Custom program report: Music Player in Python

Bui Minh Thuan

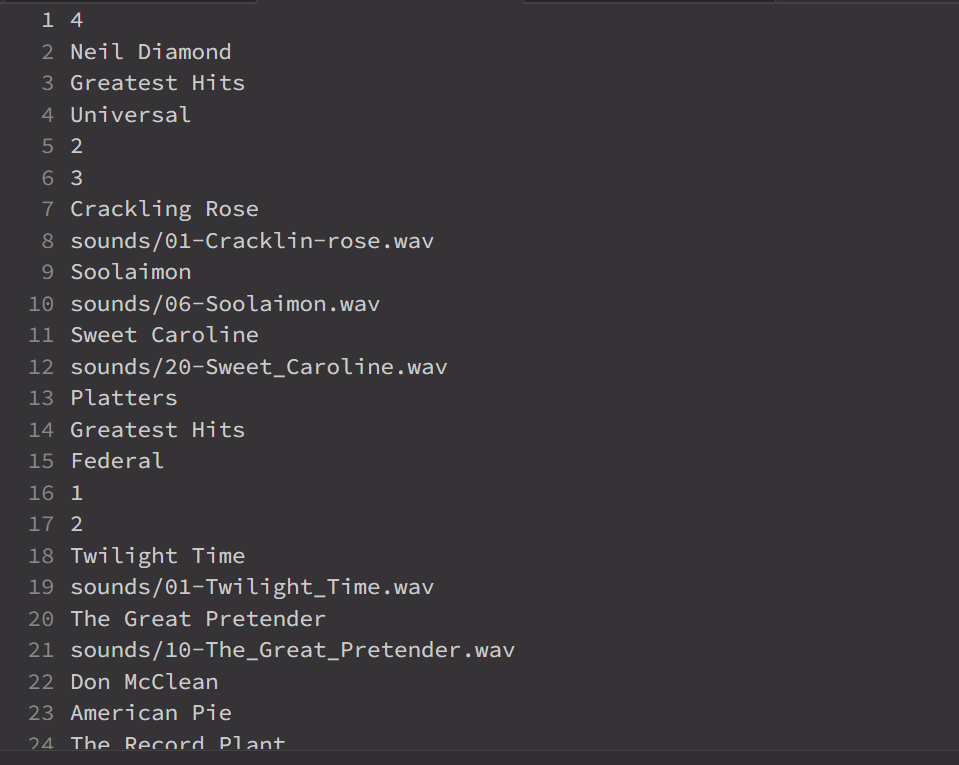
Swinburne University of Technology

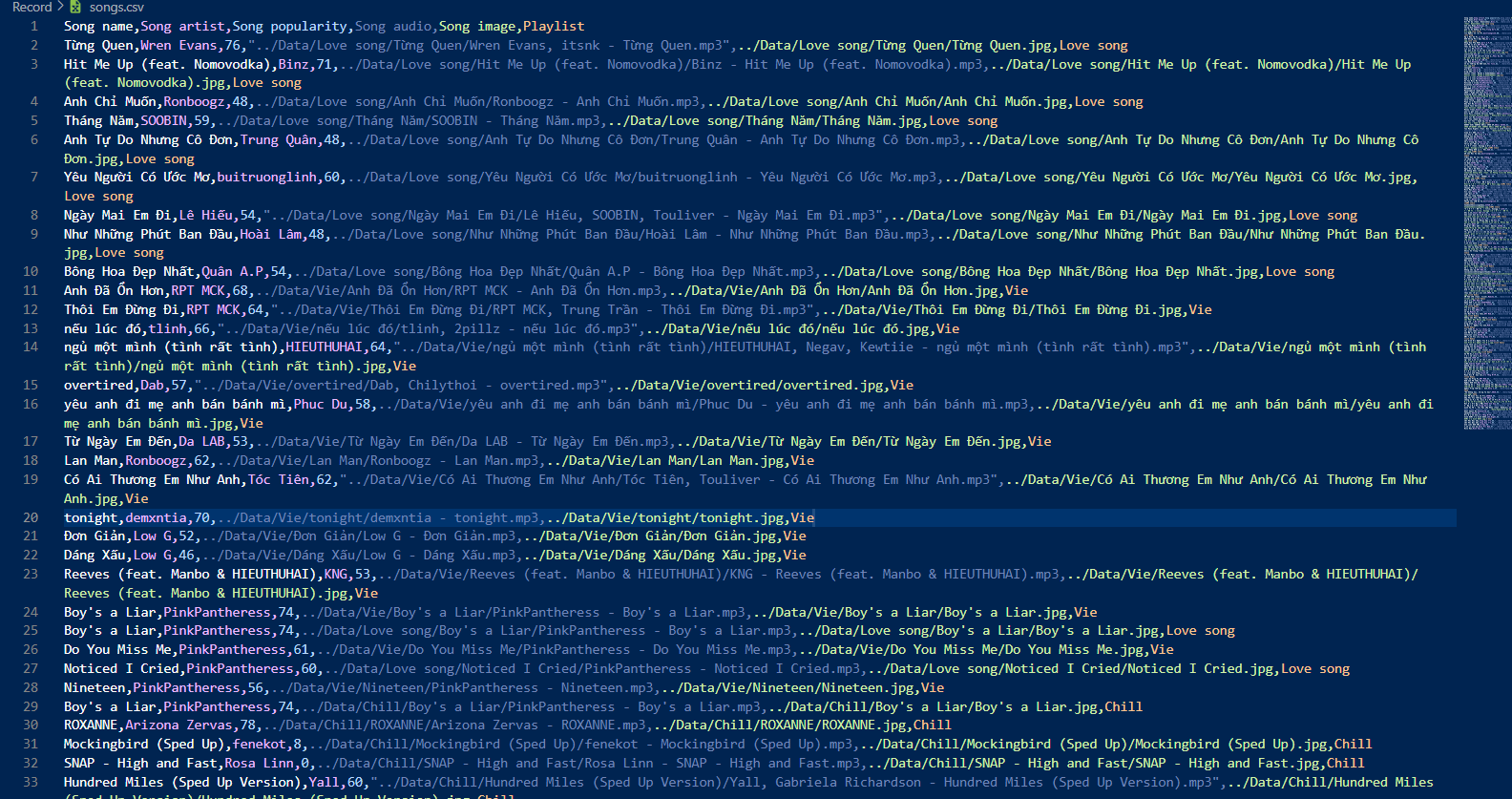
