Basics of Data Science Course - Assignment 3

Soroush heidary

96222031

Data Set : Spotify

Summary :

The data is consisted of around 20 features each describing some feature like the lordliness of the music or which musical key is it written on, some of these features we wont be needing like the name or the id of the song so we'll just drop them, the remaining 13 features are listed below along side of their box plot

Although we'll just select 6 of these features who seem to be the most characteristic features to be used in the clustering algorithms

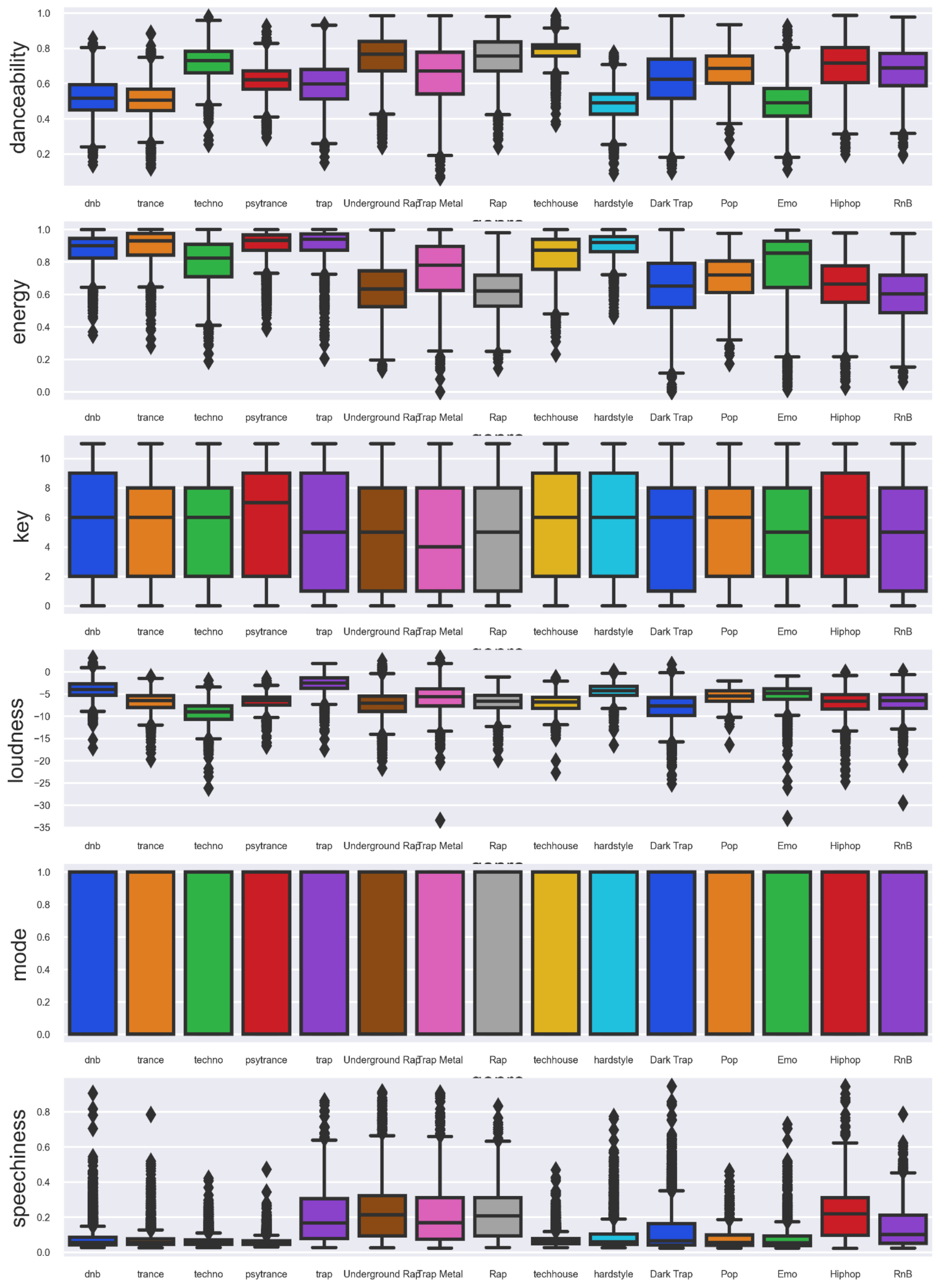
As the known "Curse of dimensionality" describes too, we cannot visualize nor mathematically prove that our clustering was a proper clustering, (the data in non-linear so methods like DB-index, or Dunn-index cannot prove the worth of our clustering to the fullest potential) so we're left with two options:

