université
PARIS-SACLAY

Introduction à l'Apprentissage

Projet : Classification des chansons par genre Musical

Groupe :

- ★ AMRANE Lydia
- ★ BENALI Laetitia
- ★ NAIT-LARBI Takfarinas

Chargé cours et TP : Mr.Giancarlo Fissore

Sommaire

A. Introduction:

● Contexte :

Dans le cadre de notre formation Master Data Science de l'université paris saclay, et du module **introduction à l'apprentissage automatique,** nous sommes chargés de réaliser un projet dans le thème de l'apprentissage automatique sur les notions que nous avons abordées.
Le projet sur lequel nous avons décidé de travailler, et un projet de prédiction d'une variable qualitative (problème de classification Multi Classes), pour cela on a choisi un projets parmi les projets proposés par notre chargé du module, (Monsieur Giancarlo Fissore), il s'agit de celui de prédiction du genre musicale

● Objectifs du projet :

La réalisation du projet a plusieurs objectifs :

- Mise en pratique des notions vues en cours
- rédaction d'un rapport succinct et détaillé du projets
- Techniquement, Prédire le genre d'une musique à partir de ses différentes caractéristiques

● Etapes du Projet:

Pour atteindre ces objectifs on a suivi une approche qui est répartie en 3 étapes :

1 - Exploration des données/ Data Processing
2 - Application des modèles prédictifs
3 - Interprétation des résultats

● Répartition des Tâches :

Au cours de ce projet nous avons décidé en premier lieu de nous retrouver lors d'ateliers de réflexion et de travail collectif afin d'avancer plus rapidement et plus efficacement.
Puis, une fois que notre ligne directrice a été définie, nous nous sommes partagés les tâches concernant les modèles à explorer chacun de notre côté ainsi que la rédaction du rapport.

B. Exploration des données :

1. **Source de données :**

Le Dataset sur lequel nous nous sommes basés est fourni sur :
https://www.kaggle.com/mrmorj/dataset-of-songs-in-spotify

2. **Description des données :**

Notre source de données est constituée de 2 fichiers: genres.csv et playlists.csv.

Lors de l'exploration des deux fichiers, nous nous sommes rendus compte que le fichier playlists.csv ne nous apporterait pas d'informations pertinentes pour ce projet que nous voulions mener, à savoir la prédiction de genres musicaux.
Genres.csv est notre "main" fichier.

Genre.csv est un fichier composé de :

- 22 Features (colonnes) :

    - **danceability (float64)** : Mesure qui décrit à quel point un morceau est adapté à la danse.
    - **energy (float64)** : Mesure l'énergie d'une musique.
    - **key (int64)**: une clé sur une musique donnée.
    - l**oudness (float64)**: Mesure l'intensité d'une musique.
    - **mode (int64)**
    - **speechiness (float64)** : Mesure qui détecte la présence de mots prononcés dans une piste.
    - **acousticness (float64)** : Mesure l'acoustique d'une musique.
    - **instrumentalness (float64)** : Mesure le taux d'instrumentalisation d'une musique.
    - **liveness (float64) :** Mesure qui décrit la probabilité que le morceau ait été enregistré avec un live.
    - **valence (float64)** : Mesure qui décrit la positivité musicale véhiculée par un morceau.
    - **tempo (float64)** : Le tempo moyen d'une musique.
    - **type (object):** Audio_features.
    - **id (object) :** Identifiant d'une musique.
    - **uri (object) :** L'uri d'une musique.
    - **track_href (object) :** Un lien vers le point de terminaison de l'API Web fournissant des détails complets sur la piste.
    - **analysis_url (object) :** URL HTTP pour accéder à l'analyse audio complète de cette piste.
    - **duration_ms (int64) :** Durée d'une musique.

- **time_signature (int64) :** signe qui indique la longueur d'une composition
- **genre (object) :** composé des genres musicaux (Trap, Techno, Techhouse, Trance, Psytrance, Dark Trap, DnB (drums and bass), Hardstyle, Underground Rap, Trap Metal, Emo, Rap, RnB, Pop and Hiphop)
- **song_name (object) :** Nom de la musique
- **Unnamed: 0 (float64) :** Top si la chanson est nommée ou pas (1 pour oui, 0 pour non)
- **title (object) :** Titre de la musique (proportionnel au fait que la chanson n'a pas été nommée)

- 42 297 exemples (individus) :

  - Le fichier a une volumétrie de données sur 42 297 lignes, chaque ligne représente donc une musique différente.

NB :

_____Dans le reste du rapport le terme **"DataSet"** fait référence au fichier **GENRE.CSV**

3. Analyse Statistique des données:

Statistiques descriptives univariés :

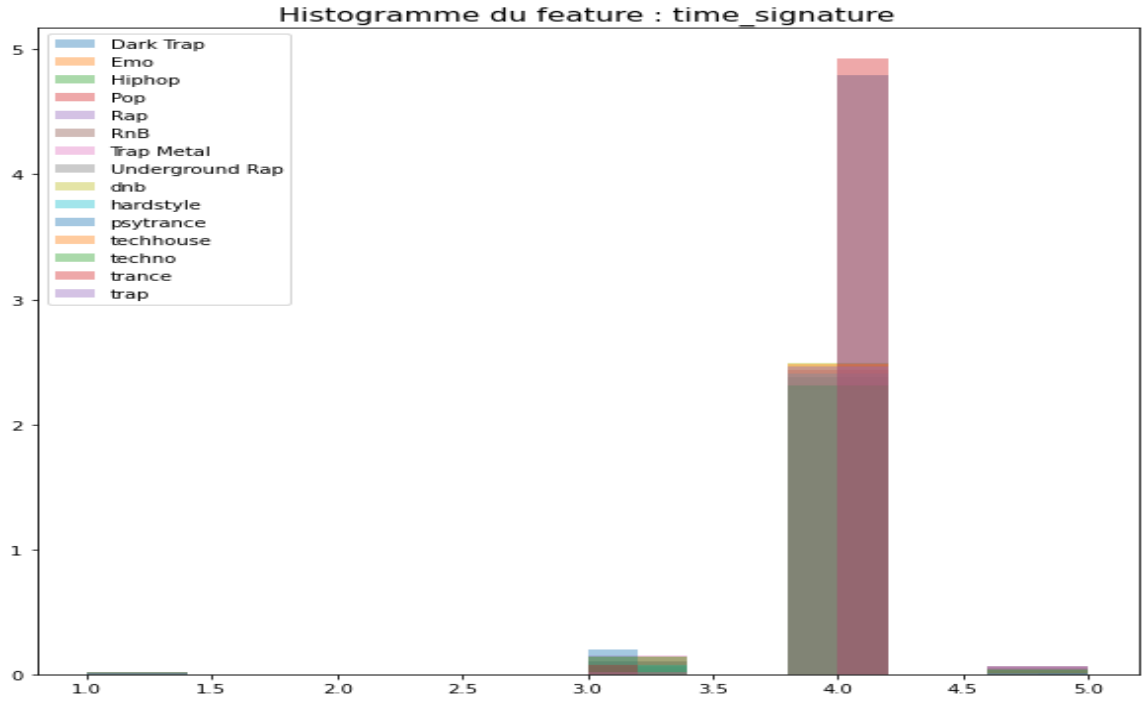
Description du dataSet :

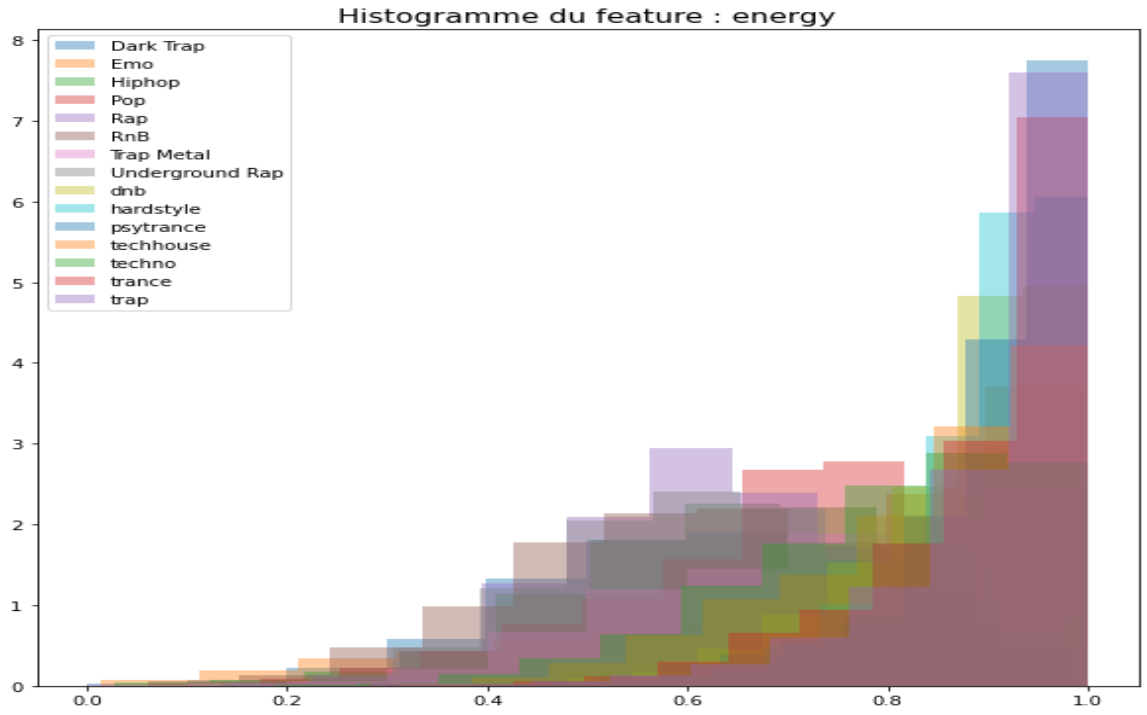
	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
count	42297	42297	42297	42297	42297	42297	42297	42297	42297	42297	42297	42297	42297
mean	0.64	0.76	5.37	-6.47	0.55	0.14	0.10	0.28	0.21	0.36	147.47	250884.77	3.97
std	0.16	0.18	3.67	2.94	0.50	0.13	0.17	0.37	0.18	0.23	23.84	102954.45	0.27
min	0.07	0.00	0.00	-33.36	0.00	0.02	0.00	0.00	0.01	0.02	57.97	25600.00	1.00
25%	0.52	0.63	1.00	-8.16	0.00	0.05	0.00	0.00	0.10	0.16	129.93	179853.00	4.00
50%	0.65	0.80	6.00	-6.23	1.00	0.08	0.02	0.01	0.14	0.32	144.97	224771.00	4.00
75%	0.77	0.92	9.00	-4.51	1.00	0.19	0.11	0.72	0.29	0.52	161.45	301133.00	4.00
max	0.99	1.00	11.00	3.15	1.00	0.95	0.99	0.99	0.99	0.99	220.29	913052.00	5.00

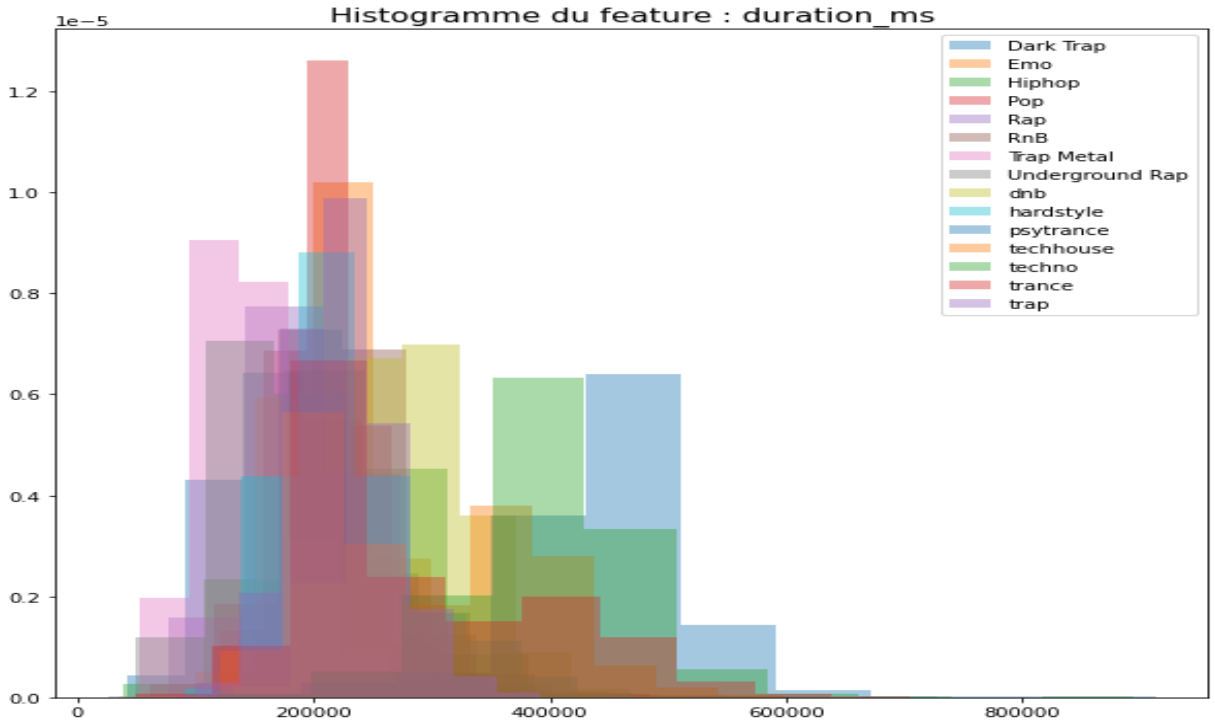
Le tableau ci-dessus nous donne la description de toutes les variables quantitatives de notre jeu de données directement.

