Recommendation System and Trend Analysis on Spotify

A detailed analysis of music marketing through Big Data

IDS 561 Analytics for Big Data- Fall 2024

University of Illinois Chicago

Amin Abbasi (mabba@uic.edu)

Matthew Yuan

Chia-Hsuan Lin

Bella Chen

TABLE OF CONTENTS

1) Problem Setting

Today, the overwhelming volume of diverse and complex music data far exceeds the basic needs and processing capacity of listeners, often resulting in information overload. High-end music charts aim to address this by analyzing various attributes to identify the unique selling points of curated songs. As a result, the ability to leverage personalized music recommendations to help users efficiently and accurately discover tracks of interest within the vast music library has become increasingly crucial.

1.1 Motivation

This project focuses on predicting which songs will be recommended to users by utilizing a large dataset. Developing this recommendation system has the potential to transform how music apps interact with their users, enabling companies in the online media industry to optimize their return on investment. By analyzing the data collected from users, such as their music preferences and purchase behavior, businesses can better tailor their offerings and maximize revenue.

1.2 Related Literature

Music recommendation systems have become crucial in enhancing user experience on platforms like Spotify and Pandora. With the advancement of machine learning techniques, several approaches have emerged to optimize the performance of these systems. This section reviews recent literature on collaborative filtering, content-based filtering, and hybrid models and discusses their application in industry settings such as Spotify.

Collaborative Filtering: Collaborative filtering (CF) predicts user preferences by analyzing historical behavior. Deldjoo et al. (2021) provided an extensive review of content-based music recommendation techniques, emphasizing the importance of CF in music recommendations. However, CF faces challenges like the cold-start problem and sparsity in user-item interactions. Jing et al. (2024) proposed a deep Bayesian network model that integrates heterogeneous data to address these issues and improve prediction accuracy.

Content-Based Filtering: Content-based filtering (CBF) recommends items by analyzing their attributes, such as genre, tempo, and lyrics. Luo et al. (2024) developed a multi-layered weighted hypergraph embedding learning method for diversified music recommendation, showcasing the effectiveness of CBF in capturing musical features and user preferences.

Hybrid Models: Hybrid recommendation systems combine the strengths of CF and CBF to improve recommendation accuracy. Deldjoo et al. (2021) emphasized the application of hybrid models in music recommendation, noting their advantage in handling complex user preferences and improving recommendation quality.

1.2.1 Industry Applications

Spotify: Spotify employs a hybrid recommendation system that integrates CF, CBF, and natural language processing to deliver personalized music recommendations. Gomez-Uribe and Hunt (2015) detailed Spotify's approach, highlighting its success in maintaining high user engagement and satisfaction through personalized recommendations.

Pandora: Pandora uses the Music Genome Project, a content-based approach that analyzes songs based on hundreds of musical attributes. Furnas (2014) discussed Pandora's strategy, emphasizing the platform's ability to match songs with user preferences based on detailed song profiling.

1.3 Relevant existing module

The following Python libraries and modules are highly suitable based on the project's requirements to develop a music recommendation system for Spotify, which involves collaborative filtering, content-based filtering, clustering, and time series analysis. These libraries are practical for handling the various aspects of the recommendation system and can help streamline the implementation.

1.3.1 Collaborative Filtering

a. Surprise

Functionality: Surprise is a Python library designed to build and analyze recommender systems. It includes built-in collaborative filtering algorithms such as K-Nearest Neighbors (KNN), Singular Value Decomposition (SVD), and others.

Key Features: Supports KNN-based collaborative filtering.Includes matrix factorization techniques like SVD and SVD++. Built-in evaluation tools like cross-validation and error metrics (e.g., RMSE, MAE).

Rationale: Surprise is a powerful tool for collaborative filtering and matrix factorization, ideal for predicting user preferences based on past behavior. It is particularly useful for implementing collaborative filtering models efficiently.

b. Implicit

Functionality: Implicit is optimized for collaborative filtering using implicit feedback, such as user interactions with songs (e.g., plays or skips). It supports matrix factorization algorithms like Alternating Least Squares (ALS).

Key Features: Efficient for large, sparse datasets. Works well with implicit feedback data.

Provides implementations of ALS and other matrix factorization techniques.

Rationale: Implicit is highly suitable for scenarios where user feedback is implicit, like song plays, making it an excellent choice for music recommendation systems that deal with such data.

c. SciPy

Functionality: SciPy is a general-purpose library for scientific computing in Python, which includes a K-Nearest Neighbors (KNN) implementation useful for collaborative filtering.

Key Features: Implements KNN algorithms for user-based or item-based collaborative filtering. Supports sparse matrix operations, which is important for handling large, sparse datasets.

Rationale: SciPy is versatile for building collaborative filtering systems using KNN. It works well with smaller datasets or simpler collaborative filtering methods and can handle sparse matrices efficiently.

d. LightFM

Functionality: LightFM is a hybrid recommendation library that combines collaborative and content-based filtering. It is optimized for large-scale, sparse datasets.

Key Features: Supports hybrid collaborative filtering using implicit and explicit feedback.

Includes algorithms like Bayesian Personalized Ranking (BPR) and Weighted Approximate-Rank Pairwise (WARP). Efficient at handling large data and integrating song metadata.

Rationale: LightFM is an ideal choice for building a hybrid recommendation system that combines user behavior and song metadata. It is suitable for handling large-scale music recommendation tasks.

e. TensorFlow Recommenders (TFRS)

Functionality: TensorFlow Recommenders is a deep learning-based framework for building collaborative filtering models. It integrates well with TensorFlow and allows for building more complex models.

Key Features: Deep learning-based collaborative filtering models. Integrates easily with TensorFlow for more advanced feature representations. Supports both user-based and item-based models.

Rationale: TFRS is suitable for building deep learning-based recommendation systems, providing advanced techniques that can enhance recommendation accuracy by leveraging neural networks.

1.3.2 Content-Based Filtering

a. spaCy

Functionality: spaCy is a powerful natural language processing (NLP) library that can be used to analyze song lyrics and metadata.

Key Features:Fast and efficient NLP processing.Pre-trained models for text processing tasks such as tokenization and named entity recognition.Integrates well with other machine learning libraries.

Rationale: spaCy is well-suited for text-based analysis, such as analyzing song lyrics to extract meaningful information for content-based filtering. It can be used to process metadata or lyrics for content-based recommendations.

b. Gensim

Functionality: Gensim is an NLP library used for topic modeling and document similarity. It can be used to analyze the semantic meaning of song lyrics or metadata.

Key Features:Efficient handling of large text corpora.Provides Word2Vec and other models for semantic analysis.Suitable for content-based recommendation using song descriptions or lyrics.

Rationale: Gensim is excellent for generating vector representations of song lyrics or descriptions, which can then be used to find similarities between songs and make recommendations based on content.

c. TfidfVectorizer (from sci-kit learn)

Functionality: TfidfVectorizer is a text feature extraction tool that converts text data into a numerical format using the Term Frequency-Inverse Document Frequency (TF-IDF) method.

Key Features: Converts song metadata or lyrics into a feature vector. Measures the importance of each word within the context of all documents (or songs).

Rationale: TfidfVectorizer is useful for transforming text data, like song lyrics, into features that can be used for content-based recommendations. It helps identify key terms that define a song's content.

1.3.3 K-Means Clustering

a. scikit-learn

Functionality: scikit-learn is a widely-used library for machine learning that includes K-Means clustering and other clustering algorithms.

Key Features: Implements K-Means and other clustering algorithms. Provides evaluation metrics like silhouette score for assessing clustering quality.

Rationale: scikit-learn is an excellent tool for clustering songs or users based on features like genre or listening history. It is easy to use and integrates well with other machine learning tasks.

b. HDBSCAN

Functionality: HDBSCAN is a density-based clustering algorithm that can identify clusters of arbitrary shapes and densities, making it useful for complex music data.
Key Features: Can identify arbitrarily shaped clusters. Works well with high-dimensional data.Does not require the number of clusters to be specified ahead of time.
Rationale: HDBSCAN is useful when K-Means is not suitable, especially for music data with varying density and complex relationships. It allows for better clustering in such cases.

c. MiniBatchKMeans (from scikit-learn)
Functionality: MiniBatchKMeans is an optimized version of K-Means that processes smaller batches of data, making it suitable for large datasets.
Key Features: Faster than traditional K-Means for large datasets.Suitable for clustering high-dimensional music feature data.
Rationale: MiniBatchKMeans is ideal for efficiently clustering large music datasets, making it well-suited for music recommendation systems that need to process a large number of songs.

1.3.4 Time Series Analysis

a. pandas
Functionality: pandas is a powerful data manipulation library that supports time series operations like rolling averages and time-based aggregations.
Key Features: Supports date-time functionality and time series operations. Can handle and manipulate time-based data, like user listening behavior over time.
Rationale: pandas is indispensable for handling time-based data and is ideal for analyzing trends in music preferences or user behavior over time.

b. stats models
Functionality: stats models provide tools for statistical modeling and include time series models like ARIMA for forecasting.
Key Features: Includes ARIMA and SARIMA models for time series forecasting. Can handle seasonal and trend components in time series data.

Rationale: stats models is well-suited for time series forecasting tasks and can be used to predict future trends in music preferences or user behavior.

c. Prophet
Functionality: Prophet is a forecasting tool developed by Facebook, designed to handle seasonal trends and missing data in time series.
Key Features: Handles daily, weekly, and yearly seasonalities. Built-in methods to handle missing data and outliers.
Rationale: Prophet is highly effective for forecasting music trends or user preferences, especially with seasonal patterns or incomplete data.

2) Experimental Procedure

2.1 Data Preparation

Data cleaning is a crucial step in ensuring the accuracy and effectiveness of machine learning models. This process typically involves identifying and handling outliers (either by removal or imputation), addressing missing values, eliminating duplicate entries, and removing features with little or no significance. Additionally, data visualization plays a vital role in providing an initial understanding of the dataset, offering insights that guide preprocessing before model implementation. The group focused on removing irrelevant features and prioritizing highly correlated ones to maximize their influence on predicting satisfaction.

