**Wilhelm Travers**

# Billboard “The Hot 100” Songs

## *Introduction*

### *Overview*

This Python script processes a CSV file named 'charts.csv', analyzing Billboard chart data to determine and showcase the top 100 artists based on their ranking scores over a specific period. It employs various functions to calculate scores, determine date ranges, and extract relevant artist information from the provided dataset.

### *Purpose*

The primary objective of this script is to identify the top-performing artists based on their rankings in the 'charts.csv' file and generate a new CSV file, 'top_100_artists.csv', containing essential details about these top artists. The ranking score for each artist is calculated based on their positions in the Billboard charts.

### *Functionality*

The script utilizes functions to determine date differences, calculate scores from rankings, and organize artist data. It reads the 'charts.csv' file, processes the information, identifies the top 100 artists, and creates a new CSV file summarizing their details.

### *Data Source*

The code assumes the presence of a 'charts.csv' file containing columns such as 'date', 'rank', 'song', 'artist', 'last-week', 'peak-rank', and 'weeks-on-board'. The 'artist' column holds the names of artists, and 'rank' signifies their positions in the Billboard charts.

### *Notes*

The user is expected to provide the 'charts.csv' file with the specified columns and format for successful execution.

## *Functions*

### *min_date(date_1, date_2)*

**Description:** Determines the minimum date between two given dates.

**Parameters:**

- **date_1:** First date for comparison.
- **date_2:** Second date for comparison.

**Returns:** The smaller of the two input dates.

**Usage:** “min_date(date_1, date_2)”

***max_date(date_1, date_2)***

**Description:** Determines the maximum date between two given dates.

**Parameters:**

- **date_1:** First date for comparison.
- **date_2:** Second date for comparison.

**Returns:** The larger of the two input dates.

**Usage:** “max_date(date_1, date_2)”

***position_to_score(position)***

**Description:** Converts a position to a score using a simple formula (101 - position).

**Parameters:**

- **position:** The position in the chart.

**Returns:** Score calculated based on the input position.

**Usage:** “position_to_score(position)”

***time_delta(min_date, max_date)***

**Description:** Calculates the time difference (in days) between two dates.

**Parameters:**

- **min_date:** The starting date.
- **max_date:** The ending date.

**Returns:** Time difference in days (max_date - min_date).

**Usage:** “time_delta(min_date, max_date)”

***Notes***

These functions are utilized within the script to perform date comparisons, score calculations, and duration calculations between dates. They serve crucial roles in the determination of artist rankings and career lengths.

## *Variables and Data Structures*

***linecount***

**Description:** A counter to keep track of the number of lines read from the 'charts.csv' file.

**Usage:** Incremented for each line read from the CSV file to count the total number of lines.

***artists***

**Description:** A dictionary storing information about artists extracted from the 'charts.csv' file.

**Structure:** Key-value pairs with the artist's name as the key and a dictionary containing artist details (such as score, min_date, max_date) as the value.

**Usage:** Stores and updates artist information during data processing.

***header***

**Description:** A list defining the expected column headers in the 'charts.csv' file.

**Structure:** Contains strings representing column names such as 'date', 'rank', 'song', 'artist', 'last-week', 'peak-rank', 'weeks-on-board'.

**Usage:** Used to map CSV columns to their respective indices during file reading and data extraction.

***artist_out***

**Description:** A list holding artist information in a structured format for further processing.

**Structure:** Contains dictionaries, each representing an artist's details.

**Usage:** Used to sort and display the top 100 artists based on their scores.

### *cols*

**Description:** A list defining columns for the output CSV file, 'top_100_artists.csv'.

**Structure:** Contains strings representing columns such as 'rank', 'artist', 'min_date', 'max_date', 'score', 'career_length (days)'.

**Usage:** Defines the order and content of columns in the output CSV file.

### *Notes*

These variables and data structures play pivotal roles in storing, organizing, and processing data obtained from the 'charts.csv' file. They facilitate the extraction and manipulation of artist-related information, contributing to the identification and summarization of top artists.

## *Code Explanation*

### *CSV File Processing*

The script begins by opening the 'charts.csv' file and initializing variables for processing. It reads the CSV file using the csv.reader module, iterating through each row. During iteration, it populates a dictionary named artists with artist information based on their rankings in the Billboard charts. The code skips the header row by checking if the 'rank' field equals 'rank'.

### *Top 100 Artists Generatio*n

After processing the CSV file, the script identifies the top 100 artists based on their ranking scores. It sorts the artist information stored in artist_out by their scores in descending order using the sort method and the lambda function. It then prints the top 100 artists to the console, displaying their names and related information.

### *CSV Output*

To create a new CSV file named 'top_100_artists.csv', the script initializes a list of columns cols required for the output. It opens a new CSV file in write mode and writes the column headers

using csv.writer. Then, it iterates through the top 100 artists stored in artist_out, extracting specific columns' data and writing it to the output CSV file using csvwriter.writerow.

***Notes***

The code orchestrates the entire process of reading, analyzing, and summarizing artist data from the 'charts.csv' file, culminating in the generation of a new CSV file containing details of the top 100 artists based on their rankings and duration on the Billboard charts.

***Usage Instructions***

***Input Requirements***

To successfully execute this Python script, ensure the following:

- **Input File:** Provide a CSV file named 'charts.csv' containing the following columns: 'date', 'rank', 'song', 'artist', 'last-week', 'peak-rank', 'weeks-on-board'. The 'artist' column should contain the names of artists, and 'rank' should denote their positions in the Billboard charts.

***Execution Steps***

Follow these steps to use the script:

**Data Preparation:**

- Ensure the 'charts.csv' file adheres to the specified column structure and contains valid data for analysis.

**Running the Script:**

- Place the Python script file in the same directory as the 'charts.csv' file.
- Execute the script in a Python environment using your preferred method (e.g., command line, IDE, text editor).

**Output:**

- Upon execution, the script will process the 'charts.csv' file and generate a new CSV file named 'top_100_artists.csv' in the same directory. This file will contain details of the top 100 artists based on their rankings.

***Execution Command (Command Line)***

Run the script 'data_analysis.py' in a command-line interface (CMD/Terminal):

“python data_analysis.py”

***Notes***

- Ensure proper file paths and permissions when executing the script.
- The output file 'top_100_artists.csv' will contain information about the top 100 artists ranked based on their scores derived from the 'charts.csv' data.

## *Additional Notes*

***Assumptions***

- The script assumes the presence of a 'charts.csv' file with the specified columns and a well-formatted structure. Ensure the input file matches the expected format to prevent errors during processing.
- It assumes valid date formats (ISO format) in the 'date' column of the CSV file for accurate date comparisons.

***Limitations***

- The script processes the entire 'charts.csv' file to identify the top 100 artists, which might be resource-intensive for significantly large datasets. Consider system capabilities while handling extensive data.

***Potential Enhancements***

- Optimization: Implementing optimizations for handling larger datasets, such as batch processing or efficient memory usage, could improve script performance.
- User Interaction: Adding user prompts or command-line arguments to customize input file names or output locations could enhance user experience.

***Maintenance***

- Regularly review and update the code to ensure compatibility with future Python versions or potential changes in data structures or input file formats.

***Notes***

- This documentation aims to provide an overview of the script's functionality, usage, and potential areas for improvement. Adaptations and modifications may be necessary based on specific use cases or evolving requirements.