Coursework Assignment Specification
(100 % of the module assessment)
Submission Deadline: at 11am on Monday 20th January 2025

Song Dataset Analyser

1 Description

1.1 Overview

Your task is to develop a simple song dataset analyser for a user. The analyser will use a publicly available dataset named songs.csv. This dataset contains information on the top 2000 tracks on Spotify from 1998 to 2020. It includes details on various attributes of each song, such as artist, song title, duration, popularity, danceability, energy, and genre. These features allow for an in-depth analysis of musical trends and characteristics. A detailed description of the dataset columns is provided in the source link.

1.2 Store Cleaned Data

Your first task is to develop a Python program to prepare the song dataset for analysis by first cleaning and formatting the data, then storing it in SQLite database tables. This preprocessing step ensures the dataset is consistent, relevant, and structured for efficient querying.

Detail of the Task:

A. Load and Rename Columns:

- Load the dataset into a pandas DataFrame named dfSongs.
- Rename the column duration_ms to duration and convert its values from milliseconds to seconds, rounding to the nearest whole number.

B. Apply Filters: Filter the data to retain only the relevant records which are:

- Songs with a Popularity value greater than 50, ensuring focus on widely recognised tracks.
- Songs where Speechiness values are between 0.33 and 0.66, filtering for tracks with moderate speech content.
- Songs with Danceability values greater than 0.20, selecting tracks with measurable rhythm and groove.

C. Database Creation:

- Create a SQL database named CWDatabase.db.
- Programmatically define SQL tables according to the following data model, ensuring correct data types and constraints (e.g., primary and foreign keys).
- Populate the tables with the cleaned data from dfSongs.

Data Model:

Table Name	Song	Genre	Artist
Columns	ID, Song, Duration, Explicit, Year, Popularity, Danceability, Speechiness	ID, Genre	ID, Artist Name

Save the program as CW_Preprocessing.py.

1.3 Genres

Write a Python program to produce statistics for each genre in a given year. The program must use the relevant records in CWDatabase.db.

The program should first prompt the user to input a specific year. It must validate this input to ensure it falls within the dataset's range (1998–2020). Using the input, the program will query the database to retrieve all records for the specified year. The results should include a summary for each genre, displaying key statistics such as the average danceability, the total number of songs, and, where relevant, additional metrics like the average popularity. The summary must include at least three meaningful columns.

To make the output more informative, at least one of the columns must be visualised using Matplotlib. A pie chart, for example, could effectively display the total number of songs per genre. The program should also handle cases where no data exists for a given year, displaying a clear message to inform the user.

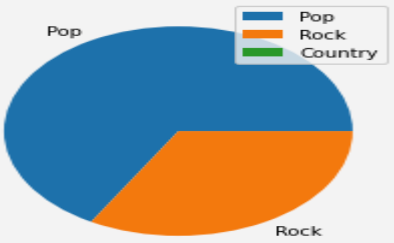
This program should be designed to dynamically handle any variations in the data, such as new genres or missing values, ensuring robustness and flexibility. The output should be clear and well-structured, offering both a tabular summary and a visual representation to support analysis.

Example Output:

Songs in Year 2016:

Genre	Average Danceability	…	Total Songs
Pop	0.3	…	100
Rock	0.7	…	90
Country	Null	Null	Null

Danceability



Save the program as Genres.py.

1.4 Artists

Write a Python program to display the average popularity of a single artist across multiple genres. The key metric is the average popularity of the artist's songs in each genre. Additionally, the program must include the overall popularity of each genre (averaged across all artists) to allow for comparisons between the artist's performance and general trends.

The program should prompt the user to input an artist's name and validate the input by checking its existence in CWDatabase.db. Once validated, the program should calculate the average popularity of the artist's songs within each genre and compare these values to the overall genre popularity. The results should be presented in a tabular format where any genre where the artist's popularity is above the overall average should be clearly highlighted (e.g., bold text or a different background colour).

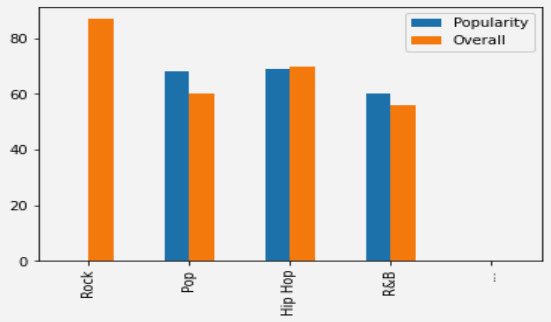
To provide additional insight, the program should also visualise the popularity data using a bar chart created with Matplotlib. This chart can represent the artist's popularity across genres alongside the overall genre averages.

The program must include error handling to manage cases where the artist is not found in the database or where no data exists for specific genres. It should also be adaptable to new data, ensuring that additional genres or missing values are handled carefully.

Example Output: (Highlighted Rows Indicate Above-Average Popularity):

Popularity table for Aaliyah.

Genre	Popularity	Overall
Rock	Null	87
Pop	68	60
Hip Hop	69	70
R&B	60	56
…	…	…



Save the program as Artist.py.

1.5 Top Artists

Write a Python program to identify the top 5 artists within a user-specified range of years. The program should calculate a ranking value for each artist based on criteria such as the total number of songs released by an artist within the specified range of years and the average popularity of those songs. Students are expected to develop their own formula for calculating the ranking value. For example, the formula could be structured as:

> Rank Value=(Number of Songs×Weight$_x$)+(Average Popularity×Weight$_y$)
> where Weight_x and Weight_y represent the relative importance of each factor.

