MINISTRY OF EDUCATION & TRAINING

HO CHI MINH CITY UNIVERSITY OF ECONOMICS AND FINANCE

SCIENTIFIC RESEARCH

Major: E commerce

BUILDING A MUSIC RECOMMENDATION SYSTEM FOR USERS

Instructor : **Tran Thanh Cong (Mr.)**

Students : **Hoang Minh Chau- 215122115**

Vo Ngoc Le Duy- 215120125

Luong Gia Nhi- 215120159

Le Ngoc Quang- 215120577

Tran Thi Nguyet Vien- 215122223

Ho Chi Minh city - 3, 2024

MINISTRY OF EDUCATION & TRAINING

HO CHI MINH CITY UNIVERSITY OF ECONOMICS AND FINANCE

SCIENTIFIC RESEARCH

Major: E commerce

BUILDING A MUSIC RECOMMENDATION SYSTEM FOR USERS

Instructor : **Tran Thanh Cong (Mr.)**

Students : **Hoang Minh Chau- 215122115**

Vo Ngoc Le Duy- 215120125

Luong Gia Nhi- 215120159

Le Ngoc Quang- 215120577

Tran Thi Nguyet Vien- 215122223

Ho Chi Minh city - 3, 2024

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 . INTRODUCTION

1.1 OVERVIEW

Data analysis is an important part of identifying popular content of interest so that algorithms can be set up for each individual element. It is applied in many fields and music is no exception as it is an essential part of every person's life. And with the strong development of music platforms, access and discovery have become easier and richer than ever. But with that development, finding music that suits your preferences has also become more complicated because the number of music songs has become very diverse, thereby showing that music recommendations and suggestions are becoming more and more complex, increasingly respected and developed.

A recommendation system is an algorithm that recommends content to customers based on the content that customers are interested in. It first appeared in 1990 and has been developed even more strongly in recent years, and it is also different from the traditional search function that relies on retrieving data from customer queries to recommend information based on what customers are looking for, while the recommendation system will provide the content information customers need based on consumer behavior habits.

This research will focus in depth on building the music recommendation system based on the dataset of Spotify as follows:

- Collecting data on Kaggle
- Data acquisition and data preprocessing
- Exploratory data analysis
- Building music recommendation

1.2 PROBLEM STATEMENT

In recent years, the music industry has seen a significant shift from owning music to listening to music online. Online music streaming services such as Spotify, Apple Music, YouTube Music is becoming increasingly popular, attracting hundreds of millions of users worldwide because of their convenience, portability, and huge music

library. Users can easily access a huge music store at a reasonable cost, enjoy music anytime, anywhere and discover new artists and genres.

However, along with the abundance of online music services, users also face a challenge of information overload. With millions of songs available, choosing and discovering music according to personal taste is more difficult than ever. This is the reason why the demand for personalized music recommendations is increasing. Users expect music services to understand their music preferences, introduce new songs and artists that suit their preferences, help them save time searching and provide better listening experiences. Therefore, building a music recommendation system based on the Spotify dataset is necessary, the system can recommend songs that are likely to attract users, thereby increasing service usage time, satisfaction and loyal user's status.

1.3 OBJECTIVES OF THE STUDY METHODOLOGY

The study focuses on information processing of the Spotify dataset with the aim of understanding popular trends in music songs and what specific factors may influence each user's music preferences. This is the basis for creating a music suggestion model for users based on elements in the song.

From analyzing Spotify data, the research will produce a music recommendation model using Cosine Similarity Algorithms. Specifically, the study will detail the musical features used to calculate similarity, such as genres, tempo, loudness, and valence, etc. After that, music suggestions will be given to the user based on user’s Spotify playlist.

1.4 RESEARCH STRUCTURE

The research includes the following main contents:

Chapter 1: Introduction

Chapter 2: Data acquisition and data preprocessing

Chapter 3: Exploratory data analysis

Chapter 4: Data preprocessing

Chapter 5: Building Recommendation System

Chapter 6: Conclusion

Chapter 2 . DATA ACQUISITION AND DATA PREPROCESSING

2.1 DATA ACQUISITION

Data plays an important role in developing music recommendation systems. To build a music recommendation system, it is necessary to first analyze the collected data.

The music dataset used for the project is collected at Kaggle and includes 2 files: artist.csv and track.csv, in which the artist.csv file contains the id, followers, genres, name, popularity, and the track.csv file contains data such as track id, popularity, track duration in milliseconds, artist name and id, publish date, danceability, energy, key, volume, mode, voice, sound, instrumentation, liveliness, valence, tempo, and time signature. Below are the detailed meanings of main components in track.csv:

- **Popularity**: indicates how popular a song is, often based on factors such as streams, downloads, and shares.
- **Duration_ms:** represents the length of the song in milliseconds.
- **Explicit:** indicates whether the song contains explicit content (0: no, 1: yes).
- **Danceability:** reflects how easy it is to dance to a song based on its rhythm, tempo, and musical structure.
- **Energy:** represents the energy level of a song based on loudness, tempo, and changes in sound.
- **Key:** represents the song's midrange, such as C major or A minor.
- **Loudness:** represents the loudness of the song in decibels (dB).
- **Mode:** The mode differentiates between the major and minor scales of the song.
- **Speechiness**: indicates the degree to which a song uses speech compared to music.
- **Acousticness:** describes whether the song uses primarily acoustic instruments or electronic/electric instruments
- **Instrumentalness:** index that predicts whether a song is instrumental or not. The closer it is to 1, the higher the percentage of songs that are instrumental music.
- **Liveness**: indicates the degree to which a song was recorded live, without editing.

- **Valence:** represents the level of positive emotions in a song.
- **Tempo**: shows the speed of a song in beats per minute (BPM).

2.2 DATA PREPROCESSING

Data preprocessing is an important step because the data after being collected is raw data and if used in a project, it will cause errors and unclear results that the system can give. Therefore, data preprocessing is the step to clean, organize and model that data clearly.

2.2.1 Import Python Libraries

To manipulate and process number tables, the libraries used are pandas and numpy, in which pandas takes on the role of reading dataset, data manipulation and analysis and numpy is number computing.

```
import pandas as pd
import numpy as np
```

To load and preprocess text data, libraries such as json and re will be used

```
import json
import re
```

2.2.2 Reading Dataset

After uploading the track.csv and artist.csv files and importing the library, the next step is to read the data files. Start by reading the track.csv file.

```
spotify_df = pd.read_csv('/content/tracks.csv')
```

Checking data by code:

```
spotify_df.head()
```

	id	name	popularity	duration_ms	explicit	artists	id_artists	release_date	danceability	energy	...	valence	tempo	time_signature	artists_upd_v1	artists_upd_v2	artists_upd	artists_song
0	3u1C6nWVRoP5F0w8gGrDL3	사랑의 미로	25	222380	0	['최진희']	['1NSrAf8XJYJVgAXKoxaMet']	1987-06-01	0.367	0.194	...	0.367	144.316	4	[최진희]	[]	[최진희]	최진희사랑의 미로
1	1Mv4u308L16NZDZID6HZCy	사랑은 힘든가봐	28	213440	0	['지수']	['4c9QIMfEbIlynuaswyxGx9']	2005-12-23	0.675	0.785	...	0.623	103.008	4	[지수]	[]	[지수]	지수사랑은 힘든가봐
2	1jvoY322nxyKXq8OBhgmSY	어떡하죠	44	244360	0	['지선']	['2Mo9NQaNCFCWSR5CnlfmbN']	2011-10-13	0.606	0.341	...	0.294	135.667	4	[지선]	[]	[지선]	지선어떡하죠
3	2ghebdwe2pNXT4eL34T7pW	그아픔까지사랑한거야	32	237688	0	['조정현']	['2WTpsPucygbYRnCnoEUkJQ']	1989-06-15	0.447	0.215	...	0.177	71.979	4	[조정현]	[]	[조정현]	조정현그아픔까지사랑한거야
4	7rxpWwcXNgDUXI0wN0gUvp	전국의 기억 장정우 Version	31	280372	0	['장정우']	['5L7zKs2fhwENWOMI7LFaN1']	2003-12-24	0.494	0.656	...	0.420	82.003	4	[장정우]	[]	[장정우]	장정우전국의 기억 장정우 Version

Figure 2.1 First 5 row values of dataframe spotify df.

File artist.csv will be read and checked by code:

```
data_w_genre=pd.read_csv('/content/artist.csv')

data_w_genre.tail()
```

	id	followers	genres	name	popularity
1162090	3cOzi726lav1toV2LRVEjp	4831.0	['black comedy']	Ali Siddiq	34
1162091	6LogY6VMM3jgAE6fPzXeMl	46.0	[]	Rodney Laney	2
1162092	19boQkDElay9GaVAWkUhTa	257.0	[]	Blake Wexler	10
1162093	5nvjpU3Y7L6Hpe54QuvDjy	2357.0	['black comedy']	Donnell Rawlings	15
1162094	2bP2cNhNBdKXHC6AnqgyVp	40.0	['new comedy']	Gabe Kea	8

Figure 2.2 First 5 row values of dataframe artists

Next, check the data type of the artist.csv file data_w_genre.dtypes

```
id              object
followers      float64
genres          object
name            object
popularity       int64
dtype: object
```

Figure 2.3 Data type of artists.csv file

2.2.2.1 Genres Processing

- **Process dataframe "data_w_genres"**

According to the data type checking results of "data_w_genres", Python is reading the data in the "genres" column as a string type. This can have an impact on future data processing and usage, as it would require changing the data type to an object type.

Here is an illustration of how Python reads the category as a string:

- data_w_genre['genres'].values[1200]: This code accesses the value at position 1200 in the "genres" array of the "data_w_genre" dataframe.
- ['kleine hoerspiel', 'lesen']: This is the value at position 1200 in the "genres" array of the "data_w_genre" dataframe.
- data_w_genre['genres'].values[1200][0]: This code accesses the first value in the music genre list of the 1200 songs in the "data_w_genre" dataframe.
- [': This is the first value in the music genre list of the 1200 songs in the "data_w_genre" dataframe.

To address this issue, a new column called "genres_upd" is added to the "data_w_genre" dataframe, with the data in the new column represented in double quotes and spaces replaced with underscores. This helps to accurately represent the data.

data_w_genre['genres_upd'] = data_w_genre['genres'].apply(lambda x: [re.sub(' ', '_', i) for i in re.findall(r'"([^'])"', x)])

After making the necessary changes, rechecking the results with the same code provides more accurate outcomes:

data_w_genre['genres_upd'].values[1200][0]: The result is 'kleine hoerspiel', which is the first value in the music genre list of the song at position 1200 in the "data_w_genre" dataframe.

In this section, we are importing the "genres"column from the "data_w_genre" dataframe into the "spotify_df" dataframe. The goal is to combine the data from both dataframes and enhance the "track.csv" file with the genre information from the "artist.csv" file.

To achieve this, we perform the following steps:

Create additional columns and process them:

Enter the genre column into "spotify_df":

Merge "data_w_genre" with "artists_exploded" based on the "artists_upd" and "name" columns:

Remove null values resulting from the merging process:

Further clean the data by grouping the ids and consolidating the genres:

Merge the consolidated genres with the "spotify_df" dataframe

In summary, we import the "genres" column from "data_w_genre" into "spotify_df" by performing the necessary data processing and merging steps. The final result is a merged dataframe that includes the consolidated genre information for each track in the "spotify_df" dataframe.