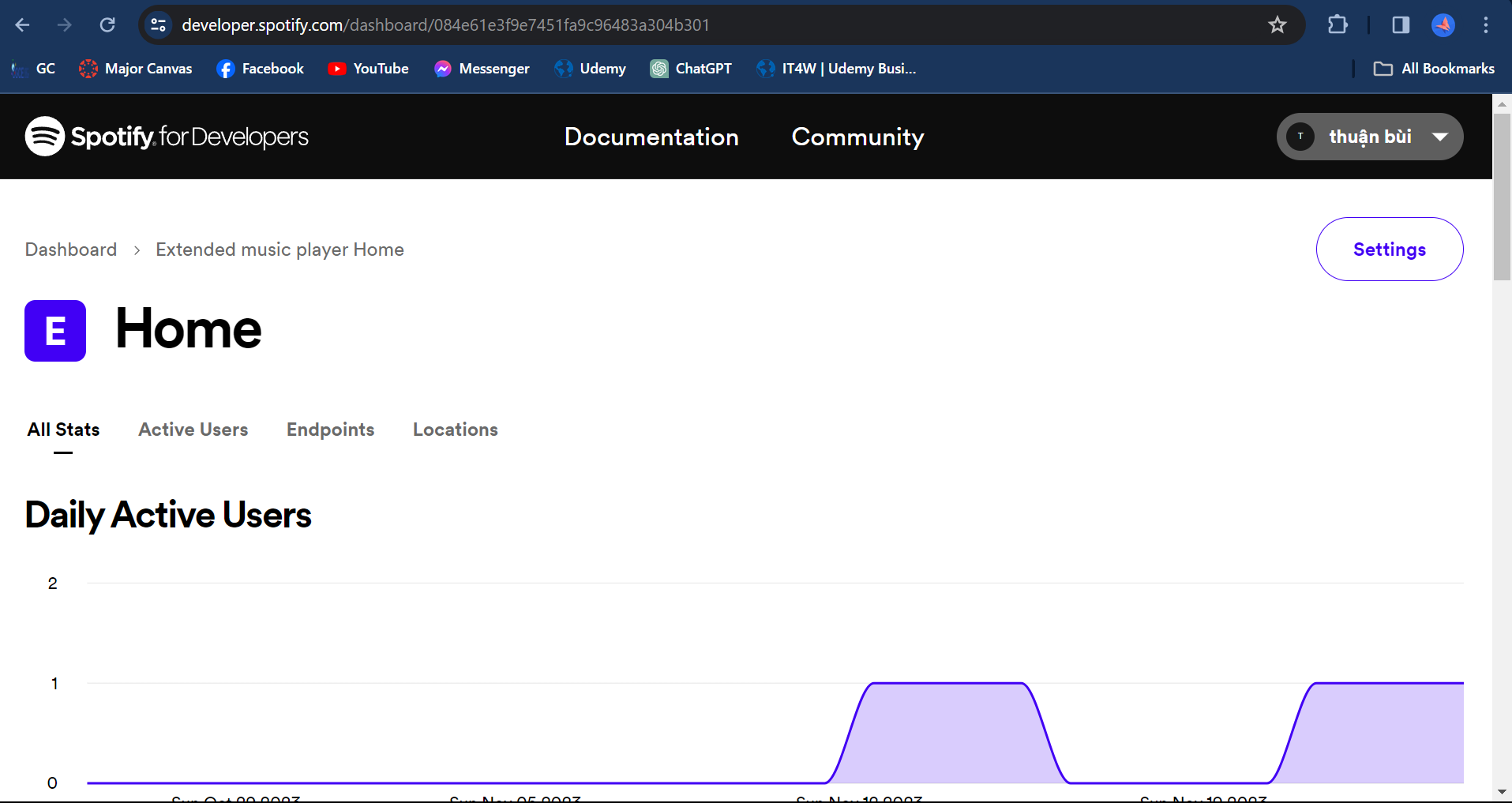
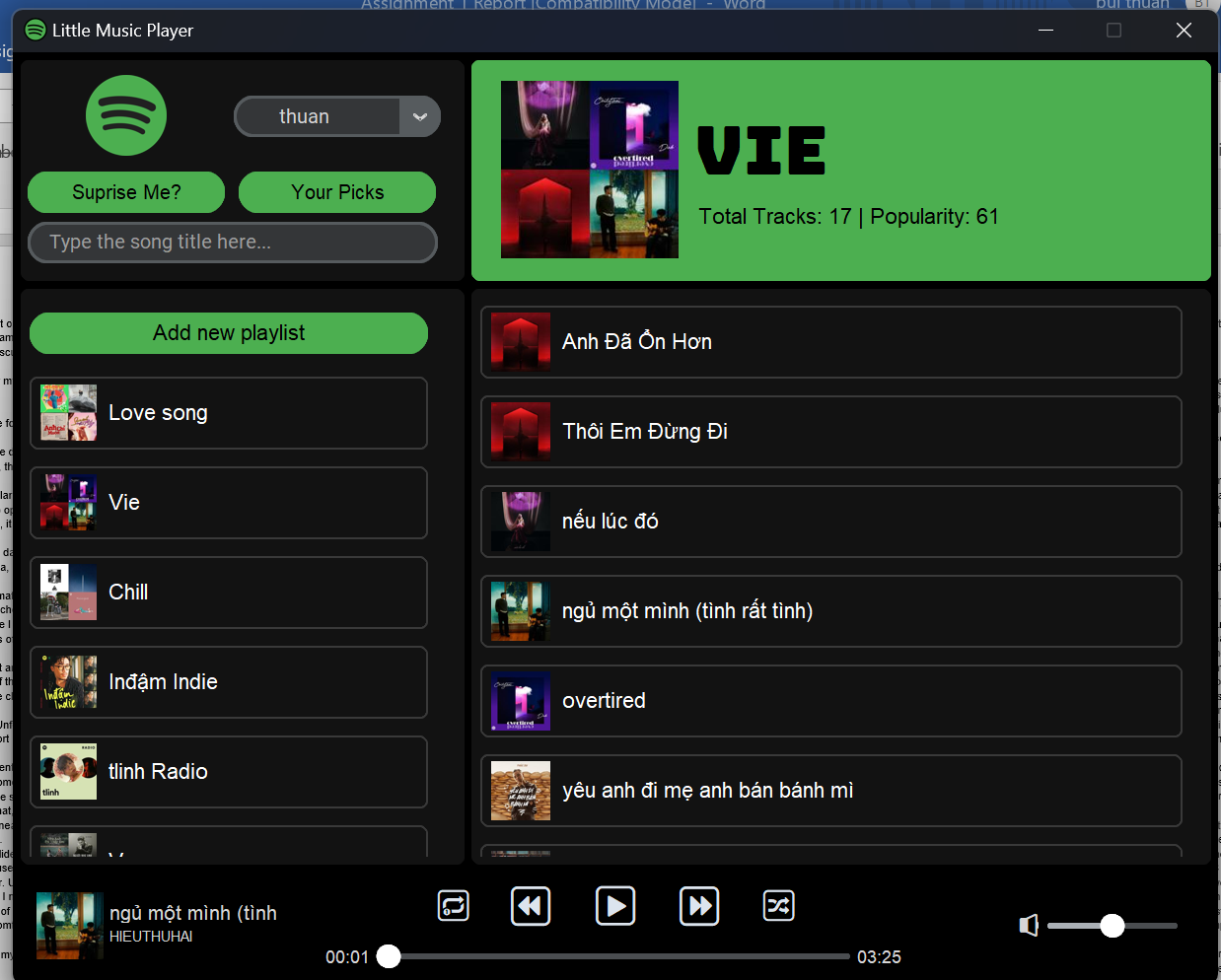
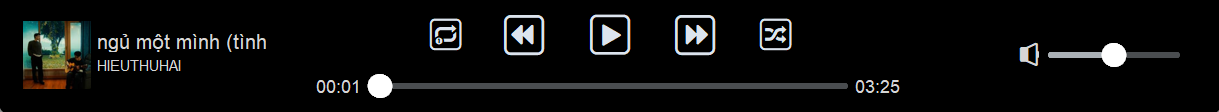
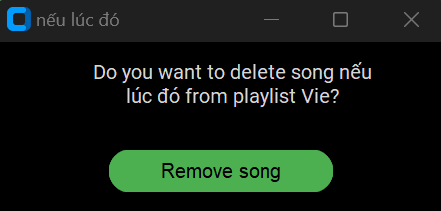
