- Handball Data Scientist -

L'ESBF, Entente Sportive Besançon Féminin, fait partie des meilleurs clubs de handball féminins de France depuis de nombreuses années. Engagé cette année sur de multiples tableaux (championnat de France, coupe de France, coupe d'Europe), le club a choisi d'étoffer son staff en engageant un expert analyste de données : Et c'est à vous que revient ce rôle !

L'objectif est d'évaluer les joueuses via l'outil informatique, que ce soit celles de l'ESBF, ou des adversaires ou des joueuses ciblées pour un prochain recrutement.

1 Préliminaire

Vous pouvez ouvrir le fichier `handball.py` *et compléter la trame proposée.*

▼ Script 1

Écrire une fonction `dist(x1,x2)` qui prend en argument deux listes de même taille et calcule la distance euclidienne entre les deux.

2 Note de Performance

Les handballeuses professionnelles du championnat de France possèdent des statistiques personnelles (taux de réussite au tirs, taux de présence sur le terrain, taux de duels gagnés,...) qui sont répertoriées dans un tableau de type array nommé `joueuses` de taille $N \times p$.

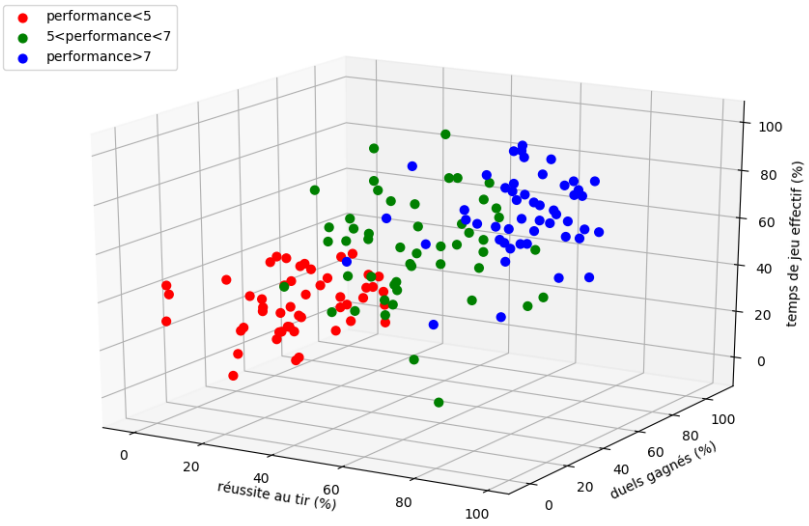
La i^e ligne correspond aux statistiques de la handballeuse numéro i(i est donc compris entre 0 et $N-1$) et chaque donnée est un nombre compris entre 0 et 100 (c'est un pourcentage).

Parallèlement, des experts(journalistes, sélectionneur national, cadres de la fédération) ont établi en début de saison une liste nommée `performances` de longueur N contenant une évaluation de chaque joueuse : la case numéro i correspond à une note (entre 0 et 10) relative à la performance globale de la handballeuse numéro i.

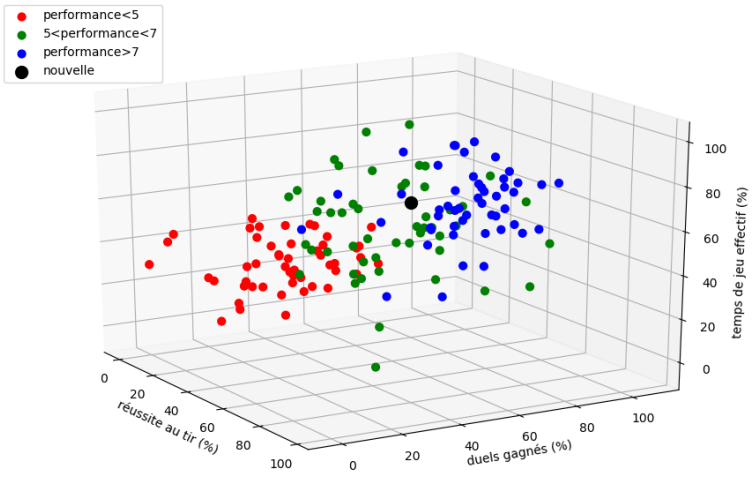
Cela permet d'établir un classement selon 3 catégories.