- Data has been collected from the GitHub platform. (https://github.com/AmolMavuduru/SpotifyRecommenderSystem/tree/master/data)
- The data collected is over a span of 100 years, 1921 - 2020.
- The datasets we are using for the project are - data.csv, data_by_year.csv, data_by_genre.csv.
- Our main dataset data.csv(df_Spotify) consists of 170653 entries and 19 columns with non-null values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   valence           170653 non-null  float64
 1   year              170653 non-null  int64
 2   acousticness      170653 non-null  float64
 3   artists           170653 non-null  object
 4   danceability      170653 non-null  float64
 5   duration_ms       170653 non-null  int64
 6   energy            170653 non-null  float64
 7   explicit          170653 non-null  int64
 8   id                170653 non-null  object
 9   instrumentalness  170653 non-null  float64
 10  key               170653 non-null  int64
 11  liveness          170653 non-null  float64
 12  loudness          170653 non-null  float64
 13  mode              170653 non-null  int64
 14  name              170653 non-null  object
 15  popularity        170653 non-null  int64
 16  release_date      170653 non-null  datetime64[ns]
 17  speechiness       170653 non-null  float64
 18  tempo             170653 non-null  float64
dtypes: datetime64[ns](1), float64(9), int64(6), object(3)
memory usage: 24.7+ MB
```

Fig.1

- data_by_year.csv shows audio features of songs in different years, it consists of 100 entries and 14 columns with non-null values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   mode              100 non-null    int64
 1   year              100 non-null    int64
 2   acousticness      100 non-null    float64
 3   danceability      100 non-null    float64
 4   duration_ms       100 non-null    float64
 5   energy            100 non-null    float64
 6   instrumentalness  100 non-null    float64
 7   liveness          100 non-null    float64
 8   loudness          100 non-null    float64
 9   speechiness       100 non-null    float64
 10  tempo             100 non-null    float64
 11  valence           100 non-null    float64
 12  popularity        100 non-null    float64
 13  key               100 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 11.1 KB
None
```

