## Artificial Intelligence and Machine Learning

Project Report

Semester-IV (Batch-2022)

MUSIC RECOMMENDATION SYSTEM

A red and white sign

Description automatically generated with low confidence

**Supervised By: Submitted By:**

Dr. Kiran Deep Singh Phalya Jhamb 2210990649

Rachit Walia 2210990695

Raghav Thakur 2210990701

Pratham Tandon 2210990676

Priyanshu 2210990685

**Department of Computer Science and Engineering**

## Chitkara University Institute of Engineering & Technology,

## Chitkara University, Punjab

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Title** | **Page** |
| 1. | Abstract | 2 |
| 2. | Introduction:  2.1 Background  2.2 Objective  2.3 Significance | 4-6 |
| 3. | Problem Definition and Requirement:  3.1 Problem Statement  3.2 Software Requirement  3.3 Hardware Requirements | 7-8 |
| 4. | Proposed Design/Methodology | 9-13 |
| 5. | Screenshots | 14-37 |
| 6. | Result | 38 |
| 7. | Conclusion | 39 |
| 8. | Reference | 40 |

1. **Abstract**

The objective of this project is to develop a music recommendation system. The system will determine the musical preferences of the users based on the analysis of their interaction during use. This way the system is able to estimate what artist or group would match user preferences to the user at a given time. It has been taken into account the fact that we do not always want to hear the same artists or genres, we do have favorite bands, but sometimes we appreciate a surprise, a new discovery.

The system uses music information collected from online music services that make available their music catalogs for developers’ community to be used inside new applications. It has been created a web system that connects to the music service providers to obtain these musical catalogs. This system implements the necessary communication features to use this information in the client web browser. This system helps users discover new artists, albums or songs making the musical catalog available for listening.

The dynamic characteristics of the interface allows the user to browse music collections while listening to a song or playing a video. The user will receive information related to her interaction patterns in form of recommendations of items. These items will probably match user preferences and they are shown as the user interacts with the system and only when it has enough information about user preferences.

1. **Introduction**

**In today's digital age, the abundance of music available at our fingertips presents both a blessing and a challenge. While we have access to an unparalleled variety of music across genres and cultures, navigating through this vast sea of choices can be overwhelming. This is where music recommendation systems step in, utilizing algorithms and data analysis to assist users in discovering new music tailored to their tastes and preferences. In this report, we delve into the development and implementation of a music recommendation system using Python, outlining its background, objective, and significance in the realm of modern music consumption.**

**2.1 Background**

**The concept of music recommendation systems has its roots in the early days of the internet, where platforms like Pandora pioneered personalized radio stations based on user feedback. Since then, advancements in machine learning and data analytics have revolutionized the way music is recommended and consumed. Today, companies like Spotify, Apple Music, and YouTube Music employ sophisticated algorithms to curate playlists and suggest tracks, enhancing user experience and engagement.**

**These recommendation systems rely on a combination of collaborative filtering, content-based filtering, and hybrid approaches to analyze user behavior, preferences, and music attributes. Collaborative filtering techniques leverage the wisdom of the crowd by recommending items based on the preferences of similar users. Content-based filtering, on the other hand, focuses on the characteristics of items themselves, recommending music with similar acoustic features or metadata. Hybrid approaches combine these methods to provide more accurate and diverse recommendations.**

**Python, with its extensive libraries for data analysis and machine learning such as Pandas, NumPy, and Scikit-learn, has become a popular choice for building recommendation systems due to its flexibility and ease of use. By harnessing the power of Python and leveraging relevant data sources, we can develop a music recommendation system that enhances user satisfaction and engagement.**

**2.2 Objective**

**The primary objective of this project is to design and implement a music recommendation system using Python that provides personalized recommendations to users based on their music preferences. Specifically, we aim to achieve the following goals:**

**1. Data Collection and Preprocessing: Gather a comprehensive dataset of music tracks, including metadata such as genre, artist, album, and user interactions (e.g., listening history, likes, dislikes).**

**2. Feature Engineering: Extract relevant features from the music dataset, including acoustic attributes (e.g., tempo, energy, danceability), textual data (e.g., artist bios, album descriptions), and user behavior metrics.**

**3. Algorithm Selection and Implementation: Explore and implement various recommendation algorithms, including collaborative filtering, content-based filtering, and hybrid models, to generate accurate and diverse music recommendations.**

**4. Evaluation and Optimization: Evaluate the performance of the recommendation system using metrics such as precision, recall, and user satisfaction. Fine-tune the algorithms and parameters to optimize recommendation quality and relevance.**

**5. Deployment and Integration: Deploy the recommendation system as a user-friendly application or API, integrating it with existing music streaming platforms or standalone services for seamless user experience.**

**By accomplishing these objectives, we aim to create a robust and scalable music recommendation system that enhances user engagement, promotes music discovery, and enriches the overall listening experience.**

**2.3 Significance**

**The significance of developing a music recommendation system using Python extends beyond mere convenience for users. It plays a crucial role in shaping the future of music consumption and distribution in several ways:**

**1. Personalized User Experience: By analyzing user preferences and behavior, the recommendation system tailors music suggestions to individual tastes, fostering a more personalized and enjoyable listening experience. This not only increases user satisfaction but also encourages prolonged engagement with music streaming platforms.**

**2. Music Discovery and Diversity: The recommendation system introduces users to a diverse range of music genres, artists, and tracks they may not have discovered otherwise. By surfacing hidden gems and niche content, it promotes exploration and appreciation of musical diversity, enriching the cultural landscape of the digital music ecosystem.**

**3. Artist Promotion and Exposure: For emerging artists and independent musicians, recommendation systems serve as a powerful tool for gaining exposure and reaching new audiences. By recommending their music to relevant users based on similarities with established artists or genres, the system helps amplify their presence and support their careers.**

**4. Data-driven Insights: The analysis of user interactions and preferences within the recommendation system generates valuable insights for music industry stakeholders, including artists, labels, and streaming platforms. By understanding consumer behavior and trends, they can make informed decisions regarding content creation, promotion strategies, and platform optimization.**

**In summary, the development of a music recommendation system using Python represents a significant advancement in the field of music technology, with far-reaching implications for both users and industry stakeholders. By harnessing the power of data analysis and machine learning, we can create a more personalized, diverse, and engaging music listening experience for audiences worldwide.**

1. **Problem Definition and Requirements**

**3.1 Problem Statement**

**The problem at hand revolves around the need to alleviate the overwhelming abundance of music choices faced by users in today's digital landscape. With an ever-expanding catalog of songs and albums available across various streaming platforms, users often struggle to discover new music that aligns with their preferences and interests. This leads to frustration, decision paralysis, and ultimately, a less satisfying music listening experience.**

**To address this problem effectively, the following requirements must be met by the music recommendation system:**

**1.The system must deliver personalized recommendations tailored to each user's unique music tastes, preferences, and listening history. By analyzing user interactions and behavior, it should adapt and refine its recommendations over time to reflect evolving preferences.**

**2. Recommendations should be both accurate and diverse, striking a balance between familiar favorites and new discoveries. The system should avoid over-reliance on popular or mainstream content, instead surfacing a wide range of genres, artists, and tracks to cater to diverse tastes.**

**3. The user base grows and the music catalog expands, the system must be scalable and capable of handling large volumes of data efficiently. It should deliver recommendations in real-time or near-real-time, ensuring a seamless user experience without significant delays or bottlenecks.**

**4. Users should be provided with insights into how recommendations are generated, including the underlying algorithms and factors influencing each recommendation. This enhances trust and confidence in the system, empowering users to make informed decisions about their music choices.**

**5.The recommendation system should seamlessly integrate with existing music streaming platforms or standalone applications, allowing users to access recommendations within their preferred environment. It should be compatible with various devices and operating systems to maximize accessibility.**

**3.2 Software Requirements:**

**1. Programming Language:** Python 3.x for machine learning and data analysis.

**2. Libraries:** NumPy for numerical computations, pandas for data manipulation, and scikit-learn for machine learning algorithms.

**3.** **IDE:** Any text editor or Integrated Development Environment (IDE) like VSCode, Sublime Text, or Atom.

**4.** **Data Visualization:** Matplotlib and Seaborn for creating visualizations. 5. Version Control: Git for collaborative development and tracking changes.

**5. Documentation:** Markdown syntax for project documentation.

* 1. **Hardware Requirements**

**Processor:** A multi-core processor (e.g., Intel Core i5 or AMD Ryzen 5) for efficient computational tasks during data processing and model training.

**RAM**: Minimum 8 GB RAM for smooth execution of machine learning algorithms, especially with large datasets. Additional RAM may be beneficial for handling larger datasets.

**Storage**: Adequate storage space for datasets, project files, and software libraries. A Solid-State Drive (SSD) is recommended for faster data access.

**GPU**: While not mandatory, a dedicated GPU (e.g., NVIDIA GeForce or AMD Radeon) can accelerate model training, especially for deep learning algorithms.

**Operating System**: Choose from Windows, macOS, or Linux distributions like Ubuntu based on personal preference and software compatibility.

