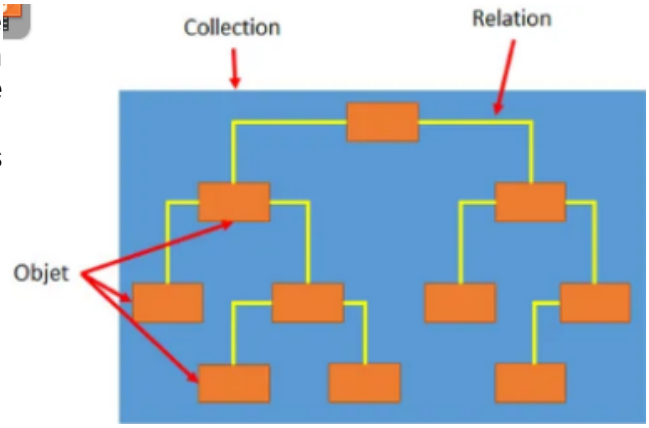
OPTIMISATION 4

Leçon : les arbres de décision

Les arbres de décision (AD) représentent une méthode de classification et permettent des prédictions ou des décisions.

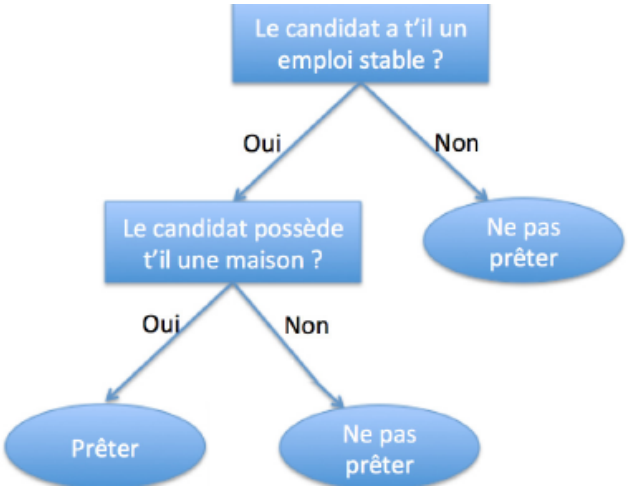
Ils structurent une collection d'objets de manière arborescente. Une structure courante est un arbres binaire. Chaque élément de la structure peut avoir un parent et de 0 à deux enfants.
Un élément de la structure relié à plusieurs autres éléments est appelé un **nœud (node)**.
L'élément de départ de la structure est appelé le parent ou la **racine (root).**
Chaque élément final sans enfant est appelé une **feuille (leaf).**
Le mot binaire signifie que chaque nœud a au plus 2 enfants.



Dans un arbre de décision, chaque nœud représente un test sur une composante d'un vecteur de données. Chaque feuille, appelée une classe, est atteinte après le parcours d'une partie de l'arbre c'est à dire après plusieurs tests. Chaque donnée d'observation qui doit être placée dans une classe est décrite sous forme de vecteur où chaque composante est la valeur d'une variable. Pour être classée, la donnée d'observation parcourt l'arbre en partant de la racine puis ses composantes sont testées à chaque nœud de l'arbre, ce qui détermine un parcours de la donnée dans l'arbre jusqu'à sa terminaison dans une feuille. A chaque test, selon le résultat, l'observation est dirigée vers le nœud enfant gauche ou droit qui lui-même effectuera le test suivant puis dirigera l'observation vers le nœud suivant … jusqu'à la feuille.
Les arbres de décision représentent une méthode de classification et permettent les prédictions ou décisions.

Exemple : un arbre de décision simplifié pour un prêt bancaire
Chaque individu est une observation qui parcourra l'arbre pour être classé : éligible ou non au prêt bancaire.
Dans l'arbre simplifié ci-contre, un dossier est réduit à 2 données qui seront testées : emploi stable et propriétaire. La donnée d'observation est un vecteur à deux composantes.