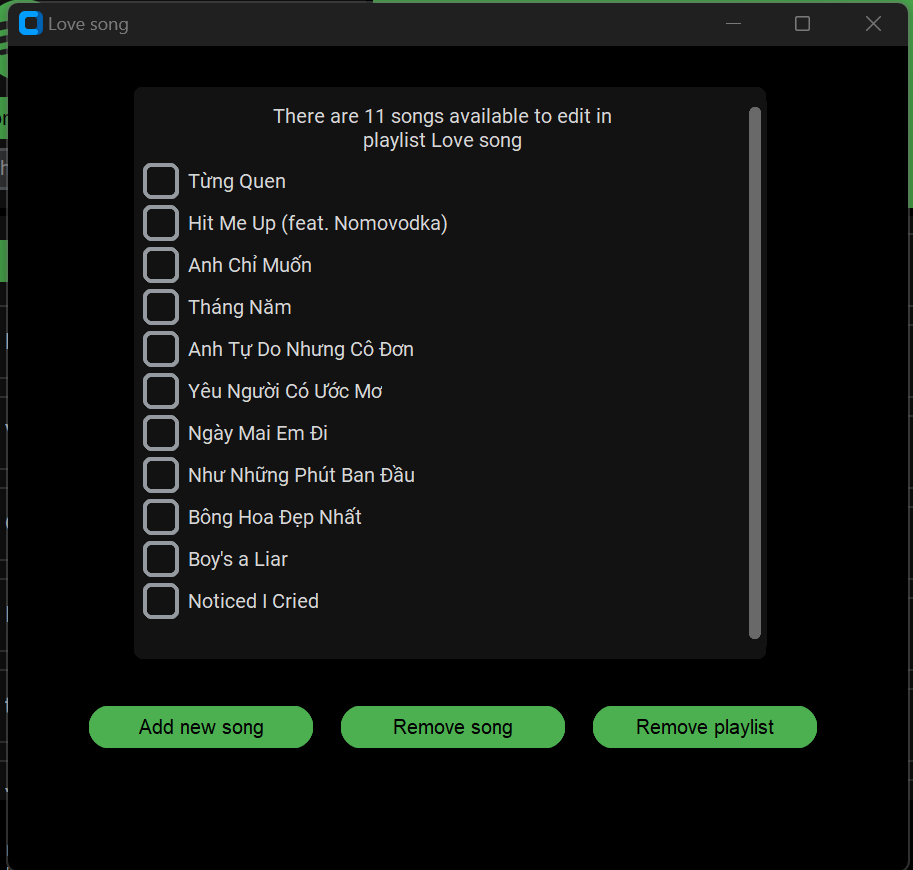
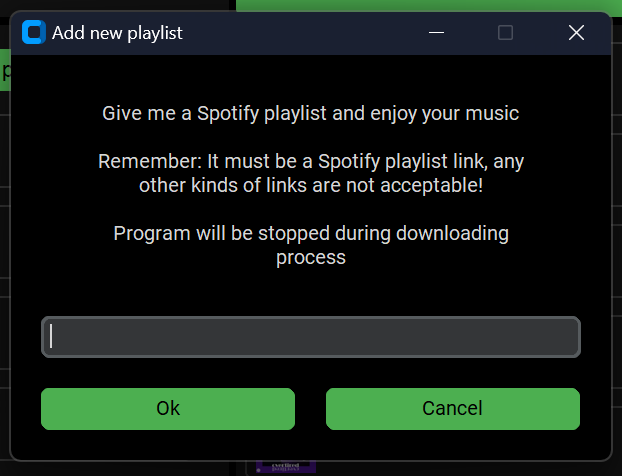
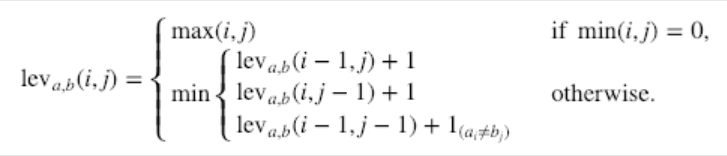
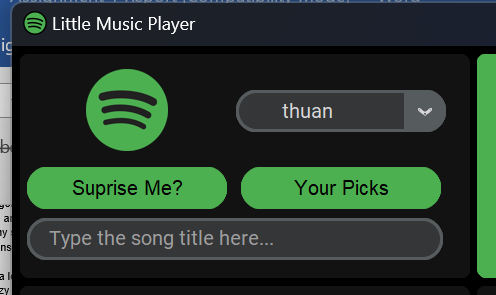
SWH01635