**Internet Connection**: A stable internet connection is necessary for downloading datasets, software, and accessing online resources.

**Peripheral Devices:** Standard input/output devices like keyboard, mouse, and monitor are essential. External storage devices aid in storing and sharing project files.

**By meeting these requirements, the music recommendation system can effectively address the challenge of music discovery, providing users with personalized, accurate, and diverse recommendations that enhance their overall listening experience.**

1. **Proposed Design/ Methodology**

**A music recommendation system is built upon a comprehensive methodology that encompasses various stages, from data collection to deployment. The initial phase involves gathering a vast dataset comprising diverse music-related information, ranging from user preferences and listening history to intricate song metadata and social interactions. This data forms the foundation upon which the recommendation system operates, providing the raw material necessary for analysis and modeling.**

**Interface Result**

Recommendation Engine

Recommendation List

Acquire User Query

**Fig. (1) General Framework of Recommendation System**

**Once collected, the data undergoes rigorous preprocessing to ensure its integrity and suitability for further analysis. This involves cleaning the dataset to handle missing values, duplicates, and inconsistencies, as well as structuring the data in a standardized format conducive to modeling. Following preprocessing, relevant features are extracted from the dataset, including genre, artist, tempo, mood, and user preferences. These features serve as the building blocks for generating personalized recommendations tailored to individual users.**

**The next crucial step in the methodology is the selection of an appropriate recommendation algorithm. This decision is informed by the characteristics of the dataset, as well as the desired outcomes of the recommendation system. Common approaches include collaborative filtering, content-based filtering, and hybrid methods, each offering unique advantages and challenges. Once a model is selected, it undergoes training using historical user interactions and feedback. This iterative process involves optimizing model parameters and evaluating performance metrics to ensure the accuracy and relevance of recommendations.**

**With the model trained, the recommendation system is poised to generate personalized recommendations for users based on their preferences, listening history, and contextual information. These recommendations are continuously refined and improved through ongoing evaluation and iteration. Key evaluation metrics such as precision, recall, and user satisfaction are used to assess the quality and effectiveness of the recommendation system, guiding further enhancements and optimizations.**

**Finally, the recommendation system is deployed in a production environment, seamlessly integrated with music streaming platforms or other relevant applications. Continuous monitoring and updates ensure that the system remains responsive to changing user preferences and trends, delivering tailored and engaging music suggestions that enhance the overall listening experience. In summary, by adhering to this comprehensive methodology, music recommendation systems can**

**Once collected, the data undergoes rigorous preprocessing to ensure its integrity and suitability for further analysis. This involves cleaning the dataset to handle missing values, duplicates, and inconsistencies, as well as structuring the data in a standardized format conducive to modeling. Following preprocessing, relevant features are extracted from the dataset, including genre, artist, tempo, mood, and user preferences. These features serve as the building blocks for generating personalized recommendations tailored to individual users.**

**The next crucial step in the methodology is the selection of an appropriate recommendation algorithm. This decision is informed by the characteristics of the dataset, as well as the desired outcomes of the recommendation system. Common approaches include collaborative filtering, content-based filtering, and hybrid methods, each offering unique advantages and challenges. Once a model is selected, it undergoes training using historical user interactions and feedback. This iterative process involves optimizing model parameters and evaluating performance metrics to ensure the accuracy and relevance of recommendations.**

**With the model trained, the recommendation system is poised to generate personalized recommendations for users based on their preferences, listening history, and contextual information. These recommendations are continuously refined and improved through ongoing evaluation and iteration. Key evaluation metrics such as precision, recall, and user satisfaction are used to assess the quality and effectiveness of the recommendation system, guiding further enhancements and optimizations.**

**Finally, the recommendation system is deployed in a production environment, seamlessly integrated with music streaming platforms or other relevant applications. Continuous monitoring and updates ensure that the system remains responsive to changing user preferences and trends, delivering tailored and engaging music suggestions that enhance the overall listening experience. In summary, by adhering to this comprehensive methodology, music recommendation systems can effectively facilitate the discovery of new artists and songs while promoting user engagement and satisfaction.**

Start

Import Libraries

Suppress Warnings

Load Data

Prepare Data for Visualization

End

Visualize Feature Correlation

Display Info

Fig. (2) EDA and Visualization Workflow

The flowchart outlines the process of exploring and visualizing data from the Spotify dataset. It begins by importing necessary libraries and loading the dataset. Information about the datasets is displayed. Feature names are defined, and data is prepared for visualization. Finally, the flowchart concludes after displaying the correlation heatmap.

Select Numeric Features for ‘X’

Define Cluster Pipeline

End

Create Scatter Plot Using Plotly Express

Create Data Frame ‘projection’

Perform t-SNE Pipeline

Define t-SNE pipeline

Fit Cluster Pipeline to ‘X’

Start

Import Libraries

Fig. (3) Clustering Genres with K-means

This flowchart outlines the steps, including importing necessary libraries, defining a pipeline for K-Means clustering, fitting the pipeline to the data, defining a pipeline for t-SNE embedding, performing t-SNE embedding, creating a Data Frame for visualization, and generating a scatter plot using Plotly.

Perform PCA dimensionality reduction on ‘X’

Start

Create Scatter Plot using plotly

Create Data Frames ‘Projection’

Select Numeric features for ‘X’

Define Song Cluster Pipeline

Import libraries

Define PCA pipeline

End

Fig. (4) Clustering Songs with K-means

This code performs clustering and dimensionality reduction on a dataset of songs. First, it creates a pipeline (`song\_cluster\_pipeline`) for KMeans clustering with 20 clusters, scaling the numeric features with Standard Scaler. It selects numeric columns from the dataset (`X`) and fits the clustering pipeline to this data, predicting cluster labels for each song. Then, it creates another pipeline (`pca\_pipeline`) for PCA dimensionality reduction to 2 components. It fits this pipeline to the same dataset `X`, transforms the data, and constructs a DataFrame (`projection`) with the transformed coordinates. This DataFrame includes columns for the x and y coordinates, song titles, and cluster labels. Finally, it generates a scatter plot using Plotly Express, coloring points by cluster and displaying hover data with coordinates and song titles.

1. **Screenshots**

In conclusion, the development of a music recommendation system using Python offers significant benefits in enhancing user engagement and satisfaction. By leveraging machine learning algorithms and data-driven approaches, personalized song suggestions can be generated based on individual preferences and behavior. With careful consideration of system requirements, data processing, model training, and user interface design, the system can deliver tailored recommendations, thereby enriching the overall listening experience and promoting exploration of diverse musical content.

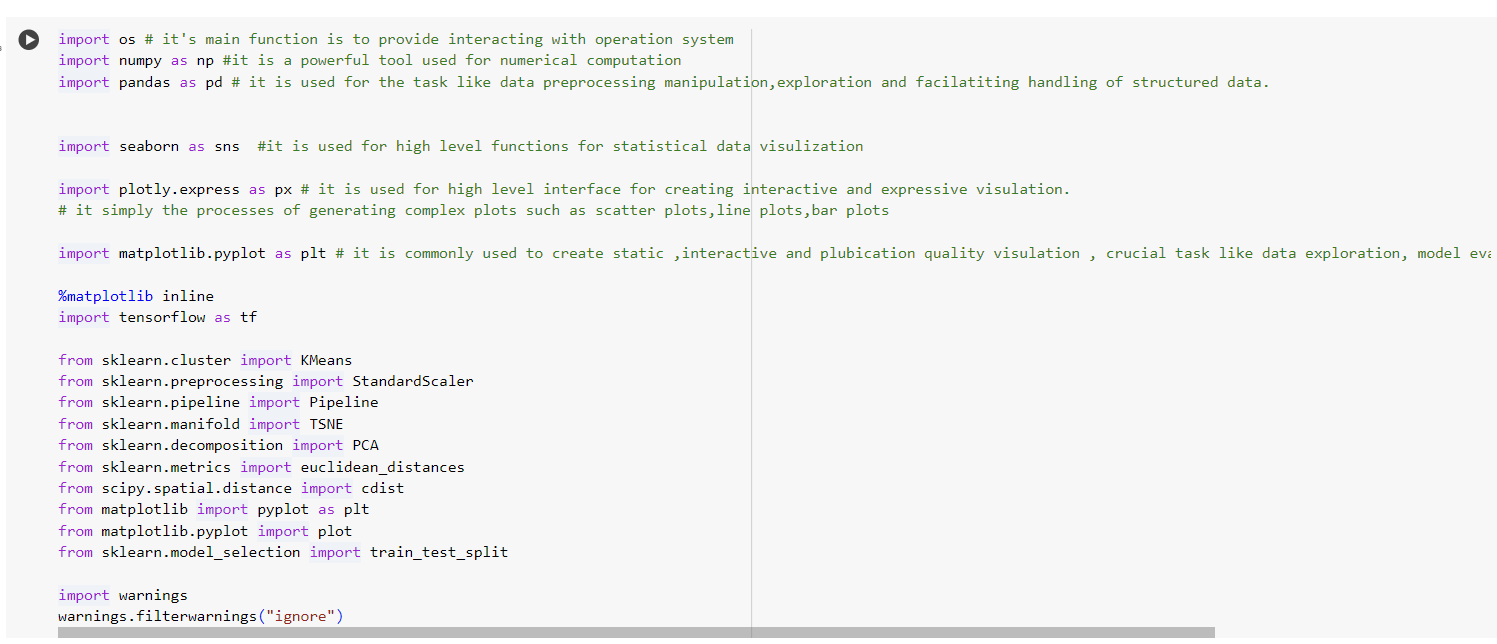
****

Fig. (5)

This Python code snippet imports necessary libraries for data analysis and visualization, as well as machine learning with TensorFlow. It includes modules for data manipulation (NumPy, Pandas), visualization (Seaborn, Plotly, Matplotlib), machine learning (Scikit-learn), and neural networks (TensorFlow). Additionally, it suppresses warnings. The code sets up a data analysis environment, including tools for clustering (KMeans), feature scaling (StandardScaler), dimensionality reduction (PCA, t-SNE), and model evaluation (euclidean\_distances). It also imports train\_test\_split for splitting data into training and testing sets.