Chaque dossier parcourt l'arbre en fonction des réponses aux questions de chaque nœud pour terminer dans une feuille. La feuille est associée à la décision de prêter ou non de l'argent.



Exercice 1 : construction manuelle d'un arbre à critères fixés

Dans ce TD, on considère un jeu de données créé par Ronald Fisher, biologiste, statisticien. Les données caractérisent 3 variétés différentes de fleurs particulières : des iris. Pour chaque fleur, 4 grandeurs ont été mesurées : longueur des sépales, largeur des sépales, longueur des pétales et largeur des pétales. Toutes les mesures sont exprimées en cm. Nous allons utiliser ces données pour modéliser les variétés, les différencier et prédire la variété d'une fleur à partir de ces mesures.

Chaque fleur est caractérisée par une liste de 4 valeurs (un vecteur de dimension 4), chacune indiquant la valeur d'une grandeur mesurée dans l'ordre numéroté de 0 à 3 : longueur des sépales, largeur des sépales, longueur des pétales et largeur des pétales.
Les mesures sont accessibles dans l'objet iris du module Python sklearn.datasets qui contient les données de 150 fleurs. Les données et les variétés de chacune des fleurs sont accessibles dans les attributs de l'objet iris.

Par exemple :

```
>>> import sklearn.datasets
>>> exemple = sklearn.datasets.load_iris()
```

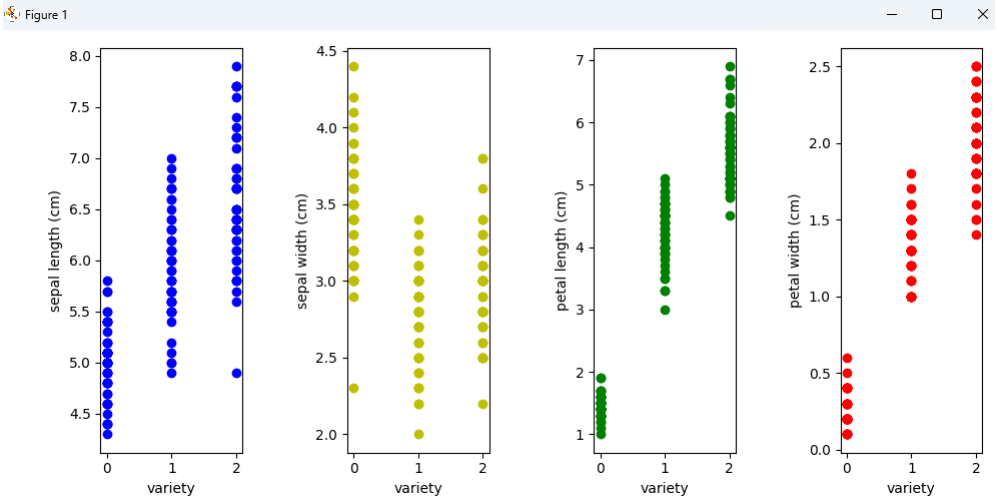
```
>>> print(exemple.data[7])
[5.  3.4 1.5 0.2]
>>> print(exemple.target[7])
0
```

La 8eme fleur du jeu de données est un iris de variété 0.
La variété 0 correspond à « setosa », 1 à « versicolor » et 2 à « virginia ». Les variétés sont enregistrées dans l'attribut appelé target.
La 8eme fleur possède 4 mesures : ses sépales mesurent 5 cm de long, 3.4 cm de large, ses pétales mesurent 1.5 cm de long et 0.2 cm. Les mesures sont enregistrées dans l'attribut data.

