Recommendation System

Applying Machine Learning in Real Life

Supervised by Bruno Yun

Code:

https://colab.research.google.com/drive/18qwhDJApTnHW2LwkMi_ml9tVOqkqsKfN?usp=sharing

Mekondoum Eric -12317177 -eric.mekondoum@etu.univ-lyon1.fr

M2 DISS 2023/2024

Université Claude Bernard Lyon 1

Table of Contents

Recommendation System

Introduction

Project Overview

We are given a dataset listing the songs some users have listened to. The columns of the dataset are :

- “user” is the ID of the listener
- “song” is the ID of the song listened
- “listen_count” is the number of times the song was listened to by the user.
- “title” is the name of the song
- “release” is the name of the album
- “artist_name” is the name of the artist
- “year” is the release year.

From this dataset, we should implement a song recommendation system.

Work Distribution

For this project, we did not separate the tasks, all members contributed equally in each part of this project.

Data Exploration

Most Listened Songs

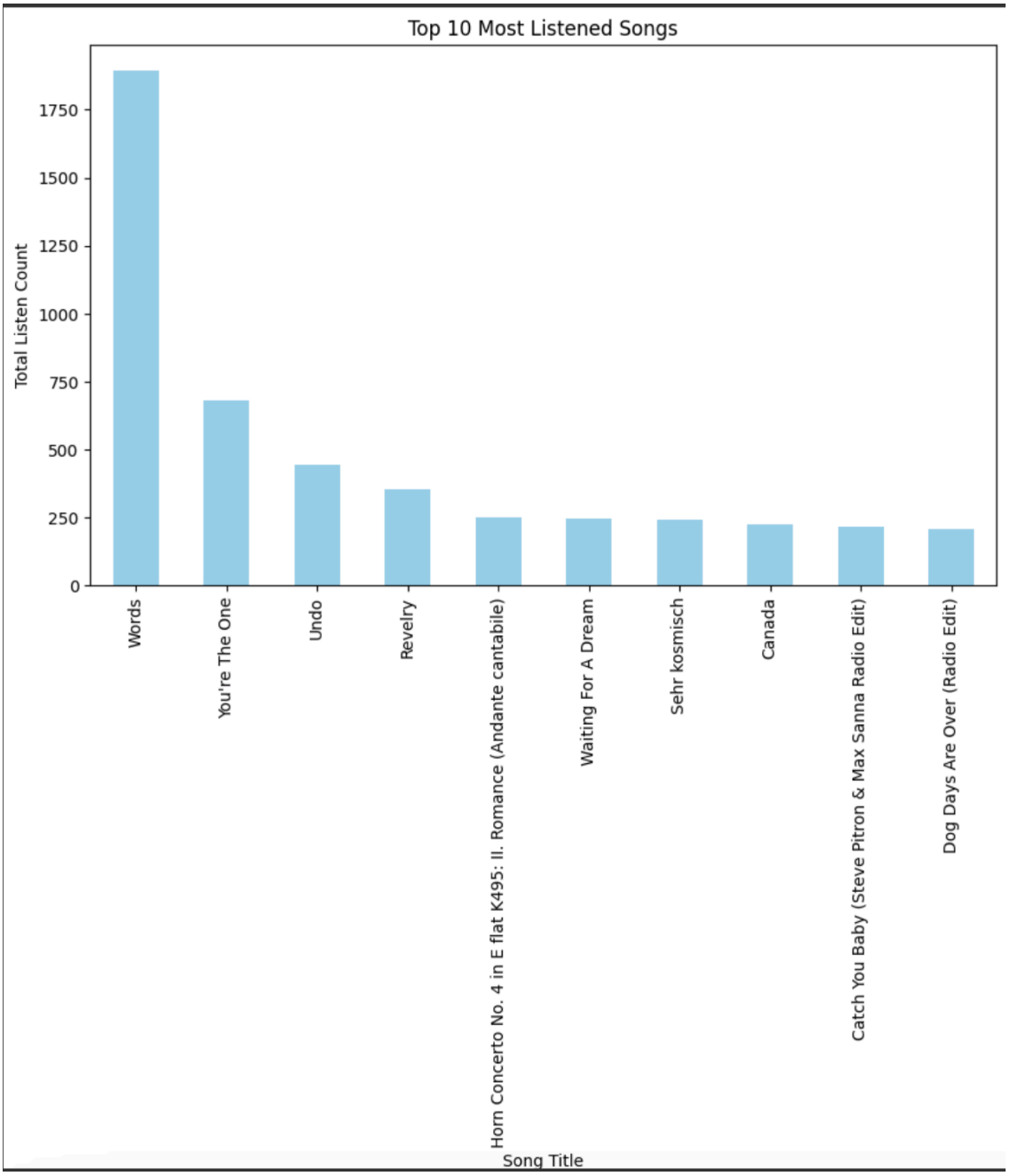
Objective:

The goal of this analysis is to identify and visualize the most listened songs in the dataset.