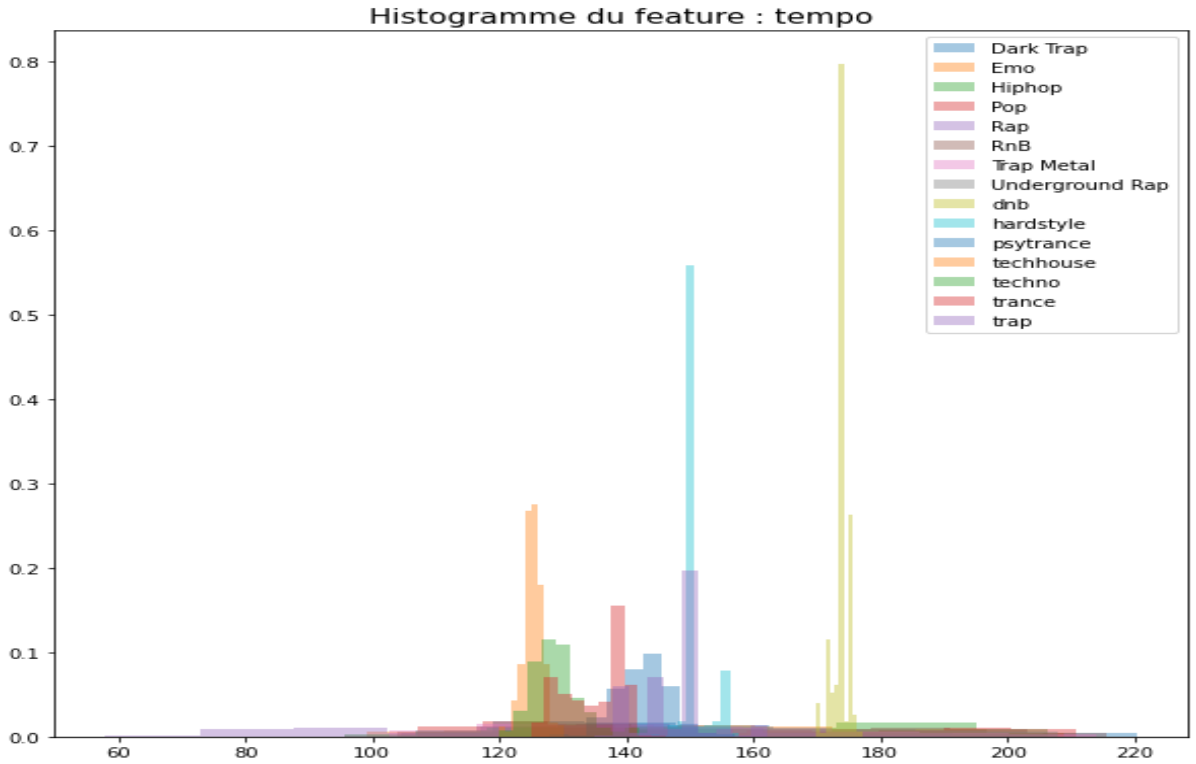
- Histogrammes :

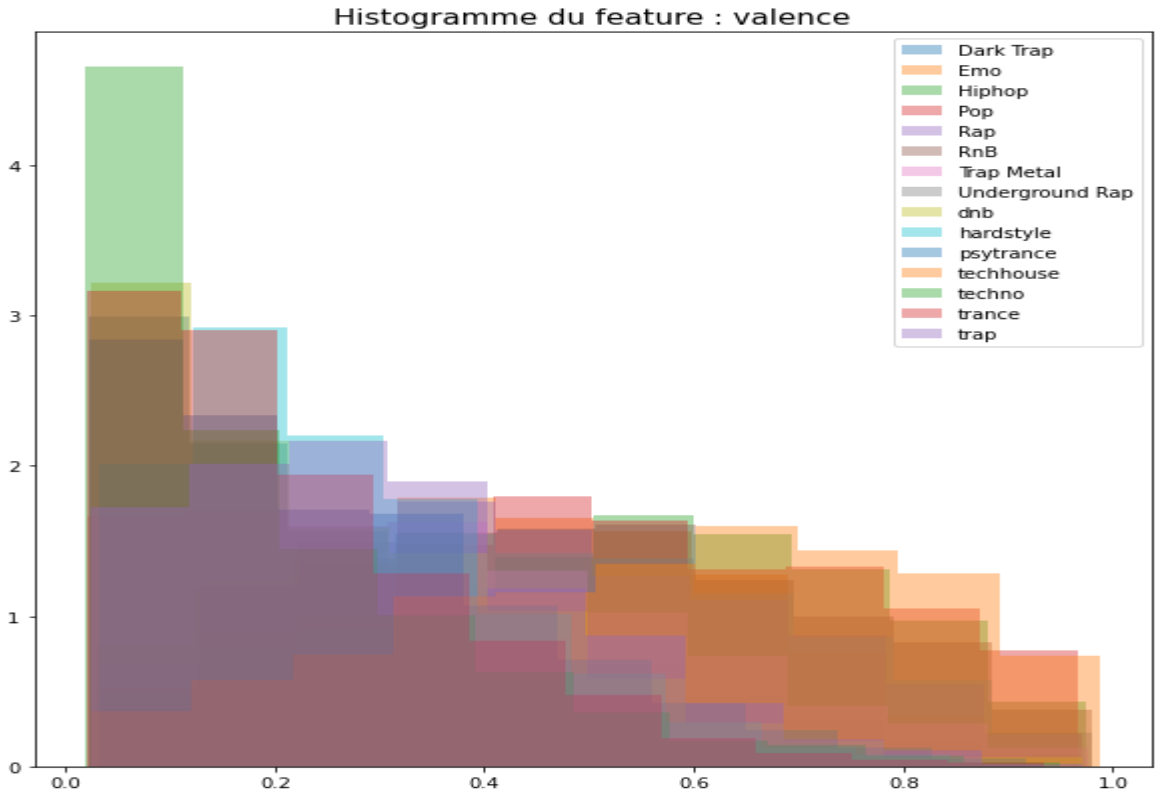
Dans cette étape on va afficher pour chaque feature numérique son histogramme, ce qui va nous permettre d'estimer la distribution de chaque feature en fonction des genre musicaux :

