DataScientest

Projet prédictions et recommandations musicales - Rapport final

Equipe SpotiPy

- Maxime CERISIER
- Caolan GUEGUEN
- Louis LEVEAUX

Bootcamp octobre 2021 - Cursus Data Scientist

Sommaire

1. Introduction

Pour notre projet “fil rouge” nous avons opté pour un projet de recommandation musicale.

L’objectif de celui-ci est double :

- Prédire si une musique sera plus ou moins appréciée.
- Développer un algorithme de recommandation musicale.

Pour réaliser cela nous disposons de trois datasets :

- ***context_content_features.csv*** : contenant les caractéristiques audios de chaque titre. Ce jeu de données compte plus de 11 millions d’entrées.
- ***sentiment_values.csv*** : contenant des hashtags provenant de Twitter et les notes sentimentales pour chacun d’eux. Ces notes ont été calculées avec quatre algorithmes différents : AFINN, Opinion Lexicon, Sentistrength Lexicon et Vader. On possède ces notes pour un peu plus de 5000 hashtags.
- ***user_track_hashtag_timestamp.csv*** : contenant des musiques et leurs hashtags associés. Ce troisième datasets compte plus de 17 millions d’entrées.

Le détail de nos données est présent ci-dessous :

#	Column	Dtype
0	coordinates	object
1	instrumentalness	float64
2	liveness	float64
3	speechiness	float64
4	danceability	float64
5	valence	float64
6	loudness	float64
7	tempo	float64
8	acousticness	float64
9	energy	float64
10	mode	float64
11	key	float64
12	artist_id	object
13	place	object
14	geo	object
15	tweet_lang	object
16	track_id	object
17	created_at	object
18	lang	object
19	time_zone	object
20	user_id	float64
21	id	int64

context_content_features.csv

#	Column	Dtype
0	hashtag	object
1	vader_min	float64
2	vader_max	float64
3	vader_sum	float64
4	vader_avg	float64
5	afinn_min	float64
6	afinn_max	float64
7	afinn_sum	float64
8	afinn_avg	float64
9	ol_min	float64
10	ol_max	float64
11	ol_sum	float64
12	ol_avg	float64
13	ss_min	float64
14	ss_max	float64
15	ss_sum	float64
16	ss_avg	float64
17	In1	float64
18	In2	float64
19	In3	float64
20	In4	float64

sentiment_values.csv

#	Column	Dtype
0	user_id	int64
1	track_id	object
2	hashtag	object
3	created_at	object

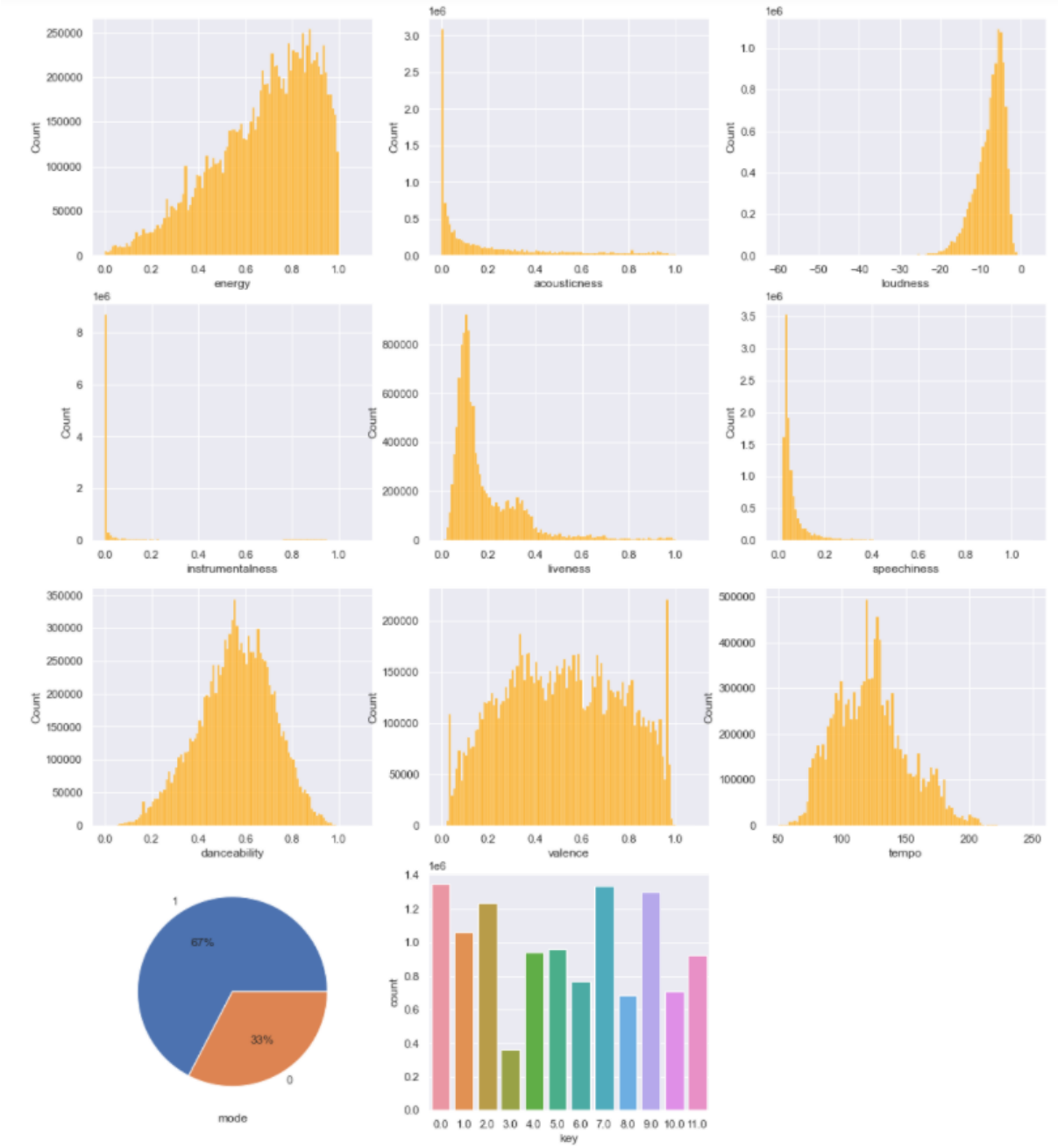
user_track_hashtag_timestamp.csv

2. Visualisation des données

2.1. Dataset n°1

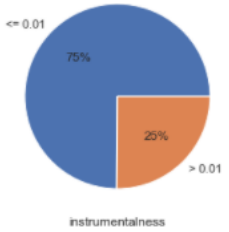
Dans un premier temps, nous avons affiché la distribution de chacunes des valeurs numériques à savoir : “instrumentalness”, “liveness”, “speechiness”, “danceability”, “valence”, “loudness”, “tempo”, “accousticness”, energy”, “mode” et “key”.

On obtient les graphiques suivants :

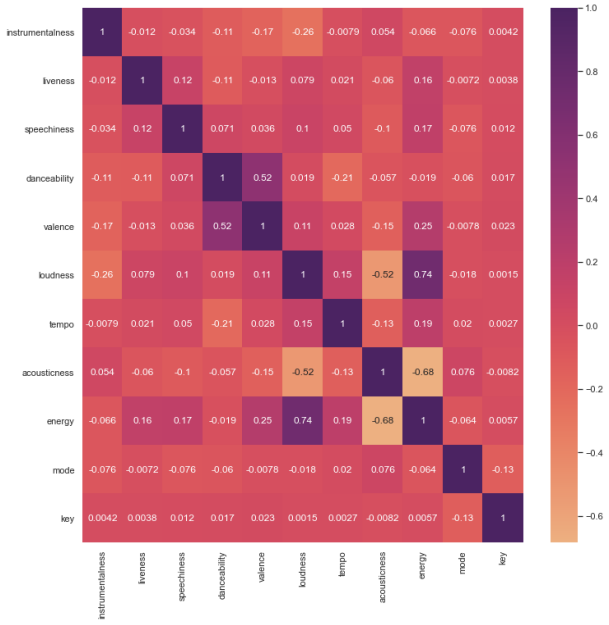


Cela nous a permis de remarquer quelques tendances. En effet, sur les 11 millions de musique du dataset, on remarque que la plupart ont :

- une énergie élevée (>0.6)
- une acoustique faible (<0.1)
- une forte intensité(>-10)
- une instrumentalité extrêmement faible (on remarque effectivement que l'immense majorité des musiques se situe à 0 ou très proche de 0, voir camembert ci-contre).
- une “liveness” faible (<0.2)
- une “speechiness” très faible (<0.1)
- une note de danse entre 0.4 et 0.7
- un tempo compris entre 100 et 150 bpm
- un mode qui vaut 1
- les répartitions des variables “valence” et “key” sont, quant à elles, assez homogènes.

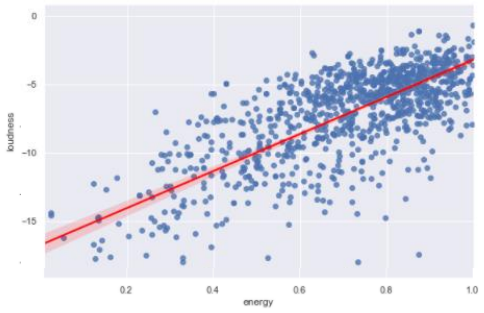


Nous avons ensuite tenu à visualiser la matrice de corrélation du jeu de données afin d’étudier les liens entre les différentes colonnes :



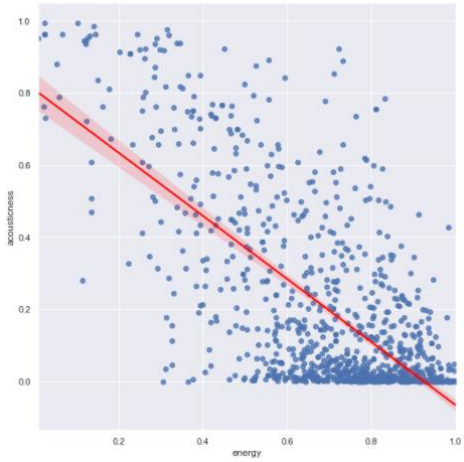
La grande majorité des données ne sont pas corrélées, en revanche on constate tout de même que :

- Les colonnes “energy” et “loudness” ont tendance à être proportionnelles. Cette tendance est confirmée par le graphique ci-dessous :



**graph tracé sur un échantillon de donnée réduit -> 1000 valeurs*

- Les colonnes “energy” et “accousticness” ont tendance à être inversement proportionnelles. Cette tendance est moins nette mais tout de même visible :