Vous pouvez ouvrir et valider le fichier `donnees.py` *contenant* `joueuses` *et* `performances`

A titre d'exemple de visualisation, voici représenté en trois dimensions le tableau `joueuses` selon $p = 3$ statistiques.
Chaque point correspond à une handballeuse, la couleur faisant référence à sa catégorie de performance.



Une joueuse espoir du club est sur le point de passer professionnelle.
On souhaite programmer l'algorithme des k plus proches voisins pour déterminer sa note de performance, à partir de ses statistiques. On visualise cette joueuse à l'aide du point noir.



▼ Script 2

Écrire une fonction `distance_au_point(joueuses,x)` qui prend en argument le tableau contenant les statistiques des handballeuses et une liste de longueur p contenant les statistiques de la nouvelle joueuse `x`, et qui renvoie une liste `distances` de longueur N où la i^e case contient la distance de x à la handballeuse numéro i. Quel est son rôle/objectif ?

▼ Question 3

Lire le code de la fonction `trier(distances)` suivante et la tester. (`distances` désigne une liste de valeurs)

```
1 def trier(distances):
2     n=len(distances)
3     def f(i):
4         return distances[i]
5     return sorted(range(n),key=f)
```

Réponse : cette fonction prend en argument une liste de nombres et renvoie une liste contenant les indices de la liste rangées par ordre croissant.

Par exemple pour `distances=[5,9,1]`, alors `trier(distances)` renvoie `[2,0,1]` puisque la case d'indice 2 est le plus petit élément de distance, puis c'est la case d'indice 0 et enfin celle d'indice 1

▼ Script 4 (facultatif)

Écrire votre propre fonction `k_proche(distances, k)` qui prend en argument la liste des distances et un entier `k` et qui renvoie la liste des indices des `k` plus proches voisins.

Quel est l'avantage par rapport à l'utilisation d'une fonction de tri ?

▼ Script 5

Écrire une fonction `moyenne_des_k_voisins(indices_tries,performances,k)` prenant en argument la liste des observations triées par distance croissante à `x`, la liste des notes des handballeuses et le nombre de voisins à considérer, et qui renvoie la moyenne des notes des k plus proches voisins de x.

▼ Script 6 En déduire une fonction `note_nouvelle(joueuses,performances,x,k)` qui renvoie la note de performance de la joueuse espoir `x`.

Tester votre fonction avec la liste `x` proposée. On pourra faire varier le paramètre k.

3 Groupes de Performance

Après une saison complète, les statistiques des joueuses ont évolué, certaines ont pris leur retraite sportive et des espoirs sont passées professionnelles. Les données des handballeuses professionnelles ont été actualisées et figurent désormais dans le tableau `saison2` (lui aussi présent dans le fichier `donnees.py` validé initialement). C'est toujours un tableau de type `array` de taille $M \times p$, la i^e ligne correspondant aux statistiques personnelles de la handballeuse numéro i.

Cependant, la liste de performance doit être remise à plat.

Le coach vous demande alors de répartir les handballeuses selon k groupes en utilisant uniquement les données objectives figurant dans `saison2`.
L'algorithme des k-moyennes consiste à fixer k pôles aléatoirement, à étiqueter chaque handballeuse selon le pôle le plus proche, de calculer les valeurs des nouveaux pôles comme barycentres des groupes, et de recommencer jusqu'à ce que les pôles ne bougent plus.

Les caractéristiques des pôles seront stockées dans un tableau de type `array` appelé `poles` de taille $k \times p$.

▼ Script 7

Pour fixer les valeurs initiales des k pôles, on procède de la manière suivante : pour chacun des k pôles, pour chacune de ses p coordonnées, on choisit une valeur aléatoire uniformément répartie entre la valeur minimale et la valeur maximale de cette coordonnée calculée sur l'ensemble des M données.