1a. Créer un fichier **decision_tree.py** dans votre répertoire de travail.
Coder une fonction **set_data** sans paramètres. Cette fonction :

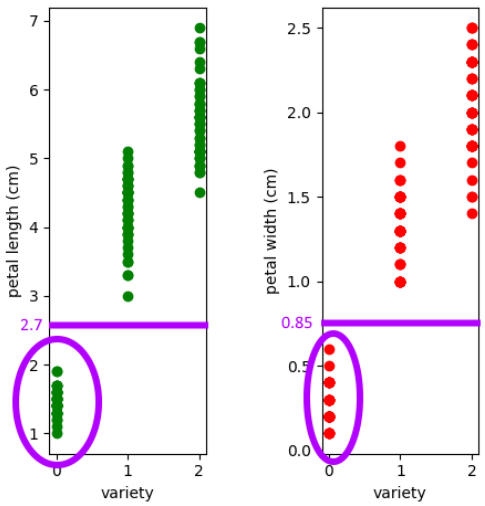
- utilise le jeux de données iris des datasets du module sklearn
- récupère l'objet iris généré par load_iris du module datasets
- complète 3 variables globales X avec data, y avec target et xlabels avec les noms des 4 valeurs mesurées pour l'attribut data
- complète 4 variables globales : X_pre,y_pre,X_train,y_train. Une donnée sur 5 de X et y sont placées dans X_pred et y_pred (jeu de données de prédiction) et 4 données sur 5 sont placées dans X_train, y_train (jeu de données d'entraînement)

1b. Coder une fonction **exercice1b** qui affiche chaque caractéristique en fonction de la variété, c'est à dire la représentation graphique ci-dessous.
Vous utiliserez les fonctions scatter et subplot de matplotlib.pyplot sur les données X et y.
En ordonnée, vous placerez les noms des 4 variables mesurées pour chaque fleur à l'aide de xlabels.

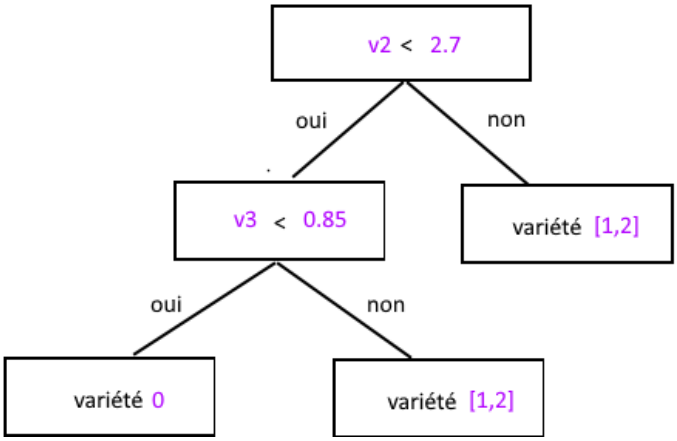


A partir des représentations graphiques, la variété 0 est facilement séparable des 2 autres en considérant les variables 2 (petal length) et 3 (petal width).

On peut définir la procédure ci-dessous pour déterminer si une fleur appartient à la variété 0.
Sur un vecteur de 4 mesures [v0,v1,v2,v3] caractérisant une fleur, on teste la valeur v2 (petal length). Si elle est inférieure à 2,7 alors c'est sans doute une variété 0.
Pour confirmer son appartenance à la variété 0, on effectue un 2eme test sur la valeur v3 (petal width), Si elle est inférieure à 0,85 alors c'est la variété 0.



On peut représenter cette procédure par un arbre binaire effectuant ces deux tests au niveau de la racine et d'un nœud. Ce sera un arbre de décision simplifié. Une feuille détermine avec certitude la variété 0. Les deux autres feuilles caractérisent les variétés 1 ou 2, sans pouvoir les distinguer.



