**Bulgarian Diploma Thesis**

**Music Recommendation System using kNN**

**Eva Diana Maria Conevski, 200083191**



**Student:**   **, Date: 01/12/2022**



*signature*

**Supervisor:**   **, Date: 01/12/2022**

*signature*

**Department of Computer Science, AUBG**

**Blagoevgrad, 2022**



**Declaration template for Senior Project and Diploma Thesis**

**Title**: Music Recommendation System using kNN

**Author**: Eva Diana Maria Conevski

**Abstract:** half-page description

The objective of this project is to create a music recommendation system. The program will determine the users' musical preferences using data analytics. This method makes it possible to anticipate the genre and/or artist(s) a user will enjoy at any given time. Although we all have our favorite bands, it has been acknowledged that we don't always prefer to listen to the same musicians or genres. A listener needs to be surprised occasionally in order to appreciate a new finding. In order to achieve the best results, my application will combine content-based and collaborative filtering. The music collection that will be displayed is drawn from a dataset from Spotify.

**Declaration of authorship:**

“The Senior Project/Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of ….. All sources, used in development, are cited in the text and in the Reference section.”

Author:



# 

# Table of Contents

[Introduction 5](#_Toc120787498)

[Motivation for the Project 6](#_Toc120787499)

[Description and Inspiration of the Application 6](#_Toc120787500)

[Technologies Used 6](#_Toc120787501)

[Specification of the Software Requirements and their Analysis 7](#_Toc120787502)

[Project Definition 7](#_Toc120787503)

[Functional Requirements 8](#_Toc120787504)

[Non-functional Requirements 9](#_Toc120787505)

[UML Use Case and Activity Diagrams 13](#_Toc120787506)

[Design of the Software Solution 18](#_Toc120787507)

[Software Architecture 18](#_Toc120787508)

[Algorithm 27](#_Toc120787509)

[User Interface 37](#_Toc120787510)

[Implementation 42](#_Toc120787511)

[Computing Platform 42](#_Toc120787512)

[Tech Stack 43](#_Toc120787513)

[External Frameworks and Libraries 45](#_Toc120787514)

[Installation Requirements 48](#_Toc120787515)

[Testing 50](#_Toc120787516)

[Dataset 51](#_Toc120787517)

[Algorithm 53](#_Toc120787518)

[Unit Testing 55](#_Toc120787519)

[Conclusion 58](#_Toc120787520)

[References 60](#_Toc120787521)

[Appendix 62](#_Toc120787522)

# 

# Introduction

I have made the decision to create a Music Recommendation System as my Senior Project. These days, recommendation systems are quite common and are utilized to improve the overall quality of the user experience. According to the blog [4] as many as 62% of consumers rated across platforms like Spotify and YouTube among their top sources of music discovery in the year 2020. This can be a valuable representation that a sizeable portion of that discovery is going to be mediated by recommender systems.

These sorts of recommendation algorithms are utilized by a wide variety of businesses, including YouTube, Netflix, Spotify, Deezer, and other streaming services. According to a survey that the blog mentioned, "Made for You" suggestion sessions on Spotify were responsible for the discovery of more than one third of all new musical artists. In a nutshell, recommendation systems provide suggestions for content that individuals might enjoy depending on the individual's own viewing history or the aggregate viewing history of their friends or people that have similar music taste to them.

In addition to being useful for the average person, this is also a major plus for music professionals who rely on recommender systems across platforms such as Spotify and YouTube to amplify their advertising budgets, connect with new audiences, and all-around carry out successful release campaigns. TikTok is another mainstream application that does this, and it is built entirely around an engine for a recommendation system. Because of this, TikTok's algorithms are regarded as being unique, and they are promising the app's creators many more opportunities to grow organically - or, to put it another way, with the assistance of recommender system algorithms according to [4].

My algorithm's main goal is to use statistical analysis and artificial intelligence to make music recommendations that a user will find interesting. The way my recommendation system will differ from others is that it will adopt a hybrid approach that combines item-based and user-based (collaborative) filtering methods.

## Motivation for the Project

As mentioned from the previous paragraphs this idea is highly intriguing to me since it gives a level of automation and simplicity to our lives. We could quickly put on a ready-made playlist with songs that are targeted to our personal likes rather than opening our preferred streaming app and become overwhelmed by the vast music archives while having no clue what to listen to.

## Description and Inspiration of the Application

The objective of my recommendation algorithm is to propose or make predictions about items that a user would like based on their data as well as the data from all of the other users in the database and the characteristics of the music. The content-based and collaborative approaches are the two forms of filtering that the app uses in order to get the best possible results, as stated on the Spotify API website [11] and in the article [5]. From this I have decided to implement both kinds of filtering by combining them and creating a hybrid filter.

The purpose of item-based filtering is to provide a description of the music by analysing the track's content. To determine the similarities between the objects, it examines the characteristics of each one. I will propose an item based on how similar it is to all of the other things in the dataset by assigning a score to the degree to which each item is similar to the others. By analysing user-generated content and assets, collaborative filtering, on the other hand, seeks to provide a description of the track in the context of its relationship to other songs on the site. In the context of user-to-user song recommendations, I will examine the songs that are similar among users and suggest a song that the user hasn't rated based on the ratings of the other users. Accordingly, songs between users will only be suggested if the users are similar and suggest unrated songs. Finally, the output from both filters will be combined to produce a hybrid music recommendation filter.

## Technologies Used

The technologies selected were based on research that was carried out by examining the techniques applied by well-known social networking applications. In order to create scalable web applications, many multinational corporations, including Netflix, PayPal, Gmail, and others, use the Angular framework for their user interface. For this reason, I've chosen to build the front end of my application using Angular. Python, which has a simplified syntax and is simple to use, can be built and run on a variety of operating system platforms for the back-end. Finally, I used R because I have experience with it from class lectures for the mathematical analytics portion of the project. In this paper, under the headings of "Tech stack," and “External framework and libraries" you can find more information on this.

# Specification of the Software Requirements and their Analysis

This section will give a brief project definition and information on the application's software requirements and their analysis. An outline of the functional and non-functional requirements and general problems that the program faces will also be covered.

## Project Definition

The creation of a music recommendation system is the objective of this project. Based on data analytics, the program will identify users' musical tastes. In this approach, it's feasible to predict the genre, artist, or artists the user will like at any given moment. It has been recognized that while we all have our favorite bands, we don't always want to hear the same artists or genres. Sometimes, a listener needs to be surprised and appreciate a new discovery. Therefore, my application will use both content-based and collaborative filtering in order to create a hybrid filter to obtain the best results. The music library that will be shown is taken from a Spotify dataset.

By analyzing the content of each track, item-based filtering aims to provide a description of the music. It looks at each object's traits to ascertain how similar they are to one another. The degree to which each item is similar to the others will be scored, and I will then suggest an item based on how similar it is to everything else in the dataset. Collaborative filtering, on the other hand, aims to provide a description of the track in the context of its relationship to other songs on the website by analyzing user-generated content and assets. In the sense of user-to-user song recommendations, I'll look at the songs that are close among users and make a recommendation for a song that the user hasn't rated depending on the ratings of the other users. This means that users will only suggest unrated songs to each other if they are similar.

After I get the results of the two filters, I will put them together to make my hybrid filter. I was able to combine the results from the two filters by taking the song's genre from the active song track. When the results come back, this genre will be given more weight in the user-based or collaborative filter. For example, if the active song that the user wants suggestions for is from the pop genre, this will give the user-based filtering results more weight by only showing the results that are most like the active song from the pop genre. To create a hybrid music recommendation filter, the output from both filters will be combined in the end.

## Functional Requirements

In the following paragraphs, we will discuss the distinct functions that the Recommendation System performs.

The client-side of the system will be a web-application characterized as a multiuser system with access to shared resources, data persistence system, and dynamic generation of the content displayed to the user. It will feature a dynamic interface that is produced based on how the user interacts with the program, thus its appearance can change at any time. Some features related to this are:

* Search for song – the user will be able to search for songs of their linking from the available songs in the dataset in order to get similar recommendations based on the chosen song.
* Listen to song snipped – the user will be able to listen to a 30 second preview of the song that will be recommended to him.
* Redirect to full information on the track – the user will be able find more information on the recommended tracks by clicking on the Spotify icon of the card, which will navigate him to the Spotify website of the song track.
* Play/Pause the song snipped – the user will be able to play and pause the preview at any given moment.
* Speed of preview – the user will be able to control the speed of the preview
* Download preview – the user will be able to download the preview of the song.
* Full song preview – the user can be navigated to other pages where he can find the full length of the song.
* Adjust volume of preview – the user will be able to adjust the volume of the preview by using the sound controller.

There are several methods for collecting information on users, such as monitoring their interactions, asking them to complete certain activities or fill out forms with personal information, but due to the time constraints of this project, I will be generating the user's information myself.

In the future, one of my goals is to incorporate more capabilities, such as a page for logging in and signing up, which will give users the ability to open their own profiles, rate songs, make playlists, get recommendations from other users, and provide recommendations to other users as well. The filter will have extra features, such as in addition to filtering by a certain parameter, the user will also receive playlist recommendations based on those options.

## Non-functional Requirements

In this part, I'll outline the constraints that will influence how the Recommendation System carries out its various tasks.