2.2.3 Check for Duplication

The raw data collected will certainly have duplicate parts and in music data it will happen that a song can be sung by many singers at different times, so the data cannot be processed. To avoid data being too large or giving duplicate results, the duplicate data will be removed through code:

```
spotify_df.drop_duplicates('artists_song',inplace =
True)
```

2.2.4 Data Reduction

The collected data set has a very large amount of data, which affects the running of the data, so data reduction is an essential task. That will be done based on the year of the track. The year of the songs in the collected dataset ranges from 1925 to 2020. So it is too large to process so the solution here is only take data from 2010 onwards and will discard the rest because data before 2009 will cause problems affect performance

```
    # Convert 'year' column to integers

    spotify_df['year'] = spotify_df['year'].astype(int)

    # Assuming 'year' column contains the year
    information

    condition    =    (spotify_df['year']    >=   1925)
   & (spotify_df['year'] <= 2009)
    # Filter out records within the specified range

    filtered_df = spotify_df[~condition]

    # If you want to modify the original DataFrame, you
can reassign the filtered DataFrame back to it
    spotify_df = filtered_df.copy()
```

To summarize, Data preprocessing is considered the first important step in building a music recommendation system because raw data has so many problems like duplicate data, unnecessary data makes data too large, format of data has some mistakes. Therefore, raw data often needs to be cleaned, structured, and refined to make it suitable for analysis or model development. After preprocessing, proceed to analyze the dataset, preparing for the next stage in building a music recommendation model.

Chapter 3 . EXPLORATORY DATA ANALYSIS

3.1 STATISTICS SUMMARY

	count	mean	std	min	25%	50%	75%	max
popularity	114573.0	39.741911	20.096225	0.000	30.0000	43.000000	54.00000	99.000
duration_ms	114573.0	228522.705454	135713.569204	4937.000	185062.0000	217142.000000	254705.00000	5403500.000
explicit	114573.0	0.139405	0.346370	0.000	0.0000	0.000000	0.00000	1.000
danceability	114573.0	0.616372	0.159564	0.000	0.5150	0.629000	0.73300	0.988
energy	114573.0	0.650373	0.217709	0.000	0.5130	0.678000	0.82300	1.000
key	114573.0	5.327712	3.581864	0.000	2.0000	6.000000	9.00000	11.000
loudness	114573.0	-7.428410	3.935782	-57.093	-8.7300	-6.630000	-5.04600	1.933
mode	114573.0	0.594957	0.490903	0.000	0.0000	1.000000	1.00000	1.000
speechiness	114573.0	0.096132	0.105345	0.000	0.0359	0.051800	0.10600	0.966
acousticness	114573.0	0.291940	0.292568	0.000	0.0356	0.186000	0.49500	0.996
instrumentalness	114573.0	0.092468	0.249386	0.000	0.0000	0.000002	0.00111	1.000
liveness	114573.0	0.202358	0.175856	0.000	0.0969	0.129000	0.25600	0.998
valence	114573.0	0.512633	0.248596	0.000	0.3160	0.509000	0.71200	1.000
tempo	114573.0	122.006989	28.914801	0.000	98.9990	123.019000	139.91900	229.862
time_signature	114573.0	3.931467	0.374350	0.000	4.0000	4.000000	4.00000	5.000

Figure 3.1 The statistical table summarizes the data of the song's attributes

Data partition

- The dataset includes 114,573 songs, along with 15 attributes that describe the musical characteristics of each song.
- The data has many attributes (numeric attribute) such as duration_ms, loudness, tempo, etc. and some category properties (classification properties) like mode, key, etc.

From the statistics summary, we can infer the below findings:

- The 20.09 standard deviation shows a wide distribution of popularity, with some songs possibly being very popular (high popularity) and some songs being little known (low popularity). Reflecting the general trend of the music market, where a few famous artists attract the majority of attention and listenership.
- The value max duration_ms = 5403500 is equivalent to 90 minutes, possibly due to an input error or because the song is a symphony or live album.
- Songs with sensitive content have a rate of 13.94%, that song may contain content related to sex, violence or drugs.
- Valence index Relatively evenly distributed, with many songs having average levels of positive emotions.
- Min value 0 shows that there are some songs that can evoke negative emotions.
- A max value of 1 shows that there are some songs that can evoke extremely strong positive emotions.
- Energy has an Average index of 0.69 indicating that most songs have an average energy level. This element has Standard deviation 0.22 shows the difference in energy levels between songs, with some songs having higher energy and some songs having lower energy.
- A min value of 0 indicates a song with low energy (0) which could be a soft ballad, or an input error.
- A max value of 1 indicates: There are some songs with extremely high energy that could be a vibrant rock or EDM song.
- Before performing EDA, it is necessary to separate variables and classify them for ease of analysis. This set of values can be classified into two types of variables: categorical and numerical.

```
cat_cols=spotify_df.select_dtypes(include=['object']).columns
num_cols = spotify_df.select_dtypes(include=np.number).columns.tolist()
print("Categorical Variables:")
print(cat_cols)
print("Numerical Variables:")
print(num_cols)
```

Figure 3.2 Two types of variables: categorical and numerical of the data set

Results after performing data separation:

```
Categorical Variables:
Index(['id', 'name', 'artists', 'id_artists',
'release_date', 'artists_upd_v1',
      'artists_upd_v2', 'artists_upd', 'artists_song',
      'id_x', 'consolidates_genre_lists', 'year'],
     dtype='object') NumericalVariable:
['popularity', 'duration_ms', 'explicit',
'danceability', 'energy', 'key', 'loudness', 'mode',
'speechiness', 'acousticness', 'instrumentalness',
'liveness', 'valence', 'tempo', 'time_signature',
'popularity_red']
```

After analyzing the data, it can be seen that the values ‘artists_song', 'consolidates_genre_lists', 'year' belong to the category of categorical variables. Meanwhile, the variables 'popularity', 'duration_ms', 'explicit', 'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature', 'popularity_red' are of type variables.

3.2 EDA UNIVARIATE ANALYSIS

To visualize the dataset, the research uses data visualization using the Matplotlib and Seaborn libraries. At the same time, using the EDA Univariate Analysis to better understand the meaning of the variables in the data set.

```
for col in num_cols:
    print(col)
    print('Skew :', round(spotify_df[col].skew(), 2))
    plt.figure(figsize = (15, 4))
    plt.subplot(1, 2, 1)
    spotify_df[col].hist(grid=False)
    plt.ylabel('count')
    plt.subplot(1, 2, 2)
    sns.boxplot(x=spotify_df[col])
    plt.show()
```

Figure 3.3 Performing Univariate analysis using Histograms and Box Plots for Continuous Variables.

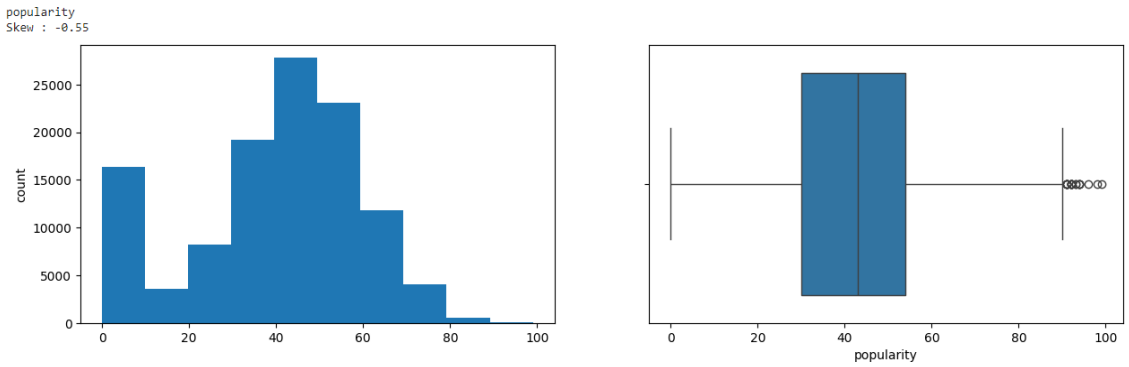

Figure 3.4 Popularity Histograms and Boxplots

Based on figure 3.4, the skews value is -0.55, indicating a skew to the left. The histogram shows the distribution of popularity data. Most songs cluster in popularity from 30 to 60. There are a few songs with popularity higher than 80. Looking at the popularity chart, it shows that the number of songs with the highest popularity is 99/100, accounting for very little less than 1000 songs. This research will rely on the number of most popular songs to analyze the score of its elements later.

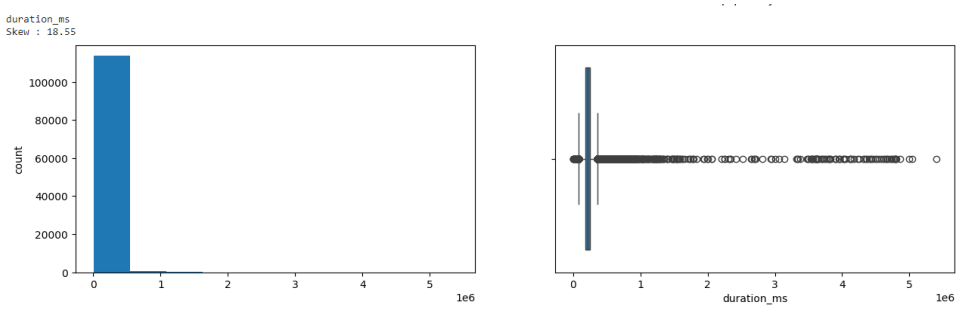

Figure 3.5 Duration_ms Histograms and Boxplots

Figure 3.5 shows the duration_ms value has a skewness to the right with a skew value of 18.55. This means that the majority of duration_ms values are concentrated at the low end, while a few higher values span the right tail of the graph. The reason for this is that most songs have a short duration, usually less than 5 minutes. There are only a handful of songs that are longer. The figure 3.6 below shows some songs with extremely long lengths.

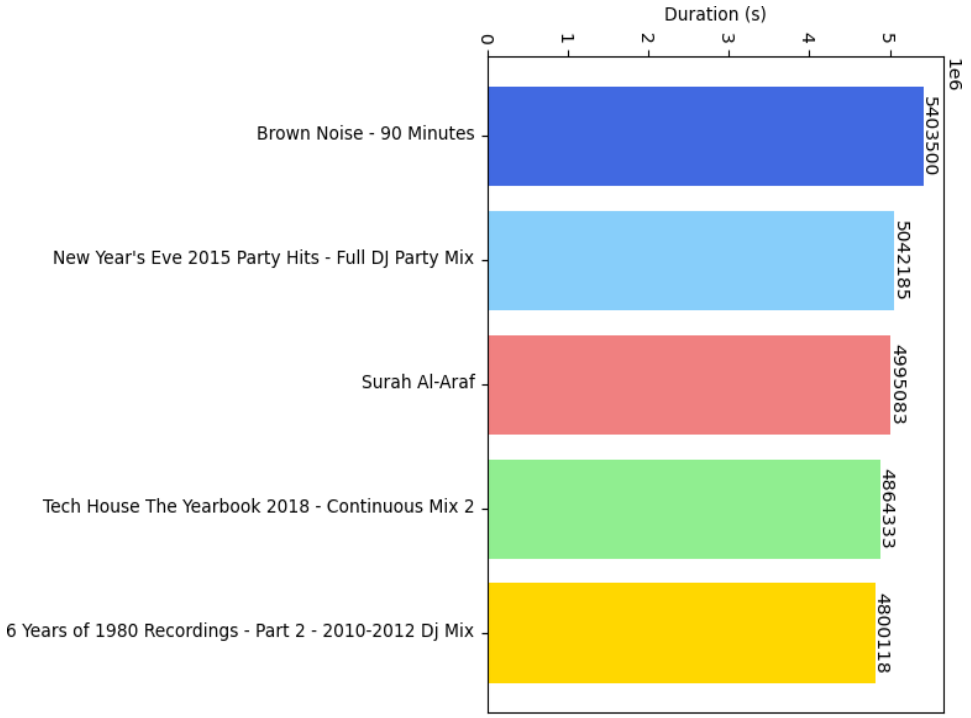

Figure 3.6 Top 5 songs with the longest duration

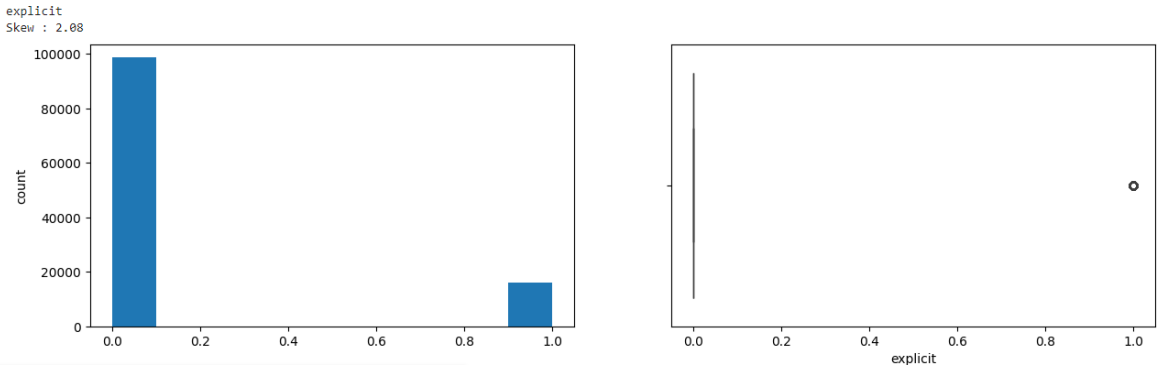

Figure 3.7 Explicit Histograms and Boxplots

Figure 3.7 shows that the Apparent Deviation Value is 2.08, which indicates a rightward deviation of the apparent data. Explicit has only 2 values: zero and one and most songs have zero value, this means these songs do not contain sensitive words. The exceptions are due to a minority of songs containing inappropriate language. This is because many artists refrain from using vulgar language in their music either to reach a broader audience or because they feel it doesn't align with their musical style, and many listeners dislike explicit language for those reasons.