Fig.2

- data_by_genre.csv shows audio features for each genre, it consists of 2972 entries and 14 columns with non-null values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   mode              2973 non-null   int64
 1   genres            2973 non-null   object
 2   acousticness      2973 non-null   float64
 3   danceability      2973 non-null   float64
 4   duration_ms       2973 non-null   float64
 5   energy            2973 non-null   float64
 6   instrumentalness  2973 non-null   float64
 7   liveness          2973 non-null   float64
 8   loudness          2973 non-null   float64
 9   speechiness       2973 non-null   float64
 10  tempo             2973 non-null   float64
 11  valence           2973 non-null   float64
 12  popularity        2973 non-null   float64
 13  key               2973 non-null   int64
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KB
None
```

Fig.3

Clean and Transform Dataset: Data preparation is the method of cleaning and transforming raw data subsequent to processing and analysis. It's a crucial stage before processing that often entails reformatting data, making data corrections, and merging data sets to enrich data. To transform data into information and remove prejudice induced by poor data quality, it is vital to put it into context. We normalized our data by deleting duplicate and missing entries, removing superfluous symbols like quote marks and square brackets, and standardizing the dates

2.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) plays a crucial role in identifying relevant features such as song attributes, user preferences, and listening patterns. By analyzing data through visualizations and statistical methods, EDA helps uncover trends, correlations, and anomalies in user behavior and music features. Additionally, we can select features that are relevant to create a recommendation system.

- The correlation heatmap shows the relationships between numerical variables in the Spotify dataset. It helps identify key features that are highly interrelated or independent, guiding the selection of relevant variables for building the recommendation system.

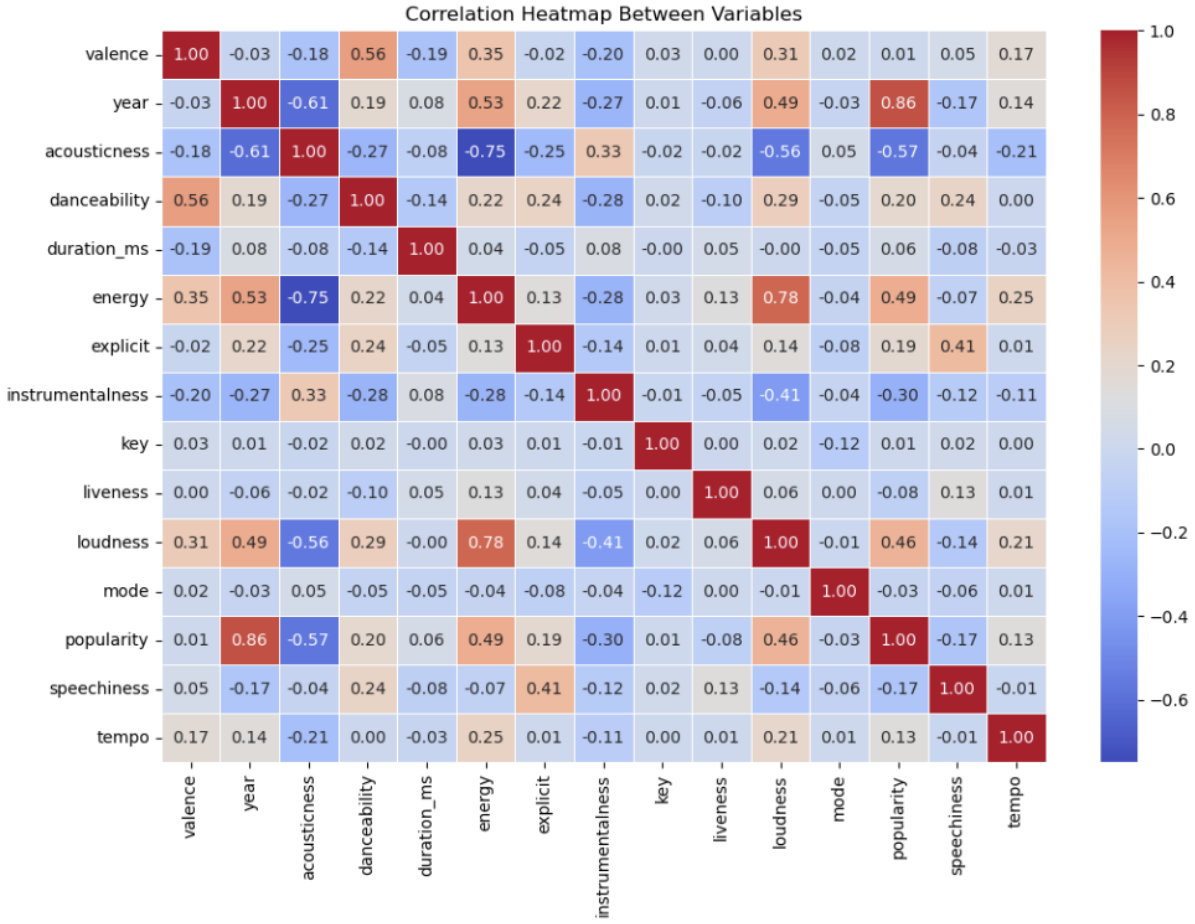


Fig.4 Correlation Matrix between variables

- Music over time: Using the data grouped by year, we can understand how the overall sound of music has changed from 1921 to 2020.

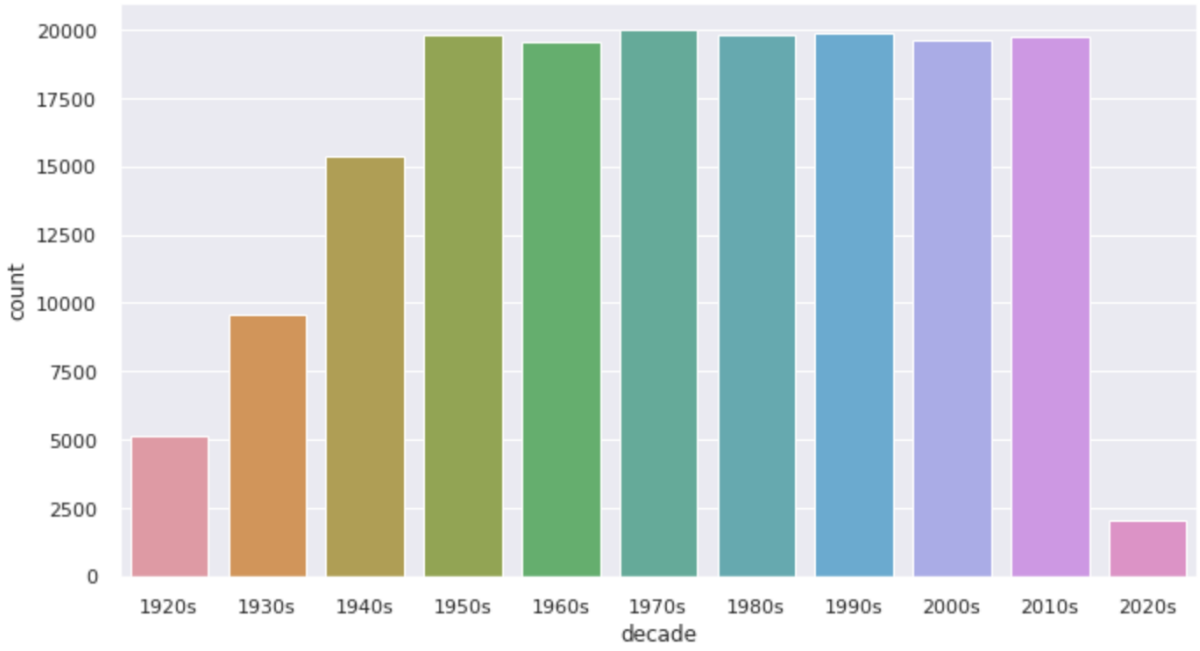


Fig.5 Number of songs per decade

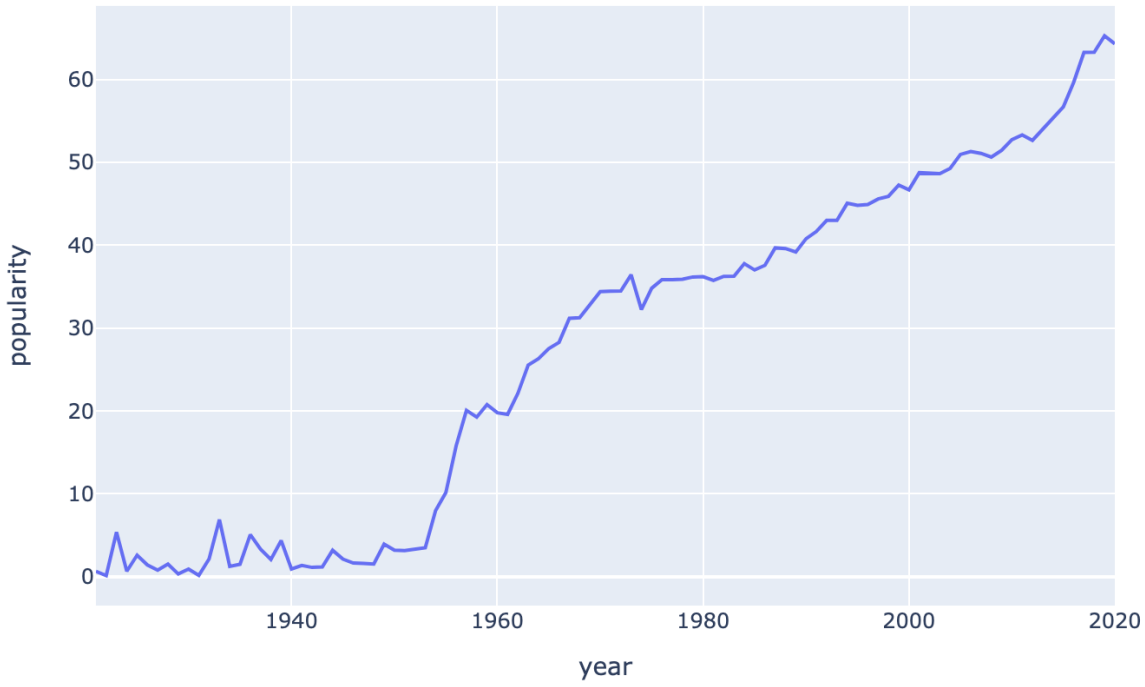


Fig.6 Popularity trends over years

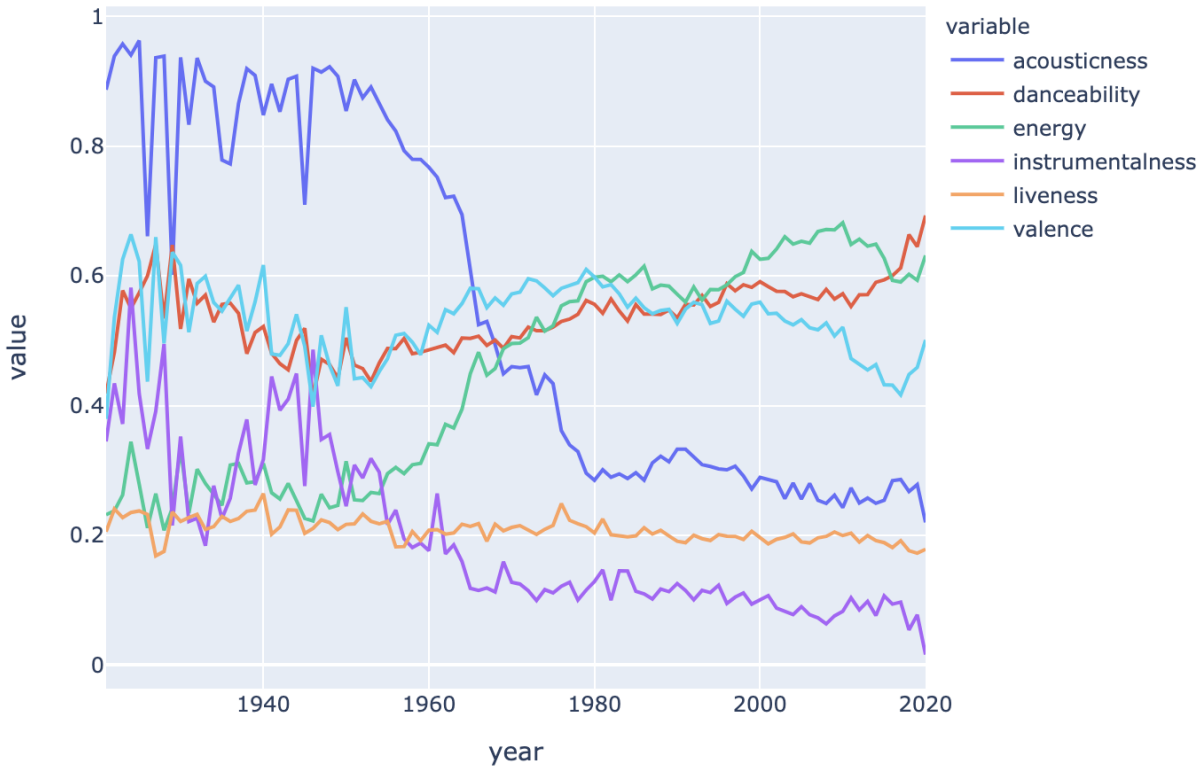


Fig.7 Variable trends over years

- Characteristics of different genres: This dataset contains the audio features for different songs along with the audio features for different genres. We can use this information to compare different genres and understand their unique differences in sound.

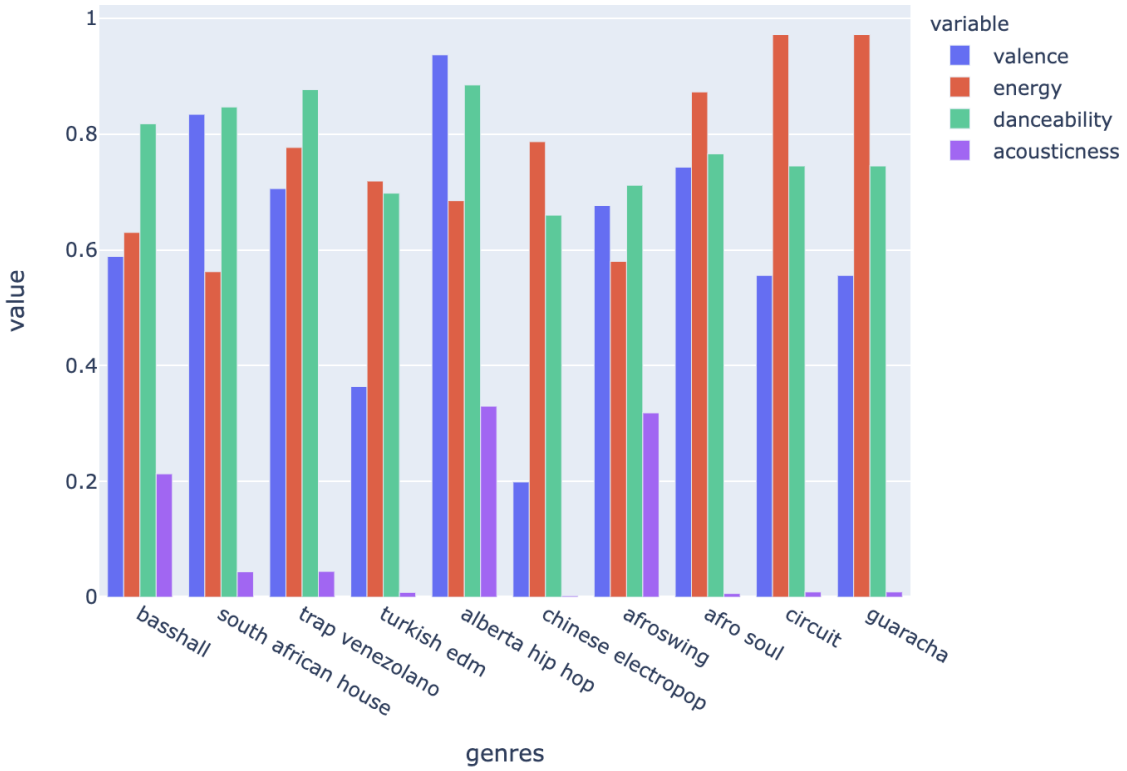


Fig.8 Characteristics of different genres

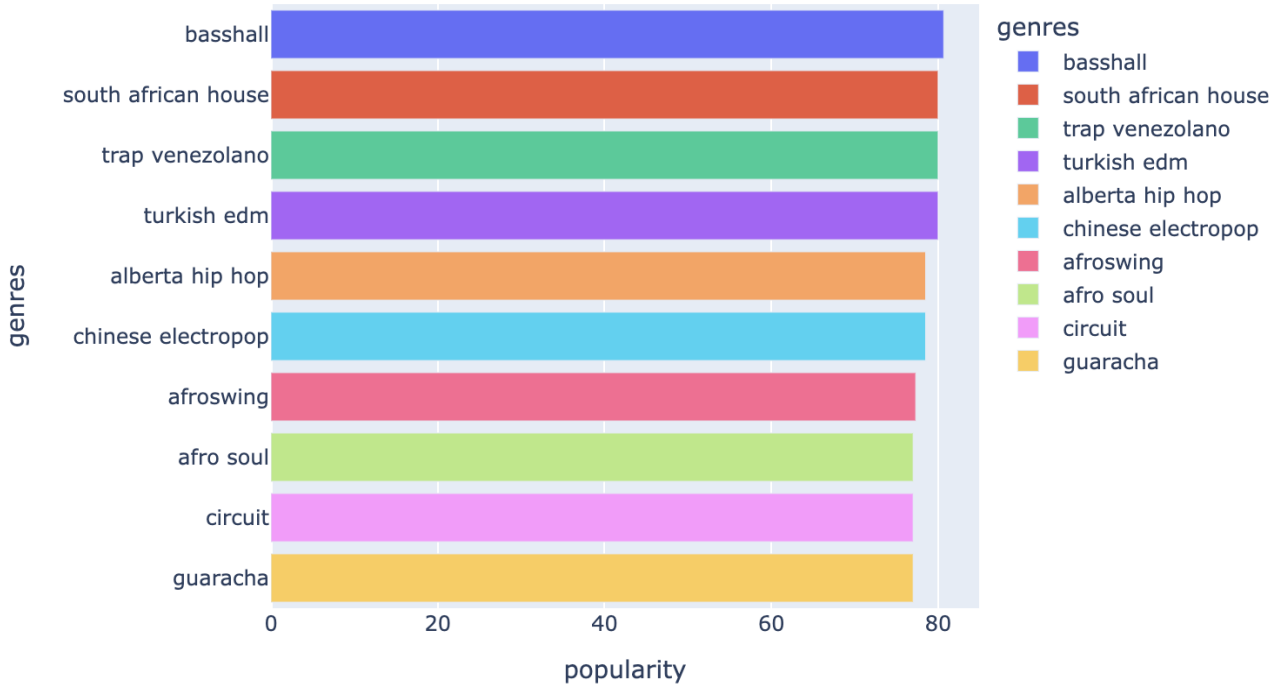


Fig.9 Top genres by popularity

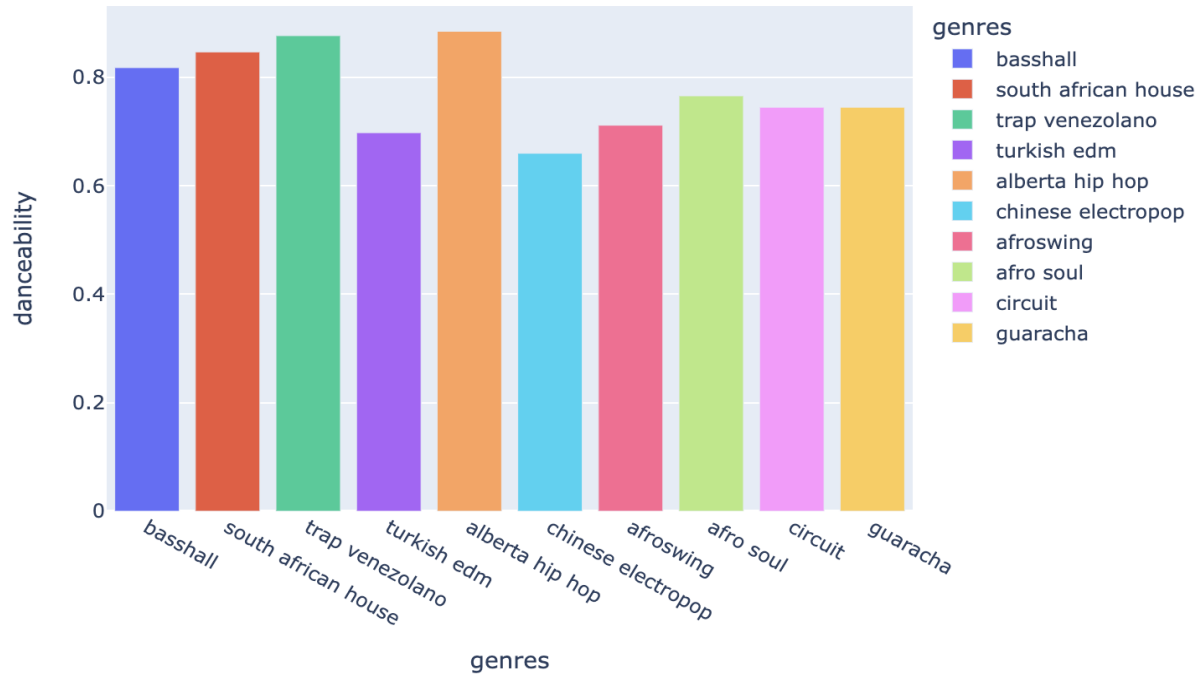


Fig.10 Danceability distribution for top 10 popular genres

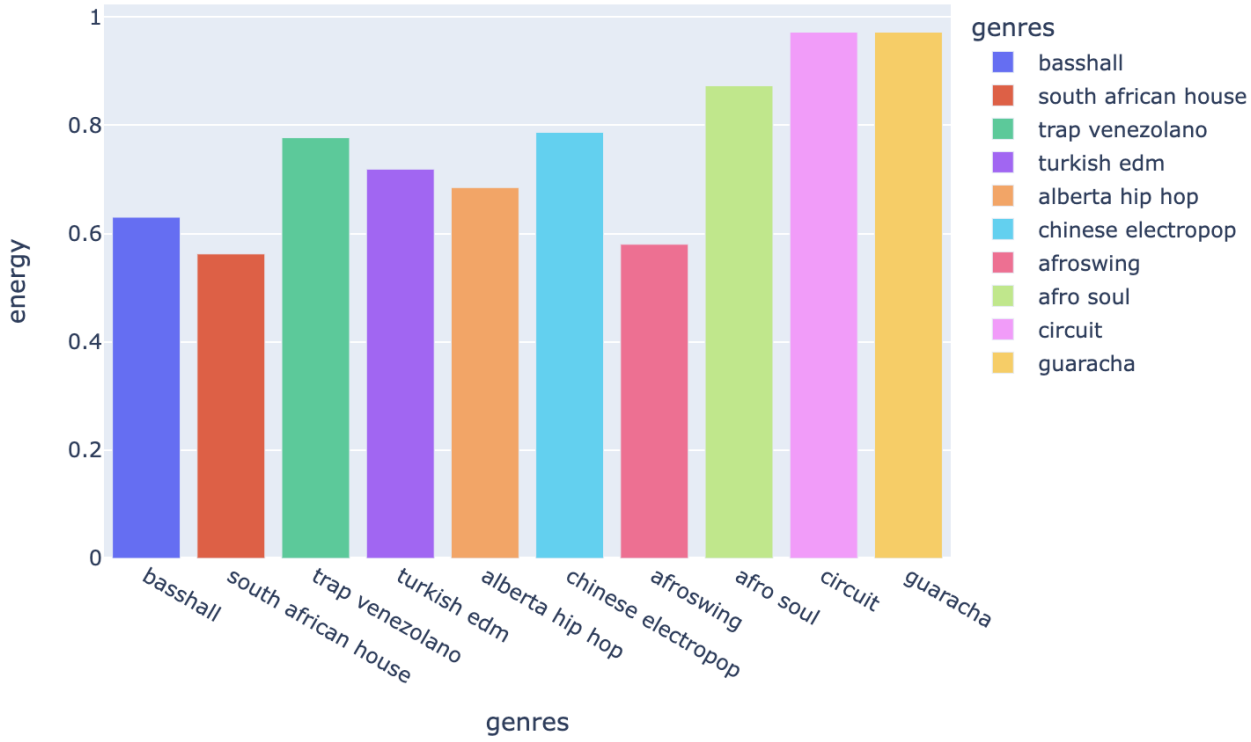


Fig.11 Energy distribution for top 10 popular genres

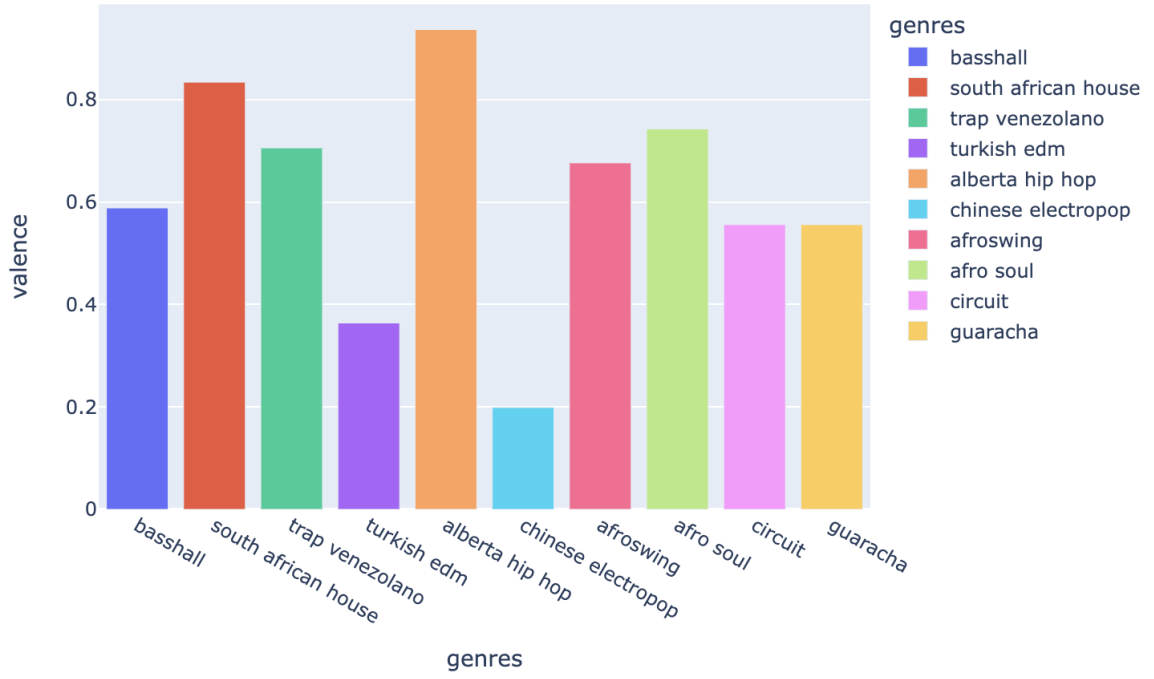


Fig.12 Valence distribution for top 10 popular genres

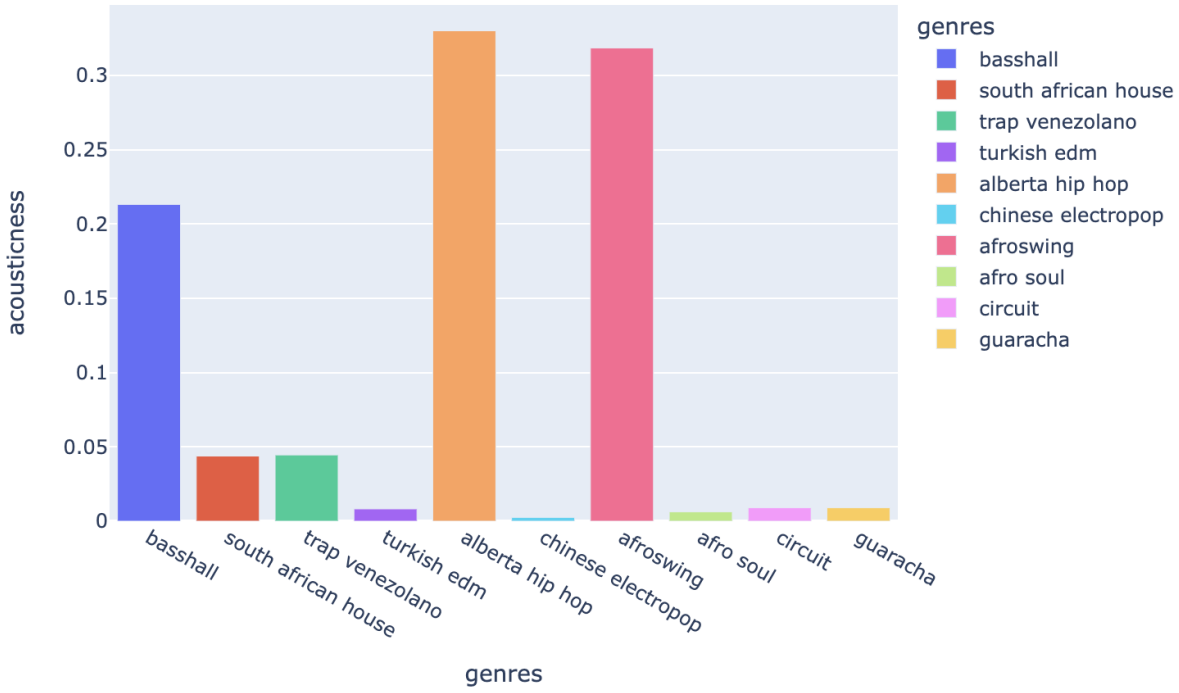


Fig.13 Acousticness distribution for top 10 popular genres

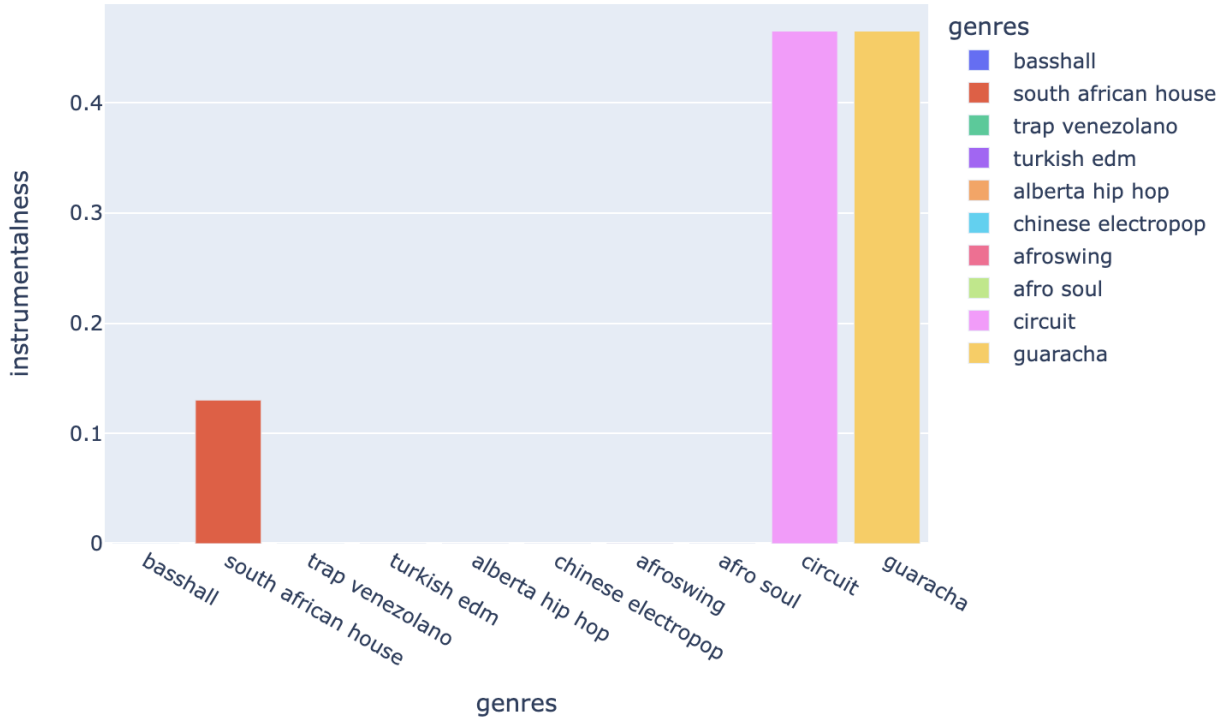


Fig.14 Instrumentalness distribution for top 10 popular genres

Fig.15 Genre WordCloud

3) Techniques and Results

3.1 Time Series Analysis