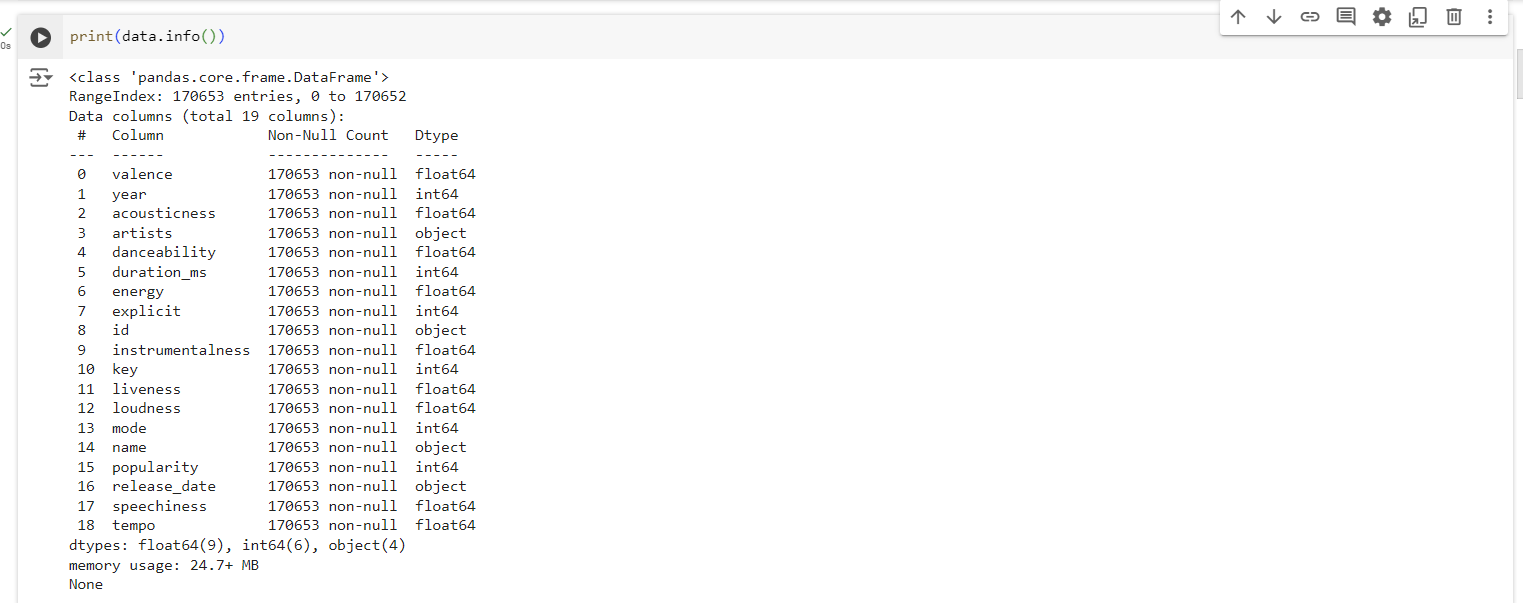
****

Fig. (6)

The `print(data.info())` line of code provides a summary of the data DataFrame. It prints information including the number of entries, the number of columns, each column's name, data type, and the number of non-null values. This summary is useful for understanding the structure of the DataFrame, checking for missing values, and confirming the data types of each column, aiding in further data analysis and preprocessing.

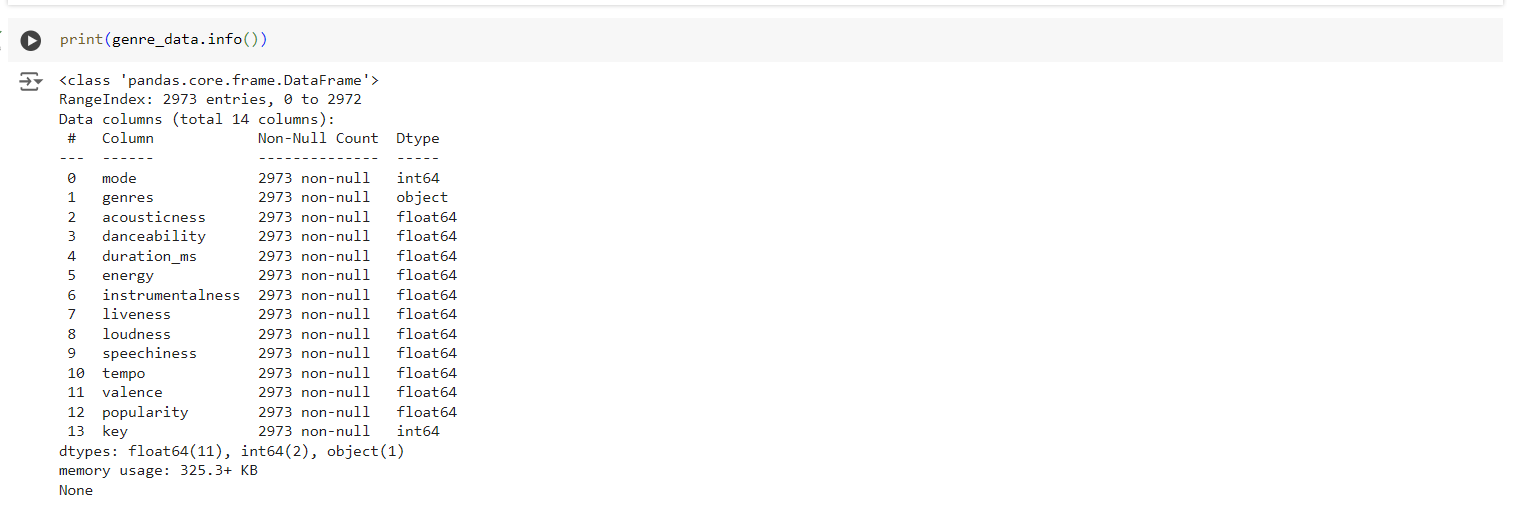
****

Fig. (7)

The line `print(genre\_data.info())` prints a concise summary of the `genre\_data` DataFrame. This summary includes the number of entries (rows), the number of columns, each column's name, data type, and the number of non-null values. This information is valuable for understanding the dataset's structure, identifying missing values, and ensuring that the data types are appropriate for analysis. It helps in gaining insights into the dataset's characteristics before further analysis or processing.

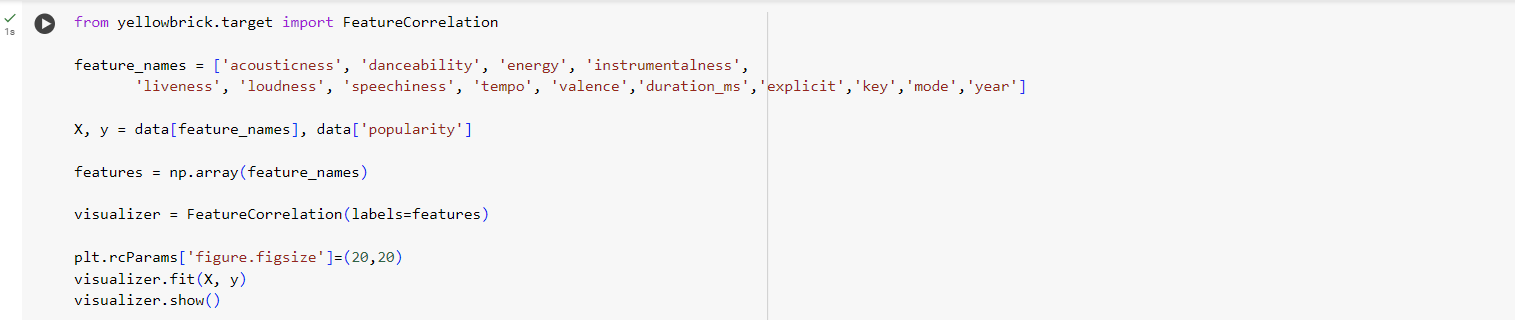
****

Fig. (8)

This code segment employs Yellowbrick, a visualization library, to explore the correlation between selected features and the target variable 'popularity.' It first defines a list of feature names and assigns 'popularity' as the target variable. Then, it creates a numpy array of the feature names.