Ecrire une fonction `initialisation_poles(saison2,k)` qui génère le tableau `poles` initial.

On pourra utiliser les fonction `min` et `max` de Python ainsi que la fonction `uniform(a,b)` du package `random` qui renvoie un nombre aléatoire compris entre a et b.

▼ Script 8

Après l'étape d'initialisation, il faut affecter chaque observation à son pôle le plus proche.

Ecrire une fonction `pole_le_plus_proche(poles,x)` qui prend en argument le tableau contenant les coordonnées des poles et une liste `x` contenant les caractéristiques d'une joueuse et qui renvoie l'indice du pôle le plus proche de `x`.

▼ Script 9

Ecrire une fonction `poles_les_plus_proches(poles,saison2)` qui renvoie une liste nommée `repartition`, dont la i^e case contient l'indice du pôle le plus proche de la joueuse numero i.

▼ Script 10

La troisième étape consiste à recalculer les coordonnées des pôles pour les placer au barycentre de leur catégorie. Autrement dit, on affecte au pôle d'indice ℓ la moyenne des observations qui ont été catégorisées comme étant rattachée/proche du pôle d'indice ℓ.

Ecrire une fonction `nouveaux_poles(saison2,repartition,k)` qui renvoie un tableau `poles` contenant les coordonnées des barycentres de chaque catégorie.

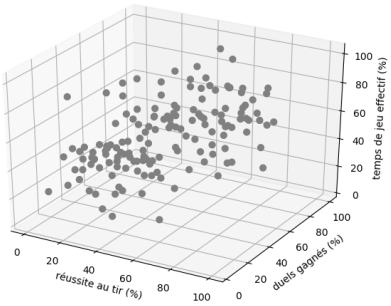
▼ Script 11

En déduire une fonction `k_moyennes(saison2,k)` qui renvoie la matrice `poles` correspondant aux k classes trouvées par l'algorithme des k-moyennes.

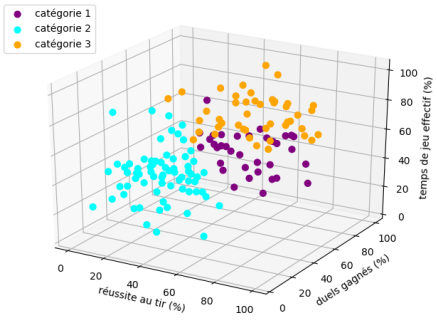
un peu d'aide : lorsque A et B désignent deux tableaux de même taille, la commande `(A==B).all()` renvoie `True` lorsque les coefficients des tableaux A et B sont (tous) égaux terme à terme et renvoie `False` sinon (i.e. lorsqu'il existe une coordonnée où le coeff de A et différent de celui de B

▼ Question 12

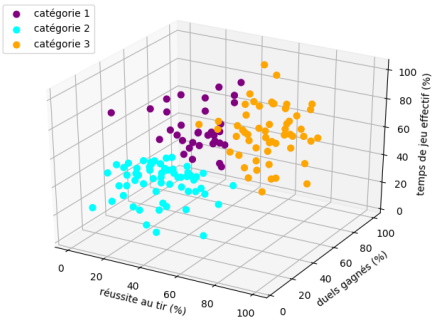
On représente le tableau `saison2` puis, après avoir exécuté la fonction `k_moyennes`, on colore chaque point de façon à faire apparaitre les catégories. Voici ci-dessous les résultats obtenus par vous et votre voisin(e).



`saison2`



résultat de l'algorithme des k-moyennes avec $k = 3$, étudiant(e)1



résultat de l'algorithme des k-moyennes avec $k = 3$, étudiant(e)2

Pourquoi ces sont-ils différents ? Peut-on savoir si un des résultats est plus pertinent que l'autre ?
Le coach sera-t-il satisfait ?

▼ Bonus

Combien, en moyenne, d'itérations sont nécessaires lors de la mise en place de l'algorithme des $k-$moyennes ?