PTML: Analyse des streamers twitch et Méthodes de regroupement sur un dataset d'étoiles

Corentin Pion, Youri Colera

Sommaire

I. Analyse des streamers twitch
   A. Visualisation de la donnée
   B. Méthodes utilisées
      1. Régression linéaire
      2. Random forest
      3. Gradient boosting
II. Méthodes de regroupement sur un dataset d'étoiles
   A. Visualisation de la donnée
   B. Méthodes utilisées
      1. Kmeans
      2. Linkage
III. Conclusion

I. Analyse des streamers twitch

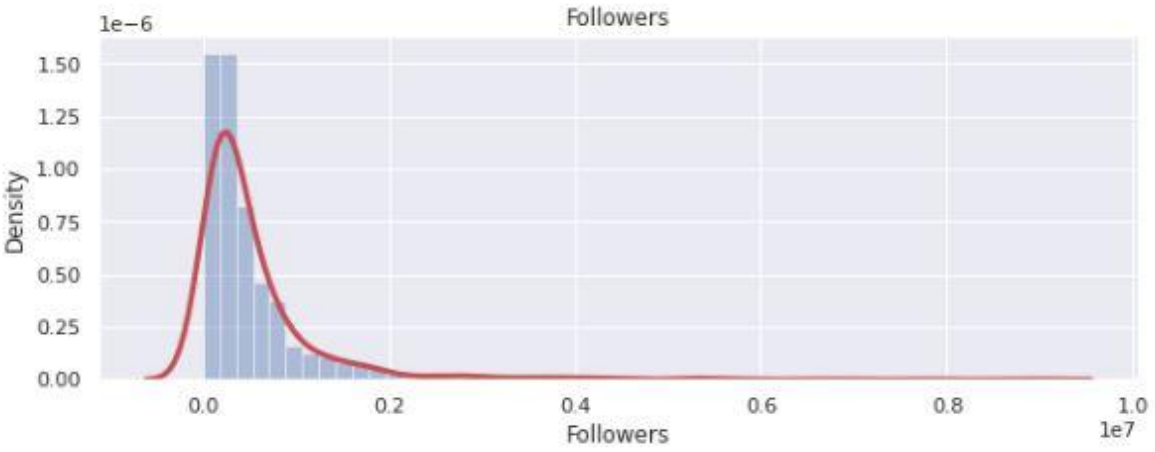
Concernant notre premier jeu de donnée, nous avons utilisé des données provenant de twitch (plateforme de streaming) sur les plus grands streamers de la plateformes (1000 personnes)
Le but d'utiliser ce dataset était d'analyser le gain de followers pour un utilisateur déjà important. Cette information pourrait être nécessaire pour simuler une rentrée d'argent et possiblement de planifier en amont une transition sur la plateforme à temps plein.

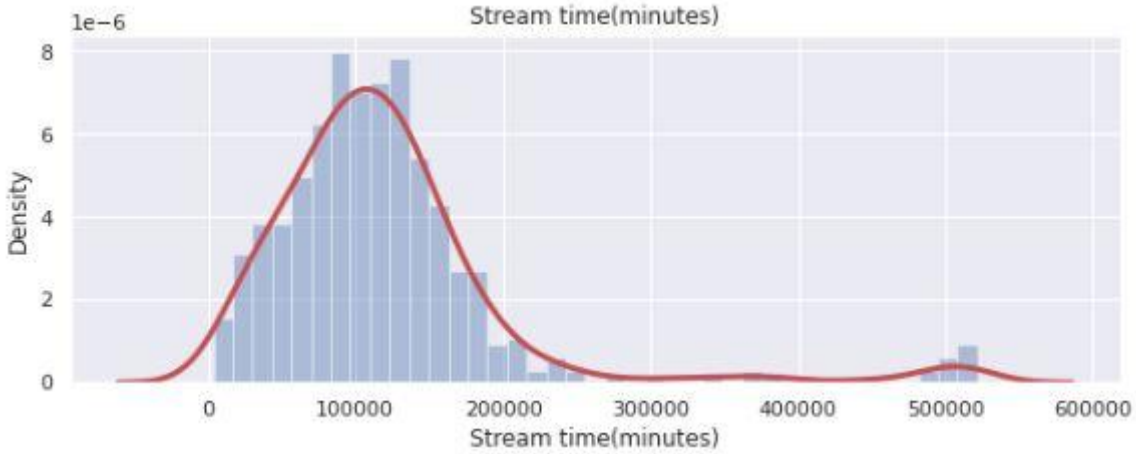
A. Analyse de la donnée

Premièrement nous avons sorti les différentes valeurs caractéristique de notre dataset :

