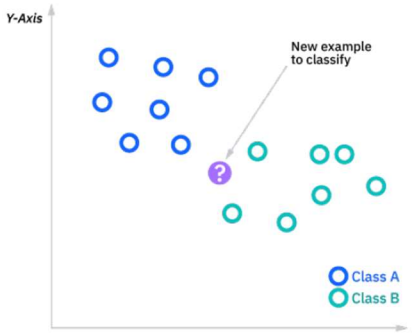
Introduction :

Dans ce TP nous allons nous intéresser à la méthode dite des « ***des k plus proches voisins*** » (en anglais « *k nearest neighbors* » ; d'où le sigle KNN).

Le principe est assez simple, on dispose d'un ensemble de données, chaque donnée dispose de **deux paramètres numériques que l'on utilisera en tant qu'abscisse et ordonnée.** (**Exemples :** taille, poids, couleur, …).
Cela nous permettra de comparer les différentes données via ces paramètres, et notamment **les distances** qui séparent les différents points dans une représentation graphique.

Mais il ne faut pas perdre de vue le but initial de l'algorithme qui est de **classifier les données de l'ensemble** de manière optimisée !
C'est pour cela qu'intervient un troisième critère destiné à la classification d'un élément (**Exemples :** espèces de fleurs, poste des joueurs dans une équipe, …).

Représentation :



*On cherche à **classifier un nouvel élément.***

*On calcule **les distances euclidiennes qui l'entoure.***

*On en retire qu'elle est **la classe majoritaire dans son entourage proche** (en l'occurrence ici c'est la classe B)*

Cas Pratique :

Nous allons maintenant nous atteler à la **conception d'un programme** qui indexera la taille et le poids des différents joueurs du Top14 ainsi que leur poste sur le terrain au cours de la saison 2019/2020.
Nous nous retrouvons donc bien avec nos **deux types d'informations** :

- Deux données numériques destinées à la comparaison de deux éléments de l'ensemble : **Taille** (*x*) et **Poids** (*y*)
- Un critère destiné à la classification d'un élément : **Poste du joueur**

L'objectif est tout d'abord de **représenter cet ensemble dans un graphique**, puis, par la suite, l'utilisateur sera invité à **saisir une taille et un poids arbitraire** dans le but de **créer un nouvel élément** que l'algorithme KNN devra parvenir à **classifier dans le poste le plus approprié à la morphologie saisie** !

Programmation :

Afin de rendre plus compréhensible le programme, nous allons décomposer l'algorithme en plusieurs blocs/fonctions :

1. Importation des librairies et dépendances :

Chaque module est utilisé dans un but bien précis.

```
1 import pandas                          # Manipulation du fichier csv
2 import math                            # Calcul des distances
3 import matplotlib.pyplot as courbe     # Représentation graphique
```

2. Extraire toutes les données du fichier CSV :

Dans cette fonction on récupère, tout d'abord, absolument toutes les données de la base dans un « DataFrame", puis dans un second temps, une liste de tous les descripteurs du fichier.

```
1 def extractionDonnees(nomFichier='JoueursTop14.csv'):
2     """Cette fonction récupère les données d'un fichier csv et renvoie deux
valeurs :
3     La liste des descripteurs et un "DataFrame" de toutes les données"""
4     players = pandas.read_csv(nomFichier, sep=';', encoding='utf-8')
5
6     descripteurs = list(players.columns)    # Je récupère les descripteurs.
7
8     return (players, descripteurs)          # On retourne un Tuple
contenant les deux ensembles de données.
```