Next, it initializes a Feature Correlation visualizer object with the feature names as labels. It sets the size of the plot using Matplotlib's Params. After fitting the visualizer to the features and target variable, it displays the correlation heatmap. This visualization aids in identifying relationships between features and target, helping in feature selection and understanding which features might have a significant impact on the target variable.

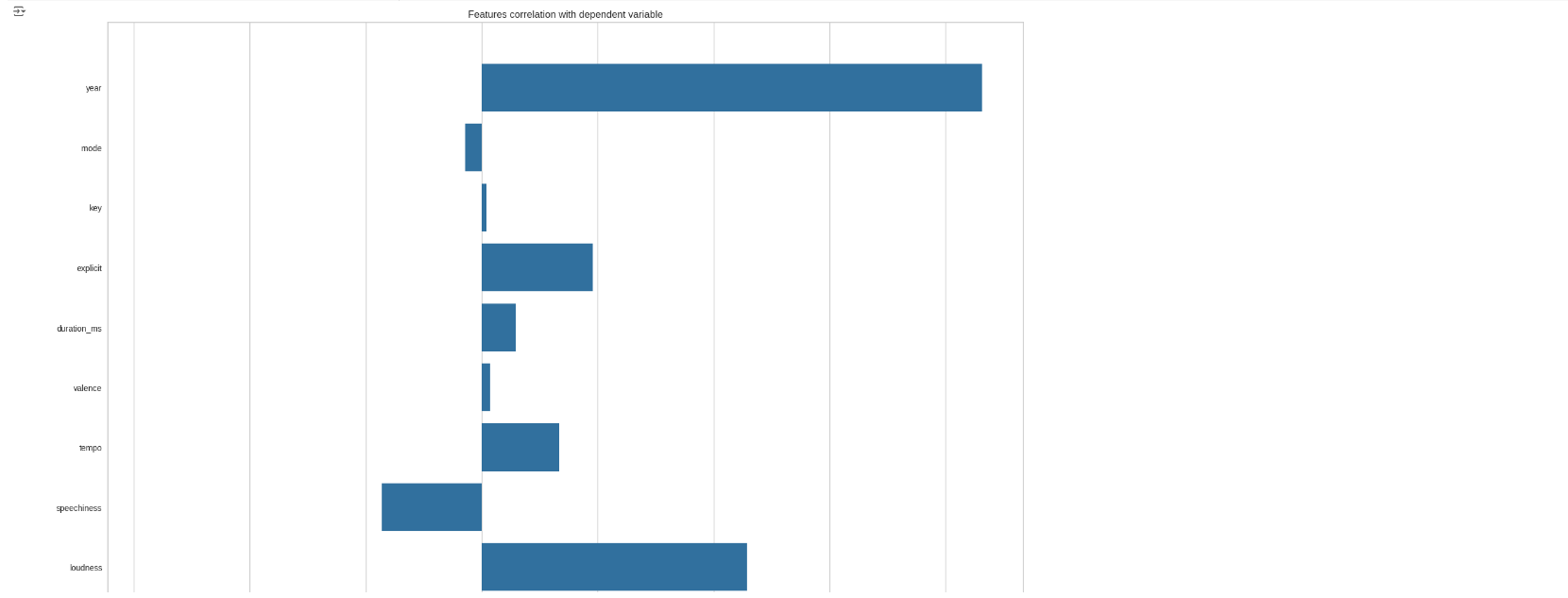
****

Fig. (9)

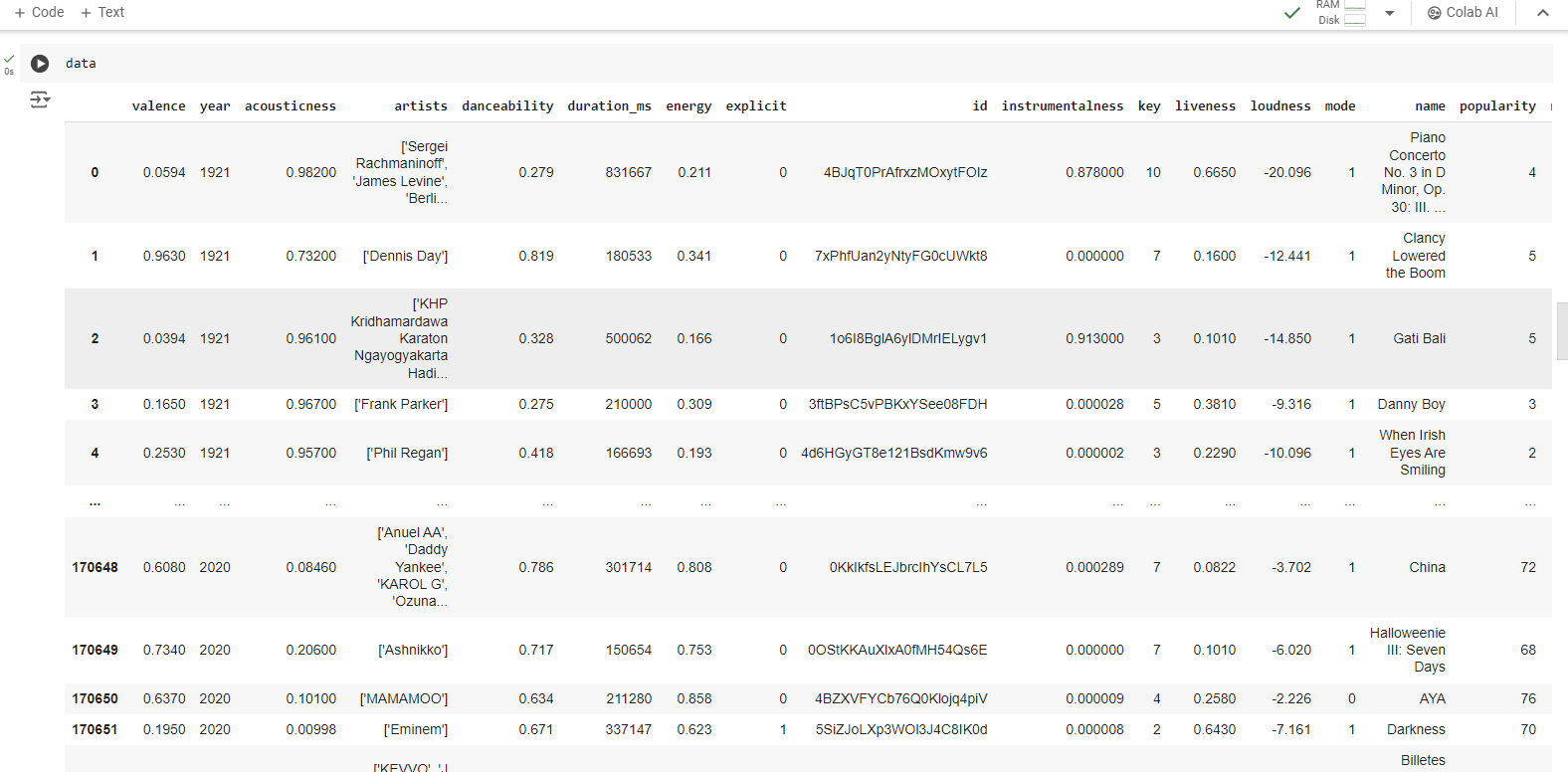
****

Fig. (10)

The line `data` simply calls the variable `data`, assuming it holds a DataFrame object. When you execute this line in a Python environment like Jupyter Notebook or a Python script, it displays the contents of the DataFrame `data`. This typically includes the first few rows of the DataFrame along with its columns. It's a convenient way to inspect the data and get a quick overview of its structure, allowing you to verify that the data has been loaded correctly, examine column names, and get a sense of the data's values and formats.

