

Cosine Similarity	Generalized Dataset Global Weekly Top 100 Songs Dataset	Five Songs	One of five songs was selected. This song was already in my library. Two of five songs were selected. One song was already in my library.	20% 40%

- At this point, it really triggers my curiosity to analyze the generalized dataset further. Here are the steps I followed.

  - Data Quality Recheck
    - Range
    - Missing Values
    - Outliers

    I have performed this step in the data preparation assignment, but it is worthwhile for me to double check. Upon re-examining the generalized dataset, I have confirmed that there are no missing values or outliners among the 11 audio features I used to predict the streaming time. The ranges of key, loudness, and tempo were scaled between 0 to 1 using min-max scaling method. Therefore, the data quality is good for modeling.

  - Visualizations

    - I decided to visualize the range and distribution of the audio features between the generalized dataset and global weekly top 100 songs dataset to dig into further.

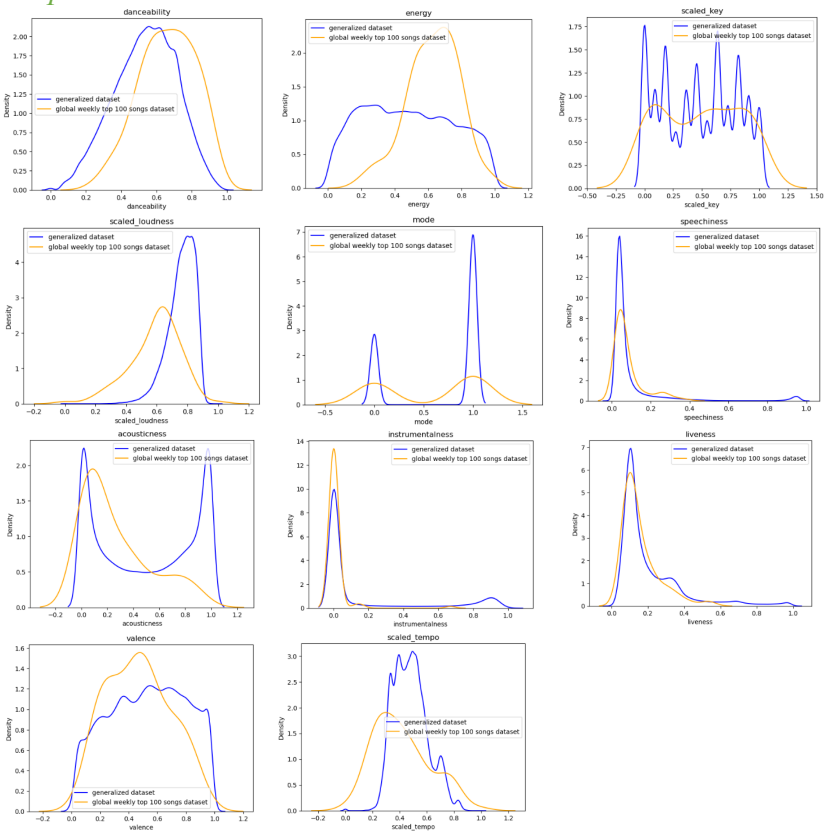
    - *Code*

```
#Generalized Dataset
#Data Quality Recheck
#Visualizations between genl and gt100

#Impoart Datasets
import pandas as pd
gt100=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Da
genl=pd.read_csv(r'C:\Users\madlin\Desktop\les notes de madeline\mademoiselle madeline stuff boooom\DS\Applied DS Project\Dat

#Scale some of the attributes
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Scale the key, loudness, tempo attributes using min-max scaling in global weekly top 100 songs dataset
key_scaler = MinMaxScaler()
scaled_key = key_scaler.fit_transform(gt100[['key']])

loudness_scaler = MinMaxScaler()
scaled_loudness = loudness_scaler.fit_transform(gt100[['loudness']])

tempo_scaler = MinMaxScaler()
scaled_tempo = tempo_scaler.fit_transform(gt100[['tempo']])

gt100['scaled_key']=scaled_key
gt100['scaled_loudness']=scaled_loudness
gt100['scaled_tempo']=scaled_tempo

gt100.head()
```

```
# Scale the key, loudness, tempo attributes using min-max scaling in generalized dataset
key_scaler = MinMaxScaler()
scaled_key = key_scaler.fit_transform(genl[['key']])

loudness_scaler = MinMaxScaler()
scaled_loudness = loudness_scaler.fit_transform(genl[['loudness']])

tempo_scaler = MinMaxScaler()
scaled_tempo = tempo_scaler.fit_transform(genl[['tempo']])

genl['scaled_key']=scaled_key
genl['scaled_loudness']=scaled_loudness
genl['scaled_tempo']=scaled_tempo

genl.head()
```

