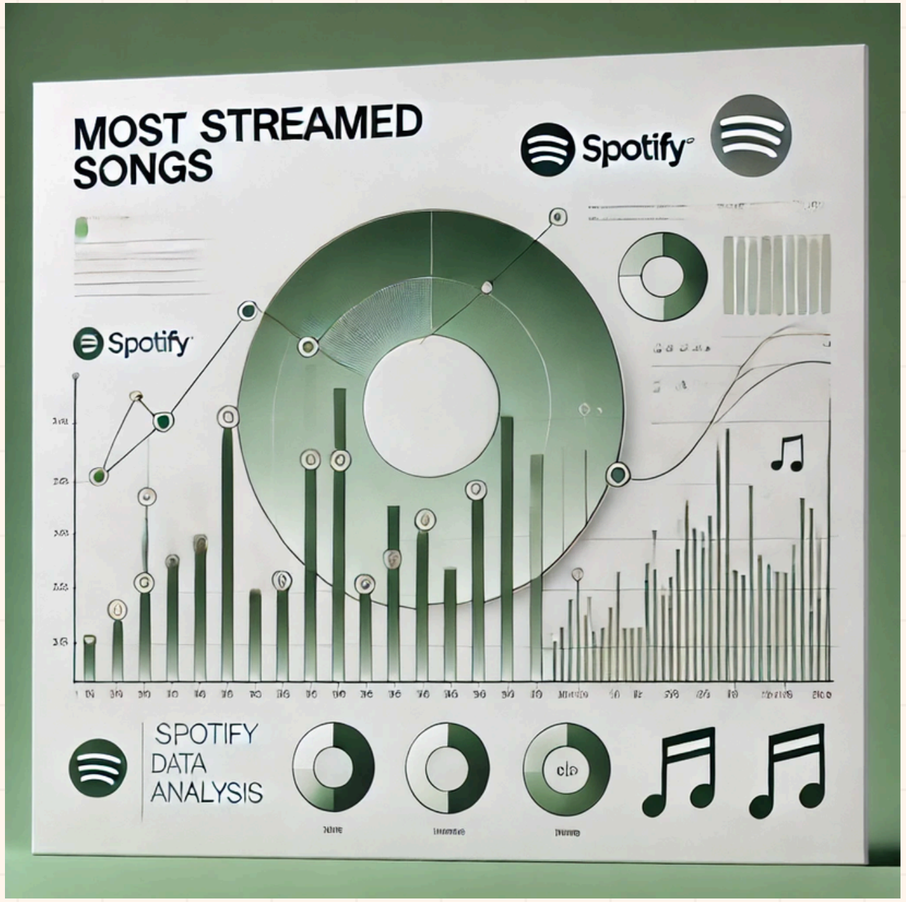
SPOTIFY MOST STREAMED SONGS DATA ANALYSIS

DATABASES AND BIG DATA



Ege John Işık
M. Emin Albayram
Berke Tayfun Akseki
Adasu Akel

PROJECT OVERVIEW

The goal of the project is to make a data analysis on the given dataset “Spotify Most Streamed Songs.csv” by using Python and SQL.

By using these two languages we made an application which can be used to analyze song attributes and get some interesting insights from the data.

We analyzed song attributes like danceability, energy and trends over the years.

DESCRIPTION OF DATASET

SPOTIFY MOST STREAMED SONGS

The dataset is composed of 25 columns which are describing the features of some of the most streamed songs on Spotify. It consists of features which give the details of :

1. **Basic Track Information,**
   - track_name: Name of the song,
   - artist(s)_name: Name of the artist(s) performing the song and etc.
2. **Streaming Metrics,**
   - in_spotify_playlists: Number of Spotify playlists the song is featured in,
   - streams: Total number of streams on Spotify and etc.
3. **Musical Attributes.**
   - bpm: Beats per minute, representing the tempo of the song.
   - key: Key of the song.

The dataset is stored as a CSV file and loaded into MySQL database using python.