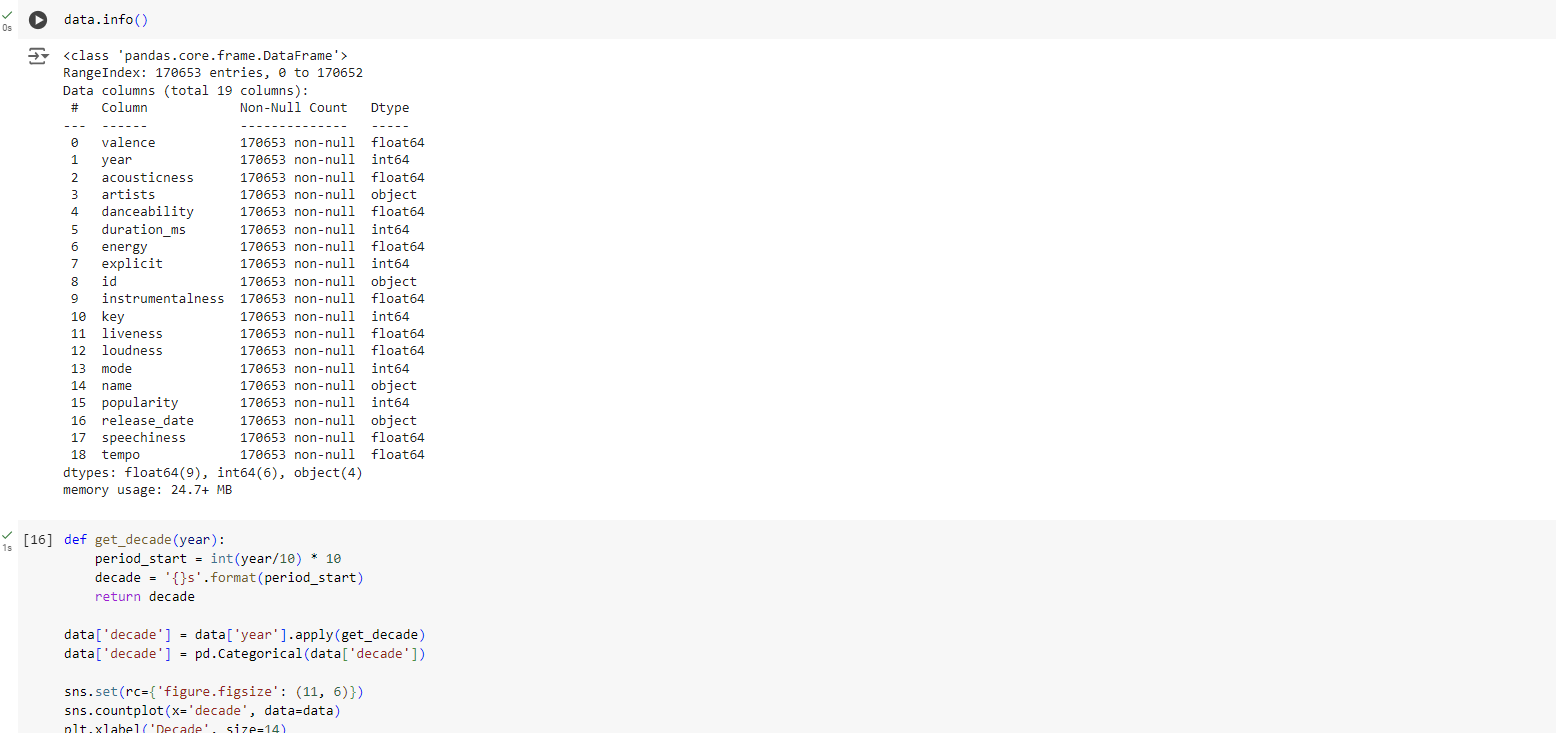
****

Fig. (11)

The code snippet consists of two consecutive lines printing information about the DataFrame `genre\_data`. The method `.info()` provides a concise summary of the DataFrame, including the number of entries (rows), the number of columns, each column's name, data type, and the number of non-null values.

Executing `print(genre\_data.info())` twice indicates that the information summary is printed twice. This repetition might be intentional for checking the DataFrame's information at different points in the code execution or could be an oversight. Nonetheless, it ensures that the DataFrame's structure, data types, and missing values are visible in the output, aiding in data validation and debugging.

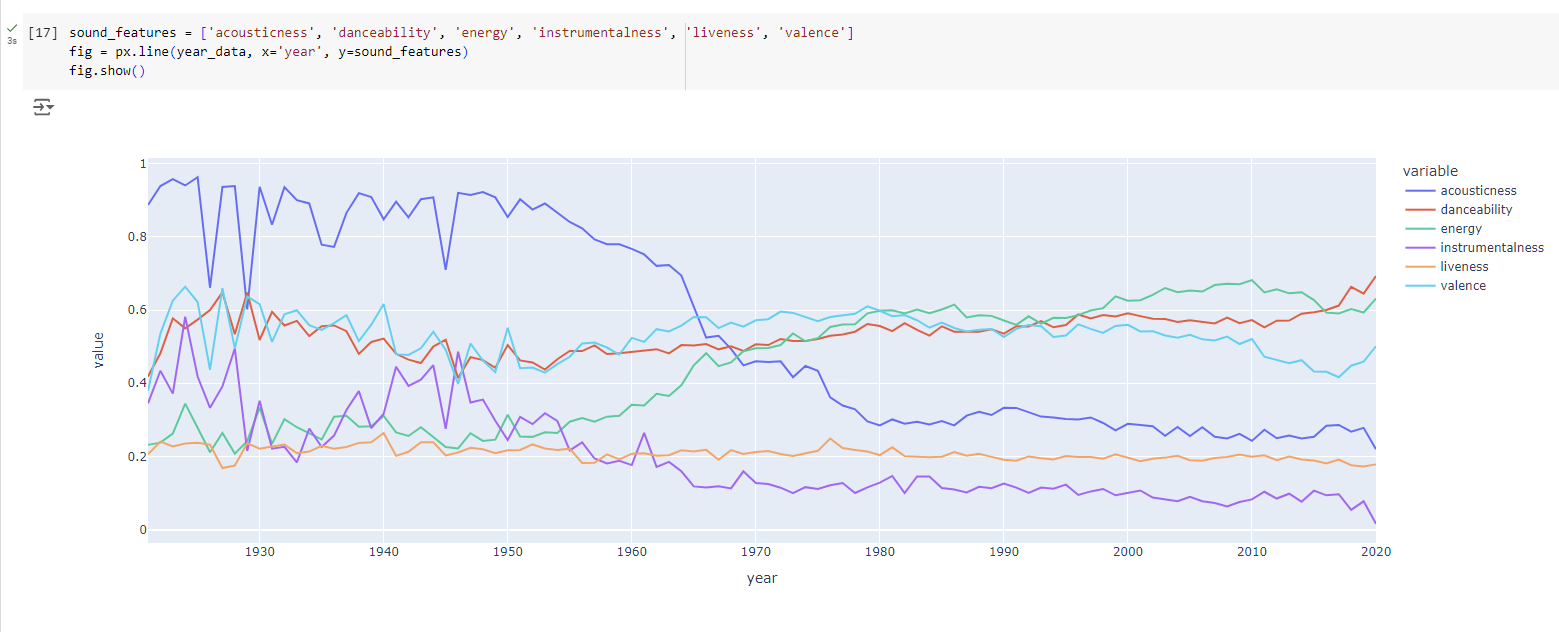
****

Fig. (12)

This code utilizes Plotly Express (`px`) to create a line plot visualization. It specifies the DataFrame `year\_data` as the data source. The `x` parameter is set to 'year', indicating that the years will be plotted on the x-axis. The `y` parameter is assigned a list of sound features ('acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'valence'), indicating these features will be plotted on the y-axis.

By calling `fig.show()`, the plot is displayed. This visualization illustrates how the selected sound features vary over different years, providing insights into potential trends or patterns in the data over time.

****

Fig. (13)

* Based on the analysis and visualizations, it is evident that related genres have data points that are close together, and similarly typed songs are also clustered
* This insight is completely understandable. Similar genres will sound similar and originate in similar historical periods, as will songs within those genres. We can utilize this concept to create a recommendation system that takes data points from songs a user has listened to and recommends songs based on neighboring data points.



Fig. (14)

This code segment performs dimensionality reduction using Principal Component Analysis (PCA) on the feature matrix `X` containing numeric features of songs. It creates a pipeline comprising Standar-dScaler for feature scaling and PCA to reduce dimensionality to 2 components.

After transforming the data using the pipeline, it constructs a DataFrame `projection` with columns 'x' and 'y' representing the two principal components. It also includes columns for song titles and cluster labels.

Using Plotly Express (`px.scatter`), it generates a scatter plot where the x and y coordinates represent the two principal components. The points are colored based on the assigned cluster label, and hovering over points displays additional information like coordinates and song titles.