1.c Pour coder cet arbre de décision, programmer une **classe Node** avec un **constructeur** et une méthode **predict**.
Le constructeur récupère les paramètres : variable, seuil, variété, gauche (valeur par défaut None) et droite (valeur par défaut None) et copie les valeurs dans ses 5 attributs : variable, seuil, variété, gauche, droite avec la signification suivante :

- variable indique le numéro d'une des 4 valeurs à tester (0 à 3), ce qui permet de tester v2 ou v3. Cet attribut est complété pour les nœuds mais il est vide (None) pour les feuilles.
- seuil indique la valeur seuil en dessous de laquelle v2 et v3 sont considérées comme caractéristique de la variété 0. Cet attribut est complété pour les nœuds mais il est vide (None) pour les feuilles.
- variété indique la variété de la feuille mais cet attribut est vide (None) pour les nœuds

La méthode predict :

- possède un paramètre data représentant un vecteur de mesure (exemple [5.1 3.5 1.4 0.2])
- si variable et seuil existent, la méthode compare la valeur d'indice variable de data avec seuil et appelle predict du nœud enfant gauche si strictement inférieur sinon elle appelle predict du nœud enfant droit. L'appel est récursif.
- dans le cas contraire (variable et seuil valent None), predict retourne variété

A la suite du codage de la classe Node, vous coderez la fonction **exercice1c** sans paramètres. Celle-ci :

- crée les nœuds et les feuilles de l'arbre en plaçant les numéros des variables v1 et v2 ainsi que les seuils s1 et s2 Par exemple : `feuille1 = Node(None,None,[0],None,None)` créé la feuille 1 de classe 0, `noeud = Node(3,0.85,None,feuille1,feuille2)` crée le nœud testant v3 avec le seuil associé de 0,85.
- Pour chaque valeur de X_pre, la fonction parcourt l'arbre à l'aide de la méthode predict de la racine. Le résultat de predict devrait correspondre à la variété attendue pour une variété 0 et aux deux autres variétés si la feuille atteinte est variété [1,2].
- affiche le résultat de 3 prédictions pour 3 variétés différentes X_pre[0], X_pre[10] et X_pre[20]
- calcule et affiche le taux de prédictions correctes. Une prédiction est correcte si une seule réponse est renvoyée et si elle correspond à la variété dans y_pre pour la donnée considérée.
- utiliser la donnée [7.5, 3.8, 2.2, 0.7] dans votre méthode predict pour déterminer sa catégorie : « variété 0» ou «variété [1,2] ». Affiche le résultat de la prédiction.

Affichage partiel :

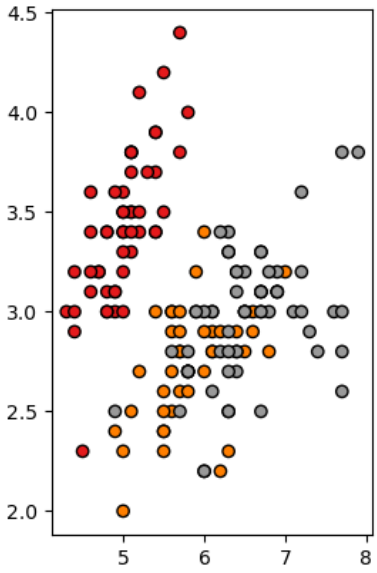
```
Donnee : [6.3 3.3 6.  2.5]  variété :  2
Prediction :  [1, 2]
30  données récupérées dans le jeu initial
Taux de prédiction :  0.3333333333333333
Donnee : [7.5, 3.8, 2.2, 0.7]  attendu : 0
Prediction :  [0]
```

1d. Coder une fonction **exercice1d** sans paramètres en charge d'afficher deux nuages de points :

- longueur sépale en fonction de largeur sépale
- longueur pétale en fonction de largeur pétale

Pour chaque nuage de points, colorez dans une couleur différente les points pour chaque variété. En vous appuyant sur une observation des graphique 1.b et 1.d, ajouter un commentaire dans votre fonction expliquant qu'il n'est pas possible de déterminer des critères de séparation pour les trois variétés.