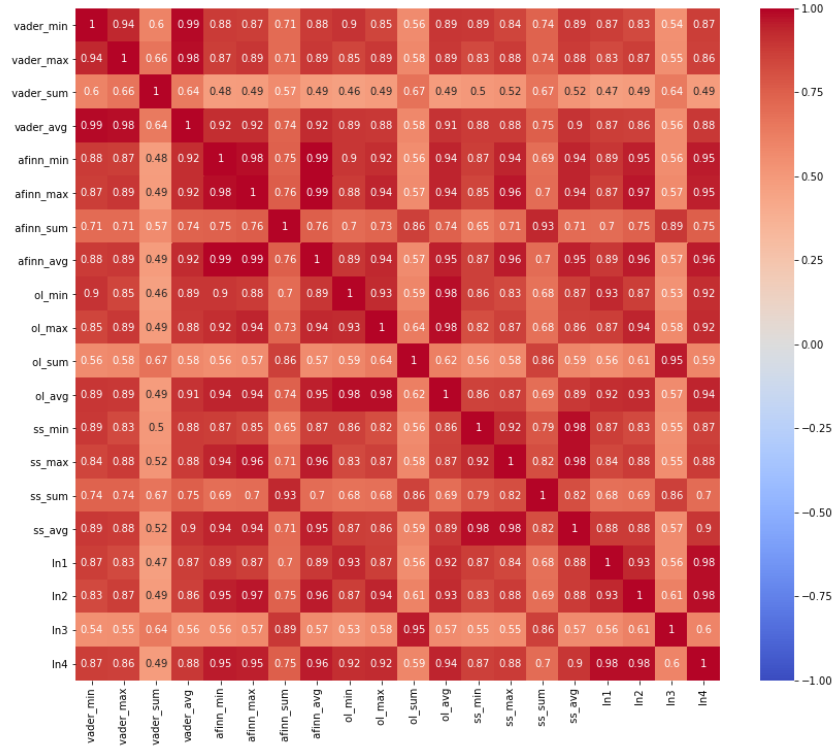


**graph tracé sur un échantillon de donnée réduit -> 1000 valeurs*

2.2. Dataset n°2

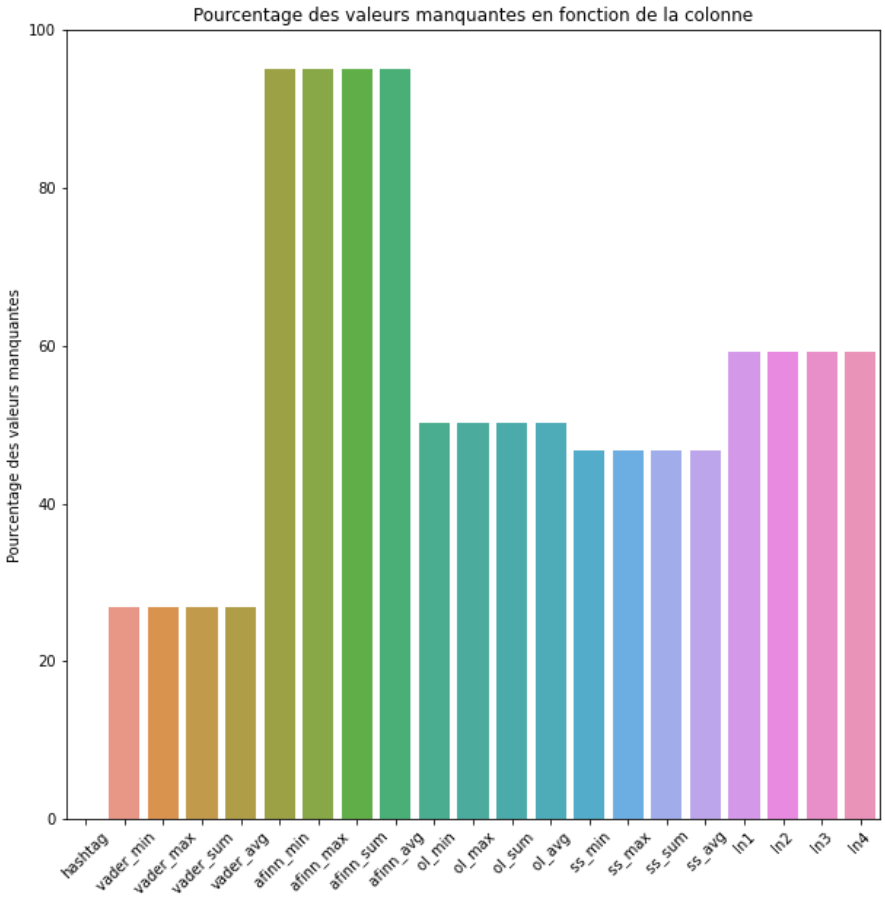
Le deuxième dataset est un jeu de données qui représente les données obtenues avec un des algorithmes d’analyse de sentiments dont “sentimental analysis using vader” pour différents hashtags. Dans ce jeu de données, nous avons rencontré un problème lors de l’import des données car dans le fichier csv il manque le nom de quatre colonnes, nous les appellerons pour l'instant In1, In2, In3, In4 (In pour inconnus).

Pour ce jeu de données nous avons représenté la matrice de corrélation afin d’étudier les liens entre les colonnes :



Les différentes méthodes d’estimation des sentiments sont fortement corrélées, ce qui était attendu car elles traduisent toutes la même chose.

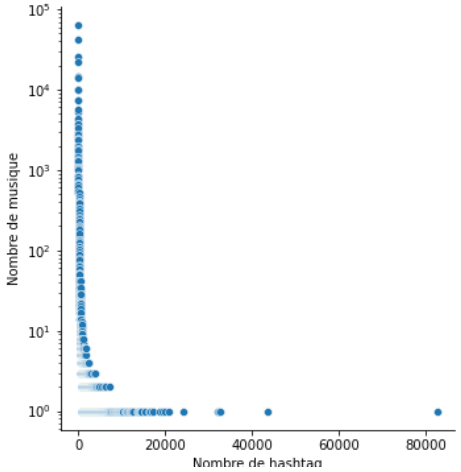
Nous avons aussi pu constater que ce jeu de données contient beaucoup de valeurs manquantes :



Il faudra donc faire attention lors du traitement de données pour pouvoir utiliser correctement ces données.

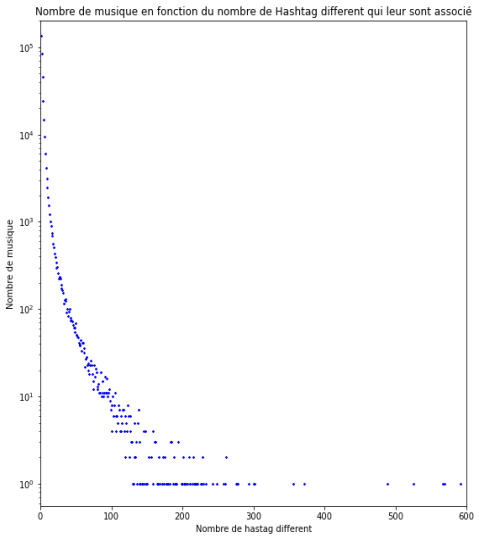
2.3. Dataset n°3

Le troisième dataset répertorie les hashtags (obtenus à partir de Twitter) qui ont été associés à nos musiques. Nous avons d'abord visualisé le nombre de hashtags associé à chaque musique :



Sur le graphique précédent, on remarque que la plupart des musiques de ce jeu de données ont peu de hashtags pour les définir. Lors du traitement, il faudra donc se poser la question de la pertinence de garder ces musiques car moins il y a d'avis et plus on augmente le risque de mal catégoriser le titre.

Nous avons également voulu observer dans ce jeu de donnés le nombre de musique en fonction de leur nombre de hashtags différents :



Nous pouvons voir que beaucoup de musiques ont peu de hashtags différents pour les définir (la grande majorité en a moins de 50). Cela peut être dû au fait que ces musiques ont très peu de hashtags associés ou au fait que l'avis des gens est unanime.

Dans la suite de ce projet il faudra se poser la question de la manière de traiter nos musiques : A-t-on assez des hashtags pour que l'avis soit fiable ? Si oui, véhiculent-ils des sentiments similaires, différents ou contradictoires ?

3. Clustering

La première idée avec ce jeu de données a été d'effectuer un clustering à l'aide des différentes caractéristiques techniques des musiques. Il y a eu dans un premier temps une étape de preprocessing, dans un second temps une étape de clustering et enfin l'utilisation de PCA afin de visualiser notre clustering graphiquement et évaluer la qualité de notre clustering.

3.1. Preprocessing des données

La première étape de preprocessing a été de choisir quelles variables conserver pour notre clustering. A l'aide de notre data visualisation nous avons pu voir que les différentes variables quantitatives étaient exploitables et peu corrélées entre elles. L'idée de toutes les utiliser dans notre cas paraît donc pertinente.

Nous avons donc conservé les variables suivantes :

- instrumentalness
- liveness
- speechiness
- danceability
- valence
- loudness
- tempo
- acousticness
- energy
- mode
- key

Nous avons ainsi supprimé les variables suivantes :

- coordinates (99% du temps égale à 0)
- place
- geo
- time zone
- artist_id
- tweet_lang
- created_at
- lang
- user_id
- id

Nous avons aussi placé le track_id en index.

La dernière étape du preprocessing consiste maintenant à centrer réduire nos données car, bien que la majorité de nos variables soient comprises entre 0 et 1 d'autres comme le tempo ou la loudness s'en éloigne beaucoup (compris entre 60 et 220 pour le tempo et -25 et -1 pour la loudness).

Nous avons à notre disposition 2 outils potentiels de type scaler venant de sklearn :

- MinMaxScaler
- StandardScaler

Dans notre étude nous avons choisis le MinMaxScaler, nous expliquerons ce choix dans la partie suivante.

3.2. Détermination du nombre de clusters

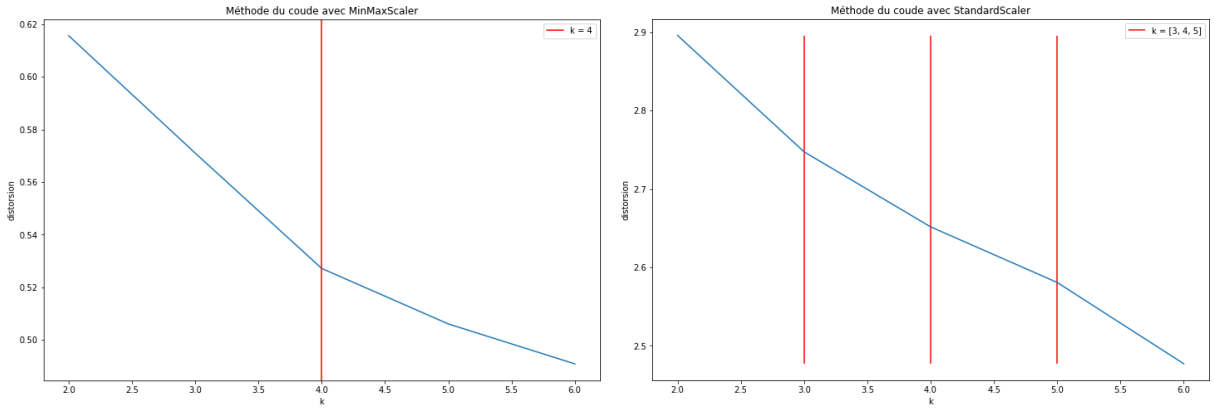
Pour effectuer un clustering nous connaissons 2 méthodes possibles :

- La méthode du KMeans
- La méthode CAH (Classification Ascendante Hiérarchique)

La méthode CAH nécessitant l'utilisation d'un dendrogramme, et ayant un jeu de données très lourd, la méthode du KMeans paraît plus judicieuse.

La première chose à faire dans la méthode du KMeans est de prévoir le nombre de clusters que nous allons paramétrer.

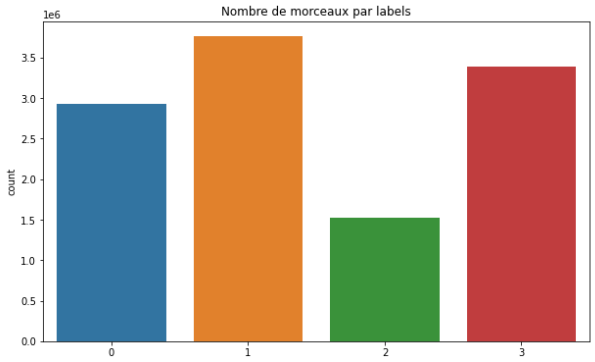
Nous utilisons la méthode du coude. Cette méthode en fonction du Scaler que nous utilisons dans notre preprocessing ne donne pas les mêmes résultats.



Avec le StandardScaler on distingue 3 coudes sur la courbe de la distorsion. Ces trois coudes sont peu marqués contrairement à l'unique coude obtenu sur la courbe de la distorsion avec la méthode du MinMaxScaler. Nous avons choisi par la suite de conserver notre jeu de données centré réduit à l'aide du MinMaxScaler et le nombre de clusters égal à 4 pour le paramétrage de notre clustering.