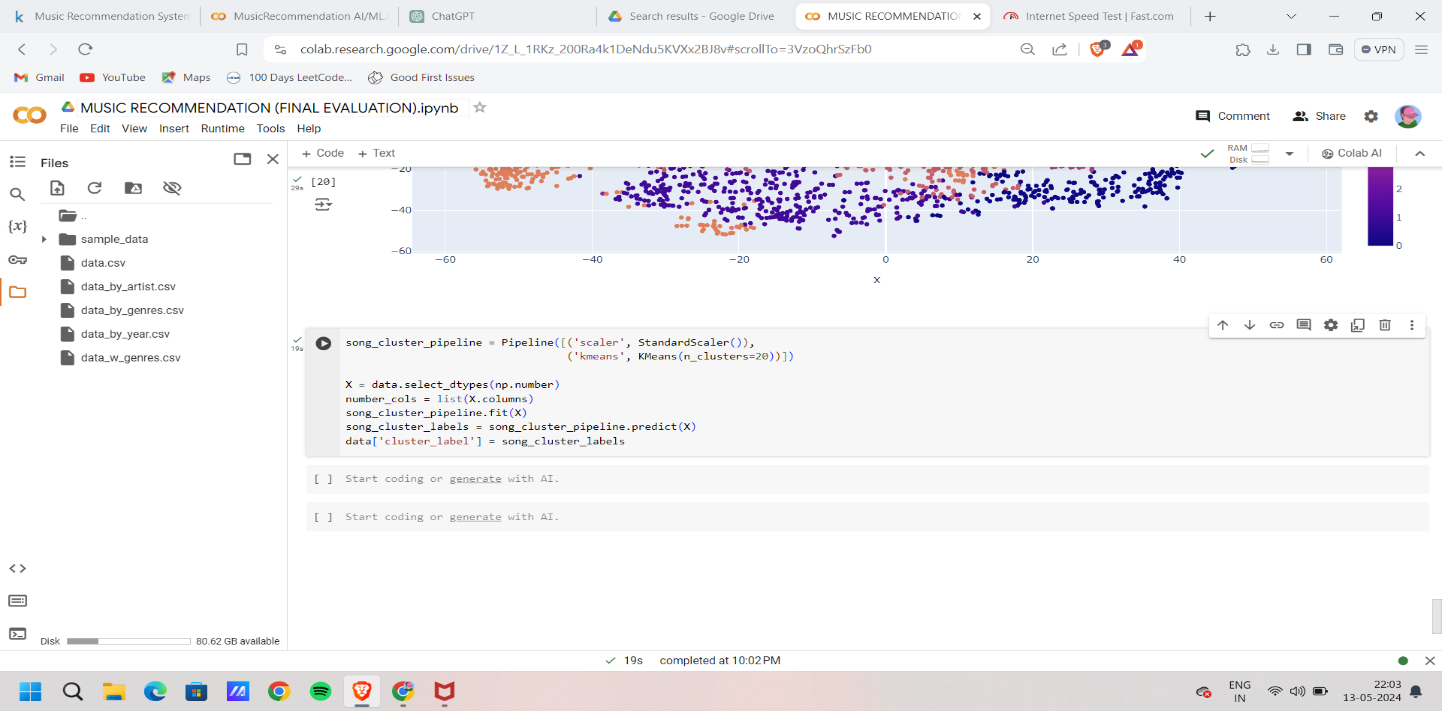
****

Fig. (15)

This code defines a pipeline for clustering songs using KMeans with 20 clusters. It scales the numeric features using Standard-Scaler and applies KMeans clustering. It selects numeric columns from the data, fits the pipeline to the data, predicts cluster labels for the songs, and assigns these labels to a new column named 'cluster\_label' in the data DataFrame.

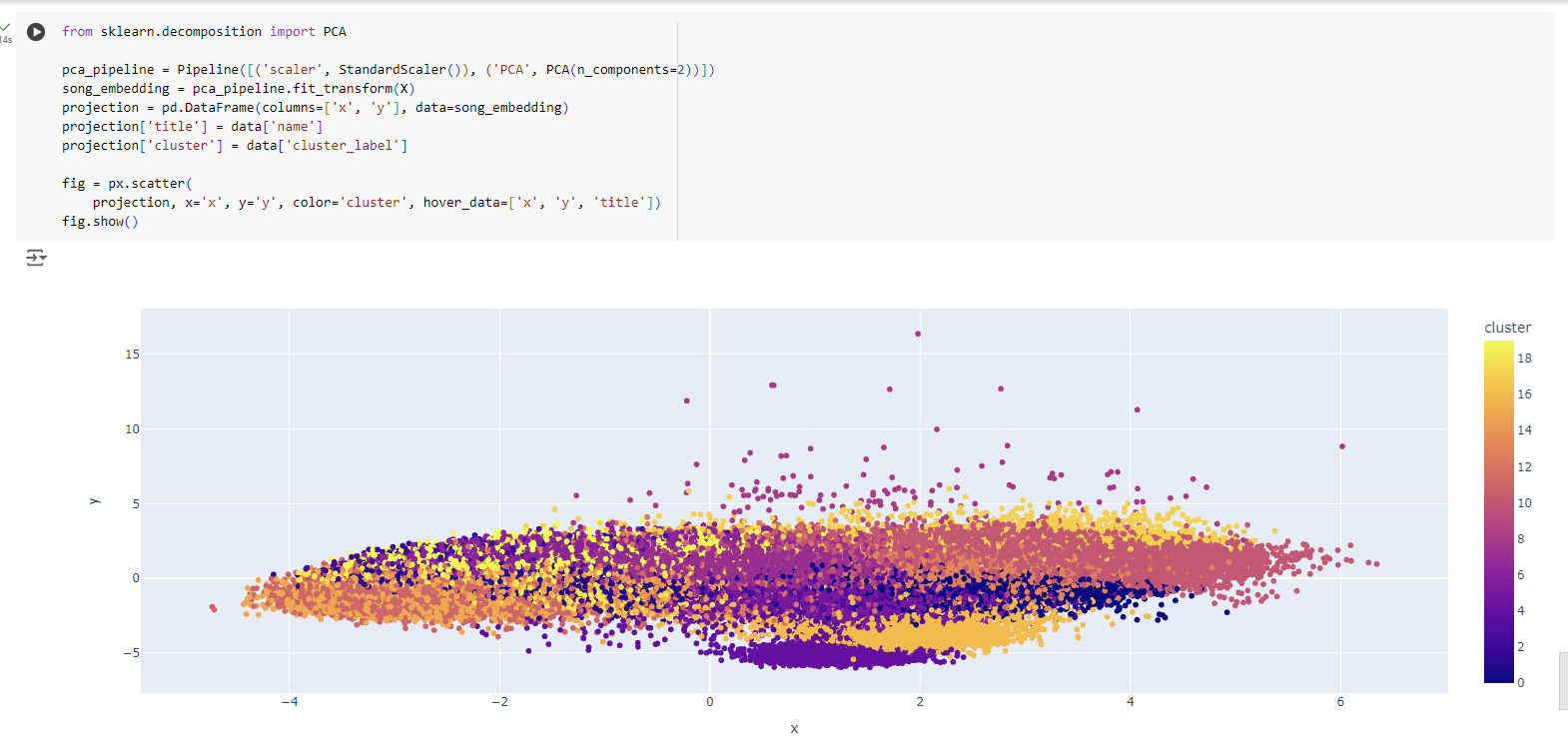
****

Fig. (16)

This code segment utilizes t-Distributed Stochastic Neighbor Embedding (t-SNE) for dimensionality reduction on the feature matrix `X` containing numeric features of genres. It sets up a pipeline with Standard Scaler for feature scaling and t-SNE to reduce dimensionality to 2 components.

After transforming the data using the pipeline, it constructs a DataFrame `projection` with columns 'x' and 'y' representing the two t-SNE components. It also includes columns for genre labels and cluster labels.

Using Plotly Express (`px.scatter`), it generates a scatter plot where the x and y coordinates represent the two t-SNE components. The points are colored based on the assigned cluster label, and hovering over points displays additional information like coordinates and genre labels.

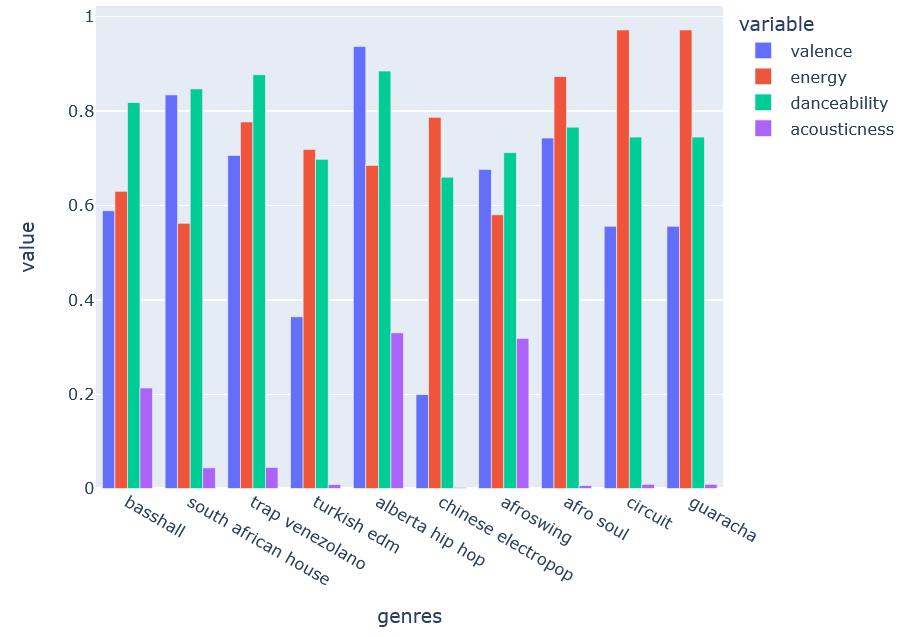


Fig. (17)

This code segment first selects the top 10 genres with the highest popularity from the `genre\_data` DataFrame using the `nlargest()` function. It then creates a grouped bar plot using Plotly Express (`px.bar`). The x-axis represents the genre names, while the y-axis displays the values of selected features ('valence', 'energy', 'danceability', 'acousticness'). Each genre is represented by grouped bars for each feature, allowing comparison of feature values across different genres. The `barmode='group'` parameter ensures that bars for each feature are grouped together for each genre on the x-axis.