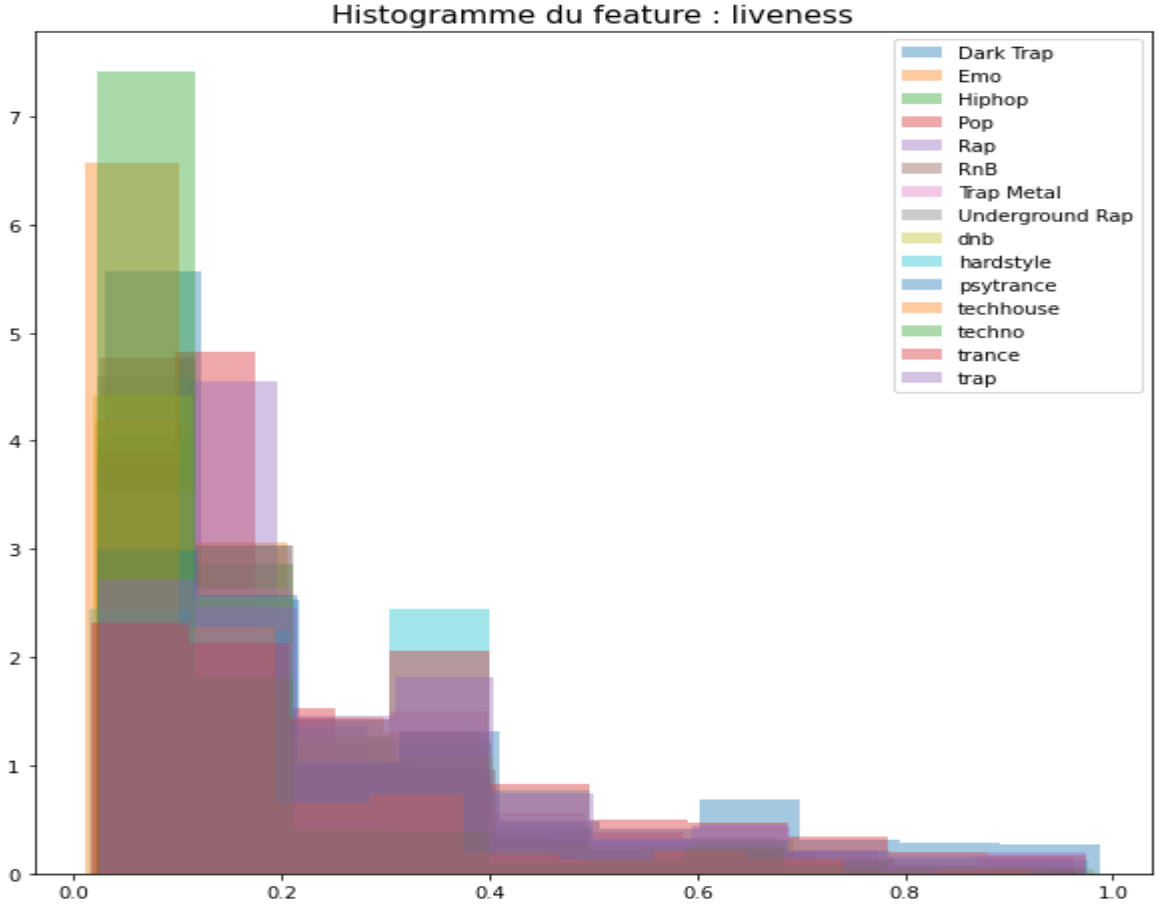


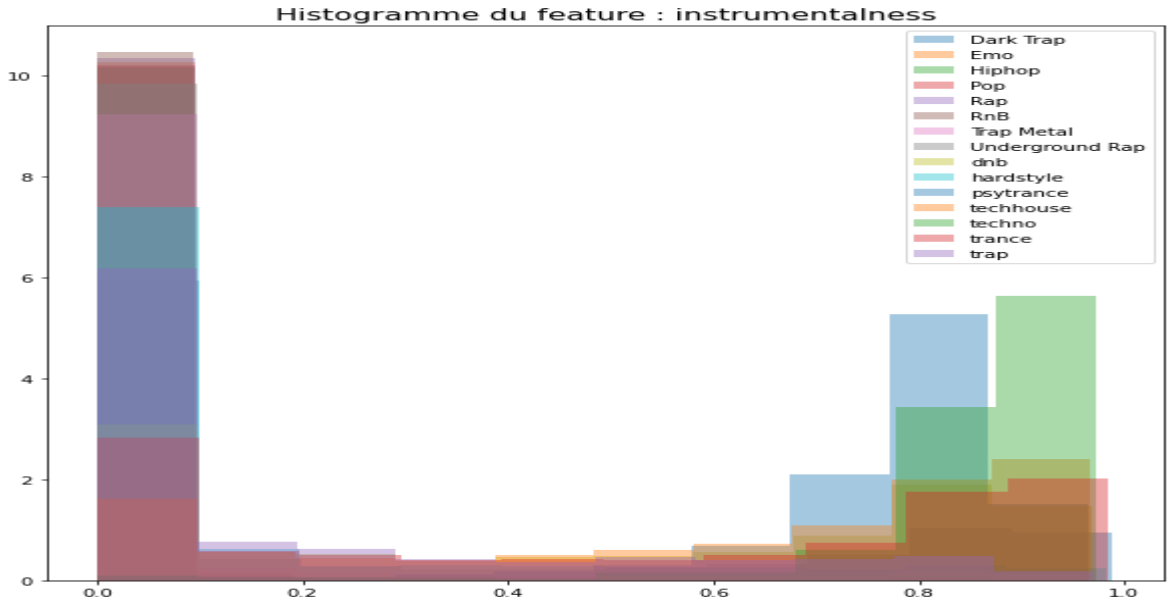


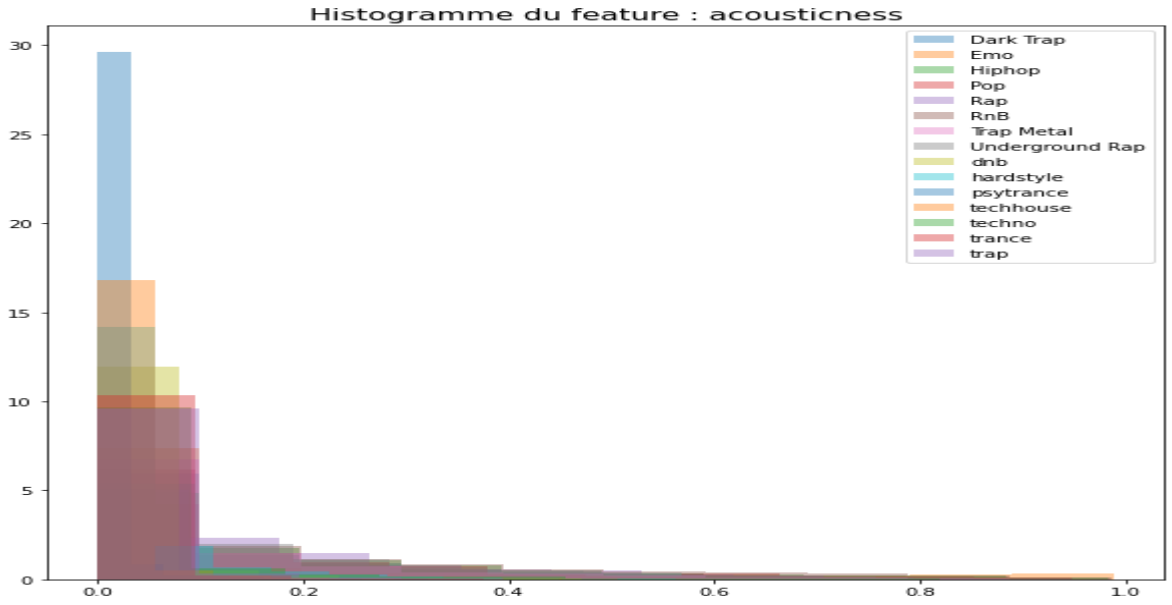

Histogramme du feature : duration_ms
1e−5
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
200000
400000
600000
800000
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap

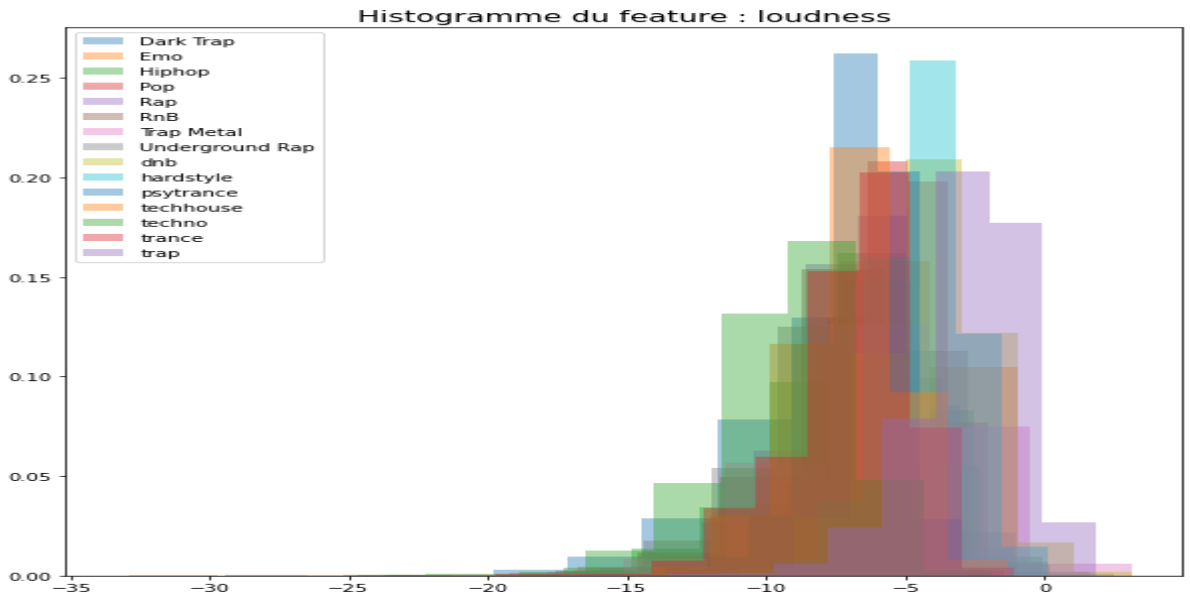

Histogramme du feature : tempo
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
60
80
100
120
140
160
180
200
220
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap

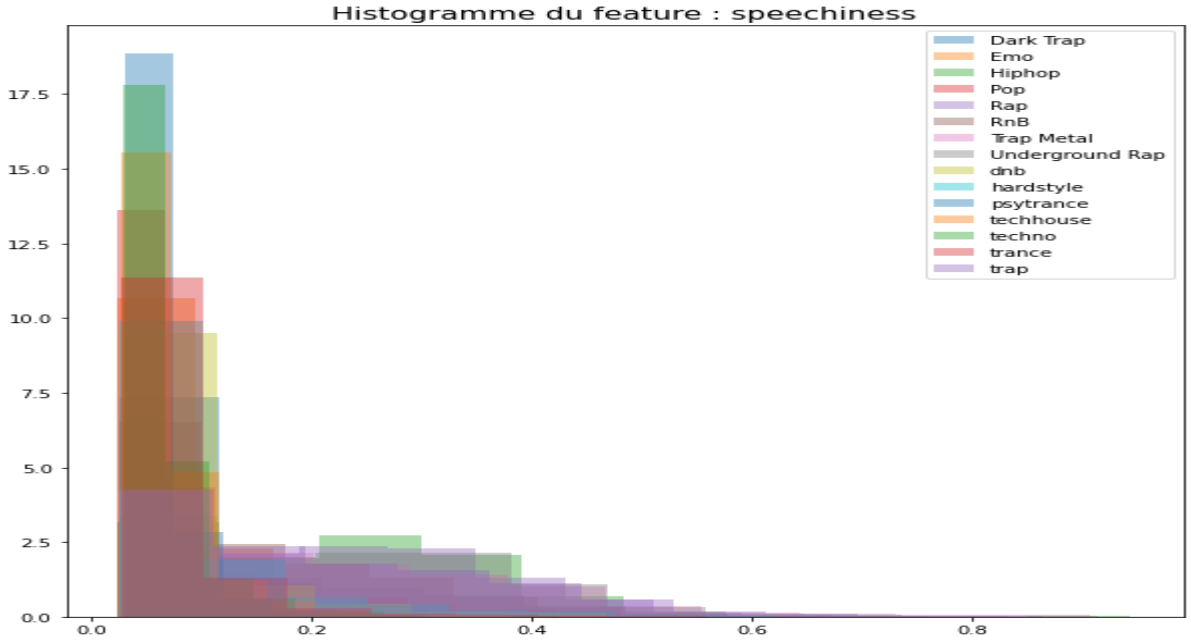

Histogramme du feature : valence
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0

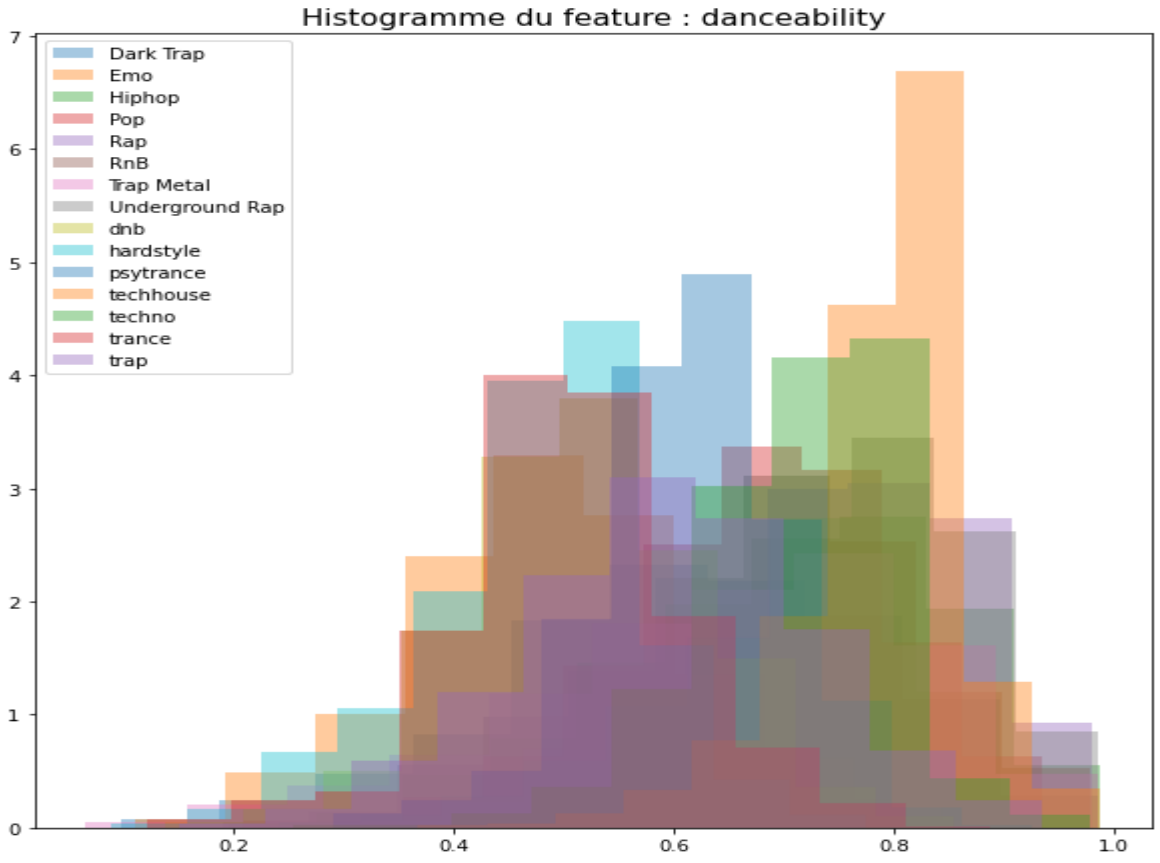

Histogramme du feature : liveness
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0
1
2
3
4
5
6
7
0.0
0.2
0.4
0.6
0.8
1.0


Histogramme du feature : instrumentalness
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


Histogramme du feature : acousticness
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0