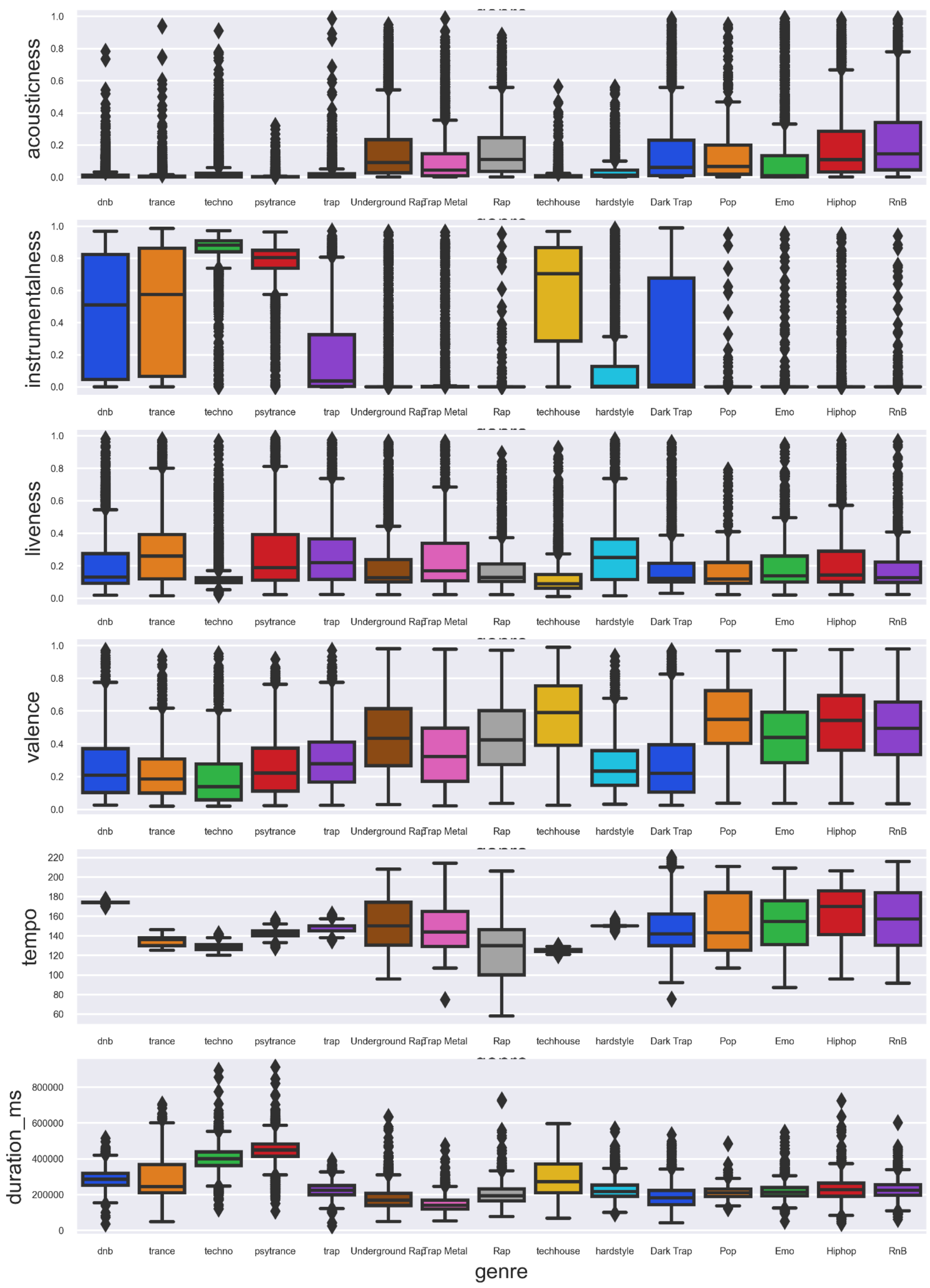
- One to reduce the dimensionality of our data to the point that we could visualize our results so that some intuition could help us with the tuning of our model
- And Two to use criteria systems like DB index or Dunn index and tune the clustering

My focus has been mainly dedicated to the first option.

As for data pre processing I've used both a standardized and normalized data, as well as the raw data, as some of the models do not simply need any data scaling also for the sake of showing the difference too.

But before we hop into the details lets see the box plot distribution of each feature as it helps us in selecting 10 of these (dropping 2 of them as they don't really show any difference in each genre), we won't be trying to predict our genre but it still is helpful to reduce the dimension and use only the most distinguishing features


danceability
energy
key
loudness
mode
speechiness
dnb
trance
techno
psytrance
trap
Underground Rap
Trap Metal
Rap
techhouse
hardstyle
Dark Trap
Pop
Emo
Hiphop
RnB
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
0
-5
-10
-15
-20
-25
-30
-35


acousticness
instrumentalness
liveness
valence
tempo
duration_ms
dnb
trance
techno
psytrance
trap
Underground Rap
Trap Metal
Rap
techhouse
hardstyle
Dark Trap
Pop
Emo
Hiphop
RnB
genre

10 features selected to be processed from now on are :

'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms',

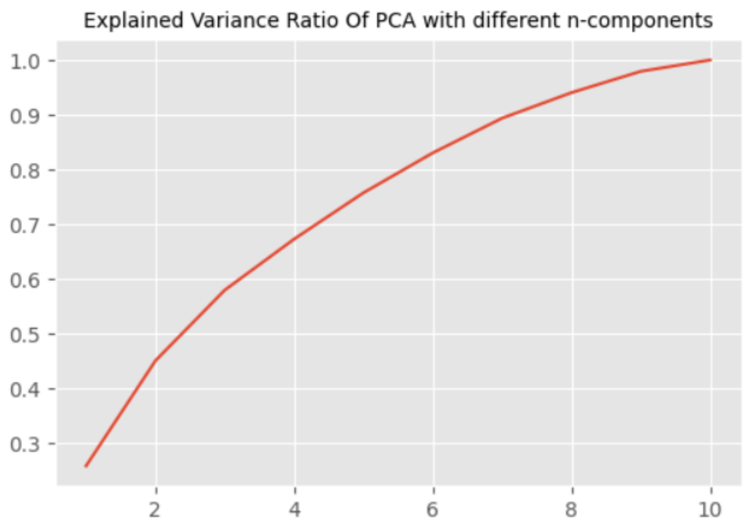
Below we see each of the dimension reduction methods used to reduce the dimensionality for us to be able to cluster more precisely each of the clustering methods use a different clustering algorithm. As each clustering algorithm requires a specific type of data. Some of them would be more successful using a density based approach others would work much better if we pre assign the number of clusters, for example, DBScan uses a density based clustering and it does not require pre defining the number of clusters