COS10009

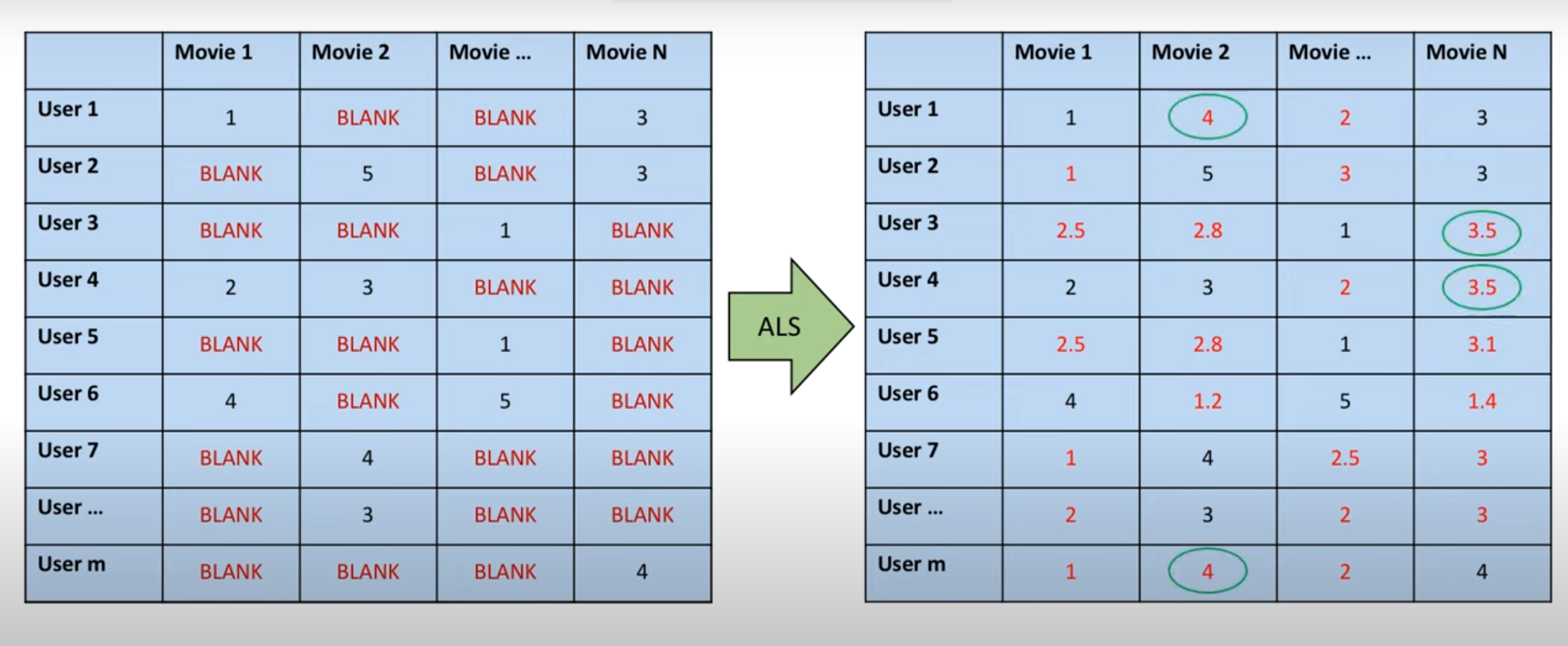
1. **Program overview**My custom program is a music player written in Python. In my music player, user can not only listen to music, but also can edit it based on their personal preferences and get suggested song for them