### Performance and Scalability

For optimum performance of the application on the client-side, the system should allow generation of web pages dynamically. This function can be performed using Angular own API for loading component dynamically, which allows the updating of concrete page components within the client browser, eliminating the requirement to refresh the entire page. In addition, to enhance the overall quality of the web pages, I made use of the open-source and automated program known as Lighthouse. It will consider a number of different circumstances, like the type of connection, whether the load is coming from a mobile device or a desktop computer, and the sort of material that is being presented. It provides a variety of suggested performance ratings to evaluate the website by basing them on the total of the elements. My goal is to obtain a threshold that is between 0.1 and 1 second, since I believe that a response time that is any longer than that would result in a poor user experience.

The two most crucial performance criteria on the server-side will be accuracy and speed. Since we will prioritize software correctness, the Music Recommender's performance is determined by how accurately it makes suggestions. The system must simultaneously create and provide tailored recommendations to consumers in a timely manner.

### Portability and Compatibility

The program has excellent browser compatibility, which means that it will be able to be translated properly through any browser and operating system that the user selects to make use of. Because it will be incredibly lightweight and will only make use of JavaScript when it is absolutely necessary, the program will be completely light weight and will be compatible with all browsers. A reliable internet connection will be an essential need for successfully operating the program.

### Maintainability, Availability, and Reliability

When it comes to the maintainability of the program, the source code is broken up into components and functions. This will make the process of adding new features to the application much simpler. Because of this, the corrective maintenance will also be improved, and it will be much simpler to locate and rectify any errors that may have been introduced. I have also taken into consideration adaptive maintenance, which means that the system will be able to adapt to changes in the environment in which it functions. These changes may include improvements to the technology or operating system. I have taken both of these things into consideration. Because everything is functional, it will be much simpler to preserve the preventive maintenance. This is because changing parts in one function will be less likely to affect other parts of my code, which will both stop new bugs from appearing in the future and improve the overall quality of the code.

Under typical conditions, the accessibility of the application needs to be kept at an optimum level, and the recommendations ought to be shown in a reasonable timeframe. Since the backend architecture will be functional this will be suitable to execute.

Considering the reliability of the system, it should produce correct results and have a quick reaction time to the shifting behaviors of the users. Additionally, it should continue to function correctly for a substantial period. The likelihood of failing due to one's own actions should be quite minimal. In order to obtain better results, I will construct exceptions that can deal with any type of failure and alert the user to the scope of the potential issue.

### Security

Since the users are be formed automatically in the initial stage of the project, I won't be implementing any security measures at that time because the login and profile features for the users will be added in a later stage of the project. When such capability is finally deployed, maintaining a high level of system security will be one of the primary issues. When a new user registers and creates a profile, the user's password will be hashed in order to protect against brute-force attacks. I will also install password requirements, which will compel the user to select a robust password and will guarantee that the highest level of security is maintained.

### Constraints

In content-based filtering, songs are organized into multiple perspectives depending on the metadata (properties like as tempo, vocalist, mood, etc.) that is already accessible for them. This metadata includes information about the music's composition. In contrast to collaborative filtering, user behavior is not studied in this process, which may result in findings that are less accurate. Because of this, I have decided on the idea of developing a hybrid recommendation system by combining the two distinct types of filtering. One of the limitations of my application may have been the possibility of producing results that are not very accurate because the user data will be automatically generated, which is because there is a shortage of data for training in public domains.

Again, the absence of data information was to blame for the other constraints that I found myself up against. In the original dataset, there was no indication of the genre associated with each song or artist. Because of this, I needed to find a different approach to solving the problem. Since I am working with a Spotify dataset, my first thought was to make a request to the Spotify API for each song. However, this would result in making more than 170 000 requests, which would likely take a very long time or, in the worst-case scenario, get me banned from the service. Instead, I devised a different solution, which involved web scraping the genres from some website. The execution of this might take a considerable amount of time, but other than that, there were no other constraints regarding this.

In general, regardless of the size of the data, item-based and collaborative filtering both have limitations that are caused by a lack of data, and these limitations can be considered limits. The capacity of item-based filtering to locate songs that are similar to one another is limited due to the fact that it is dependent on the characteristics of the music. My dataset has seven features that characterize the music, but this ignores the fact that there are still lots of latent features that cannot be measured, and as a result, the recommender system will not be able to locate songs that humans perceive as being similar to one another.

On the other hand, it is still an improvement over collaborative filtering, which has a wide variety of flaws that make it extremely challenging to recommend songs, particularly for users who are just starting out. The so-called cold start problem, matrix sparsity, and popularity bias or thoughtless rating are some of the challenges that it must overcome. The cold start problem refers to the absence of data on new users or new items; in my case, a new user would not have rated anything yet, and as a result, it would be difficult to find users with preferences that are comparable to this new user because there is nothing to compare their preferences to when comparing this new user to other users. In a similar vein, it would be challenging to recommend a new music because there would not be enough users who had rated that particular song. The existence of a very big matrix, in which the rows and columns, respectively, are composed of users and their song ratings, is the root cause of the matrix sparsity problem. The actual ratings only make up a very small portion of this matrix; the remaining space is blank. One user would have access to the complete catalog of the platform, which could number in the millions of songs, but they would only listen to a small subset of those songs (usually below 1000). Because of this, it is difficult to locate similar users because there are not many songs that share ratings, and in addition to this, the amount of time necessary to compute the recommendation increases, which slows down the system. The last problem emerges from songs that have high ratings and are played frequently by a large number of people, as well as songs that have high ratings for no discernible reason. These songs have the potential to change the suggestions, preventing the recommender system from recommending a varied range of music and inhibiting its capacity to provide correct recommendations.

## UML Use Case and Activity Diagrams

In software engineering, the process of gathering and analyzing requirements takes place before moving on to the design phase of the system's structure and behavior. In this case, the process of analyzing requirements includes thinking about the settings and situations in which the system will be deployed. The use case diagrams are constructed with the assistance of the user scenarios. They illustrate, in a general sense, the particular functionalities that are made available to users of the system. I will be presenting the stages of the use case diagram with the help of the activity diagram. This means that I will visually present the sequence of activities that are performed by the system.

### Use Case Diagram

A use case is essentially a written description of how users will carry out tasks on the website. It describes how a system behaves in response to a request from the perspective of a user. Each use case is represented as a series of easy steps that start with the user's goal and end when that goal is achieved.

Next, I will be discussing the high-level functions and scopes of my system using a Use Case Diagram that can be found further down the page, Figure 1. I will be analyzing the communications that takes place in the dynamic behavior of the system between the user (actor) and the system. I will be modeling the tasks, services and functions required by the system of my application.

Diagram

Description automatically generated

Figure 1: Use case diagram for Music Recommendation System

From the diagram, Figure 1, we can see that the user can search for a song (UC7) from the database (UC10) and set it as a template for the kind of tracks he wants to base the recommendations on (UC8). The recommended songs will be a combination of collaborative (UC11) and item based (UC12) filtering which will be retrieved form the CSV file (UC10). The user will be also able to listen to the searched or recommended tracks by using the music player which will control the flow of the tracks (UC1). The user will be able to play (UC2), pause (UC3), stop (UC4), and also go to the next (UC6) and previous (UC5) track.

|  |  |
| --- | --- |
| **Use Case Title (ID)** | **Description** |
| Music Control (UC1) | Generalization of managing the recommended tracks |
| Play (UC2) | The user can play the track |
| Pause (UC3) | The user can pause the track |
| Stop (UC4) | The user can pause the track that is being played |
| Previous Track (UC5) | The user can navigate to the previous track |
| Next Track (UC6) | The user can navigate to the next track |
| Search Songs (UC7) | The user can search for a track |
| Set Example Song (UC8) | The user can set a track as template for the recommendations |
| Get Recommendation (UC9) | Getting recommendation on given track |
| Get from CSV (UC10) | Selecting the tracks that can be recommended from a CSV file |
| Collaborative (UC11) | Recommending tracks based on collaborative filtering |
| Item-based (UC12) | Recommending tracks according to user-based filtering |

Figure 2. Table with description of Use Case titles

### Activity Diagram

A UML activity diagram helps to give a more in-depth visual representation of a particular use case. It is a behavioral diagram that shows how an activity flow would occur in a system. The sequence of events in a business process can also be represented using UML activity diagrams. In this section I will visually present the sequence of activities that are performed by the system.

The first action is performed when the user picks a sample track from the available tracks, as shown in Figure 3. the start of the action is when the user's input is entered, the system checks to see if there are any previous results. If the condition is met, the system will simply read the CSV file's contents. If the condition is false, the model will process the data. The recommendation scores will then be retrieved and added to the CSV file. The findings will be displayed to the user as the final action.

Diagram

Description automatically generated

Figure 3: Activity Diagram for Music Recommendation System

# Design of the Software Solution

Software designs are the process used by developers to specify the artifacts of software that are intended to achieve goals through the use of components and are subject to limitations. It also entails problem-solving and organizing a solution for the software that is being developed. Additionally, according to what we have studied in software engeneering, software architecture refers to the basic building blocks of a software system and the methodology for developing such structures and systems.

In the next section, I will provide an overview of the software architecture, along with ULM component diagrams for the software components of the software architecture, a description of the algorithms that were utilized and their performance, and an overview of the user interface.