Detailed Report of each method :

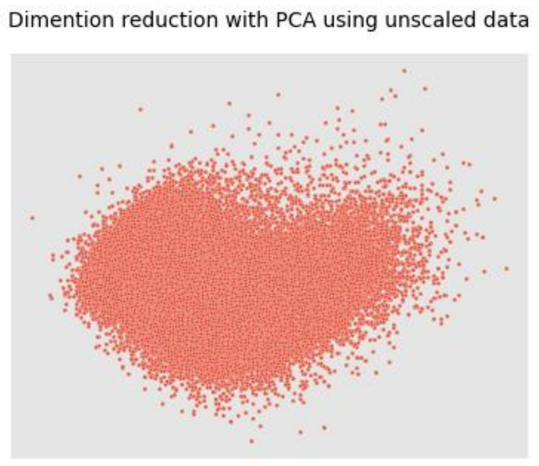
Linear techniques: Principal Component Analysis(PCA), Factor Analysis(FA):

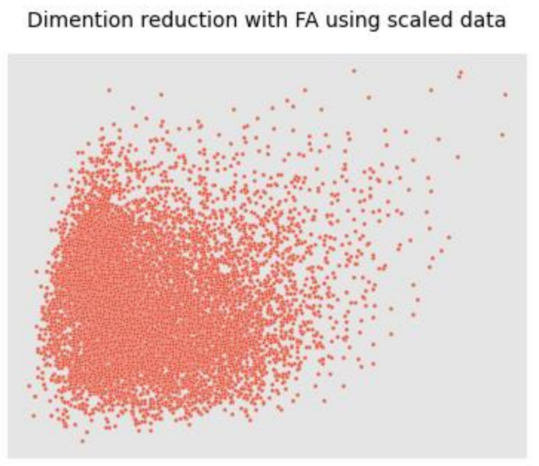
For these two methods I've used 4 components to reduce to, because 2 is just too little information, below you see why our data doesn't work very well with PCA and FA as they are methods used for linear data, and the chart below is an indication that our data is not linear indeed.

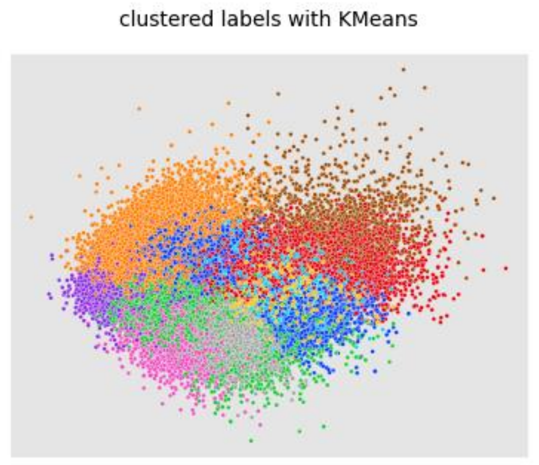
(Explained Variance Ratio suggest how much of the actual data(y-axis) does our n-component PCA(x-axis) explains, in a perfect linear space this chart should have an exaggerated curve towards the upper left section of the plot)

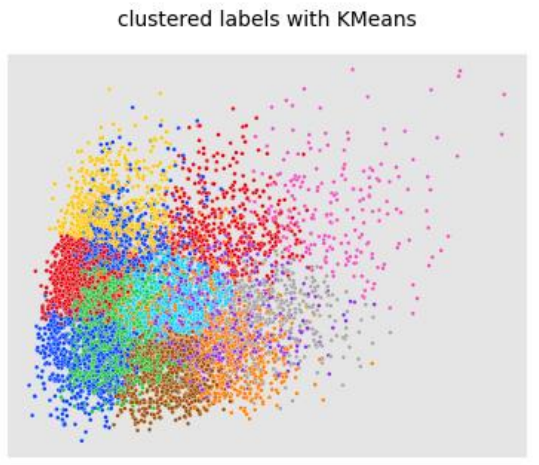


Although we still use FA and PCA of 4 components reduction and cluster upon those 4 features with K-Means clustering technique with 14 cluster labels, and use the first 2 components to visualize the results, and use this pipeline (collection of connected functions applied to a data) to get recommendations later on, here is the results :









Non-Linear techniques:

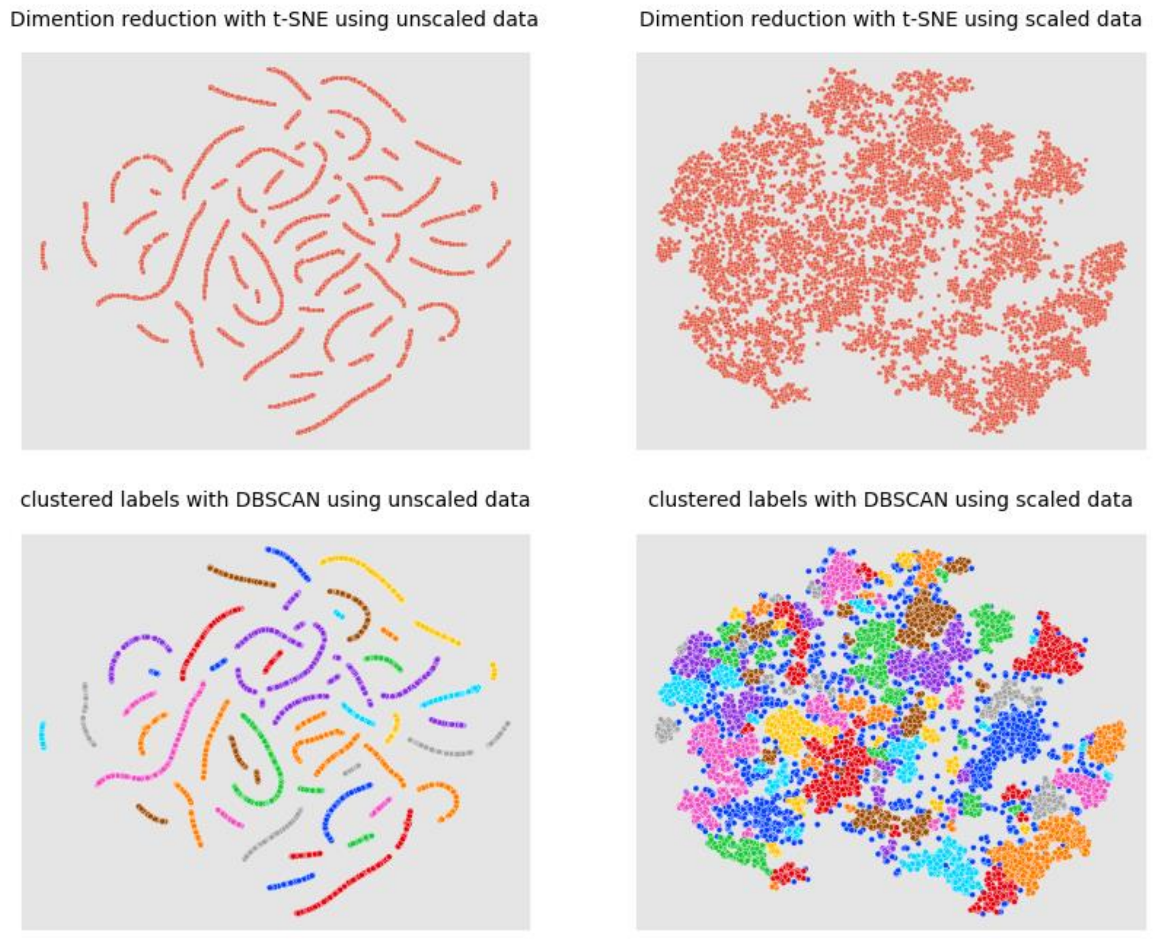
- T-Distributed Stochastic Neighbor Embedding(t-SNE)
- Auto encoders
- An ensembled model using autoencoders and t-SNE
- And a Variational Autoencoder which seemingly outperforms other techniques

t-SNE :

this method showed very promising results (although t-SNE is mostly used for visualization purposes but in our case even 2 components could make our data seem very much clusterable). Though there was this huge problem which I didn’t realize until the end that t-SNE is not a single data point transformer, meaning that you cannot transform a single data point using an already fitted t-SNE model, this method uses the whole data and has only a fit_transform method so we cannot use this in later recommendation

system, such a shame
left column is a t-SNE model fitted on the unscaled data, and the right one uses standardized data (unscaled data had feature that would overshadow the other one's so left plots are not really scientifically approved but they looked cool :D so I put them here)

and for the clustering I used DBScan this time, as DBScan is a method which uses density of each neighbourhood of points to create clusters, You can think of this method as a contagious virus spreading in a city selecting a random point and assigning neighbors to it by a predefined radius, This method could also be used to detect outliers, he would sit and radius and a minimum number of members each cluster so if some data points fail to reached at minimum threshold they be detected as outliers actually hear blue colored points are those who do not belong to any of the Clusters and they are detected as outliers