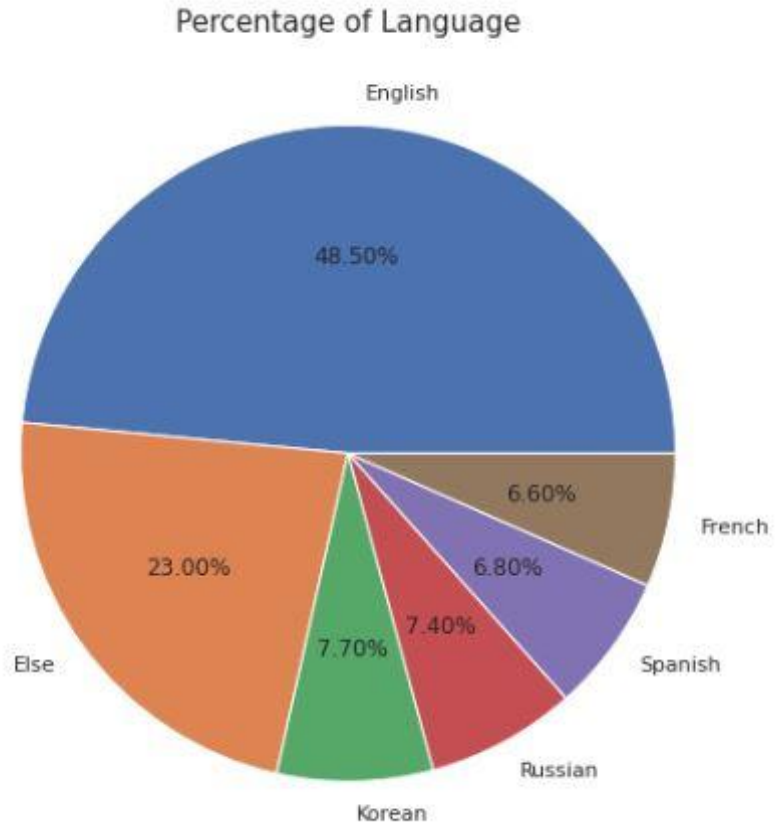
	id	Channel	Watch time(Minutes)	Stream time(minutes)	Peak viewers	Average viewers	Followers	Followers gained	Views gained	Partnered	Mature	Language
count	1000.000000	1000	1.000000e+03	1000.000000	1000.000000	1000.000000	1.000000e+03	1.000000e+03	1.000000e+03	1000	1000	1000
unique	NaN	1000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2	2	21
top	NaN	xQcOW	NaN	NaN	NaN	NaN	NaN	NaN	NaN	True	False	English
freq	NaN	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	978	770	485
mean	500.500000	NaN	4.184279e+08	120515.160000	37065.051000	4781.040000	5.700541e+05	2.055185e+05	1.166817e+07	NaN	NaN	NaN
std	288.819436	NaN	5.496355e+08	85376.201364	60314.307686	8453.684965	8.044134e+05	3.399137e+05	2.490572e+07	NaN	NaN	NaN
min	1.000000	NaN	1.221928e+08	3465.000000	496.000000	235.000000	3.660000e+03	-1.577200e+04	1.757880e+05	NaN	NaN	NaN
25%	250.750000	NaN	1.631899e+08	73758.750000	9113.750000	1457.750000	1.705462e+05	4.375825e+04	3.880602e+06	NaN	NaN	NaN
50%	500.500000	NaN	2.349908e+08	108240.000000	16676.000000	2425.000000	3.180630e+05	9.835200e+04	6.456324e+06	NaN	NaN	NaN
75%	750.250000	NaN	4.337399e+08	141843.750000	37569.750000	4786.250000	6.243322e+05	2.361308e+05	1.219676e+07	NaN	NaN	NaN
max	1000.000000	NaN	6.196162e+09	521445.000000	639375.000000	147643.000000	8.938903e+06	3.966525e+06	6.701375e+08	NaN	NaN	NaN