Spotify Most Streamed Songs.csv > data

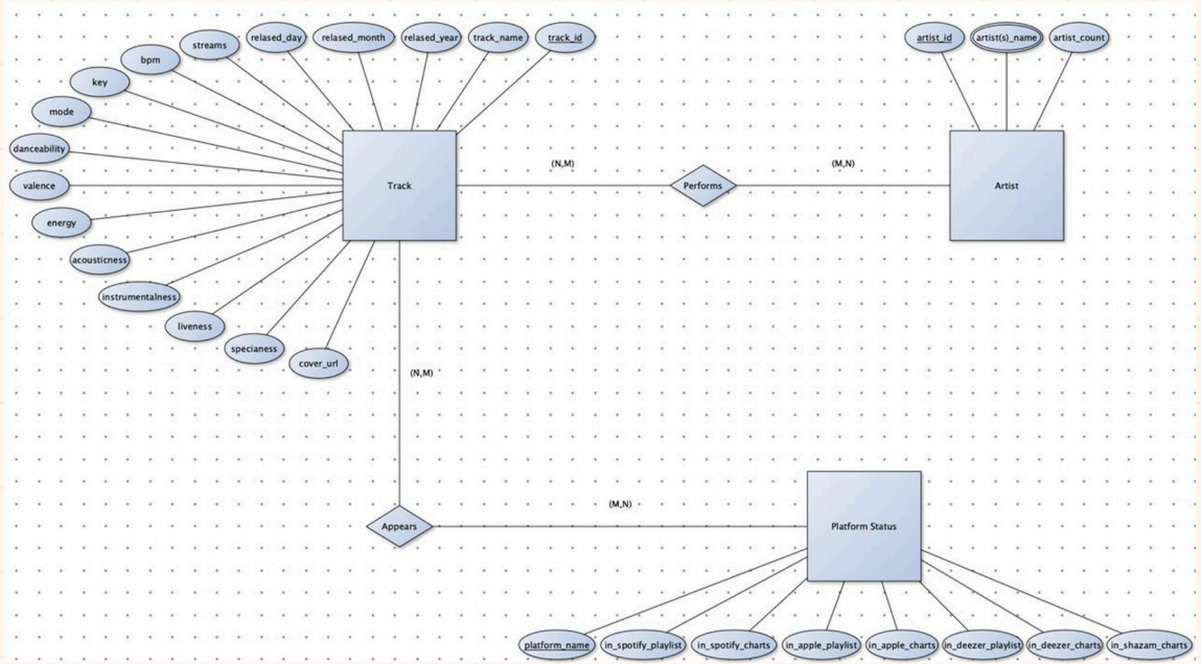
```
track_name,artist(s)_name,artist_count,released_year,released_m
Seven (feat. Latto) (Explicit Ver.),"Latto, Jung Kook",2,2023,7
LALA,Myke Towers,1,2023,3,23,1474,48,133716286,48,126,58,14,382
vampire,Olivia Rodrigo,1,2023,6,30,1397,113,140003974,94,207,91
Cruel Summer,Taylor Swift,1,2019,8,23,7858,100,800840817,116,20
WHERE SHE GOES,Bad Bunny,1,2023,5,18,3133,50,303236322,84,133,8
Sprinter,"Dave, Central Cee",2,2023,6,1,2186,91,183706234,67,21
Ella Baila Sola,"Eslabon Armado, Peso Pluma",2,2023,3,16,3090,5
Columbia,Quevedo,1,2023,7,7,714,43,58149378,25,89,30,13,194,100
fukumean,Gunna,1,2023,5,15,1096,83,95217315,60,210,48,11,953,13
La Bebe - Remix,"Peso Pluma, Yng Lvcas",2,2023,3,17,2953,44,553
un x100to,"Bad Bunny, Grupo Frontera",2,2023,4,17,2876,40,50567
Super Shy,NewJeans,1,2023,7,7,422,55,58255150,37,202,21,5,168,1
Flowers,Miley Cyrus,1,2023,1,12,12211,115,1316855716,300,215,74
Daylight,David Kushner,1,2023,4,14,3528,98,387570742,80,156,182
As It Was,Harry Styles,1,2022,3,31,23575,130,2513188493,403,198
Kill Bill,SZA,1,2022,12,8,8109,77,1163093654,183,162,161,12,187
Cupid - Twin Ver.,Fifty Fifty,1,2023,2,24,2942,77,496795686,91,
"What Was I Made For? [From The Motion Picture ""Barbie""]",Bil
Classy 101,"Feid, Young Miko",2,2023,3,31,2610,40,335222234,43,
Like Crazy,Jimin,1,2023,3,24,596,68,363369738,8,104,23,2,29,120
LADY GAGA,"Gabito Ballesteros, Junior H, Peso Pluma",3,2023,6,2
I Can See You (Taylor¡¼¡¼¡¼s Version) (From The  Taylor Swif
```