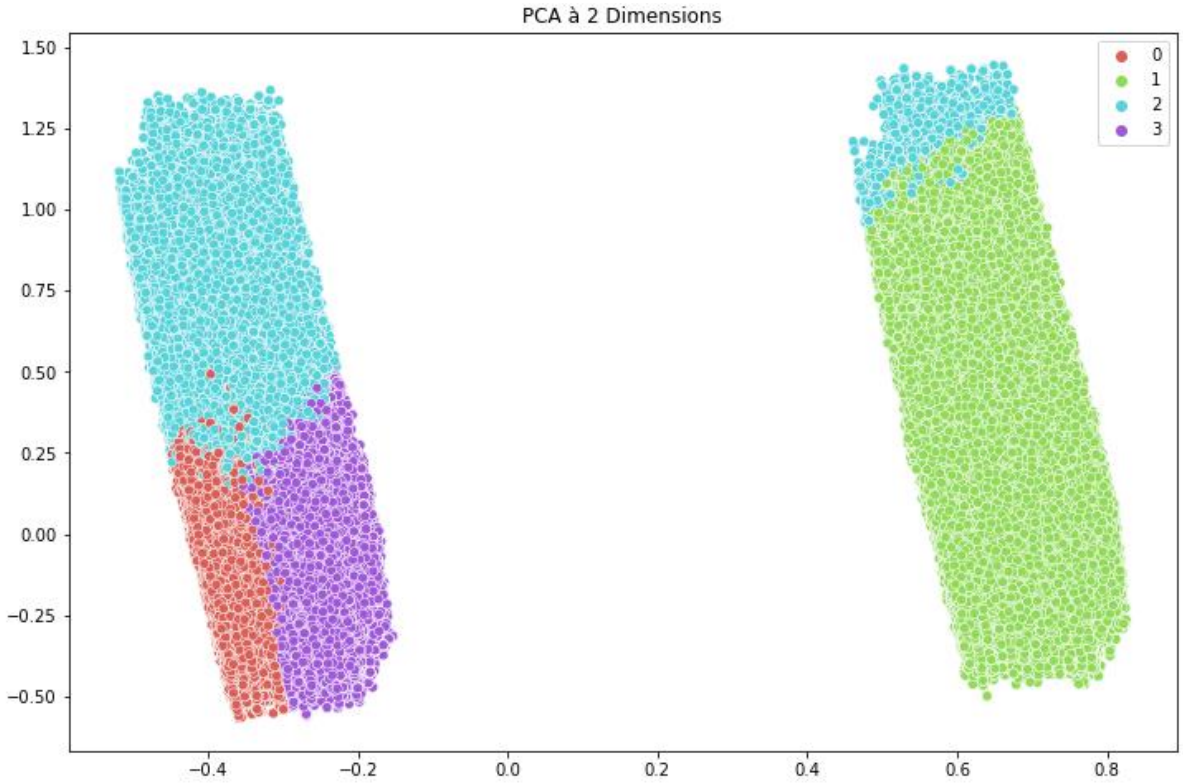
3.3. Evaluation du modèle

Une fois notre clustering effectué il faut maintenant essayer de l'évaluer pour voir s'il est pertinent. Le premier réflexe a été de voir la distribution dans les différents clusters afin de voir si nous avons quatre groupes répartis de manière équitable.



Nos morceaux sont relativement bien répartis dans les différents clusters (nombre de morceaux par clusters allant de 1,5 à 3,6 millions).

On vérifie à l'aide d'une PCA que nos clusters forment quatre groupes bien distincts. Nous choisissons une PCA à 2 composantes afin de représenter nos clusters dans 2 dimensions issues de nos 11 variables quantitatives.



On remarque des démarcations assez claires entre les quatre clusters, cela indique que notre clustering a distingué 4 clusters de façon précise. Cependant le cluster 2 (en bleu sur la figure) semble être séparé en deux parties distinctes, ce qui ne paraît pas logique. Cette interprétation reste discutable car les 2 composantes de notre PCA (abscisse et ordonnée de la figure) ont respectivement un ratio de variance de 35% et 17%. Notre PCA "explique" donc notre jeu de données qu'à une hauteur de 52% ce qui est très faible.

La dernière évaluation aurait été de faire quelques tests subjectifs pour voir quelles musiques notre clusterings a associées. Nous verrons dans la prochaine partie pourquoi nous n'avons pas pu effectuer ces tests, et en quoi cela pose un sérieux problème pour la suite du projet.

4. Problèmes

Lors du traitement des données nous nous sommes rendu compte de différentes difficultés rendant impossible la poursuite du projet avec ces jeux de données.

4.1. Jeux de données utilisés

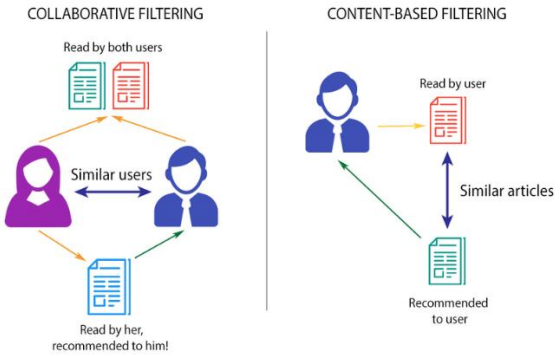
Le premier problème rencontré était que nous ne pouvions pas savoir à quelle musique correspondait chaque ligne du jeu de donnée. En effet nous ne disposions ni du titre de la musique, ni de l'album, ni même de l'artiste ; et nous n'avons pas pu utiliser l'id fourni pour retrouver ces informations car nous n'avons pas réussi à savoir comment l'utiliser. Nous avons d'abord pensé que comme le jeu de données venait de Spotify, cet id était celui de Spotify mais il ne correspondait pas (et cela malgré nos tentatives de convertir l'id fourni dans différentes bases mathématiques). Ainsi, sans titres exploitables, il nous a été impossible de conclure quant à la judiciosité du choix de nos clusters.

Le second problème venait des autres jeux de données qui rendaient la réalisation d'un système de recommandation impossible. En effet, pour donner l'avis des utilisateurs sur une musique, nous disposons de : l'utilisateur, la musique, un hashtag et une note de "sentiment". Dans les faits, avec ceci, nous n'avons pas pu obtenir des notes fiables pour les utilisateurs car les notes de "sentiments" pour chaque hashtag ont été obtenues à partir de très peu de tweets analysés. Cela remet donc en cause la justesse de cette note. De plus, certains hashtags sont neutres comme "nowplaying" ou "rock" donc cette analyse ne semble pas pertinente car le tweet associé à ce hashtag peut très bien exprimer un avis positif comme négatif et ainsi fausser les résultats. Il aurait sans doute été plus judicieux d'étudier directement les tweets plutôt que les hashtags. Enfin, la majorité des hashtags associés aux musiques sont les mêmes, on a notamment plus de 95% des hashtags qui sont des "nowplaying" (qui ne nous apprennent rien sur le sentiment des utilisateurs), et la majorité des hashtag restants peuvent encore être regroupés dans un groupe de 20 hashtags différents.

Quand bien même nous aurions eu les titres des musiques, nous ne pouvions pas avoir des notes fiables pour les utilisateurs, rendant impossible la mise en œuvre d'un système de recommandation correct.

4.2. Résolution attendue

Au vu des données fournies, nous pouvons tout de même présumer du système de recommandation attendu. C'est un système de filtrage collaboratif (voir schéma ci-dessous).



Ce genre de système de recommandation utilise les opinions et évaluations d'un ensemble d'individu sur un ensemble d'objet pour recommander à un autre l'individu certain de ces objets.

Dans notre cas, si les données nous l'avaient permis nous aurions pu créer un système pouvant prédire la note que donnerait un individu à une musique qu'il n'a pas écouté en fonction des avis des autres utilisateurs.

Pour mettre en œuvre ce système dans ce cas, il y a plusieurs étapes à suivre :

- Premièrement on normalise les notes de chaque utilisateur par rapport à sa moyenne sur toute les notes :

$$dev(i,j) = r(i,j) - \overline{r_i}$$

$\overline{r_i}$: *la moyenne de toute les notes de l'utilisateur i*

$r(i,j)$: *la note de l'utilisateur i pour l'objet j*

$dev(i,j)$: *La deviation par rapport à la note moyenne de l'utilisateur i pour l'objet j*

Cette normalisation permet de corriger le biais entre des notations dite "sévère" et "gentille". Elle permet également de savoir si l'utilisateur aime ou n'aime pas un objet pas rapport à son écart à sa moyenne.

- Ensuite on calcule les poids entre notre utilisateur i et les autre utilisateur i' :

$$w_{ii'} = \frac{\sum_{j \in \Omega_{ii'}} (r(i,j) - \overline{r_i})(r(i',j) - \overline{r_{i'}})}{\sqrt{\sum_{j \in \Omega_{ii'}} (r(i,j) - \overline{r_i})^2} \sqrt{\sum_{j \in \Omega_{ii'}} (r(i',j) - \overline{r_{i'}})^2}}$$

$w_{ii'}$: *poids entre l'utilisateur i et i' (le poids représente la similitude entre deux utilisateurs, plus ils sont semblables en notation plus le poids tend vers 1, plus ils sont différend plus le poids tends vers -1).*

$\Omega_{ii'}$: *ensemble des objets notées par l'utilisateur i et i'*

- On peut ensuite déterminer la déviation estimée de la note de l'utilisateur i pour l'objet j, puis sa note :

$$\widehat{dev}(i,j) = \frac{\sum_{i' \in \Omega_j} w_{ii'}(r(i',j) - \overline{r_{i'}})}{\sum_{i' \in \Omega_j} |w_{ii'}|}$$

$$r(i,j) = \overline{r_i} + \widehat{dev}(i,j)$$

$$r(i,j) = \overline{r_i} + \frac{\sum_{i' \in \Omega_j} w_{ii'}(r(i',j) - \overline{r_{i'}})}{\sum_{i' \in \Omega_j} |w_{ii'}|}$$

Ω_j: *Ensemble des utilisateurs ayant noté l'objet j*

Pour mettre en œuvre cette méthode il faut prendre plusieurs points en compte :

- Pour le calcul de la note de l'utilisateur i par rapport à d'autres utilisateurs, il faut que ces derniers aient plusieurs notes en commun avec i, de sorte que le poids soit le plus correct possible (et non une valeur aberrante), le nombre minimum de note en commun dépend du type de données.
- Pour réduire le temps de calcul de la note il vaut mieux sélectionner les utilisateurs ayant les poids les plus importants, donc les utilisateurs étant le plus proche de notre utilisateur i. Pour les déterminer de manière efficace et rapide dans un grand jeu de données, il existe plusieurs solutions utilisant des méthodes de factorisation de matrices comme la SVD (Singular Value Decomposition). Elles permettent de déterminer les utilisateurs les plus proches les uns des autres avec des méthodes de similarité cosinus.

Une autre possibilité pour prédire la note d'un utilisateur sur un objet est d'utiliser un réseau de neurones. Pour l'entraînement du modèle, nous avons en entrée les utilisateurs et les musiques, et en sortie les déviations (un exemple est fourni en annexe).