Suivant le type de nos colonnes, certaines statistiques ne seront pas répertoriées puisqu' elle n'ont aucune valeur ou elles ne sont justes pas calculables.
Pour avoir une plus nette représentation de chaque valeur numérique nous avons tracé leur histogramme à l'aide de la fonction "**distplot"** de seaborn. Cette fonction nous permet aussi d'obtenir le **"kernel density estimate".** En voici un exemple pour 2 colonnes, le reste étant dans le jupyter associé.



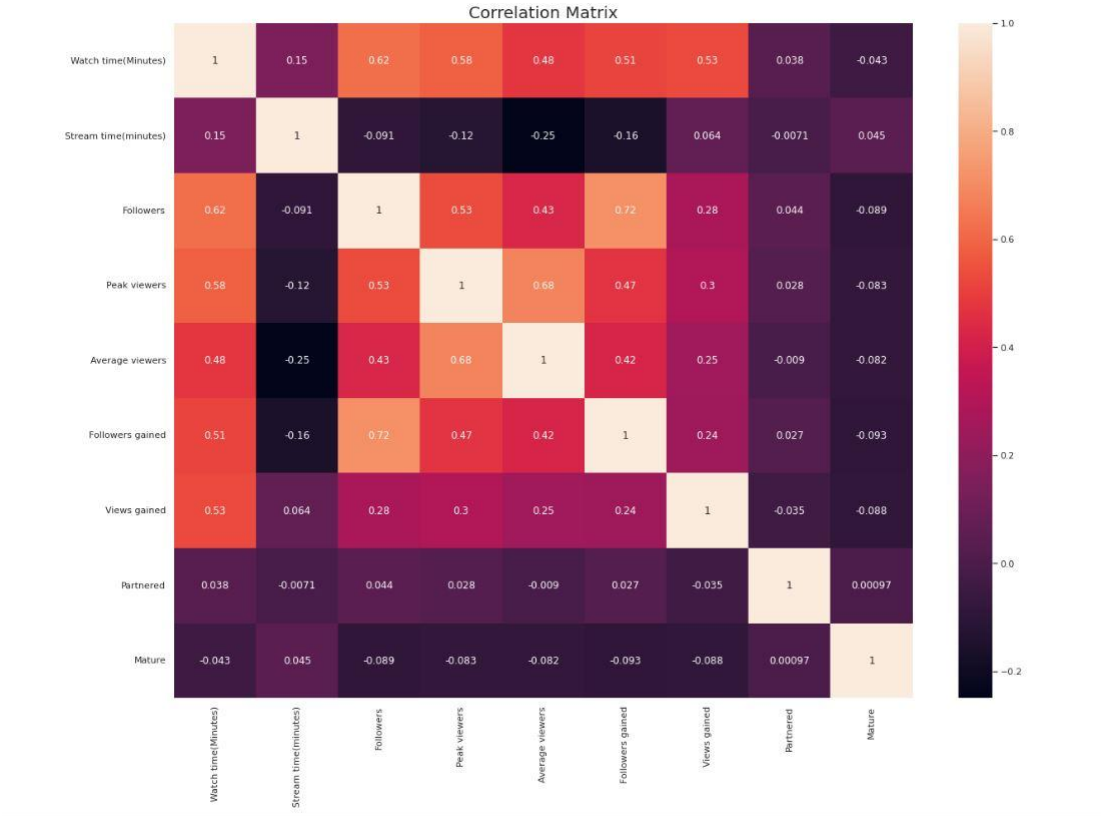


Pour finir sur la partie visualisation de nos données nous avons choisis plusieurs représentation pour la colonnes langues qui nous permet de nous rendre compte de l'importance des différentes valeurs:



Une autre façon de représenter notre graphique, fut d'utiliser un wordcloud, ce qui nous donne ceci:

Pour finir nous avons établi la matrice de corrélation pour essayer de trouver les variables les plus importantes pour notre régression:



On remarque que certaines colonnes sont bien corrélés ensembles (0,68 ; 0,72)