Histogramme du feature : loudness
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0.00
0.05
0.10
0.15
0.20
0.25
−35
−30
−25
−20
−15
−10
−5
0


Histogramme du feature : speechiness
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
0.2
0.4
0.6
0.8


Histogramme du feature : danceability
Dark Trap
Emo
Hiphop
Pop
Rap
RnB
Trap Metal
Underground Rap
dnb
hardstyle
psytrance
techhouse
techno
trance
trap
0
1
2
3
4
5
6
7
0.2
0.4
0.6
0.8
1.0

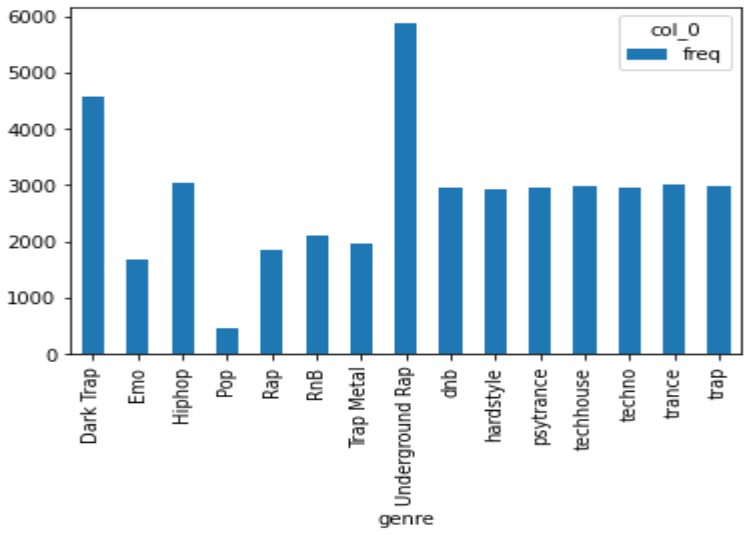
- **Variable Qualitative [GENRE] :**

- Pour les variables qualitatives, il y a plusieurs façons de faire pour obtenir la table d'occurrences (ou des effectifs), ainsi que la table des proportions
- la variable qui nous intéresse le plus c'est la variable genre qui représente notre variable prédictive.

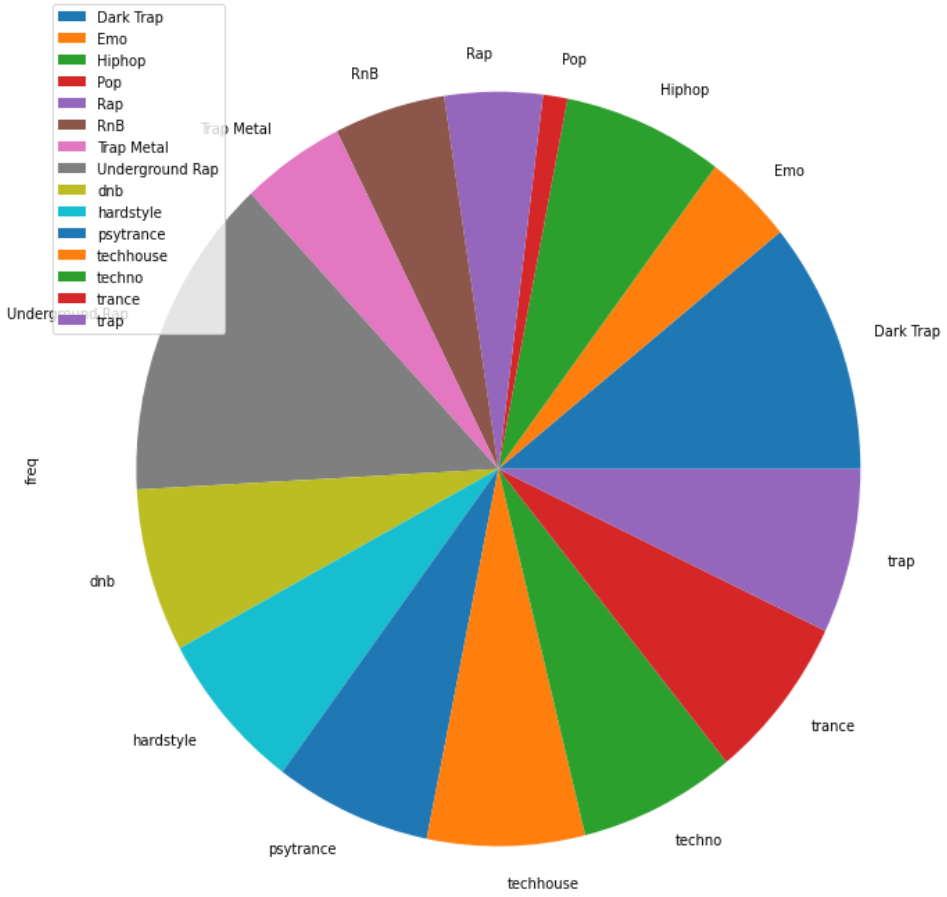
- **<u>crosstab:</u>**

genre	col_0 freq
Dark Trap	4572
Emo	1680
Hiphop	3027
Pop	461
Rap	1848
RnB	2099
Trap Metal	1955
Underground Rap	5875
dnb	2966
hardstyle	2936
psytrance	2961
techhouse	2975
techno	2956
trance	2999
trap	2987

- **<u>Diagramme en barres :</u>**



- **Pie :**



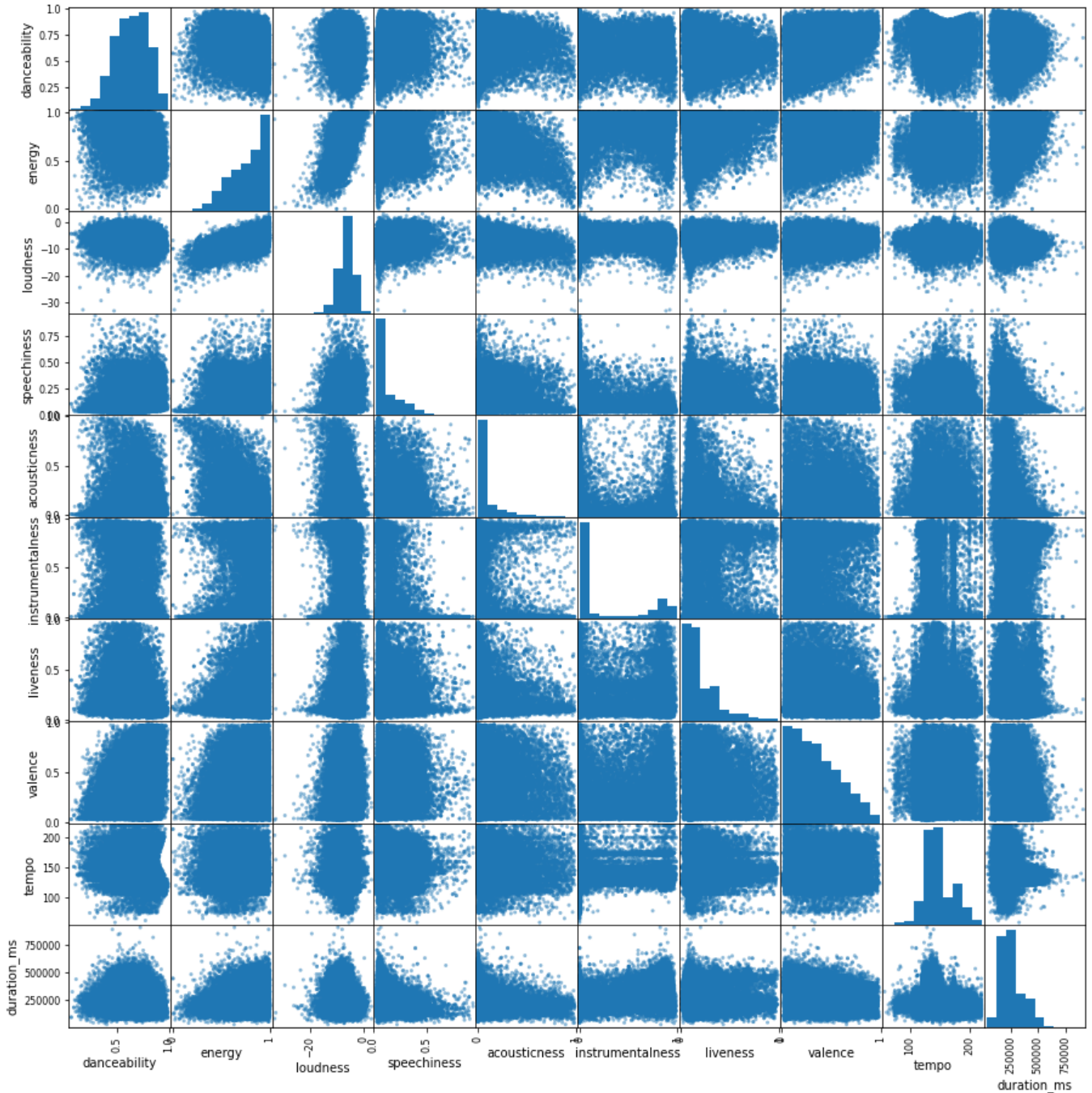
Remarque :

- On remarque que les données sont déséquilibrées selon les genres musicaux en effet on peut constater que la classe ***Underground rap*** possède **5875** exemples, En revanche, la classe ***pop*** possède seulement **461** exemples, ce qui représente une difference considerable.

- Ce déséquilibre entre les classes nous oblige à faire un traitement supplémentaire avant l'entraînement du modèle prédictif, à savoir rééquilibrer les données . [vous allez trouver la méthode qu'on a utilisé pour se remédier à ce problème dans la section II Modèles]

- **Statistiques descriptives bivariées:**

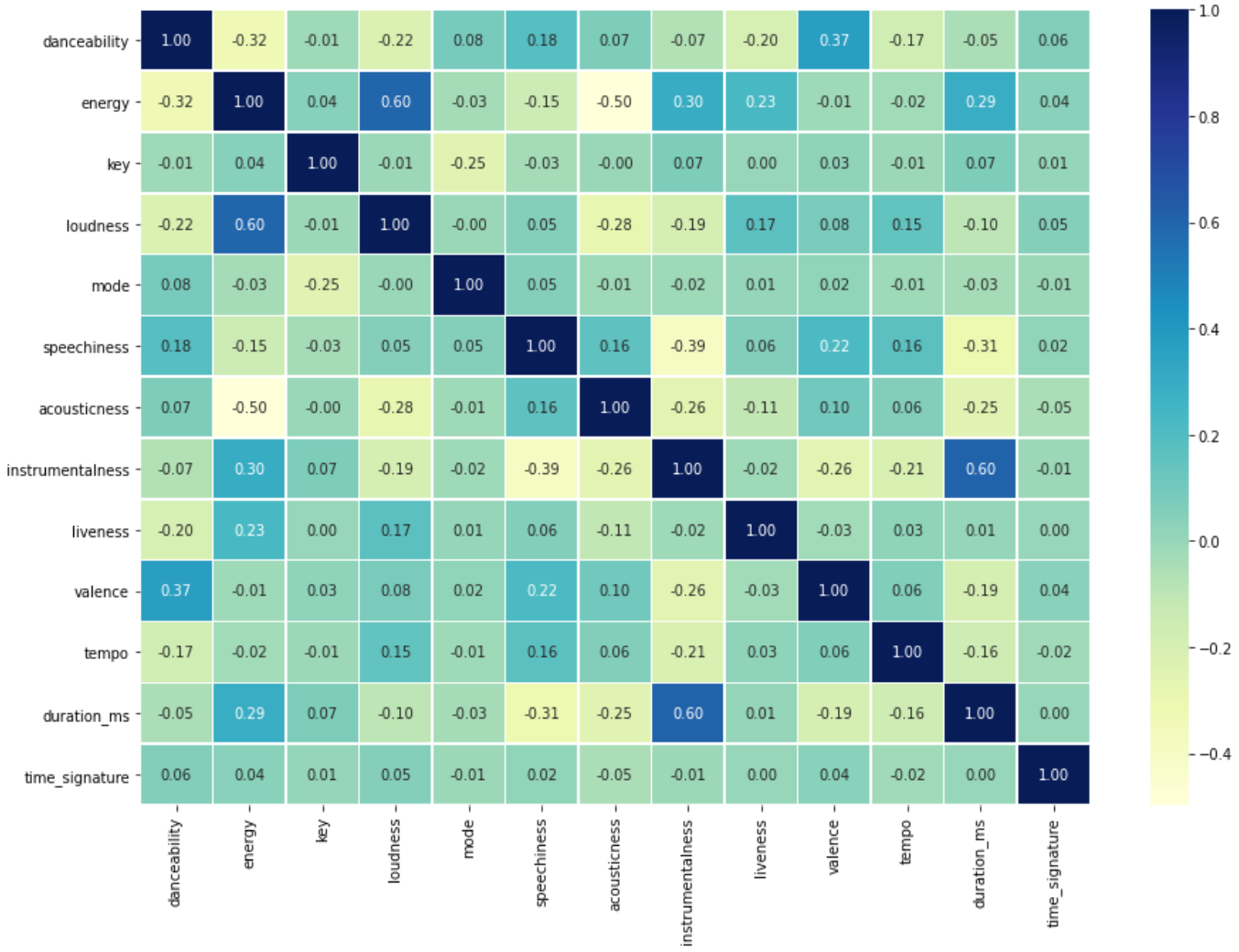
  - **Nuage de points et distributions des features:**



Remarque :

L’ensemble de données est utilisé pour faire des plots, comme le montre l'image . Chaque colonne est tracée par rapport aux autres en dehors de la diagonale. La diagonale montre les distributions des valeurs de chaque colonne.