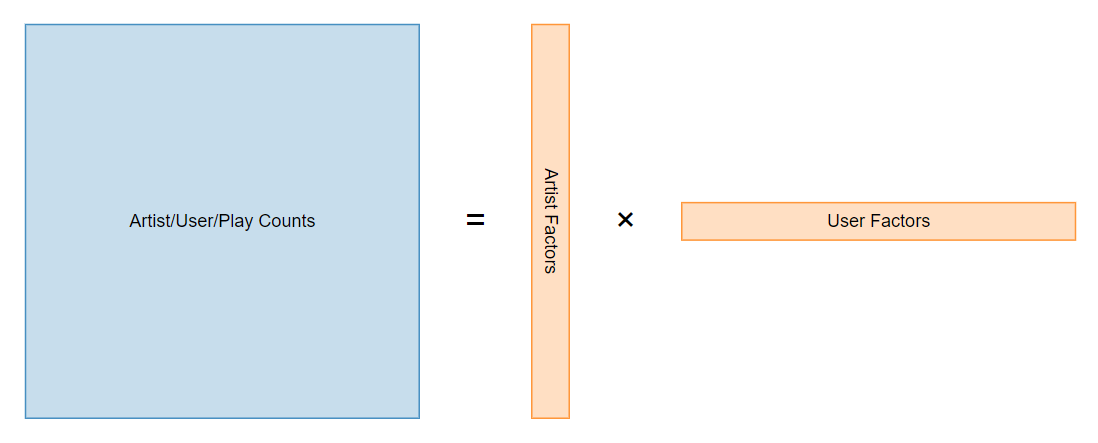
**2. Program features**

Below is all the features inside my program:   
1. Programming language and design  
 Before diving deeper inside how my porgram works, I want to talk a little bit about the tools that I use for this project.  
 In term of programming language, my choice is Python. This is because, Python has a large number of supporting libraries, making easier for me during coding time. More than that, my program also contains some features relating to AI, which Python performs better than Ruby. As the content of Ruby and Python is pretty similar, I have no difficult switching these 2 languages.  
 For the GUI library, I am using a new library called customtkinter. This library is built on top of tkinter, a very popular Python GUI library. In customtkinter, the UI is updated more moderned, make my program more beautiful compared to Gosu or Tkinter. It also has built-in fucntions like scrollable frame or input window to help me improve my program.  
 Lastly, for the design and function inspirations, it mostly comes from Spotify, with a couple of changes to match my preferences.  
2. Database   
 The first feature I want to discuss about is the database of the program. Actually, for the range of this project, we don’t have too much data to handle, but in my opinion, a well-designed database will make the coding process much more easier.  
 During this unit, we have learned a very basic database, utilising read/write with .txt file. This approach has some advantages, like it is easy to use, don’t require external tools or libraries, but, because data is stored in sequence, with each data is stored in each line, from top to bottom, it makes the data file becomes very hard to understand. This is because, in this approach there is nothing like a label or a nametag to help user know what is the value represented for. In case of string, user can guess it based on the meaning of the word, but if it is a number, that is impossible to know what this number tells. Another disadvantage of this approach is it is very difficult to update data, for example, adding new song to a existing playlist or remove a song,…. With all that reasons, using this techiniques is not a wise choice for a custom project, especially with a kind program that contains a wide range of data like music player.  
  
 To solve this problem, my approach is to replace the .txt file with the CSV file. CSV(Comma seperated file) is a file in which data is stored in a table-like format, with rows and columns, seperated by comma. By using CSV file, there are a lot of advantages compared to txt file:  
 1. Easier to read: with field names, we know what the value represents for, make it easier to retrieve data, coding or even debug when needed  
 2. Easier to edit: In csv file, it is very easy to edit a value, because we can specify it based on it location in the database. For example, when we delete a song, simple remove the row that has it songs, but not effect other rows.

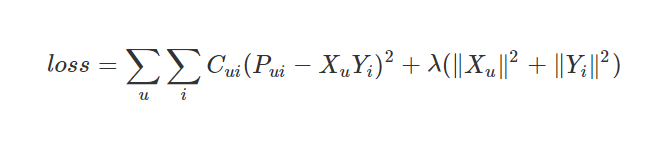
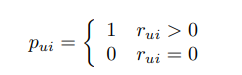
Below is my final database, which is displayed in differrent rows and columns  
  
 And that is how I solve the problems relating to database  
  
3. Getting songs data.  
 In the previous section, I just solved the problem related to database, now my program has a well-designed database. Next stage is to get the data for this database.  
 For this project, the data I need is songs related data. For a song, I need all its information, like its name, artist, cover image,… and the most important data: the audio. This is for 1 song, and I need multiple songs in a playlist, and multiple playlists for the whole program. But the question is, how to get all that data?  
 A very simple approach is going to Chrome, typing the name of the song you want, find a website that allow downloading, download the song, and for the information, just fill it by hand.  
 This approach works, just fine, and the fact is that you can get as many songs as you want with this way, whatever it is. But, the problem is, it is an extreme dauting task. If you want a complete music player, with a lists of playlists, and about at least 10 songs a playlist, you will have to repeat the above process, for, like… 100 times. Obviously not a good tactic.  
 My solution for this problem is to automatic everything using coding. Python is a very good programming language in term of automatic things. It has a lot of libraries supporting that work. In this case, I am using 2 libraries: Spotipy and spotdl.  
 To explain what it does, we need to separate the goal: download the song automatically into 2 parts: 1 is download the song itself, 2 is download the information of this song.  
 At the moment, Spotify is a very popular music player that contains both 2 things we need. So, the question now switch to: scraping Spotify to get that information.  
 Eventhough it is pretty hard for normal person to get information from Spotify, for developers, it is much more easier. Spotify has publiced their own API, allows everybody to use it free. API( Application Programming Interface) is a set of tools that allows users to do things without coding from scratch. For example, with Sptify API, I can turn on the music from my VS CODE. It will run the code that Spotify write to turn on the music and turn it on. I don’t need to know what is the actual code inside Spotify, I just need to know that this API allows me to run it.  
 That is the basic of API, and, in my case, Spotify API allows me to get the data of any songs, or playlists. With supports of Spotipy (htt), a library that works with Spotify API, all I need to do is register with Spotify for some IDs, and get whatever data I want from them.  
  