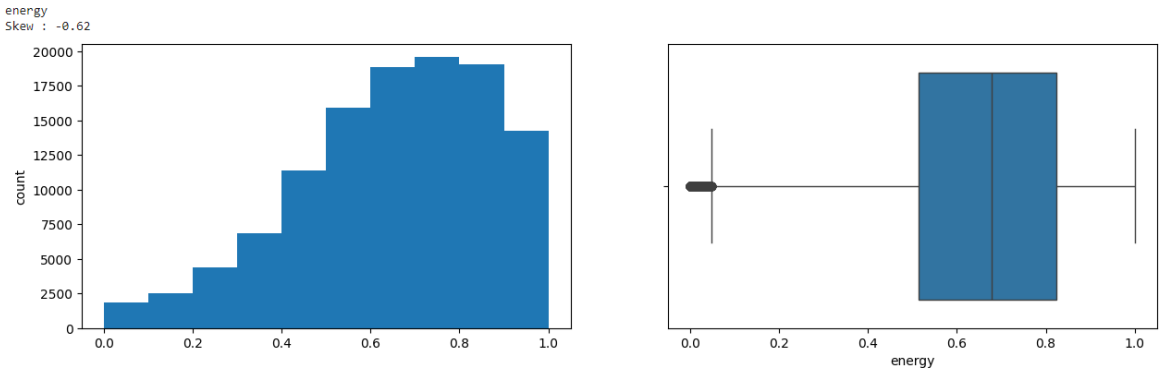

Figure 3.8 Energy Histograms and Boxplots

Based on Figure 3.8, it shows that most songs have an average energy level (about 0.5 - 0.8). However, there are some songs with lower, more than average energy (left side of the chart) that pull the chart to the left. It's possible that the dataset includes many songs in classical rock, pop, or electronic genres, which typically have a higher energy level than genres such as ballad, acoustic.

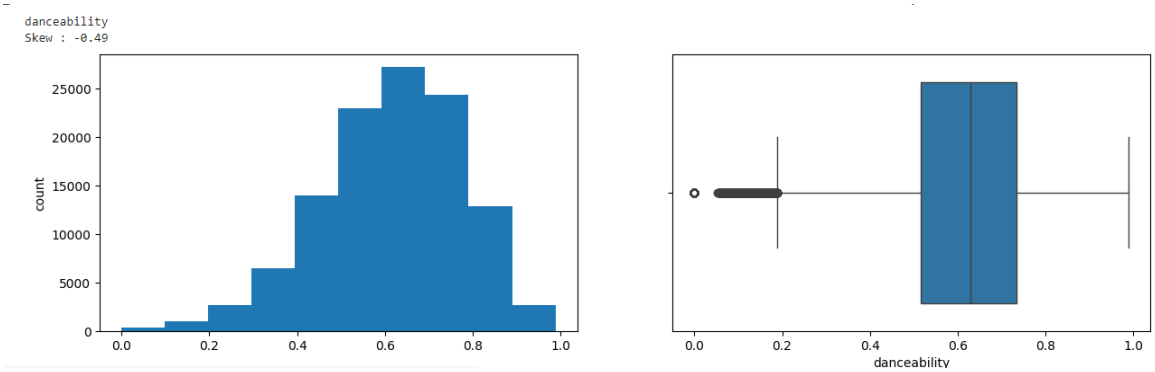

Figure 3.9 Danceability Histograms and Boxplots

Figure 3.9 shows that most of the songs have average danceability (about 0.5 - 0.7). However, there are more songs with lower than average danceability levels (left side of the chart) than there are songs with higher danceability levels (right side). The -0.49 skewness confirms this left-skewed distribution.

This shows that the data set includes many songs in the pop, rock, or dance genres, which often have higher danceability levels than ballad, acoustic, or classical genres.

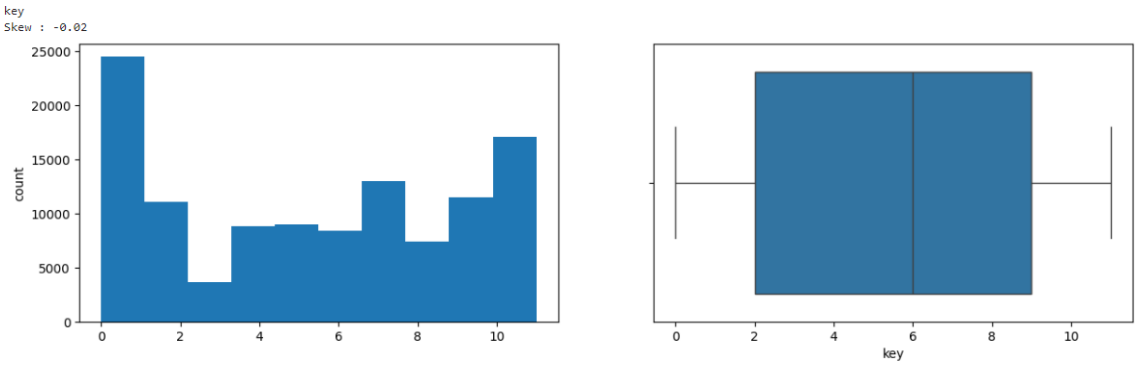

Figure 3.10 Key Histograms and Boxplots

The keys in Figure 3.10 are distributed relatively evenly (not skewed to the left or right). The -0.02 skewness shows that this distribution is roughly balanced, with roughly equal numbers of songs in different keys. The dataset includes many songs from a variety of genres, resulting in a relatively uniform key distribution.

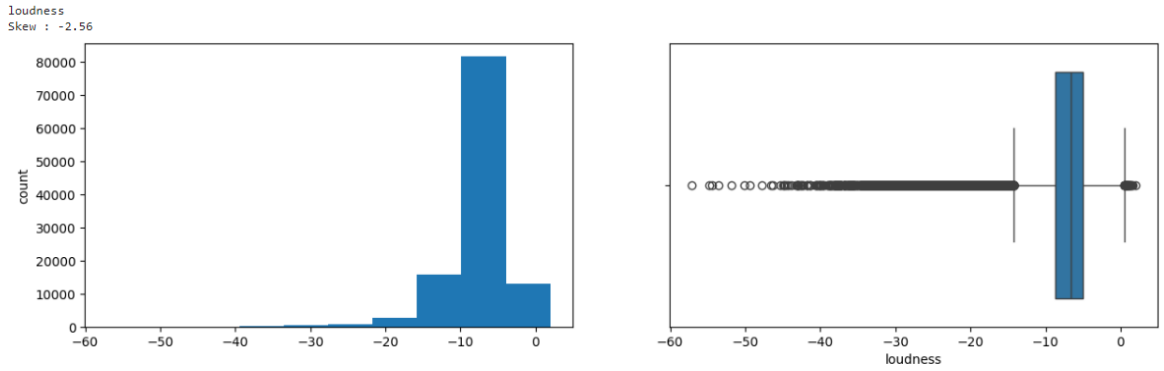

Figure 3.11 Loudness Histograms and Boxplots

Figure 3.11 shows that most songs have relatively high loudness levels (about -5 dB to 0 dB-). There are some songs with low loudness levels (20 dB to -10 dB) The -2.56 skewness confirms this left-skewed distribution. This means that the proportion of songs with a high volume is higher.

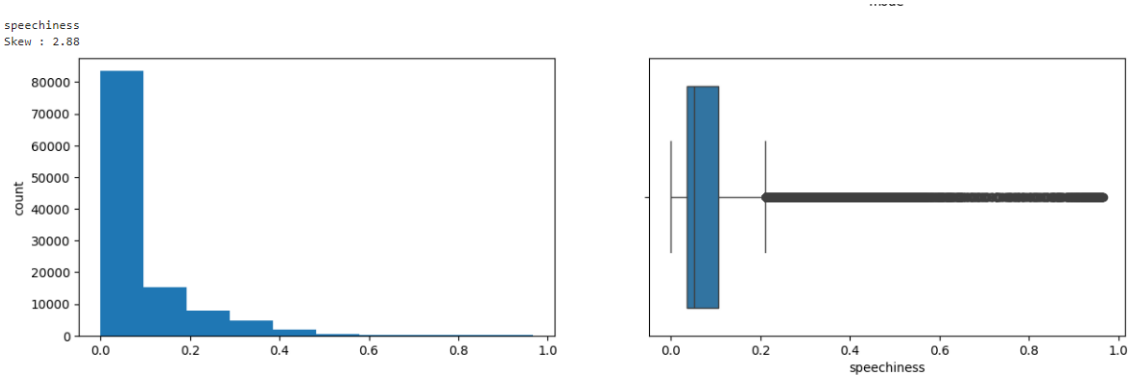

Figure 3.12 Speechiness Histograms and Boxplots

Figure 3.12 shows that most songs have a low level of speechiness (about 0 - 0.2). There are some songs with higher levels of speechiness (0.4 - 0.6) and the distribution is heavily skewed to the left, which is understandable because the data here are all songs, not audiobooks or podcasts. The data set includes many instrumental, electronic, or ambient songs, which typically have lower speechiness than pop, rock, or rap.

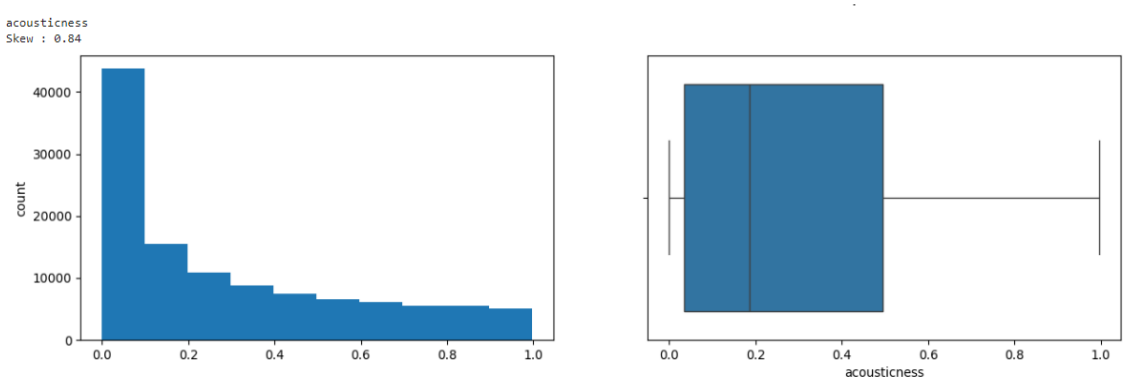

Figure 3.13 Acousticness Histograms and Boxplots

The figure 3.13 shows that the majority of songs have high sound levels, concentrated between 0 and 0.2. The number of songs with lower sound levels (from 0.4 to 1) is less than the group of songs with high sound levels. A skewness of 0.84 confirms this right-skewed distribution, indicating that the data set has more "electronic songs" than "acoustic" songs.