- **Matrice de Corrélations:**



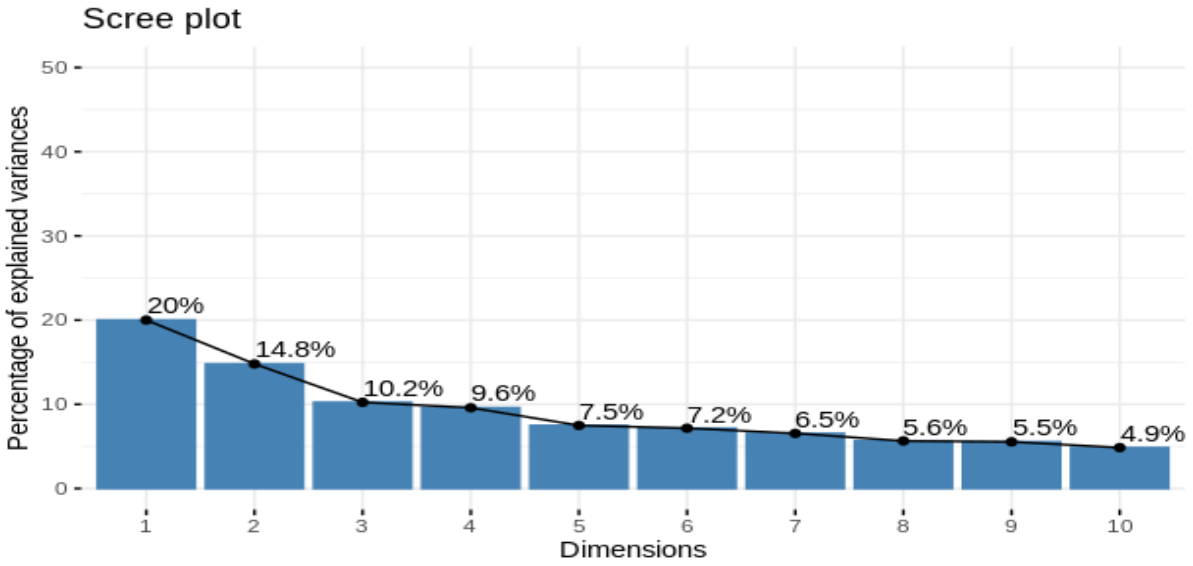
<u>Remarque :</u>

- A partir de cette matrice on peut détecter les _qui sont le plus corrélées et on peut citer :
  - Energy et Loudness
  - Instrumentalness et durration_ms
  - une anti-correlation entre energie et accousticness

- **Analyse en Composantes principale du Data Set :**

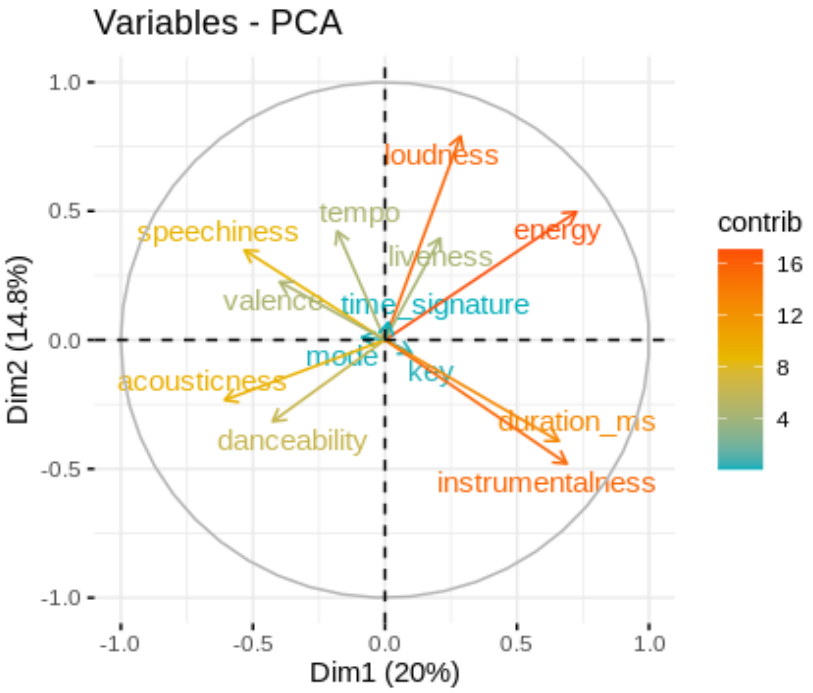
Dans cette partie on a effectué une analyse en composantes principales (ACP) afin de voir quelle variable représente le plus de variation et mieux comprendre nos données .

- Premièrement on visualise les valeurs propres, et le pourcentage de variance sur chaque axe, autrement dit, la contribution de chaque nouvelle dimension à la représentation de la donnée .



- On remarque que les 4 premières dimensions de la PCA représentent approximativement 45% de la donnée .

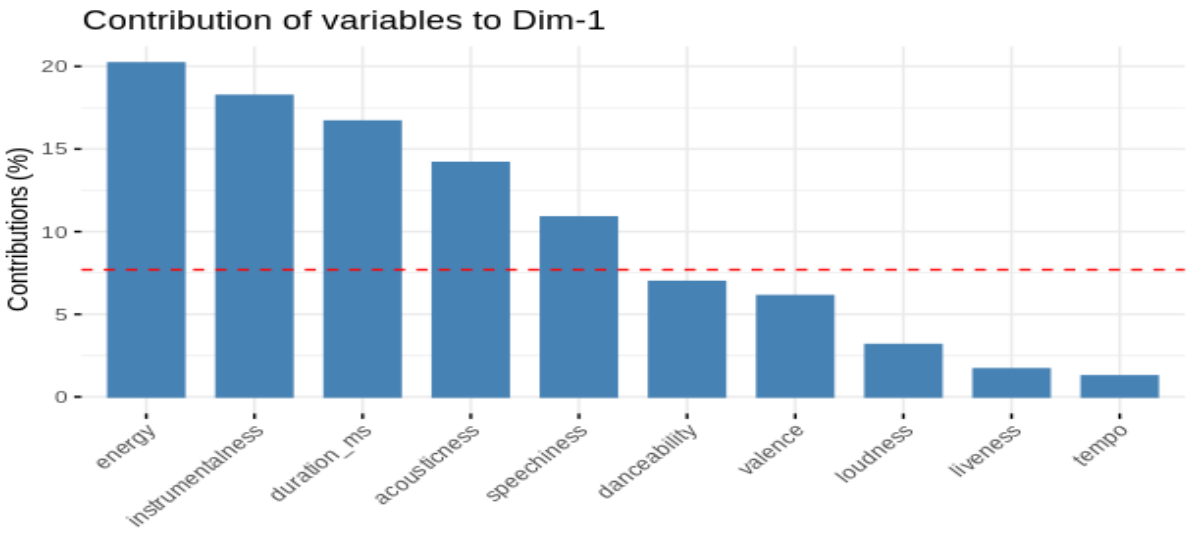
- **Cercle des correlation :**



- Ce graphe montrent les features (les variables), la colorations est en fonction de la contribution de des variables.
- Les variables corrélées positivement sont du même côté du graphique.

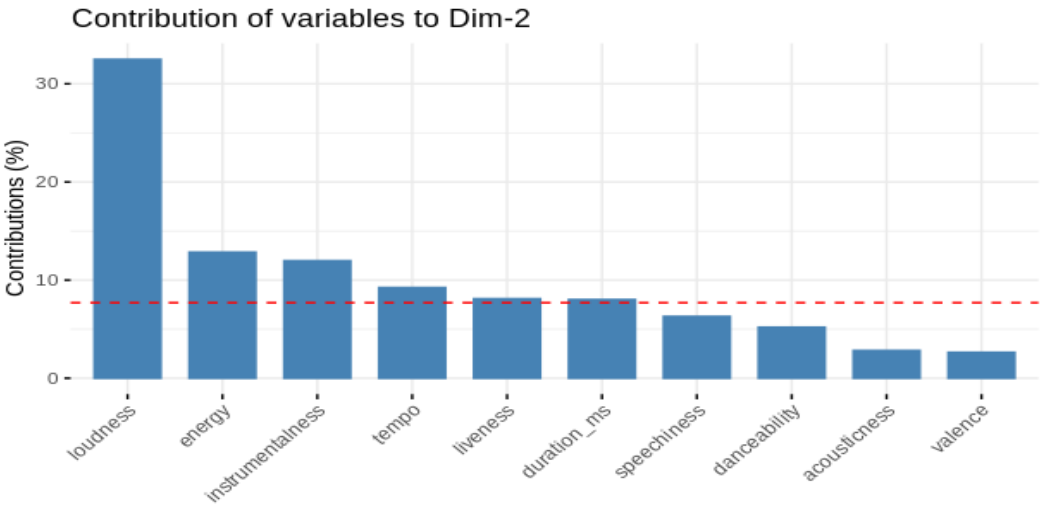
- Les variables corrélées négativement sont sur des côtés opposés du graphique.

- **Contribution des features sur la 1ere dimension de la PCA :**



- On remarque que les variables ***energie*** et ***instrumentalness*** représentent le plus de variations sur la première dimension de la PCA .

- **Contribution des features sur la 2eme dimension de la PCA :**



- On remarque que la variable ***loudness*** représente le plus de variation sur la deuxième dimension de la PCA .

C. Modèles

Dans cette deuxième partie nous allons utiliser nos données et les appliquer à plusieurs modèles de prédiction, qui seraient bien sûr potentiellement adaptés, afin d'en choisir le meilleur.
Dans un premier lieu il a fallu traiter le problème de ré-échantillonnage de nos données afin d'avoir la meilleure base possible à donner à nos modèles.
Dans un deuxième temps on appliquera des modèles déjà vus en cours tels que SVM, MLP, Naive Bayes, etc. afin de comparer les résultats.
Puis, pour aller plus loin nous avons décidé d'appliquer des modèles de Deep Learning qui seraient potentiellement intéressants à regarder tel que RandomForest

Préparation des Données :

★ **Ré-échantillonnage des données:**

- Afin de remédier au problème de déséquilibre des données entre les classes, on a opté pour un oversampling en utilisant la bibliothèque imblearn et plus exactement la classe smote.

- Cette bibliotheques nous permet de créer des nouveaux exemples pour les classes qui représentent un nombre d'exemple inférieur de ligne par rapport aux autres classes.