 However, there is still a minor problem, that is Spotify API doesn’t allow to download the song from them. That means, we need to download from other sites. Luckily, there is a library that helps us to download a song, based on Spotify link. This library is spotdl, and what it does is when we give it a Spotify link and run it, it will find the song in that list, then goes to Youtube Music to download it in MP3 format for us.  
 So now I have all the tools I need to get data for my project, I will combine it into a function called “get\_playlist”. The process of this fucntion will be, I give it the spotify playlist link, it retrieve the songs, and its information, then download it using spotdl. After finish everything, all data will be written into the CSV database for further using.  
 The fanciest thing about this function is that not only the creator like me can use this, but users can also use it to update the program. In the demonstration video, I have described how users can use it, and I will discuss about it in detail in the section for Program Modification.  
 And that is how I can get all the data I need for my project.  
  
4.Play the music  
 Now, I have all the data I need, stored in a nice format. The next thing to do is actually run the program with that data.  
 First thing I want to talk about is how to load the data into the program. While it is pretty easy to load normal data like text or image, things get a little tricky with the audio.   
 There are some ways to do it, and a very popular one is using FileDialog. So, when open the program, users will have option to open the file dialog and choose the song it self. While this techiniques works just well, it seems a little bit outdated, and also not really automatic.  
 My solution for this problem is when I store the data, I also store the file path to the audio file itself. So, when I open the program, data, including this file path, will be loaded into the program automatically.  
 With the above approach, I have enough information loaded into my program, but the next obstacle is, in all of those songs, how to choose what playlist to display, and what song to play. This is very important, because I want to play the exact song that I choose, and I also want to page to multiple pages of playlists but not affect the chosen song.   
 My solution for this is by using index for playlist and song. When a playlist or a song is chosen, the program will take the index of this playlist and this song, and performs action with them. This index can only be changed when another playlist or user button is chosen.  
  
 Now, the next problem is how to play a song. Unfortunately, while customtkinter has updated a lot of from tkinter, it still now support audio file. That means, I need to use an external library for sound handling.  
 There are a couple of options for this requirement, but I think the most effective one is using Pygame. Pygame is a game-development library in Python, but it has great sound supports. With pygame, I can play the song, pause/unpause it, or even re start to the beginning of the song. More than that, Pygame also allows config the volume, or get the position of the song. This has nearly solve all the problems relating to play and control a song in my program.  
 The last problem with play song is creating a slider bar for the song. In customtkinter, there is already a slider for me to use, but I aslo need the positon of the song and also its length to make the full slider. Unfortunately, Pygame doesn’t support getting the length of the song. Therefore, I need to use another library called Mutagen. This library allows me to get the length of an MP3 file very easily. Combining with the built-in slider function of customtkinter, I can build a slider to see and control the song very easily.  
  
 And that is how I play and control the songs in my music player program.  
  
5. Program modification  
 With all the features above, my music player now can work just fine. But, what if I want to edit that, for example, delete a song, add a song or even delete a whole playlist.  
 This part is actually very simple in term of removing songs or playlist. All I need to do is deleting all the records and data relating to the chosen songs, and then reloading the program to see the changes. The same thing is also applied with adding songs from a playlist to another playlist, in which I just need to create a new record of the added song to the new playlist. The only problem here is to remember if a song exists in 2 playlists, if it is deleted in a playlist, it should stay the same in the remaining playlist.   
  
 Things get a little tricky but also interesting in adding new songs externally. Like I have pointed above, I apply the function I used to get songs at first, but now allow users to use it, what it does is poping up a window so user can paste the Spotify link to it, and then let the program do the rest.  
  
  
  
 And that is how the modification in my program works.  
  
6. Search algorithm  
 All the above features cover all the needed function of a music player program, and the next features will be for improving users experience. A very needed feature for this need is a search function, which allows users to search for a song they want to play.  
 This thing seems to be very easy, customtkinter has an option to support taking entry from users, and all I need to do is comparing what user enter with the song in the program. But, in reality, it is not simple like that. Users can not remember the name of the song all the time, so, the program can not find the song that user want.   
 The solution for this is using string-matching algorithm. This is the algorithm that compares the similarities between what user type and the actual song name. The flow of this search will be: user type something, my search function takes this string and calculate the matching rate with all the song inside my program. The song with the highest rate will be the chosen one.  
 But, how can we calculate this rate? There are a lot of matching algorithms, but the most effective and popular one has to be the Fuzzy algorithm. The idea of this algorithm is that it calculate the number of edits we need to do to turn a string into another string. The smalled number, the higher similarites rate. This number is calculated using The Levenshtein Distance, a metric named by Vladimir Levenshtein, who found out the formula for measuring the differences between 2 words. You can see the formal calculation below:  
****  
 Using this formula, I can build an algorithm myself to calculate the similarities between 2 words. But luckily, there is library that helpe me do that. The library is called TheFuzz, which is considered as the most popular string matching algorithm library. By appling this into my program, it will automatically find the name with the highest match to what user had enter, and play it.  
  
 And that is how I apply string-matching algorithms to solve searching-related problem in my program.