Time series analysis is most effective for datasets that cover a long time span. Our database includes over a century of song recordings. As the term suggests, a time series represents data distributed over time. In this study, we use song features provided by Spotify to visualize trends over time and explore how our musical preferences have changed. The moving average method is applied in our model to uncover interesting patterns in the data.

Moving average smoothing is a simple yet effective technique for forecasting time series. It is useful for processing data, generating new features, and even making direct predictions. This method reduces small fluctuations in data between time periods by smoothing, aiming to minimize noise and reveal the underlying patterns or signals in the data. In time series analysis and forecasting, moving averages are one of the most commonly used methods for smoothing.

To calculate a moving average, a new data series is created by averaging the values of the original data over a specific period, known as the window size or window width. This window size determines how many data points are included in each average.

The "moving" in moving average indicates that the window slides across the time series, recalculating the average for each step to create the new series. The two most common types of moving averages are centered moving averages, which balance the window around a data point, and trailing moving averages, which calculate the average based on previous data points.

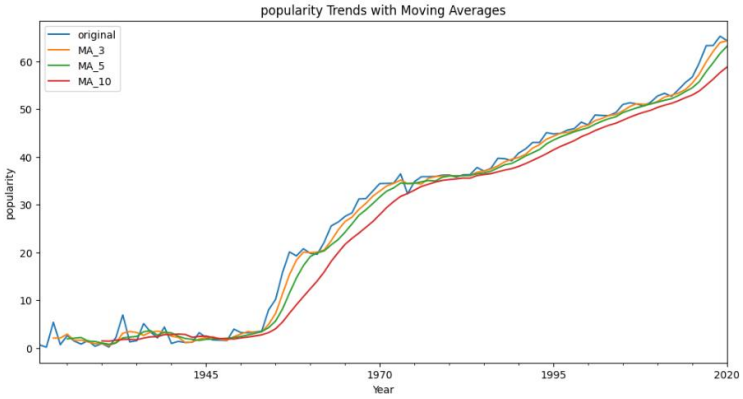


Fig.16

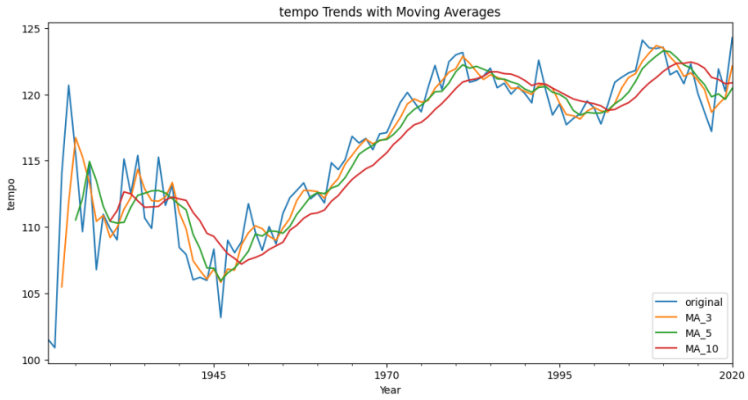


Fig.17

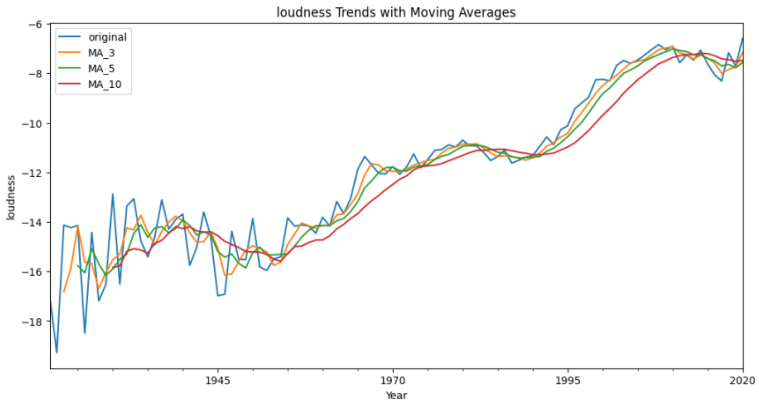


Fig.18

Popularity has steadily increased, with significant growth starting around the 1970s. Tempo has shown a gradual rise, peaking in the late 20th century before stabilizing in recent decades. Loudness has consistently increased over the years, reflecting modern production techniques that favor louder music. The use of moving averages effectively highlights these long-term trends by smoothing out short-term fluctuations, offering a clearer view of the overall evolution in music characteristics.

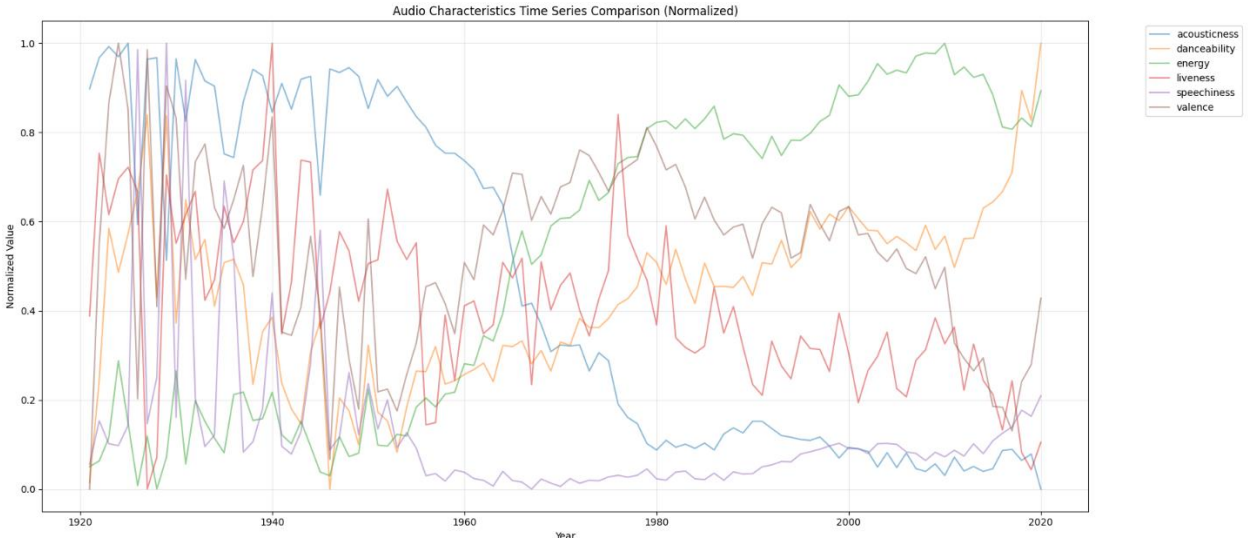


Fig.19

We selected the audio features 'acousticness,' 'danceability,' 'energy,' 'liveness,' 'speechiness,' and 'valence' for various songs, as well as the corresponding features for different genres. This information allows us to compare genres and gain insights into their unique sonic characteristics.

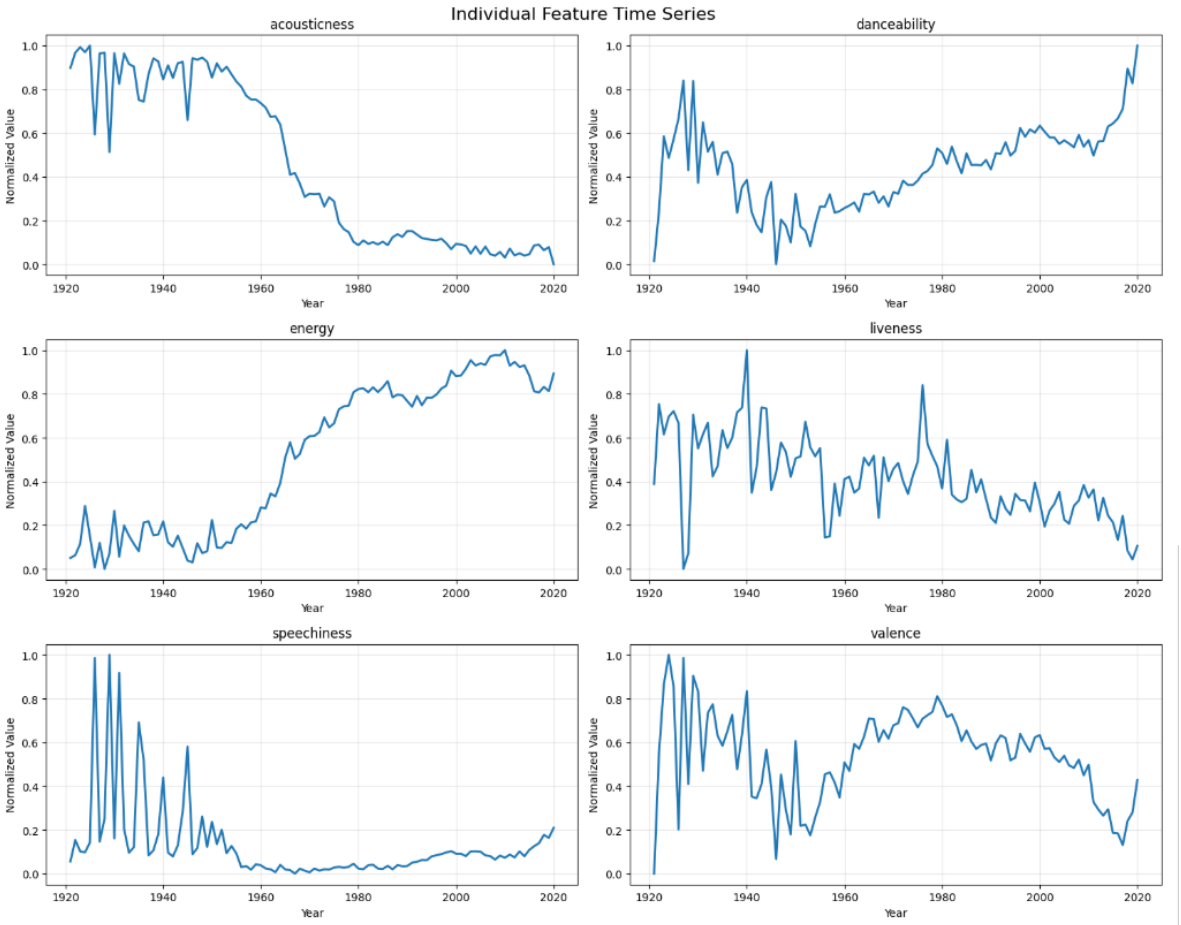


Fig.20

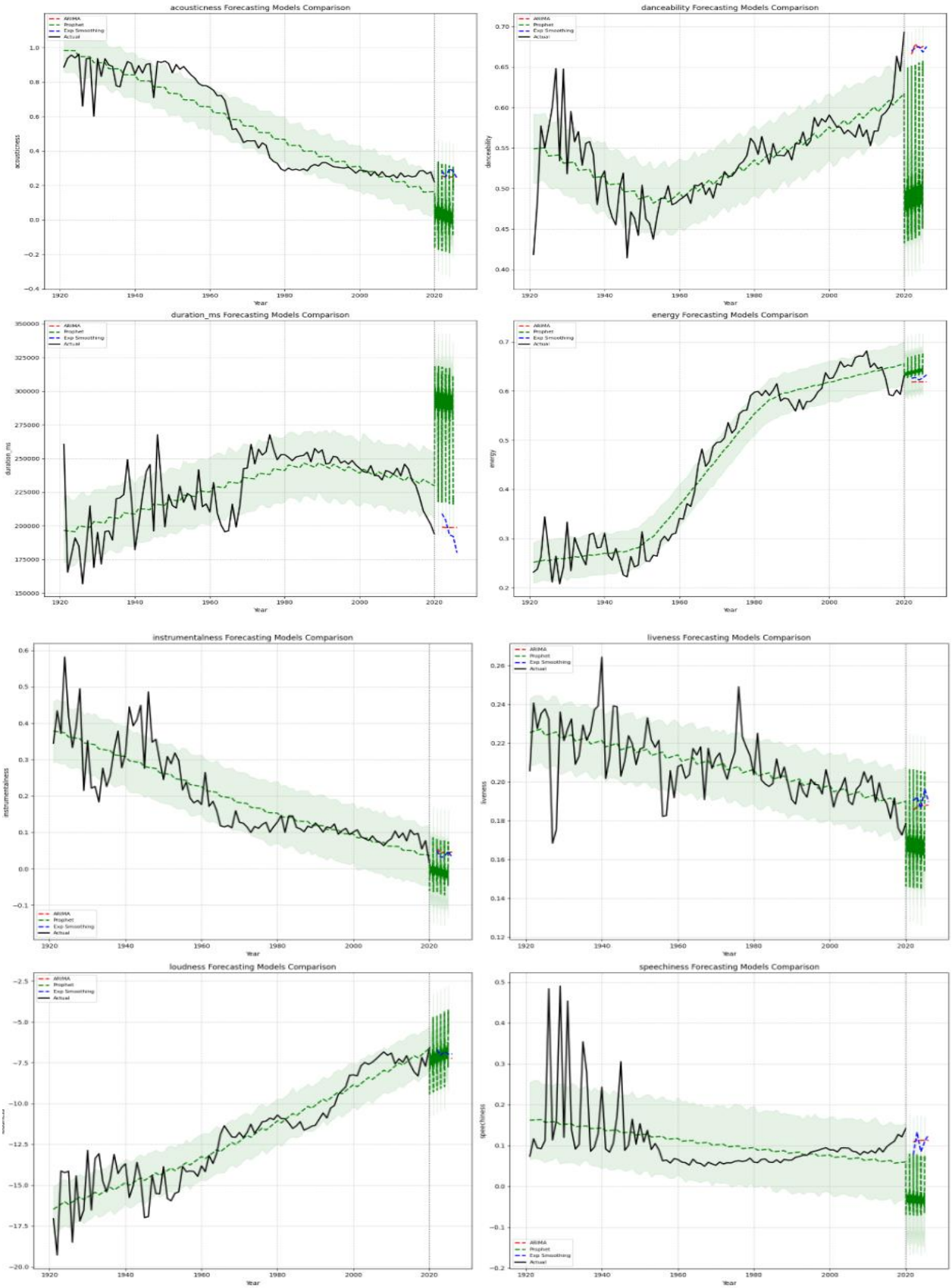
The analysis of normalized audio features over time reveals distinct trends in music evolution. Acousticness has declined, indicating a shift away from acoustic sounds, while danceability and energy have generally increased, reflecting a growing emphasis on upbeat and dynamic music. Liveness has remained stable, suggesting consistent inclusion of live performance elements, whereas speechiness shows low levels with occasional spikes, highlighting the rarity of spoken-word-heavy songs. Valence has slightly decreased in recent years, pointing to a potential shift towards less positive or more complex emotional themes in music. These trends illustrate the changing preferences and production styles in the music industry over the decades.

Seasonality Analysis:

Feature	Seasonal Strength	Pattern
acousticness	0.060	Weak
danceability	0.122	Moderate
duration_ms	0.149	Moderate
energy	0.040	Weak
instrumentalness	0.084	Weak
liveness	0.221	Moderate
loudness	0.076	Weak
speechiness	0.220	Moderate
tempo	0.113	Moderate
valence	0.145	Moderate
popularity	0.038	Weak

Fig.21