## Software Architecture

When it comes to the application's architecture, it is composed of three distinct layers that are separated from one another. My decision to segment my applications into separate parts was motivated by a number of considerations, including the following: the enhancement of the functionality of individual components, the simplification of future development and addition of enhancements, the facilitation of the detection of errors, and so on.

In the previous paragraph, I discussed three layers: the presentation layer, the application layer, and the business layer. The capabilities that are provided by each of these directories to the application serve as the criterion for the division of the architecture that contains them into their respective categories. Each file contains an implementation of a particular feature that is helpful for the application as a whole. In addition to this, on the basis of this, the application may be segmented and illustrated through the use of the layer architecture method.

The files referred to as CSS, HTML, and TypeScript are all included in the presentation layer. Since I am utilizing Angular, the application's components are utilizing the modular structure. This means that the code is organized in various buckets, which can include components, directives, pipes, or services. Since I am utilizing Angular, the modular structure is being utilized by the application's components. Because modules make it simple to organize application functionality by separating it into features and chunks that can be reused, I decided to make use of this framework in addition to its simplified MVC pattern structure. Modules divide application functionality into features. All the code from the Angular directive is making the user interface, which is how the users will interact with the application.

Next in the application layer, I have a mix of front-end and back-end functionalities. For example, basic functions like displaying the recommendations for a song are done in this layer. For displaying all of the recommended songs, there is a separate component that runs each of the operation. The component calls a service that is part of the front-end part of the application, which makes a get request to the Spotify API. If the user chooses another song, accordingly recommendations will be shown for the new song. In the service is included all the logic that doesn’t align with either the presentation or business layer, like some code definitions and most basic functions of the developed application.

A great number of these functionalities can be found in the back end. For instance, there are a significant number of functions that are solely devoted to the process of cleaning the data and bringing it to the final state required for making predictions. There were also functionalities for creating users, which are used for making additional predictions on the song that will be selected by the user. These predictions are based on the fact that there were also functionalities for creating users. And finally, there were the functionalities for receiving the additional data that was lacking in the initial dataset. This includes everything that is connected to the musical genre that the song belongs to.

Last but not least, the business layer is exemplified by the primary algorithm that drives this project and generates recommendations based on the song that is being played. Since I do not make use of a database to store the information but rather a csv file, the data layer is considered to be a component of the business layer in my particular setup. In a nutshell, this layer is responsible for operating the application's entire business logic. These are similar to a collection of rules because they instruct the application on how to retrieve and process data based on certain standards. This layer is essentially responsible for determining the entire behavior of the application. In the event that there are no recommendations previously stored for a specific song, it will run a series of functions that are responsible for calculating the similarity score and providing a list of recommendations to the layers that were discussed earlier. In addition, it stores a copy of that information, so that it won't have to be computed again in the future, which is a huge time saver. On the other hand, if the information is saved, it will simply obtain it from the csv file. In a nutshell, this layer is responsible for carrying out a series of actions, and once those actions have been completed, it informs the application of the next step.

### Functional Block Diagram

A functional architecture is a collection of functions and their subfunctions that specify how the system will convert input flows into output flows in order to fulfill its purpose [16]. Software architecture can benefit greatly from the functional programming principles, and I used the following ones in my design:

* Functions being standalone values – meaning that they can be assigned to variables, saved in lists, used as parameters in other functions, returned as results of other functions, and so on. Each workflow is an example of a feature, use case, scenario, or story, and they all represent a unit of functionality.
* Building systems with composition – which means combining two simple functions into a single one by simply linking the output of one function to the input of another function. The end result is another function that can serve as a foundation for additional compositions.
* Having similar workflow structure – data is read, business decisions are made, and the data is transformed as necessary; finally, any newly created data or events are output at the other end of the function. Every one of these sub-steps can be broken down and analyzed as its own individual function. Even though there is the possibility of branching and other types of complexity coming into play, the data will always flow in one direction, even as workflows grow larger and more complicated. If we find that multiple workflows require the exact same functionality, we can save time by defining that functionality only once as a subfunction and then reusing it as a shared step in the workflows that call for it.
* Using pure functions – meaning that there will always be the same type of output regardless of what the input is. The fact that this is so simple to test and comprehend is one of the many advantages offered by it.

Workflows are created as independent units with only the functionality they require, but we can still take advantage of the reuse and componentization advantages when we need to. I have chosen to use a functional workflow rather than an object-oriented workflow for the reasons mentioned above. This is also evident visually in the Figure 4.

Diagram

Description automatically generated with medium confidence

Figure 4. Visual representation of a functional-style workflow vs. object-oriented workflow

In the following section, I'll present a visual representation of my software architecture and explain how the aforementioned principles helped me. The process of cleaning the raw data and bringing it up to par with the filtering algorithm's implementation will also be covered in the paragraph after this one. This will serve as a manual for obtaining the final clean dataset with all the records from the raw data obtained from Kaggle [9].

In the next paragraph I will be explain the processes and decisions made from my functions as seen in Figure 5. The chronological order of these functions has boughten the dataset ready for filtering:

* find\_duplicates: takes in the whole dataset and checks if there are duplicate records according to the track name, stores those indices in an array.
* delete\_duplicates: takes in the array of duplicate indices and removes them from the dataset
* remove\_special\_chars: removes the record that contain special characters in the song name.
* reset\_indices: takes in the dataset and resets the index of the track
* get\_unique\_artists: takes in the whole dataset and appends the unique artists to an array
* get\_first\_artist: returns the first artist form a track
* get\_scraped\_genre: takes in the array of unique artists and scrapes the web for the artist genre
* get\_general\_genre: scrapes the web for genres and sub-genres
* get\_genres: gets the scraped genres and appends them to a dictionary along with the artist
* set\_genre: sets the genres of the artists in the dataset
* generalize\_genre: takes in the whole dataset and the general genres, then sets the sub-genre to the generalized genre.

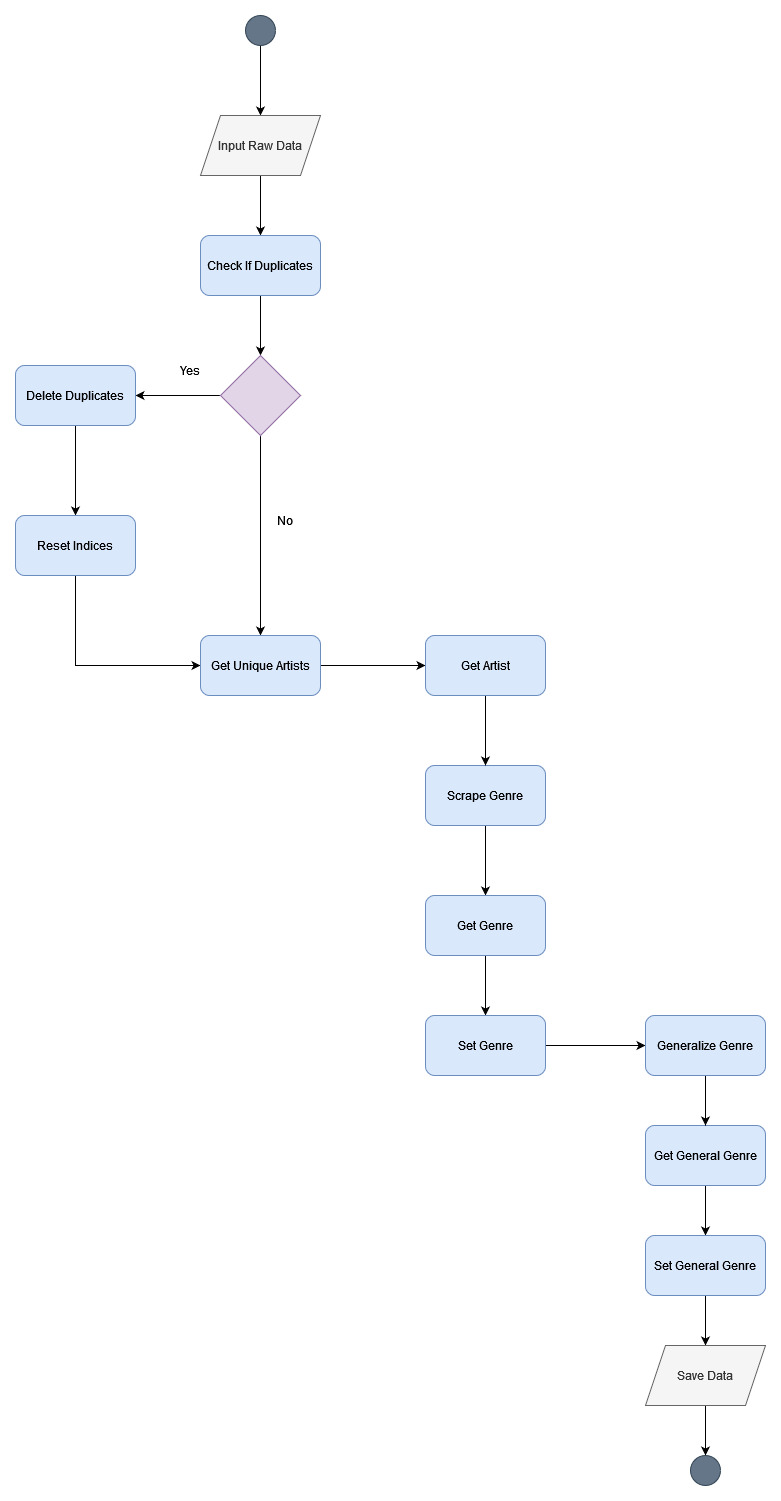


Figure 5. Visualizing the functional workflow for preparing the data

Next, I'll proceed to describe the steps taken by each function in order to obtain the filtering results shown in Figure 6. The filters are divided into item-based and user-based categories as can be seen in the figure, and they are ultimately combined to create a hybrid filter. In the Algorithm section, more information regarding the filters will be provided. Just the architecture and how the functions work will be discussed here:

1. Item-based filtering flow:

* get\_feature\_list: takes in the dataset and gets the values of the features columns only and converts them to floats (this is done because when the values are stored in the dataset they are strings), returns the values in a list.
* eucledian\_distance: takes in two feature lists and returns the Euclidean distance between them.
* kNN: takes in the index of the song for which we are determining its similarity, the dataset from which we will determine the song, and the number of songs that are its nearest neighbors in order to return an appropriate similarity metric.