Graphique partiel :



Les arbres de décision : modélisation et construction automatique

Dans le paragraphe précédent, la construction de l'arbre est manuelle : on choisit les seuils puis on construit l'arbre manuellement en nœud par nœud et feuille par feuille.
Cette méthode a ses limites. Dans ce paragraphe, nous allons construire les arbres à l'aide d'une méthode algorithme : on définit une métrique indiquant le niveau d'homogénéité d'un groupe de valeurs (aussi appelée sa pureté) pour regrouper de la meilleure manière possible les valeurs en sous-groupes de plus en plus homogènes en espérant avoir des sous-groupes finaux les plus homogènes possibles. Cette méthode est utilisée dans la construction d'un arbre de décision.

La construction d'un arbre de décision est effectuée en 2 phases :

- Construction : au départ, les données (aussi appelées les points d'apprentissage) sont tous placés dans le nœud racine. Chaque nœud est coupé (opération de séparation ou split) donnant naissance à plusieurs nœuds enfants possédant un coefficient d'homogénéité le plus important possible. Un point d'apprentissage est situé dans un seul nœud. L'arbre est construit par partitions successives en partant de la racine, en fonction de la valeur de la métrique testée à chaque itération. Le processus s'arrête quand les éléments d'un nœud ont la même valeur pour la métrique : les données sont homogènes dans une feuille.
- Elagage (pruning) : la méthode précédente présente un défaut, dénommé le surapparentissage : il suffit d'isoler chaque donnée dans une feuille et d'en créer autant que nécessaire pour obtenir une homogénéité parfaite. La phase d'élagage évite le surapparentissage en supprimant certaines branches dans les parties terminales peu représentatives pour garder de bonnes performances prédictives. Il faut un critère pour sélectionner les branches à élaguer. Après élagage, les nouvelles feuilles ne sont pas nécessairement homogènes mais elles possèdent un caractère majoritaire.

Ces deux mécanismes de construction d'un arbre de décision seront codés et illustrés dans les questions suivantes.

Exercice 2 : construction automatique d'un arbre de décision

Dans cet exercice, on utilise les variables globales X, y, X_train,y_train, X_pre, y_pre de l'exercice 1. On rappelle que : X, y regroupent toutes les données. Le jeu de données est séparé en deux parties : une partie des données, appelée données d'entraînement (X_train,y_train) sera utilisée pour la construction de l'arbre. La partie restante représente les données de prédiction (X_pre, y_pre) utilisées dans un arbre construit pour déterminer leur classe.
Dans cette partie, vous appellerez la fonction set_data au début de l'espace de nom __**main**__.

2a. Dans cette question, on calcule le coefficient de Gini, mesure standard de l'homogénéité d'une liste de données. On considère une liste y de valeurs y_i prises dans { $0,1,2$ } représentant des variétés d'iris. A y, on associe g(y), son coefficient de Gini. g(y) prend une valeur comprise entre 0 et 1 et se calcule par la formule :

$g(y)=\sum_{k=0}^{2} p_k \times (1-p_k)$ avec p_0 proportion de 0, p_1 proportion de 1 et p_2 proportion de 2 dans y de taille N g(y) est comprise entre 0 et 1.

Exemples :
Si y, de taille N, contient que des valeurs 0 alors $p_0 = 1$, $p_1 = 0$, $p_2 = 0$ donc le coefficient de Gini vaut $g(y)=1\times 0+0\times 1+0\times 1=0$. Les données sont homogènes
Si y de taille N contient 1/3 de 0, 1/3 de 1 et 1/3 de 2 alors le coefficient de Gini vaut
$g(y)=\frac{1}{3}\times\frac{2}{3}+\frac{1}{3}\times\frac{2}{3}+\frac{1}{3}\times\frac{2}{3}=3\times\frac{2}{9}=\frac{2}{3}$ les données ne sont pas homogènes
Lorsque g(y) est proche de 0, les valeurs de y sont homogènes (y est pur).
Si g(y) est proche de 1, les valeurs de y sont hétérogènes (y est impur).