DATABASE STRUCTURE

There are five tables created in SQL to give the database a proper structure for dataset.

1. **Artist** (artist_id, artist(s)_name, artist_count)
2. **Track** (track_id, track_name, release info, streams, audio features…)
3. **Platform_Status** (song_id, playlist/chart data for Spotify, Apple, Deezer, Shazam…)
4. **Performs** (Relationship between Artist and Track)
5. **Appears** (Relationship between Track and Platform_Status)

THE ER DIAGRAM



SQL CODE EXPLANATION

```sql
-- Artist Table
CREATE TABLE Artist (
    artist_id VARCHAR(255) PRIMARY KEY,
    artist_name VARCHAR(255) NOT NULL,
    artist_count INT
);

-- Platform_Status Table
CREATE TABLE Platform_Status (
    song_id VARCHAR(255) PRIMARY KEY,
    in_spotify_playlist BIGINT,
    in_spotify_charts BIGINT,
    in_apple_playlist BIGINT,
    in_apple_charts BIGINT,
    in_deezer_playlist BIGINT,
    in_deezer_charts BIGINT,
    in_shazam_charts BIGINT
);
```

```sql
-- Track Table
CREATE TABLE Track (
    track_id VARCHAR(255) PRIMARY KEY,
    track_name VARCHAR(255) NOT NULL,
    released_year INT,
    released_month INT,
    released_day INT,
    streams BIGINT,
    bpm FLOAT,
    song_key VARCHAR(5),
    song_mode VARCHAR(10),
    danceability FLOAT,
    valence FLOAT,
    energy FLOAT,
    acousticness FLOAT,
    instrumentalness FLOAT,
    liveness FLOAT,
    speechiness FLOAT,
    cover_url TEXT
);
```

Firstly, we created the tables Artist, Track, and Platform Status with the format of the image on the top.

```sql
-- Performs Relationship Table
CREATE TABLE Performs (
    artist_id VARCHAR(255),
    track_id VARCHAR(255),
    PRIMARY KEY (artist_id, track_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (track_id) REFERENCES Track(track_id)
);
-- Appears Relationship Table
CREATE TABLE Appears (
    track_id VARCHAR(255),
    song_id VARCHAR(255),
    PRIMARY KEY (track_id, song_id),
    FOREIGN KEY (track_id) REFERENCES Track(track_id),
    FOREIGN KEY (song_id) REFERENCES Platform_Status(song_id)
);
```

Then, we created the relationship tables which are Performs and Appears with the format of the image on the bottom.

PYTHON EXPLANATION

In the beggining of the code, we simply asked the user to enter their MySQL credentials to set up a database connection.

Then we wrote a function that creates a database.

Once the MySQL connection established, the createdb() function creates SpotifyDB.

After database creation, the code executes the creattables() function which is responsible for creating tables for the SpotifyDB dataset.