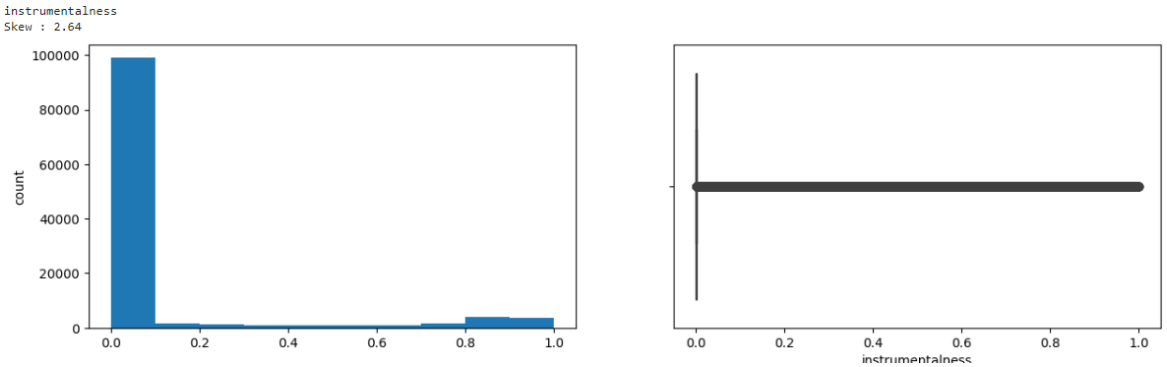

Figure 3.14 Instrumentalness Histograms and Boxplots

Based on figure 3.14, the instrumentalness of the songs mostly has a value of zero and the distribution is heavily skewed to the left, meaning that most of the songs have lyrics

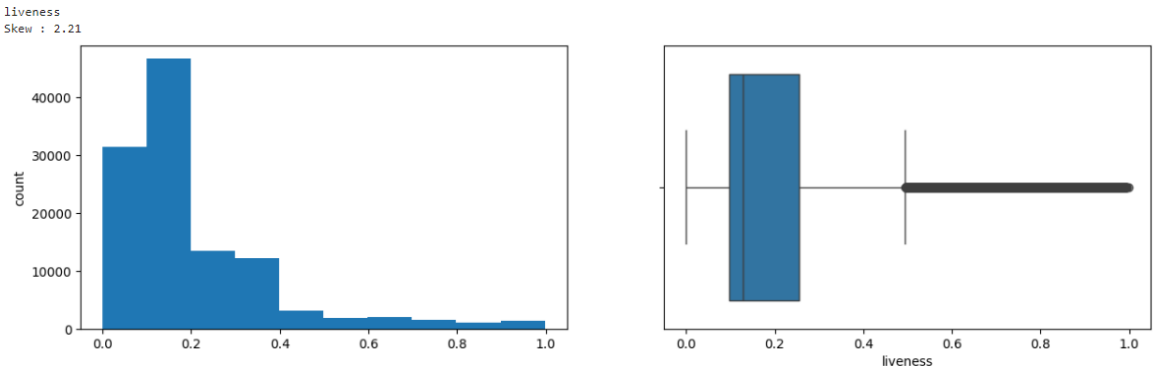

Figure 3.15 Liveness Histogram and Boxplots

The figure 3.15 shows that the majority of songs have low liveness levels, concentrated between 0 and 0.4. The number of songs with higher liveness levels (from 0.4 to 0.6) is much smaller than the group of songs with low levels. The skewness of 2.21 confirms this right- skewed distribution, indicating that the dataset has more "recorded" songs than "live" songs.

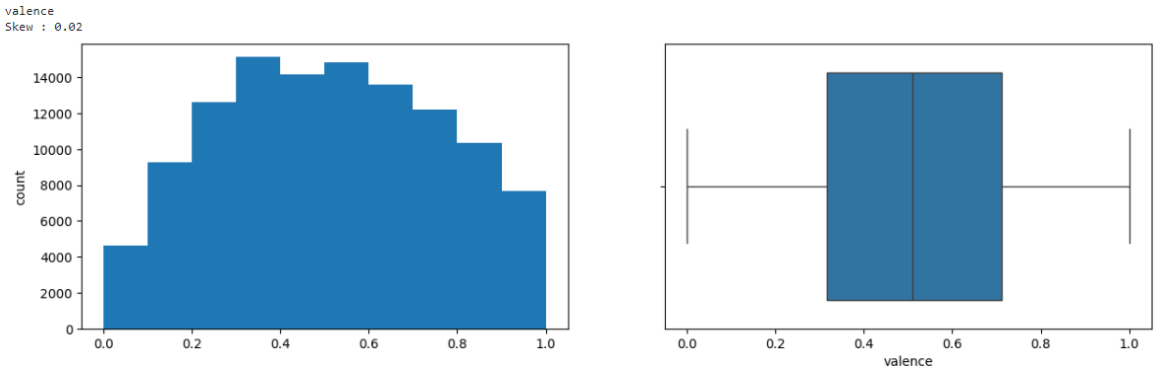

Figure 3.16 Valence Histogram and Boxplots

Figure 3.16 shows that A valence deviation of 0.02 represents an almost symmetrical distribution, which infers that the data set has an almost equal number of happy and sad songs, meaning the data set has many diverse song genres.

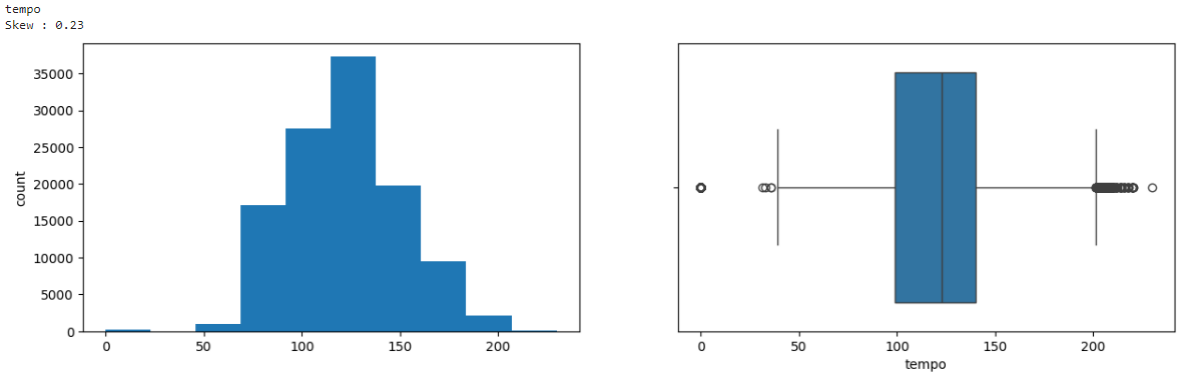

Figure 3.17 Tempo Histograms and Boxplots

Figure 3.17 shows that most songs have a medium tempo (about 100 - 120 BPM). There are some songs with higher (140 - 160 BPM) and lower (80 - 100 BPM) tempo levels, but the number is less than average tempo songs. A skewness of 0.23 indicates a slightly right-skewed distribution, meaning that the data set has more fast songs than slow songs. However, this difference is not much.

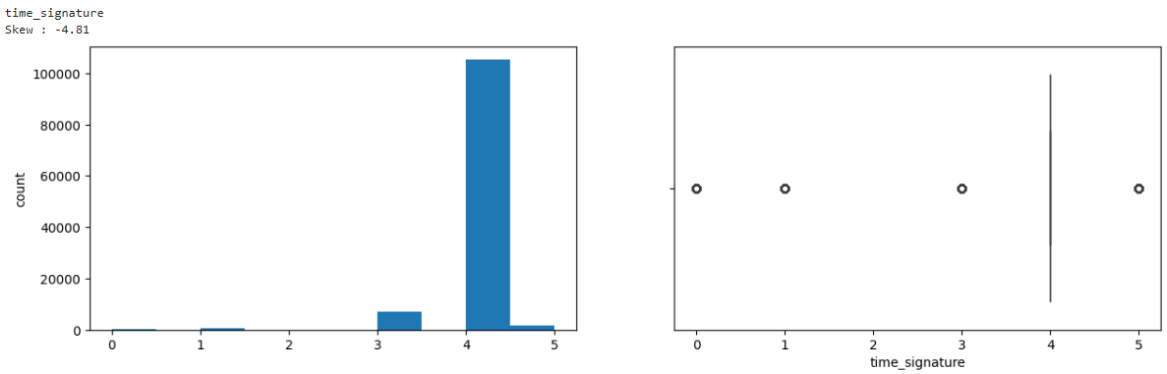

Figure 3.18 Time_signature histogram and Boxplots

Based on the figure 3.18 above, the majority of songs have a simple time signature (4/4, 3/4). There are a few songs with more complex time signatures (5/4), but they are much fewer than songs with simple time signatures.

The pie chart was chosen to present the distribution of the mode index because it is suitable for displaying the percentage of elements in a set. This case represents the percentage of songs that are classified by formal rules and musical consonance. With two categories of mode metrics, "major scale" and "minor scale," the pie chart makes it easy to compare the percentage of each. This helps the reader easily perceive the distribution of the data and better understand the ratio between the two types of music modes.

```
mode_counts = spotify_df['mode'].value_counts()SS
plt.pie(mode_counts, labels=mode_counts.index, autopct='%1.1f%%')
plt.title("Mode")
plt.show()
```

Figure 3.19 Performing Univariate analysis using pie chart for Mode variables.

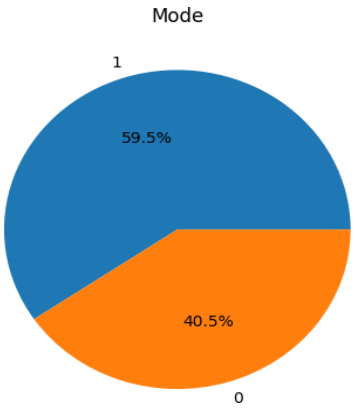

Figure 3.20 Mode Pie chart

According to the figure 3.20, the mode index has a fairly even distribution of songs with 59.5% composed in the major scale and 40.5% composed in the minor scale. The major scale is represented by the number 1, while the minor scale is represented by the number 0.

3.3 DATA TRANSFORMATION

Univariate analysis shows that the two variables Duration_ms and Time_signature have high skewness and are on a larger scale than other variables. Therefore, applying logarithmic transformation to these two variables is necessary in order to:

Helps Normalize data: helping narrow the gap between values, bringing data to the same scale as other variables. This helps ensure uniformity and accuracy in the analysis process.

Minimize the influence of outliers: reducing the influence of extremely large or extremely small values, which can distort the analysis results.

Improve the accuracy and efficiency of statistical analysis: helping data be more normally distributed, suitable for more statistical methods. This helps improve the reliability and efficiency of the analysis process.

Duration_ms represents the length of the song in units of 1 ms. This chart shows that most songs have almost the same duration. But because the average value of this graph is very much skewed to the right, which indicates that the data has been concentrated on 1 side, this graph needs to transform the data for clearer analysis.