Dans le fichier decision_tree.py, coder une fonction **gini** avec un paramètre y qui retourne le coefficient de Gini g(y) pour le vecteur y.
Coder également une fonction **exercice2a**, sans paramètres, qui affiche et retourne la valeur du coefficient de Gini pour y initial.
Affichage attendu :

```
gini y initial :  0.6666666666666667
```

2b. Dans cette question, coder la fonction de séparation, appelée **split**, de paramètres :

- X : liste de listes de 4 données portant mesurant chaque variable descriptive (longueur du sépale, largeur du sépale, longueur du pétale, largeur du pétale)
- y : liste des variétés associées à chaque donnée dans X, dans le même ordre
- s : seuil de séparation
- k : numéro de la variable considérée parmi les 4 possibilités (0, 1, 2, 3)

La fonction split sépare le jeu de données (X , y) en deux sous ensembles (X_1 , y_1) et (X_2 , y_2). Le jeu (X_1 , y_1) regroupe toutes les données de (X , y) dont la valeur pour la variable descriptive k est inférieure ou égale au seuil k. Dans le cas contraire, les données sont placées dans (X_2 , y_2). La fonction retourne X_1 ,y_1, X_2 ,y_2.

Coder une fonction **exercice2b**, sans paramètres, qui appelle split avec le jeu de données X, y, un seuil à 5.2 et la variable 0 et affiche la longueur des vecteurs X_1 et X_2 retournés.
Affichage attendu :

```
Split variable 0, seuil 5.2 en 2 vecteurs de longueur :  45 , 105
```

2c. La séparation de la question précédente nécessite un seuil. Dans cette question, le meilleur seuil est déterminé à l'aide du coefficient de Gini dans une fonction **split_opt** qui prend en paramètres :

- X : liste de listes de 4 données portant mesurant chaque variable descriptive (longueur du sépale, largeur du sépale, longueur du pétale, largeur du pétale)
- y : liste des variétés associées à chaque donnée dans X, dans le même ordre
- k : numéro de la variable considérée parmi les 4 possibilités (0, 1, 2, 3)

Dans une boucle, coder la fonction split_opt qui :

- prend un seuil donné s dans un intervalle I pour la variable k dans la liste des données X allant de sa valeur minimale à sa valeur maximale avec un pas de 0.1
- appelle la fonction split pour récupérer deux jeux de données (X_1 ,y_1) et (X_2 ,y_2)
- calcule le coefficient de Gini pour y_1 et y_2
- calcule la métrique p_1 g(y_1) + p_2 g(y_2) à minimiser. Cette métrique est une combinaison linéaire des coefficients de Gini : elle utilise p_1, la proportion de valeurs de X dans X_1 et p_2 , la proportion de valeurs de X dans X_2
- Les calculs précédents sont répétés dans une boucle pour tous les seuils s de l'intervalle I. A chaque tour, la fonction enregistre le meilleur seuil pour la métrique la plus faible

La fonction split_opt retourne le meilleur seuil et le gain sur le coefficient initial de Gini (coefficient_initial – coefficient_pour_le_meilleur_seuil).

Coder également une fonction **exercice2c**, sans paramètres, qui appelle la fonction split_opt(X,y,3) pour la variable 3, qui affiche les valeurs ci-dessous et renvoie dans l'ordre : le seuil optimal, le gain optimal, les deux coefficients de Gini pour chaque sous-groupe de données après la séparation

Affichage attendu :

```
Variable 3 : seuil optimal  0.6000000000000002  gain optimal  0.3333333333333334
Coefficient de Gini du groupe 1= 0.0
Coefficient de Gini du groupe 2= 0.5
```

2d. Il reste à déterminer les seuils et les gains pour les 4 variables descriptives, ce qui permettra de choisir, lors de la construction de l'arbre, la variable présentant le meilleur gain pour chaque nœud.
Coder une fonction **K_opt** de paramètres: X et y, qui :