B. Méthodes utilisées

Preprocessing

Pour appliquer nos méthodes de régression, il faut d'abord prétraiter la donnée. Dans un premier temps nous supprimons les colonnes inutiles de notre dataset:

- La colonne 'Channel' ne sera pas utile pour prévoir le gain de follower. De plus, celle-ci est unique pour chaque streamer ce qui va forcément induire en erreur nos algorithmes (si le nom de chaîne est associé à un streamer connu, alors notre algorithme va forcément prévoir un gain de follower conséquent)
- Les colonnes 'Language', 'Partnered', 'Mature' n'ont pas d'impact sur le gain de followers d'après la matrice de corrélation

Enfin nous séparons la colonne 'Followers gained' de notre dataset puisqu'il s'agit de la valeur à prédire. On sauvegarde ce nouveau dataset dans la variable X puis le dataset contenant uniquement la colonne 'Followers gained' dans la variable y. Maintenant nous pouvons une nouvelle fois séparer notre dataset en deux pour avoir une partie d'entraînement et une partie pour tester nos résultats. On se retrouve donc avec 4 variables: X_train, X_test, y_train et y_test.

1. Régression linéaire

Pour commencer, nous utilisons la méthode la plus basique pour essayer de prévoir le gain de follower. Il s'agit de la régression linéaire. Nous avons décidé de laisser les paramètres par défaut pour nous donner un ordre d'idée sur la marge de progression que nous pouvons avoir. Sur la partie entraînement, nous pouvons voir que le r2 score est de 52% ce qui est plutôt faible. En revanche, sur la partie test, notre r2 score est de 67%.

2. Random forest regressor

Le random forest regressor consiste en la construction de plusieurs arbres de décision. Pour prévoir le gain de followers d'un nouveau streamer, il suffit de faire passer les informations dans l'ensemble des arbres de décisions qui ont été construits. On compte les décisions de chaque arbre, puis on détermine gain dans la partie qui a été le plus comptée. Le paramètre n_estimators représente le nombre d'arbres de décision que l'on souhaite construire. Nous avons décidé de laisser la valeur par défaut qui est de 100, car c'est avec cette valeur que nous obtenons les meilleurs résultats. Le paramètre "max depth" est la profondeur maximale d'un arbre. C'est avec une maxdepth de 3 que nous obtenons les meilleurs résultats. Le paramètre "max feature" est le nombre de caractéristiques à prendre en compte lors

de la recherche du meilleur split. Le paramètre auto qui correspond à “max_features=n_features” est le paramètre qui nous a permis d’atteindre les meilleurs résultats. Le paramètre “min sample split” correspond au minimum d’échantillons requis pour séparer un nœud interne. C’est avec un min sample split de 3 que nous obtenons les meilleurs résultats.

Sur la partie entraînement, nous pouvons voir que le r2 score est de 73.5% ce qui est plutôt satisfaisant. Enfin, sur la partie test, notre r2 score est de 74.2%. En utilisant la cross Validation sur 10 itérations, nous obtenons ce résultat:
Cross Validation: [0.27561129 0.36132888 0.66788129 0.54694341 0.19706004 0.51276179 0.40958683 -0.04267366 0.42044416 -0.04355774]

3. Gradient boosting regressor

Le gradient boosting regressor consiste en la construction d’un arbre de décision à taille fixe basé sur les erreurs de l’arbre précédent. Contrairement à AdaBoost, chaque arbre peut être plus grand qu’une souche (stump). Le paramètre learning rate réduit la contribution de chaque arbre par le taux d'apprentissage. Il existe un compromis entre le taux d'apprentissage et les n_estimateurs. Les valeurs doivent être comprises dans l'intervalle (0.0, inf). Un learning rate de 0.05 apporte les meilleurs résultats dans notre cas. Le paramètre n_estimators représente le nombre d'étapes à exécuter. Le boosting de gradient est assez robuste à l'ajustement excessif, donc un grand nombre donne généralement de meilleures performances. Les valeurs doivent être comprises dans l'intervalle [1, inf). un n_estimators de 90 apporte les meilleurs résultats dans notre cas. Le paramètre subsample représente la fraction d'échantillons à utiliser pour l'ajustement des apprenants de base individuels. S'il est inférieur à 1.0, il en résulte un Boosting de gradient stochastique. Subsample interagit avec le paramètre n_estimators. Le choix de subsample < 1.0 entraîne une réduction de la variance et une augmentation du biais. Les valeurs doivent être comprises dans l'intervalle (0,0, 1,0). un subsample de 0.9 apporte de meilleurs résultats.