5. Changement du jeu de données

Compte tenu des problèmes évoqués ci-dessus nous avons, avec l'accord de notre mentor, pris la liberté de changer de jeu de données. Le dataset que nous avons récupéré ressemble fortement au premier dataset du projet initial. Il est composé de 23 colonnes (+ un index correspondant au track_id de Spotify) et regroupe 1.2 millions de musiques :

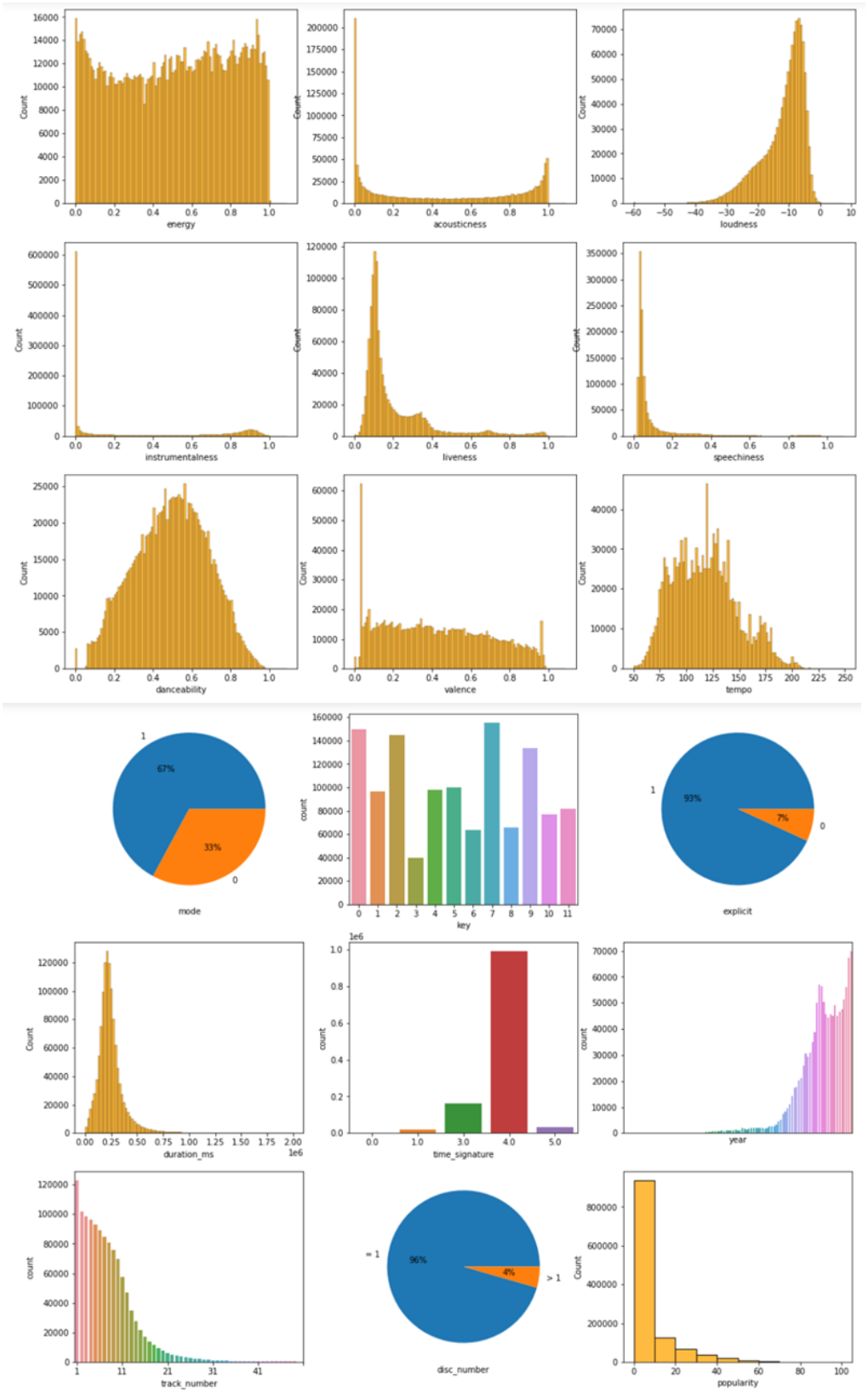
#	Column	Non-Null Count	Dtype
0	name	1204025 non-null	object
1	album	1204025 non-null	object
2	album_id	1204025 non-null	object
3	artists	1204025 non-null	object
4	artist_ids	1204025 non-null	object
5	track_number	1204025 non-null	int64
6	disc_number	1204025 non-null	int64
7	explicit	1204025 non-null	bool
8	danceability	1204025 non-null	float64
9	energy	1204025 non-null	float64
10	key	1204025 non-null	int64
11	loudness	1204025 non-null	float64
12	mode	1204025 non-null	int64
13	speechiness	1204025 non-null	float64
14	acousticness	1204025 non-null	float64
15	instrumentalness	1204025 non-null	float64
16	liveness	1204025 non-null	float64
17	valence	1204025 non-null	float64
18	tempo	1204025 non-null	float64
19	duration_ms	1204025 non-null	int64
20	time_signature	1204025 non-null	float64
21	year	1204025 non-null	int64
22	release_date	1204025 non-null	object

Les données étant, cette fois-ci, compatibles avec Spotify, nous avons pu utiliser l'API de l'application afin d'extraire des données supplémentaires. Nous avons ainsi rapporté deux nouvelles colonnes :

- *"popularity"* : qui contient une note entre 0 et 100 traduisant la popularité du titre. Les données prises en compte pour calculer cette valeur sont le nombre total d'écoutes du titre et à quel point ces écoutes sont récentes. Cette colonne nous permettra par la suite de prédire si un titre est plus ou moins apprécié.
- *"genre"* : qui contient la liste des genres associés à l'artiste ayant réalisé la musique. Cette colonne nous sera utile dans la suite du projet pour réaliser un algorithme de recommandation.

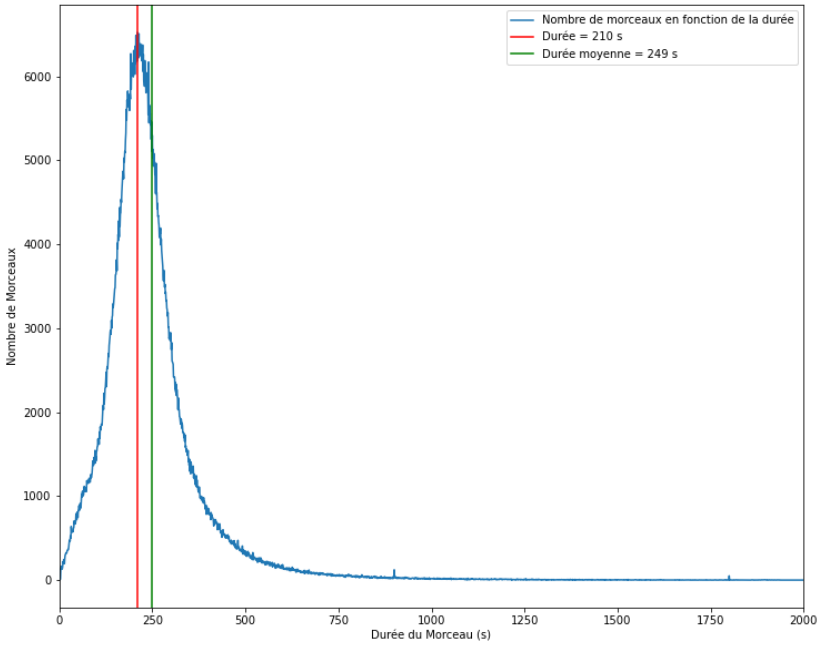
6. Visualisation du nouveau jeu de données

De la même manière qu'avec les jeux de données précédents, nous avons affiché la distribution de chacune des valeurs numériques :



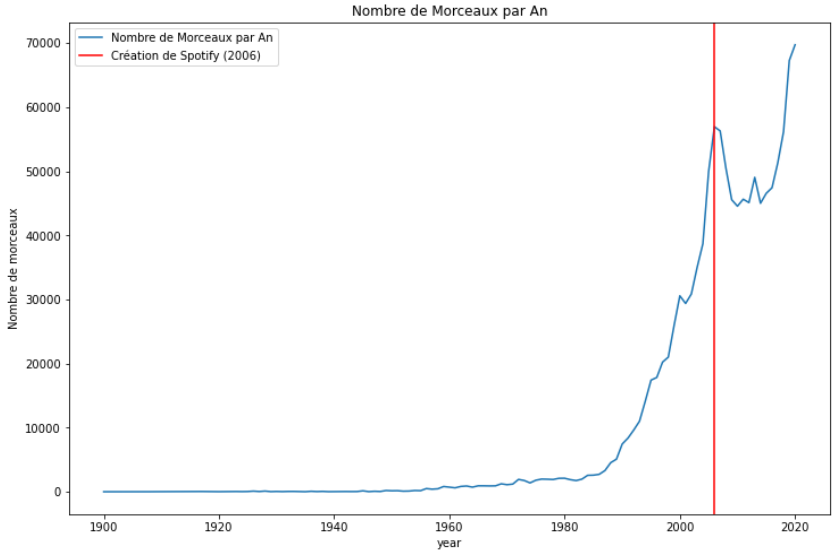
Les tendances que nous avions précédemment observées sur le premier dataset (variables allant de “energy” à “key”) sont globalement toujours valables. On note tout de même que la répartition est moins hétérogène.
Nous avons également des nouvelles variables :

- **explicit** (indicateur binaire quant à la censure) : cette variable est très déséquilibrée.
- **time_signature** (nombre de temps par mesure) : elle est également très déséquilibrée avec une majorité écrasante pour la valeur 4. Ce n’est pas étonnant car la majorité des musiques actuelles (musique pop) utilisent cette structure.
- **duration_ms** (durée de la musique en ms) :



Les durées moyennes de morceaux correspondent à des morceaux “pop”. Il est logique que la majorité des morceaux soient de ce style car il est beaucoup plus représenté que d'autres tels que le jazz, le métal ou encore le classique qui ont des durées pouvant dépasser les 10 minutes. Le durée moyenne est de 4min10 et la durée ou le maximum de morceaux se trouve est de 3min30.

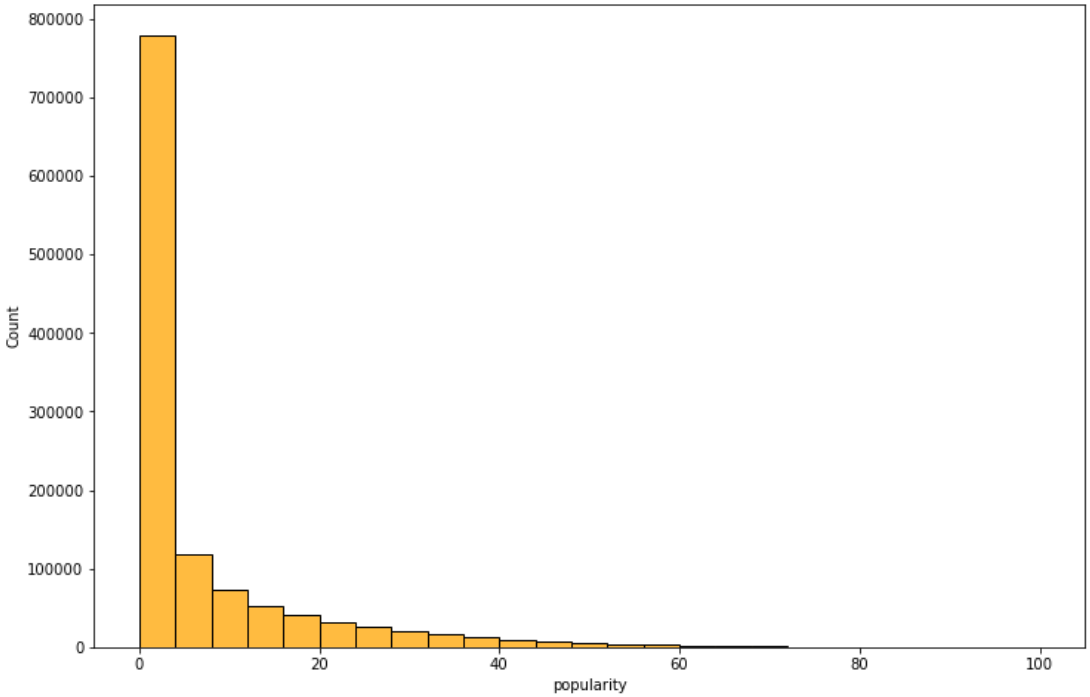
- **year** (l’année de sortie de l’album) :



on remarque que notre jeu de données est majoritairement composé de morceaux récents (moins 25 ans). On peut supposer que ce phénomène est dû à une accessibilité plus facile à la musique et donc une augmentation du nombre d'artistes et

de morceaux de musique. Cependant il peut aussi s'agir du choix qui a été fait par les personnes qui ont créé ce jeu de données. La croissance de la courbe est presque linéaire si on met à part l'année 2006 qui correspond à l'année de création de Spotify. Il est possible que certains morceaux n'ayant pas d'année de sortie aient été attribués par défaut à l'année 2006.