It implements the SQL codes which are necessary for creating tables by using .cursor() object.

```
    try:
        # MySQL DB credentials
        user = input("Enter your MySQL username: ")
        password = input("Enter your MySQL password: ")
        # Attempt to connect to the database to validate credentials
        db = mysql.connect(host="localhost", user=user, passwd=password)
        break  # Exit loop if connection is successful
    except Error as e:
        print("--> Error: Invalid username or password. Please try again.")
```

```
# Database Creation
def createdb(user:str, passw:str):
    db = mysql.connect(host="localhost", user=user, passwd=passw)
    curs = db.cursor()

    databasecreation = """
                            CREATE DATABASE SpotifyDB
                       """

    curs.execute(databasecreation)
```

```
# Table Creation
def creattables(user:str,passw:str):
    db = mysql.connect(host="localhost", user=user, passwd=passw, database="SpotifyDB")
    curs = db.cursor()

    name_table_1 = """
                    CREATE TABLE Artist (" \
                    "artist_id VARCHAR(255) PRIMARY KEY, " \
                    "artist_name VARCHAR(255) NOT NULL, " \
                    "artist_count INT " \
                    ");
                """
```

PYTHON EXPLANATION

The dataload() function reads the “Spotify Most Streamed Songs.csv” file and inputs each value into the according table.

While doing this process, it assigns ID numbers to each row and inserts them into tables as well. This is used to connect the tables to each other.

With using .cursor() object, it implements the SQL queries which are necessary to load the data.

```python
# Data extraction and insertion
def dataload(user: str, passw: str):
    datafile = "Spotify Most Streamed Songs.csv"
    db = mysql.connect(host="localhost", user=user, passwd=passw, database="SpotifyDB")
    curs = db.cursor()

    data = pd.read_csv(datafile, index_col=False, delimiter=",", encoding="UTF-8")

    # Insert data into all tables using a single loop
    for index, row in data.iterrows():
        # Convert NaN values to None
        row = row.where(pd.notnull(row), None)

        # Prepare data for Artist table
        artist_names = row['artist(s)_name'].split(', ')
        artist_id = f"{index}"  # Use just the row index as the artist_id
        artist_count = len(artist_names)
        artist_query = """
                        INSERT IGNORE INTO Artist (artist_id, artist_name, artist_count) VALUES (%s, %s, %s)
                    """
        curs.execute(artist_query, (artist_id, ', '.join(artist_names), artist_count))
```

PYTHON EXPLANATION

In this part, the user_interface() function is defined.
It gives analyzing options to user.

According to the input it implements the SQL codes necessary for the related query.

Also, it gives the user an option to exit the program.

```python
# User Query Interface
def user_interface():
    db = mysql.connect(host="localhost", user=user, passwd=password, database="SpotifyDB")
    curs = db.cursor()

    print("\n Welcome to Spotify Data Analysis!")

    while True:

        print("\n Choose an option to analyze:")
        print("1. Yearly Trends: Top Song Per Year by Streams and Danceability")
        print("2. Top 5 High Valence Songs")
        print("3. Top 3 High Energy Years (>80%)")
        print("4. Top 10 Most Streamed Songs")
        print("5. Average Danceability by Release Year")
        print("6. BPM Distribution on Each Decade")
        print("0. Exit.")

        choice = int(input("\n Enter your choice: "))
```

```python
elif choice == 2:
    query = """
            SELECT track_name, streams, valence
            FROM track
            WHERE valence > 0.7
            ORDER BY streams DESC
            LIMIT 5;
        """
    curs.execute(query)
    results = curs.fetchall()
    # Convert results to a DataFrame for better formatting
    df = pd.DataFrame(results, columns=['Track Name', 'Streams', 'Valence'])
    print("\n Top 5 High Valence Songs: \n")
    print(df.to_string(index=False))  # Display DataFrame without the index
```

PYTHON EXPLANATION

Finally, there comes a main() function.

The purpose of this function is to let the user know about the database creation, table creation and data loading.