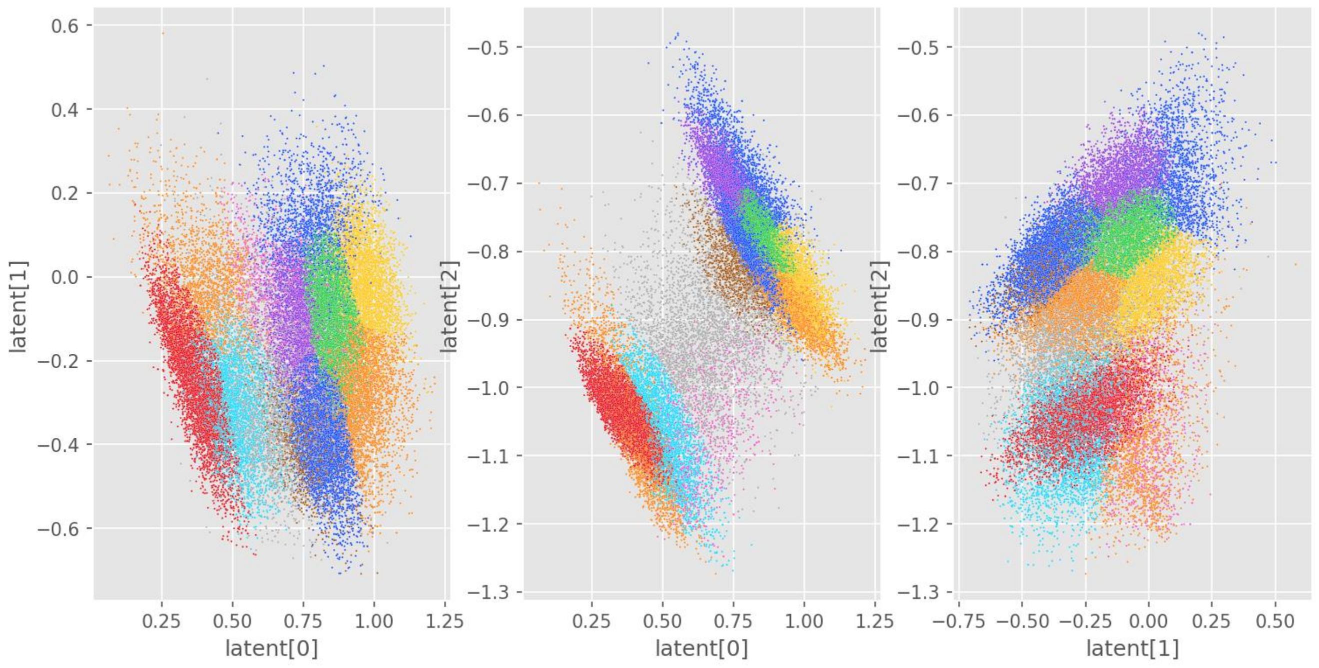


Although later on I realize that you cannot use this method that to transform new data points in real-time this method uses whole data as it's fitting parameter some approaches tried to solve this issue for example I try to train a neural net for too approximate the behavior of t-SNE and use that Neural Network instead this way we could have real-time predictions for each new entry.   But the idea was failed to be implemented because the neural network couldn't find the right Behavior with acceptable accuracy.

Auto encoders :

This method uses a simple autoencoder to reduce the dimensions of the data, different tunings they're tried out. But this one showed the most promising results also had the least reconstruction loss.

Do you see the results As the 10 dimensional data was mapped into a three dimensional latent vector with the reconstruction loss of 0.01 mean squared error using the normalized data, as mapping a 10 dimensional data into a three dimensional data proves to be a very difficult task. we could expect our reconstruction loss to be a high number. We see each of the latent variables plotted with respect to the other ones below, also clustering is done with the latent vector as input to the clusterer with 12 clusters

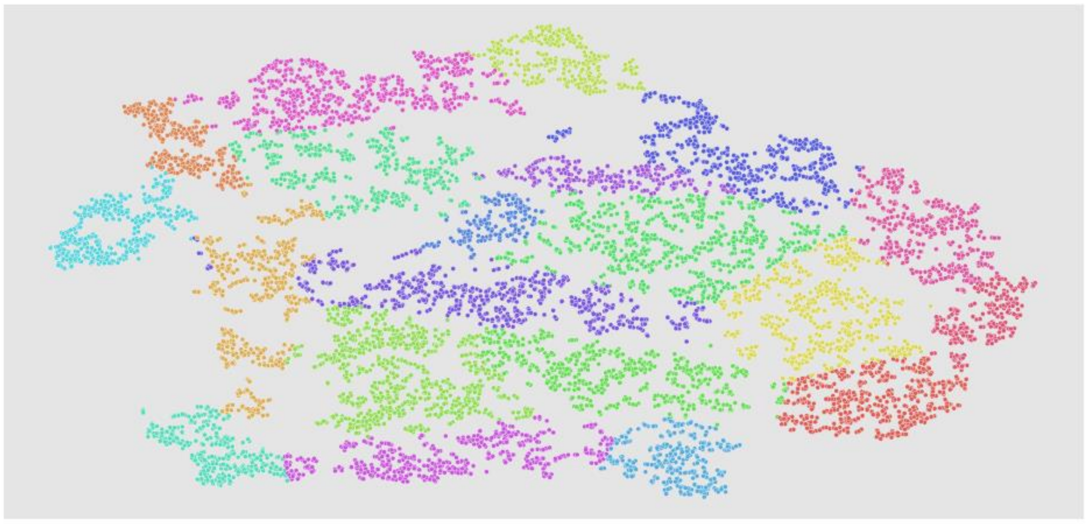


Auto encoder followed by a t-SNE :

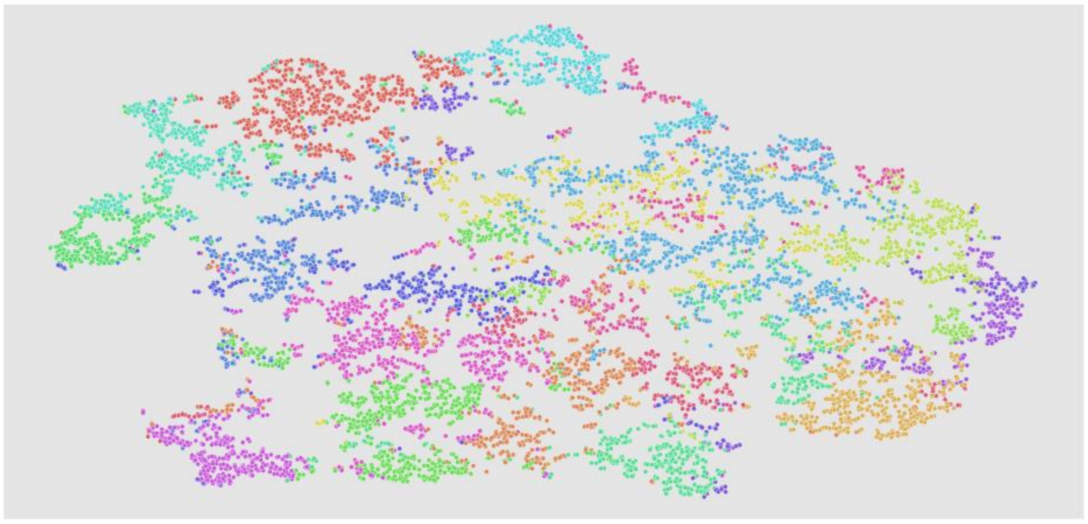
This method had the most promising results and even by riddled reducing the 10 dimensions into two And only two dimensions. We still had a very clear cluster Bill data with low reconstruction loss on our auto encoder (0.06 MSE) First an autoencoder were used to reduce the dimensionality to five latent variables then a TSN. A model was trained to map those five variables into a two dimensional data.