The program should prompt the user to input the start and end years for the analysis. It must validate the input to ensure the years fall within the dataset range (1998–2020). Based on this input, the program should query the CWDatabase.db to retrieve the relevant data and calculate the top 5 artists for the specified period.

The output should be a table showing the yearly rank values for each of the top 5 artists, along with their overall average rank value. The table must highlight the top artist for each year (e.g., bold text or a different background colour). The table should be sorted by the overall rank value, with the highest-ranking artist at the top.

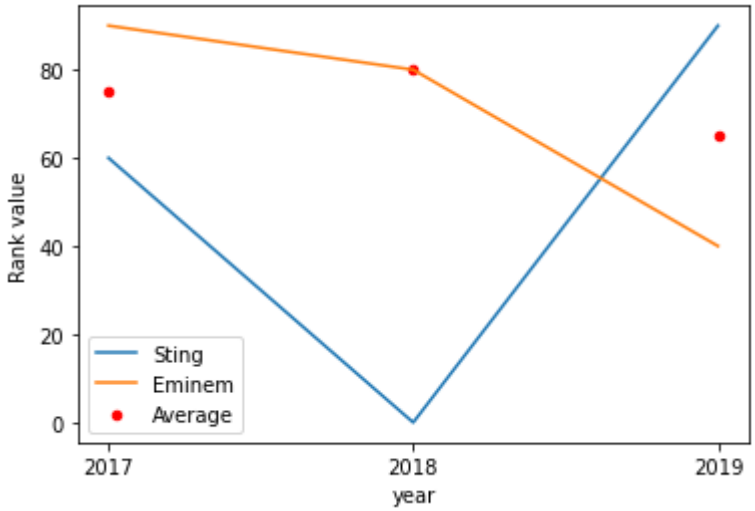
Example Output:
Top Artists Table (2017–2019):

Artist	2017	2018	2019	Average
Sting	60	Null	90	75
Eminem	90	80	40	70
…	…	…	…	…
Average	75	80	65	…

To enhance understanding, the program should visualise the yearly rank values using a line chart created with Matplotlib. The chart should include:

- A line for each artist showing their yearly rank values.
- A line or marker indicating the average rank value for each year.

Example Chart:



The program must include error handling to address cases where no data exists for specific years or artists. It should also be flexible enough to handle new data or missing values without causing errors.

Save the program as Top5.py.

2 What to Submit

To successfully complete your submission, ensure that the following files are included:

2.1. Data Files

- CWDatabase.db: An SQLite database containing three tables that store the details of each song.
- Songs.csv: The original dataset provided on Learn. You are not allowed to modify this file.

2.2. Program Files

- **CW_Preprocessing.py:** A program containing functions to clean the dataset and store the data in SQLite tables, as outlined in Section 1.2.
- **Genres.py:** A program containing functions to generate genre-based statistics for a given year, as described in Section 1.3.
- **Artist.py:** A program containing functions to compute and display an artist's popularity across genres, as explained in Section 1.4.
- **Top5.py:** A program to calculate and display the top 5 artists within a given range of years, as described in Section 1.5.
- **menu.ipynb:** A Jupyter notebook implementing the main menu for accessing all functionalities. This menu should use a graphical user interface (GUI) developed with Jupyter Widgets. The GUI must be intuitive and provide clear instructions for the user to navigate through the features. Ensure that all results from the functionalities are presented within the GUI itself.

Compress all the files (including the SQLite database and the CSV file) into a single ZIP file. Submit this ZIP file electronically as directed on the module Learn page.

3 Notes on Expectations

Your coursework will be assessed based on the Assessment Matrix provided separately. Below is an outline of general expectations to help you understand the standards required for this assignment:

Technical Proficiency: Your programs should demonstrate a strong understanding of Python programming principles and effective use of libraries like Matplotlib, Pandas, and SQLite. Solutions must efficiently solve the given tasks while holding to good coding practices.

Design and Usability: Your programs must be well-structured to ensure they are:

- Easy for users to operate, with clear inputs and outputs.
- Readable and maintainable for future development by other programmers.
- Designed for reusability, enabling portions of the code to be used in related applications.

Clarity and Self-Documentation : Given the structure of your programs, they should be as easy to read and understand as possible. Lay your code out so that it can be listed sensibly on a variety of devices: avoid having any lines longer than 80 characters as these may wrap (to reduce the number of "problem lines" you should use 4 spaces for indentation rather than tabs). Sensible names should be chosen for all variables, methods etc. Include a markdown cell to describe any special instructions or noteworthy aspects (limited to 200 words). Documentation strings should be included for each:

Program: Fully explain what the program does and how it should be used. Also state who (ID only) wrote it and when.

Function/method: State what each function does and explain the roles of its parameters.
In addition, you should include occasional comments in your code; these may be (a) to introduce a new section in the code, or (b) to explain something that is not obvious. Bear in mind that pointless comments make your code harder to read, not easier.

Structural requirements and Restriction

- Object-Oriented Design: Your code might include class definitions where appropriate to enhance reusability and maintainability.
- Database Use: The program must store all data in the SQLite database.
- Conda Environment: Your code must be able to run in the provided Conda environment (specified via the environment YAML file). The environment includes:
  - Python 3.10
  - Standard Python libraries
  - Third-party libraries: Matplotlib, Pandas and Numpy.