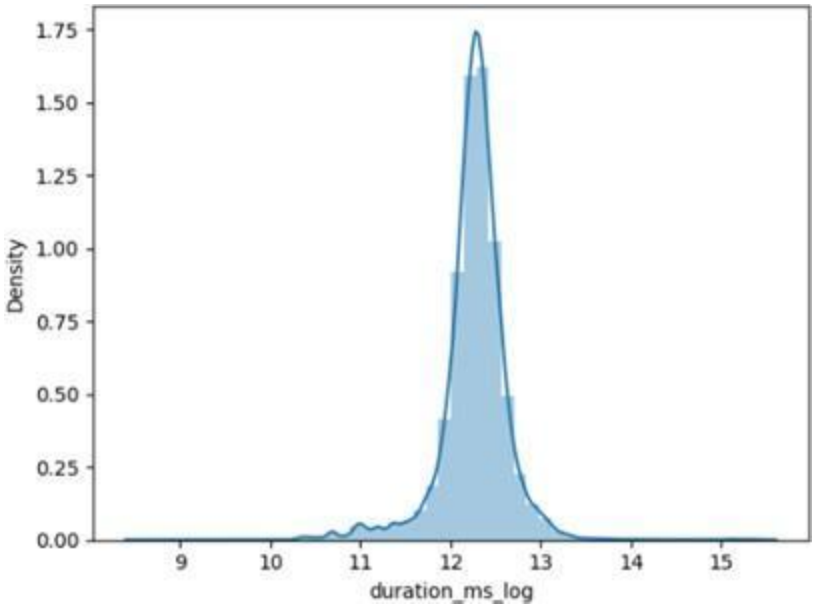

Figure 3.21 Log transformation for duration_ms variable

Through the figure 3.21, it can be clearly seen that the data is only concentrated in the highest column of 12.5, equivalent to e12.5 ms. Most songs of all genres have a duration of e12.5 ms.

```
sns.distplot(spotify_df["time_signature"], axlabel="time_signature");
```

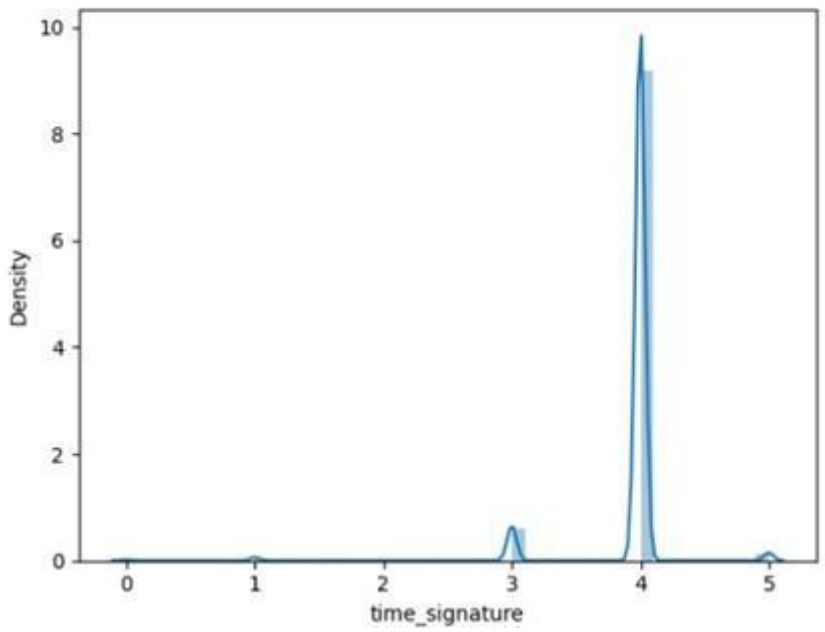

Figure 3.22 Log transformation for Time_signature variable

Time_signature means the beat of the song. Looking at the 3.22, all of the songs composed in beat 4.

3.4 EDA BIVARIATE ANALYSIS

Bivariate Analysis helps to understand how variables are related to each other and the relationship between dependent and independent variables present in the dataset.

```
sns.pairplot(spotify_df,vars=["popularity","explicit",
"loudness",'danceability', "acousticness",'duration_ms_log'])
```

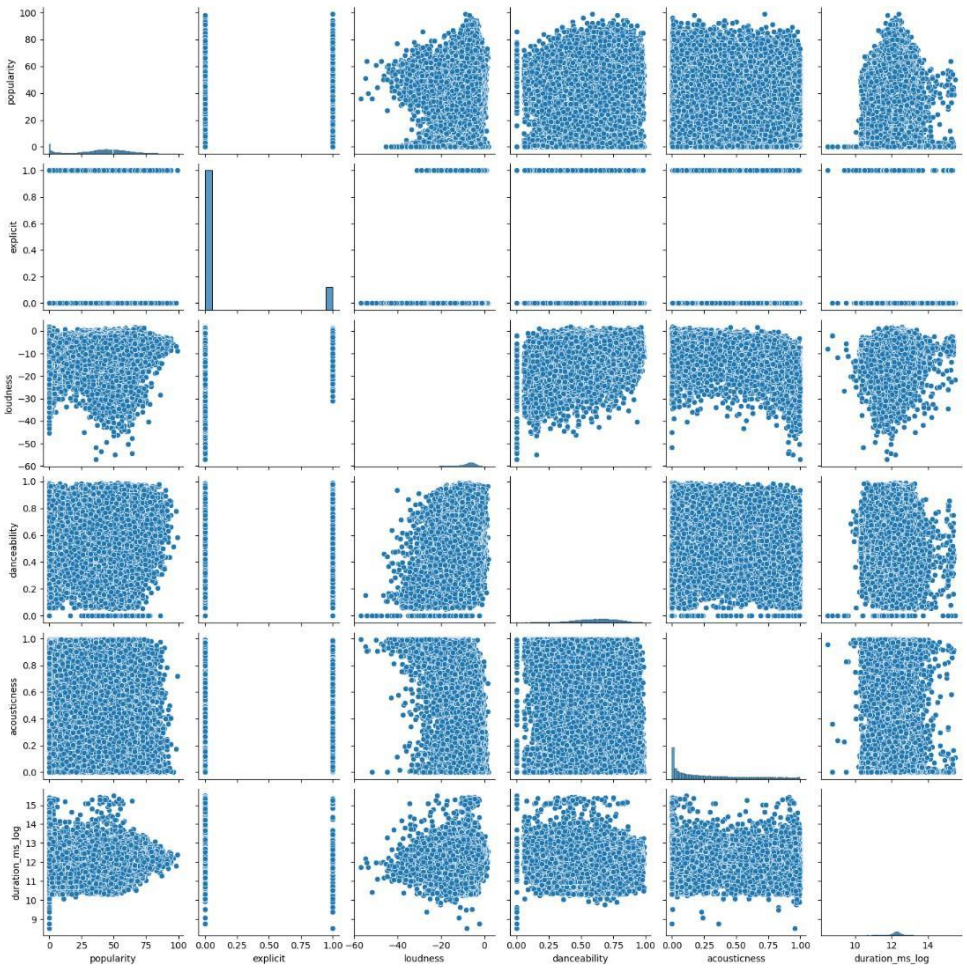

Figure 3.23 Pair plot for variables (1)

Popularity and Loudness have a slight positive correlation. This shows that popular songs (high popularity) tend to be louder (high loudness).

The trend lines for **Danceability** and **Loudness** tend to slope upward. However, this correlation is not too strong and there are many exceptions.

Danceability and **Popularity** have a positive slope regression line showing that songs that are easy to dance to (high danceability) tend to be more popular (high popularity).

```
sns.pairplot(spotify_df,vars=["energy",'valence','acoustic
ness','loudness','instrumen talness','popularity'])
```

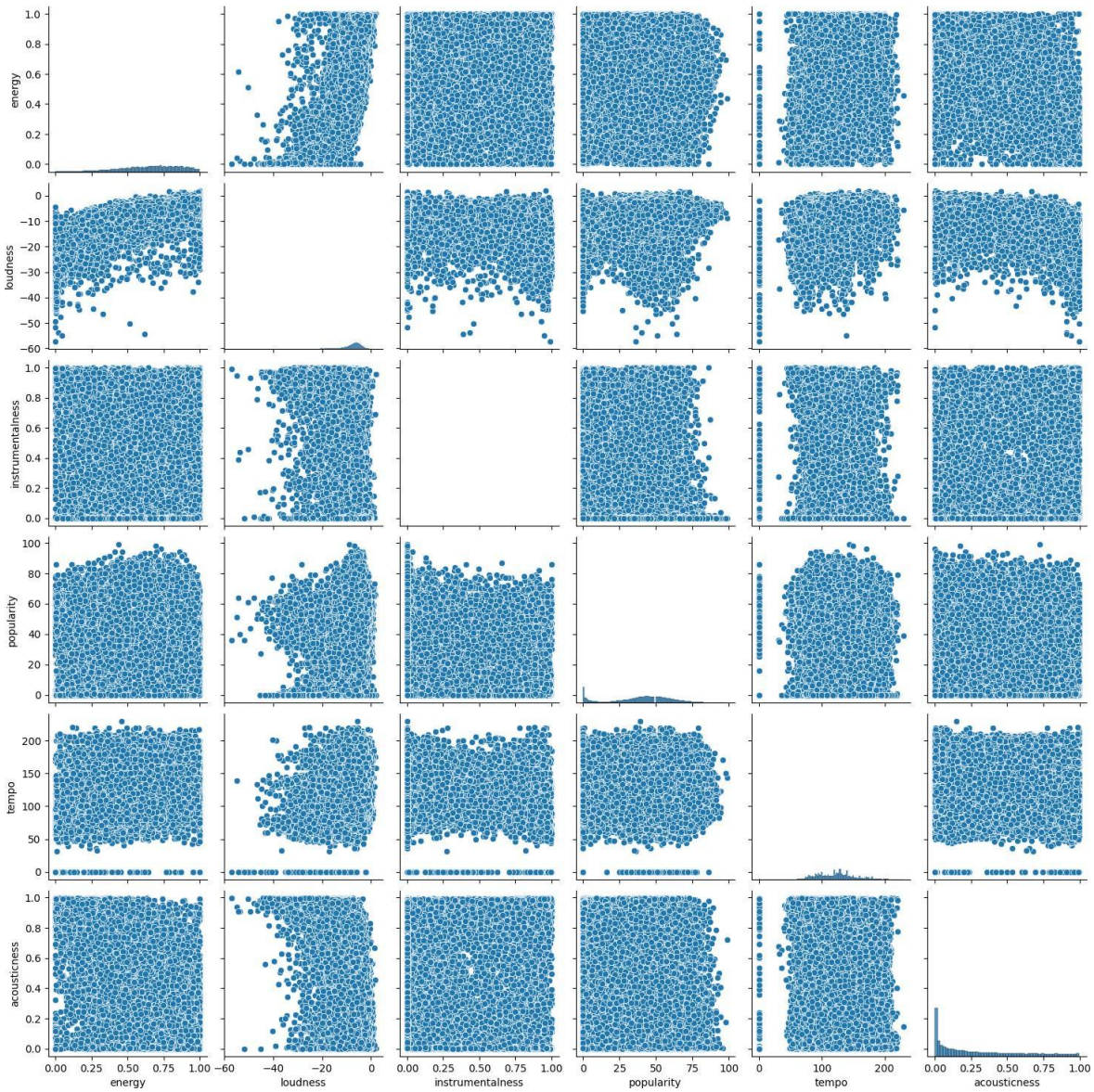

Figure 3.24 Pair plot for variables (2)

Instrumentalness has a slight negative correlation with **Popularity**, **energy**, **loudness** and has a slightly upward sloping regression line with **acousticness.**

It can be seen that the relationship between **tempo**, **energy** and **loudness** have a positive impact on each other. Therefore, the more tempo a song has, the louder and stronger the melody will be. In addition, the relationship between noise and energy also has a positive correlation. However, the relationship between **tempo** and **acousticness** is inversely proportional to each other.

In short, this pair plot helps us to know what kind of abstracts we can connect to each other to recommend songs that have the same abstracts.