- parcourt chaque numéro de variable (0, 1, 2, 3), détermine le seuil et le meilleur gain sur le coefficient de Gini pour chaque valeur de k
- retourne le numéro de variable présentant le meilleur gain, le seuil et le gain optimums

Coder également une fonction **exercice2d**, sans paramètres, qui :

- appelle K_opt pour X et y
- affiche le numéro du critère de séparation optimal, le seuil et le gain
- sépare les données X,y selon le numéro et le seuil obtenu avec K_opt
- calcule puis affiche le coefficient de Gini pour chaque sous groupe (X_1, y_1) et (X_2, y_2)

Affichage attendu :

```
Critère de séparation optimal pour k= 2  s= 1.9000000000000008
Gain en pureté = 0.3333333333333334
Coefficient de Gini du groupe 1= 0.0
Coefficient de Gini du groupe 2= 0.5
```

2e Créer une **classe Node2** à partir de laquelle on va construire l'arbre. Un objet de type Node2 sera soit un nœud de test dans l'arbre soit une feuille déterminant une classe. Un nœud peut avoir 2 enfants générés par une séparation optimale (k_opt,s_opt). Chaque nœud (ou feuille) contient une liste de 4 mesures caractérisant les fleurs qu'il représente. Un nœud contient également k et s (numéro de variable et seuil considéré) et deux enfants left, right.

Coder le **constructeur** qui aura comme paramètres : X, y. Il copie ces données représentant les iris dans les attributs de la classe : X, y. Les autres attributs sont initialisés à None : k (numéro de variable testée), s (seuil considéré) et deux enfants left, right. Si l'objet est une feuille, k et s seront à None ainsi que left et right.

Coder dans Node2, une méthode **leaves** retournant la liste des feuilles pour un arbre.

Coder dans Node2 une méthode **grow,** de paramètre pruning (valeur par défaut 0), qui :

- parcourt l'ensemble des feuilles de l'arbre, calcule l'indice, le seuil et le gain optimal avec K_opt pour chaque feuille
- détermine la feuille présentant le meilleur gain (et le meilleur indice, pour le meilleur seuil)
- sépare la feuille avec le meilleur gain (qui devient un nœud) en deux sous feuilles (enregistrées dans left, right) et sauvegarde le seuil et l'indice optimal dans les attributs k,s du nouveau nœud. La séparation est conditionnée à la valeur de pruning. Si pruning vaut 0, la séparation a lieu dans tous les cas. Si pruning est différent de 0, la séparation a lieu si les 2 nœuds enfants possèdent un effectif strictement supérieur à pruning, sinon la séparation n'a pas lieu.

La méthode grow renverra la valeur True si un gain en pureté a pu être réalisé. Sinon, elle renverra la valeur False. La méthode grow permet de construire un arbre existant en créant de nouvelles branches en séparant au mieux les données contenues dans la feuille présentant le gain le plus élevé tout en respectant la contrainte d'élagage.

Coder dans Node2 une fonction **print** pour afficher l'ensemble des caractéristiques de tous les nœuds d'un arbre. Chaque nœud affiche la taille des données X (ou y), le coefficient de Gini des données, l'indice k de la variable à tester, le seuil s, les nœuds enfants (L pour left et R pour right) de manière récursive avec indentation.

Coder enfin une fonction **exercice2e**, sans paramètres, qui crée un nœud racine à partir de X,y, appelle grow (pruning=0) sur la racine puis affiche les informations de la racine (k,s,coefficient de Gini de l'enfant gauche et droit) puis affiche l'arbre avec print. Affichage attendu ci-contre.

```
- Affichage des informations de la racine :
Séparation pour k: 2  s: 1.9000000000000008
Coefficient de Gini gauche : 0.0
Coefficient de Gini droit : 0.5
- Affichage arbre avec print :
taille= 150
gini= 0.6666666666666667
k= 2  s= 1.9000000000000008
L :
        taille= 50
        gini= 0.0
R :
        taille= 100
        gini= 0.5
```

2f Ajouter dans Node2 une méthode **extend** avec le paramètre pruning à la valeur par défaut 0. Cette méthode appelle grow tant q'un gain en pureté est possible. Extend transmet également le paramètre de pruning à grow lors de l'appel.