- **disc_number** et **track_number** (correspondant respectivement au numéro de l'album et au numéro du morceau au sein de l'album). Ces valeurs sont mal équilibrées et ne seront pas intéressantes pour la suite.
- **popularity** (note de popularité donnée par Spotify) :



cette variable est fortement représentée pour des valeurs proches de 0 et à l'inverse très mal représentée pour des valeurs élevées. Ce n'est a priori pas incohérent car à l'échelle de toutes les musiques, les morceaux très populaires ne représentent qu'une infime partie. En revanche, cela nécessitera sans doute des précautions supplémentaires pour entraîner les modèles (notamment une phase de rééquilibrage).

L'étude de la matrice de corrélation ne nous a rien appris de plus que précédemment, nous ne l'avons donc pas ré-analysée ici (la nouvelle matrice de corrélation est cependant disponible en annexe).

7. Clustering du nouveau jeu de données

Comme pour l'ancien jeu de données nous avons effectué un clustering pour tenter de regrouper nos morceaux en fonction de leurs caractéristiques musicales.

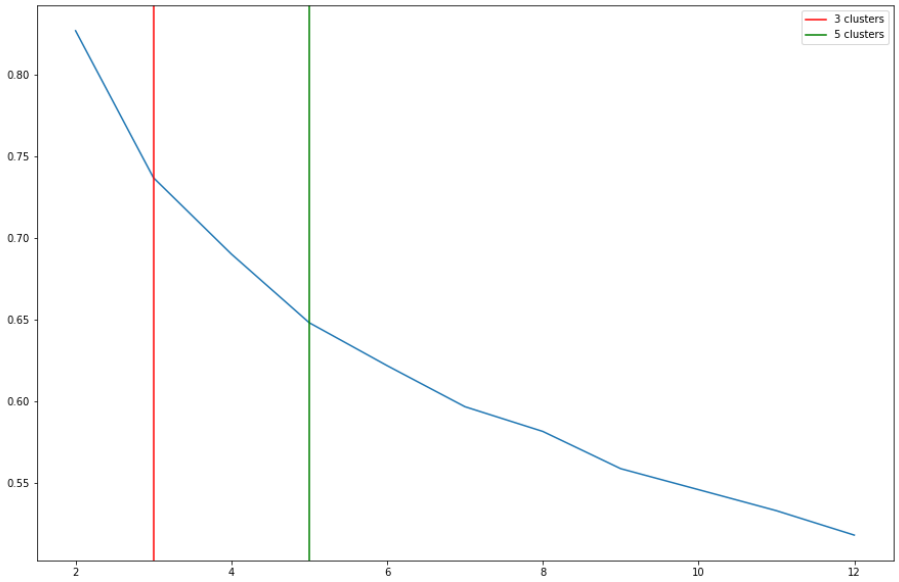
7.1. Preprocessing des données

Nous conservons les mêmes colonnes que dans le jeu de données précédents, en y ajoutant la variable 'duration_ms' du nouveau jeu de données qui renseigne sur la durée du morceau.

Nous avons à nouveau utilisé le MinMaxScaler pour centrer réduire nos données afin de considérer de la même façon chaque variable.

7.2. Clustering

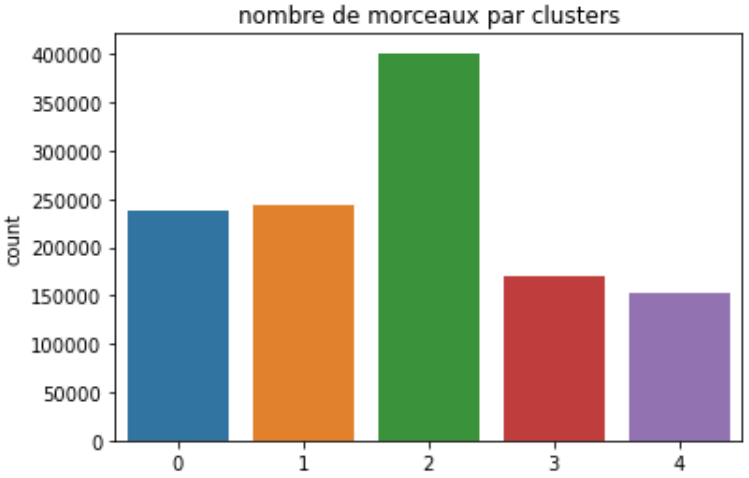
Pour la méthode de clustering nous avons utilisé celle de l'algorithme du KMeans. Puis, pour déterminer nombre de clusters à considérer, nous avons utilisé la méthode du coude.



Nous remarquons sur la courbe de la distorsion en fonction du nombre de clusters 5 coudes se former, dont les plus marqué correspondent à 3 et 5 clusters. Nous choisissons de sélectionner 5 clusters pour notre étude car 3 clusters paraissent trop peu pour regrouper des musiques semblables dans les mêmes groupes.

7.3. Evaluation du modèle (PCA et cercle de corrélation)

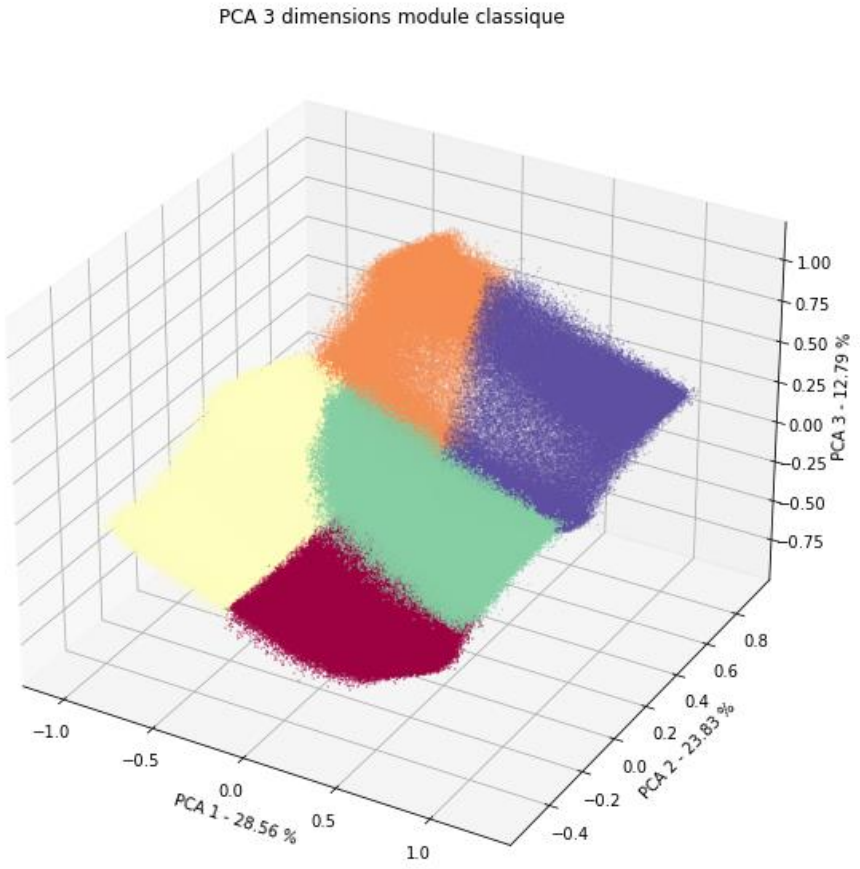
Une fois notre clustering effectué il faut maintenant essayer de l'évaluer pour voir s'il est pertinent. Le premier réflexe a été de voir la distribution dans les différents clusters afin de voir si nous avons cinq groupes répartis de manière équitable.



Nos morceaux sont relativement bien répartis dans les différents clusters (nombre de morceaux par clusters allant de 150000 à 400000).

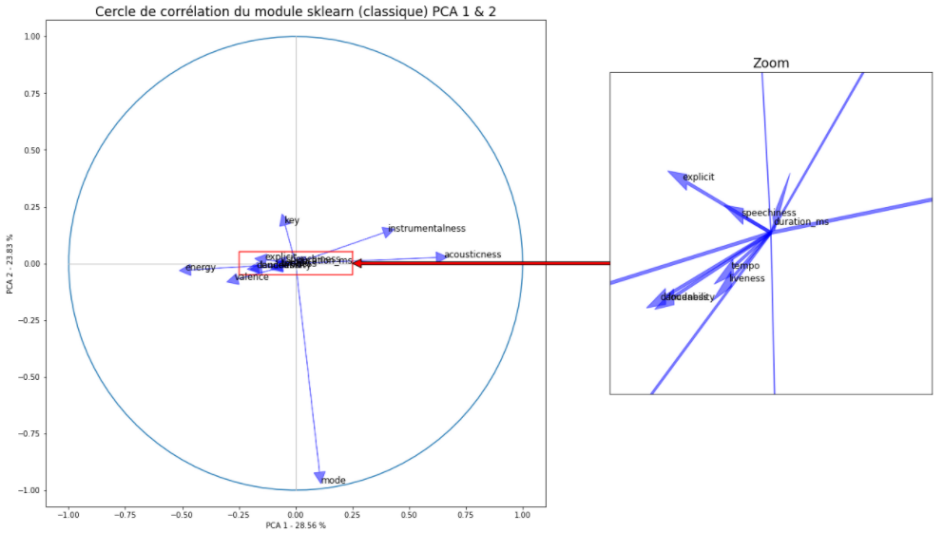
7.3.1. PCA avec scikit-learn

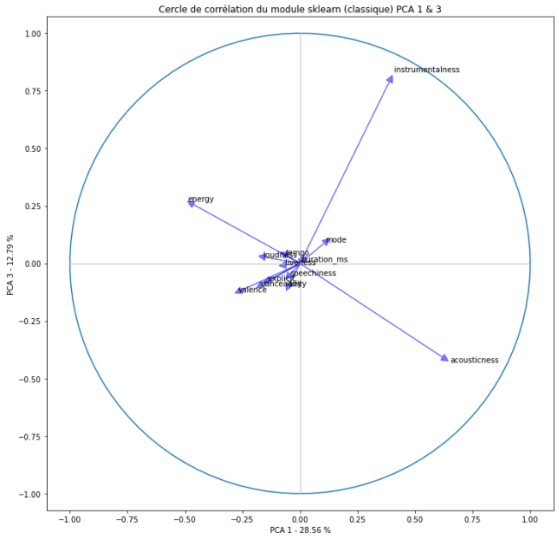
On vérifie à l'aide d'une PCA que nos clusters forment cinq groupes bien distincts. Nous choisissons une PCA à 3 composantes afin de représenter nos clusters dans 3 dimensions et ainsi minimiser la perte d'information. Nous allons utiliser deux modules de PCA (sklearn et prince) différents pour les comparer entre eux.



On remarque des démarcations assez claires entre les cinq clusters, notre cluster a donc discerné des groupes bien distincts dans notre jeu de données. On remarque cependant que les ratios de variance de chaque composante de la PCA sont assez faibles (28.56%, 23.83% et 12.79%). Si la PCA avait été performante avec 3 composantes nous aurions eu un total de ces facteurs aux alentours des 80-90%, or nous ne sommes qu'à 65%, ce qui est trop faible pour faire totalement confiance à ce modèle de réduction de dimension.

Pour compléter la PCA il est intéressant de représenter le cercle de corrélation de nos variables. Ce cercle nous permettra de savoir quelles variables du jeu de données initial composent majoritairement nos 3 composantes de PCA.