Methodology:

- Grouping the dataset by song title and calculating the total listen count for each song.
- Sorting the songs in descending order based on the total listen count.
- Visualizing the top 10 most listened songs using a bar chart.

Result



- The chart provides a clear view of the most popular songs in the dataset.
- Users seem to have a higher preference for "words","you're the one" and "Undo", as indicated by their higher total listen counts.

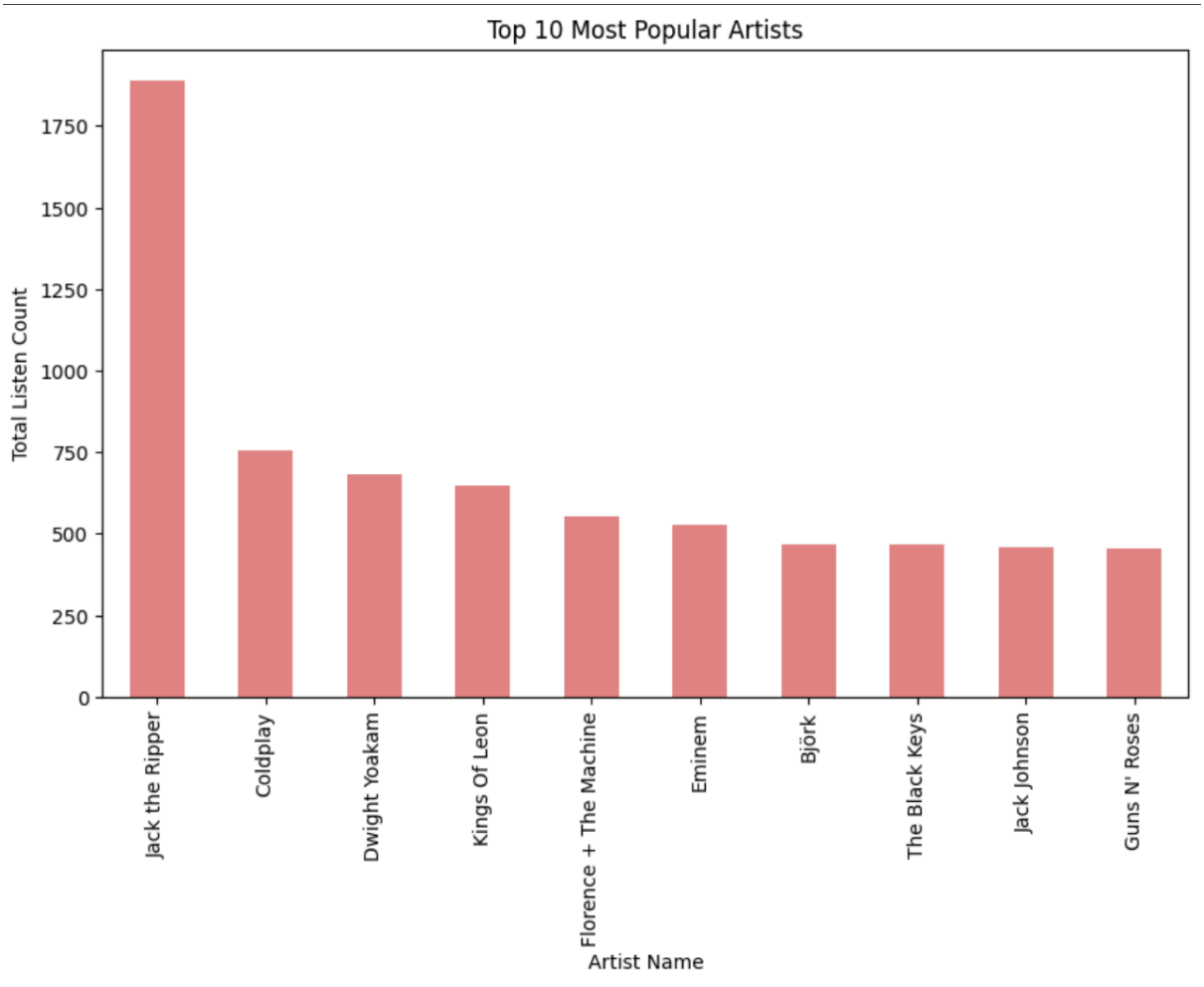
Most Popular Artist

Objective:

The objective of this analysis is to identify and visualize the most popular artists based on the total listen count in the dataset.

Methodology:

- Grouping the dataset by artist name and calculating the total listen count for each artist.
- Sorting the artists in descending order based on the total listen count.
- Visualizing the top 10 most popular artists using a bar chart.



Result:

- The chart provides insights into the artists with the highest total listen counts, indicating their popularity among users.
- Artists with a larger total listen count are “Jack The Ripper”, “Coldplay” and “Dwight yoakam”.

Distribution of Song Count per User

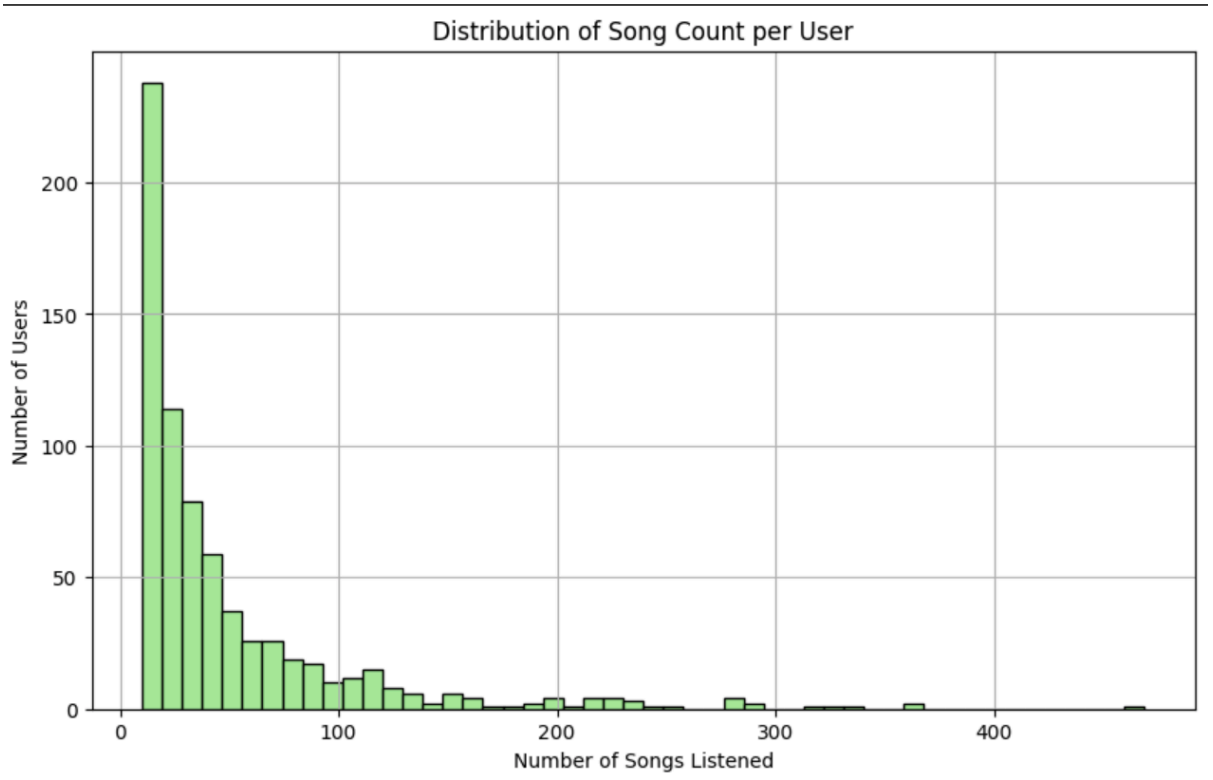
Objective:

The objective of this analysis is to examine the distribution of song count per user, providing insights into user engagement and activity.

Methodology

- Grouping the dataset by user and calculating the count of unique songs listened to by each user.
- Visualizing the distribution of song counts per user using a histogram.

Result



- The majority of users seem to have listened to a relatively small number of unique songs.
- There may be a long tail, indicating a smaller number of users who have listened to a large variety of songs.

Recommendation Model

Objective

The objective of this implementation is to create a recommendation engine for song recommendations based on collaborative filtering using the Surprise library.

Approach:

- Collaborative filtering, specifically the K-Nearest Neighbors (KNN) algorithm, is employed to recommend songs.
- The Surprise library is used for its easy implementation of collaborative filtering.

Explanation:

- The Surprise library simplifies collaborative filtering implementation by providing pre-built algorithms like KNNBasic.
- The dataset is loaded into the Surprise framework using the Reader and Dataset classes. The data is split into training and testing sets.
- The KNNBasic model is built using collaborative filtering, where similarity between items (songs) is considered.
- The get_recommendations function takes a user ID and returns personalized song recommendations for that user.

KNNBasic Model and Collaborative Filtering:

K-Nearest Neighbors (KNN):

- KNN is a machine learning algorithm used for both classification and regression tasks. In the context of collaborative filtering for recommendations, it operates by finding the "nearest neighbors" based on user-item interactions.
- The algorithm considers the similarity between users or items to make predictions for missing values, such as user ratings for items.

Collaborative Filtering:

- Collaborative filtering is a recommendation technique that relies on user-item interactions and user preferences to make personalized recommendations.
- It can be broadly categorized into two types:
    1. User-Based Collaborative Filtering: Recommends items based on the preferences of users who are similar to the target user.
    2. Item-Based Collaborative Filtering: Recommends items similar to those the target user has liked or interacted with.

KNNBasic in Surprise Library:

- The Surprise library provides an implementation of collaborative filtering using KNN through the KNNBasic class.
- The KNNBasic model in Surprise is user-based collaborative filtering, where the similarity between users is considered to make predictions for items.
- The term "Basic" in KNNBasic refers to the simplicity of the algorithm, which calculates similarities based on a basic mean or weighted mean approach.

Building the Model:

In the provided code, the following steps are taken to build the KNNBasic collaborative filtering model:

- Load the dataset using the Surprise library's Dataset class.
- Split the data into training and testing sets using train_test_split.
- Create an instance of the KNNBasic model, specifying user_based=False to indicate item-based collaborative filtering.
- Fit the model using the training set.

Prediction and Recommendations:

- Once the model is trained, the get_recommendations function is defined to generate recommendations for a specific user.
- The function identifies songs the user has not listened to (user_unseen_songs) and uses the trained model to predict ratings for these songs.
- Recommendations are then sorted based on the predicted ratings, and the top recommendations are returned.

Conclusion:

The KNNBasic collaborative filtering model in the Surprise library is a user-based collaborative filtering approach used for making personalized song recommendations based on user interactions with songs in the dataset.

Web Application and Deployment

The source code for the application can found at
https://github.com/loiccdk/recommendation_ml

The application can be runned at https://recommendations-ml.onrender.com

Web Application

Technology

To create a web application using our model, we used the framework Flask (https://flask.palletsprojects.com/en/3.0.x/). This framework is used to build small web applications using Python language.

How to use it

The application itself is pretty simple, and consists of two pages. The starting page contains a list of the songs in the dataset, and allows the user to select the songs he has listened to. When the user submits its list of songs, a result page is shown, containing a list of recommended songs.

How it works

The application is composed of one Python file, the CSV dataset file, and a directory of HTML templates files.

We mostly worked on the backend of the application, and we did not do any CSS code. The frontend could be improved a lot, but we judged that it was not the purpose of this course. Therefore, the application is very minimalist and not very appealing.

The HTML templates are HTML files in which it is possible to use loops and variables. For example, one can use a For loop to add one line in the HTML file for each cell in an array. From a Python file, it is possible to call those templates (using the function *render_template()*), giving variables as arguments.
In our applications, we have 2 templates. One creates a drop-down selection element from a list given as an argument. The second displays the elements of a list in order on the HTML page. At the start, the list of songs is retrieved from the dataset, and put in alphabetical

order. The first template is then called with this list to create the drop-down selection list. When the user selects some songs and submits the list, the recommendations are computed, and the second template is called with the resulting list to display the results.

In order to compute the recommendations, we used the same model as before. In order to do it, we had to address a major issue : The model as we built it takes as input a user (already existing in the dataset), and outputs recommendations for this specific user, based on the songs they listened to. But in the application, we want a new user to be able to select the songs he has listened to, and then get a recommendation.
To address the problem, we had the idea of creating a temporary new user in the dataset. When the user of the application selects the list of songs they have listened to, we add those songs to the dataset. More precisely, we create a new user in the dataset, who has listened to the songs selected by the user of the application. We can then use our model to get recommendations for this new user.
The data are only added temporarily to the dataset in order to use the model, and are not stored to train the model.

Deployment

Technology

To deploy our application, we used Render (https://render.com/), a cloud service to build and run apps. We used Render because it can be used for free. However, the web service we can use for free is the smallest one (0.15 CPU, 512 MB).
The web service can be connected to a Github instance, and the application is automatically redeployed at every new commit.

How it works

For the application to work, some libraries need to be installed (pandas, scikit…). The web service can be configured so that the libraries are installed automatically before each new deployment. On the Github repository, a file ‘requirements.txt’ lists the libraries required. With a command line in the web service settings, all the libraries in the file are installed before the deployment.

Drawbacks

As explained previously, as we have used a free web service, we only had access to a small one. So in order to be able to fit the size and computation power of the web service, we had to considerably reduce the size of the dataset. In the application, we only use 2.5% of the dataset. That is why one may notice some missing songs in our applications.

When testing the application, we noticed that the recommendation list was the same every time. We first thought it was due to the fact that we temporarily created new users just to compute the recommendations, whereas the model was trained without this user existing. So we tried to re-train the model every time a list of songs was provided by the user. This did not change the results, and we still got the same recommendations every time.
We do not know if the problem comes from the implementation of the model in the application, or if the problem comes from the model itself.