3. Extraire seulement les données sur une équipe choisie :

Dans cette fonction, on localise dans la colonne « Equipe » les lignes qui comportent la valeur exacte de l'équipe choisie par l'utilisateur.
(**Exemples :** Toulouse, Racing92, …).
Si l'utilisateur choisi plutôt d'étudier **tous** les joueurs du Top14, alors la fonction n'est pas exécutée car on manipulera par la suite l'ensemble des données !

```
 1 def extraireEquipe(players, choix_equipe):
 2     """De l'ensemble des listes, on extrait seulement celles d'une équipe
 3     Parmi les équipes, on trouve "Agen", "Bayonne", "Bordeaux", "Brive",
"Castres", "Clermont",
 4     "La Rochelle", "Lyon", "Montpellier", "Paris", "Pau", "Racing92",
"Toulon" et "Toulouse"
 5     On peut aussi écrire "Tous" pour avoir tous les joueurs du top 14"""
 6
 7     precise_play = players.loc[(players["Equipe"] == choix_equipe)]
 8
 9     return precise_play
10
11 equipes_disponibles = ["Agen", "Bayonne", "Bordeaux", "Brive", "Castres",
"Clermont", "La Rochelle",
12                        "Lyon", "Montpellier", "Paris", "Pau", "Racing92",
"Toulon", "Toulouse"]
```

```
13
14 team = str(input('Choisissez votre équipe : \n'))   # On interroge
l'utilisateur sur l'équipe qu'il souhaite étudier !
15
16 if team == 'Tous' :    # Possibilité d'étudier tous les joueurs du Top 14
17
18     data_team = extractionDonnees()[0].values.tolist()    # On convertis
le DataFrame entier en liste.
19
20 elif team in equipes_disponibles :  # Vérification que l'équipe choisie
est bien dans la liste disponible.
21
22     data_team = extraireEquipe(extractionDonnees()[0], team)  # On extrait
seulement les joueurs de l'équipe choisie.
23     data_team = data_team.values.tolist()      # On convertis le DataFrame
de l'équipe en liste.
24
25 else :
26     print("Désolé, cette équipe n'est pas disponible...")
27     exit()                                  # On stoppe l'exécution du code.
```

4. Représenter graphiquement le nuage de points :

Dans cette partie nous allons utiliser la librairie Python « MatplotLib » qui va nous permettre de réalise un nuage de points avec la taille des joueurs en abscisse et leurs poids en ordonnée !

```
 1 def representation(data,descripteurs):
 2     global team
 3     """data est une liste de joueurs avec leurs caractéristiques.
 4     On extrait leur taille et poids puis on représente ces données dans un
repère
 5     (une couleur par type de poste la position étant déterminée par la
liste des descripteurs)
 6     Les types de postes sont "Avant", "2ème ligne", "3ème ligne", "Demi",
"Trois-Quarts" et "Arrière" """
 7
 8     taille = []
 9     poids=[]
10     poste=[]
11
12     # On récupère tous les abscisses et les ordonnées des points (ainsi
que leurs postes correspondants) :
13     for i in range(len(data)) :
14         val_temp = data[i]
15         taille_temp = val_temp[descripteurs.index('Taille (en cm)')]
16         poids_temp = val_temp[descripteurs.index('Poids (en kg)')]
17         poste_temp = val_temp[descripteurs.index('Type poste')]
18         taille.append(taille_temp)
19         poids.append(poids_temp)
20         poste.append(poste_temp)
21
22     # Création de la courbe :
23     title = "Analyse de l'équipe de rugby de " + team
24     courbe.title(title, fontsize=18)
25     courbe.grid(color = 'green', linestyle = '--', linewidth = 0.5)
26     courbe.xlabel('Taille (en cm)')
```

```
27      courbe.ylabel('Poids (en kg)')
28      for i in range(len(data)) :
29          if poste[i] == 'Avant' :
30              point1 = courbe.scatter(taille[i], poids[i], marker="x",
edgecolors="blue", c="blue")
31          if poste[i] == '2ème ligne' :
32              point2 = courbe.scatter(taille[i], poids[i], marker="+",
edgecolors="red", c="red")
33          if poste[i] == '3ème ligne' :
34              point3 = courbe.scatter(taille[i], poids[i], marker="1",
edgecolors="green", c="green")
35          if poste[i] == 'Demi' :
36              point4 = courbe.scatter(taille[i], poids[i], marker="o",
edgecolors="purple", c="purple")
37          if poste[i] == 'Trois-Quarts' :
38              point5 = courbe.scatter(taille[i], poids[i], marker="*",
edgecolors="brown", c="brown")
39          if poste[i] == 'Arrière' :
40              point6 = courbe.scatter(taille[i], poids[i], marker="^",
edgecolors="orange", c="orange")
41
42      # Création de la légende :
43      courbe.legend([point1, point2, point3, point4, point5,
point6],['Avant','2ème ligne','3ème ligne','Demi','Trois-Quarts','Arrière'],
shadow=True, title=team, title_fontsize=15, loc='lower right')
44
45      return (taille, poids, poste)
46
47 # Appel de la fonction :
48 data_for_search = representation(data_team, extractionDonnees()[1])
```