Les cercles sont très utiles pour comprendre la raison des mauvaises valeurs des ratios de notre PCA. Uniquement 4 variables sur 12 de notre jeu de départ sont représentées de manière significative dans notre PCA. Il s'agit de :

- mode (compose à lui seul PCA2)
- Energy
- Accousticness
- Instrumentalness

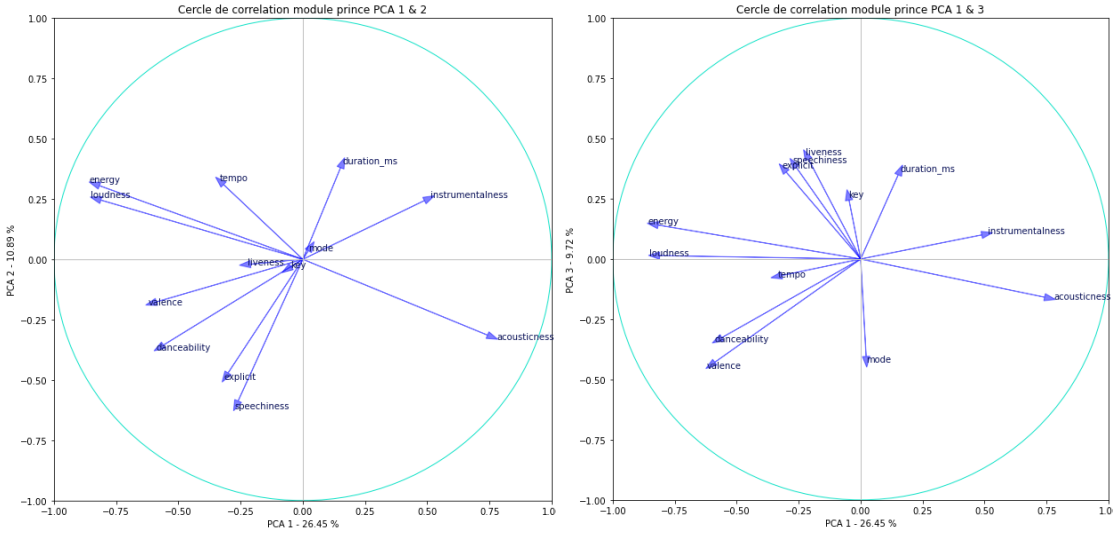
Ces résultats ne sont pas étonnant compte tenu des corrélations entre les variables que nous avions observées lors de la Data Visualisation.

On savait que nos variables étaient très peu corrélées entre elles, il est donc logique qu'une méthode de réduction de dimension soit compliquée à mettre en place.

7.3.2. PCA avec prince

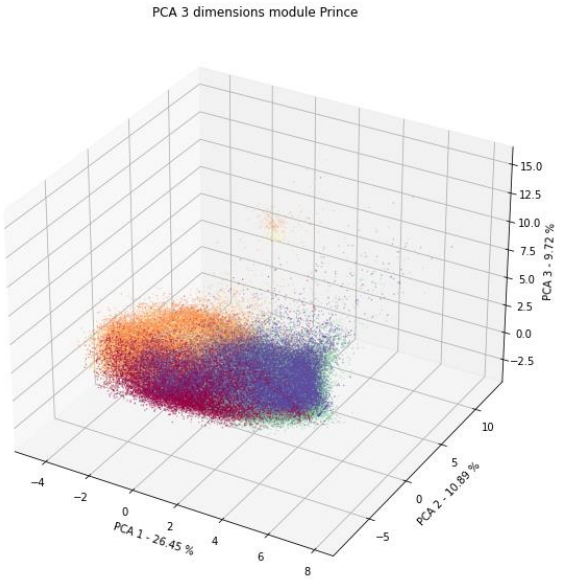
Afin de tenter trouver une alternative, nous avons utilisé le module prince pour voir et comparer ses résultats à celui du module sklearn.

Nous avons ainsi tracé les cercles de corrélation dans un premier temps et la pca dans un second temps.



A première vue on peut penser que ce module prince est bien meilleur que sklearn, car les variables de notre jeu de départ composent presque toutes de façon significative les 3 composantes de la PCA prince.

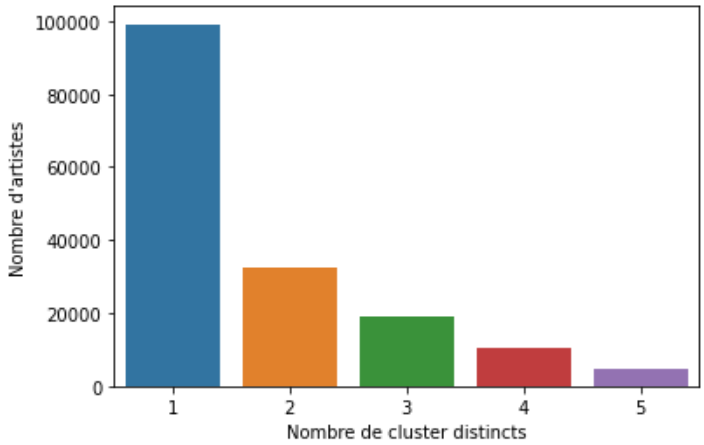
Cependant les ratios de variance sont encore plus faibles que pour l'ancienne PCA (26.46%, 10.89%, et 9.72%). Cette fois leur somme est aux alentours des 47% ce qui est extrêmement faible pour une PCA et nous permet de dire que la réduction de dimensions est compliquée à mettre en place dans le cadre de ce projet. On s'en aperçoit très bien lorsque l'on trace notre PCA :



On en déduit que le module prince est encore moins exploitable que le module sklearn. On conservera ainsi les résultats et les observations de la première PCA.

7.4. Bilan

Pour finir notre analyse sur le clustering, nous avons effectué un regroupement par artistes de notre jeu de données, en y ajoutant le nombre de clusters différents qu'il y avait par artiste. L'idée est qu'un artiste se positionne souvent dans un même registre et que bon nombre de ses musiques seront semblables. Si nous avons un nombre important d'artistes dont les morceaux se retrouvent dans un ou deux clusters différents alors nous pourrons juger que notre clustering est pertinent.



Le résultat est plutôt satisfaisant, en effet plus de 70 % de nos artistes ont leurs musiques classées dans moins de 2 clusters distincts (2 inclus).

Notre clustering est donc difficile à représenter dans un espace réduit (PCA peu performante) mais il donne quand même des résultats cohérents par artistes et morceaux. En choisissant de paramétrer 5 clusters dans notre algorithme du KMeans il est cependant évident que certains regroupements seront discutable (puisqu'il existe bien plus que 5 genres musicaux). Nous avons notamment remarqué des rapprochements comme "Do It For Love" de *Daryl Hall & John Oates* avec le morceau "Testify" de *Rage Against the Machine* qui ne sont pas vraiment du même registre.

8. Classification

Nous avons ensuite établi un modèle de classification afin de prédire si une musique est populaire ou non.

8.1. Classification sur la popularité

8.1.1. Déséquilibre de classe

Nous avons pu observer lors de l'étape de visualisation des données (cf. partie 4) que la répartition de notre note de popularité était très concentrée autour de 0. Cette répartition n'a rien d'étonnant car on comprend assez simplement qu'à l'échelle de la musique, seule une poignée de morceaux sont populaires. En revanche, dans le cadre de notre étude, cette disparité est un problème car elle engendre un fort déséquilibre de classe. Pour pallier celui-ci, nous avons scindé nos données en deux groupes distincts :

- Les musiques considérées populaires *(note de popularité supérieure à 40)*
- Les musiques considérées impopulaires *(note de popularité inférieure à 40)*

*Le seuil de 40 a été choisi car il est assez discriminant pour qu'on ne récupère que les musiques réellement populaires, tout en en gardant un nombre suffisant pour entraîner nos modèles de machine learning.

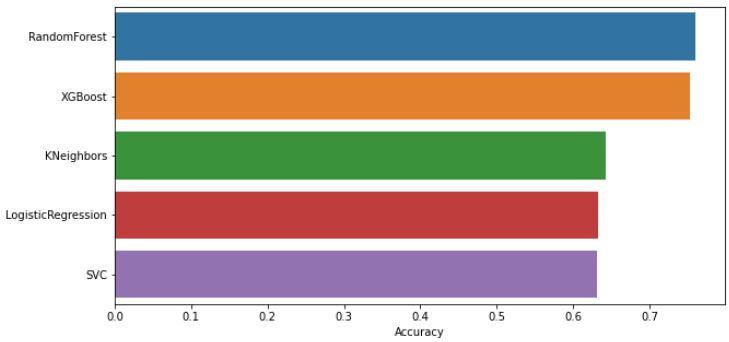
Et nous lui avons ensuite appliqué un "RandomUnderSampler" afin d'équilibrer ses deux classes. Il nous reste ainsi 49320 musiques (24660 par classe). Le sous-échantillonnage a été privilégié au sur-échantillonnage afin de diminuer les temps d'exécutions de nos modèles.

8.1.2. Modèles

Une fois le déséquilibre traité, nous avons entraîné plusieurs modèles dans le but d'avoir la meilleure prédiction possible. Ces modèles sont au nombre de cinq :

- Régression logistique
- SVM
- K plus proches voisins
- Forêt aléatoire
- XGBoost

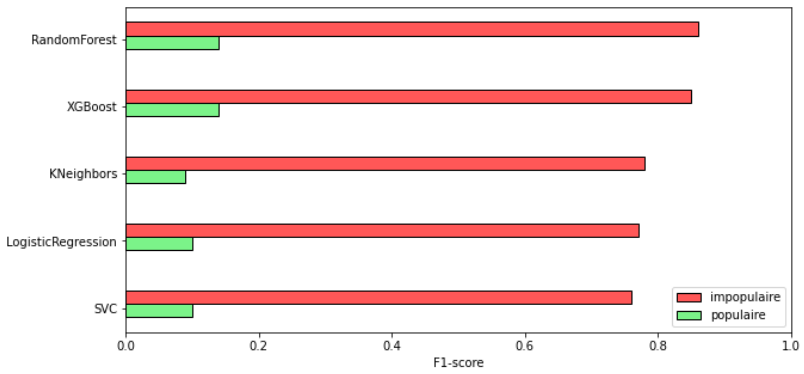
Une grille de recherche a été appliquée à chacun d'eux afin d'optimiser les différents hypers paramètres. Nous avons abouti aux résultats suivants :



On observe deux groupes dans nos modèles :

- un premier avec les algorithmes de forêt aléatoire et XGBoost dont la précision est aux alentours des 75%
- un second avec les trois autres algorithmes dont la précision vaut 63%.

On peut cependant contraster ces résultats en observant le score f1 pour chacune des classes :



On remarque effectivement que les prédictions sur les musiques populaires sont nettement moins fiables que celles pour les musiques impopulaires. Cette différence est due au fort déséquilibre entre nos deux classes au sein de l'ensemble de test. L'algorithme est donc bien plus performant sur la classe majoritaire que sur la classe minoritaire.

8.1.3. Interprétation

Il n'est pas étonnant que nous n'arrivions pas à prédire parfaitement la popularité d'une musique étant donné que nous nous basons exclusivement sur ses caractéristiques audios. Effectivement, on peut facilement intuiter que cela dépend de plusieurs autres facteurs notamment l'exposition médiatique et la popularité de l'artiste.

Par exemple, si demain Ariana Grande sort un son qui ne présente pas les caractéristiques d'un "tube", le modèle va prédire que ce son ne sera pas populaire. Or dans les faits, celui-ci sera quand même énormément écouté et aura sans doute une popularité élevée sur Spotify. A l'inverse, un artiste peu ou pas connu pourra sortir un morceau qui répond à toutes les attentes d'un morceau populaire, dans la réalité, il ne sera probablement jamais populaire.

Par ailleurs, notre donnée de sortie est elle aussi discutable car elle comporte le "défaut" d'être influencée par le temps. Ainsi, un morceau qui aurait été populaire en 2010 mais très peu écouté depuis 2012 aurait aujourd'hui une note de popularité faible sur Spotify. Ce qui a pour conséquence directe d'accentuer le déséquilibre entre nos classes.

Compte tenu de ces observations, nos résultats ne sont clairement pas ridicules. Il serait utopique de s'attendre à des résultats de l'ordre de 90% au vue des données passées en entrée.

8.2. Classification avec de nouvelles données cibles

8.2.1. Récupération des données

Nous avons tout de même tenu à effectuer un second modèle de classification en prenant une valeur cible différente pour tenter d'améliorer nos résultats. Pour cela nous avons utilisé des méthodes de "web scraping".