3.5 EDA MULTIVARIATE ANALYSIS

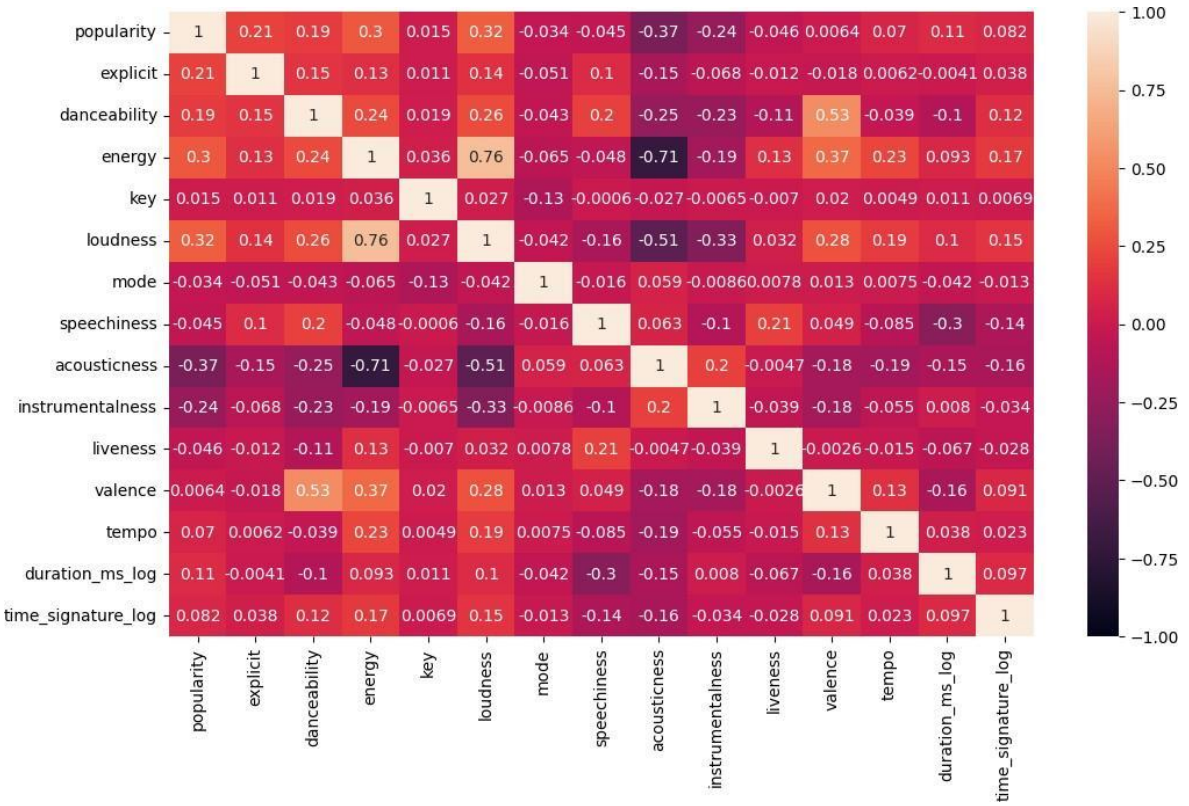

Figure 3.25 Heatmap chart for the dataset

Popularity has a positive correlation with **danceability, energy, acousticness, explicitness, and time signature.** This indicates that, during that period:

- People generally prefer music that is upbeat, energetic, and easy to dance to. This aligns with the popularity of genres like pop, dance, and electronic music.

- There is also an appreciation for music with raw, acoustic sounds, as seen in the popularity of folk, indie, and acoustic genres.
- Explicit content can attract listeners due to its boldness and taboo nature, although it may not be suitable for everyone.
- Simple and catchy rhythms are appealing to a wide audience, making songs with these characteristics more popular.

Explicit vs Danceability and **Speechiness** have a positive correlation. Music with explicit content often has lively, danceable melodies and a high level of spoken words. For example, rap and hip-hop music often contain sensitive content, use slang and vernacular language, and have energetic and danceable beats.

Danceability vs **Energy** and **Valence** are positively correlated. Danceable music typically has high energy and conveys positive emotions. For instance, EDM and pop dance music usually feature fast-paced, powerful beats, and evoke feelings of joy and excitement.

Energy vs **Loudness** and **Valence** are positively correlated. This can be explained by the fact that high-energy music tends to have high volume and convey positive emotions. For example, rock and metal music often have powerful, energetic sounds and evoke feelings of excitement and intensity. However, high-energy music tends to use fewer acoustic elements. Therefore, energy has a negative correlation with **Acousticness.**

Tempo vs **Energy** and **Loudness** are strongly positively correlated. Music with fast tempo, such as dance and electronic music, typically has high energy, a fast-paced rhythm, and high volume

To summarize, the last chapter delved into how to display patterns, correlations, and visualize data with the naked eye. Creating visually appealing statistics about the data is important because raw csv files are a bit confusing, especially in general terms. Through charts and diagrams, we will be able to better understand our data and get a better overview of it which will in turn help in building a music recommendation model in next chapter

.

Chapter 4 . DATA PROCESSING

At this stage of the process, the research paper demonstrates how to create a data frame for training models. This dataframe contains the song's vectors representing the properties of each song. “complete_feature_set”, which is the training data that we are going to create, will be based on these features from the main Spotify dataframe:

- **Technical description of the particular song:** including danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence and tempo by normalizing these float variables.
- **“release_date” and “popularity”** will be one-hot encoded to “year” and “popularity_red”.
- **“consolidates_genre_lists”** will be featuring to “genres” via TF-IDF.

4.1 NORMALIZE FLOAT VARIABLE

Comparing between loudness and tempo variables, it can be seen that the magnitudes of the raw data are so different, which is not ideal for creating a feature set because the model will skew towards one attribute versus another. The target is re-scaling all of these attributes in a range of [0, 0.2].

danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0.645	0.4450	0	-13.338	1	0.4510	0.674	0.7440	0.151	0.127	104.851
0.695	0.2630	0	-22.136	1	0.9570	0.797	0.0000	0.148	0.655	102.009
0.434	0.1770	1	-21.180	1	0.0512	0.994	0.0218	0.212	0.457	130.418
0.321	0.0946	7	-27.961	1	0.0504	0.995	0.9180	0.104	0.397	169.980
0.402	0.1580	3	-16.900	0	0.0390	0.989	0.1300	0.311	0.196	103.220

Figure 4.1 Data before processing

danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0.137045	0.1874	0.177905	0.011594	0.010221	0.147600	0.033467	0.1122	0.085281
0.093117	0.1300	0.178355	0.008447	0.134538	0.000000	0.007154	0.1310	0.132884
0.146356	0.1368	0.172615	0.006004	0.126707	0.000000	0.032064	0.1910	0.112253
0.085830	0.1308	0.174459	0.006605	0.067671	0.000000	0.018958	0.0414	0.121817
0.104656	0.0986	0.180124	0.005528	0.160442	0.000000	0.021443	0.0444	0.069588
0.107085	0.0958	0.173510	0.005859	0.062048	0.000000	0.022846	0.0236	0.118249
0.155466	0.1464	0.176075	0.025052	0.095181	0.000000	0.018076	0.0662	0.090446
0.170243	0.0984	0.168346	0.024017	0.115060	0.000000	0.019619	0.0888	0.078443

Figure 4.2 Data after processing in the range [0-0.2]

Normalizing these float variables is necessary in order to include these variables in the same set as the code shown below

```
float_cols=spotify_df.dtypes[spotify_df.dtypes=='float64']
.index.values
```

float_cols will contain an array of technical description from spotify_df since these columns have the data type 'float64'

```
    #function to build entire feature
set    def    create_feature_set(df,
float_cols):
        #tfidf genre lists
        #OHE   year   and
    popularity      #scale
    float columns
        floats = df[float_cols].reset_index(drop =
    True) scaler = MinMaxScaler()
     floats_scaled                                    =
pd.DataFrame(scaler.fit_transform(floats),    columns    =
floats.columns) 0.2
        #concatenate   all
    features #add song id
        return final
```

“float_cols” variable will get passed on when executing the function of creating a feature

set:

Focusing on “#scale float columns” piece of code. The code will scaling the whole values in chosen float columns to the range [0, 1] (via MinMaxScaler), and then multiplying all value to 0.2, making the maximum possible value 0.2 instead of 1 (which mean the scale right now is [0, 0.2]). This is an unusual choice in data preprocessing for machine learning but it is necessary since this feature is integrating with one-hot encoded and TF-IDF. It might be beneficial to ensure that the scaled numeric features do not dominate the feature space. This can help in maintaining a balance in the influence of different types of features on the model's predictions.

4.2 ONE-HOT ENCODE

Because the raw data have some feature that is useful for model, but its value is not ideal for training since it would taking so much computing resource. The point of this part is bucketed two features of the raw data before one-hot encode.

One hot encode is one method of converting data to prepare it for an algorithm and get a better recommendation. With one-hot, each categorical value will be converted into a new categorical column and assign a binary value of 1 or 0 to those columns. Each integer value is represented as a binary vector. All the values are zero, and the index is marked with a 1. Take an example in music recommendation scenario, this is the data before and after using one-hot encode:

Song	Year
Starboy	2016
Moth to a Flame	2022

One-hot encode →

Song	Year | 2016	Year | 2022
Starboy	1	0
Moth to a Flame	0	1

Figure 4.3 Example of one-hot Encode

In this model, there are two attribute that needs to be bucketed before one-hot encode: **release_date** > **year**: The raw data of “spotify_df” does have the “release_date” attribute as string data types with YYYY/MM/dd format. If one-hot encoding “release_date”, which is not recommended, it could result in a sparse representation, where most of the encoded features are zeros. This sparse representation may not be efficient in terms of memory usage and computational resources, especially if the dataset is large. Beside that, each unique date would become its own binary feature, resulting in potentially thousands of new features, which is not ideal for recommendation models in terms of performance and computational

requirements. By simplifying the attribute to only get the year from “release_date”, which means bucketing the track in a year, not the whole release date, helps mitigate these issues.

```
    # This will extract the year in "release_date" to a new
value into "year" attribute
```

```
    spotify_df['year']                                          =
spotify_df['release_date'].apply(lambda x: x.split('-')[0])
```

After executing this line of code, “spotify_df” now has a new attribute as “year”.

name	release_date	year
어떡하죠	2011-10-13	2011
아쉬운 마음인걸	2011-10-13	2011
My Island Home	2015-07-31	2015
Magic∞world	2011-11-30	2011
还可以爱吗	2013	2013

Figure 4.4 Year attribute added

popularity > **popularity_red**: The same reason with “release_date”, the songs will be grouped based on their popularity into buckets of width 5 and storing the bucket index in a new column 'popularity_red'. This transformation may be useful for simplifying the popularity scores and categorizing songs into broader popularity ranges, which could aid in the recommendation process.

Simplification of Data: Popularity is likely a continuous variable, meaning it can take on any value within a certain range. By bucketing it into five-point buckets, the data is simplified into discrete categories. This simplification can make it easier to analyze and model the data. Improving Model Generalization: Bucketing can help prevent overfitting by reducing the complexity of the data. Instead of trying to learn the exact relationship between popularity and other variables, the model only needs to learn the relationship within each bucket.

Handling Skewed Data: Popularity data might be skewed, with many songs having relatively low popularity and a few having very high popularity. Bucketing

can help address this skewness by grouping together songs with similar levels of popularity.

```
# create 5 point buckets for popularity
spotify_df['popularity_red'] =
spotify_df['popularity'].apply(lambda x: int(x/5))
```

After executing this line of code, "spotify_df" now has a new attribute as

name	popularity	popularity_red
어떡하죠	44	8
아쉬운 마음인걸	45	9
My Island Home	20	4
Magic∞world	35	7
还可以爱吗	34	6

Figure 4.5 New attribute "popularity_red" is added

"popularity_red".