- Cette méthode est considérée comme une technique de pointe et fonctionne bien dans diverses applications. Cette méthode génère des données synthétiques basées sur les similitudes d'espace de fonctionnalités entre les instances minoritaires existantes. Afin de créer une instance synthétique, il trouve les K voisins les plus proches de chaque instance minoritaire, sélectionne au hasard l'un d'entre eux, puis calcule des interpolations linéaires pour produire une nouvelle instance minoritaire dans le voisinage.

★ **Exploitation des données textuelles :**

- Comme vous pouvez le constater dans la section précédente, notre data set contient deux colonnes textuelles à savoir **Song_name** et **Title**.

- Au début on a décidé de supprimer ces deux colonnes et ne pas les exploiter et se contenter uniquement des autres features,mais à cause des score trop bas de nos premiers tests de models (on avait des scores accuracy aux alentours de 0.6), on s'est vite rendu compte que peut être y'a des l'information caché dans ces deux colonnes qui peuvent nous aider lors de l'entraînement du modèle

- Pour exploiter ces colonnes textuelles on a utiliser l'approche suivante:

➢ **<u>Step 01 : Création d'un Corpus :</u>**

- Dans cette étape on va créer un texte constitué de l'ensemble des titre des chansons précédé de leurs genre musical, voici un exemple explicatif :

- Supposons on ses lignes du data set :

ID (object)	energy	……..	genre	Title
0	0.004	……..	Dark Trap	ProductOfDrugs (Prod. The Virus and Antidote)
1	0.5	……..	psytrance	From Full on to Forrest Trance

- A partir de ces deux ligne on va créer le text suivant :

" Dark Trap : ProductOfDrugs Prod The Virus and Antidote . psytrance : From Full on to Forrest Trance "

- Et on va répéter le même processus sur l'ensemble du data set

➢ **<u>Step 02 : Word Embedding :</u>**

- Dans cette étape on va créer un réseau de neurone pour le word embedding en utilisant la bibliothèque gensim.word2vec
- Notre word2vec est entraîné avec le corpus créé dans la précédente step.

- Notre word2Vec nous permet de transformer notre corpus en un espace vectoriel ou chaque mot est représenté par un vecteur .

- À la fin de cette étape on aura que tous les mots qui se ressemblent se retrouvent proche dans l'espace vectoriel **<u>par exemple</u>** :

❏ nom chanteur ou une production qui produit du **'Pop'** va se retrouver proche avec le mot ***'POP'*** dans l'espace vectoriel créé (**<u>NB :</u>** <u>Parfois dans la colonne Title on trouve le nom du chanteur ou le nom de la prod)</u>

➢ **Step 03 : Score de similarité :**

- Une fois que l'entraînement du Word2vec est terminé, on va utiliser le résultat pour calculer un score de similarité entre le titre et chaque genre musical. voici un exemple explicatif : Supposons le titre suivant : **"Grimoire (feat. King Plague)"** , après le calcul des scores de similarités on obtient le tableau suivant :

Dark Trap	0.8028022
Emo	0.90462404
Hiphop	0.9416228
techno	0.45823577
trance	0.42820308
trap	0.59222835
Pop	0.94308484
Rap	0.8313074
RnB	0.9438629
Trap Metal	0.8459748
Underground Rap	0.9742258
dnb	0.5122036
hardstyle	0.549396
psytrance	0.54583746
techhouse	0.45417893

- Enfin on insère Horizontalement ces information dans notre DataSet sur la ligne correspondante au titre en question .

NB : Après cette étape on tester des models avec des Hyper Paramètres par défauts mais un data set avant ce traitement et un data set après ce traitement et voici les résultats obtenus

Modèle	Before this Step	After This Step
SVM	0.62	0.73
MLP	0.59	0.68
RandomForest	0.63	0.7

Et Remarque que nettement les résultats sont meilleurs lorsqu' on exploite les données textuelles

★ **StandarScaler et PCA :**

- *StandarScaler :*
  l'idée derrière le StandarScaler est de transformer nos données afin d'obtenir un moyenne de 0 et une variation de 1, et d'après nos tests et résultats, on a remarqué que cette méthode améliore considérablement nos scores .
- *PCA :*
  Comme vous pouvez le constater on a appliqué la PCA sur nos données dans la partie Analyse Statistique du Dataset (page 14) afin de mieux comprendre les données .
  et dans cette partie on a fait de même mais pour un autre but qui est d'améliorer le score, mais après plusieurs Tests on a constaté que ce traitement n'améliore pas le score du modele.

NB : Afin d'appliquer ces deux derniers traitement on a utilisé la bibliothèque make_pipline afin de créer un pipeline de traitement, vous pouvez vous référer au code pour plus de détails

Applications des modèles vus en cours:

Les modèles que nous avons décidé d'explorer ont un point commun: ce sont des modèles de classification et de catégorisation de données, ils seraient donc bien adaptés à notre DataSet

a. *Support Vector Machine (SVM):*

- Comment?

Afin d'intégrer ce modèle nous avons utilisé la svm de la bibliothèque sklearn de scikit learn sur nos données déjà préparées. Vous pouvez vous reporter au code pour plus de détails.
Nous avons obtenu grâce à ce modèle un score de : 0.74256

- ***What's next?***

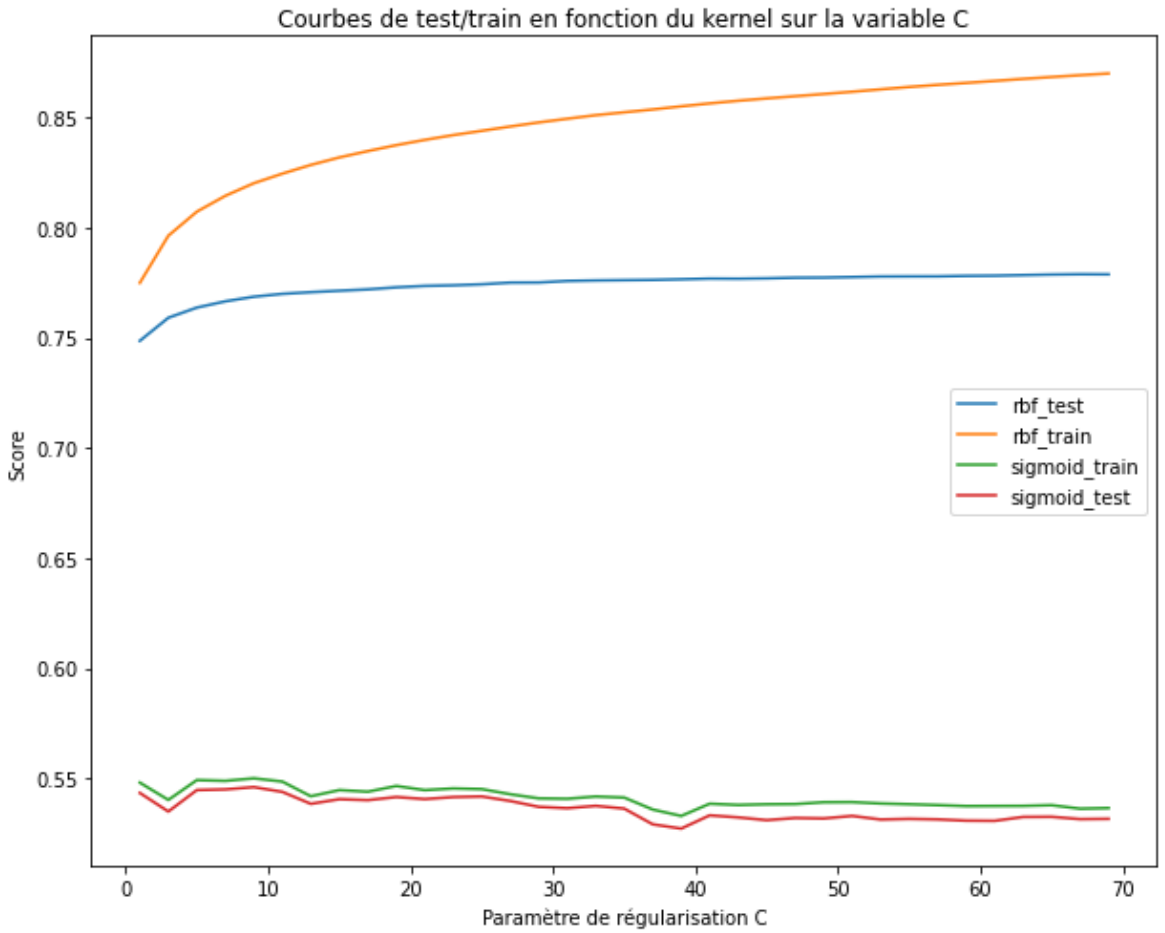
Au vu de ce score nous avons voulu essayer de l'améliorer afin d'en tirer le meilleur de la SVM, la suite logique était donc de modifier les hyperparamètres.
Pour ce faire, nous avons utilisé la méthode du GridSearch afin d'automatiser notre recherche.
Il a donc fallu repérer les hyperparamètres les plus intéressants à faire tester, et dans notre cas nous avons décidé de travailler sur:

- **le Kernel:**
  Il a un rôle important dans la SVM car il détermine le type de classification: linéaire, sigmoïde, polynomiale, etc.

- **le nombre d'itérations (max_iter):**
  Détermine le nombre d'itérations max de notre solver

- **la variable de régularisation (C):**
  La régularisation est proportionnelle à l'inverse de C, ce qui pourrait être également un bon paramètre à explorer

- ***Résultats:***

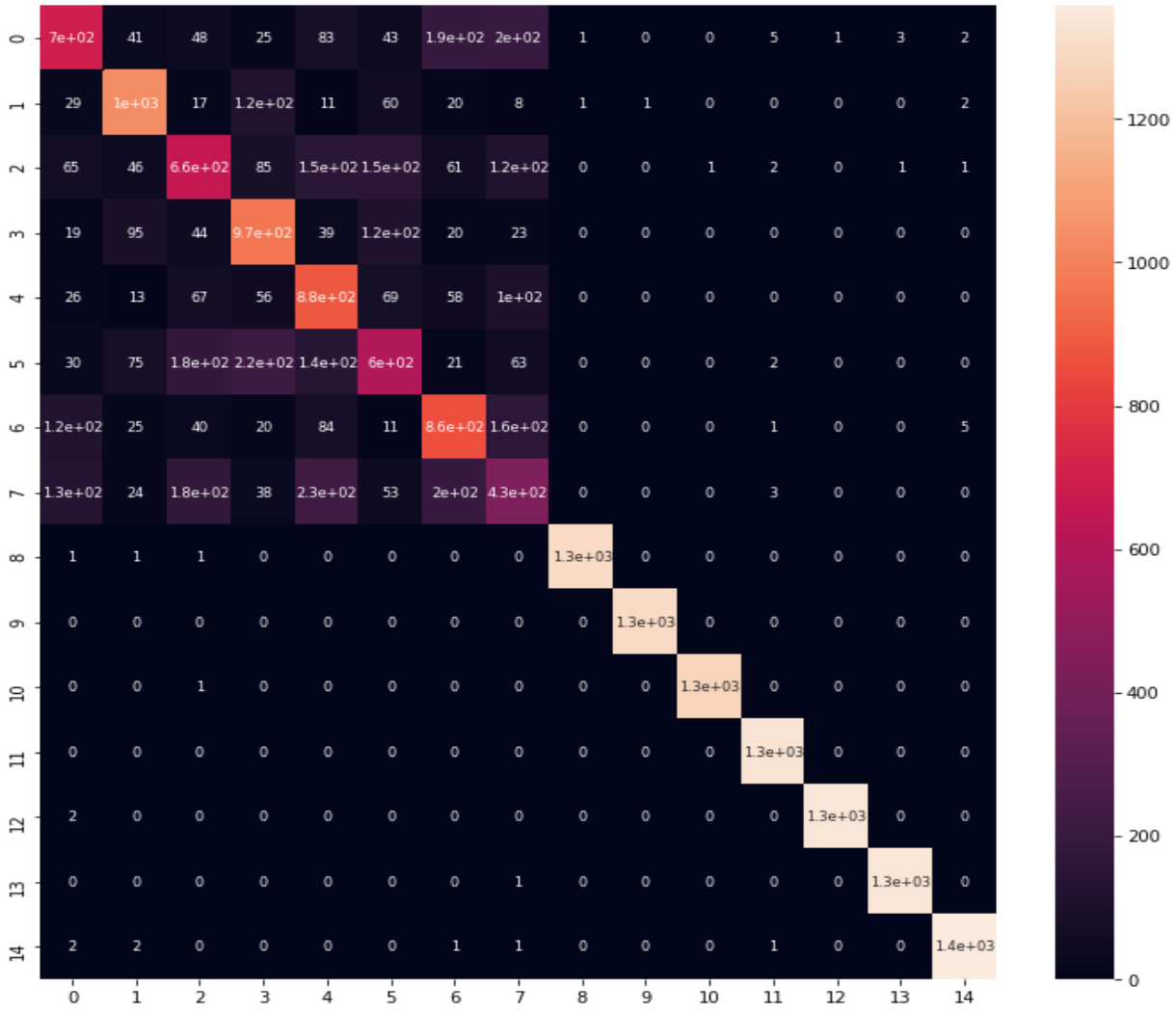


- Ce graphique représente l'évolution des courbes de test et de train selon le *Kernel* testé et en fonction de la *variable de régularisation C*.