5. Balayage et calcul des distances entre le point de l'utilisateur et ceux des joueurs :

Via les deux fonctions suivantes nous allons calculer les différentes distances euclidiennes, entre le point saisi par l'utilisateur et tous les points des joueurs.
On rappelle la formule pour calculer une distance euclidienne :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Par la suite on enregistrera toutes ces distances dans une liste avec la forme suivante :
« ***list_ecart*** = [[distance entre les deux points, poste du joueur], [distance entre les deux points, poste du joueur], etc…] ».

```
 1 def distance(x1,y1,x2,y2):
 2     """Distance euclidienne entre les points de coordonnées (x1;y1) et
(x2;y2)"""
 3     result_dis = math.sqrt((x2-x1)**2+(y2-y1)**2) # Formule
 4     return result_dis
 5
 6 def balayage_points(data_graph, taille_user, poids_user):
 7     """Calcul des distances entre le point de l'utilisateur et les points
des différents postes"""
 8     taille_x = data_graph[0]
 9     poids_y = data_graph[1]
```

```
    poste_val = data_graph[2]

    list_ecart = list()

    # On calcule la distance pour chaque élément :
    for element in range(len(taille_x)) :
        verdict =
distance(taille_user,poids_user,taille_x[element],poids_y[element])

        # Puis on ajoute la distance ainsi que le poste actuellement
étudié :
        list_ecart.append([verdict, poste_val[element]])

    return list_ecart

# Ajout du point de l'utilisateur :
taille_saisie = int(input("Saisissez votre taille en cm :\n"))
poids_saisie = int(input("Saisissez votre poids en kg :\n"))

list_distances = balayage_points(data_for_search, taille_saisie,
poids_saisie)
```

6. Classifier les distances en fonction des postes :

Grâce à la fonction suivante, **on classifie les distances en fonction du poste lié à ces dernières.**
Le principe final va être de faire une **moyenne des distances pour chacun des postes**, cela nous permettra de savoir quel poste à **la distance moyenne** (avec tous ses points), **la plus faible**, cela signifiera qu'il s'agit du **poste le plus proche du point de l'utilisateur, et par conséquent de sa morphologie !**

Pour calculer nos moyennes, on utilise la **méthode classique** :

- On **additionne toutes les distances d'un même poste.**
- On compte le **nombre d'occurrence** d'un **même poste** dans la **liste entière.**
- On divise la première somme par la seconde, et voilà **on a notre moyenne des distances entre le point de l'utilisateur et chaque poste !**

```
def classification(data_ecart):
    sum_avant = 0
    nb_avant = 0
    sum_2emeligne = 0
    nb_2emeligne = 0
    sum_3emeligne = 0
    nb_3emeligne = 0
    sum_demi = 0
    nb_demi = 0
    sum_trois_quarts = 0
    nb_trois_quarts = 0
    sum_arriere = 0
    nb_arriere = 0

    for k in range(len(data_ecart)) :
        if data_ecart[k][1] == 'Avant' :
            sum_avant += data_ecart[k][0]
```

```
            nb_avant += 1
        elif data_ecart[k][1] == '2ème ligne' :
            sum_2emeligne += data_ecart[k][0]
            nb_2emeligne += 1
        elif data_ecart[k][1] == '3ème ligne' :
            sum_3emeligne += data_ecart[k][0]
            nb_3emeligne += 1
        elif data_ecart[k][1] == 'Demi' :
            sum_demi += data_ecart[k][0]
            nb_demi += 1
        elif data_ecart[k][1] == 'Trois-Quarts' :
            sum_trois_quarts += data_ecart[k][0]
            nb_trois_quarts += 1
        elif data_ecart[k][1] == 'Arrière' :
            sum_arriere += data_ecart[k][0]
            nb_arriere += 1

    list_of_all_avg = list()

    avg_avant = sum_avant / nb_avant
    list_of_all_avg.append(['Avant', avg_avant])
    avg_2emeligne = sum_2emeligne / nb_2emeligne
    list_of_all_avg.append(['2ème ligne', avg_2emeligne])
    avg_3emeligne = sum_3emeligne / nb_3emeligne
    list_of_all_avg.append(['3ème ligne', avg_3emeligne])
    avg_demi = sum_demi / nb_demi
    list_of_all_avg.append(['Demi', avg_demi])
    avg_trois_quarts = sum_trois_quarts / nb_trois_quarts
    list_of_all_avg.append(['Trois-Quarts', avg_trois_quarts])
    avg_arriere = sum_arriere / nb_arriere
    list_of_all_avg.append(['Arrière', avg_arriere])

    return list_of_all_avg

# Appel de la fonction :
predictions = classification(list_distances)
```

7. Trier la liste en fonction des distances moyennes :

Étant donné qu'il s'agit d'une liste de liste, la méthode de tri est différente de celle habituellement utilisée…
Le paramètre que va nous servir de classement est la moyenne, située en index 1 de chaque sous-liste, en index 0 se trouve le poste correspondant.