If any error doesn't happen during these processes, it runs the user interface and the program starts working efficiently.

```python
# Main Function
def main():


    try:
        print("Creating the database...")
        createdb(user=user, passw=password)
    except Error as e:
        print("--> The database already exists.")

    try:
        print("Defining tables...")
        creattables(user=user, passw=password)
    except Error as e:
        print("--> The tables already exist.")

    try:
        while True:
            # Ask user if they want to load the dataset again
            load_again = input("Do you want to load the dataset again? (y/n): ").strip().lower()
            if load_again == 'y':
                dataload(user=user, passw=password)
                print("--> Dataset loaded successfully.")
                break
            elif load_again == 'n':
                print("--> Skipped dataset loading.")
                break
            else:
                print("--> Invalid input. Try again")
    except Error as e:
        print("--> There is an error occurred while loading the dataset.")

    # Run the user interface
    user_interface()
```

SQL QUERIES

```
 Welcome to Spotify Data Analysis!

 Choose an option to analyze:
1. Yearly Trends: Top Song Per Year by Streams and Danceability
2. Top 5 High Valence Songs
3. Top 3 High Energy Years (>80%)
4. Top 10 Most Streamed Songs
5. Average Danceability by Release Year
6. BPM Distribution on Each Decade
0. Exit.

 Enter your choice:
```

The interactive console displays menu with query options and allows users to do specific data analysis and select one of the options.

We gave the user six different options:

1. **Yearly Trends:** The top song per year based on streams and danceability.
2. **Top 5 High Valence Songs:** Lists of songs with the highest “positivity”
3. **Top 3 High Energy Years:** Reveals the years with the highest proportion of energetic songs
4. **Top 10 Most Streamed Songs:** Shows the 10 most streamed tracks.
5. **Average Danceability by Year:** Shows how danceability has evolved over time.
6. **BPM Distribution by Decade:** Analyzes the distribution of BPM across decades

After they choose, users can enter their choice by the number and start the analysis.

SQL QUERY EXAMPLES

This query selects Top 3 Years with the Most High-Energy Songs from the track table using their released_year. Then identifies the songs with an energy level greater than 0.8 and label it as high energy songs. Then puts these results in a descending order.

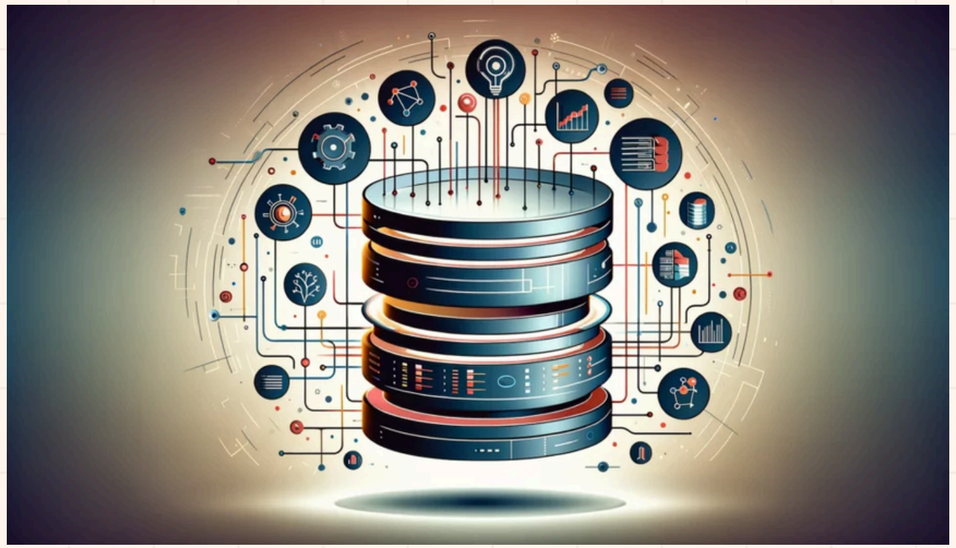
```sql
-- 3. Top 3 Years with the Most High-Energy Songs (>80%)
SELECT released_year, COUNT(*) AS high_energy_songs
FROM track
WHERE energy > 0.8
GROUP BY released_year
ORDER BY high_energy_songs DESC
LIMIT 3;
```