After bucketed the songs on both "year" and "popularity_red", we now create a one-hot encode feature based on these two attributes:

```
    #simple function to create OHE
features #this gets passed later on
        def OHE_prep(df, column,
    new_name): """
        Create One Hot Encoded features of a specific column


        Parameters:
            df (pandas dataframe): Spotify
        Dataframe column (str): Column to be
        processed new_name (str): new column name
        to be used


        Returns:
            tf_df: One hot encoded features
        """
        tf_df                    =
    pd.get_dummies(df[column])
    feature_names = tf_df.columns
     tf_df.columns = [new_name + "|" + str(i) for i
in feature_names]
        tf_df.reset_index(drop = True, inplace =
    True) return tf_df
```

```
    #function to build entire feature
set     def     create_feature_set(df,
float_cols):
        """
    Process spotify df to create a final set of features that will
be used to generate recommendations
        Parameters:
            df (pandas dataframe): Spotify Dataframe
            float_cols (list(str)): List of float columns that will
        be
    scaled
        Returns:
            final: final set of features
        """
        #tfidf genre lists


        #OHE year and popularity
        year_OHE = OHE_prep(df, 'year','year') 0.5
        popularity_OHE = OHE_prep(df, 'popularity_red','pop') 0.15

    #scale float columns


        #concanenate     all

    features #add song id


        return final
```

Focusing on “#OHE year and popularity” piece of code. "year_OHE" has been multiplied to 0.5, the same thing with “popularity_OHE” when it has been multiplied to 0.15 because one- hot encoded features for “year” and “popularity_red” might result in binary features (0 or 1) after encoding. These binary features might have relatively high values compared to other features (TF-IDF and scaled float columns). Multiplying them by 0.5 and 0.15 could be a way to normalize their impact, ensuring they have a similar scale of influence as other features in the final feature set.

pop|1	pop|2	pop|3	pop|4	pop|5	pop|6	pop|7	pop|8	pop|9	pop|10	pop|11	pop|12	pop|13	pop|14	pop|15	pop|16	pop|17	pop|18	pop|19
0.0	0.0	0.0	0.0	0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00	0.15	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.00	0.15	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00	0.15	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.15	0.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.15	0.00	0.00	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 4.6 Bucketed OHE popularity feature(1)

year|2010	year|2011	year|2012	year|2013	year|2014	year|2015	year|2016	year|2017	year|2018	year|2019	year|2020	year|2021
0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 4.7 Bucketed OHE year feature(2)

4.3 CREATE TF-IDF FEATURES OFF OF ARTIST GENRES

In information retrieval, TF-IDF, short for term frequency–inverse document frequency, is a measure of importance of a word to a document in a collection or corpus, adjusted for the fact that some words appear more frequently in general.

Put TF-IDF in music recommendation scenarios, we will get the genres from the dataset via “consolidates_genre_lists” from Spotify dataframe and creates one large feature set that can be used to generate recommendations. More details of how TF-IDF calculate in this model:

Term Frequency (TF): Calculate the frequency of each genre in the dataset. This measures how often a genre appears in a list of genres.

Inverse Document Frequency (IDF): Calculate the inverse document frequency of each genre. This measures how unique or important a genre is across all songs. Genres that appear frequently across many songs will have a lower IDF, while those that appear rarely will have a higher IDF.

TF-IDF: Multiply the TF and IDF values to get the TF-IDF score for each genre in each song's list of genres.

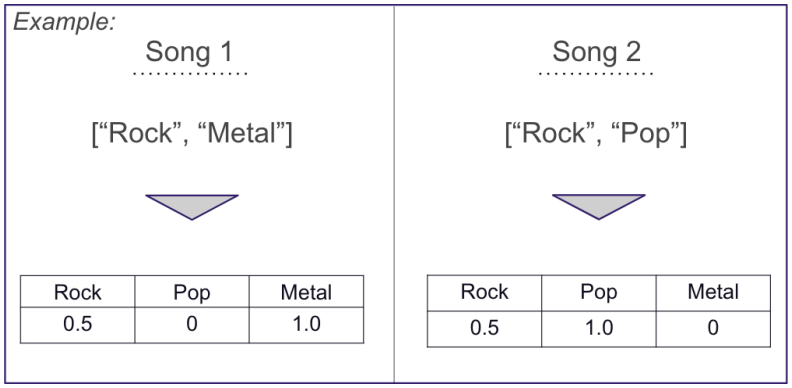

Figure 4.8 Example for TF-IDF

By creating TF-IDF features from the genres associated with songs, this recommendation system can leverage the genre information to better understand the musical characteristics of each song. This can lead to more accurate and relevant recommendations for users based on the user's playlist.

Below is the TF-IDF feature in the feature set function.

```
    #function to build entire feature set
def create_feature_set(df, float_cols):
        #tfidf genre lists
        tfidf                 =
    TfidfVectorizer()
    tfidf_matrix =
    tfidf.fit_transform(df['consolidates_genre_lists'].apply(lambda
x: " ".join(x)))
        genre_df                                        =
    pd.DataFrame(tfidf_matrix.toarray())
    genre_df.columns = ['genre|' + i for i in
     tfidf.get_feature_names_out()]
    genre_df.reset_index(drop=True, inplace=True)
        return final
```

And after creating 3 features above, we now have a function of complete feature set as shown below:

```
    #function   to   build   entire
feature          set          def
create_feature_set(df,
float_cols):
        """
        Process spotify df to create a final set of features
    that will be used to generate recommendations
        Parameters:
            df (pandas dataframe): Spotify Dataframe
            float_cols (list(str)): List of float columns that
        will be
    scaled
        Returns:
            final: final set of features
        """
        #tfidf genre lists
        tfidf          =
    TfidfVectorizer()
    tfidf_matrix =
    tfidf.fit_transform(df['consolidates_genre_lists'].apply(
lambda x: " ".join(x)))
        genre_df                          =
    pd.DataFrame(tfidf_matrix.toarray())
    genre_df.columns = ['genre|' + i for i in
     tfidf.get_feature_names_out()]
    genre_df.reset_index(drop=True,
    inplace=True) #OHE year and popularity
        year_OHE = OHE_prep(df, 'year','year') 0.5
        popularity_OHE  =  OHE_prep(df,  'popularity_red','pop')
    0.15 #scale float columns
        floats = df[float_cols].reset_index(drop = True)
```

```
        scaler = MinMaxScaler()
    floats_scaled                                          =
pd.DataFrame(scaler.fit_transform(floats),     columns     =
floats.columns) 0.2
        #concanenate all features
    final     =     pd.concat([genre_df,     floats_scaled,
popularity_OHE, year_OHE], axis = 1)
        #add     song     id
    final['id']=df['id'].valu
    es return final
```

Finally, executing this below code will create a feature set based on "spotify_df"

```
complete_feature_set      =      create_feature_set(spotify_df,
float_cols=float_cols)
```

In this chapter, our exploration of the data has been supplemented with visual representations using a variety of graphs, which help understand key patterns and distributions in the data set. At the same time, we perform normalization of float variables to ensure uniformity and scalability across the entire dataset. Additionally, we leveraged one-hot encoding techniques to convert categorical variables, such as artist genre, into a format suitable for modeling. Furthermore, by generating TF-IDF features from song-related genres, this recommender system can leverage genre information to better understand the musical characteristics of each song. Through these processes, we are equipped with a refined dataset to implement the recommendation system.

Chapter 5 . BUILDING RECOMMENDATION SYSTEM

In this chapter, Cosine Similarity algorithms will be used for building music recommendation model. This algorithm will be based on user's Spotify playlist to generate the songs that available in Spotify dataset.

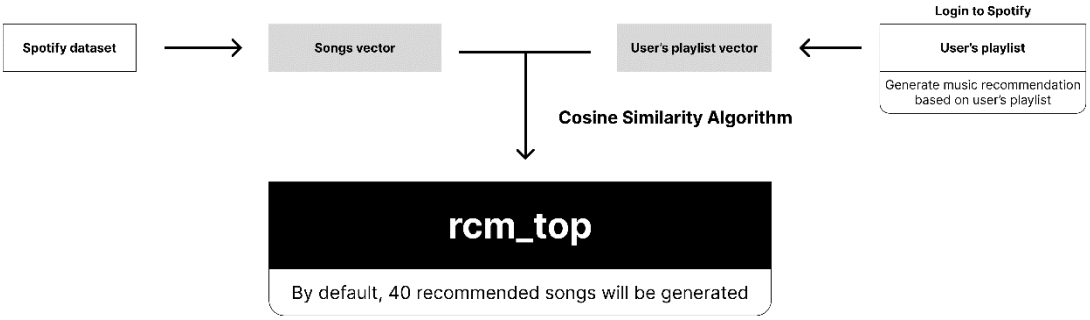

Figure 5.1 Illustrating how to build a music recommendation system

5.1 USING COSINE SIMILARITY ALGORITHM TO GENERATE RECOMMENDATION

Cosine Similarity is an algorithm used to measure the degree of similarity between two vectors in vector space. It works on the principle of calculating the cosine of the angle between two vectors. Generating a good music recommendation required the recommended song has to similar to the songs that already in playlist, and this is where Cosine Similarity algorithm showing its strength.

In this model, Cosine Similarity algorithms will generate music recommendation based on the angle calculation between two vectors: user's playlist vector and each vector of the song in Spotify dataset.

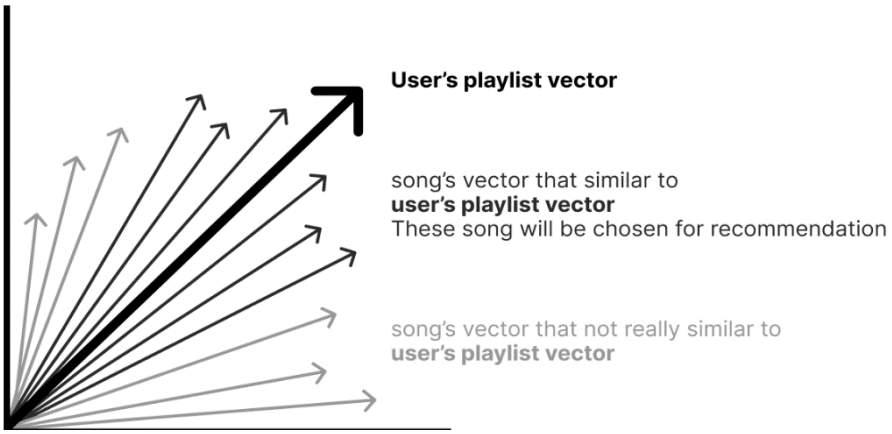

Figure 5.2 Illustration of how Cosine Similarity works in music recommendation

5.2 DEVELOPMENT OF COSINE SIMILARITY-BASED MODEL

Deep down to the code, we write a function "generate_playlist_recos", which is Cosine Similarity Algorithms, to generate playlist recommendations from a Spotify dataset. Below is the explanation:

```
def generate_playlist_recos(df, features, nonplaylist_features):
    """
    Pull songs from a specific playlist.

    Parameters:
        df (pandas dataframe): spotify dataframe
        features (pandas series): summarized playlist feature
        nonplaylist_features (pandas dataframe): feature set of songs that are not in the selected playlist

    Returns:
        non_playlist_df_top_40: Top 40 recommendations for that playlist
    """

    non_playlist_df = df[df['id'].isin(nonplaylist_features['id'].values)]
    non_playlist_df['sim'] = cosine_similarity(nonplaylist_features.drop('id', axis = 1).values, features.values.reshape(1, -1))[:,0]
    non_playlist_df_top_40 = non_playlist_df.sort_values('sim',ascending = False).head(40)
    non_playlist_df_top_40['url'] = non_playlist_df_top_40['id'].apply(lambda x: sp.track(x)['album']['images'][1]['url'])

    return non_playlist_df_top_40
```

Figure 5.3 Function "generate_playlist_recos"

df: The first parameter is a dataframe of Spotify data, containing information about songs and playlists. In this project, it is "spotify_df"

features: The second parameter is a Series (column) of DataFrame, containing summary information about the characteristics of the selected playlist. In this project, it is "complete_feature_set_myplaylist_vector"

nonplaylist_features: The third parameter is a DataFrame containing features of songs that are not in the selected playlist to avoid the model re-generate the songs that already in the playlist. In this project, it is "complete_feature_set_nonmyplaylist"

cosine_similarity: The cosine_similarity function is used to calculate the similarity between nonplaylist_features and features. The result is a 1D array (vector) of similarity values.