1. User-based filtering flow:

* User class: generating a unique UserID, a list of genre preferences and a (song: ratings) dictionary.
* get\_music\_preferences: takes in the list of generalized genres and randomly assigns 6 genres to each user.
* rate\_song: takes in the whole dataset and genre preferences, if the song genre is in the genre preferences array it assigns a integer from 6-10, otherwise from 1-5.
* get\_song\_ratings: it takes in the data that contains all the songs with all the features, n which is the number of records that the dictionary will contain and the genre preferences list. Then it takes random songs from the dataset and returns a dictionary of random (song: ratings) pairs
* generate\_users: generates a number (n) of users with genre preferences and random chosen songs they have listened to along with their ratings and an (m) number of song ratings.
* create\_user\_rating\_matrix: creates a matrix from the generated users and their song ratings. In each row there are the generated users with the userID and each column represents the score given to a song.
* reset\_indices\_user: takes in the dataset and resets each index for the users
* find\_intersection: takes in two users and finds the intersection of rated songs between them.
* get\_user\_ratings: gets the rating that the user has assigned to each song.
* manhattan\_distance\_users: finds the similarity distance between each user from the intersection of user ratings
* normalize\_distance:
* get\_user\_song\_recommendations: takes in a list of similar users to the active user at a given index. Returns the top 5 songs as recommendations from the songs that the active user has not listened to.
* kNN\_users: takes in the index of the song we are getting the similarity for, dataset to find the song from index and number if nearest neighbors to return according similarity metric

1. Hybrid filtering flow:

* hybrid\_filtering: combines the item-based with the user-based filtering through the genre of the active song. It takes in the parameters of the item-based and user-based k-NN algorithm along with the genre of the active song.

Graphical user interface

Description automatically generated

Figure 6. Visualizing the functional workflow of the filters

## Algorithm

As previously stated, my algorithm is going to be a combination of item-based filtering and collaborative (user-based) filtering.

### Item-based filtering

The item-based filtering uses the features of each track to find similar tracks. By assigning a score to how similar each track is, we can recommend a track based on how similar it is to all other track in the dataset. The dataset that I have for my project, I was able to find on Kaggle, which is an online forum for practitioners of machine learning and data scientists. Although I had looked at other datasets, the convenience of this one made it the best choice for my project. This dataset was the most settable for my project since it provided features that I was able to utilize both for the back-end logarithm and for the front-end of my project.

From it I have extracted only the valuable features in order to avoid overfitting because it can lead to an increased model complexity. In general, feature selection may be helpful in terms of making models more interpretable, ensuring that models actually generalize rather than overfit, and speeding up the building of models when expensive algorithms are being used. In addition, feature selection may help ensure that models do not overfit specific data. In addition to that, removing the noisy features will help with memory, which will help reduce the computational cost, and it will also help improve the accuracy of my model.

There were some features that were redundant and wouldn’t be important for the prediction that I have taken off like: duration, explicit, key, and popularity. For example, the duration of the tracks, weather a song is explicit or not, the key to the song, and the popularity are quite irrelevant when comparing tracks.

Next, I was able to do this by determining which characteristics have a strong correlation with each other and then removing those characteristics that don't have a strong correlation from my prediction model while keeping the highly correlated characteristics. The term "correlation" refers to a statistical method that analyzes the linear connection between two or more independent variables. Creating two correlation matrices consisting of all of the features was the method that I used to accomplish this. For the purpose of calculating the correlation matrix, I made use of my familiarity with R by making use of the corrplot library. I discovered that there are a few feature selection methods after doing some research. In the end, I made the decision to use the schema I discovered in the article [2] for my project. Here, the Spearman's and Pearson's rank correlation coefficients were suggested based on the input and output variables of my project. Correlograms which can be seen in Figure 7 and Figure 8, were created by me in order to facilitate a clearer understanding of the results.

Chart

Description automatically generated

Figure 7: Visualization of the correlation matrix with Pearson method

Chart

Description automatically generated with medium confidence

Figure 8: Visualization of the correlation matrix with Spearman method

From these characteristics, we can deduce that the outcomes of the two different correlation methods were comparable. The values in the correlograms for loudness, liveness, and tempo are represented by a lighter color, which indicates that they are closer to zero than the values for the other features. This indicates that the correlation between these three features is not as strong as the correlation between the other features.

In order to select the features, I have set the threshold limit to be greater than 0.3. This indicates that in order to select the features, I will only take those features that have a strong or moderate correlation. My research revealed that liveness, speechiness, and tempo have a modest relationship to the other characteristics, as shown in the figure to the right. Because none of the values have a correlation of more than 0.2 or even close to that number, I have made the decision to not use those features.

After the dataset was normalized, I utilized the k-NN algorithm in conjunction with the Euclidean distance to locate the songs that are the most comparable to the sample track. The result that is returned by the algorithm is a number that ranges from zero to one. If the result is closer to zero, then the tracks are closer to one another and are more similar. On the other hand, if the result is closer to one, then the songs are quite dissimilar to one another.

### Collaborative based filtering

Collaborative filtering will utilize the same approach as the item-based filtering. One of the differences will be that instead of using Euclidean distance for calculating the distance between the dimensions, I will be using Manhattan distance. This is because the dimensions of this matrix will be quite large in comparison to the item-based matrix. The dimension of my matrix will be where n represents the number of users, and m will represent the rating of songs. Next, I will be able to calculate how similar each user is according to the songs that they have rated. After this I will be able to recommend content that users with similar taste in songs like.

In order to accomplish this, I used the web scraping to retrieve supplementary data for each song. The extra information that I have provided for each song is going to be the genre that the artist of the song is classified under and applied this information to each song. This will make it easier for me to organize the music and provide a more precise prediction when suggesting a song to the user. I have collected the genre for each song by using the Python library Beautiful Soup. The library works by pulling data out of HTML and XML files. The website that I have used for pulling this data is Wikipedia which will be linked in the appendix [8]. In the end I appended all the retrieved data accordingly to each song.

I discovered that I have many different genre categories after removing every genre from the song; I was left with about 50 genres. I made the decision to categorize the data at this point. I repeated this by web scraping [10], which was the website I used. The names of the genres and subgenres were the information I took from the website. Then I created a function to create a dictionary, with the genres serving as the keys and an array containing every subgenre that falls under that genre as the value. After completing that, I went and assigned the artist's general genre.

After that, I have allotted six categories of music to each user. The user will have musical subgenres that they appreciate and take pleasure in listening to, in addition to those that they despise and do not wish to hear. After that, I have selected a section of the data at random and assign scores to it based on the genres that the people prefer. If the user is a fan of the musical style from which the song was chosen, then the rating that is automatically created for the song will range from 6 to 10. On the other hand, if the user does not enjoy this genre, the ratings that are given for the song will range from one to five.

I made a user-ratings matrix after collecting information about each user, which included the songs that user has listened to and the ratings that have been assigned to those songs. The users, who were organized into rows, and the songs, which were arranged in columns made up the components of the user-ratings matrix. The implementation of the algorithm and the utilization of the Manhattan distance made it very efficient, despite the fact that computations done on such a matrix (n x m) can cost a lot of computation power. The algorithm operates in the same manner as that used for item-based filtering; however, in addition to making use of the Manhattan distance, it computes the degree to which users are alike.

In the end, after finding other users who were similar to the active user, I developed a function that would recommend five songs to the active user that were not previously heard or rated by the user. These songs came from the similar users. I have made the decision to assign weights to the similar users who have been assigned to the active user in order to ensure that the recommendations I provide regarding what the user wants to hear are as accurate as possible. Because of this, the results will be impacted less by the fifth user, who shares the fewest similarities with the active user, in comparison to the first user, who shares the most similarities with the active user. I have accomplished this by first multiplying the rating that a user with similar tastes as the active user gives to a song by the similarity score that exists between that user and the active user, and then dividing that value by the similarity score. After running these computations, I will have an estimate of the rating that an active user will give to a song. After everything was said and done, I put all of the predicted ratings into descending order and listed the top five songs that he rated as a direct result of the recommendations. The implementation of prioritizing the recommendations of a user in comparison to another user by assigning a weight to the similarity score can be seen in Figure 9.