But as I mentioned before, we cannot use a t-SNE model to predict new data in real time. So, I could not use this model in the recommender system, but it still did worth a mention.

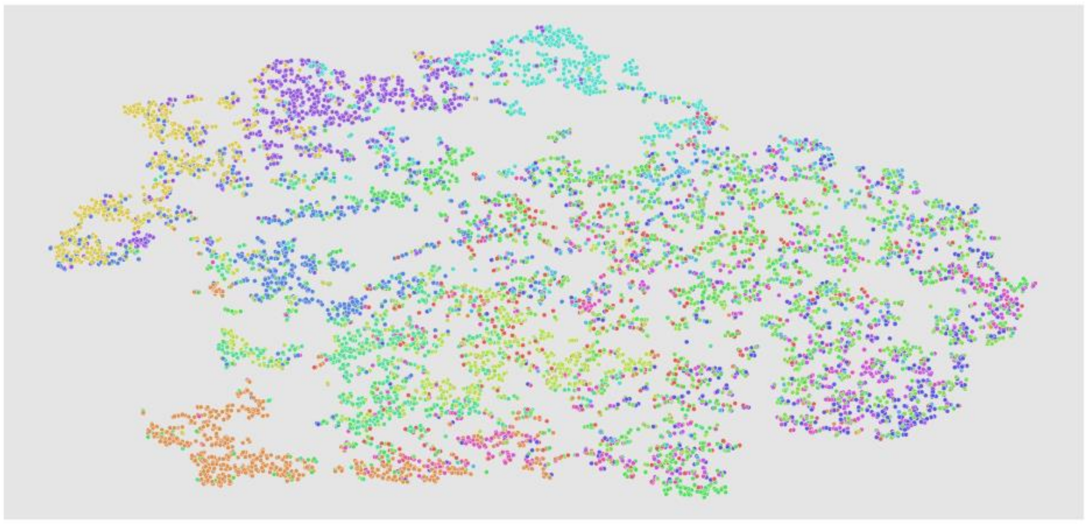
Dimention reduction with Autoencoder and t-sne, clustered using gaussian mixure over tsne output



Dimention reduction with Autoencoder and t-sne, clustered using gaussian mixure over autoE output



Dimention reduction with Autoencoder and t-sne, colored over real genres

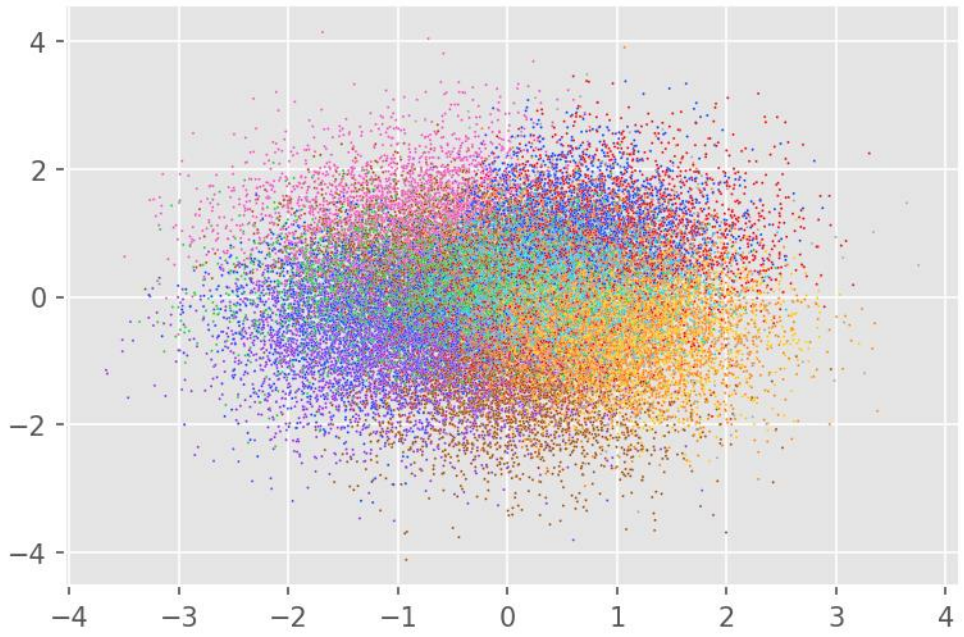


Variational Autoencoder :

The idea behind using a variational autoencoder was to extract distinct features each of which being responsible for some kind of feeling in the song, something like sadness, happiness, anger etc.. though further actions could be made to test the above theory I did not have the time to do so, so I'll just use it to recommend songs and see what happens, also having an variational auto encoder could possibly enable us to further more customize customer experience meaning we could let the customer to choose if they want a little bit more of each feeling in their recommendation (assuming each of the latent variables shows each feeling) and that would be a real high level customization to offer, but this idea requires much more tuning and effort to "practicalize" (if that's even a word :D).

This model had a loss of 0.36 MSE loss which is not the best (did have the time to tune it though) but the idea itself is worth mentioning, the clustering is fitted using the output of the encoder which was a 5 dimensional data but I applied a simple PCA just to have a quick visualization.