7. Recommend song  
 The last feature I want to discuss about is recommend the song to users. Actually, in my program, there are 2 kinds of recommendations: random and recommend  
  
 In term of random, all I need to do is randomize values for the index of playlist and song, then playlist. This part is very simple in Python, by using random.randint. It is a little bit strange to call this a recommend option, but sometimes, it is good to seek new song rather listerning to the same types, genres, and randomize is a great way to do that.  
 Things get tricky in the real recommendation system. Nowadays, recommendation is a very common features in almost any fields. We have “New to you” to recommend new video in Youtube, “You may like” for new content in Facebook, or something similar in retail app like Shoppee,… The same trend also appears in music industry, with Spotify is the leading one. They are very famous for things like “Daily Weeks”, “Made for you”,…  
 Their recommendations system is very complicated, applying a lot of algorithms. But the whole system works around 3 layers:  
 The first one is based on the popularity of a song. Let’s say a song is on trend, it should be recommended to user as they might like it. This is the simpliest type of recommend, and also the less effective. This is because, even this song is trendy now, it may not in the genre that an user like. More than that, the popularity not always show the quality of a song, for example, a song from an idol might get more popularity than a new-debut singer, make the recommendation isn’t precise. This techiniques will work most effective for new user, whom Spotify have no idea about preferences, so recommend to them the on trend music is reasonable  
 The second layer is based on Content-based filtering. The general idea of this technique is it calculate the similarity between items, in this case, between songs. The song will be recommended in like this: User listen to song A and song B, song A and B and C has high similratiy rate, so , recommend song C to user. It just cares about the similarities between items, so that why it is called “Content-based”. This approach solves the problem of the first techniques as it will suggest the song based on previous listening history of users. However, the problem with it is the require a lot of data and analysis to get the exact result.  
 The final one is called Collaborate Filtering. What it does is basically find the relationships between users. For example, user A listen to song C,D,E, while user B listened to song D,E,F. It is recognized that user A and user B has the same taste of a music, and it will suggest song F for user A, song C for user B. This one is the most effective overall as it not required detail analysis between songs, and generally more precise than Content Based Filtering. However, a minor problem with that is it doesn’t work for new user, who have no previous listening history.  
 These 3 techniques are not the only parts of a complete recommendation in big tech, but it covers some very important concepts of a recommender system. For my custom project, because it runs locally, so the popularity approach is not a good choice, and I also can not do external analysis between all the songs. Therefore, I will apply Collaborate Filtering in my program.  
 Before applying it into my program, it is necessary to have a deep understanding about the techniques itself:  
 Actually, Collaborative Filtering based on 2 types of inputs: explicit and implicit. Explcit is the direct kind of interactions, for example, if an user like a song, they must like this song, and this is called explcit feedback. This feedback works well in theory, but in reality, there isn’t enough data for this, and even when user do this, most of the time, they just press like to pass the menu as soon as possible. So, in reality, Collaborate Filtering relies more on implcit feedback.  
 Implicit feedback is different types of interactions from user to a song. It can be, how many times user listen to a song, how long the listen to it,….. By analysing these kinds of feedback, we can get the general ideas about the thought of users with that song.  
 There is a couple of characteristics of implicit feedaback that we need to remember:  
1: There is no negative result: It means, if an user havent’t listened to a song yet, it doesn’t mean this user don’t like the song, it just means “the rate that user like this song is LOW” . LOW, but not 0  
2: The value in implicit feedback represents the confidence that an user might like a song or not.  
3: The data measures must be suitable in different context: for example, the times user listen to a song might not be preffered than the amount of actuall time and user listen to it. Each implicit feedback must have a specific number of weights.

That is for the implicit feedback, now let’s move on to how to make recommendation based on that feedback. There are many ways to do that, like Latent semantic analysis with SVD or Neighborhood mode,… but the one that I focus on is called Alternating Least Squares. Alternating Least Squares is a Matrix Factorization techniques. Just image, a matrix between user and song is like the image below:   
  
You can see, there is a lot of sparse in this matrix, that is why we called its sparse matrix. The row will be for user, and the column is for the songs. The task is fill out all the sparse cell, to have a full matrix, and recommend a song to an user.   
 With Matrix Factorization, the idea is that we separate the matrix into 2 smaller matrix for user and song. Each matrix will have new K number, this one i

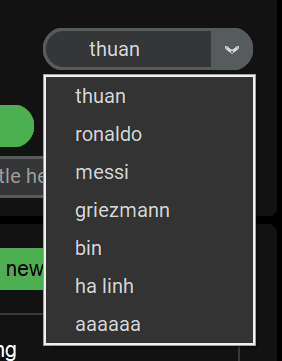


You might wonder, what is the rank of newly created matrixes. The rank of this is the latent factors of this dataset, or, the features of this.   
The aim of this model is to optimize the data again and again to get the value for each song.The effeciency of the model can be calculated by this formula.

