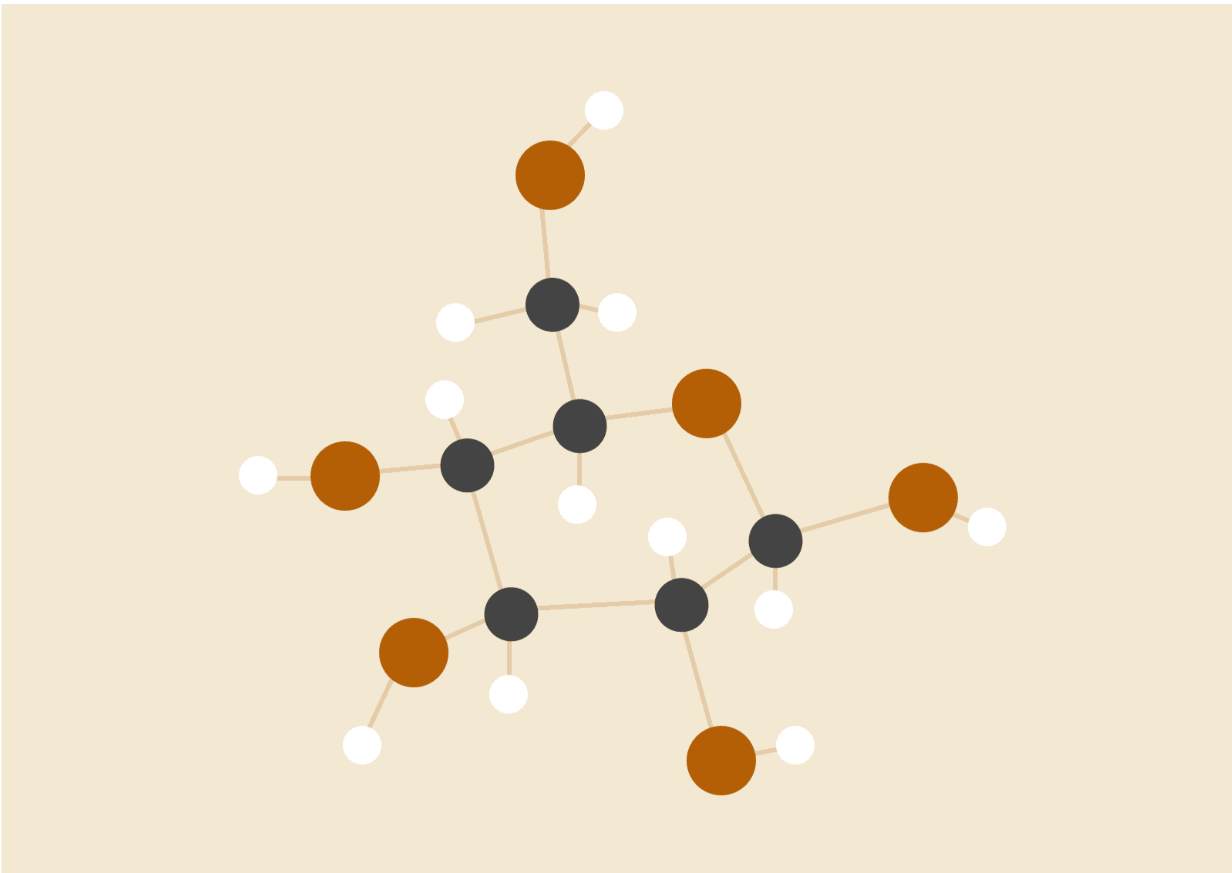
MACHINE LEARNING



Maxime BALLESTEROS - PEREZ

Brunilda QUSHKU

Leo LAGARDE

Anthony BERNABEU

Tiago CLARENC

16TH June 2022

EXECUTIVE SUMMARY

I - Import & data extraction

II - EDA

III - Data cleaning

IV - Feature Engineering

V - Model Training

- **Logistic Regression**
- **Random Forest**
- **XGBoost**

VI - Performance Evaluation

- **Performance Metrics (F1 Score, Accuracy score, Recall score, Precision score)**
- **Confusion Matrix**
- **Classification Report**
- **Learning curves**

INTRODUCTION

Our company asked us to work in a machine learning model that can help them to classify music tracks into genres.

Our smart and beautiful team, composed of five data scientists, is going to help the company to find a way to classify music.

Before starting our journey with the dataset we need to import all the tools that will help us to assess cleaning, feature engineering and implement ML models later :

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_sc
from sklearn.metrics import plot_confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
# code so we can import functions from src folder

import os
import sys

# import python scripts we created to help with feature engineering
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)


from src.helpers import identify_number_categories, identify_missing_data
from src.outliers import calc_outliers
from src.feature_importance_plot import feature_importance_plot
from src.learning_curve_plot import learning_curve_plot
```

Data extraction

Our company provided us with the dataset from their archives. We determined it was usable thanks to two criterias:

- Sufficient number of columns (x19)
- Sufficient number of rows (x32833)

Thanks to the code :

```
# Info on all datatypes of each column

df_tracks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32833 entries, 0 to 32832
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   track_id          32833 non-null  object
 1   track_name        32828 non-null  object
 2   track_artist      32828 non-null  object
 3   track_popularity  32833 non-null  int64
 4   playlist_name     32833 non-null  object
 5   playlist_id       32833 non-null  object
 6   genre             32833 non-null  object
 7   danceability      32833 non-null  float64
 8   energy            32833 non-null  float64
 9   key               32833 non-null  int64
 10  loudness          32833 non-null  float64
 11  mode              32833 non-null  int64
 12  speechiness       32833 non-null  float64
 13  acousticness      32833 non-null  float64
 14  instrumentalness  32833 non-null  float64
 15  liveness          32833 non-null  float64
 16  valence           32833 non-null  float64
 17  tempo             32833 non-null  float64
 18  duration_ms       32833 non-null  int64
dtypes: float64(9), int64(4), object(6)
memory usage: 4.8+ MB
```

We understand that there are 10 numerical columns and 9 non- numerical columns.

Our future step will be to encode the non - numerical columns that we think are important for our model.

Exploratory Data Analysis (EDA)

To start, we decided to identify the columns that are numericals, the one that are categoricals and the one that are booleans, thanks to this two codes:

```python
# Numerical Values
numerical_columns = list(df_tracks.select_dtypes(['int64']) + df_tracks.select_dtypes(['float64']))

#numerical_columns
print('- The numerical columns are : ', numerical_columns)
```

```
- The numerical columns are :  ['acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness', 'key',
'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'track_popularity', 'valence']
```

```python
# Categorical Values
categorical_columns = list(df_tracks.select_dtypes(['object']))

#categorical_columns
print('- The categorical columns are : ', categorical_columns)
```

```
- The categorical columns are :  ['track_id', 'track_name', 'track_artist', 'playlist_name', 'playlist_id', 'genre']
```

```python
# Boolean Values
boolean_columns = list(df_tracks.select_dtypes(['bool']))

#boolean_columns
print('- The boolean columns are : ', boolean_columns)
```

```
- The boolean columns are :  []
```

The fact that comes in our eyes is that there are no booleans .

After that we decided to analyze the Standard Deviation which is a number that describes how spread out the values are.

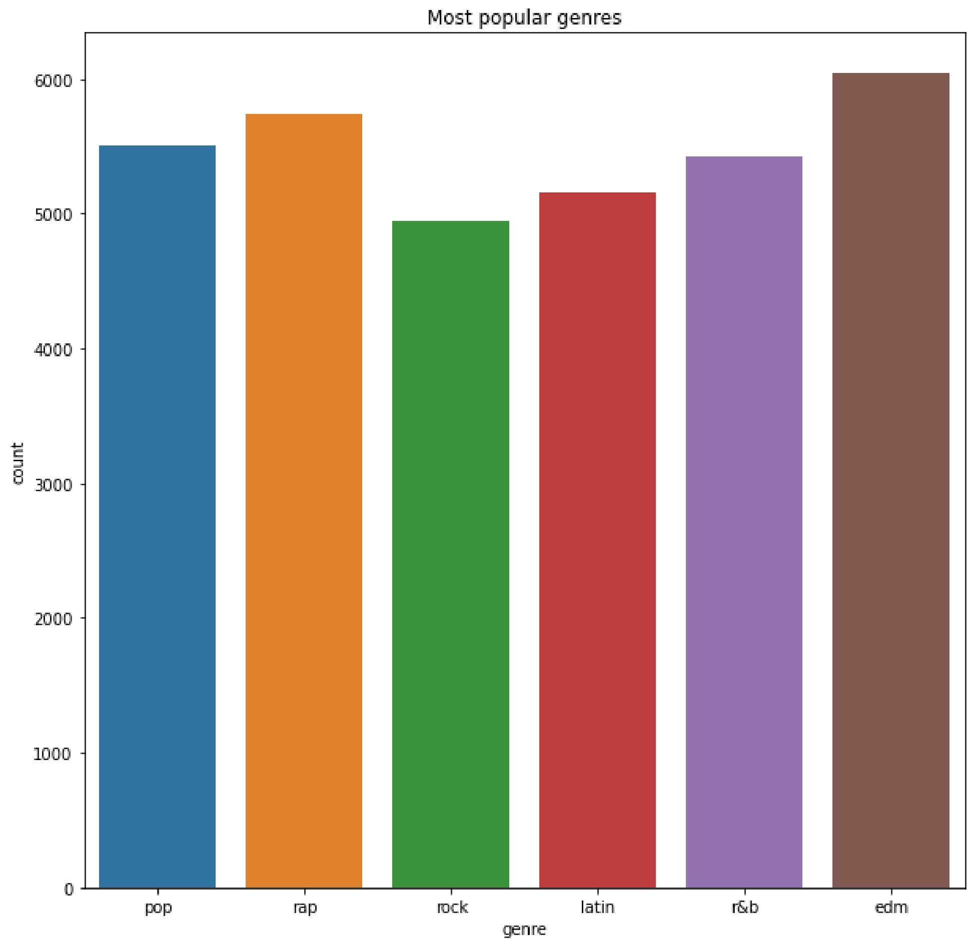
A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range.

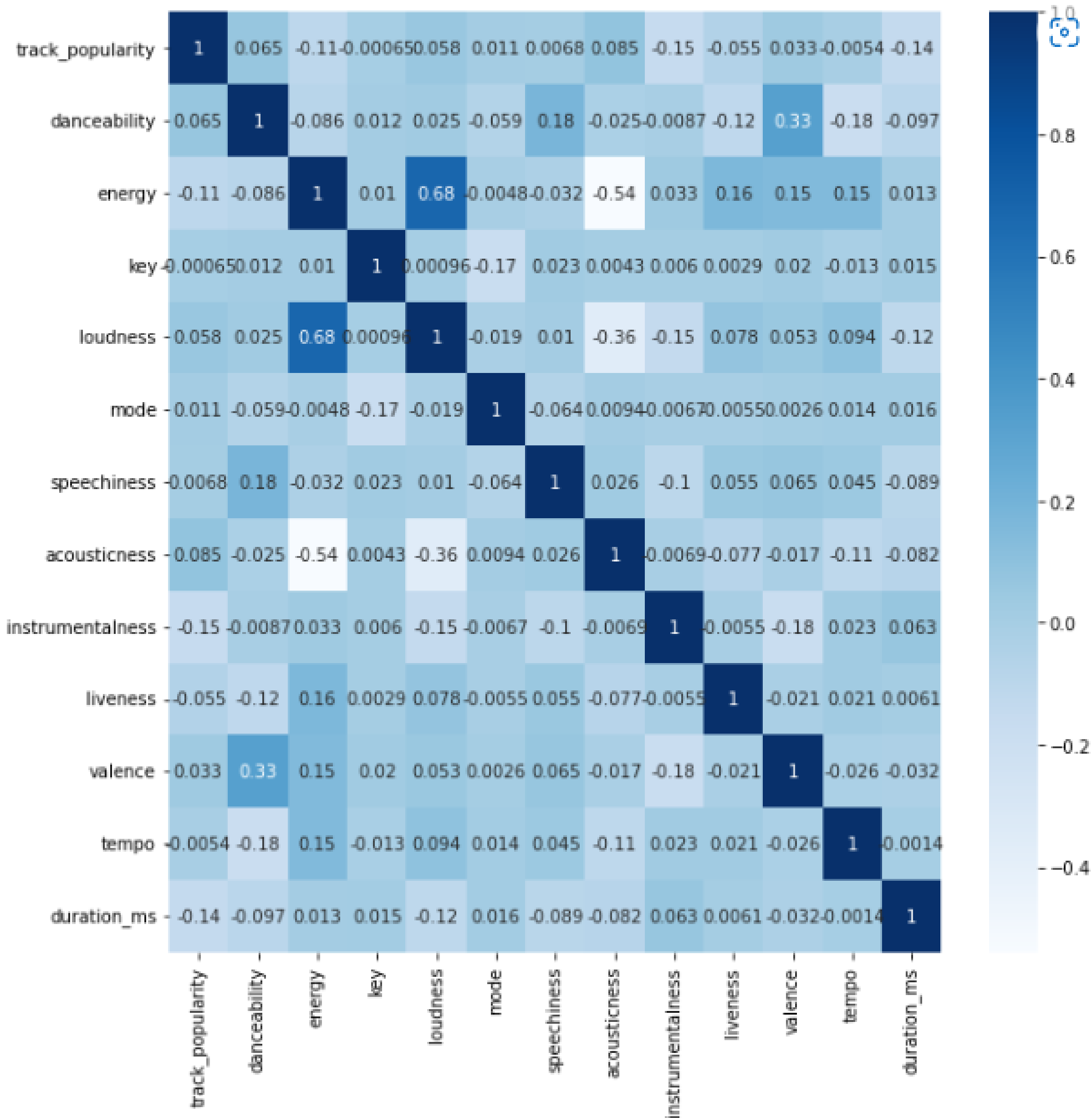
In [19]:
```python
# Check the Standard Deviation for each column

std_dev = df_tracks.std()

print("Standard Deviation of each column :  ", std_dev)
```

```
Standard Deviation of each column :   track_popularity       24.984074
danceability            0.145085
energy                  0.180910
key                     3.611657
loudness                2.988436
mode                    0.495671
speechiness             0.101314
acousticness            0.219633
instrumentalness        0.224230
liveness                0.154317
valence                 0.233146
tempo                  26.903624
duration_ms         59834.006182
dtype: float64
```

We therefore tried to explore some of the features of the dataset and visualize what was the most popular artists or what was the most popular genres for example:

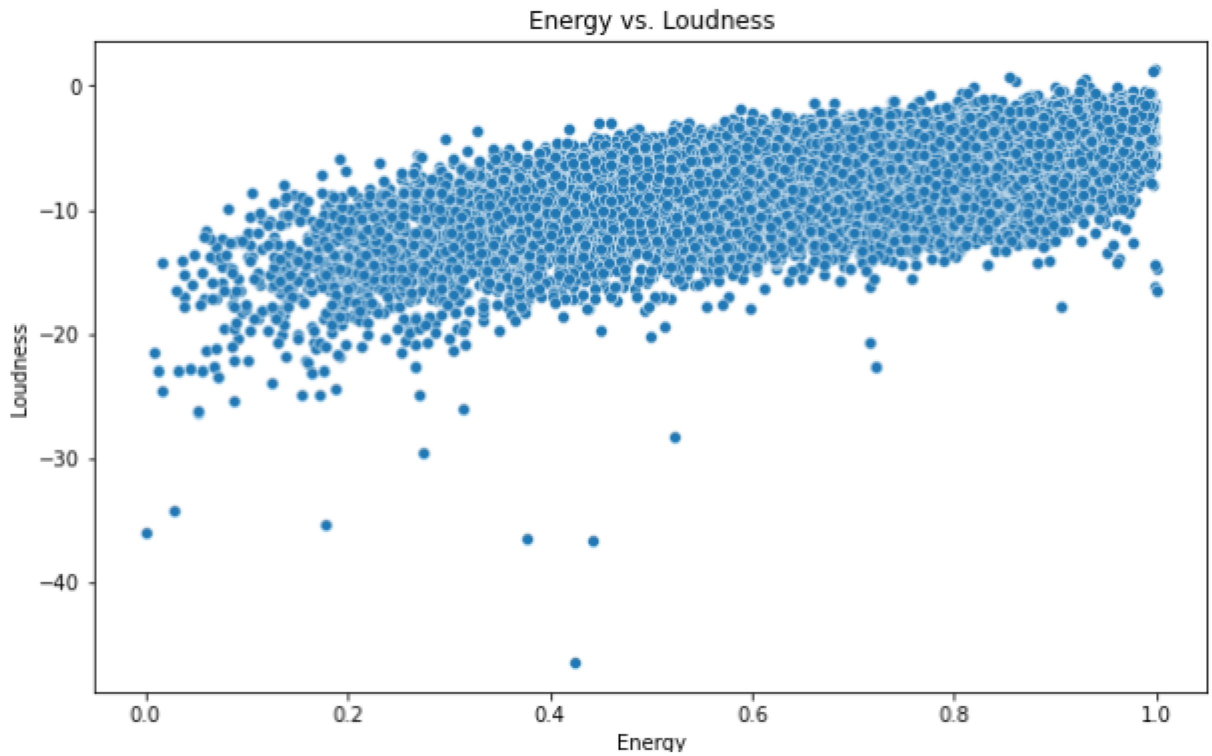


An important step of EDA is to create a correlation matrix to look at the correlations between the data. To better understand and visualize the correlation between variables we plot using a heatmap :

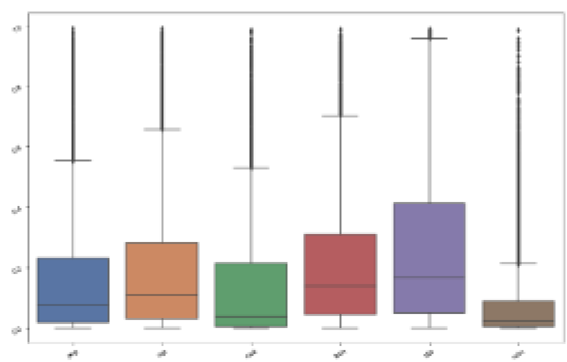


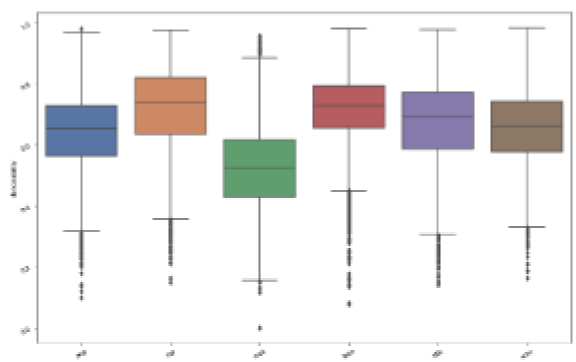
We can see a high correlation rate of 0.68 between loudness and energy or a potential correlation between valence & danceability (0.33).

To explore those potential relationship other tools were used in this part for visualization like scatter plot or boxplot :



The scatter plot shows indeed a relationship between Energy Vs Loudness which makes sense because Energy represents how high sound variations are on a specific song and loudness represents the volume level of the song.





Data cleaning

To initiate the data cleaning, we started to identify all missing values in the data set :

```
Entrée [108]: # Determine all NULL values in our dataset

              df_tracks.isnull().sum()

Out[108]: track_id          0
          track_name        5
          track_artist      5
```

We noticed 5 rows in the track_name & track_artist columns filled with NaN values. No need to keep empty rows in the dataset so we created a new data set called **df_tracks_cleaned** without those 5 rows.

```
Entrée [109]: df_tracks_cleaned = df_tracks[~df_tracks["track_artist"].isnull()]
```

To pursue our data cleaning process, it was necessary to check the categorical values for each categorical columns :

	categorical_feature_name	count
0	track_id	28352
1	track_name	23449
2	track_artist	10692
3	playlist_name	449
4	playlist_id	471
5	genre	6

At this point, we decided to remove all the "big" categorical features with a lot of different values : track_id, track_name.

We also wanted to remove the 2 features related to playlists : **playlist_name** & **playlist_id**, since they seem to confuse us about genres of each music.

We decided to keep our last categorical feature : track_artist, and extend our analysis on how to exploit it to feed our future machine learning models.

Here is a first observation on this column :

```
Entrée [127]: df_tracks_cleaned["track_artist"].value_counts().nlargest(50)

   Out[127]: Martin Garrix                161
             Queen                        136
             The Chainsmokers             123
             David Guetta                 110
             Don Omar                     102
             Drake                        100
             Dimitri Vegas & Like Mike     93
             Calvin Harris                 91
             Hardwell                      84
             Kygo                          83
             Guns N' Roses                 79
```

Let’s not forget the goal of our project here : Predict the genre of any music in the Dataset.

In order to accomplish this we need to define a “Target”. In our case, Target is created based on the genre of the dataset ***df_tracks_cleaned.genre***.

TARGET DEFINITION

```
# Creating a target value, containing the target used in our future models

target = df_tracks_cleaned.genre
target
```

```
0        pop
1        pop
2        pop
3        pop
4        pop
        ...
32828    edm
32829    edm
32830    edm
32831    edm
32832    edm
Name: genre, Length: 32828, dtype: object
```

FEATURES SELECTION

As stated during the previous Exploratory Data Analysis step, the Loudness and Energy features are highly correlated, and since loudness does not distinguish between genres as much as energy, the Loudness feature will be dropped for modeling.

We drop all categorical features (except track_artist) and loudness :

```
# Feature definition

features = df_tracks_cleaned.drop(['genre', 'track_id', 'track_artist', 'track_

features
```

Feature engineering

In this part, the next step is to encode our last categorical variables that we think are relevant to keep to feed our future prediction models. We decided to focus on the genre and track_artist columns.

Let's begin with encoding our target (genre) values using **sklearn preprocessing.LabelEncoder()** that encodes target labels with values between 0 and n_classes-1.
This transformer should be used to encode target values, *i.e.* `y`, and not the input `X`.

```
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df_tracks_cleaned['genre_encoded'] = label_encoder.fit_transform(df_tracks_clea

df_tracks_cleaned['genre_encoded'].unique()
```

We stored the encoded genre column into the target set given a new array of integers from 0 to 5.

```
target = df_tracks_cleaned['genre_encoded']
target
```

Applying a value_counts we were able to gather rows (tracks) by genre regarding the new indexing :

```
df_tracks_cleaned['genre_encoded'].value_counts()
```

```
0    6043
4    5743
2    5507
3    5431
1    5153
5    4951
Name: genre_encoded, dtype: int64
```

Now, we will encode our last categorical column : track_artist.

Model training

As we are working on the classification of audio tracks by musical genre, we will apply 4 models that we have seen in class: Linear Regression, Decision Tree, Random Forest and XGBoost Regressor

1. Decision Tree

Now we can use the lower dimensional PCA projection of the data to classify songs into genres.

Here, we will be using a simple algorithm known as a decision tree. Decision trees are rule-based classifiers that take in features and follow a 'tree structure' of binary decisions to ultimately classify a data point into one of two or more categories. In addition to being easy to both use and interpret, decision trees allow us to visualize the 'logic flowchart' that the model generates from the training data.

2. Logistic Regression

We will start by applying **logistic regression**. Logistic regression makes use of what's called the logistic function to calculate the odds that a given data point belongs to a given class. It’s important to test different ML models to compare them on a few performance metrics, such as false positive and false negative rate (or how many points are inaccurately classified).

3. Random Forest

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

4. XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

Performance evaluation

To assess the performance of our models, we need to use the appropriate metrics for each of Them, then gather these collected results in the same table to compare it and find the best model to complete prediction on our test set.

- ACCURACY
- RECALL
- PRECISION
- F1 SCORE

Algorithm Name	ACCURACY Score	RECALL Score	PRECISION Score	F1 Score
Logistic Regression				
Random Forest				
XQBoost				

REFERENCES

1. https://www.geeksforgeeks.org/ml-label-encoding-of-datasetsin-python/
2. https://docs.google.com/spreadsheets/d/11o0En1bFxQD5BpPdRfU_wMPAFvCVTvtCch5hvZDwREM/edit#gid=0
3. https://github.com/shaq31415926/audio_data_clustering
4. https://xgboost.readthedocs.io/en/stable/python/python_intro.html
5. https://towardsdatascience.com/random-forest-in-python-24d0893d51c0