A gaussian mixture algorithm was used for the clustering with 14 cluster labels (GassianMixture was a good choice here for the algorithm because of several reasons, one the data itself had a somewhat normally distributed form, another that KMeans tend to lose it's effectiveness in high dimensional inputs)

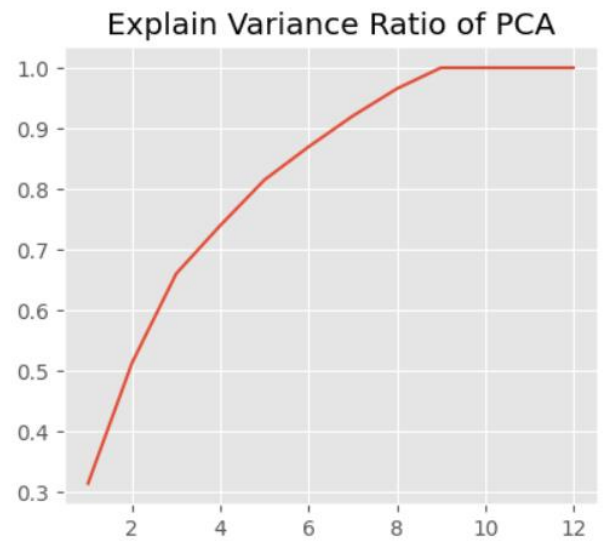


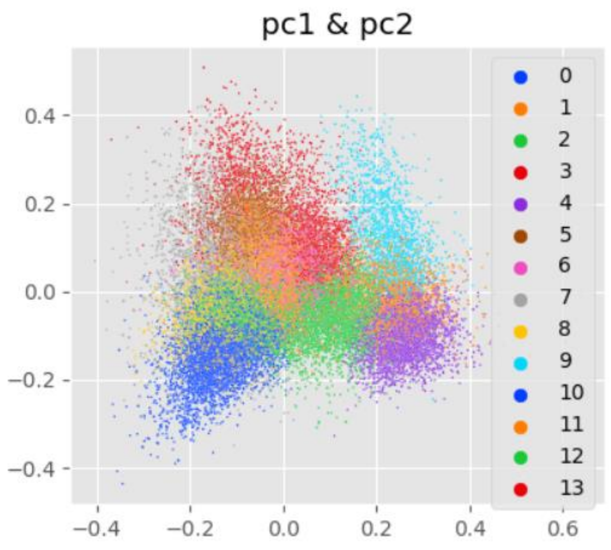
Also I tried to transform our non-linear data to a linear one which are these two techniques :

Linearizing autoencoder :

An autoencoder which it's latent vector does not do reduction but also adds to the input's dimensions, but in doing so, tries to linearize the data, this idea didn't perform as fancy as the theory suggests so I won't be using this in the actual recommender system.

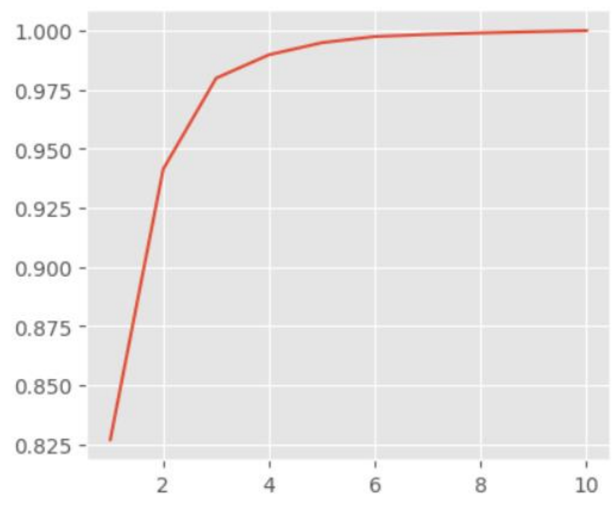
This autoencoder had a very low reconstruction loss (about 0.06 MSE) but not the latent vector which is of size 14 isn't as linear as we'd hope for as you see the Explained Variance Ratio of a PCA applied to it.
I've used 4 pc's for the clustering, and shown its result on the first 2 pc's

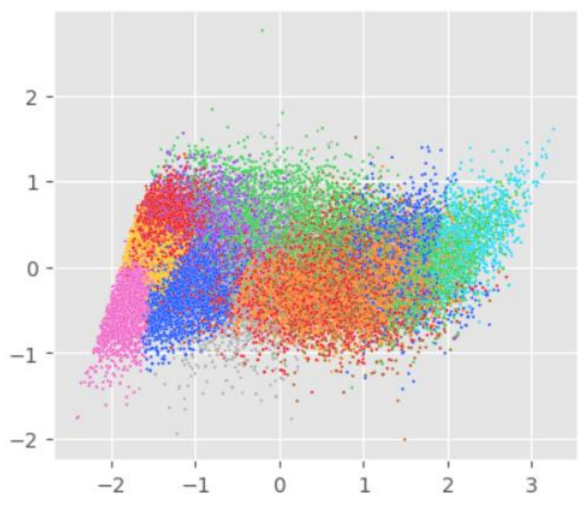




Box Cox transformer :

A power transformer called box-cox transformation which claims to do exactly what we're looking for but in theory it would need a lot of tuning around our actual data, meaning we have know how our data look likes in the n-dimensional space and select the right function to linearize it, you can think about it this way: when we have a dataset who's features have this relation: f1 = f2 ^ n by applying a logarithmic function we can achieve linearity, so you get the idea, we need to know how our data looks like and in theory it would be very hard to get practical data out of this transformation, although in practice it worked perfectly :/ and only god knows why (unless there is a data leakage somewhere that I'm not aware of), after this transformation I applied a simple PCA on the data and this chart shows the "explained variance ratio" which has an improvement comparing to fig1 :
Here you see the explained var ratio follow by the data that has been reduced to 4 pc's and clustered using Gaussian Mixture with 14 clusters





Recommender System :

Now that we have our clustering models fitted we can pickle them to use for further recommending songs it would just transform the new data point which is the user input and by finding the cluster label we could just randomly select 5 songs in the same cluster I recommend it to the user another approach would be to select five of the most closest data points in the same cluster that our user input has been calculated in.