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


# Define the list of audio features
audio_features = ['danceability', 'energy', 'scaled_key', 'scaled_loudness', 'mode', 'speechiness', 'acousticness', 'instrume

# Plot the KDE line for each audio feature for dataset1 and dataset2
for feature in audio_features:
    plt.figure()
    sns.kdeplot(gen1[feature], color='blue', label='generalized dataset', fill=False, common_norm=False)
    sns.kdeplot(gt100[feature], color='orange', label='global weekly top 100 songs dataset', fill=False, common_norm=False)
    plt.title(feature)
    plt.legend()
    plt.show()
```

Output



- I used density (relative value) as the y-axis instead of count (absolute value) because the there was a stark contrast in the number of observations between two datasets. The x-axis is the range of the audio features. Key, loudness and tempo had been scaled. Then I visualized each audio feature between two datasets and found some interesting patterns.
    - Mode, speechiness, instrumentalness, and liveness had similar distributions in both datasets, while the distributions of the other seven audio features differed significantly from one another[7].

[7] The density is estimated based on the kernel function and bandwidth used for Kernel Density Estimation (KDE). The density values between two datasets are not directly comparable. The shape and scale of the KDE curves are what most informative in comparing the distributions of the audio features in the two datasets.

        - On average, songs in global weekly top 100 songs dataset have higher danceability, energy but lower loudness, valence and tempo compared to songs in generalized dataset. My guess is that the current trend is leaning towards a genre like techno, which is a type of electronic dance music (EDM) but has a lower loudness and slight lower tempo.
        - The most significant difference between two datasets is key. Songs in generalized dataset have more variances in key compared to songs in global weekly top 100 songs dataset. This makes sense to me because as there are more genres of songs in the generalized dataset while songs in the global weekly top 100 songs dataset focus on some mainstreams with less diversity.
        - Another interesting point is that in terms of acousticness, songs in the generalized dataset are either low (close to 0.05) or very high (close to 1) while songs in the global weekly top 100 songs dataset are generally low (close to 0.1). I am not exactly sure about this difference.

    - Based on the above explorations, there are variances of distribution between certain audio features including the danceability, energy, loudness, valence, tempo and acousticness between two test datasets. These variances potentially impact the performances of the models. Hence, while building the model is crucial, the selection of the appropriate test datasets is equally important. In certain cases, even with a powerful model, incorrect selection of the test dataset can result in the failure of the model. For instance, if someone is not a fan of J-pop, using a test dataset comprising solely of J-pop won't give favorable results no matter how good the model is. In my case, which is not as extreme as the J-pop example, I would prefer to use recent global songs as the test data pool because the generalized dataset consisting of all songs from 1921 to 2020 really is outdated and does not match my preferences.

Now, the models are ready to use. As this is not a company-wise project but a personal project, I do not have the resources to plan for deployment from a company level, such as building the platform for deployment and continuously monitoring. However, the models can still be utilized for recommendation purposes and provide benefits to the public.

Below is a summary of my strategy with the detailed steps.

First, once I finalize my paper, I will share it in my github page. I hope people who are interested in Spotify recommendation system can offer their comments or ask questions of the work I did. By exchanging ideas and thoughts, we can work together to polish the work and improve the algorithms.

Second, I will consolidate the codes (revise if needed) and give instructions on how to apply my codes to user's personal music data.

Third, I will collect each user's results, including 1) which supervised learning model has the best performance in each user's case; 2) which test dataset has a better result in each user's case; 3) the accuracy of the models.

Fourth, I will conduct an analysis to determine whether the results based on my personal music data overall are consistent with those of other users. If consistent, it would indicate that the algorithms are fairly accurate. If it is not consistent, it would indicate further improvements still need to be made.

At the same time, I will also follow the latest news and research related to this topic and keep myself up to date with the latest developments in this field.

Task 6.2: Planning Monitoring and Maintenance
Deliverable: Monitoring and Maintenance Plan

This section pertains to the monitoring and maintenance, including reviewing of the model's performance, updating models based on model performance or changes in available data. I have already completed these steps at the beginning of this assignment to ensure that the models are correct from a code perspective before my strategic planning deployment.

I agree that data-mining work, like application development, is a cycle. As I plan to publicize my recommendation system algorithms, more people will be involved in testing the performance of the models. If there is any discussion arise regarding the model maintenance or change in available dataset, I am open to them.