Sur la partie entraînement, nous pouvons voir que le r2 score est de 87%. Sur la partie test, notre r2 score est de 74,15%. En utilisant la Cross Validation, nous obtenons ce résultat:
Cross Validation: [0.32274685 0.38407395 0.70045822 0.51549312 0.09895212 0.54477991 0.30978975 -0.09512956 0.52621568 -0.13063257]

II. Méthodes de regroupement sur un dataset

A. Analyse de la donnée

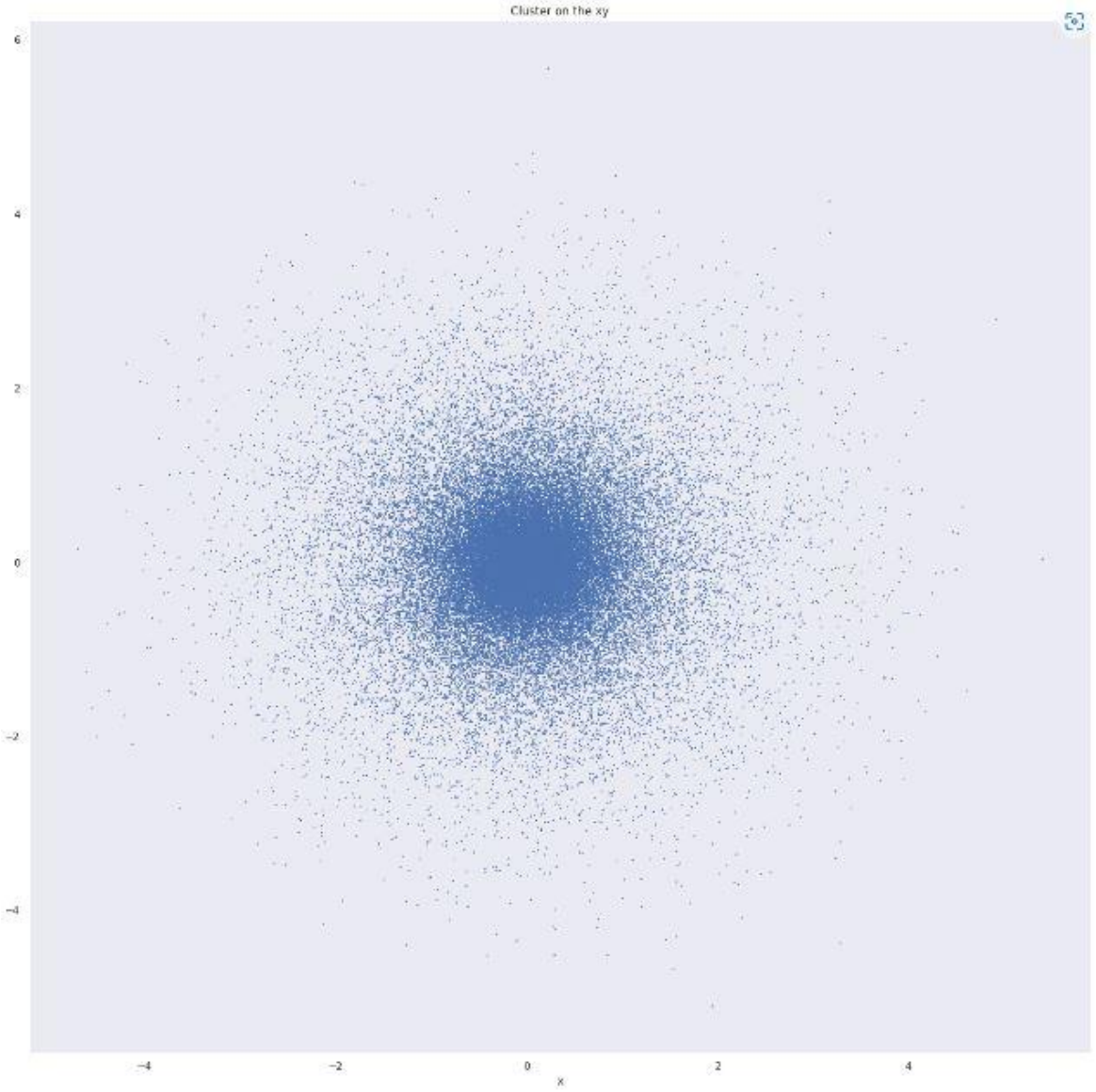
Concernant notre deuxième dataset nous allons utiliser un dataset qui simule la position d'étoiles. Le dataset est composé de plusieurs CSV qui chacun représente une position sur un plan pour un temps donné. Ce dataset nous permettra d'appliquer des algorithmes de clustering pour noter les différents groupes d'étoiles. A noter que le dataset créée peut-être utilisé principalement en physique pour essayer d'expliquer différents modèles a partir de données tel que leur vitesse, position, distance, force, etc mais nous ne nous intéresserons pas à modéliser ce problème.