another method to create a recommender system is to simply not use any clustering algorithms at all but to use different metrics of distances between our user input and the whole data set this way we could use five of the most closest data points to our user data one of these distances is the simple you eucilidian distance the other one could be the cosine distance which by some sense calculates the angle between two vectors in space one being the user input the other being each row of our data set will use these two metric distances too

these 5 methods are used to recommend a playlist each:

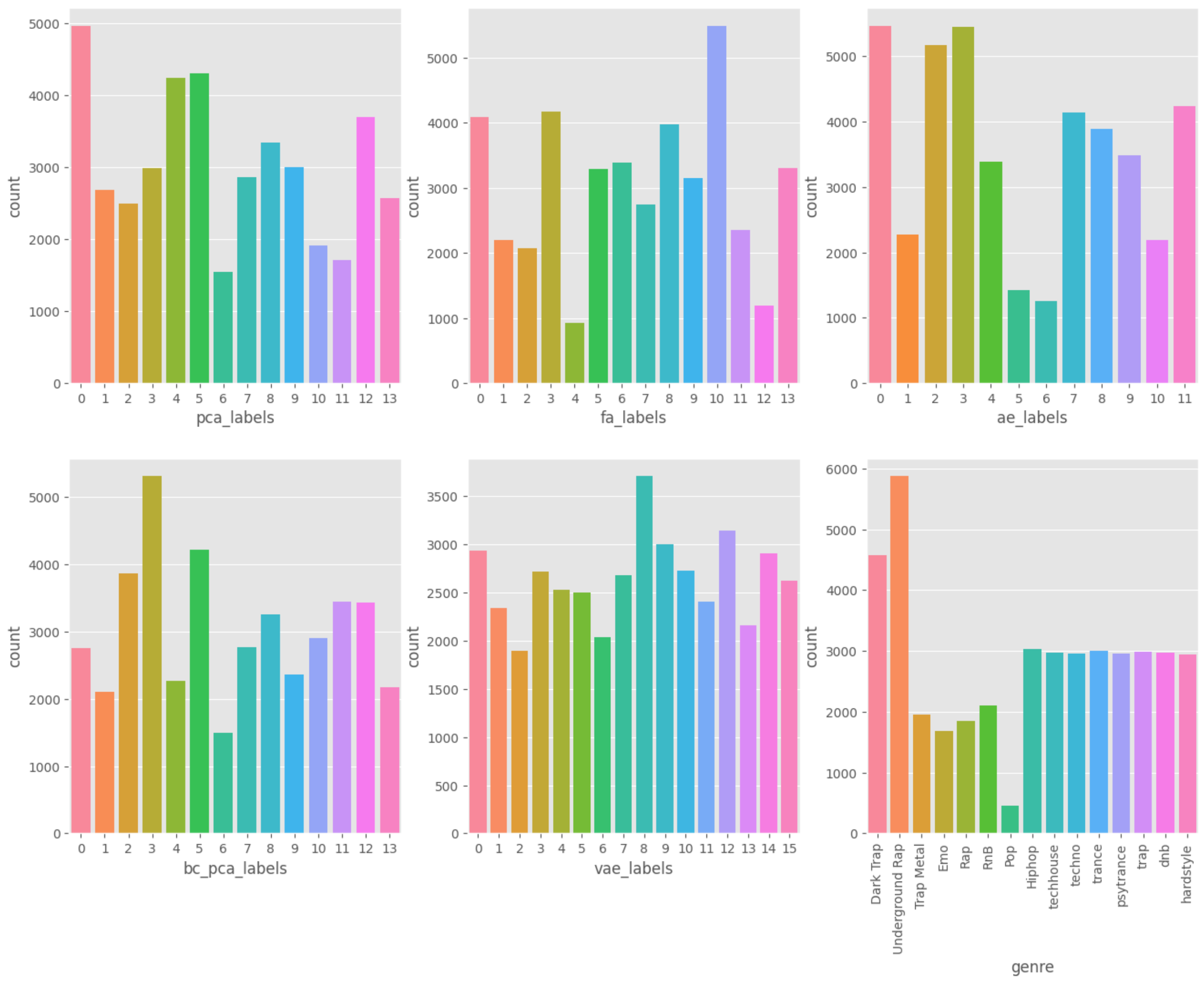
- PCA into KMeans
- FA Into KMeans
- Autoencoder Into GaussianMixture
- Box-Cox into PCA into GaussianMixture
- Variational Autoencoder into GaussianMixture

In addition to those of which above I have used these three methods to find similarities without any clustering but just different distance metrics between data points,

- Euclidian distance
- Cosine similarity distance
- Dot product

We use each of the three methods above to predict 3 recommendation for each of the user_input songs, then randomly selecting 10 of these recommendations, and for each method we'll have one csv file.

The final distribution of labels in each pipeline as well as the genres are shown below :



Conclusion

probably the most obvious observation we could derive from this assignment was that the curse of dimensionality strongly affects the act of clustering and beside that the linearity of our data set is an important factor to have in mind when using different dimension reduction techniques.

Also when building a recommender system we cannot scientifically have a criteria which could tell us if our clustering were good enough or not so as non supervised method itself we would need to have real time customer feedback to evaluate our clustering performance Although there are some metrics that could be used to tell us if our clustering had a good behavior or not which divide into two general evaluations one evaluates the internal integrity of a cluster and the other one evaluates the external distinguishability of each cluster meaning that a good clustering should be the one who has the most integrated clusters within and most distinguished clusters without MSE and Silhouette Index are two examples of these metrics