Nous avons, tout d'abord, voulu récupérer la note moyenne de chacune de nos musiques via le site https://www.senscritique.com/. Cette solution s'est rapidement avérée inefficace du fait du trop grand nombre de musiques non notées ou non référencées sur le site (environ 85% de notre jeu de données). Nous avons alors changé de site et choisi YouTube pour s'affranchir

de ce genre de problème. Deux colonnes ont ainsi été rajoutées à notre dataset : le nombre de vues et le nombre de mentions “j’aime” correspondant à chaque musique.

id	name	artists	views_YT	likes_YT
1mKXFLRA179hdOWQBwUk9e	Just Give Me a Reason (feat. Nate Ruess)	['P!nk', 'Nate Ruess']	1354746625	5694757
4saklk6nie3yiGePpBwUoc	Dynamite	['BTS']	1319834325	31799014
3ZFTkvIE7kyPt6Nu3PEa7V	Hips Don't Lie (feat. Wyclef Jean)	['Shakira', 'Wyclef Jean']	1009127864	5241065
1rfofaqEpACxVEHIZBJe6W	Havana (feat. Young Thug)	['Camila Cabello', 'Young Thug']	1841791372	9966870
41Ffig7qHiVOD6dEPvsCzO	Worth It (feat. Kid Ink)	['Fifth Harmony', 'Kid Ink']	2031154327	10183650

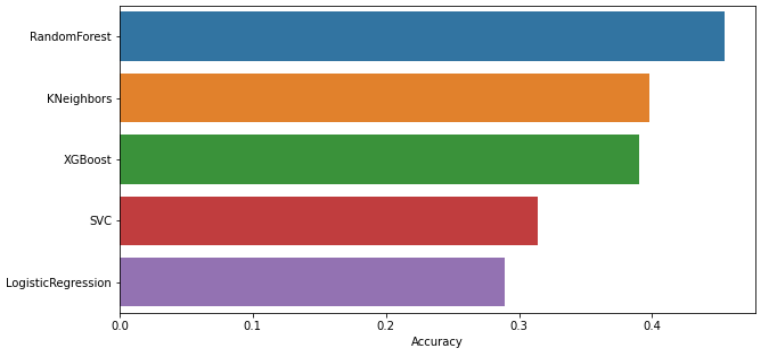
**Ce scraping n’a été fait que pour les 12000 musiques les plus populaires par souci de temps*

A partir de ces données, nous avons tenté d’établir des modèles de régression pour prédire le nombre de vues de nos morceaux. Ces derniers n’ayant pas donné de résultats convaincants (2% de précision) nous nous sommes tournés vers des modèles de classification avec les classes suivantes :

- Classe 0 : vues < 1M
- Classe 1 : vues ∈ [1M, 10M [
- Classe 2 : vues ∈ [10M, 100M]
- Classe 3 : vues > 100M

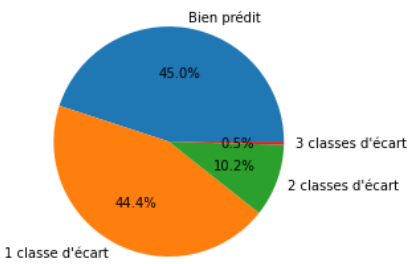
8.2.2. Modèles

Les mêmes modèles que précédemment ont été appliqués, voici les résultats obtenus :



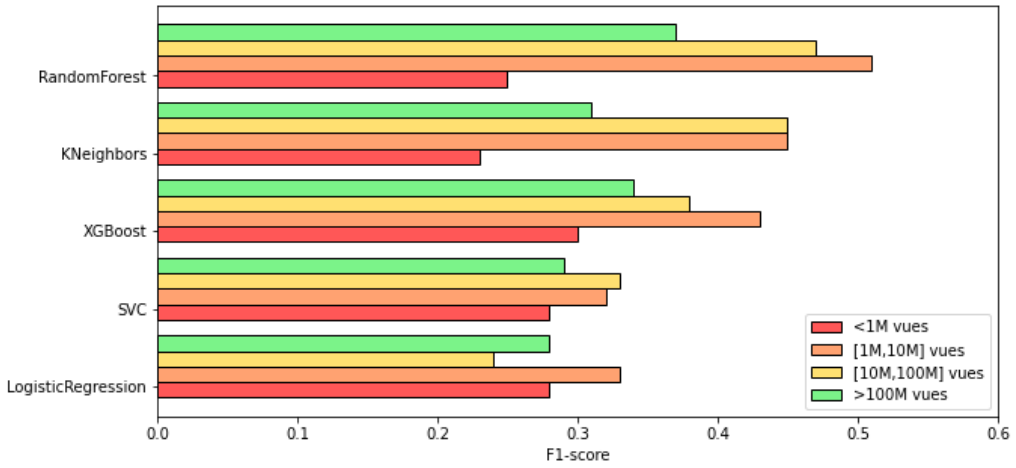
On remarque que nos modèles sont tous meilleurs que le hasard (25% de précision), et que l’un d’eux sort nettement du lot. Certes le modèle de foret aléatoire se trompe encore 55% du temps mais il donne quand même une bonne idée de la catégorie de la vidéo (tout ceci, en sachant qu’il ne se base que sur des caractéristiques audios dont nous avons vu les limites précédemment). Cette précision de 45% est donc plus que correcte.

Nous avons ensuite quantifié l’erreur de ce modèle (Random Forest) :



On constate ainsi que la plupart des erreurs que fait le model sont des erreurs sur les classes voisines. Elles sont donc moins grave que des erreurs sur les classes extrêmes. En pratique, seuls 10,7% de nos prédictions sont vraiment très éloignées de la réalité ce qui est rassurant.

On peut là encore visualiser le score f1 pour chacune de nos classes :



Les résultats sont bien plus regroupés que dans les modèles utilisant la popularité comme variable cible, en revanche la valeur des scores est assez faible ce qui confirme le fait que la fiabilité de nos prédictions est discutable.

8.2.3. Interprétation

L'interprétation est la même que précédemment à savoir que nos données d'entrées sont trop pauvres pour expliquer à elles seules le nombre de vues d'un titre. En partant de ce constat, bien que notre meilleur modèle soit encore loin de la perfection on peut dire qu'il fournit une prédiction honorable.

9. Recommandation

Nous avons enfin tenté de mettre en place un système de recommandation musicales.

9.1. Utilisations des caractéristiques des musiques

Dans un premier temps nous avons utilisé deux caractéristiques de la musique, le genre musical et la popularité. Nous avons créé trois dictionnaires :

- Le premier met en relation une musique avec ces genres musicaux
- Le second met en relation un genre musical avec les musiques qui lui sont associé
- Le troisième met la relation une musique et sa popularité

A partir de ces dictionnaires nous pouvons obtenir, pour une musique, les genres musicaux associés. Puis, à partir de ces genres nous obtenons les musiques associées que nous classons ensuite en fonction du nombre de genres musicaux en commun et de leur popularité. Nous pouvons alors recommander pour chaque musique, celles qui sont les plus proches au niveau du genre et ayant la popularité la plus importante (les plus écoutées).
On obtient par exemple le résultat suivant :

Musique écoutée

	track_id	titre	album	artist
0	1MQTmpYOZ6fcMQc56Hdo7T	Sleep Now In the Fire	The Battle Of Los Angeles	Rage Against The Machine

Recommandation

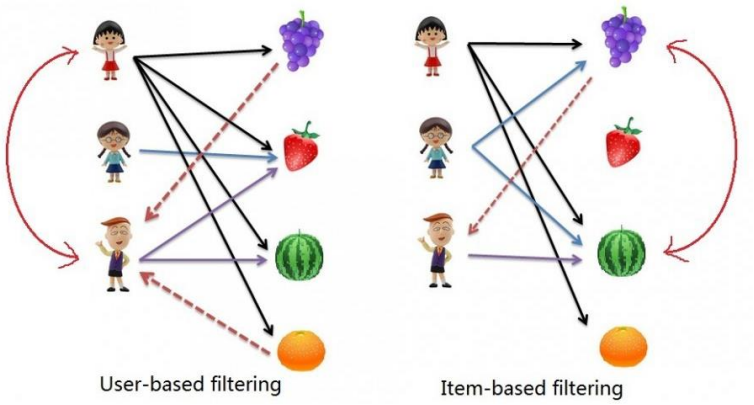
	track_id	titre	album	artist
0	59WN2psjkt1tyaxjspN8fp	Killing In The Name	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine
1	0tZ3mElWcr74OOhKEiNz1x	Bulls On Parade	Evil Empire	Rage Against The Machine
2	1wsRitfRRtWyEapl0q22o8	Guerrilla Radio	The Battle Of Los Angeles	Rage Against The Machine
3	5YBVDvTSSSiqv7KZDeUlXA	Renegades Of Funk	Renegades	Rage Against The Machine
4	7lmeHLHBe4nmXzuXc0HDjk	Testify	The Battle Of Los Angeles	Rage Against The Machine
5	2rBHnlxbhkMGLpqmsNX91M	Bombtrack	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine
6	2QiqwOVUctPRVggO9G1Zs5	Wake Up	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine
7	1XTGyfJeMiZXrZ1W3NolcB	Know Your Enemy	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine
8	25CbtOzU8Pn17SAaXFjIR3	Take The Power Back	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine
9	4K1DB7EedHPuVnhVrnvf2U	How I Could Just Kill a Man	Renegades	Rage Against The Machine

Cette méthode de filtrage basée sur les caractéristiques donne des résultats corrects mais de nos jours les systèmes de recommandation les plus utilisés sont ceux prenant en compte le choix des utilisateurs, ce sont des systèmes de filtrage collaboratif.

9.2. Utilisations des playlists

Pour mettre en œuvre un système de filtrage collaboratif, nous avons besoin de données utilisateur. Pour cela, nous avons utilisé un second jeu de données contenant un million de playlist avec en moyenne 62 chansons dans chacune d'elles.

Ce système de recommandation nous permet d'effectuer deux types de recommandation différentes :



- La recommandation “items items” qui prendra un produit particulier, trouvera les utilisateurs qui ont aimé ce produit et trouvera d'autres éléments que ces utilisateurs ont également aimé. Dans notre cas, nous recommandons des musiques en fonction d’une autre musique.
- La recommandation “user user” prend un utilisateur, recherche des utilisateurs similaires à ce dernier sur la base d'évaluations ou de comportements similaires et recommande des produits que ces utilisateurs similaires ont aimés. Dans notre cas, nous recommandons des musiques en fonction d’une liste de musique correspondant à un utilisateur.

Nous avons ensuite voulu appliquer ces méthodes à la jointure du jeu de données playlist et musique puis l’élargir à tout le jeu de données playlist.

Pour mettre en œuvre ce système de recommandation, nous avons d'abord voulu utiliser une méthode de réduction de matrice pour déterminer les musiques ayant le plus grand nombre de playlist en commun ou les playlists ayant le plus grand nombre de musiques en commun. Pour ce faire nous avons utilisé les méthodes de matrice sparse et de SVD de l’api scipy mais les résultats obtenus ne furent pas concluants.