Text

Description automatically generated

Figure 9: Implementation of distributing weights to users from user similarity score

### Hybrid filter

I will combine the outcomes of the two filters after obtaining them to create my hybrid filter. By taking the song's genre from the active song track, I was able to combine the results from the two filters. When obtaining the results from it, this genre will then be used in the user-based or collaborative filter by giving it more weight. For instance, if the active song for which the user is seeking recommendations is from the pop genre, this will give the results of the user-based filtering more weight by returning only the results with the highest similarity to the active song from the pop genre. The results of those filters will be added to the results of the item-based filtering, and the hybrid filter will function as a result of this combination. The implementation of the final outcome of the algorithm can be seen in Figure 10.

Text

Description automatically generated

Figure 10: Implementation of hybrid filter

### Model

Different approaches to selecting important data are provided by models including content-based, item-based, and collaborative filtering in order to provide precise suggestions. I also have the mathematical underpinnings of recommender systems, which are being developed concurrently with the recommender model. I'll be employing a vector space-based mathematical model to create the recommendation system. This will estimate which object best fits the user profile segment because of the quantitative data modeling.

Items are represented in the vector space model by a vector of characteristics, often words or concepts that are expressed quantitatively as frequencies, relevance percentages, or probabilities. These systems use the multidimensional characteristics of vector spaces to represent substitutes for an object. For the surrogates to be used in a multi-dimensional context, they must be numerically performable [7]. As implied by the model's name, domain entity substitutes are vectors with each dimension standing in for a different attribute of an object. The type of the attribute should be real numbers. User profiles are examples of features that have previously been employed to meet specific needs When objects' surrogates are discovered to be comparable to those shown on a user profile, it is presumed that they have a utility value proportionate to the degree of resemblance they share [7].

Model-based solutions go a step further by developing a comprehensive template for recommender systems. Each model is responsible for defining the item surrogates, as well as the development and maintenance of the profile. After that, the algorithms that are employed for matching purposes can be picked analytically, based on the behavior that was wanted for the system. For this case my model I have employed the K-Nearest Neighbor algorithm.

### K-Nearest Neighbor

The k-nearest neighbor (k-NN) algorithm is a type of supervised machine learning that requires labeled input data in order to train a function that, when presented with additional unlabeled data, generates a suitable output [1].  In my particular scenario, I will apply this approach in order to resolve a challenge involving categorization. An unidentified item is assigned the label that appears the most frequently among its k closest neighbors when using this approach. If a neighbor has the shortest distance between them in feature space as measured by the cosine angle, then that neighbor is regarded as being the nearest neighbor. The outcome of a k-nearest neighbor classification is a class member. The classification of an item is determined by the majority vote of its neighbors, with the object being placed in the category that is the most prevalent among its k closest neighbors (k is a positive integer, typically small). In the event when k equals one, the item in question is merely designated as belonging to the category of its one closest neighbor.

I have implemented the k-NN algorithm in both the item-based and user-based (collaborative) filtering. The integration of the algorithm is different in both of the filters. Figure 11 and Figure 12 provides a clearer view of this.

In the item-based filtering illustrated in Figure 11, when I have determined the distances that exist between the items, I save them in a temporary variable, and the algorithm then selects the k songs that are the most comparable to one another. Because the first value in x[0] represents the currently playing song, the code snippet enables us to deduce that the for loop begins at 1, as expected. In the end, the k tracks that are most comparable to the result are added to it.

Text

Description automatically generated

Figure 11: Implementation of item-based k-NN

On the other hand, when it comes to sorting the distances between individuals who are similar, the user-based (collaborative) filtering method, which can be seen in Figure 12, follows the exact same processes as the item-based filtering method. However, after I have sorted them, I will normalize the distances because the data consists of scales of varying lengths and the technique that is being used (k-NN) does not make any assumptions about the distribution of the data. Following that, I’m rounding the gotten distances and appending them to the result variable.

Text

Description automatically generated

Figure 12: Implementation of User-based k-NN

#### Drawbacks

Although the k-NN approach is theoretically straightforward to implement, it has a high computing cost when predicting over a set of unrated things given a set of rated items. The performance of the method suffers as a result of the high cost of computing the distance between the new point and each old point in large datasets, which brings down overall performance [1].

When dealing with data that has a high number of dimensions, the k-NN algorithm is not very effective since it gets more difficult for the algorithm to compute the distance in each dimension as the number of dimensions increases. Therefore, in my project I utilized Euclidean and Manhattan distance to resolve this issue.

### Euclidean Distance

For calculating the neighbors’ shortest distance, I used the Euclidean distance formula. In mathematics, the length of a line segment connecting two points in Euclidean space is known as the Euclidean distance. The Pythagorean theorem is be used to determine it from the points' Cartesian coordinates in n-dimensional Euclidean space, and the formula will be presented as below:

From the formula above we have the two points in Euclidean n-space, p and q, the Euclidean vectors, starting from the initial point of the n-space and the n-space.

In the item-based filtering for calculating the Euclidean distance I used 6 dimensions which will be important for estimating which songs are the closest to each other. The initial dataset included more characteristics for each song, but I have narrowed it down to the ones that fit the best in order to avoid the issue of overfitting, which can be problematic. The dimensions that I have chosen are: acousticness, danceability, energy, instrumentalness, loudness and valence. My algorithm will store the dimensions in two different lists of tuples. The first list will contain the sample track with all of the dimensions as well as additional features such as the track's name, id, and artist. The second list will contain the next sample track from all of the songs list, and it will then return the Euclidean distance between the two songs.

### Manhattan Distance

In light of the fact that the dimensions of the matrix for the collaborative filtering will be quite a bit larger in comparison to the matrix for the item-based filtering, I have made the decision to make use of the Manhattan distance. Both Manhattan and Euclidean are algorithms from the same family of distances, but according to the article [6] Manhattan distance is preferred over the Euclidean distance metric as the dimension of the data increases. This occurs due to the so-called the ‘curse of dimensionality’.

Manhattan distance or also know as city block distance represents the sum of the absolute differences between the opposite values in the vectors. The formula will be represented below:

## User Interface

My project's user interface is a very important component because it will be used to provide a better visualization of both the project as a whole and the results that have been obtained from the recommendation system.

### Design and User Guide

When the user opens the interface, they can navigate to the search icon shown in Figure 13, and when they click on it, it expands so they can type. Users will be able to look for a song that is contained in my database by using the search bar that is provided. As shown in Figure 14, when a user begins typing the name of a song into the search bar, a list of songs that contain those prefixes will appear below the search results. If the user searches for a song that is not included in the database, a message will appear informing them that the database does not contain any tracks that match their criteria.

Graphical user interface

Description automatically generated with low confidence

Figure 13: Visual representation of the main UI page

Graphical user interface

Description automatically generated with medium confidence

Figure 14: Visual representation of the search field

If the user is successful in locating the song that he or she prefers, then after clicking on the song from the dropdown of the search bar, recommendations will be generated that are based on that song, as shown in Figure 15. The user will be able to see additional information, such as the name of the song, the album of the song, a pin from the song, and in addition, I will be adding the similarity score to the song that the user has chosen. A card will be generated for each recommended song, and within that card, the user will be able to see additional information. This card will also include a music player, through which the user will be able to listen to a snippet of the recommended song. If the user wants to hear the entire track, full links to the song will be provided to various platforms so that they can listen to the song in its entirety.

Graphical user interface, application

Description automatically generated

Figure 15: Visual representation of the generated recommendations

The user will be able to alter the tempo of the song within the music player by clicking on the three dots that are located within the music player, which will cause a dropdown menu to appear. The user is then given the option to select a speed ranging from 0.25 to 2 when the playback speed button is clicked. If the user wants to download the music snippet, there is an additional option available for them to choose from in the dropdown menu. In addition, there is a sound icon, which provides the user with the ability to control the volume the song.

In addition, I have also included a link that, when clicked, will take the user to the official Spotify site. Once there, the user will be able to listen to the entire song, read the lyrics to the song, and find additional information relating to that song, such as the album and other songs that are included in that album, as shown in Figure 16. When the user clicks on the Spotify icon that will be located in the top right corner of the card, they will have the ability to do this. I have also added a new figure directly below that icon, which will display the similarity score that has been calculated between the currently selected song in the card and the song that is being actively searched for.A screenshot of a computer

Description automatically generated with medium confidence

Figure 16: Visual representation of navigation to the official Spotify site for chosen track.

### Web API endpoint

I have made use of the Spotify Application Programming Interface in order to obtain the information required to display the recommendation cards. Get Track was the endpoint that I used to retrieve the information that I needed from the Tracks submenu. The response is constructed on the basis of a straightforward REST principle, in which the endpoint returns a JSON object. In order to successfully make a GET request to the endpoint, I was required to supply a parameter with the Spotify ID for the track. I was able to gather the following information based on the response:

|  |  |  |
| --- | --- | --- |
| **Key Name** | **Type** | **Description** |
| href | string | A link to the Web API endpoint providing full details of the album |
| images | array of objects | The cover for the album in various sizes, widest first |
| name | string | The name of the album. In case of an album takedown, the value may be an empty string. |
| uri | string | The Spotify URI for the album |
| artists | array of objects | The artists of the album. Each artist object includes a link in href to more detailed information about the artist. |
| preview\_url | string | A link to a 30 second preview (MP3 format) of the track. Can be null |

Figure 17: References of web API endpoint

As can be seen in the table that is located above, the preview URL might be null. If this is the case, then some of the music controllers for the tracks will be disabled, indicating that the track will not be able to play.

