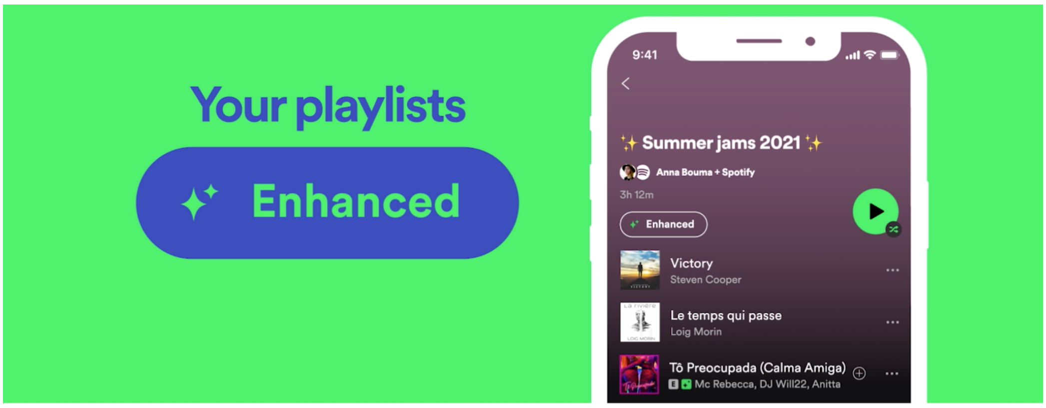
SONG RECOMMENDER

MARIA KOILALOU
JUSTINE LE BOURGE
GROUP 11

Project for Artificial Intelligence [EARIN]
Summer 2023



Summary

PROBLEM DEFINITION

Our project aims to develop a sophisticated Song Recommender System using Spotify datasets. The goal is to create a model that can provide song recommendations to users based on the song's characteristics. By analyzing the information provided by the datasets, we will identify the key attributes that determine whether a user will like a song or not.

In this project we are going to develop a program that uses some Spotify datasets. Spotify is a song streaming app and platform that most people use and almost every artist uploads their songs there. We will start by exploring the Spotify datasets, which contain extensive information on songs, artists, and song characteristics, including popularity data. These datasets will serve as a valuable resource for training our models.

First in our project we are going to examine the datasets. The songs, the artists and the characteristics of the songs and the relationship between them.
Based on the target attribute we are going to determine if a song is “good” or “bad” for the user. Through our analysis, we will identify the most important attributes that contribute to a song's appeal to a user. By focusing on these attributes, we will build a robust model that can accurately predict user preferences and provide song recommendations.

In our project we are going to use both Python Notebooks for a quicker way to examine data and run code, a complete Python script that is going to run the whole project and a report explaining exactly how the script is working.

DATASET

a. Overview

First we are going to examine the whole dataset and create some graphs and tables with helpful information about the data that we are going to use later.

We import the libraries that we are going to need and read the dataset:

```
# Import necessary libraries

import pandas as pd
import seaborn as sns
import warnings
import plotly.graph_objs as go
import matplotlib.pyplot as plt
warnings.simplefilter("ignore")

# Read Dataset
data = pd.read_csv("songs.csv")
```

We notice that the first column of the Dataset is empty so we fix it not to be unnamed. Then we print the head of the dataset to check the attributes and their values:

```
# Drop the first Column
data.drop('Unnamed: 0', axis=1, inplace=True)

# Print the Start of the Dataset to examine the datas
print("Dataset:", data.head())
```

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	target	song_title	art
0	0.0102	0.833	204600	0.434	0.021900	2	0.1650	-8.795	1	0.4310	150.062	4.0	0.286	1	Mask Off	Fut
1	0.1990	0.743	326933	0.359	0.006110	1	0.1370	-10.401	1	0.0794	160.083	4.0	0.588	1	Redbone	Chi Ga
2	0.0344	0.838	185707	0.412	0.000234	2	0.1590	-7.148	1	0.2890	75.044	4.0	0.173	1	Xanny Family	Fut
3	0.6040	0.494	199413	0.338	0.510000	5	0.0922	-15.236	1	0.0261	86.468	4.0	0.230	1	Master Of None	Be Ho
4	0.1800	0.678	392893	0.561	0.512000	5	0.4390	-11.648	0	0.0694	174.004	4.0	0.904	1	Parallel Lines	Jur Bo

Then we check for null or duplicate values:

```
# Looking for missing values in the dataset
print("Null Values:", data.isna().sum())
```

```
# Drop Duplicate Values
data = data.drop_duplicates().reset_index(drop=True)
```

```
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
key                 0
liveness            0
loudness            0
mode                0
speechiness         0
tempo               0
time_signature      0
valence             0
target              0
song_title          0
artist              0
dtype: int64
```

We print some general information about the dataset:

```
# Print General Information about the dataset
print("Information about Dataset:", data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2017 entries, 0 to 2016
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   acousticness      2017 non-null   float64
 1   danceability      2017 non-null   float64
 2   duration_ms       2017 non-null   int64
 3   energy            2017 non-null   float64
 4   instrumentalness  2017 non-null   float64
 5   key               2017 non-null   int64
 6   liveness          2017 non-null   float64
 7   loudness          2017 non-null   float64
 8   mode              2017 non-null   int64
 9   speechiness       2017 non-null   float64
 10  tempo             2017 non-null   float64
 11  time_signature    2017 non-null   float64
 12  valence           2017 non-null   float64
 13  target            2017 non-null   int64
 14  song_title        2017 non-null   object
 15  artist            2017 non-null   object
dtypes: float64(10), int64(4), object(2)
memory usage: 252.2+ KB
Information about Dataset: None
```

And some basic information about the attribute's values such as total count, mean, minimum and maximum value:

```
# Description of Attributes
print("Attributes Brief Description:", data.describe())
```

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo
count	2017.000000	2017.000000	2.017000e+03	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000
mean	0.187590	0.618422	2.463062e+05	0.681577	0.133286	5.342588	0.190844	-7.085624	0.612295	0.092664	121.603272
std	0.259989	0.161029	8.198181e+04	0.210273	0.273162	3.648240	0.155453	3.761684	0.487347	0.089931	26.685604
min	0.000003	0.122000	1.604200e+04	0.014800	0.000000	0.000000	0.018800	-33.097000	0.000000	0.023100	47.859000
25%	0.009630	0.514000	2.000150e+05	0.563000	0.000000	2.000000	0.092300	-8.394000	0.000000	0.037500	100.189000
50%	0.063300	0.631000	2.292610e+05	0.715000	0.000076	6.000000	0.127000	-6.248000	1.000000	0.054900	121.427000
75%	0.265000	0.738000	2.703330e+05	0.846000	0.054000	9.000000	0.247000	-4.746000	1.000000	0.108000	137.849000
max	0.995000	0.984000	1.004627e+06	0.998000	0.976000	11.000000	0.969000	-0.307000	1.000000	0.816000	219.331000

Print the size of the dataset:

```
# Size of dataset
print("Size of Dataset:", data.shape)
```

```
(2017, 16)
```

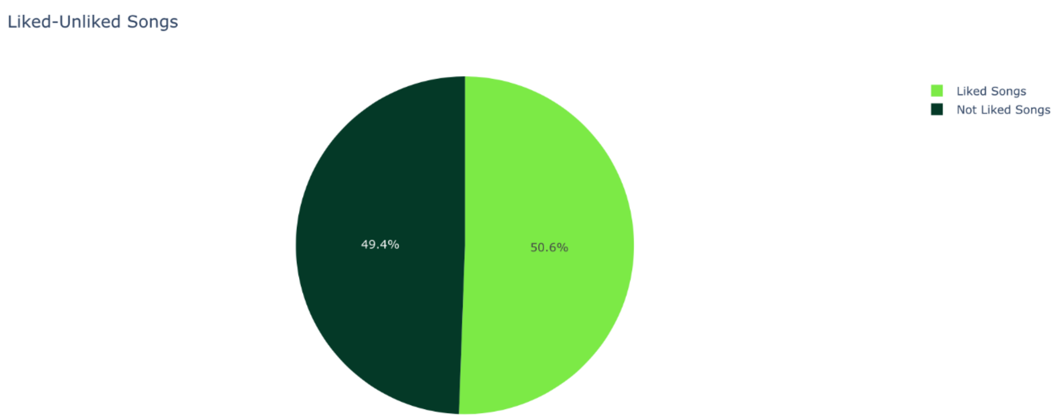
After that we are going to pay closer attention to the target attribute by checking its different values.

```
# Examine the different values of target
print("Variaty of target Values:", data.target.value_counts())
```

```
1    1020
0     997
Name: target, dtype: int64
```

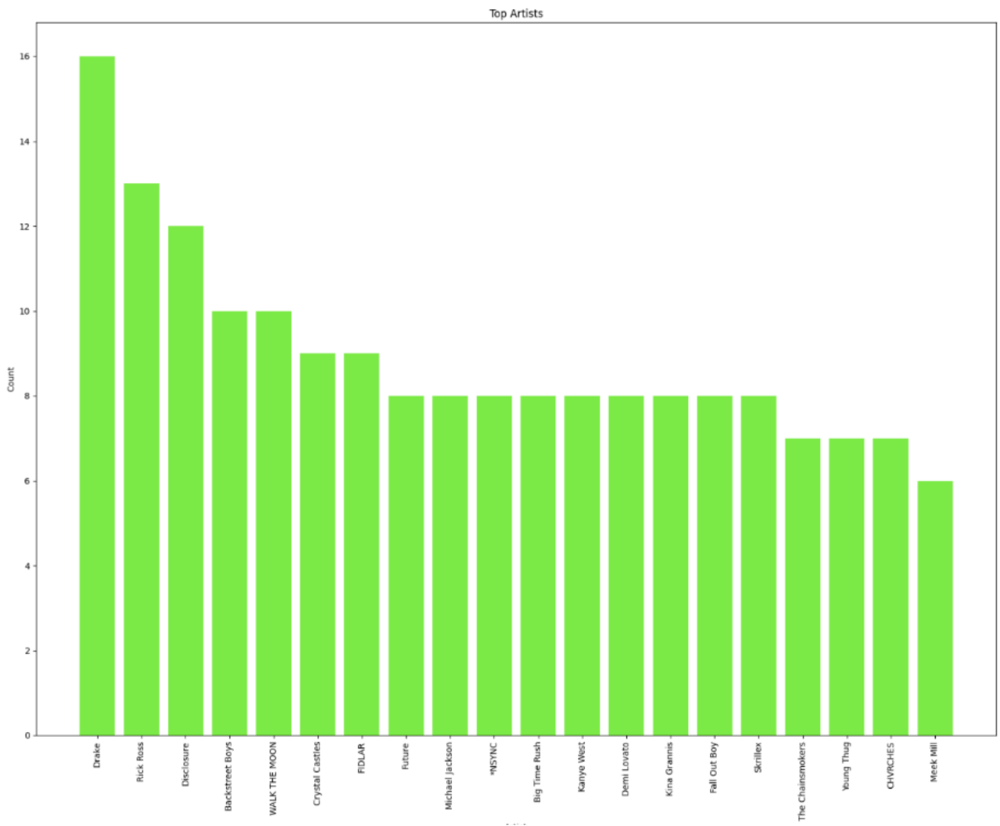
In our dataset “1” means that the song is liked and “0” means that is unliked so we visualize these results in a pie diagram
(The pie diagram in our code opens a browser tab because at first the code was written in a Python Notebook and we weren’t able to find an alternative with the same result)

```
# Create the pie chart
fig = go.Figure(data=[go.Pie(values=values, labels=labels,
title="Liked-Unliked Songs",
                            marker=dict(colors=["#7CEA46", "#043927"]))])

fig.show()
```



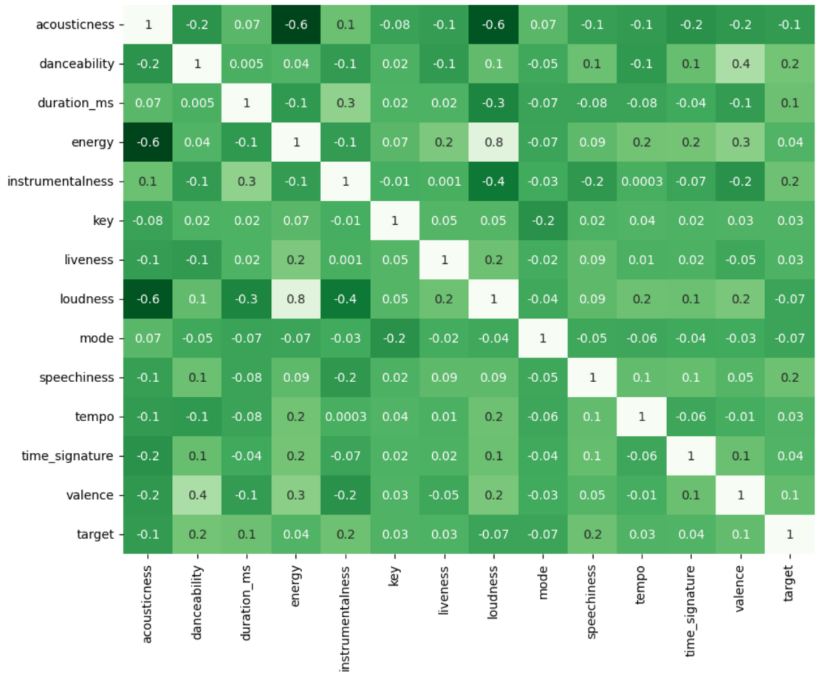
We continue our examination by checking the different artists in the dataset:

```
# Set the title and axis labels
values = data['artist'].value_counts().tolist()[:20]
names = list(dict(data['artist'].value_counts()).keys())[:20]

# Create the bar plot
fig, ax = plt.subplots(figsize=(20, 15))
ax.bar(names, values, color="#7CEA46")

# Set the title and axis labels
ax.set_title("Top Artists")
ax.set_xlabel("Artist")
ax.set_ylabel("Count")

# Display the plot
plt.xticks(rotation=90)
plt.show()

plt.show()
```



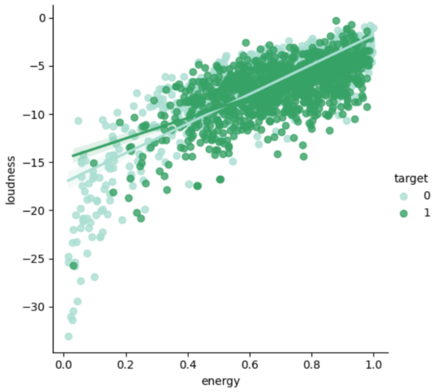
Plot the Correlation Matrix of the dataset to find the correlation between the attributes and most importantly which ones have which are the ones that determine the target value. The ones with the higher correlation with target are the ones that we are going to use to train our models on:

```
# Plot linear correlation matrix
numeric_data = data.drop(['song_title', 'artist'], axis=1)
fig, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(numeric_data.corr(), annot=True, fmt='.1g', cmap="Greens_r",
cbar=False)
plt.title('Linear Correlation Matrix')
plt.show()
```



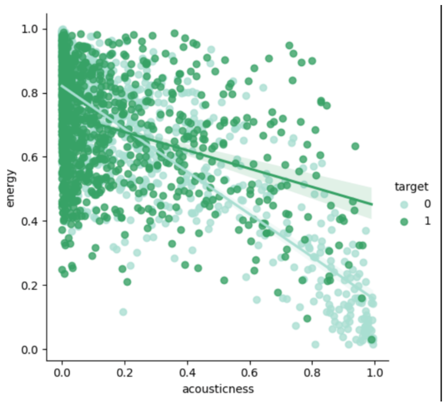
We use heatmap to review the correlation more clearly. We notice that loudness and energy have a really high correlation so we use scatter chart and trend line to view their relationship by different targets.

```
# Scatter chart for "loudness" and "energy"
sns.lmplot(y='loudness', x='energy', data=data, hue='target', palette='BuGn')
```



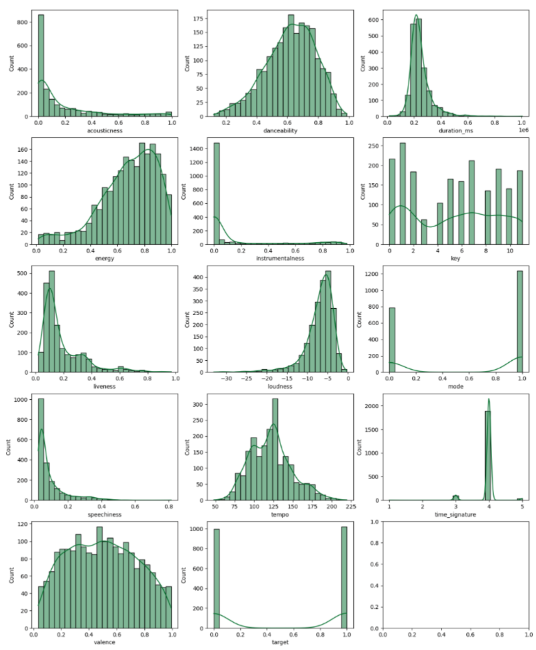
We notice that acousticness and energy have a really high correlation so we use scatter chart and trend line to view their relationship by different targets.

```
# Scatter chart for "acousticness" and "energy"
sns.lmplot(y='energy', x='acousticness', data=data, hue='target',
palette='BuGn')
```



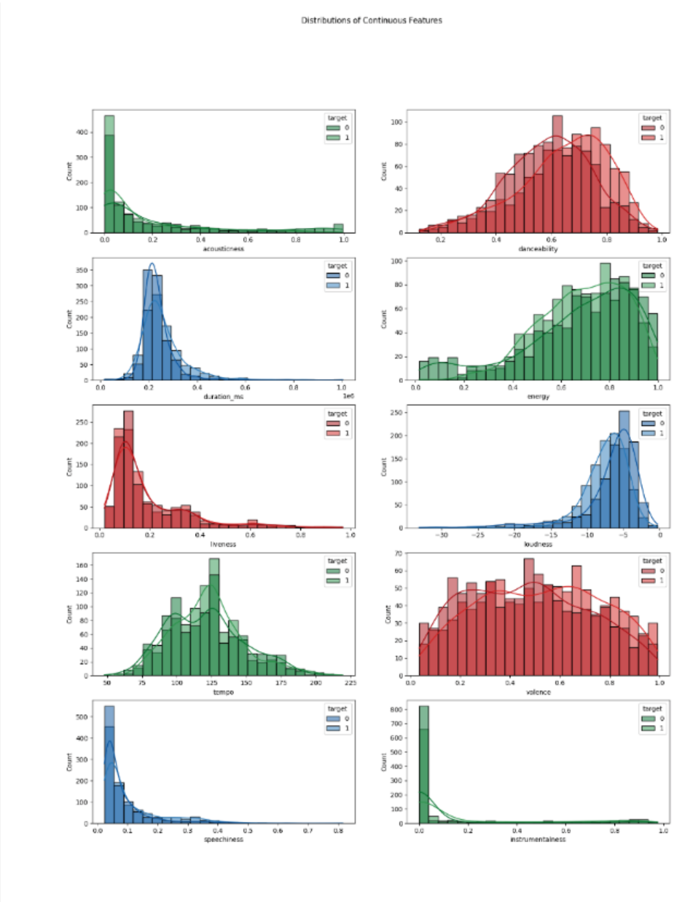
We then print the Histograms of all the attributes to examine all the attributes:

```
# Examine Histograms
sns.set_palette("Greens_r")
num_cols = data.select_dtypes(include="number").columns
fig, axes = plt.subplots(5, 3, figsize=(16, 20))
axes = axes.flatten()
ax_no = 0
for col in num_cols:
   sns.histplot(data=data, x=col, bins=25, kde=True, ax=axes[ax_no])
   ax_no += 1
plt.show()
```



We examine the continuous features based on the target's values :

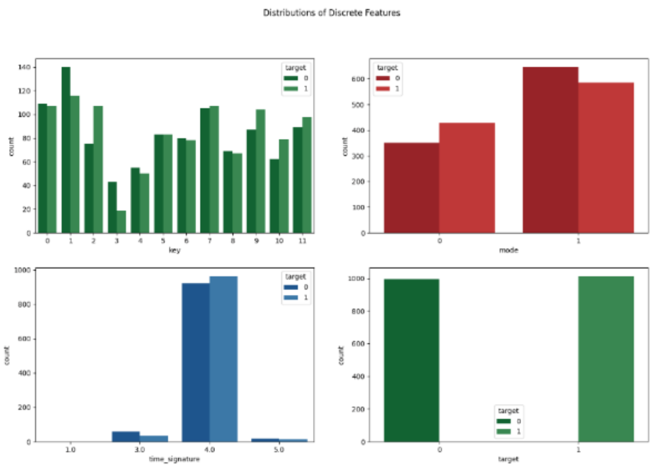
```
# Examine Continuous Data
fig, axes = plt.subplots(5, 2, figsize=(16, 20))
palettes = ['Greens_r', "Reds_r", "Blues_r"]
axes = axes.flatten()
ax_no = 0
for col in continuous_cols:
   sns.set_palette(palettes[ax_no % 3])
   sns.histplot(data=data, x=col, hue='target', bins=25, kde=True,
ax=axes[ax_no])
   ax_no += 1
fig.suptitle('Distributions of Continuous Features')
plt.show()
```



We examine the descrete features based on the target's values :

```
# Examine Descrete Data
fig, axes = plt.subplots(2, 2, figsize=(16, 10))
palettes = ['Greens_r', "Reds_r", "Blues_r"]
axes = axes.flatten()
ax_no = 0
for col in discrete_cols:
```

```
    sns.set_palette(palettes[ax_no % 3])
    sns.countplot(data=data, x=col, ax=axes[ax_no], hue='target')
    ax_no += 1
fig.suptitle('Distributions of Discrete Features')
plt.show()
```



b. Pre-processing

<u>Data preparation</u>: Collecting, cleaning, and preparing the dataset.

c. Post-processing

TECHNICAL APPROACH

a. Architecture

Data splitting: Dividing the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set is used to tune the hyperparameters of the model, and the testing set is used to evaluate the final performance of the model.

b. Training Details

Model training: Training the AI model on the training set. The goal is to optimize the model's parameters to minimize the loss function and improve the model's performance on the training set.

c. Evaluation Details

Model validation: Evaluating the performance of the trained model on the validation set. The goal is to fine-tune the model's hyperparameters to improve its performance on the validation set.

RESULTS

CONCLUSION