Finalement pour mettre en œuvre de système de recommandation nous avons employé une méthode similaire utilisée pour les caractéristiques musicales. Nous avons créé deux dictionnaires :

- Le premier met en relation une musique et ses playlists
- Le second met en relation une playlist et ses musiques

Pour la recommandation “items items” nous déterminons pour chaque musique à l’aide du premier dictionnaire les playlists dans lequel elle se trouve, puis grâce au second dictionnaire nous obtenons toutes les musiques présentes dans ces playlists. Nous recommandons ensuite les musiques qui apparaissent le plus souvent dans les playlists à l'exception de celles utilisées pour la recommandation.
On obtient par exemple le résultat suivant :

Musique écoutée

	track_id	titre	album	artist
0	1MQTmpYOZ6fcMQc56Hdo7T	Sleep Now In the Fire	The Battle Of Los Angeles	Rage Against The Machine

Recommandation

	track_id	titre	album	artist
0	59WN2psjkt1tyaxjspN8fp	Killing In The Name	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine
1	0tZ3mElWcr74OOhKEiNz1x	Bulls On Parade	Evil Empire	Rage Against The Machine
2	7lmeHLHBe4nmXzuXc0HDjk	Testify	The Battle Of Los Angeles	Rage Against The Machine
3	1wsRitfRRtWyEapl0q22o8	Guerrilla Radio	The Battle Of Los Angeles	Rage Against The Machine
4	5YBVDvTSSSiqv7KZDeUlXA	Renegades Of Funk	Renegades	Rage Against The Machine
5	6ZU9RJlZ0fNaFuQM57bDIA	Bombtrack	Rage Against The Machine	Rage Against The Machine
6	5UWwZ5lm5PKu6eKsHAGxOk	Everlong	The Colour And The Shape	Foo Fighters
7	1IDAJagxB9AQjjYXaiDK1j	Know Your Enemy	Rage Against The Machine	Rage Against The Machine
8	6W21LNLz9Sw7sUSNWMSHRu	Freak On a Leash	Follow The Leader	Korn
9	2QiqwOVUctPRVggO9G1Zs5	Wake Up	Rage Against The Machine - XX (20th Anniversar...	Rage Against The Machine

Nous avons ensuite voulu recommander des chansons pour une liste d'écoute. Pour mettre en œuvre la recommandation "user user", nous aurions normalement dû déterminer les playlists les plus proches de notre liste d'écoute (les playlists contenant le plus des musiques de notre liste d'écoute) et effectuer comme pour "items items" la liste des musiques présentes dans ces playlists mais également leur attribuer un poids en fonction de leur playlist d'origine (plus la playlist est proche de notre liste d'écoute plus le poids est important). Pour finir, on détermine le score de chaque musique qui vaut la somme de ces poids (par exemple une musique présente dans 3 playlists qui ont des poids de 1 ,0.7 et 0.5 a un score de 2.2). Ensuite on recommande les musiques ayant le score le plus élevé.

Dans notre cas, du fait du manque de temps, nous avons développé un système équivalent et plus simple à mettre en œuvre. Nous appliquons à chaque musique de la playlist la même méthode que pour la recommandation "items items". Mais contrairement à cette dernière, au lieu de faire une recommandation, nous réunissons les résultats des différentes musiques de la liste d'écoute pour recommander les musiques qui apparaissent le plus souvent.

Dans cette méthode simplifiée on retrouve l'idée de poids avec le fait qu'une playlist contenant plusieurs musiques de la liste de lecture est prise en compte plusieurs fois lui donnant plus d'importance.

On obtient par exemple le résultat suivant :

Musiques écoutées

	track_id	titre	album	artist
0	57bgtoPSgt236HzfBOd8kj	Thunderstruck	The Razors Edge	AC/DC
1	3jUTjCISntIUFL8jnAjzgc	Fallen Leaves	Billy Talent II	Billy Talent
2	1DeaByFASTvBxtYANOyFXy	Surrender	Billy Talent II	Billy Talent
3	7LRMbd3LEoV5wZJvXT1Lwb	T.N.T.	High Voltage	AC/DC
4	2QSAj76Ba6aMFX9RlXdUdO	J'ai demandé à la lune	Paradize	Indochine
5	6nTilhLmQ3FWhvrGafw2zj	American Idiot	American Idiot	Green Day

Recommandation

	track_id	titre	album	artist
0	08mG3Y1vljYA6bvDt4Wqkj	Back In Black	Back In Black	AC/DC
1	2zYzyRzz6pRmhPzyfMEC8s	Highway to Hell	Highway to Hell	AC/DC
2	7o2CTH4ctstm8TNelqjb51	Sweet Child O' Mine	Appetite For Destruction	Guns N' Roses
3	0bVtevEgtDIeRjCJbK3Lmv	Welcome To The Jungle	Appetite For Destruction	Guns N' Roses
4	2SiXAy7TuUkycRVbbWDEpo	You Shook Me All Night Long	Back In Black	AC/DC
5	1AhDOtG9vPSOmsWgNW0BEY	Bohemian Rhapsody - Remastered 2011	A Night At The Opera (2011 Remaster)	Queen
6	3YBZIN3rekqsKxbJc9FZko	Paradise City	Appetite For Destruction	Guns N' Roses
7	4CJVkjo5WpmUAKp3R44LNb	Sweet Home Alabama	Second Helping	Lynyrd Skynyrd
8	5MxNLUsfh7uzROypsoO5qe	Dream On	Aerosmith	Aerosmith
9	4bHsxqR3GMrXTxEPLuK5ue	Don't Stop Believin'	Escape	Journey

Mais ces méthodes créent également un biais, en effet, les musiques qui apparaissent dans moins de playlist ont moins d'importance dans la recommandation que celles qui apparaissent plus souvent. On obtient donc un déséquilibre où des musiques peuvent être beaucoup plus prises en compte pour la recommandation que d'autres.

10. Conclusion

Nous avons pu voir lors de la visualisation des données que nos variables étaient très peu corrélées que ce soit entre elles ou avec la variable de sortie. Dans ce contexte nous avons pu mener plusieurs études telles que de la classification, du clustering, et enfin de la recommandation.

La classification avait pour but de déterminer l'appréciation d'un titre à partir de ses caractéristiques audios. Nous avons effectué deux classifications distinctes : l'une qui considère la popularité comme variable cible et l'autre le nombre de vues. Dans les deux cas, nous avons eu des résultats corrects dans l'ensemble bien qu'on soit loin de la perfection. On peut donc dire que l'objectif initial est partiellement atteint car on arrive à fournir une prédiction mais la fiabilité de cette dernière est discutable. Ceci est cependant compréhensible car les caractéristiques à notre disposition ne peuvent pas expliquer à elles seules et de façon totalement fiable des phénomènes aussi complexes que la réaction des consommateurs a une nouvelle musique.

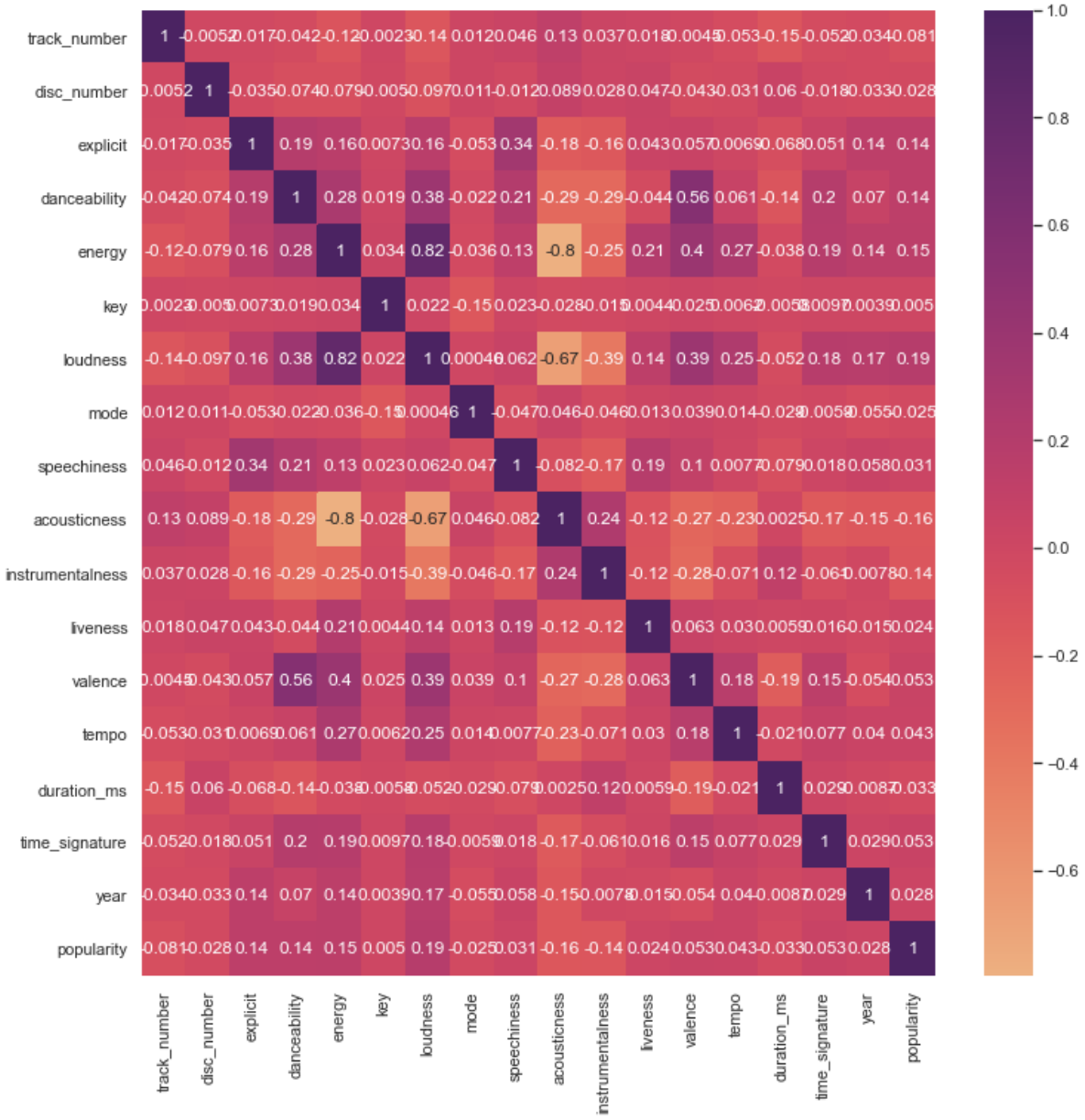
Le clustering avait, quant à lui, pour objectif de regrouper les musiques entre elles en fonction de leurs caractéristiques audios. Nous avons utilisé une méthode à partir de l'algorithme KMeans et l'avons évalué avec une méthode de réduction de dimension. Les résultats sont relativement intéressants, mais la PCA a été difficile à mettre en place et donnait de très mauvais ratios de variance. Ces résultats sont en accord avec le fait qu'il est compliqué de regrouper dans seulement cinq groupes des morceaux très différents et que nos caractéristiques audios sont peu corrélées entre elles.

L'idée suivante était de créer un système de recommandation à l'aide de la distance entre les musiques au sein des clusters mais, compte tenu du fait que nous ayons dû changer de données en cours de projet, nous n'avons pas pu aller aussi loin que prévu et avons opté pour d'autres méthodes. En effet, le système de recommandation n'a pas été fait à partir de ces clusters mais directement à partir des genres musicaux que nous avions extrait de l'API Spotify et de playlist obtenu à l'aide d'un autre jeu de donnée. Les algorithmes auxquels nous avons abouti sont capable de suggérer des musiques proches de celle(s) qu'on lui passe(nt) en entrée. Nous pouvons donc dire que l'objectif consistant à établir un algorithme de recommandation est atteint.

11. Annexes

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Dot, Add, Flatten
from tensorflow.keras.regularizers import l2

## Chargement des données
Numero_user_train=
Numero_musique_train=
note_train=

Numero_user_test=
Numero_musique_test=
note_test=


U= #Nombre d'utilisateur
M= #Nombre de Musique

#Creation donné train et test
mu_train = note_train.mean()
X_train=[ Numero_user_train, Numero_musique_train]
y_train =note_train - mu_train

mu_test = note_test.mean()
X_test=[ Numero_user_test, Numero_musique_test]
y_test =note_test - mu_test


# Variable
K=  # dimensionnalité latente a definir
regularization =  # pénalité de régularisation a definir
epochs=  # A definir
batch_size= # A definir

u=Input(shape=(1,))
m=Input(shape=(1,))

u_embedding = Embedding(U, K, embeddings_regularizer=l2(regularization))(u)
m_embedding = Embedding(M, K, embeddings_regularizer=l2(regularization))(m)

u_bias = Embedding(U, 1, embeddings_regularizer=l2(regularization))(u)
m_bias = Embedding(M, 1, embeddings_regularizer=l2(regularization))(m)

x = Dot(axes=2)([u_embedding, m_embedding])
x = Add()([x, u_bias, m_bias])
x = Flatten()(x)

model = Model(inputs=[u, m], outputs=x)
model.compile(loss='mse', optimizer='adam',metrics=['mse'])
model.fit(x=X_train,y=y_train ,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_test,y_test))
```

Annexe 1 - Exemple de réseaux de neurones pour la recommandation



Annexe 2 - Matrice de corrélation du nouveau jeu de données