# Implementation

The application's implementation is a method that is methodically structured in order to effectively develop and satisfy the goals and needs of the users. This is done through the implementation of the application. This stage of the development process represents the phase during which I have implemented code in order to accomplish the goals I have set for the output results of my application. In this section, I have taken into consideration which platforms the software is going to be used on, as well as which programming languages, frameworks, libraries, etc. are to be used in order to arrive at the finished product of the application. I have also taken into consideration which platforms the software is going to be used on. These factors are going to be discussed in the following section of the paper, along with the methodology that will be used to analyze them.

## Computing Platform

It is necessary for users to satisfy certain requirements before the application can be used and run properly. These requirements are in place to help support the application and ensure that users have the best possible experience using it. This application was created on a computing platform that runs Mac OS, but it was conceived with the goal of being usable and enjoyable across all available platforms. The web-based application is intended to be used by all different kinds of devices that have access to the internet. However, this application was developed only for desktop, and it will require some work in order to make it suitable and responsive for different kinds of phone devices. On the other hand, it is fully responsive for all different kinds of desktop and tablet devices. Because the only requirement for using this application is a reliable internet connection, all desktop users can have the same experience and use it regardless of the computing platform they use or the characteristics of their individual devices.

The web-based application or software is accessed and utilized through the use of a web browser while connected to the internet. There is nothing for the user to install, download, or worry about in terms of software updates. They make it possible for users to communicate with them through a remote server using the interface of a web browser. In light of the information presented above, I came to the conclusion that the web-based platform would be the best fit for the idea of an application that I was going to develop. The aforementioned facts are examples of some of the benefits that the web-based application offers, but they also contain a great number of other advantages. Portability, security, accessibility across a variety of devices, customization for the same devices, integration with other systems, increased flexibility and scalability, etc. are some of the additional key advantages that come along with using applications of this type. Other advantages include:

When it comes to the safety of the same, they are using they are run on dedicated servers which are constantly moderated by professional administrators. This means that if there is a possibility that an error will occur, it will be quickly noticed and dealt with by the administrators.

## Tech Stack

As was mentioned in the previous section, I have selected the programming languages that will be the most appropriate to be used when the implementation phase of the development process begins. Three programming languages (Python, R, and TypeScript) were used for this application, along with CSS and HTML via the Angular framework.

### Python

Python is a high-level programming language that is object-oriented. Python also has dynamic semantics. Because of its high-level data structures that are built in, as well as dynamic typing and dynamic binding. I went with Python as my programming language of choice because it is a general-purpose language, which means that it can be adapted to run a wide range of applications and does not concentrate on solving any one problem in particular.

In spite of the fact that Python is an object-oriented programming language, I am using it for functional architecture in the application that I am developing. Through the use of functional programming, I was able to break down the challenge of providing appropriate recommendations into a series of functions. As a result of this, I have segmented the implementation of the problem that I had been working on into smaller pieces, each of which I have placed into its own Python file and given the name of a particular component of the problem. I have, for instance, created separate files that hold the item-based, user-based (collaborative), and hybrid filtering methods. Because I took this approach to developing my program, it is modular. It is much simpler to specify and write a small function that does one thing with this than it is to write a large function that performs a complicated transformation with this. Functions with fewer steps are also simpler to read and more straightforward to validate. Because of this, the process of debugging and testing is becoming significantly less difficult, in general, because the functions are becoming smaller and more clearly specified.

### R

R provides a supportive environment for statistical computing and design, in addition to providing a wide selection of libraries dealing with statistical topics. R has enabled me to perform statistical computing and graphics, which has helped me gain a deeper comprehension of my data and work to my advantage using the information I have gathered. In order to perform analysis on my data, I have made use of the extensive collection of intermediate tools, all of which are simple to comprehend.

This programming language was used to find correlations between the song features in order to find the features that are most appropriate for carrying out valid results. The results from the research were presented in the project as figures.

### TypeScript

TypeScript is a superset of typed JavaScript (which is optional) that can assist in the construction and management of extensive JavaScript projects. It is possible to think of it as JavaScript, but with additional features such as static typing, compilation, and object-oriented programming. The purpose of this paper does not extend to the explanation of all of the features. Nevertheless, I will discuss the ones that I thought were helpful and incorporated into my project. Literal types are one of the capabilities of this programming language that I've put to use in the front-end development work that I performed. When implementing functions, these were found to be helpful because they could infer the type of the parameter that the function takes. When it came time to implement the code, I was able to locate errors much more quickly thanks to this.

## External Frameworks and Libraries

In this section, I will discuss the additional frameworks and libraries that I have used, as well as the explanation behind my selection and use of those libraries.

### Beautiful Soup

Python software called Beautiful Soup is used to parse HTML and XML documents. It is a helpful library for web scraping because it generates a parse tree for parsed pages that can be used to extract data from HTML. With the library I will be scraping the webpage [10] the genres for each of the songs since that feature was missing in the original dataset. The genres of each song were required for collaborative filtering because there are no records for user rating data, so I had to generate my own data by assigning random genres to users and rating tracks based on that.

After scraping the genres for each track, I had data that consisted of a lot of sub-genres, which I had to categorize into the most popular genres. I have done this by scraping [10], where I scraped the titles or the most popular genres in this case and all the <a> tags which all the sub-genres were stored that corresponded with the title of the main genre.

### RxJS

RxJS is a library that uses observable sequences to facilitate the composition of asynchronous and event-based programs. It allows for the handling of asynchronous events as collections by providing one core type, which is the Observable, as well as satellite types (Observer, Schedulers, and Subjects) and operators that are inspired by Array methods (map, filter, reduce, every, etc.).

The following is a list of the essential concepts in RxJS that I have used to solve asynchronous event management:

* Subject: is synonymous with an EventEmitter, and it is the only method for multicasting a value or event to multiple Observers at the same time.
* Subscription: represents the execution of an Observable and is most useful for stopping the execution of the Observable.
* Observable: is a concept that stands for the idea of an invokable collection of potential values or events in the future.
* Operators: are pure functions that make it possible to deal with collections using a functional programming style. Examples of operators include map, filter, concat, and reduce, among others.
* Observer: is a collection of callbacks that is aware of how to listen to values that are being delivered by an observable.

Observables that I have used:

* fromEvent: generates and sends out events of a particular type that originate from the specified event target. I was able to use this after taking the information from the search bar.
* debounceTime: emits a notification from the source Observable only after a certain amount of time has elapsed since the last source emission and there hasn't been another one. In order to obtain the filtered inputs for the search function, I have utilized this in the past to wait for a specific amount of time. This was done to prevent the user's search function from becoming overwhelmed with every character that they type in.
* distinctUntilChanged: returns an Observable that, if the values that were pushed by the source Observable can be distinguished from the most recent value that the result Observable emitted, then that result Observable will emit those values. This is something that I have implemented in the search filter so that whenever the input field is modified, it will produce a new value.
* tap: utilized to carry out additional actions in response to notifications received from the source observable.
* map: Applies a given project function to each value emitted by the source Observable and emits the resulting values as an Observable. I have used this function in order to project results gotten from the backend respons.

On the official RxJS page [14], you can find additional details regarding the library, as well as the events and methods that are utilized.

### Corrplot

A graphical representation of a correlation matrix, confidence interval, or general matrix can be obtained through the use of the corrplot package. In addition to that, it has a number of algorithms that can reorder matrices. In addition, corrplot is excellent at attending to the details, such as the selection of color, text labels, color labels, and layout, among other things.

A visual exploratory tool on correlation matrices that supports automatic variable reordering is made available by the corrplot R package. This package, which is part of the R programming language, is used to assist in the discovery of hidden patterns among variables. For the item-based filtering, I have made use of this to provide a visual representation of the correlation matrix.

### Angular

Angular is a web development framework that has gained a lot of popularity due to its ability to provide rich user experiences, quick responsiveness, and maintainability of code. It is a JavaScript-based framework that is open-source and written in TypeScript. The process of developing websites is simplified significantly by the numerous core features that Angular provides. The following are some examples of things that were very helpful in my development process:

* Two-way Data Binding: data between the model and view components are automatically synchronized with one another. This has made it possible for me to effortlessly create interactive applications without the need to manually push and pull the information that I want to pass along.
* High-speed performance: high-speed page loading times and excellent rendering capabilities are just two of the ways in which Angular takes advantage of hand-written code to take it to the next level.
* Low code framework: Because of this, I was able to steer clear of the laborious process of writing separate code to link MVC layers. In addition, my directives are separated from the app code, which cut down significantly on the amount of time I spent developing.

The framework has also given me a lot of architectural benefits at the business-level. The effective design architecture, code reuse, and quicker end-to-end development are some of the main advantages that I have used. With the help of all the advantages I've listed for this framework, I was able to implement a very efficient searching method that takes all the song names from the dataset, which totals about 170,000 songs, and returns all the songs that match the input characters in the user's specified order in a matter of milliseconds.