This SQL query retrieves the top 10 most-streamed songs from the track table. It selects them by their track_name and streams. Then it orders the songs in a descending order using their stream count.

```sql
-- 4. Top 10 Most Streamed Songs
SELECT track_name, streams
FROM track
ORDER BY streams DESC
LIMIT 10;
```

DESIGN AND IMPLEMENTATION DECISIONS

DataBase Structure



The Tracks table is created in order to store the data which is about the general features of tracks such as track id, name, released month and musical characteristics.

Artist table contains the names, number of artists and artist id.

Platform Status table also has an id and contains status of tracks on different platforms.

Separating the data into different tables generates a special place for us to do better categorization and analysis.

The **id columns** on each table are filled by Python code with using **indexes** giving each track a special id. The purpose of this process is to use the id numbers **to match tables** with each other.

DESIGN AND IMPLEMENTATION DECISIONS

Python Implementations

The given Python does the following functions with a user interface:

- **Creating the database,**
- **Creating the tables,**
- **Loading the data,**
- **Implementing the SQL queries.**

The user can choose to load the data again or not in case when the data is already loaded.

Also, the main function allows user to choose which queries to implement.

A **user friendly** interface makes the project more efficient.

DESIGN AND IMPLEMENTATION DECISIONS

Query Choices

While choosing which queries to implement, the main idea was to give **an insight to the global music taste** and **yearly trends.**

By looking at the output of each query, the user can make general assumptions about different timelines and their **social and musical features.**

1. YEARLY TRENDS: TOP SONG AND DANCEABILITY

WHAT IT DOES?

THE CODE FINDS THE MOST STREAMED SONG IN A YEAR AND THE ONE WITH THE HIGHEST DANCEABILITY.

WHY IT'S IMPORTANT?

IT SHOWS WHICH SONG WAS THE MOST POPULAR AND WHICH WAS THE EASIEST TO DANCE TO.

RESULT?

WE LEARN THE TOP SONG AND ITS DANCE VALUE FOR EACH YEAR.

Yearly Trends: Top Song Per Year by Streams and Danceability:

Released Year	Track Name	Streams	Danceability
1930	Agudo Mi¿½i¿½gi	90598517	65.0
1942	White Christmas	395591396	23.0
1946	The Christmas Song (Merry Christmas To You) - Remastered 1999	389771964	36.0
1950	Let It Snow! Let It Snow! Let It Snow!	473248298	60.0
1952	A Holly Jolly Christmas - Single Version	395591396	67.0
1957	Jingle Bell Rock	741301563	74.0
1958	Rockin' Around The Christmas Tree	769213520	70.0
1959	Let It Snow! Let It Snow! Let It Snow!	446390129	45.0
1963	It's the Most Wonderful Time of the Year	663832097	24.0
1968	Have You Ever Seen The Rain?	1145727611	74.0
1970	Feliz Navidad	520034544	50.0
1971	Happy Xmas (War Is Over)	460492795	33.0
1973	Dream On	838586769	39.0
1975	Bohemian Rhapsody - Remastered 2011	2197010679	41.0
1979	Wonderful Christmastime - Edited Version / Remastered 2011	403939487	75.0
1982	Pass The Dutchie	195918494	73.0

2. TOP 5 MOST STREAMED SONGS WITH HIGH VALENCE

WHAT IT DOES?

THE CODE LISTS THE TOP 5 STREAMED SONGS WITH HIGH VALENCE (POSITIVITY).

WHY IT'S IMPORTANT?

IT SHOWS WHICH HAPPY AND POSITIVE SONGS ARE MOST LOVED.

RESULT?

WE SEE PEOPLE'S PREFERENCE FOR CHEERFUL MUSIC.