Comme pour le premier dataset nous allons nous attarder sur les valeurs quantitatives au début. Nous obtenons ceci :

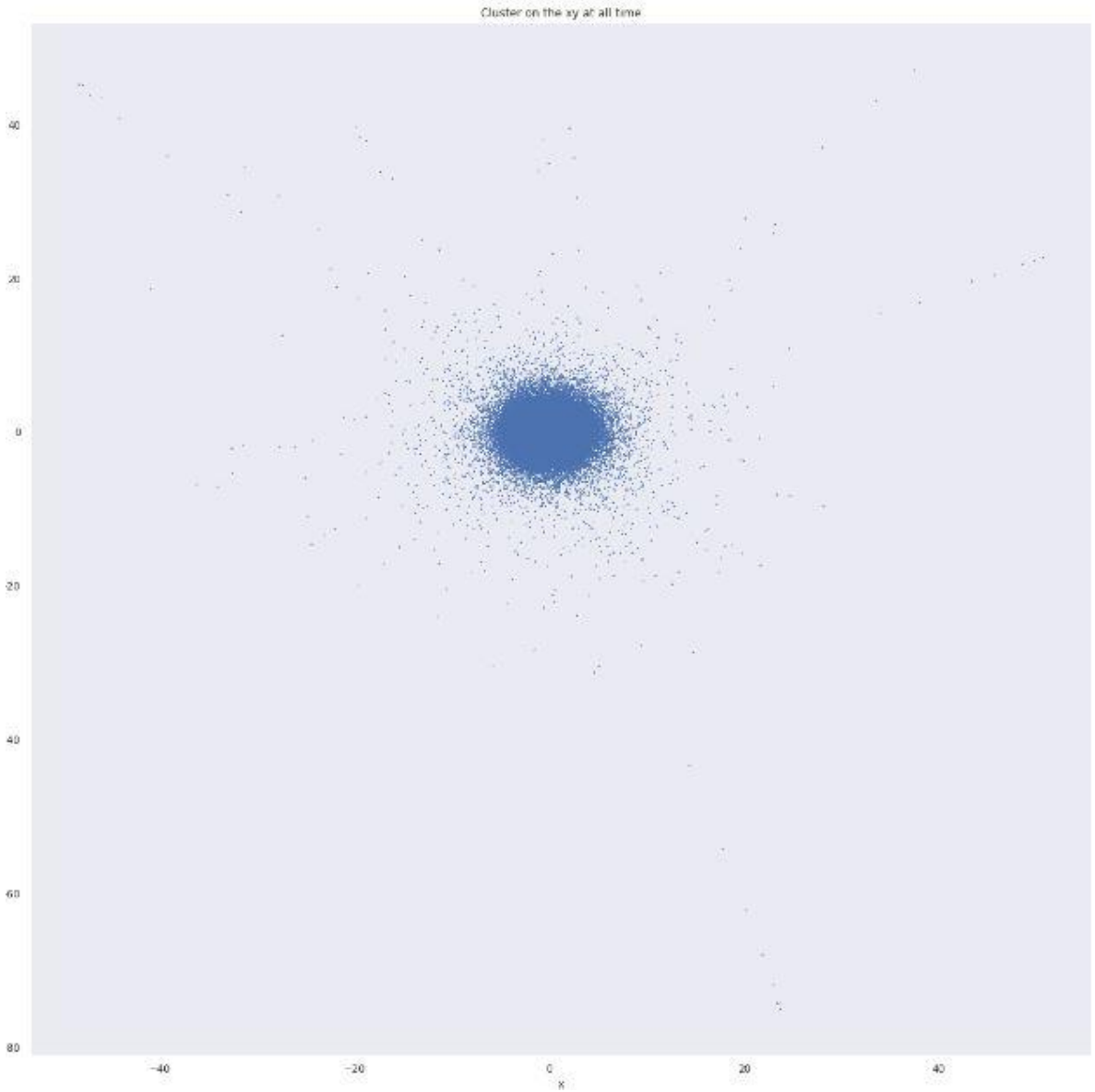
	x	y	z	vx	vy	vz	m	id
count	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	64000.000000
mean	-1.176595e-10	3.412734e-10	2.464757e-10	1.101956e-10	2.103394e-10	1.820625e-11	1.562500e-05	32000.500000
std	7.807475e-01	7.814718e-01	7.719907e-01	4.088337e-01	4.078873e-01	4.080323e-01	6.776317e-21	18475.352951
min	-4.698981e+00	-5.095714e+00	-5.015129e+00	-1.619203e+00	-1.482613e+00	-1.685246e+00	1.562500e-05	1.000000
25%	-3.348222e-01	-3.366134e-01	-3.356278e-01	-2.697400e-01	-2.714544e-01	-2.709821e-01	1.562500e-05	16000.750000
50%	-4.818202e-04	-4.321573e-04	-3.949339e-03	-2.090700e-04	1.419134e-03	1.770830e-04	1.562500e-05	32000.500000
75%	3.347058e-01	3.321014e-01	3.305091e-01	2.691137e-01	2.707544e-01	2.700553e-01	1.562500e-05	48000.250000
max	5.401047e+00	5.680950e+00	4.762590e+00	1.547319e+00	1.584202e+00	1.600681e+00	1.562500e-05	64000.000000