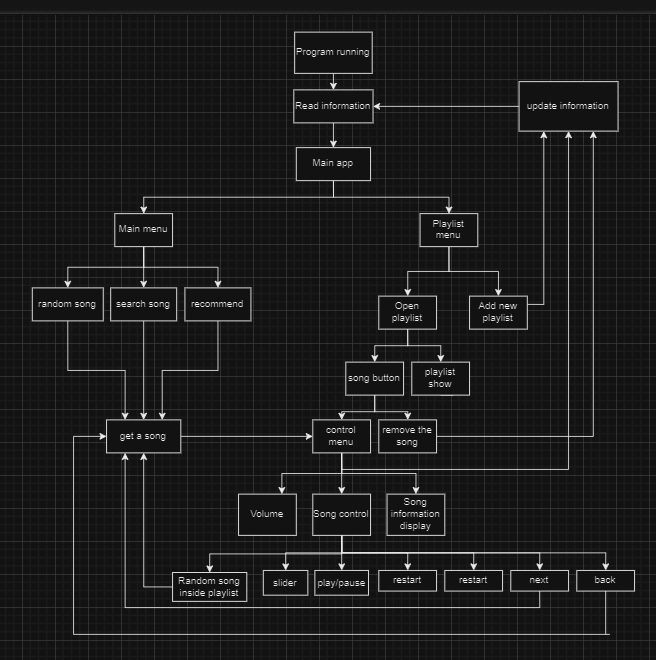
  
In this formula, C is the confidence that an user might like a song or not, while P is the value between 0 and 1 to calculate the preferences of an user with a song. These values is calculated like above:  




r in this case is the recording, like the data, and α is a parameter relates to the scale of the dataset itself.  
 That is all for the theory, now, let apply it into my program:  
 To apply Collaborate Filtering, I need more than 1 user. Because my program is a local, I will create different profiles to apply the recommendation system. Users can add new profile, or remove profile from the program. Each time a song is played, the record for song playing will store who play is.

(htt2)  
 Now we have the dataset, now let’ apply the model. Luckily, there is a powerful library called Implicit that using the theory of Collaborate Filtering to recommend song so I don’t have to code the model from scratch.

With the help of Implicit, things become so simple, all I need to do is to convert the data record into a csr matrix, then fit it into the model, and let the run does it works.  
 And that is how I apply Collaborate Filtering to recommend song for users, and is also the last features of my program.

1. **Limitations**Eventhough my programs is quite complete, there still something that I think can be improved:  
   1. Limited to Spotify. Although spotdl also allows download songs from youtube or other sources, My program is limited to only downloading songs from Spotify. Because of time limitation, I do not extend it, but I think this features is worth updated  
   2. Duplicate download: Another problem my download song is that when download a new song or playlist, even though a song already existted in my playlist, it still download this song again. This make the file duplicate, and I think it is not good for the memory.  
   3. Search function not always precise: Due to the limitaitions of this program, the search function is also limited, so sometimes, it generates the wrong output. In my test cases, 47 out of 50 is the rate, meaning it is not perfect at all
2. **Structured chart**Below is the structured chart of my program ****

**References**

*\/*. (2023, June 16). YouTube. Retrieved November 23, 2023, from https://itectips.com/tectips/solved-find-the-length-of-a-song-with-pygame-python/

*\/*. (2023, June 16). YouTube. Retrieved November 26, 2023, from https://www.youtube.com/watch?v=58OjaDH2FI0&t=1081s

*\/*. (2023, June 16). YouTube. Retrieved November 26, 2023, from https://youtu.be/Q8W2IGiSdhc?si=JMvAGwZvTKuYCq7z

*\/*. (2023, June 16). YouTube. Retrieved November 26, 2023, from https://youtu.be/n3RKsY2H-NE?si=RhrzIZ7sLOnRPbqU

*\/*. (2023, June 16). YouTube. Retrieved November 26, 2023, from https://youtu.be/n3RKsY2H-NE?si=RhrzIZ7sLOnRPbqU

*\/*. (2023, June 16). YouTube. Retrieved November 26, 2023, from https://youtu.be/NI9LXzo0UY0?si=MJfZSGq2XAcKCHES

*Documentation Introduction*. (n.d.). CustomTkinter. Retrieved November 23, 2023, from https://customtkinter.tomschimansky.com/documentation/

*Export Your Spotify Playlist to a CSV File Using Python*. (2022, May 17). Engineering for Data Science. Retrieved November 23, 2023, from https://engineeringfordatascience.com/posts/export\_spotify\_playlist\_to\_csv\_using\_python/

*Finding Similar Music using Matrix Factorization*. (2017, October 2). Ben Frederickson. Retrieved November 26, 2023, from https://www.benfrederickson.com/matrix-factorization/

*Fuzzy String Matching in Python Tutorial*. (n.d.). DataCamp. Retrieved November 23, 2023, from https://www.datacamp.com/tutorial/fuzzy-string-python

*How Spotify knows exactly what you want to listen to*. (2021, July 10). InterSTEM. Retrieved November 26, 2023, from https://www.interstem.us/2021/07/10/how-spotify-knows-exactly-what-you-want-to-listen-to

Huculak, M. (2022, November 18). *How to delete folder with subfolders using commands on Windows 11*. Pureinfotech. Retrieved November 23, 2023, from https://pureinfotech.com/delete-subfolders-folder-commands-on-windows-11/

Lamere, P. (n.d.). Welcome to Spotipy! — spotipy 2.0 documentation. Retrieved November 23, 2023, from https://spotipy.readthedocs.io/en/2.22.1/

*pygame.mixer.music — pygame v2.6.0 documentation*. (n.d.). Pygame. Retrieved November 23, 2023, from https://www.pygame.org/docs/ref/music.html#pygame.mixer.music.set\_pos

Verma, J. (2022, August 3). *Convert time into hours minutes and seconds in Python*. DigitalOcean. Retrieved November 23, 2023, from https://www.digitalocean.com/community/tutorials/python-convert-time-hours-minutes-seconds

Wichmann, N. (2021, September 22). *,*. , - YouTube. Retrieved November 26, 2023, from https://www.youtube.com/playlist?list=PLXLYwvNGGPoUzjKiGvuXm8qeiOYMnFria

Wichmann, N. (2021, September 22). *,*. , - YouTube. Retrieved November 26, 2023, from https://youtu.be/MUB5CiAdSH0?si=2GaBD5WJXv8eMrTc

Yifan Hu, Yehuda Koren, & Chris Volinsky. (n.d.). Collaborative Filtering for Implicit Feedback Datasets.