Top 5 High Valence Songs:

Track Name	Streams	Valence
JGL	323455692	97.0
Zona De Perigo	134294498	97.0
Doja	482257456	97.0
There's Nothing Holdin' Me Back	1714490998	97.0
En El Radio Un Cochinero	164856284	97.0

3. TOP 3 YEARS WITH THE MOST HIGH-ENERGY SONGS

WHAT IT DOES?
THE CODE FINDS THE YEARS WITH THE MOST HIGH-ENERGY SONGS (ENERGY > 80%).

WHY IT'S IMPORTANT?
HIGH-ENERGY SONGS ARE FAST AND MOTIVATIONAL, SHOWING ACTIVE MUSIC TRENDS.

RESULT?
WE KNOW WHICH YEARS FOCUSED ON ENERGETIC MUSIC.

```
Top 3 High Energy Years (>80%):

Released Year  High Energy Songs
         2022                402
         2023                175
         2021                119
```

4. TOP 10 MOST STREAMED SONGS

WHAT IT DOES?
THE CODE LISTS THE TOP 10 SONGS WITH THE MOST STREAMS ON SPOTIFY.

WHY IT'S IMPORTANT?
IT SHOWS THE GLOBAL FAVORITES AND LONG-LASTING HITS.

RESULT?
WE LEARN WHICH SONGS PEOPLE LOVE THE MOST WORLDWIDE.

```
Top 10 Most Streamed Songs:

                                      Track Name    Streams
                                  Blinding Lights 3703895074
                                     Shape of You 3562543890
                                Someone You Loved 2887241814
                                    Dance Monkey 2864791672
Sunflower - Spider-Man: Into the Spider-Verse 2808096550
                                      One Dance 2713922350
                    STAY (with Justin Bieber) 2665343922
                                     Believer 2594040133
                                       Closer 2591224264
                                       Starboy 2565529693
```

5. AVERAGE DANCEABILITY BY YEAR

WHAT IT DOES?
THE CODE CALCULATES THE AVERAGE DANCEABILITY OF SONGS EACH YEAR.

WHY IT'S IMPORTANT?
IT HELPS US SEE WHICH YEARS PRODUCED MORE DANCEABLE MUSIC.

RESULT?
WE UNDERSTAND THE YEARLY TRENDS IN DANCE MUSIC.

Average Danceability by Release Year:

Released Year	Average Danceability
1930	65.00
1942	23.00
1946	36.00
1950	60.00
1952	67.00
1957	62.50
1958	70.67
1959	57.00
1963	37.00
1968	74.00
1970	51.50
1971	33.00
1973	39.00
1975	44.50
1979	75.00
1982	69.50
1983	82.00
1984	65.75
1985	63.50
1986	52.50
1987	45.00
1991	48.00
1992	53.00
1994	34.00
1995	70.00
1996	89.00
1997	74.00
1998	32.00
1999	77.40
2000	63.50
2002	73.83
2003	42.50
2004	68.50
2005	63.00
2007	52.00
2008	64.00
2010	61.71
2011	60.70
2012	61.70
2013	57.00
2014	63.54
2015	53.73
2016	63.00
2017	63.52
2018	55.00
2019	60.72
2020	67.43
2021	68.18
2022	68.71
2023	70.02

6. BPM DISTRIBUTION BY DECADE

WHAT IT DOES?
THE CODE GROUPS SONGS BY DECADE AND ANALYZES THEIR BPM (TEMPO).

WHY IT'S IMPORTANT?
IT SHOWS HOW MUSIC TEMPO HAS CHANGED OVER TIME.

RESULT?
WE SEE WHETHER MUSIC IN EACH DECADE WAS SLOW, MEDIUM, OR FAST.

BPM Distribution on Each Decade:

Decade	BPM Category	Average BPM
1930	Fast	130.00
1940	Fast	139.00
1950	Very Fast	163.50
1960	Very Fast	202.00
1970	Fast	147.50
1980	Very Fast	165.50
1990	Very Fast	164.50
2000	Very Fast	171.33
2010	Very Fast	170.40
2020	Very Fast	170.09

THANK YOU