Coder également une fonction **exercice2f** qui crée un nœud racine à partir de X,y, appelle grow (pruning=0) puis extend(pruning=0) et affiche l'arbre avec print
L'arbre obtenu compte 9 feuilles contenant parfois peu de valeurs 1 ou 2 dont le coefficient de Gini est nul pour chaque feuille.

2g Ajouter dans la classe Node2, une méthode **predict** de paramètre x , une liste de 4 valeurs pour chaque variable descriptive. La méthode renvoie la variété obtenue d'une feuille après avoir parcouru différents nœuds. A chaque nœud, les critères de comparaison sont appliquées récursivement jusqu'à atteindre une feuille : si le seuil existe, la valeur d'indice k est testée. Si elle est intérieure au seuil s du noeud, les données sont transmises au nœud enfant left qui poursuit récursivement les tests de la même manière sinon les données sont transmises au nœud enfant droit (qui poursuit récursivement les tests de la même manière).
Si la feuille atteinte (k=None et s=None), après le parcours de l'arbre, n'est pas pure, on tire de manière aléatoire sa variété dans les données hétérogènes de la feuille.

Coder également une fonction **exercice2g** qui crée un nœud racine à partir de X,y, appelle grow (pruning=0) puis extend(pruning=0) puis appelle prédiction pour les mêmes données que celle de la question 1c : X_pre[0], X_pre[10], X_pre[20] et la donnée non initiale [7.5, 3.8, 2.2, 0.7] .
Les résultats sont-ils cohérents avec ceux de l'exercice 1c ? En utilisant X_train et y_train, quel est le taux de succès des prédictions en pourcentage ? Vous répondrez dans un commentaire.

2h. Coder une fonction **exercice2h** qui crée un nœud racine à partir de X_train,y_train, appelle grow (pruning=0) puis extend(pruning=0) puis appelle prédiction pour toutes les données de X_pre et y_pre.
Cette approche est courante : on crée un arbre de décision à l'aide d'une partie des données appelées données d'entraînement et on vérifie l'efficacité de la prédiction du modèle sur une autre partie des données.
En utilisant X_train et y_train, quel est le taux de succès des prédictions en pourcentage ?
Quelle valeur de pruning fait chuter ce taux en dessous de celui de la question 1c ?
Répondre à ces questions par un commentaire dans la fonction exercice2h.

Exercice 3 : un autre arbre de décision

En interrogeant ChatGPT sur l'arbre de décision des données iris, celui-ci fournit le code :

```
from sklearn import tree,datasets
import matplotlib.pyplot as plt

# Jeux de données
iris = datasets.load_iris()
X=iris.data
y=iris.target

# Phase d'apprentissage
clf=tree.DecisionTreeClassifier()
clf.fit(X,y)

# Affichage de l'arbre
tree.plot_tree(clf, filled=True)
plt.show()

# prédiction de la catégorie y pour x donnés
print(clf.predict([[6,3,4,1]]))
```

3a. Que fait ce code ? Quel est le rôle des méthodes DecisionTreeClassifier et fit ?

3b. plot_tree permet d'obtenir l'arbre de décision pour le jeu de données iris.
Comparer l'arbre afficher avec votre affichage de la question 2f. Quelles sont les différences ? Comment l'expliquer ?

3c. Séparer les données en 2 groupes 80%/20% pour le jeu d'entraînement (X_train, y_train) et le jeu de prédiction (X_pre,y_pre). Calculer le taux d'efficacité des prédictions pour un modèle entraîné sur le jeu complet de données et un autre entraîné sur (X_train, y_train). Quel est le taux de succès des prédictions ? Y-a-t'il une différence avec les observations des questions 2g et 2h.