## Installation Requirements

In the following passages, I will talk about the necessary installation requirements that must be met for the application to run properly in any environment

### Spotify API

I have used the Spotify API to obtain the data required to display the recommendation cards. I had to register for a free account in order to use the Spotify API for development purposes. In order to obtain the keys for the Client ID and Client Secret after creating the account, I went to the dashboard menu of [11] and added the app I intended to create. In order to access the information from my application I have edited the configuration settings of the Spotify dashboard and added my page’s URL as a redirect URL (<http://localhost:4200/> since I’m running the application on a local enviroment). To be able to make API requests after creating those, I generated a client token. On [12], there is additional information regarding token generation. and Figure 1 of the appendix contains a more thorough explanation of the authentication code flow. After that, I set up a GET request to get data from the endpoint of the API, more information about the endpoint can be found on [13]. From this endpoint, I am able to retrieve all of the track's details, including the song's title, artist, and album, as well as a URL for the song's snippet, which I have integrated into the music player.

### Angular

* The Angular Command Line Interface (CLI) tool must be utilized in order to successfully install and configure the local environment and workspace prior to running the Angular application. The following are some of the prerequisites that must be met before installing Angular in your local environment:
* Node.js – Depending on the version, Angular requires either an active Long-Term Support (LTS) or maintenance Long-Term Support (LTS) version of Node.js. The Angular CLI, version 12.2.18, and Node, version 14.16.0, have both been utilized in the completion of my project in particular.
* npm package manager – Many of the features and capabilities of Angular, the Angular Command Line Interface (CLI), and Angular applications are dependent on npm packages. You will need a npm package manager in order to be able to download and install npm packages. The version of the package manager known as npm 6.14.11 was the one that I used for my project.

To install the Angular CLI, open a terminal window and run the following command:

In order to start the application, you must initially go to the folder designated as the workplace and then execute the following command:

The above stated command will automatically open the browser to <http://localhost:4200/> where the UI will be loaded.

### Python

There are certain system requirements for Python Installation that must be met in order for the project to be successfully run and those are the following:

1. Operating system: Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10, with 2GB RAM (4GB preferable)

2. You have to install Python 3.6 and related packages, please follow the installation instructions given below as per your operating system.

3. In order to run the unit tests Coverage.py version 6.5.0 must be installed. Coverage.py is a tool that can measure the amount of Python code that has been executed. The measurement of coverage is typically utilized in order to evaluate the efficiency of tests. It is able to show which parts of the code are being exercised by tests and which parts are not being exercised by tests.

Python 3.8.9 is the version that I am currently using for my project; however, other versions of Python beginning with 3.6 and higher will also work.

# Testing

Testing is a significant component in determining the overall quality of software development. It is a fundamental method for determining whether the software in question contains any errors that could be potentially damaging to the system. Testing also has the potential to uncover errors in the program's requirements, designs, and even its coding. It evaluates whether or not the system is functioning in accordance with its requirements and whether or not there is room for error to occur in the system based on the user inputs that are provided.

In order to achieve significant results, I have performed a number of different kinds of testing on the dataset, algorithm and also unit tests in order to ensure that all the code meets quality standards. In the points that are listed below, I have provided a comprehensive explanation of the testing that I have carried out.

## Dataset

I was able to locate the dataset that I needed for this project on Kaggle [9], which is an online community for data scientists and practitioners of machine learning. In spite of the fact that I had looked at other datasets, the convenience of this one made it the best option for my project; despite this, it still required some adjustments to be made. In order for me to make use of this data, it was necessary for me to eliminate the necessary features, clean, and normalize the data.

In this section I will explain the necessary test I was able to conduct in the ensuing two subsections in order to gather the required dataset.

### Cleaning Dataset

Cleaning the data was one of the first steps I took in the data analytics process. I prepared and validated the data before beginning my core analysis. In the world of data analytics, a common refrain is "garbage in, garbage out," which means that if you input quality data into your code, you will get quality results as well. This is done for a variety of reasons, including efficiency, security, avoiding errors, increasing productivity, improving mapping, and so on, I have learned these techniques from the machine learning course and the book [15].

Getting rid of observations that weren't relevant to the study was the first thing I did when I started cleaning up my data. There were a few features in the dataset that were not essential for working through the issue with the recommendation system. The duration of the song, which was displayed in milliseconds, was the first element that caught my attention when I was going through the list. This was irrelevant because I there will be a snippet of the song and attach a link to the full version of the song for the user to click on if they wanted to hear the whole thing. The following addition was the ability to tell if a song contains explicit content or not by referring to them with a 0 if the answer is false and a 1 if the answer is true. The following pieces of data that I eliminated were the key and mode columns because I was unable to determine their meaning and Kaggle did not provide this information. The popularity score was the last feature that I eliminated from the app because I believe that any song, regardless of how popular it is, should be able to be recommended to other users. Additionally, I removed the duplicates that appeared in the dataset. Duplicates are quite common in third-party source like Kaggle.

The next step I did was to fix the structural errors. I corrected the occurrences that included typos and inconsistent capitalization. I also removed the songs that had symbols instead of letters in their name, since I noticed this was a mistake that happened in the whole row.

My next crucial action was to standardize the data. I made sure that each cell was adhering to the same rules in this step. In order to retrieve information that is missing from the current dataset, I first numbered each row to make it easy to find it by id. Then, in the following column, I entered the song ids from the Spotify database. The following two columns were set aside for the song's name and artist. Finally, I added all the features that remained after removing the relevant data.

I also had to deal with the data that was missing. Since some of the dataset's samples had missing values for some of the features, I chose to remove them rather than flagging the missing data because doing so might cause predictions to be made that were incorrect, such as listing some songs as being closer to one another when they are not.

Finally, I validated my dataset to see if the previous steps had been carried out correctly. I followed all of the above- mentioned instructions in order to achieve the best results and prevent irrelevant data from influencing my analysis' findings.

### Testing Dataset

I had a lot of features to draw from when I got to the data analytics portion of the item-based filtering to make my predictions. I first put my algorithm to the test using every feature that remained after the data cleaning. Valance, acousticness, danceability, energy, instrumentality, liveness, loudness, speechiness, and tempo were those qualities. My subsequent results weren't as accurate, consequently some of the songs had higher similarity scores even though they didn't seem to be that similar. I had to reduce the amount because all of these features created a complex model that overfit the features.

I've discovered a recommendation system that is similar to the one I was aiming for after doing some research. Only a few of the features—acousticness, danceability, energy, instrumentality, liveness, speechiness, and valence—were used by the project's author [3]. The predictions were more accurate after testing my algorithm with these features, but there was still room for improvement.

In the end, I made the decision to conduct my own analysis of the features and make my selections in light of the findings. I discovered that there are a few feature selection methods after doing some research. In the end, I made the decision to use the schema I discovered in the article [2] for my project. Here, the Spearman's and Pearson's rank correlation coefficients were suggested based on the input and output variables of my project.

For evaluating the correlations between the features I used the corr() function in R in order to evaluate the correlation between the point of my features. For this I have removed the features that were unnecessary for the analytics, and work only with the necessary features like acousticness, danceability, energy, instrumentality, liveness, speechiness, instrumentalness, loudness, tempo and valence. With the mentioned features I plotted two correlation plots using the library corrplot and used the Spearman and Pearson methods respectively. The results that I got from the two plots were very similar, where the features that were less correlated were liveness, speechiness and tempo. This is how I decided which features to keep for the predictions and which to remove. These were the best results that I got when testing the features with the algorithm. The similarity score seemed quite accurate and I decided to stick with this version of features from the dataset.

## Algorithm

For the particular problem of recommending similar songs I have used the k-nearest neighbor (k-NN) algorithm which is a type of supervised machine learning that uses labeled input data to train a function that produces an appropriate output when given additional unlabeled data. This has been used in my particular scenario to address a categorization problem. When using this method, an unidentified item is given the label that occurs the most frequently among its k closest neighbors. A neighbor is considered to be the closest neighbor if, as determined by the cosine angle, they are the closest to one another in feature space. A class member is the result of a k-nearest neighbor classification. An object is categorized by the majority vote of its neighbors and is then assigned to the category with the highest prevalence among its k closest neighbors (k is a positive integer, typically small). When k is equal to 1, the object is simply identified as falling under the category of its one closest neighbor.

When I put my algorithm into practice, I tested it on a variety of datasets and came to the conclusion that the dataset I obtained from the correlation matrix produced the best results. However, using the k-NN algorithm had some drawbacks after picking the appropriate features. The k-NN algorithm is not very efficient when working with data that has a lot of dimensions because it becomes more challenging for the algorithm to compute the distance in each dimension as the number of dimensions rises. As a result, I decided to use other distance measure choices in my project to find a solution to the problem. The couple distance measures that I have tested with mu algorithm were Euclidean, Manhattan, and Chebyshev distance, which are a part of the same family of distances. Additionally, I also tried using inner product distance measures, which are from a different family of measures. Those included the Cosine and Chord distance.