The seasonality analysis of music features reveals that attributes like danceability, duration, tempo, valence, and liveness have moderate seasonal patterns, suggesting some recurring trends over time. Meanwhile, features such as acousticness, energy, instrumentalness, loudness, and popularity show weak seasonality, indicating less consistent or less pronounced seasonal variations. Overall, some aspects of music demonstrate noticeable seasonal effects, while others vary with less regularity.


acousticness Forecasting Models Comparison
danceability Forecasting Models Comparison
duration_ms Forecasting Models Comparison
energy Forecasting Models Comparison
instrumentalness Forecasting Models Comparison
liveness Forecasting Models Comparison
loudness Forecasting Models Comparison
speechiness Forecasting Models Comparison
ARIMA
Prophet
Exp Smoothing
Actual
Year

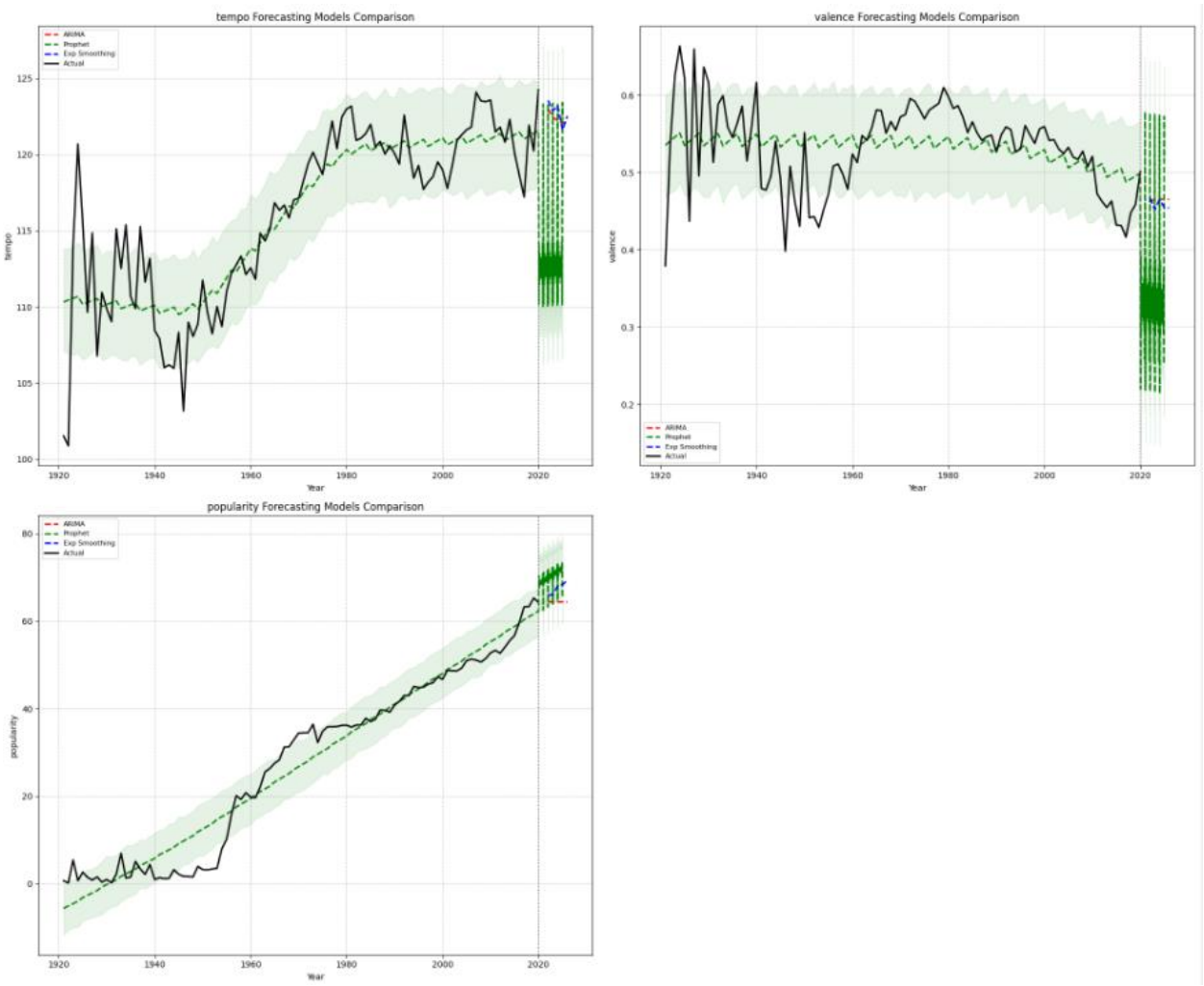


fig.22

Overall, Exponential Smoothing performs best in forecasting trends for different music features, closely aligning with both long-term trends and short-term fluctuations in the actual data. Prophet captures the general trends but lacks precision in capturing short-term variations, while ARIMA shows significant divergence in most cases, particularly in recent predictions. Therefore, Exponential Smoothing is the most suitable tool for forecasting these features, providing a more accurate representation of how music characteristics evolve over time.

Model Error Metrics Comparison Across Features:

MSE Comparison:

	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity
ARIMA	0.0004	0.0026	5.010809e+08	0.0022	0.0007	0.0002	0.3087	0.0006	5.0675	0.0048	33.1339
Prophet	2.2941	0.6965	7.470343e+10	0.9915	0.4443	0.0285	80.6431	0.0246	30696.2446	3.1604	1863.7915
Exponential Smoothing	0.0022	0.0022	5.835300e+08	0.0074	0.0030	0.0001	1.7321	0.0020	13.1469	0.0045	31.4390

RMSE Comparison:

	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity
ARIMA	0.0188	0.0511	22384.8368	0.0469	0.0263	0.0127	0.5556	0.0244	2.2511	0.0693	5.7562
Prophet	1.5146	0.8346	273319.2783	0.9957	0.6665	0.1687	8.9801	0.1567	175.2034	1.7778	43.1717
Exponential Smoothing	0.0466	0.0468	24156.3658	0.0859	0.0545	0.0109	1.3161	0.0452	3.6259	0.0672	5.6071

MAE Comparison:

	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity
ARIMA	0.0149	0.0349	16296.6344	0.0385	0.0223	0.0105	0.4311	0.0173	1.7380	0.0622	4.4363
Prophet	1.2484	0.6554	220434.8015	0.7732	0.4939	0.1261	6.6892	0.1441	135.5109	1.3982	35.1966
Exponential Smoothing	0.0359	0.0344	18419.5836	0.0731	0.0466	0.0097	1.1116	0.0378	2.7797	0.0618	4.6300

MAPE Comparison:

	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity
ARIMA	5.7530	5.4262	7.8215	6.3027	47.0175	5.7198	5.6244	14.7428	1.449	13.8851	7.2139
Prophet	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Exponential Smoothing	13.3466	5.4250	8.7037	11.9119	58.7935	5.2095	14.6300	35.3128	2.319	13.7228	7.6255

R2 Comparison:

	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity
ARIMA	-0.0569	-0.5067	-0.6772	-1.4029	-0.2075	-0.4911	-0.3939	-0.6348	-0.2947	-4.1259	-0.3507
Prophet	-6885.6043	-400.2150	-249.0423	-1082.8301	-773.3792	-261.2447	-363.1708	-66.4845	-7841.3448	-3370.5429	-74.9789
Exponential Smoothing	-5.5224	-0.2639	-0.9532	-7.0732	-4.1832	-0.0989	-6.8221	-4.6236	-2.3588	-3.8162	-0.2816

Fig.23

In summary, Exponential Smoothing generally outperforms ARIMA and Prophet across different evaluation metrics, including MSE, RMSE, and MAE, indicating higher prediction accuracy for most audio features. ARIMA performs moderately well compared to Prophet but is still less accurate than Exponential Smoothing. Prophet shows relatively high error values and poor performance, particularly with significantly negative R2 values. Despite the better performance of Exponential Smoothing, all models have negative R2 values, suggesting limitations in effectively capturing the variance in the data.

3.2 K-Means Clustering