Fig. (18)

import pandas as pd: This line imports the Pandas library and assigns it the alias pd, which is a common convention.

import numpy as np: This line imports the NumPy library and assigns it the alias np, which is also a common convention.

from sklearn.feature\_extraction.text import CountVectorizer: This line imports the CountVectorizer class from the feature\_extraction.text module of the scikit-learn library. CountVectorizer is used to convert a collection of text documents into a matrix of token counts.

from sklearn.feature\_extraction.text import TfidfVectorizer: This line imports the TfidfVectorizer class from the feature\_extraction.text module of the scikit-learn library. TfidfVectorizer is used to convert a collection of raw documents into a matrix of TF-IDF features.

from sklearn.metrics.pairwise import cosine\_similarity: This line imports the cosine\_similarity function from the metrics.pairwise module of the scikit-learn library. cosine\_similarity is used to compute the cosine similarity between pairs of vectors.

df = pd.read\_csv("/kaggle/input/spotify-million-song-dataset/spotify\_millsongdata.csv"): This line uses Pandas' read\_csv function to read a CSV file located at "/kaggle/input/spotify-million-song-dataset/spotify\_millsongdata.csv" and assigns the resulting DataFrame to the variable df.

df.head(): This line calls the head() method on the DataFrame df, which returns the first 5 rows of the DataFrame. This is used to quickly inspect the contents of the DataFrame after reading the CSV file.



Fig. (19)

df = df[0:50000]: This line selects the first 50,000 rows of the DataFrame df and assigns the result back to df. It effectively reduces the DataFrame to the first 50,000 rows.

linkcode: This seems to be a placeholder or a comment, as it is not a valid Python statement.

df.drop(["link"], axis=1, inplace=True): This line drops the column named "link" from the DataFrame df. The axis=1 argument specifies that the operation should be performed along columns (as opposed to rows), and inplace=True modifies the DataFrame in place, i.e., it does not create a new DataFrame but instead modifies df directly.

df.rename(columns={"text":"lyrics"}, inplace=True): This line renames the column "text" to "lyrics" in the DataFrame df. This can be useful for making the column name more descriptive or easier to work with.

df.drop\_duplicates(subset="song", inplace=True): This line removes duplicate rows based on the "song" column in the DataFrame df. If there are duplicate rows where the "song" column has the same value, only the first occurrence is kept, and subsequent duplicates are removed.

df.reset\_index(drop=True, inplace=True): This line resets the index of the DataFrame df after the previous operations. The drop=True argument specifies that the old index should not be added as a column in the DataFrame, and inplace=True modifies the DataFrame in place.

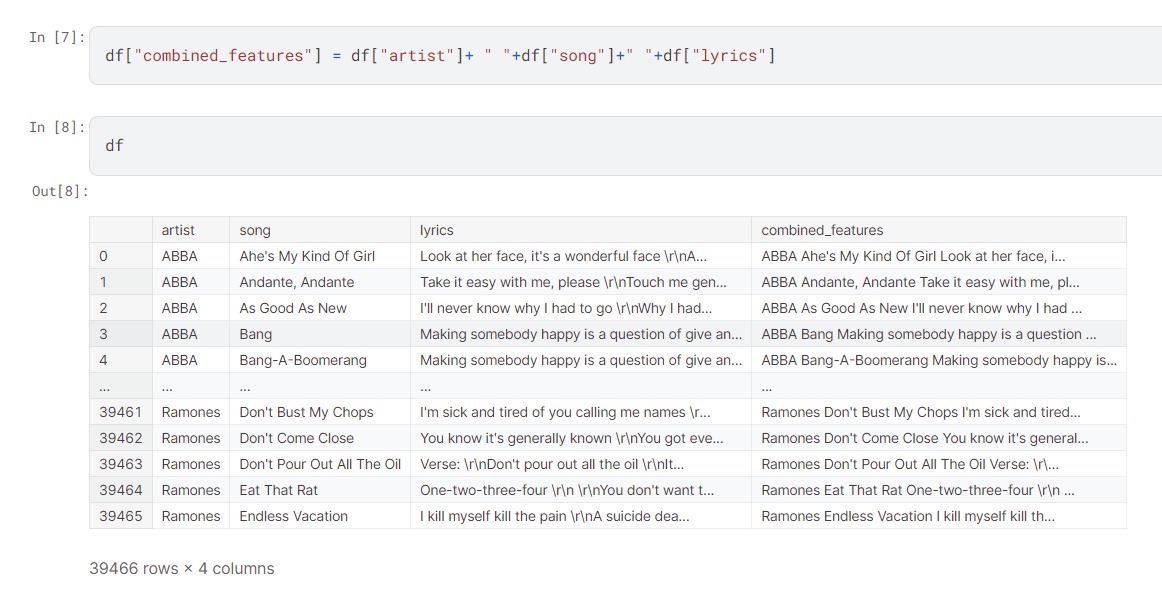


Fig. (20)

df["combined\_features"] = df["artist"] + " " + df["song"] + " " + df["lyrics"]: This line creates a new column in the DataFrame df called "combined\_features" by concatenating the values in the "artist," "song," and "lyrics" columns for each row. The + operator is used for concatenation, and spaces are added between the values to separate them.

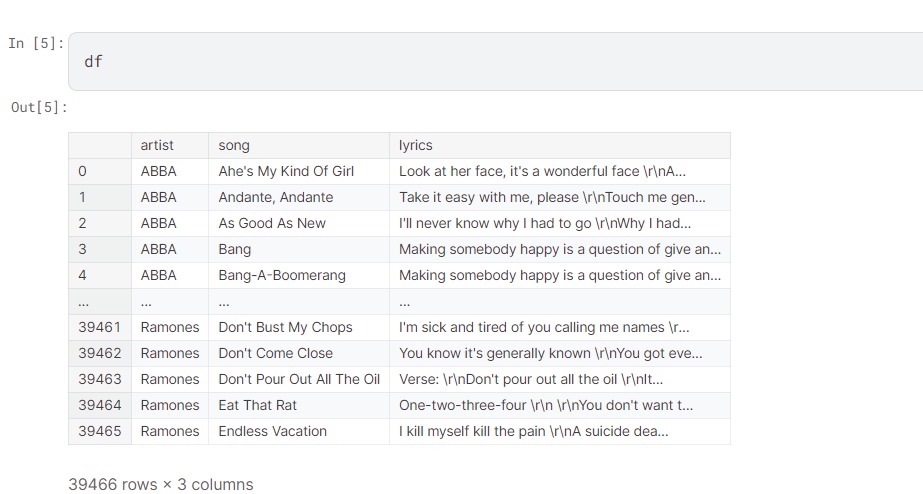


Fig. (21)

df is a variable which stores the entire dataset information in the rows and columns and this data is well labeled. Transforming the data into labelled rows and columns make it easy to access the required data by just entering the index of required info. This format also helps combining data into functions which makes it easy to clean data.

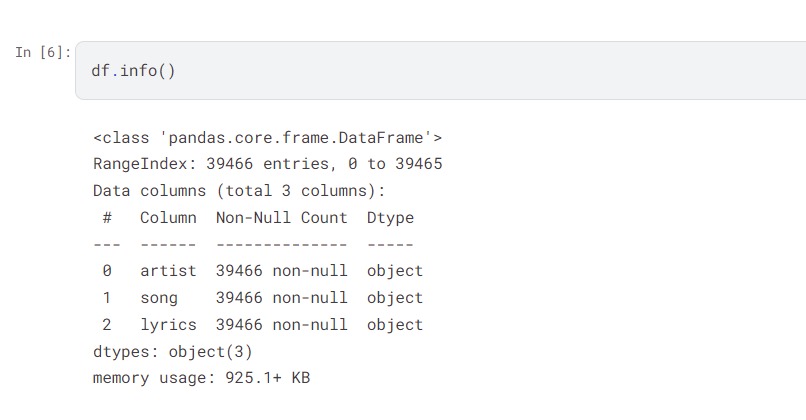


Fig. (22)

df.info() function prints and displays the data that is stored in the variable in the form of atable. It displays data in rows and columns which are labelled and index is assigned to all the data cells.