Concernant la visualisation des données, elles ont été créées suivant une loi normale comme on peut le constater sur les graphiques associés au jupyter de la partie 2.

Néanmoins pour répondre à notre problème de clustering nous avons décidé d'évaluer les différentes méthodes suivant 2 positions de chaque étoile. En effet, on se rend compte que certaines étoiles ont tendance à s'éloigner du lot d'étoiles comme le montre les 2 graphiques suivants. Le premier montre la position en x, y de chaque étoile à un temps 0



On constate bien que les étoiles sont très proches du centre de masse. Alors que lorsqu'on prend l'ensembles des coordonnées pour tous les temps on remarque que certaines étoiles s'éloignent

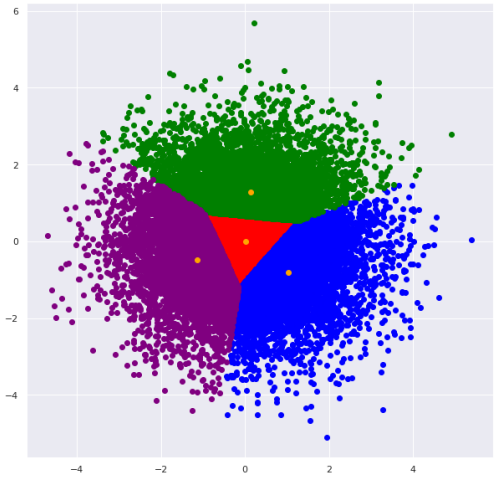


Pour essayer de comprendre la différence entre les 2 ont a voulu effectuer 2 clustering le premier a un temps t=0 et le deuxième a un temps t=15
Nous avons exprimé nos résultats sur un plan 2d x, y. En effet comme dans le jupyter le plan 3d n’est pas visuellement très explicite pour montrer nos différents clusters ainsi que les différences entre les 2 méthodes utilisées.