Nous allons donc classer les moyennes des postes de manière croissante, en premier se situera donc le poste le plus approprié pour la morphologie saisie par l'utilisateur ! En bref, cela sera **le poste le plus proche graphiquement ET littéralement** de la taille et du poids saisies !

```
def best_post(predi):
    predi.sort(key = lambda x: x[1])

    return predi

predictions = best_post(predictions)

```

```
print("\nPredictions :",predictions,"\n")

# Il s'agit donc du premier poste de la liste, d'où l'index [0][0] :
print("Le meilleur poste pour vous dans l'équipe de",team,"est
:",predictions[0][0],"\n")
```

8. Afficher la représentation graphique finale (+ point de l'utilisateur) :

Cette méthode **rajoute** dans un premier temps **le point saisi par l'utilisateur avec la taille et le poids en coordonnées.**
Il **enregistre une image du graphique.**
Et enfin il l'**affiche sur le système d'exploitation de l'utilisateur**, dans une nouvelle fenêtre !

```
def add_point_user(taille_user, poids_user):
    courbe.scatter(taille_user, poids_user, marker="X", edgecolors="black",
c="black")
    courbe.text(taille_user+0.2, poids_user+0.2, 'Vous')

    courbe.savefig('representation_graphique.png')

    courbe.show()

add_point_user(taille_saisie, poids_saisie)
```

Résultats :

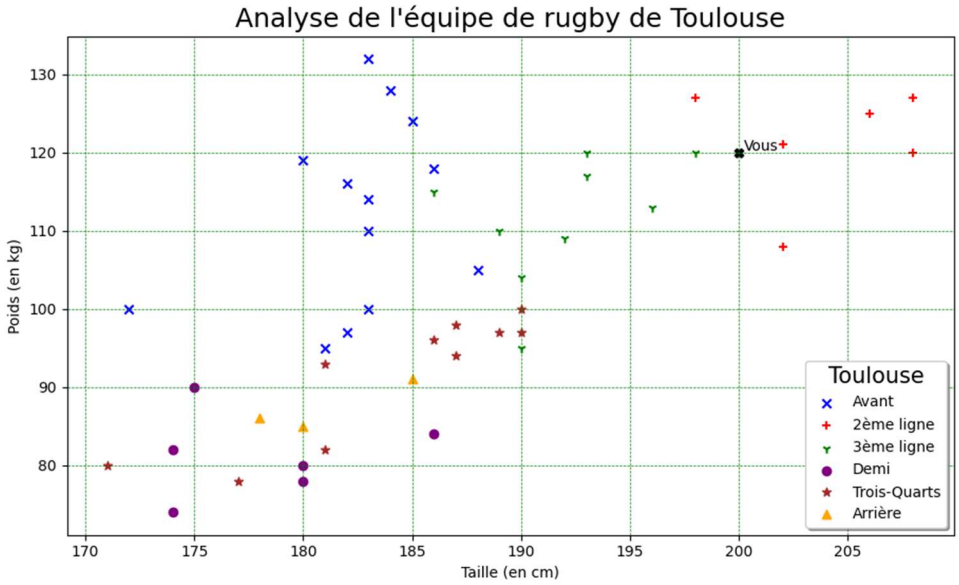
Nous avons choisi une taille de 200cm ainsi qu'un poids de 120kg. Voyons les résultats !

- ## Avec l'équipe de « Toulouse » :

Choisissez votre équipe :
Toulouse
Saisissez votre taille en cm :
200
Saisissez votre poids en kg :
120

Predictions : [['2ème ligne', 8.02034973600301], ['3ème ligne', 12.645047239702293], ['Avant', 21.927130980224206], ['Trois-Quarts', 32.81354826100172], ['Arrière', 37.819285879584974], ['Demi', 44.633445691168994]]

Le meilleur poste pour vous dans l'équipe de Toulouse est : **2ème ligne**



- **Avec le Top14 entier :**

Choisissez votre équipe :
Tous
Saisissez votre taille en cm :
200
Saisissez votre poids en kg :
120

Predictions : [['2ème ligne', 7.4939134019794365], ['3ème ligne', 14.759575816768264], ['Avant', 19.576766409826828], ['Trois-Quarts', 30.542780530800837], ['Arrière', 35.70250458187474], ['Demi', 41.967812668170694]]

Le meilleur poste pour vous dans l'équipe de Top14 entier est : **2ème ligne**