- Nous pouvons observer que le choix du kernel a un grand impact sur la classification du modèle, ici le rbf (modèle par défaut) est le meilleur.
- De plus, nous constatons que plus la variable C augmente, moins elle agit sur le test_set et nous pouvons donc observer un overfitting sûr pour C >10.
- Nous avons finis par ne pas agir sur le *nombre d'itération* car cela engendre de moins bons résultats

- En ayant fixé une marge de 0.05 sur la différence de scores du train et du test, nous arrivons à la conclusion que notre meilleur modèle est atteint avec : *kernel = 'rbf' et C= 6*
- Il en résulte donc que le meilleur score que nous avons obtenus est de:
  0.80965 sur **l'ensemble d'entraînement** et
  0.77632 sur **l'ensemble de test**

Matrice de confusion



- Nous pouvons observer la matrice de confusion des 15 genres que nous avons classifiés.
- Tout d'abord, nous pouvons interpréter certains résultats comme dans la classe 0 où nous avons sur environ 1200 données nous avons classifié 700 de vrais positifs, et 120 vrais négatifs de la classe 6.
- De plus, nous remarquons qu'à partir de la classe 8 notre modèle a très bien classifié les données. Mais, pour les 8 classes précédentes nous n'avons pas d'aussi bons résultats.
- Nous pouvons l'expliquer en nous rapportant à nos données. En effet, lors de la préparation de nos données nous avons effectué du *word embedding* sur les titres et song_names, qui étaient respectivement renseignés sur les 8 premières et 7 dernières classes.
- Grâce à une vérification nous avons compris que sur la majorité des *titres* le *genre* était renseigné (classes 0 à 7), contrairement à *song_name (classes 8 à 14)*. Ce qui expliquerait donc que la classification ait été meilleure sur les 7 dernières classes.

b. *Multi Layer Perceptron (MLP):*

- Comment?

Nous avons utilisé la MLP de la bibliothèque sklearn de scikit learn sur nos données déjà préparées. Vous pouvez vous reporter au code pour plus de détails.
Nous avons obtenu grâce à ce modèle un score de : `0.774533904`

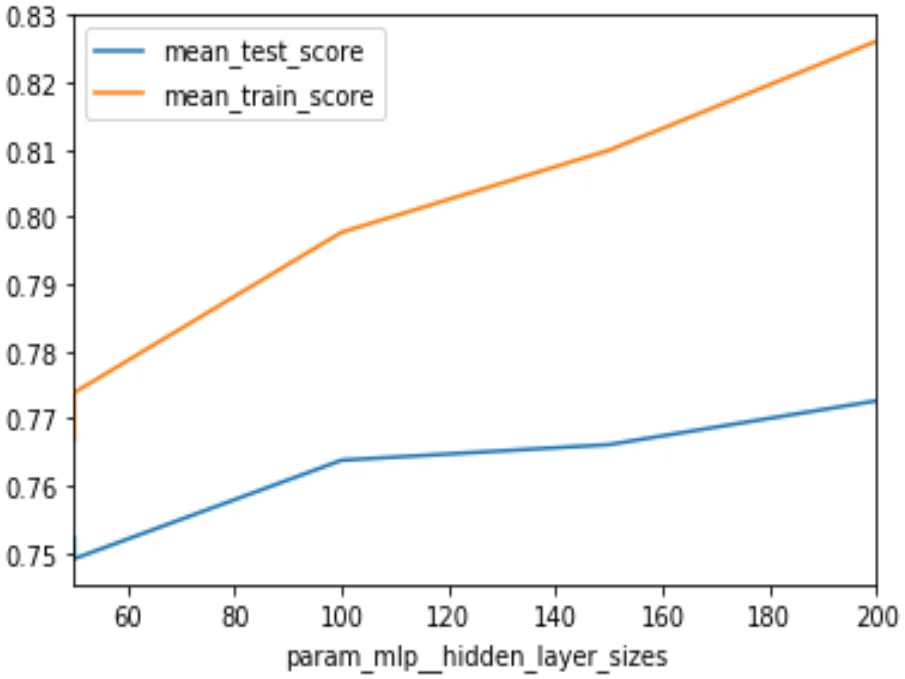
- ***What's next?***

On va procéder de la même manière que la SVM :

- ❖ Essayer d'améliorer le score en modifiant les hyperparamètres
- ❖ Utiliser GridSearch afin d'automatiser notre recherche, et pour cela on a effectué plusieurs testes pour repérer les hyperparamètres les plus intéressants et dans notre cas nous avons décidé de travailler sur:

  - ❑ **Hidden_layer_sizes**: Le ième élément représente le nombre de neurones dans la ième couche cachée.
  - ❑ **Activation:** Fonction d'activation pour la couche cachée.
  - ❑ **Learning rate init:** le taux d'apprentissage initial utilisé. Il contrôle la taille du pas dans la mise à jour des poids.

- ***Résultats:***



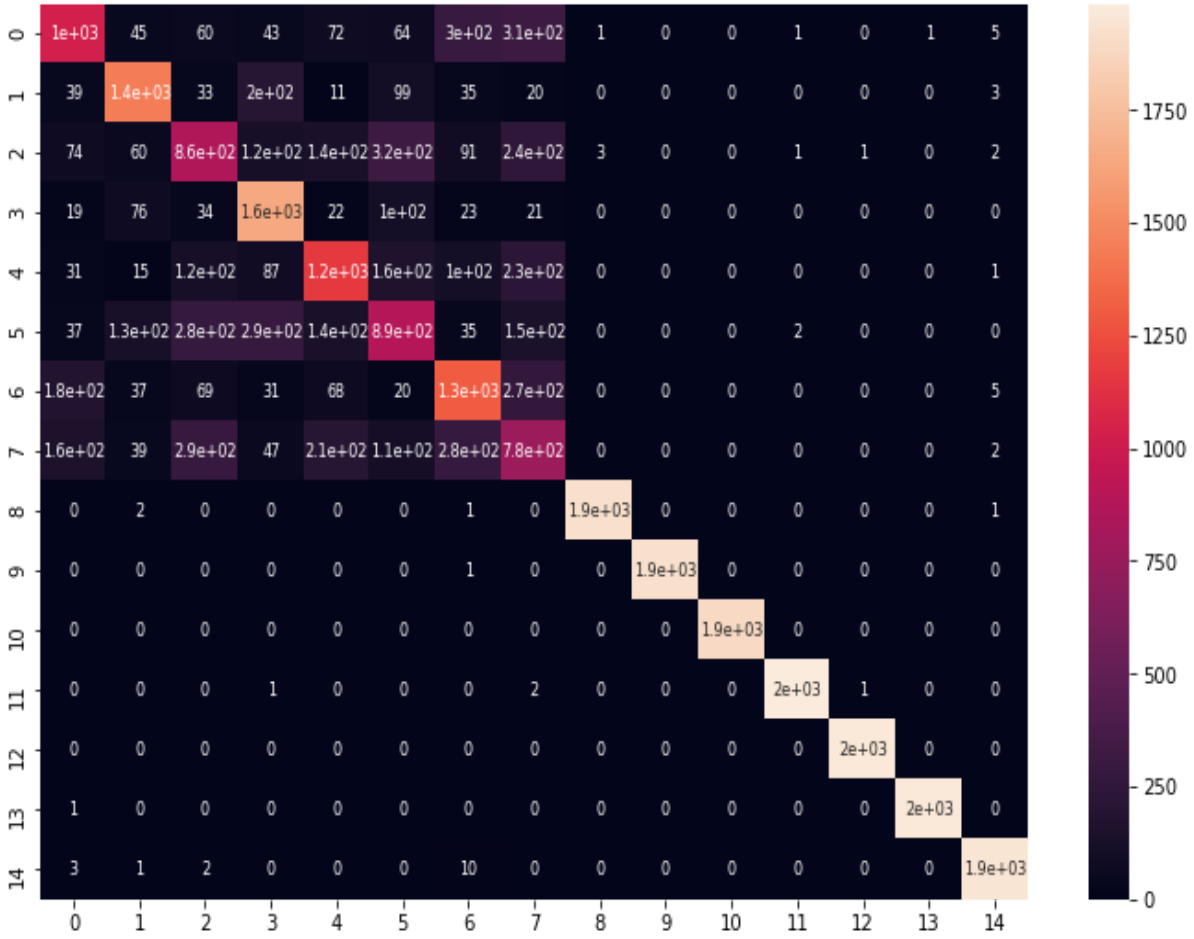
- On remarque à partir ces graphes que l'hyper paramètre **Hiden_layer_size** n'impacte pas énormément nos scores, en effet on remarque que les courbes est deux courbes sont croissante lorsqu' on augmente ce paramètre

- On a fixé une marge de 0.05 sur la différence de scores du train et du test, afin d'éviter l'**Overfitting et** mieux généraliser**.**

- On respectant cette contrainte, nous arrivons à la conclusion que notre meilleur modèle est atteint avec :
    - **Hiden_layer_size : 190**

- et avec ces hyperparametre on a obtenu les resultats suivant :
    - **score train : 0.825**
    - **score test : 0.775.**

Matrice de confusion :



- Nous pouvons observer la matrice de confusion des 15 genres que nous avons classifiés

- Nous remarquons qu'à partir de la classe 8 notre modèle a très bien classifié les données. Mais, pour les 8 classes précédentes nous n'avons pas d'aussi bons résultats.

- Nous pouvons l'expliquer en nous rapportant à nos données. En effet, lors de la préparation de nos données nous avons effectué du *word embedding* sur les titres et song_names, qui étaient respectivement renseignés sur les 8 premières et 7 dernières classes.
- Grâce à une vérification nous avons compris que sur la majorité des *titres* le *genre* était renseigné (classes 0 à 7), contrairement à *song_name (classes 8 à 14)*. Ce qui expliquerait donc que la classification ait été meilleure sur les 7 dernières classes.(même constat que SVM)

c. *Random Forest:*

- Comment?

Après avoir tester les deux algorithme SVM et MLP, notre choix c'est orienté vers le randomForest,

Random forest signifie « forêt aléatoire ». Proposé par Leo Breiman en 2001, c'est un algorithme qui **se base sur l'assemblage d'arbres de décision**. Il est assez intuitif à comprendre, rapide à entraîner et il produit des résultats généralisables.