It works according to these steps:

- Filter songs out of the list: Based on the list nonplaylist_features contains information about songs that are not in your favorite playlist (features), the function extracts those songs from df (all song data)
- Cosine similarity calculation: Use the library cosine_similarity, the function calculates the similarity between the feature vector of favorite playlists (features) with the feature vector of each song in addition to the list (nonplaylist_features). The results are saved in a new column ‘sim ' belong to non_playlist_df.
- Ranking and selection: Based on cosine similarity score ('sim’), the function sorts the songs outside the list in descending order. It then selects the 40 songs with the highest similarity scores (non_playlist_df_top_40).

Consequently, the function generate_playlist_recos takes as input a DataFrame of Spotify data, summary information about the features of the selected playlist, and features of songs not in the playlist. This function returns 40 suggested songs for the playlist based on similarity between features.

5.3 EVALUATE THE MODEL

5.3.1 Result from Cosine Similarity Algorithms

After created the model, this is the process of checking how it’s performing. This is done by testing the performance of the model on previously unseen data. The unseen data that will be used for the testing is user’s Spotify playlist dataset. In order to gather user’s Spotify playlist data, it is necessary to login to Spotify account, then choose one of user’s playlist that available in user’s Spotify account for model testing. The model was tested with 3 different playlists with 3 separate music genres to evaluate the model's music recommendation accuracy.

The model will be tested with rap songs, as this is a more specific genre than pop, which is already very popular. Choosing a rap genre will require the model to recommend rap songs and more rap-specific songs. While this playlist is primarily rap

songs, it does have some R&B and pop songs, the purpose of this was to see how the model would handle a playlist with sub-genres besides the main genre is rap.

Executing the code below will run the model test, generate recommendation

```
rcm_top = generate_playlist_recos(spotify_df, complete_feature_set_myplaylist_vector, complete_feature_set_nonmyplaylist)
```

Figure 5.4 Recommending playlists for users based on favorite playlists

[melodic_rap, dfw_rap, pop_rap, rap, trap]	16	2019	Post Malone	Sunflower - Spider-Man: Into the Spider-Verse
[melodic_rap, dfw_rap, pop_rap, hip_hop, rap, ...	16	2016	Post Malone	Congratulations
[rap, atl_hip_hop, trap]	16	2020	Playboi Carti	Sky
[pop_rap, hip_hop, rap, southern_hip_hop, slap...	14	2018	Metro Boomin	Overdue (with Travis Scott)
[canadian_contemporary_r&b, reggaeton, pop, re...	16	2020	Maluma	Hawái - Remix

Figure 5.5 Input: User's playlist that contain rap song

[Gucci Mane, Drake]	Gucci ManeBoth (feat. Drake)	5tFep7dXGd7vEJ668wTPux	[dirty_south_rap, pop_rap, toronto_rap, hip_ho...
[21 Savage, Metro Boomin]	21 SavageRunnin	5SWnsxjhdcEDc7LJjq9UHk	[pop_rap, hip_hop, rap, atl_hip_hop, southern_...
[Quavo, Drake]	QuavoFLIP THE SWITCH (feat. Drake)	1rIlrbWboTRGeKfHhgbJRZ	[melodic_rap, pop_rap, toronto_rap, hip_hop, r...
[Drake, Quavo, Travis Scott]	DrakePortland	2bjwRfXMk4uRgOD9IBYI9h	[melodic_rap, pop_rap, toronto_rap, hip_hop, r...
[Young Thug, Travis Scott]	Young Thugpick up the phone	20dP2DaMHIAmwWAbp7peSr	[melodic_rap, atl_trap, pop_rap, hip_hop, rap,...
[Migos, Drake]	MigosWalk It Talk It	6n4U3TlzUGhdSFbUUhTvLP	[pop_rap, toronto_rap, hip_hop, rap, southern_...

Figure 5.6 Output: Recommendations based on rap songs

artists_upd	artists_song	id_x	consolidates_genre_lists
[Future, The Weeknd]	FutureLow Life (feat. The Weeknd)	7EiZI6JVHIIARrX9PUvAdX	[canadian_contemporary_r&b, pop, pop_rap, rap,...
[The Weeknd, Future]	The WeekndAll I Know	0NWqNXBJTpXbkI5rPWNy3p	[canadian_contemporary_r&b, pop, pop_rap, rap,...
[SZA, The Weeknd, Travis Scott, Game of Thrones]	SZAPower is Power (feat. The Weeknd & Travis S...	4cbdPT6uaBOgOQe3fLMofl	[canadian_contemporary_r&b, pop, pop_rap, rap,...
[Justin Bieber, Quavo]	Justin BieberIntentions (feat. Quavo)	4umIPjkehX1r7uhmGvXiSV	[melodic_rap, pop, post-teen_pop, pop_rap, hip...

Figure 5.7 Output: The model also suggested rap songs with sub-genres

It can be seen from the output that the model focused on searching for rap songs based on the previous playlist. Not only that, although there are some songs from the previous playlist that are pop and R&B, it can still find some rap songs with pop sub-genres (Justin Bieber - Intentions ft. Quavo with rap and pop genres) or some R&B sub-genre rap songs (The Weeknd - All I Know ft. Future with rap and R&B genres).

5.3.2 Compare with another algorithms

5.3.2.1 Euclidean Distance

The Euclidean Distance algorithm is a similarity measure that calculates the "straight-line" distance between two points (music features) to determine their similarity. In contrast, Cosine Similarity calculates similarity based on the angle between vectors. Euclidean Distance measures the distance between two points in a straight line, and the smaller the distance, the higher the similarity

Although the two algorithms have different calculation methods, the songs recommended by Euclidean Distance are quite similar to those recommended by Cosine Similarity.

5.3.2.2 Jaccard Similarity

Jaccard Similarity is an algorithm that is based on the similarity between two sets (music features) to recommend music related to the user's taste.

The goal of this system is to recommend music based on genre. However, within the same genre, there can be various melodies. For example, within the pop music genre, there can be sub-genres like dance-pop (disco, dance-oriented pop with an upbeat rhythm) or operatic pop (classical-style pop with a slower tempo). To increase accuracy, the system also

considers other attributes besides music genre, such as technical characteristics of the music (tempo, liveness, speechness, etc.) or the year of release. That is the reason why the dataset has been engineered to prioritize genre by assigning it a higher score than other features. This continuous data is better suited for use with Cosine Similarity or Euclidean Distance.

	name	popularity	duration_ms	explicit	artists
	Who Hurt You?	73	231964	1	['Daniel Caesar']
	end of summer	45	185691	1	['slchld', 'Nathania']
	say what's on your mind	58	173740	0	['slchld']
	Rush (feat. Jessie Reyez)	66	198351	0	['Lewis Capaldi', 'Jessie Reyez']
	Save Your Tears	67	215627	1	['The Weeknd']
	Out Of Love	79	227693	0	['Alessia Cara']

	name	popularity	duration_ms	explicit	artists
	Who Hurt You?	73	231964	1	['Daniel Caesar']
	end of summer	45	185691	1	['slchld', 'Nathania']
	say what's on your mind	58	173740	0	['slchld']
	Rush (feat. Jessie Reyez)	66	198351	0	['Lewis Capaldi', 'Jessie Reyez']
	Out Of Love	79	227693	0	['Alessia Cara']
	Save Your Tears	67	215627	1	['The Weeknd']

Figure 5.8 Left: Euclidean Distance, right: Cosine Similarity

However, the Jaccard algorithm can only be used with binary data (0 and 1), which means it treats all song features as equal instead of prioritizing "genre." This can lead to confusion when recommending operatic pop songs based on a dance-pop playlist, which are both related to the pop genre but have different melodies. Additionally, the data has been engineered as continuous data, not binary data, so Jaccard cannot be used.

Chapter 6 . CONCLUSION

6.1 SUMMARIZE THE FINDINGS

Through the research process, the project has achieved some main achievements as follows:

Engineering features of Music data for each song. These descriptions help the system understand what type of music each song is.

Using the Cosine algorithm to create playlist-based recommendation sets. This way, the system will recommend music that matches user’s preferences.

To understand more about the data, we used a technique called Exploratory Data Analysis (EDA). This involves looking closely at the data using graphs such as histograms, box plots, pie charts, and heat maps. This way, we can see patterns and connections between different aspects of songs, like their genre or tempo.

6.2 LIMITATIONS AND FUTURE DEVELOPMENT

6.2.1 Limitations

Because the time to complete the project is limited, the project still has shortcomings

- The current model is the reliance on a data range to generate recommendations. Since the data is too large to analyze (contain songs from 1900 – 2020, around 600.000 songs), which is not ideal for computer resource, so we reducing the range from 2010– 2020 (around 114.000 songs).
- When someone add a song that is not in the release year 2010-2020, the songs might not available to recommend because the song doesn't have in the dataset.
- If the input data isn't diverse enough, the model might not be fair or accurate.

6.2.2 Future improvements

One limitation of the current system is the reliance on a specific dataset or set of features to generate recommendations. To overcome this limitation, future iterations could explore incorporating additional data sources, such as a user's listening history, social media activity, or music reviews. By integrating diverse data sources, the system can better understand user preferences and provide more accurate and diverse recommendations.

Another area that needs improvement is integrating the recommendation system with popular music streaming platforms such as Spotify, Apple Music or YouTube Music. This integration will allow the system to access real-time user data and take advantage of platform-specific features, such as personalized playlists, song likes and listening habits. By seamlessly integrating with existing platforms, recommendation systems can reach a broader audience and provide a more seamless user experience.

Although the current system uses a Cosine Similarity algorithm, there is scope to explore more advanced recommendation techniques. Future developments may involve testing machine learning algorithms, such as collaborative filtering, matrix factorization or deep learning models, to improve the accuracy and efficiency of recommendations. Additionally, techniques such as hybrid recommendation systems, which combine multiple algorithms or data sources, can further improve system performance and adaptability.

REFERENCES

[1] Bức tranh toàn cảnh về làng nhạc Indie Việt Nam (P.1). (2022, July 11). Spiderum. Retrieved March 21, 2024, from https://spiderum.com/bai-dang/Buc-tranh-toan-canh-ve-lang-nhac-Indie-Viet-Nam-P1-K9OYdDcf7N57

[2] Bức tranh toàn cảnh về làng nhạc Indie Việt Nam (P.2). (2022, July 18). Spiderum. Retrieved March 21, 2024, from https://spiderum.com/bai-dang/Buc-tranh-toan-canh-ve-lang-nhac-Indie-Viet-Nam-P2-oLgUzk13Enzu

[3] Fawcett, A. (2021, February 11). *Data Science in 5 Minutes: What is One Hot Encoding?* Educative.io. Retrieved March 21, 2024, from https://www.educative.io/blog/one-hot- encoding

[4] Feature engineering. (n.d.). Wikipedia. Retrieved March 21, 2024, from https://en.wikipedia.org/wiki/Feature_engineering

[5] Feature Engineering for Recommender Systems: Techniques and Examples. (2023, November 21). LinkedIn. Retrieved March 21, 2024, from https://www.linkedin.com/advice/3/how-can-you-engineer-features-recommender systems- qpane

[6] Feature (machine learning). (n.d.). Wikipedia. Retrieved March 21, 2024, from https://en.wikipedia.org/wiki/Feature_(machine_learning) *tf–idf*. (n.d.). Wikipedia. Retrieved March 21, 2024, from https://en.wikipedia.org/wiki/TF-IDF

[7] YAMAC EREN AY. (n.d.). *Spotify Dataset 1921-2020, 600k+ Tracks*. Kaggle. Retrieved March 24, 2024, from https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks