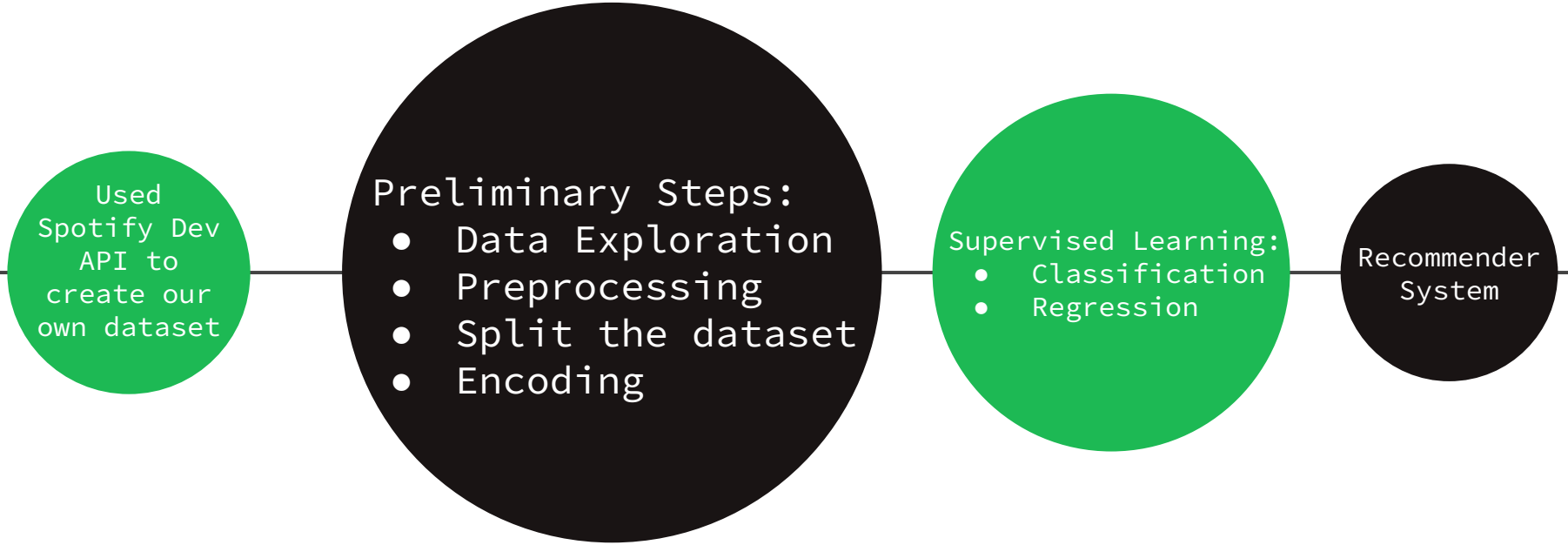
Spotify Tracks

Project of Big Data Computing
Sapienza University of Rome
Department of Computer Science

Professor: Gabriele Tolomei

Students:

- Andrea Bernini - 2021867
- Donato Francesco Pio Stanco - 2027523



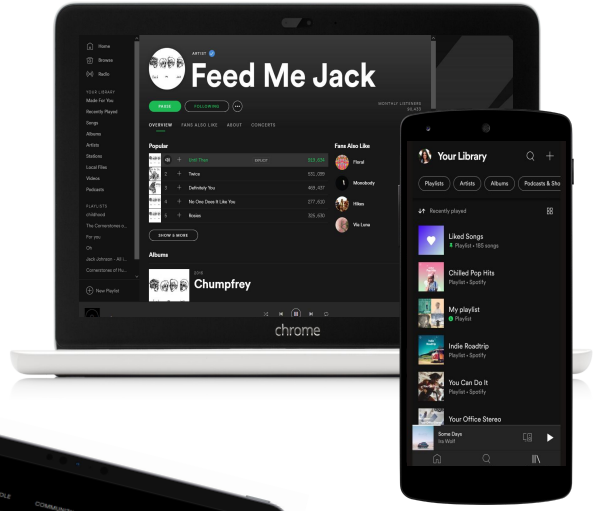
Project Overview

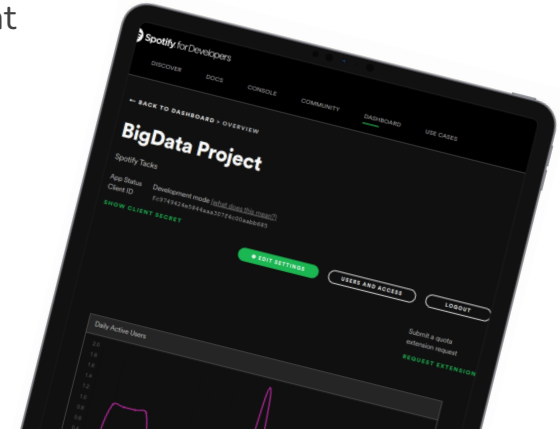
Data Acquisition: Spotify

Today Spotify is one of the most popular app for listening music (online & offline).

It counts around 1 billion of installations (reached in 2021 on Play Store).

In order to create our dataset we used the **Spotify Dev API** and a Python Program that makes request to the Spotify Server.





Data Acquisition

- For data acquisition first we look for a dataset that has the genre associated with the song.
- After several searches we have not found an adequate dataset.

The Reason:

- Spotify does not associate the genre directly with the track but exclusively with the artists;
- An artist in most cases has multiple genres.

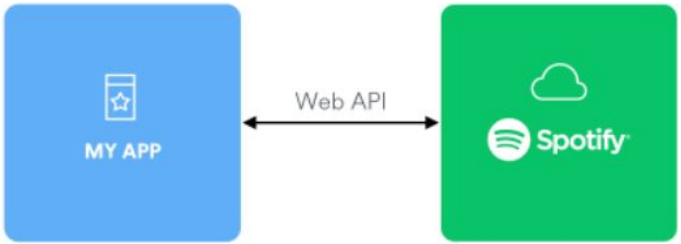
Therefore we decided to create a dataset from **scratch** using the **Spotify Web API**.

Data Acquisition

The Spotify Web API provides **queries** to get:

- the list of all genres classified by Spotify;
- a list of up to 100 recommended songs given a specific genre;
- the audio characteristics of a track given its Spotify ID;
- the playlists of the user given his Spotify ID.

We used these to create the two datasets



Description of the Dataset

Before the pre processing operations our dataset is composed by:

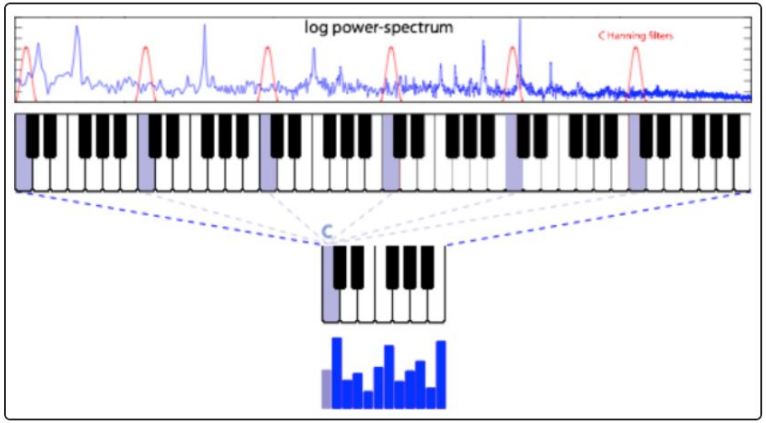
- around **80k rows;**
- each row is described by **25 columns (features).**

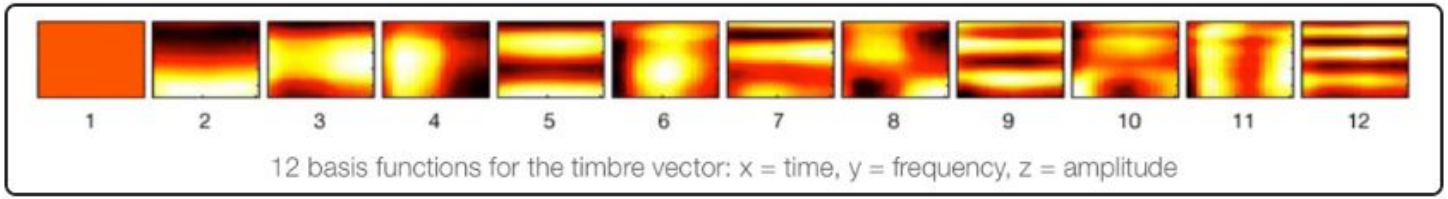
The preprocessing operations we did to the dataset led these numbers to grow significantly, reaching **100k rows** and each row is described by **47 features.**

```
root
 |-- id: string (nullable = true)
 |-- track_name: string (nullable = true)
 |-- track_explicit: boolean (nullable = true)
 |-- track_popularity: integer (nullable = true)
 |-- album_name: string (nullable = true)
 |-- album_release_date: string (nullable = true)
 |-- album_release_date_precision: string (nullable = true)
 |-- artist_name: string (nullable = true)
 |-- audio_avg_pitches: string (nullable = true)
 |-- audio_avg_timbre: string (nullable = true)
 |-- audio_acousticness: double (nullable = true)
 |-- audio_danceability: double (nullable = true)
 |-- audio_duration_ms: integer (nullable = true)
 |-- audio_energy: double (nullable = true)
 |-- audio_instrumentalness: double (nullable = true)
 |-- audio_key_1: integer (nullable = true)
 |-- audio_liveness: double (nullable = true)
 |-- audio_loudness: double (nullable = true)
 |-- audio_mode_1: integer (nullable = true)
 |-- audio_speechiness: double (nullable = true)
 |-- audio_tempo: double (nullable = true)
 |-- audio_time_signature: integer (nullable = true)
 |-- audio_valence: double (nullable = true)
 |-- track_uri: string (nullable = true)
 |-- track_genre: string (nullable = true)
```

Feature Variety

- Combined Spotify audio features and audio analysis (e.g. *audio_acousticness*, *audio_danceability*, *audio_energy*, etc.).
- Averaged pitch and timbre vectors over all song segments.
- Included miscellaneous metrics (popularity, date of release, etc.).

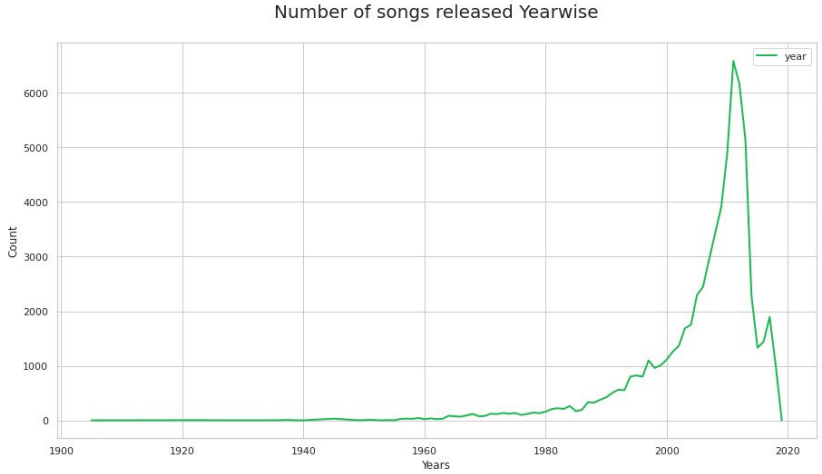


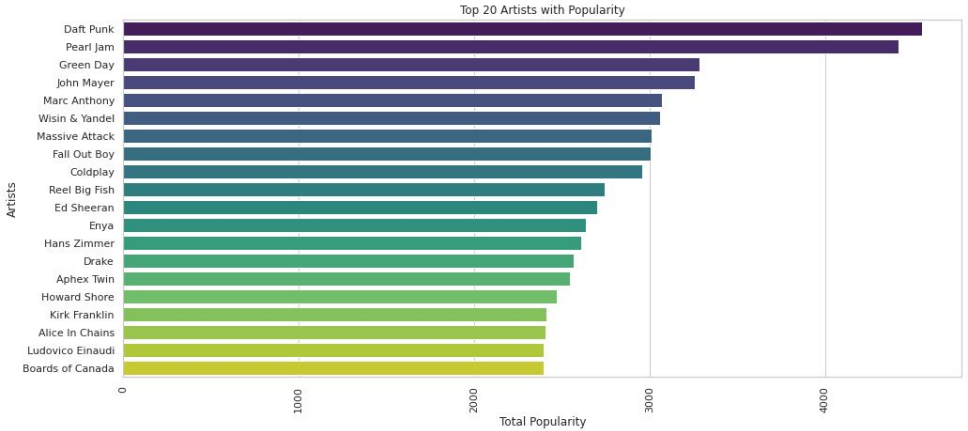


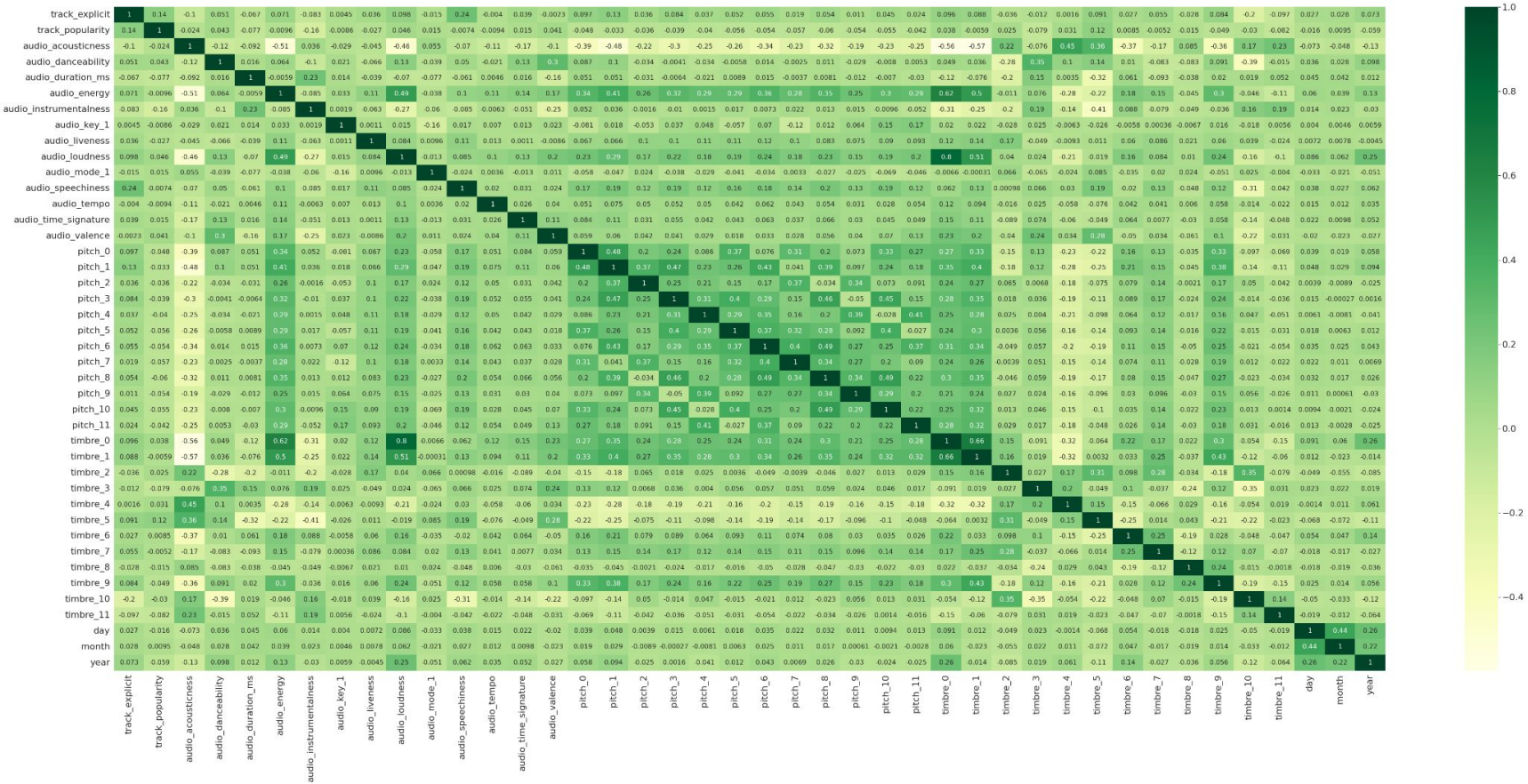
Data Exploration

In our project we decided to “explore” our dataset and we have shown some graphs including:

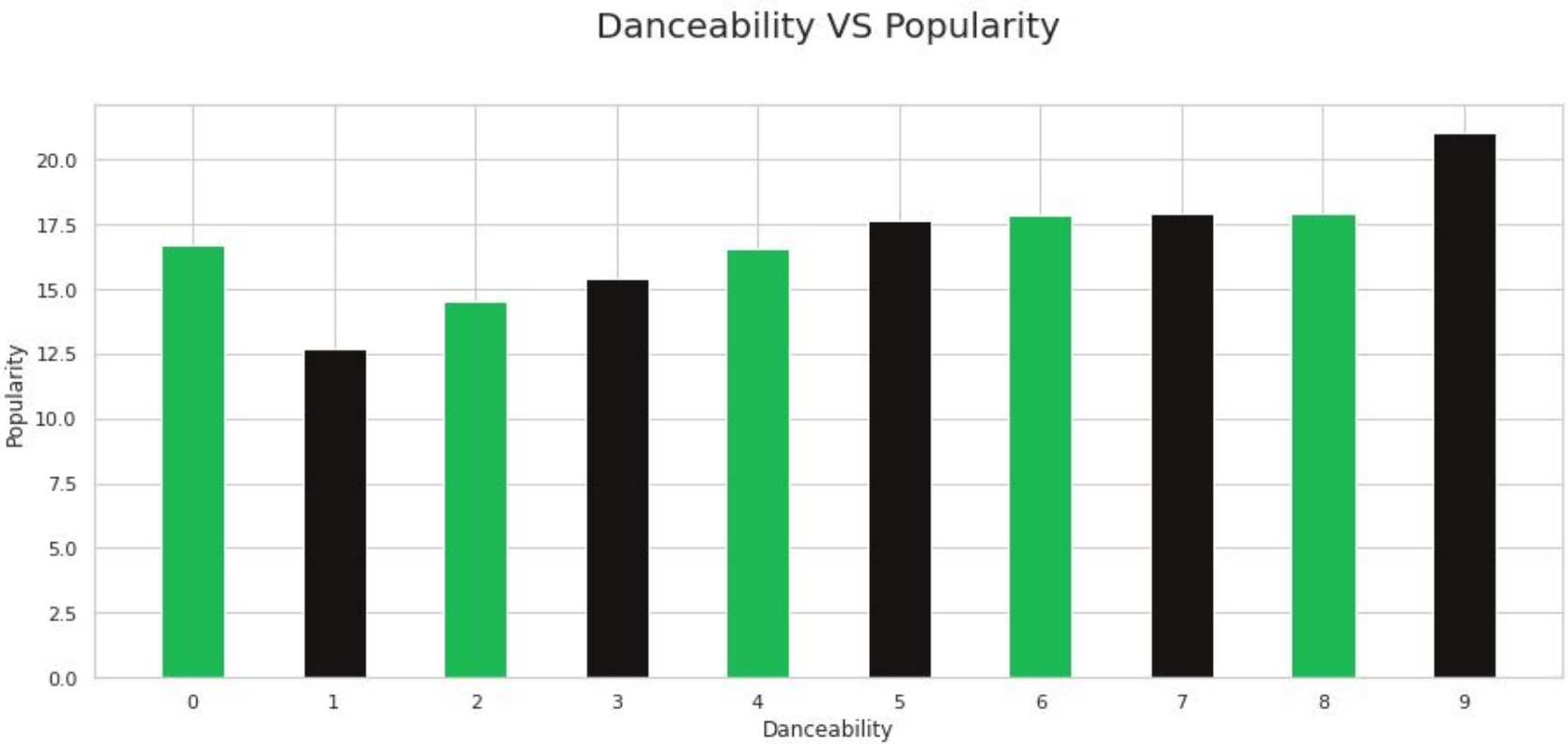
- Visualization of variability of some features
- Correlation Map
- Most Popular Tracks
- Most Popular Artists
- Most present Artist
- Number of songs released year wise
- Year VS **Popularity**
- Danceability vs **Popularity**
- Key vs **Popularity**
- Key vs **Genre**



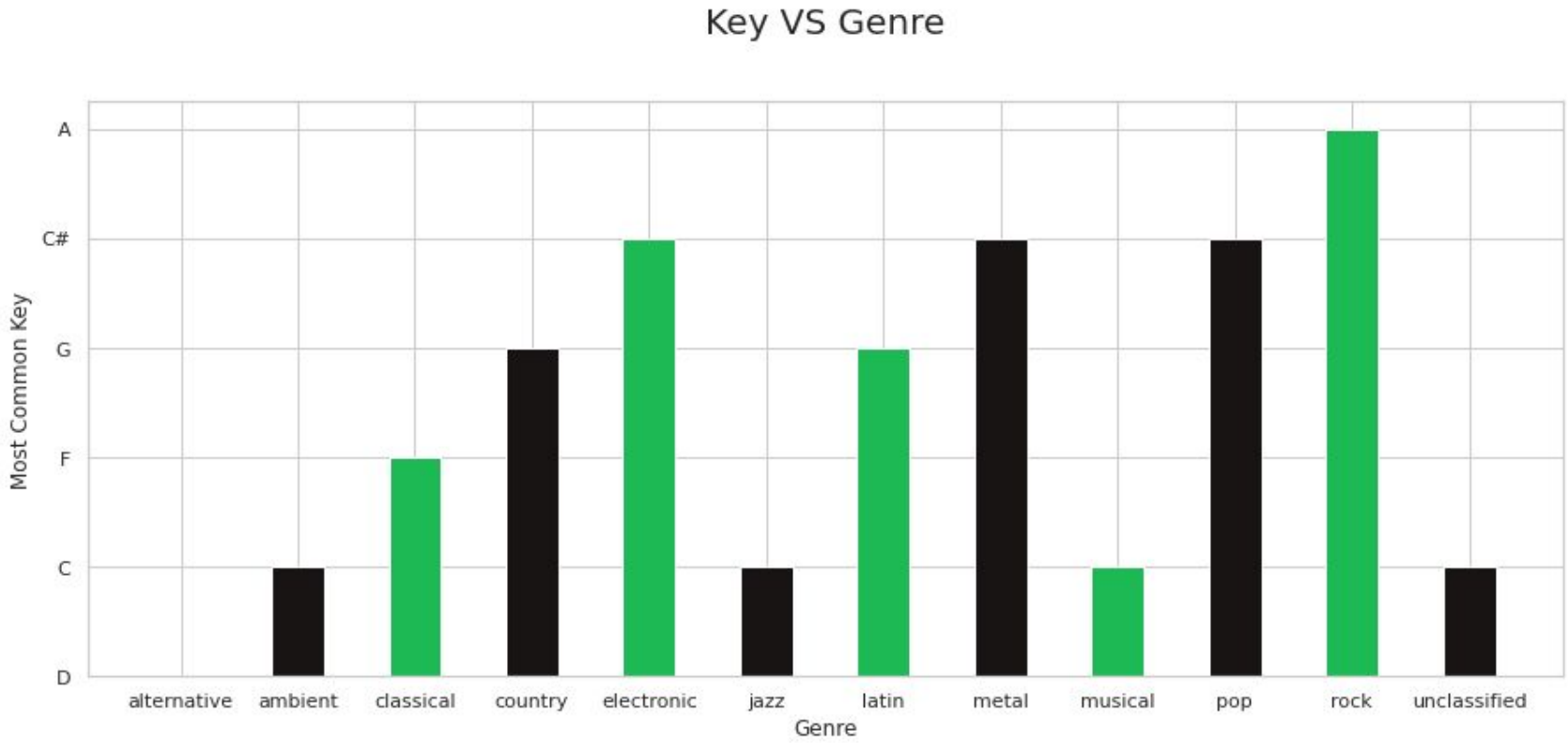




Data Exploration - Correlation Map



Data Exploration



Data Exploration

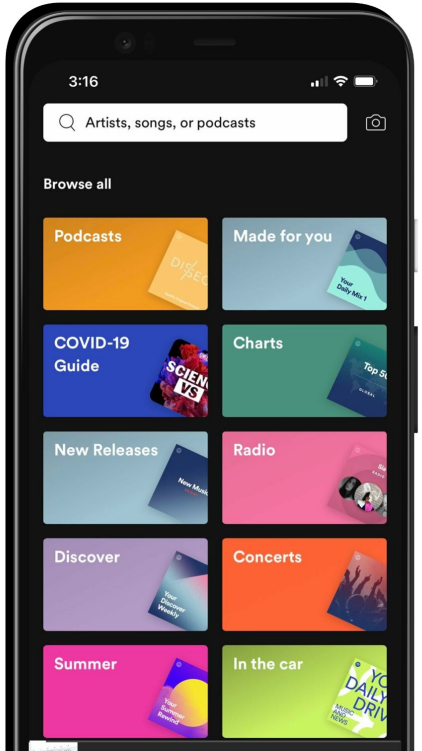
Discretization: Spotify Genre

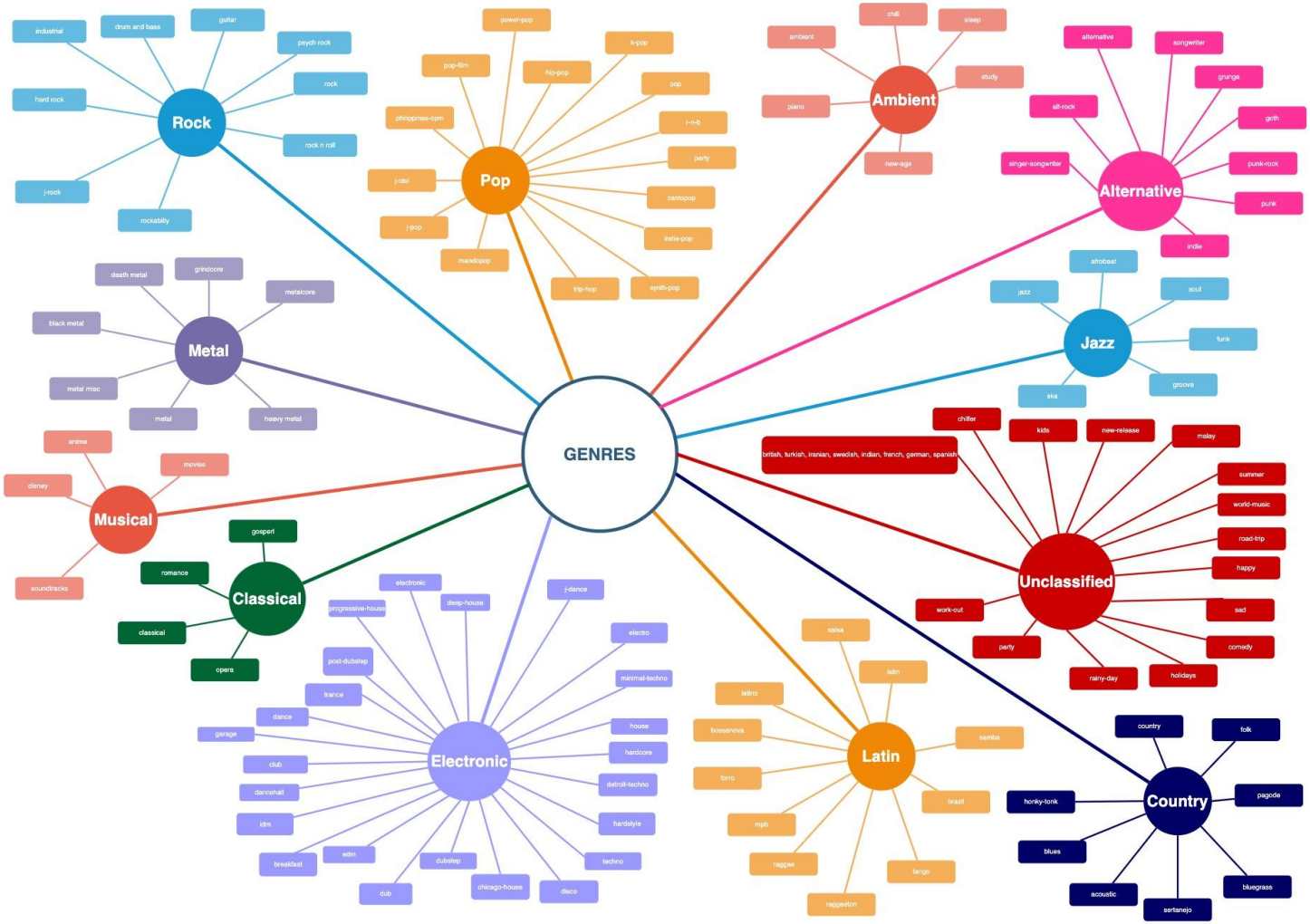
According to Spotify there are **126 genres** for the tracks, so we decided to group them into **12 “big-genres”** into a json file to simplify the classification.

In this way we did a mapping for the genre by looking at some documentation on the web to see if a particular genre was part of a "main" genre (eg j-rock is part of the rock genre category).

Spotify

Discovery




GENRES
Rock
industrial
drum and bass
guitar
psych rock
rock
rock n roll
rockabilly
j-rock
hard rock
Pop
power-pop
k-pop
hip-pop
pop
r-n-b
party
cantopop
indie-pop
synth-pop
trip-hop
mandopop
j-pop
j-idol
philippines-opm
pop-film
Ambient
chill
sleep
study
new-age
piano
ambient
Alternative
alternative
songwriter
grunge
goth
punk-rock
punk
indie
alt-rock
singer-songwriter
Jazz
afrobeat
soul
funk
groove
ska
jazz
Metal
death metal
grindcore
metalcore
heavy metal
metal
metal misc
black metal
Musical
anime
movies
disney
soundtracks
Classical
gospel
romance
classical
opera
Unclassified
chillier
kids
new-release
malay
summer
world-music
road-trip
happy
sad
comedy
holidays
rainy-day
party
work-out
british, turkish, iranian, swedish, indian, french, german, spanish
Electronic
electronic
progressive-house
deep-house
j-dance
electro
minimal-techno
house
hardcore
detroit-techno
hardstyle
techno
disco
chicago-house
dubstep
edm
breakfast
dub
idm
garage
dance
trance
post-dubstep
club
dancehall
Latin
salsa
latin
samba
brazil
tango
reggaeton
reggae
mpb
forro
bossanova
latino
Country
country
folk
pagode
bluegrass
sertanejo
acoustic
blues
honky-tonk

Discretization: Spotify Popularity

According to Spotify's Web API the popularity of a track is ranked with a number between **1** and **100.**

We decided to **discretize** our target in **10 classes (0-9)** and then we gave a label to each class according to this role:

- 0 == popularity: **unknown label**
- 0 < popularity <= 2: **low label**
- 2 < popularity <= 5: **medium label**
- 5 < popularity <= 9: **high label**

We use **Bucketizer** to "discretize" our target feature "track_popularity":

```
buck = Bucketizer(splits=splits,inputCol="track_popularity",outputCol="track_popularity_bucket")
spotify_tracks = buck.transform(spotify_tracks)
```

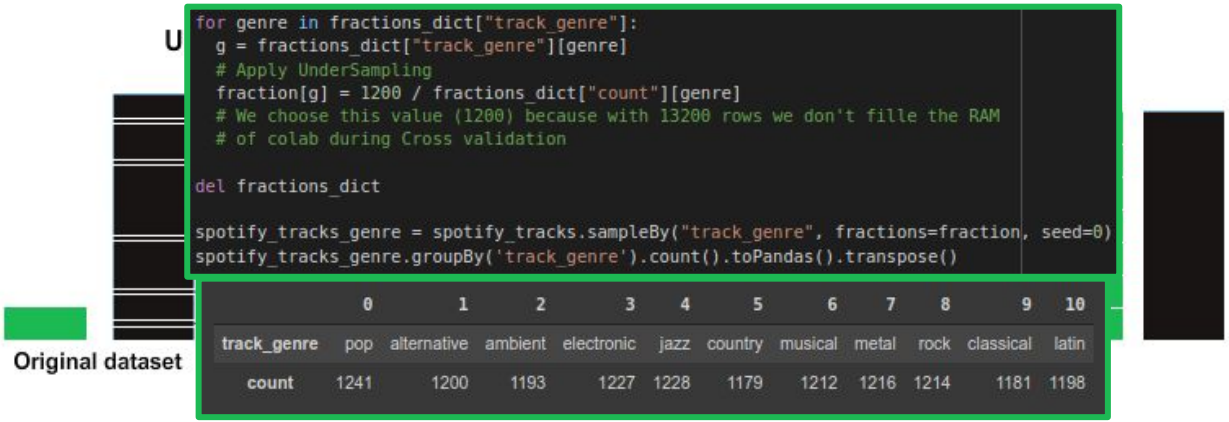
	0	1	2	3
track_popularity_bucket	low	unknown	high	medium
count	9903	31380	3046	11671

Balancement

In the project we decided to balance the dataset and create one for genre and one for popularity.

We decided to apply random sample to **undersampling** the datasets for two reasons:

- get around memory-related problems while running (during **cross validation**).
- many algorithms suffer if the classes are not balanced, if we have two classes, call it class A and class B if class A represents 90% of the dataset an algorithm can completely ignore the minority class (class B).




```
for genre in fractions_dict["track_genre"]:
  g = fractions_dict["track_genre"][genre]
  # Apply UnderSampling
  fraction[g] = 1200 / fractions_dict["count"][genre]
  # We choose this value (1200) because with 13200 rows we don't fille the RAM
  # of colab during Cross validation

del fractions_dict

spotify_tracks_genre = spotify_tracks.sampleBy("track_genre", fractions=fraction, seed=0)
spotify_tracks_genre.groupBy('track_genre').count().toPandas().transpose()
```

	0	1	2	3	4	5	6	7	8	9	10
track_genre	pop	alternative	ambient	electronic	jazz	country	musical	metal	rock	classical	latin
count	1241	1200	1193	1227	1228	1179	1212	1216	1214	1181	1198

Encoding Pipeline

This step is not always mandatory (e.g., decision trees are able to work nicely with categorical features without the need of transforming them to numerical). Still, other methods (like logistic regression) are designed to operate with numerical inputs only.

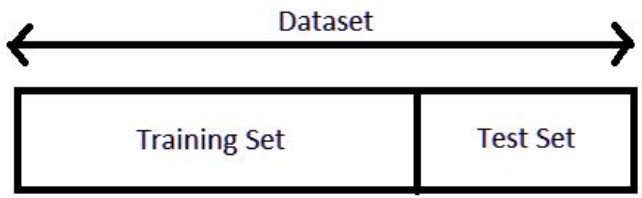
To transform *categorical features* into *numerical* ones we proceed as follows. We setup a pipeline which is composed of the following steps:

- **Label Encode target feature**
- **Label Encode categorical feature**
- **Vector Assembler**
- **Standard Scaler**

Split the dataset

We split our dataset into 2 portions:

- *training set* (e.g., accounting for **80%** of the total number of instances);
- *test set* (e.g., accounting for the remaining **20%** of instances)

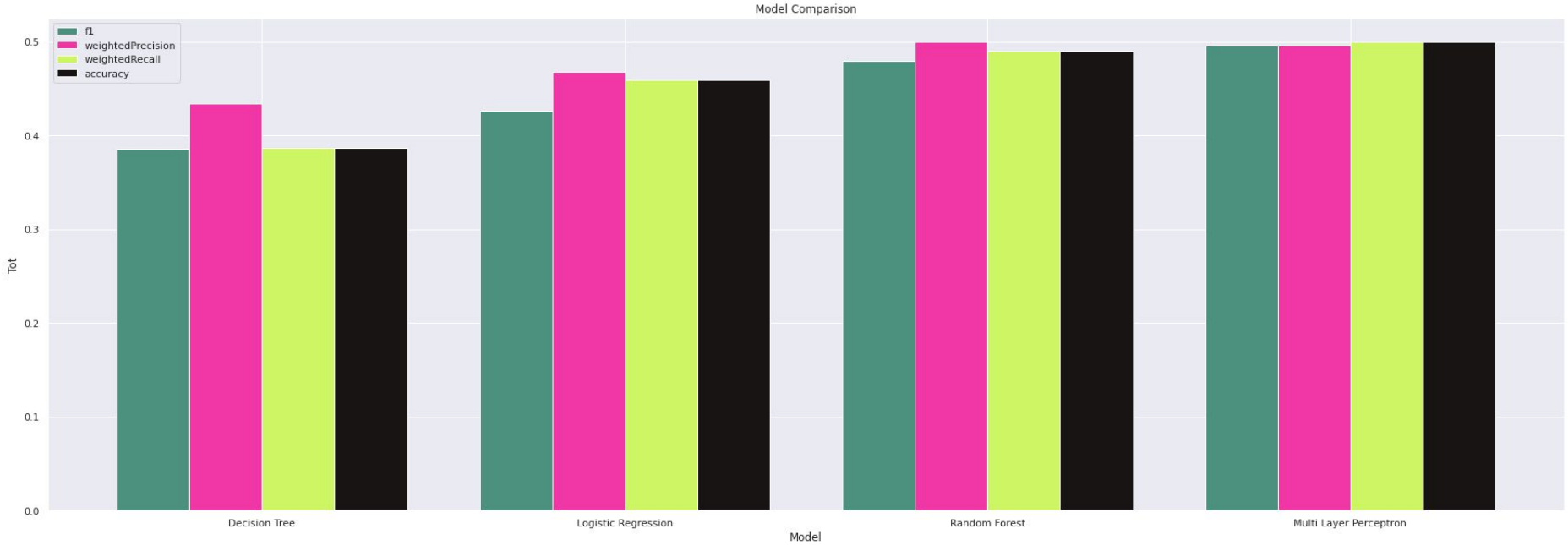


Supervised Learning: Classification

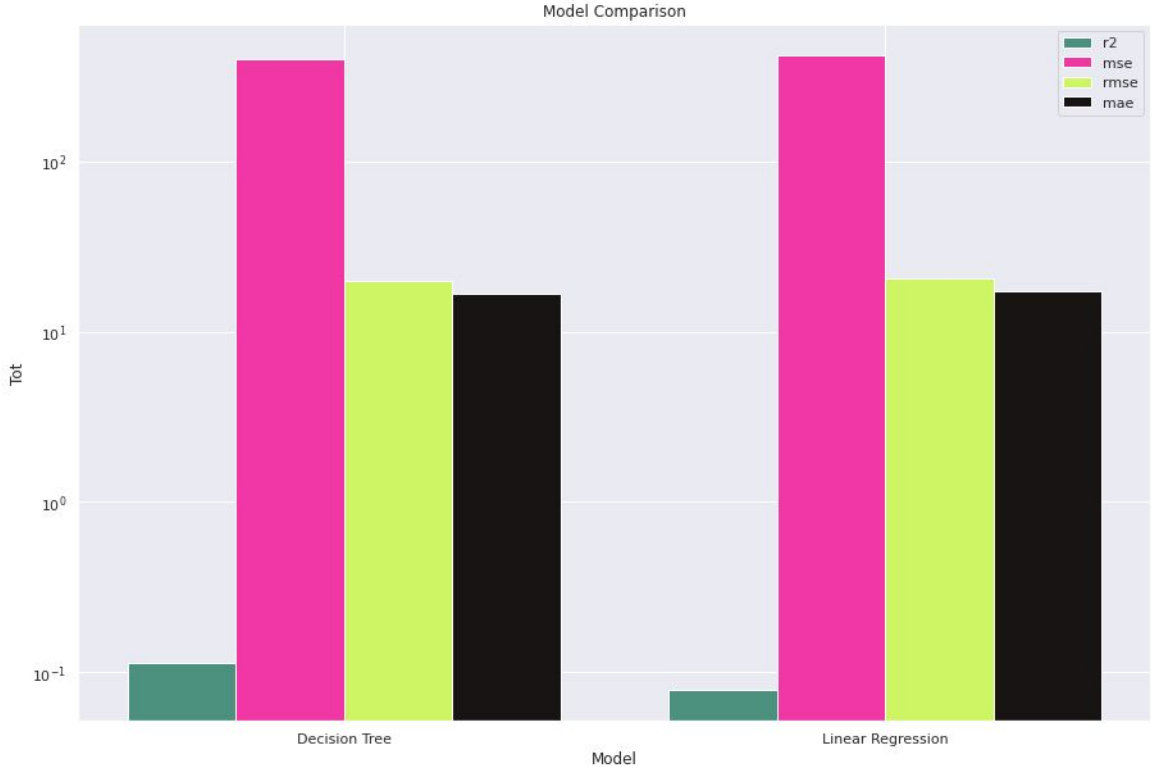
In our project we decided to test the following models:

- Decision Tree Classifier.
- Logistic Regression.
- Random Forest Classifier.
- Multi Layer Perceptron Network.

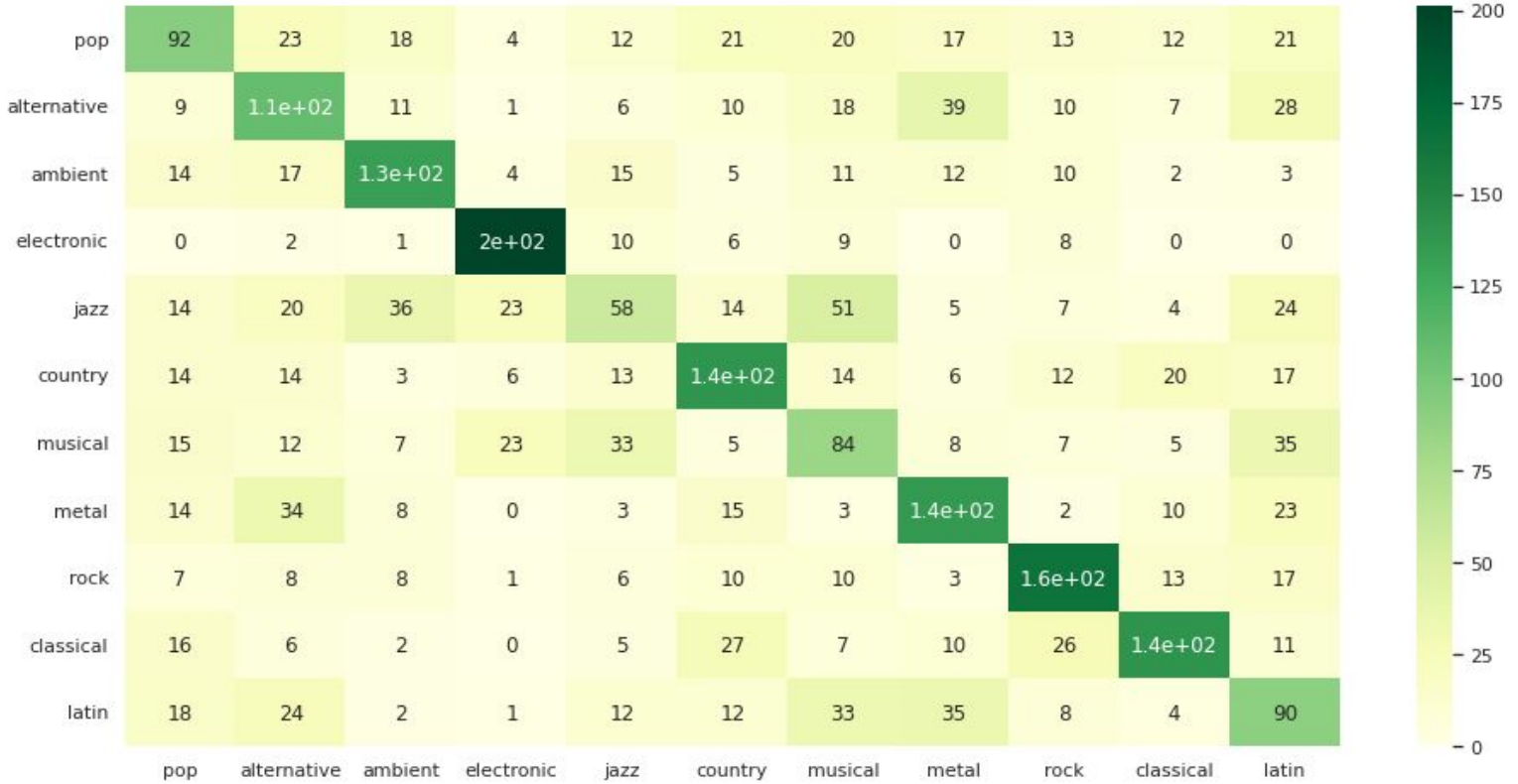
We have decided to separate the Training from the Pipeline for time and memory reasons in order to run each model independently.



Model Comparison: Music genre classification with Cross Validation



Model Comparison: Popularity regression with Cross Validation



Confusion Matrix: MLP Genre

Weakness of the Classifier

Genre:

- to make the problem easier we have reduced the number of target classes;
- this was done **manually** by looking at the **historical / social influences** between the various genres;
- this is not a **reliable** method, as it is subject to human errors, moreover a track can have influences from several different genres.

Popularity:

- Spotify for many tracks does not rank popularity (default 0) → the result is an **unbalanced** dataset
- In addition, the songs with high popularity (> 70) are in a much lower percentage than the others.

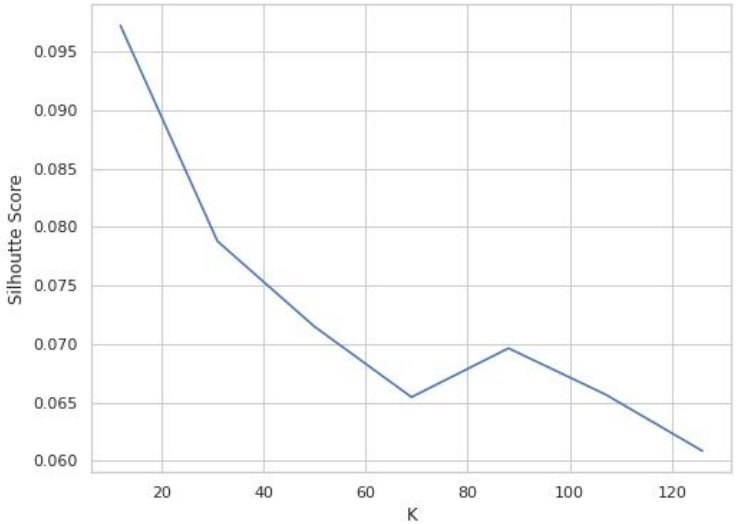
Unsupervised Learning: K-Means Clustering

We decided to train k-means clustering algorithm by setting the number of clusters (**k**) equals to 12, that represents the “big-genres”.

We used the following parameters:

- distance measure: **cosine similarity**
- metric measure: **silhouette**

To evaluate k-means we print the silhouette score in order to find a **local maxima**.



Recommender System

The idea is to recommend songs to the user, to be included in their own playlist.

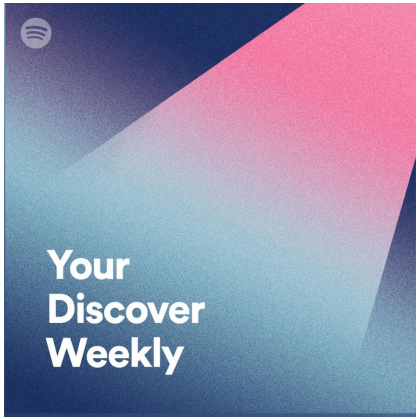
Problem:

- Spotify Web Api does not provide data on other users
- It is possible to obtain a user's playlist only if they are *public* and if you know their *Spotify ID* in advance.

For these reasons we have chosen **Content-based filtering** to develop this system.

The algorithm can be divided into two steps:

1. Creating the **summary vector**;
2. Find the **similarity** and **suggest** the songs.



Recommender System: Summary Vector

Summarize by **adding** all the songs in a playlist into a **vector** that can be compared to the main dataset to find their similarities.

Playlist Dataset			
track_ID	acousticness	...	danceability
4kDgnKgZTX6puRz9EoNjle	0,1	...	0,8
69LvvkHnFFnX3d5eNObtMo	0,4	...	0,23
3k0YJnqMKRZb8swo86vCkq	0,6	...	0,3
6mADjHs6FXdroPzEGW6KVJ	0,5	...	0,01
1VSuFS7PahCN3SWbOcQ98m	0,7	...	0,6
7Ct4wK3Fpp8vvZGcWy4NCP	0,2	...	0,4
...	...	...	...

Summary Vector	2,7	...	8,08

Recommender System: Similarity

We use **cosine similarity** to calculate the similarity between the summary vector and each song of the main dataset (not including the songs in the playlist).

$$\text{Cosine Sim}(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Once the similarity has been calculated, we add it as a new column to the dataset, and then sort it in **descending order** and select the first 50 songs to recommend.

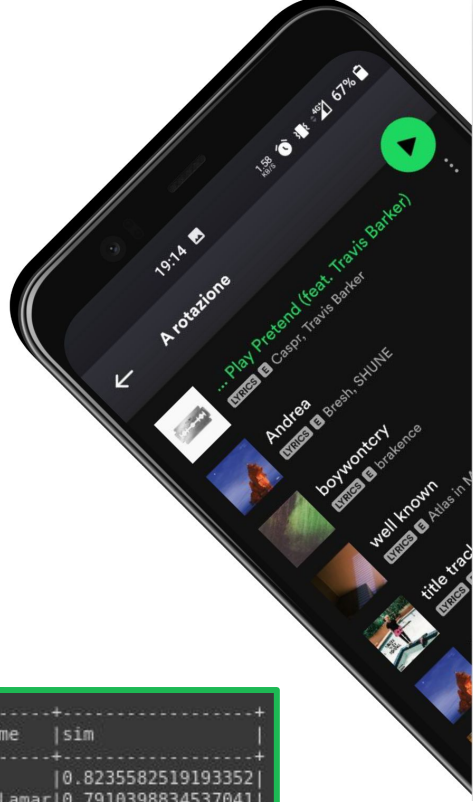
Recommender System

Advantage:

- the temporal complexity, as it is equivalent to a multiplication between matrices.

Problem:

- there are no metrics to evaluate whether the recommendation is good or bad.



track_name	album_name	artist_name	sim
For Free (feat. Drake)	Major Key	DJ Khaled	0.8235582519193352
All The Stars (with SZA)	Black Panther The Album Music From And Inspired By	Kendrick Lamar	0.7910398834537041
Do Not Disturb	More Life	Drake	0.7852022293401946
Blow Your Mind (Mwah)	Dua Lipa (Deluxe)	Dua Lipa	0.7507494193185866
Never Know	FREE 6LACK	6LACK	0.7424160329316578
Something New (feat. Ty Dolla $ign)	Rolling Papers 2	Wiz Khalifa	0.7423087497549848
Guatemala - From Swaecation	SR3MM	Rae Sremmurd	0.7384320978763972
Lost	channel ORANGE (Explicit Version)	Frank Ocean	0.7359717317889709
Devil (feat. Busta Rhymes, B.o.B & Neon Hitch)	Devil (feat. Busta Rhymes, B.o.B & Neon Hitch)	Cash Cash	0.7339734372535278
Love Scars	A Love Letter To You	Trippie Redd	0.726035836058776

only showing top 10 rows