à la fin de cette partie on vas voir les performance de cet algorithme et est ce que il se généralise bien sur nos données en faisant un contrôle de l'Overfitting

- ***What's next?***

On va procéder de la même manière que la SVM :

- ❖ Essayer d'améliorer le score en modifiant les hyperparamètres
- ❖ Utiliser la bibliothèque de GridSearch afin d'automatiser notre recherche, et pour cela on decider de faire varier les hyper paramètres suivants :

- ❑ **criterion : {"gini", "entropy"}, default="gini" :**
  La fonction pour mesurer la qualité d'un split

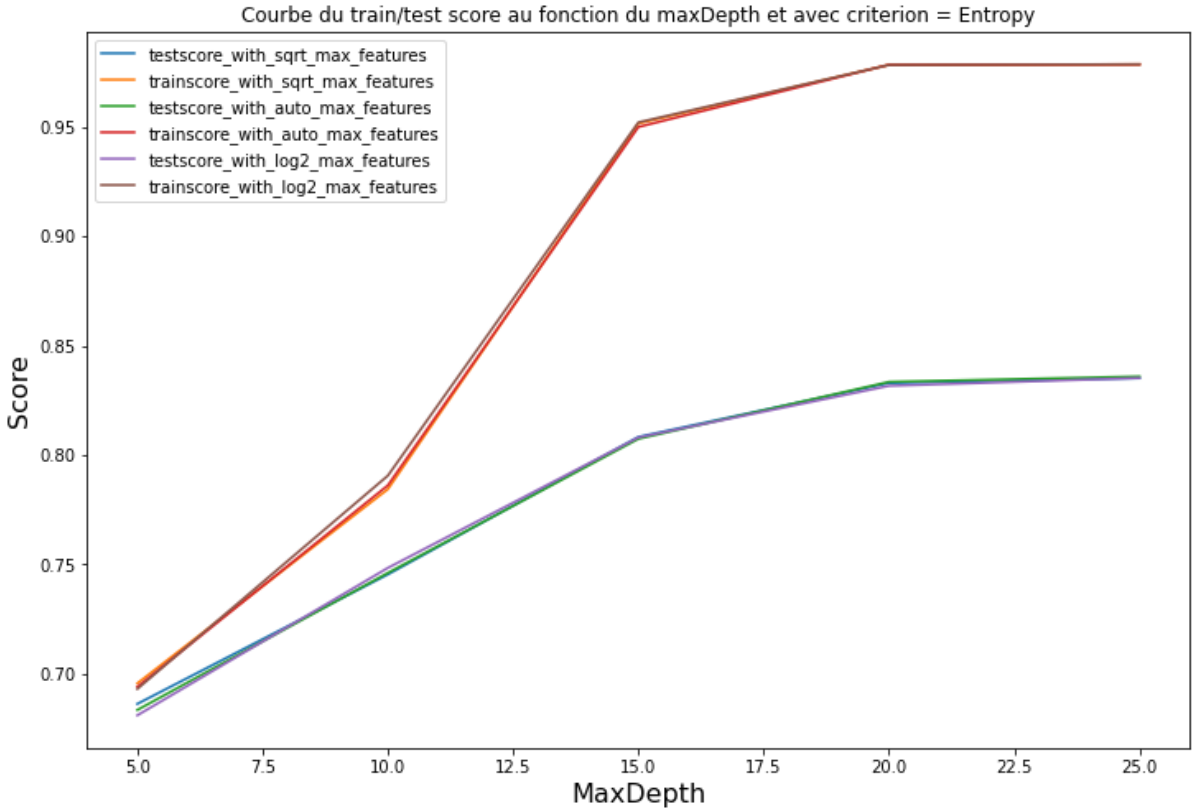
- ❑ **max_depth :** La profondeur maximale de l'arbre

- ❑ **max_features : {"auto", "sqrt", "log2"}, int or float,**
  Le nombre de features à prendre en compte lors de la recherche de la meilleure répartition

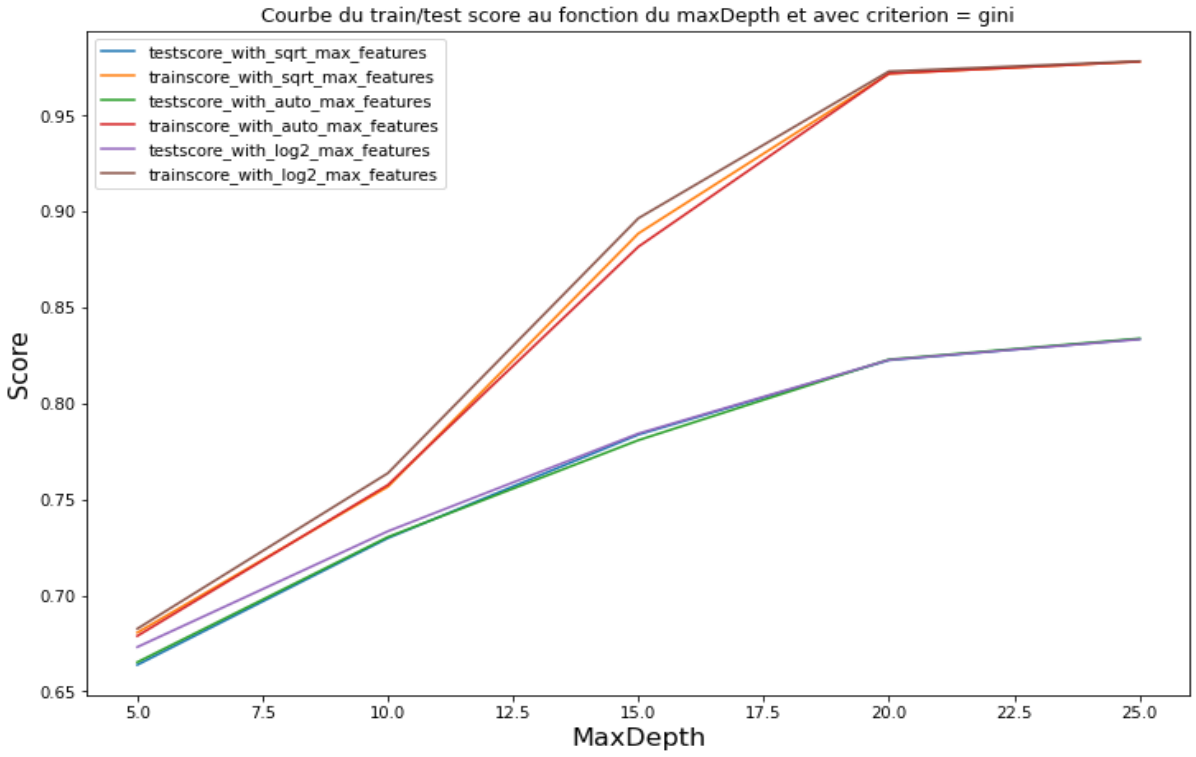
- ❑ **bootstrap : bool, default=True :**
  Indique si des échantillons de bootstrap sont utilisés lors de la création d'arbres. Si False, l'ensemble de données est utilisé pour construire chaque arbre**.**

- ***Résultats:***

➔ Avec ***Criterion*** = *“Entropy”*

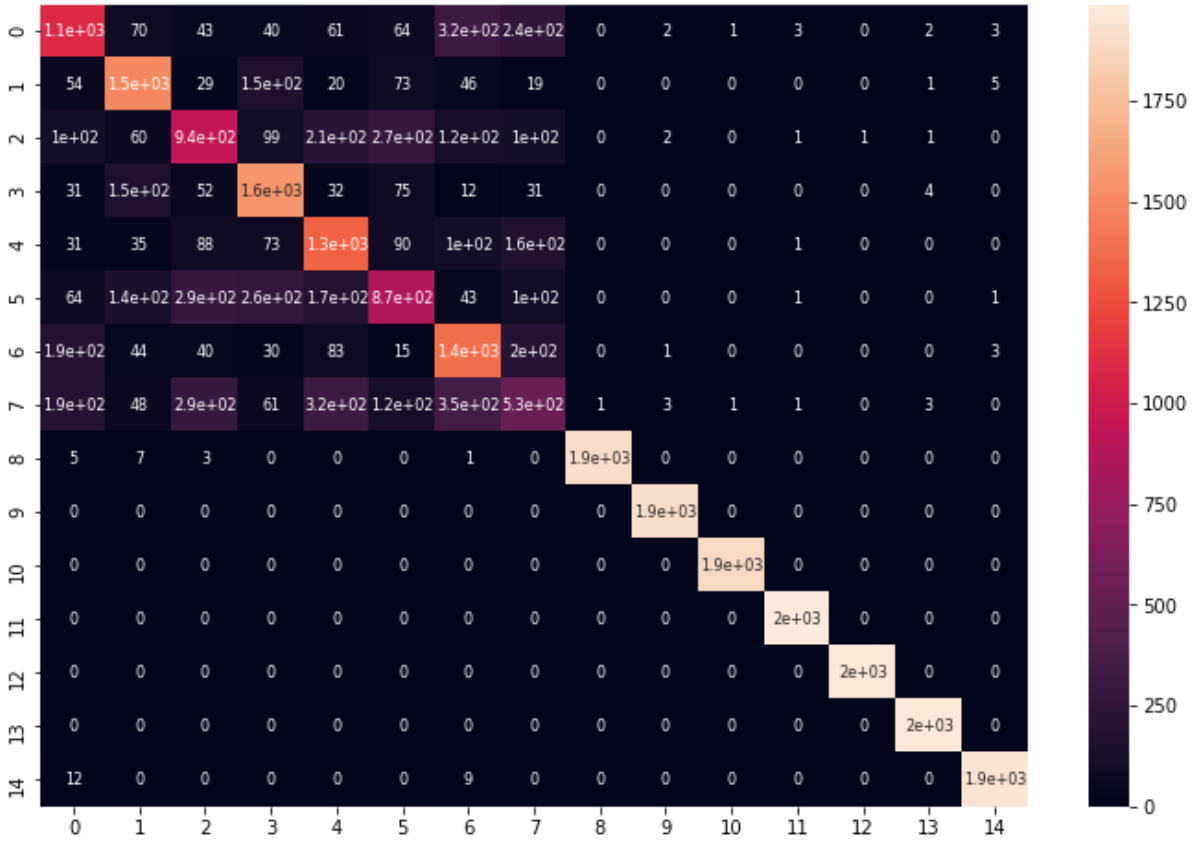


➔ Avec ***Criterion*** = *“Gini”*



- On remarque à partir ces graphes que l'hyper paramètre **max_features** n'impacte pas énormément nos scores, en effet on remarque que les courbes sont presque superposé en fonction de ce paramètre
- Le meilleur score enregistré sur l'ensemble de Test est de 0.83, chouette !!!, mais malheureusement en regardant le score sur l'ensemble de train, on constate que le score est de 0.97, c'est de l'**overfitting** !!!
- On a fixé une marge de 0.05 sur la différence de scores du train et du test, afin d'éviter l'**Overfitting et** mieux généraliser**.**
- On respectant cette contrainte, nous arrivons à la conclusion que notre meilleur modèle est atteint avec :
    - **criterion : Entropy**
    - **max_depth : 13**
    - **bootstrap = False**
- et avec ces hyperparametre on a obtenu les resultats suivant :
    - **score train : 0.84**
    - **score test : 0.789**

➔ **Matrice de Confusion du modèle Choisi**



- le même constat que la SVM s'applique sur la matrice de confusion générée par le modèle du random forest

d. Comparaison des Models :

Afin d'interpréter au mieux nos résultats nous avons décidé de les comparer entre eux, en terme de performances ainsi qu'à l'aide des visualisations que nous allons engendrer

Best Model of	SVM	MLP	RandomForest
Train Score	**0.80965**	**0.825**	**0.842**
Test Score	**0.77632**	**0.775**	**0.789**

- On remarque que le meilleur modèle est RandomForest avec un score de **0.789**
- Les scores des modèles sont très proches
- D'après les matrices de confusion on remarque que les 3 algorithme arrive à bien classifier les 8 dernière classes, et classifier moins bien les 7 premières

D. Conclusion :

- Pour conclure, on peut dire que ce projet nous a permis de mettre en pratique les concepts vus en cours et d'apprendre beaucoup de chose notamment sur la préparation des données et l'entraînement des modèles à savoir :
    - Le rééquilibrage des données
    - Exploitation des données textuelles
    - Recherche des meilleurs Hyper paramètres
    - Contrôle de l'overfitting
    - le Travail en équipe

- Le score obtenu par notre modèle est nettement supérieur au résultats qu'on a pu trouver sur le lien de la compétition sur kaggle concernant ce projet, en effet le meilleurs score de la compétition est de **0.726**, et notre modèle affiche un score de **0.79**.

- Et aussi pour ne pas oublié, on tient à remercier notre enseignant monsieur **Giancarlo Fissore**, pour tout ses efforts, malgré le context difficile du déroulement des cours lié à la crise sanitaire.