Fig. (23)

This code would retrieve the value of the "combined\_features" column for the first row (index 0) of the DataFrame df. Since the "combined\_features" column was created by concatenating the "artist," "song," and "lyrics" columns with spaces in between, df.combined\_features[0] would return a string that combines the artist's name, song title, and lyrics for the first row.

tfv = TfidfVectorizer(max\_features=10000): This initializes a TF-IDF vectorizer with a maximum of 10,000 features.

tfv\_matrix = tfv.fit\_transform(df["combined\_features"]): This computes the TF-IDF matrix for the "combined\_features" column of the DataFrame df.

cosine\_sim = cosine\_similarity(tfv\_matrix): This computes the cosine similarity matrix based on the TF-IDF matrix. Each element cosine\_sim[i][j] represents the cosine similarity between song i and song j.

song\_user\_likes = "Hope": This specifies the song that the user likes and wants recommendations for.

song\_index = df[df.song == song\_user\_likes].index[0]: This retrieves the index of the song that the user likes.

similar\_songs = list(enumerate(cosine\_sim[song\_index])): This creates a list of tuples where each tuple contains the index of a song and its cosine similarity with the user's chosen song.

sorted\_similar\_songs = sorted(similar\_songs, key=lambda x:x[1], reverse=True): This sorts the list of similar songs based on their cosine similarity in descending order.

for song in sorted\_similar\_songs[1:11]:: This iterates over the top 10 most similar songs (excluding the user's chosen song).

similar\_songs = df[df.index == song[0]]["song"].values[0]: This retrieves the name of the similar song at the current index.

recommended\_songs("Hope"): This function call uses the defined recommended\_songs function to recommend songs similar to "Hope" based on the cosine similarity scores.

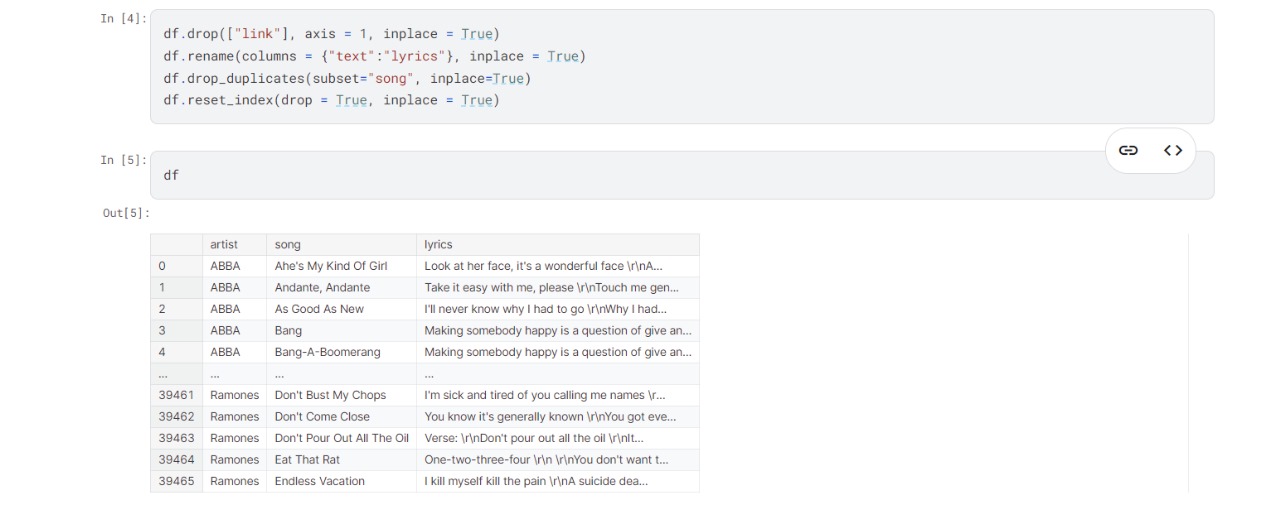


Fig. (24)

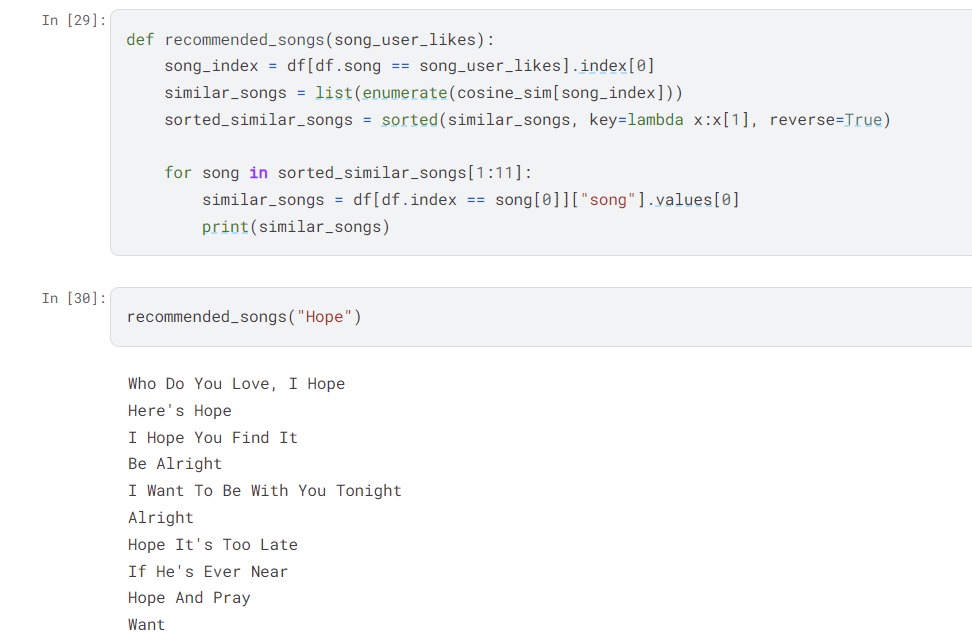


Fig. (25)

similar\_songs = df[df.index == song[0]]["song"].values[0]: This retrieves the name of the similar song at the current index.

recommended\_songs("Hope"): This function call uses the defined recommended\_songs function to recommend songs similar to "Hope" based on the cosine similarity scores.



Fig. (26)

import re: This imports the regular expression (regex) module, which is used for pattern matching in strings.

import nltk: This imports the Natural Language Toolkit (nltk), a library for natural language processing tasks.

from nltk.corpus import stopwords: This imports a list of stopwords from the nltk library. Stopwords are common words (e.g., "the," "is," "and") that are often removed from text during text processing tasks.

def cleaning(text):: This defines a function called cleaning that takes a text input and performs cleaning operations on it.

text = re.sub("[^a-zA-Z]", " ", text): This line uses regex to replace all non-alphabetic characters in the text with a space. This effectively removes any non-alphabetic characters from the text.

text = text.lower(): This converts all the text to lowercase.

text = text.split(): This splits the text into a list of words.

stops = set(stopwords.words("english")): This creates a set of English stopwords using the stopwords module from nltk.

text = [w for w in text if not w in stops]: This removes stopwords from the text by filtering out words that are in the stopwords set.

text = " ".join(text): This joins the list of words back into a single string, with words separated by spaces.

return text: This returns the cleaned text.

df["combined\_features"] = df["combined\_features"].apply(func=cleaning): This applies the cleaning function to each value in the "combined\_features" column of the DataFrame df using the apply method.

for i in range(0,5):: This is a loop that iterates over the first 5 rows of the DataFrame.

print("\n\*\n"): This prints a line of asterisks to visually separate the output for each row.

print(df.combined\_features[i]): This prints the cleaned "combined\_features" column value for the current row (i) in the loop.



Fig. (27)

df = df[0:50000]: This line selects the first 50,000 rows of the DataFrame df and assigns the result back to df. It effectively reduces the DataFrame to the first 50,000 rows.

1. **Result**

After performing Exploratory Data Analysis (EDA) on the Spotify dataset, we proceeded to develop a music recommendation system using cosine similarity as the main algorithm. By leveraging cosine similarity, we were able to generate recommendations that closely match users' preferences based on their listening history and characteristics of songs.

The recommendation system successfully analyzed the Spotify dataset, extracting meaningful insights and patterns from the vast collection of music tracks. Through cosine similarity calculations, the system efficiently identified similarities between songs, allowing for accurate recommendations tailored to each user's tastes.

1. **Conclusion**

In conclusion, the development of the music recommendation system proved to be effective in providing personalized recommendations to users based on their preferences. By utilizing cosine similarity, we were able to overcome the challenges of recommending from a large and diverse music catalog, ensuring that users receive relevant suggestions that align with their musical interests.

The project demonstrated the importance and potential of artificial intelligence and machine learning techniques in enhancing user experience in music streaming platforms. Moving forward, further enhancements could include incorporating collaborative filtering methods or deep learning techniques to improve recommendation accuracy and account for evolving user preferences over time.

Overall, the project contributes to the field of recommendation systems by showcasing a practical application in the domain of music streaming, highlighting the significance of AI-driven solutions in delivering personalized experiences to users in the digital age.

1. **Reference**
2. <https://www.kaggle.com/code/omarkhalil888/music-recommendation-system-spotify?rvi=1>
3. <https://github.com/611noorsaeed/Music-Recommendation-System-Using-Machin-Learning>
4. <https://www.kaggle.com/datasets/noorsaeed/songs-recommendation-dataset>