Euclidean and Manhattan distance produced the best results, according to the results I found using the various distances. I have chosen to use both distance measures after doing some research on which distance would be best for my model and algorithm. Euclidean distance would be ideal for fewer measures, according to the academic paper [1], so given that I have six features, I have chosen to use this for the item-based filtering. I have chosen to use the Manhattan distance for the collaborative base filtering because according to [1] it performs significantly better with a greater number of features and I have a much larger matrix, which is be n x m, where n is the number of songs and m is the rated songs from users.

I also needed to make some tests regarding the user-rating matrix, in order to check the sparsity of the matrix. In numerical analysis and scientific computing, a sparse matrix or sparse array is a matrix in which most of the elements are zero. The number of zero-valued elements divided by the total number of elements (e.g., m × n for an m × n matrix) is sometimes referred to as the sparsity of the matrix. This was quite likely to happen since I wanted to make a user-rating ratio that is realistic.

## Unit Testing

These tests represent a software testing method with which individual units of source code, such as sets of program modules together with associated control data, are tested in order to determine whether they are fit for use.

Some of the conducted unit tests for this application are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Name** | **Testing Condition** | **Needed Output** | **Result** |
| test\_reset\_indices | Checking if index is equivalent to row number | Index equals row number 576 | Success |
| test\_get\_first\_artist | Getting the artist with index 114618 | Artist at index 114618 – The Weeknd | Success |
| test\_remove\_special\_chars | Utilizing mock data to remove song name with special characters | Removing the first record from the mock data | Success |
| test\_find\_duplicates | Utilizing mock data to find duplicate song according to id | Finding the duplicate record | Success |
| test\_delete\_duplicates | Utilizing mock data and setting duplicate tracks | Deleting duplicate track | Success |
| test\_get\_unique\_artists | Utilizing mock data and setting duplicate artist’s name | Finding unique artist and appending them to array | Success |
| test\_euclidean\_distance | Utilizing mock data and calculating Euclidean distance between lists | Getting the value of Euclidean distance between lists | Success |
| test\_manhattan\_distance | Utilizing mock data and calculating Manhattan distance between lists | Getting the value of Manhattan distance between lists | Success |
| test\_get\_feature\_list | Setting mock list with values equal to record with id 313 | Getting features list of record 313 | Success |
| test\_find\_intersection | Utilizing the mock data to find which points are intersecting | Getting the points that are intersecting between the two lists | Success |
| test\_manhattan\_distance\_users | Utilizing mock data and calculating Manhattan distance between list of users | Getting the value of Manhattan distance between lists of users | Success |

Figure 18. Table representation of the unit tests conducted

In addition to that, I have compiled a coverage report, which is presented in Figure 19. The report displays, as a percentage, the proportion of the code that has been run or that has been tested. The figure demonstrates that I have covered one-fourth of the project's code.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 19. Coverage report

Despite the fact that the coverage report percentile is lower than what is recommended, I have proceeded with this in order to test the output of some of the most important functions. For instance, I had to ensure that the functions that were calculating the distance were delivering accurate results because these functions were the primary components of my algorithm. Because of this, I can now guarantee that the user will receive accurate recommendations regarding the results. The functions for data cleaning were also among the most important ones that needed to be tested. These were essential due to the fact that the database contained a large number of duplicate records, and if those records were not removed, the algorithm would advise the user to listen to the same song twice.

# Conclusion

The purpose of this project was to make music recommendations that a user will find interesting by utilizing statistical analysis and artificial intelligence. In the end, this objective was successfully accomplished. My algorithm is unique in comparison to others because I have implemented a hybrid approach to filtering. Specifically, I have combined user-based filtering with item-based filtering in order to create a collaborative filtering system.

With this method, it is possible to anticipate the musical category, artist, or artists that the user will enjoy listening to at any given time. It has come to everyone's attention that even though we all have bands that we particularly enjoy listening to, we don't always want to listen to the same artists or genres. It's important to keep the listener on their toes and make them appreciate unexpected revelations. As a direct consequence, this is why I decided to create this hybrid filter; the item-based filtering method on its own has a number of drawbacks, which led me to my decision. The best results in terms of music recommendations were produced as a result of this.

Item-based filtering is a method that aims to provide a description of the music by analyzing the content of each individual track. It examines the characteristics of each object to determine the degree to which they are comparable to the characteristics of other objects. A score will be assigned to each item based on how similarly it compares to the other items in the dataset, and then I will recommend an item based on how similarly it compares to everything else in the dataset.

The goal of collaborative filtering, on the other hand, is to analyze user-generated content and assets in order to provide a description of the track within the context of its relationship to other songs available on the website. In the context of making song recommendations from one user to another, I have considered the songs that have ratings that are relatively close to one another and then offer a suggestion for a song that the user in question hasn't rated based on the ratings that the other users have given it. Because of this, users are only suggesting unrated songs to each other if they are comparable to one another.

After I get the results of the two filters, I will put them together to make my hybrid filter. I was able to combine the results from the two filters by taking the song's genre from the active song track. When the results come back, this genre will be given more weight in the user-based or collaborative filter. For example, if the active song that the user wants suggestions for is from the pop genre, this will give the user-based filtering results more weight by only showing the results that are most like the active song from the pop genre. To create a hybrid music recommendation filter, the output from both filters will be combined in the end.

After I have determined the outcomes of the two filters, I have combined their outputs to create my hybrid filter. By using the active song track to determine the genre of the song, I was able to combine the results from the two filters into a single set. When the results are tallied up, this category of music are given additional consideration by the filter that is based on user input or collaborative perseverance. For instance, if the active song that the user wants suggestions for is from the punk genre, this will give the user-based filtering results more weight by only showing the results that are most like the active song from the same genre. In the end to create my hybrid music recommendation filter, the output of both user-based and item based were combined.

When I contrast the initial idea with the finished product, it is easy to say that there is a significant gap between the two in terms of the logic behind the application and the way it functions. The majority of the developed features were being subjected to continuous change as new ones were being developed, and there was a need to integrate the two sets of features together. The user-based (collaborative) filter was the component of the application that underwent the most significant development as it transitioned from being an idea to product. The most difficult obstacle that I had to overcome was the dearth of open-sourced user data that could be found on the internet. This is when I realized that the filter's scope needed to be changed, and I also needed to generate my own data for it.

Unfortunately, there were some functionalities that did not make it to the development process due to the time limit of this project. The features that can be implemented in the future are: user profiles – each user to be able to create a user profile and log in, the ability for users with user profiles to be able to give ratings to the tracks and those to be saved in order for the algorithm to learn from the newly added data, increase and give better results, and make the full length of the track available on the website so the user would not have to use a third party for streaming. Additionally with the implementation of these features the security of the application will also have to be increased.

The application still has a large number of features and functionalities despite the fact that these particular features were not developed. The challenging requirements for this project were complete by combine a number of different algorithms through making use of statistical investigation and artificial intelligence.

# References

[1] H. A. Abu Alfeilat et al., ‘Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review’, Big Data, vol. 7, no. 4, pp. 221–248, Dec. 2019, doi: 10.1089/big.2018.0175.

[2] J. Brownlee, ‘How to Choose a Feature Selection Method For Machine Learning’, Machine Learning Mastery, Nov. 26, 2019. https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/.

[3] L. M. P. Moraes, ‘Searching for Similar Songs on Spotify — Data Science’, Sinch Blog, Jan. 13, 2021. https://medium.com/wearesinch/searching-for-similar-songs-on-spotify-data-science-c7e3faeb66f0.

[4] G. Peoples, ‘Consumers Now Favor Streaming Services for Music Discovery Over All Other Sources’, Billboard, Sep. 30, 2020. https://www.billboard.com/pro/consumers-streaming-music-discovery-music-360/.

[5] ‘How Spotify’s Algorithm Works? A Complete Guide to Spotify Recommendation System [2022] | Music Tomorrow Blog’. https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022.

[6] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, ‘On the Surprising Behavior of Distance Metrics in High Dimensional Space’, in Database Theory — ICDT 2001, vol. 1973, J. Van den Bussche and V. Vianu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 420–434. doi: 10.1007/3-540-44503-X\_27.

[7] M. Montaner, B. López, and J. L. de la Rosa, ‘A Taxonomy of Recommender Agents on the Internet’, Artificial Intelligence Review, vol. 19, no. 4, pp. 285–330, Jun. 2003, doi: 10.1023/A:1022850703159.

[8] <https://www.wikipedia.org/>

[9] Kaggle dataset - <https://www.kaggle.com/datasets/ektanegi/spotifydata-19212020>

[10] <https://en.wikipedia.org/wiki/List_of_music_genres_and_styles>

[11] https://developer.spotify.com

[12] <https://stepzen.com/blog/spotify-oauth-2-tokens-in-3-steps>

[13] <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-track>

[14] <https://rxjs.dev/>

[15] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning: with Applications in R. Springer Science & Business Media, 2013.

Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011.pdf

[16] E. Soubiran, F. Guenab, D. Cancila, A. Koudri, and L. Wouters, ‘Chapter 23 - Ensuring Dependability and Performance for CPS Design: Application to a Signaling System’, in Cyber-Physical Systems, H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, Eds. Boston: Academic Press, 2017, pp. 363–375. doi: 10.1016/B978-0-12-803801-7.00023-7.

# Appendix

Diagram, timeline

Description automatically generated

Figure 1: Authorization code flow diagram of the Spotify API