Collaborative filtering is one of the earliest and most widely used techniques in recommendation systems. It often employs the **K-Nearest Neighbor (KNN)** method, leveraging users' historical data to calculate the similarity between different users based on their music preferences. This approach identifies the "nearest neighbor" users of the target user to evaluate and predict their preferences for specific items, thereby estimating how much the user might like a particular product.

On the other hand, **clustering** is an unsupervised learning technique that aims to identify homogeneous subgroups within the data. It groups data points into clusters so that points within a cluster are as similar as possible based on a chosen similarity measure, such as Euclidean distance. For this project, we will use **K-means clustering**, a popular algorithm for partitioning data into 'k' predefined distinct clusters. Each data point belongs to only one cluster, with the algorithm minimizing the sum of squared distances between the data points and the cluster centroids. K-means ensures that intra-cluster similarity is maximized while keeping clusters distinct from each other. Notably, the variation within clusters is inversely related to the homogeneity of the data points.

Euclidean Metric is given by:

$$d(x, x') = \sqrt{(x_1 - x_1')^2 + \ldots + (x_n - x_n')^2}$$

To determine the optimal value of **K** in K-Nearest Neighbors (KNN), it is essential to identify the **K nearest neighbors** of the unseen test data. The process involves assigning a class label to the test data point based on the majority vote or the highest probability derived from the class distribution of the K nearest data points.

When a test input xx is evaluated, it is classified into the class that has the highest frequency among its K neighbors. Essentially, the choice of K affects the model's performance:

- **Small K:** The model becomes sensitive to noise and overfits the training data.
- **Large K:** The model may oversimplify and fail to capture local patterns.

The class with the **highest probability** (or count) among the nearest neighbors is ultimately assigned to the test data point, making the selection of K crucial for achieving balanced accuracy and generalization.

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

The genres have been grouped into clusters, and to enhance our understanding, we can visualize these clusters in a two-dimensional space. This allows us to see how different genres or songs relate to each other spatially.

To further this analysis, **K-means clustering** can be applied to the songs dataset. This approach enables us to group songs into distinct clusters based on their features, helping to identify patterns and similarities within the data. For instance, songs with similar characteristics, such as tempo, genre, or key, will be located closer to each other in the plot, forming tight clusters.

By exploring the resulting plot, we can observe that songs within the same cluster often share some degree of similarity. This is a fundamental concept in content-based recommendation systems, where the proximity or similarity of songs is used to suggest new tracks to users based on their preferences. Such visualizations and clustering analyses provide valuable insights for improving recommendation systems.

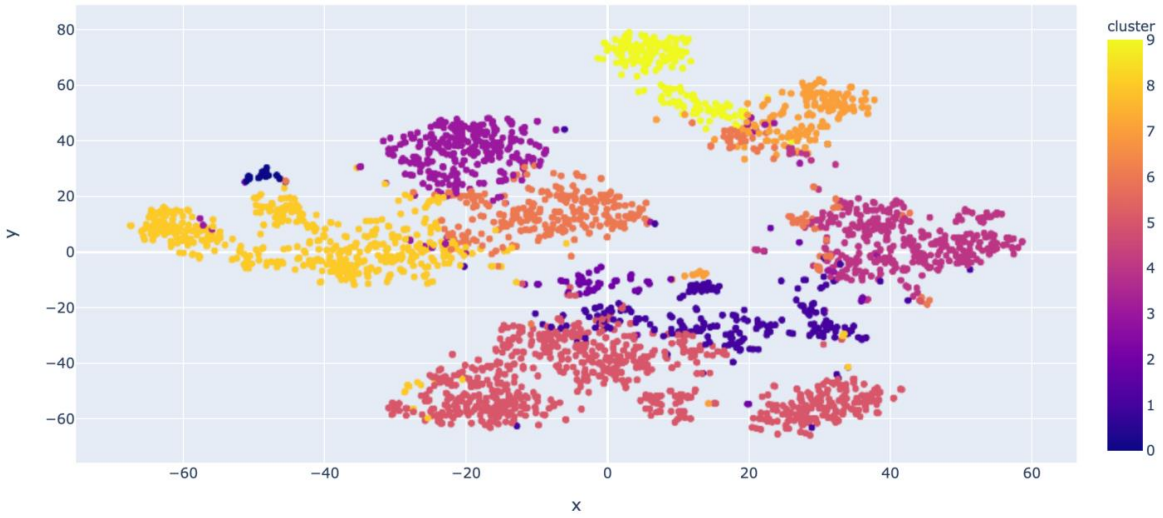


Fig.24

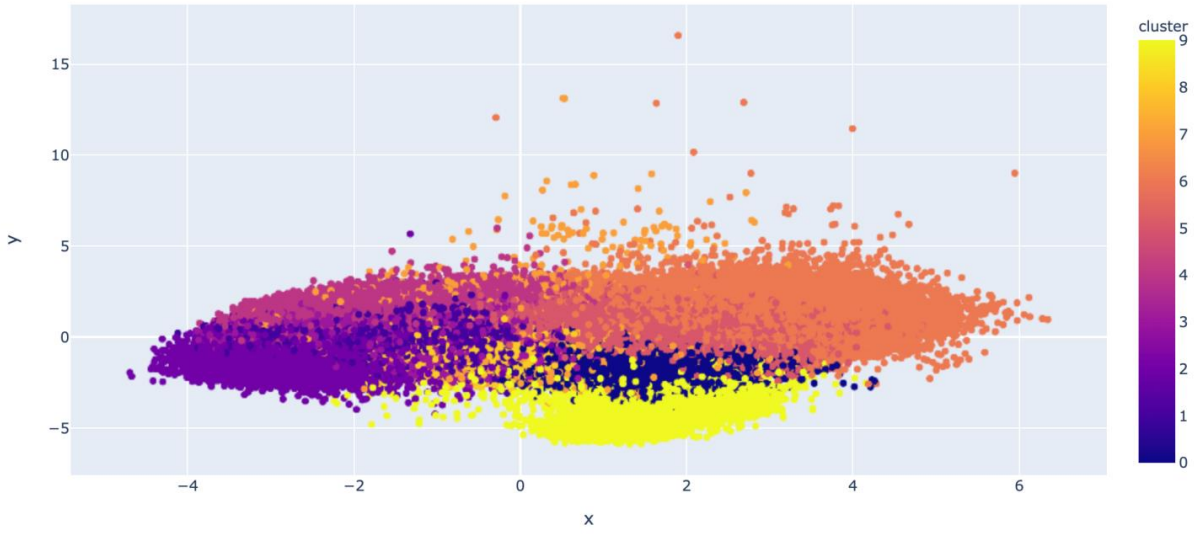


Fig.25

The optimal K value has been decided based on the SSE measure. The finalization of K=10 has been made based on the graphical representation.

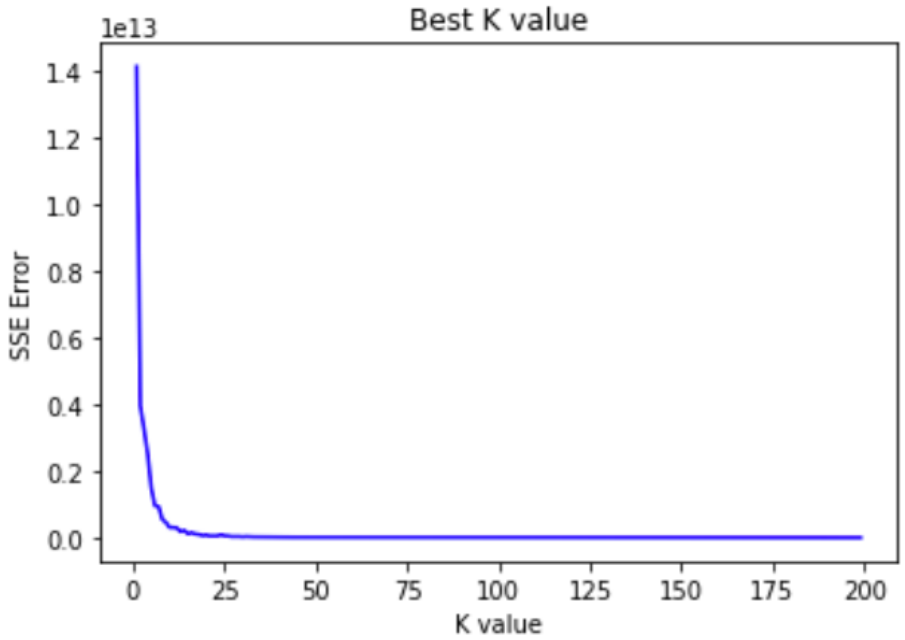


Fig.26

We further used Pyspark to perform the k mean clustering. We created a Spark Session using the following code:

```
#Creating Spark Session
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Recommendations').getOrCreate()
#Importing the necessary libraries
from pyspark.sql.functions import concat,col, when, trim,substring
from pyspark.sql.types import *
from pyspark.sql.types import StructType
```

We removed the columns which are not relevant and converted all the datatypes in string:

```
df = df.withColumn("valence",col("valence").cast("float"))
df = df.withColumn("acousticness",col("acousticness").cast("float"))
df = df.withColumn("danceability",col("danceability").cast("float"))
df = df.withColumn("energy",col("energy").cast("float"))
df = df.withColumn("instrumentalness",col("instrumentalness").cast("float"))
df = df.withColumn("key",col("key").cast("float"))
df = df.withColumn("liveness",col("liveness").cast("float"))
df = df.withColumn("loudness",col("loudness").cast("float"))
df = df.withColumn("popularity",col("popularity").cast("float"))
df = df.withColumn("speechiness",col("speechiness").cast("float"))
df = df.withColumn("tempo",col("tempo").cast("float"))
df = df.withColumn("count",col("count").cast("float"))
```

Since all the variables we are examining are either numerical or discrete numeric, we will utilize a Vector Assembler to convert them into features. A Vector Assembler is a transformer that combines a collection of features into a single vector column, commonly referred to as an array of features. We will also scale these features accordingly.