B. Méthodes utilisées

1. Kmeans

Kmeans est une méthode de clustering qui consiste à regrouper tous les éléments d'un dataset en k partie. Pour avoir un regroupement optimal, il faut que la distance entre tous les éléments d'un cluster soit minimale et que la distance entre chaque cluster soit maximale. Pour trouver le nombre de clusters optimal, on utilise la méthode du coude. Celle-ci consiste à mesurer la variance pour chaque k. La variance est la somme des distances entre chaque *centroid* d'un cluster et les différentes observations incluses dans le même cluster. Lorsque l'on augmente la valeur de k, la variance réduit, mais il existe un point à partir duquel la variance baisse beaucoup moins. C'est le point de coude. Il correspond à la valeur optimale de K. Dans notre cas, la valeur optimale de K sera de 4. L'inertie est une formule qui permet de donner une indication sur la cohérence des clusters. Nous pouvons voir qu'à partir d'un cluster on a bien un cohérence ce qui valide le choix de k = 4.
En lançant l'algorithme Kmeans avec une valeur de k = 4 nous obtenons le résultat suivant:

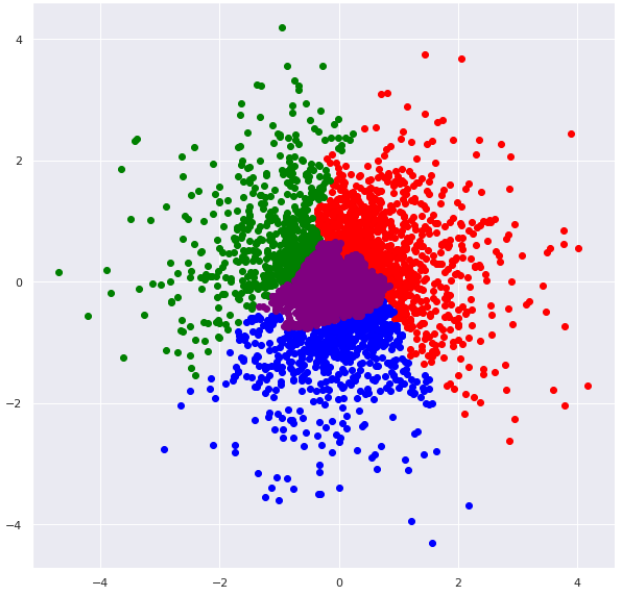


2. Linkage

Le clustering hiérarchique consiste en un regroupement pair à pair de deux objets les plus similaires entre eux. Nous avons décidé d'utiliser la méthode linkage de la bibliothèque scipy qui est une méthode de clustering hiérarchique. Le paramètre méthode permet de choisir la méthode de calcul des distance entre les nouveaux clusters u et v qui ont été formés. Nous avons décidé d'utiliser la méthode ward qui utilise l'algorithme de "Ward variance minimization". La nouvelle entrée est calculé comme ci-dessous:

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(u,s)^2 + \frac{|v|+|t|}{T}d(u,t)^2 - \frac{|v|}{T}d(s,t)^2}$$

où u est le nouveau cluster composé du cluster s et t, v étant un cluster non utilisé dans la forêt. En lançant l'algorithme de regroupement hiérarchique, nous obtenons le résultat suivant:



III. Conclusion

Nous avons vu au cours de ce projet qu'il était possible de faire des prédictions sur les gains de follower d'un streamer sur twitch avec un r2 score intéressant, avoisinant les 74% dans le meilleur des cas. Toutefois nous pouvons voir qu'il reste difficile pour nos méthodes d'être constantes, étant donné qu'il n'y avait pas assez de données (1000 streamer) ce qui nous a empêché d'avoir un r2 score plus élevé autour des 80-90%. Enfin nous avons pu voir, grâce à la matrice de corrélation que le gain de follower était fortement lié au nombre de followers (0,72) et légèrement lié au watch time (0,51), au pic de viewers (0,47) ainsi qu'au nombre de viewer en moyenne (0,42). Concernant le dataset sur les étoiles nous pouvons voir grâce à la méthode du coude que le meilleur moyen de regrouper celles-ci de manière efficace est d'utiliser 4 clusters. En revanche, il n'est pas possible de vérifier nos modèles étant donné que le dataset n'est pas labellisé.