```
from pyspark.ml.feature import VectorAssembler
df.columns
assemble=VectorAssembler(inputCols=[
 'valence',
 'acousticness',
 'danceability',
 'energy',
 'key',
 'instrumentalness',
 'liveness',
 'loudness',
 'popularity',
 'speechiness',
 'tempo','count'], outputCol='features')
assembled_data=assemble.transform(df)
assembled_data.show(2)
```

```
from pyspark.ml.feature import StandardScaler
scale=StandardScaler(inputCol='features',outputCol='standardized')
data_scale=scale.fit(assembled_data)
data_scale_output=data_scale.transform(assembled_data)
data_scale_output.show(2)
```

Next, we dove into the exciting world of K-means clustering.

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
silhouette_score=[]
evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='standardized', \
                                metricName='silhouette', distanceMeasure='squaredEuclidean')
for i in range(2,10):

    KMeans_algo=KMeans(featuresCol='standardized', k=i)

    KMeans_fit=KMeans_algo.fit(data_scale_output)

    output=KMeans_fit.transform(data_scale_output)



    score=evaluator.evaluate(output)

    silhouette_score.append(score)

    print("Silhouette Score:",score)
```

The error metric used is the Silhouette score:

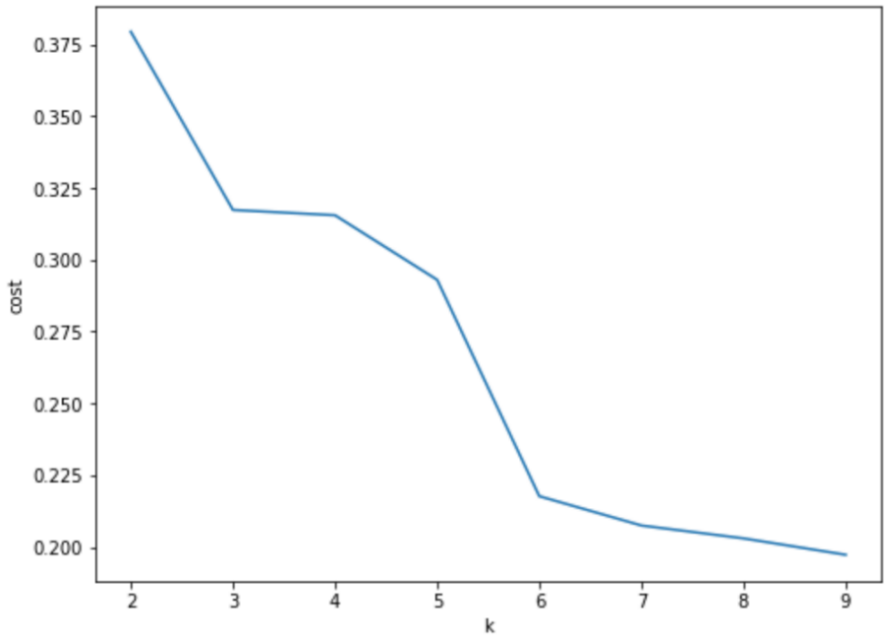


Fig.27

Based on the above graph we decided to use the k value of 5, as there is a drastic decrease after that point.

3.3 Recommender System

Recommendation systems are an evolution of information filtering technology designed to provide personalized suggestions based on user profiles without requiring explicit user feedback. These systems rely on attribute-based methods; users and items have profiles detailing their attributes. If the attributes of a user profile align with those of an item, the system suggests that item to the user.

In our dataset, each song or artist is classified with over ten attributes. When a user selects a song, the system recommends other songs with similar genres. Developing a supervised model for an attribute-based recommendation system involves estimating users' likelihood of enjoying an item they have not interacted with. The target variable for the model is whether the user likes the item, and the input features are derived from user and item attributes. Attribute-based methods are straightforward to implement and adapt to new users or items.

Using K-means clustering, users are grouped into distinct categories. The resulting matrix displays top recommendations made by the system, prioritizing items with high ratings and play counts. Attributes such as ratings and play counts are key to generating recommendation track IDs. The outputs presented are generated using PySpark and sk-learn, demonstrating the system's functionality.

```
recommend(100)                    ] recommend(100)
```

	prediction	rating	count
8602	0	5	1017.0
25406	1	5	953.0
24271	1	5	817.0
18762	0	5	528.0
7380	1	5	526.0

	genres	rating	playCount
12378	9	5	1061
24346	8	5	817
16829	17	5	456
27291	8	5	415
5832	17	5	408

Fig.28

Fig 16 contains the actual list of genres the user listens to. Our recommended genres were 9,8 and 17. They are present in the below array. Therefore, our recommender system has made accurate recommendations.

```
] # Checking which genres are user favourites and did they get same recommendations
  artists_df1[artists_df1.user_id==100].sort_values(by='rating')['genres'].unique()
```

```
array([ 4,  1,  9,  8, 15,  3,  7,  5, 17, 11, 10, 14, 12,  2, 13,  6, 16,
       18,  0, 19], dtype=int32)
```

Fig.29

4) Conclusion

Spotify maintains detailed metadata and audio features for songs, which can be leveraged to develop robust music recommendation systems.

Improvement Idea: Based on the recommendation analysis, we can further enhance user engagement by curating a biweekly mix. This mix could feature playlists with a diverse selection of genres and tracks that align with the user's listening habits, offering a personalized and dynamic music experience.

5) References

1. Deldjoo, Y., Schedl, M., & Knees, P. (2021). Content-driven Music Recommendation: Evolution, State of the Art, and Challenges. *arXiv preprint arXiv:2107.11803*. ([arxiv.org](https://arxiv.org/abs/2107.11803))

2. Jing, E., Liu, Y., Chai, Y., Yu, S., Liu, L., Jiang, Y., & Wang, Y. (2024). Personalized Music Recommendation with a Heterogeneity-aware Deep Bayesian Network. *arXiv preprint arXiv:2406.14090*. ([arxiv.org](https://arxiv.org/abs/2406.14090))

3. Luo, C., Wen, L., Qin, Y., Yang, L., Hu, Z., & Yu, P. S. (2024). Against Filter Bubbles: Diversified Music Recommendation via Weighted Hypergraph Embedding Learning. *arXiv preprint arXiv:2402.16299*. ([arxiv.org](https://arxiv.org/abs/2402.16299))

4. Singh, R., & Kanuparthi, P. (2023). Related Rhythms: Recommendation System To Discover Music You May Like. *arXiv preprint arXiv:2309.13544*. ([arxiv.org](https://arxiv.org/abs/2309.13544))

5. Gomez-Uribe, C. A., & Hunt, N. (2015). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4), 1-19.

6. Furnas, G. W. (2014). The Pandora Music Genome Project. *Proceedings of the 2014 International Conference on Music Information Retrieval*, 1-6.