AVIGNON UNIVERSITÉ

TP 1 : Introduction aux bibliothèque Python

6 octobre 2023

Master 2
Intelligence Artificielle
UE Business Intelligence
ECUE Data Analytics & Big Data

Réalisé par :

AIT-ABBOU Samir

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'ENSEIGNEMENT
ET DE RECHERCHE
EN INFORMATIQUE
ceri.univ-avignon.fr

1 Introduction

Le but de ce premier TP est de vous familiariser avec les bibliothèques Python de fouille données (Data mining) :

scikit learn (outils et algorithmes d'apprentissage automatique), **numpy** (pour les manipulations de nombres), **scipy** (pour les calculs scientifiques), **pandas** (pour manipuler les ensembles de données) et matplotlib (pour la visualisation).

2 Analyse descriptive des données

L'ensemble de données « Fruits » que nous allons étudier a été créé par Iain Murray de l'université d'Édimbourg. Il a été construit à partir de l'observation de quelques dizaines d'oranges, de mandarines, de citrons et de pommes de différentes variétés. Les mesures de chacun de ces fruits ont été enregistrées dans une table ; c'est un ensemble de taille réduite, établi à des fins pédagogiques.

a. Utilisez pandas pour lire le fichier et le convertir en data frame

```
# read input text and put data inside a data frame
fruits = pd.read_table('data/fruit_data_with_colors.txt')
print(fruits.head())
```

```
   fruit_label fruit_name fruit_subtype  mass  width  height  color_score
0            1      apple  granny_smith   192    8.4     7.3         0.55
1            1      apple  granny_smith   180    8.0     6.8         0.59
2            1      apple  granny_smith   176    7.4     7.2         0.60
3            2   mandarin      mandarin    86    6.2     4.7         0.80
4            2   mandarin      mandarin    84    6.0     4.6         0.79
```

b. Combien y a-t-il d'attributs ?

```
# print nb of instances and features
print(fruits.shape)
```

```
(59, 7)
```

j'ai donc 7 attributs

c. Que représentent-ils ?

Ils représentent les entrés(input) et sorties(output=Target) de notre futur model

d. Quel est leur type ?

```
# print feature types
print(fruits.dtypes)
```

```
fruit_label         int64
fruit_name         object
fruit_subtype      object
mass                int64
width             float64
height            float64
color_score       float64
dtype: object
```

e. Quelle est la classe à prédire ?

la classe à prédire c'est bien **fruit_label**

f. Combien y a-t-il d'instances dans le fichier ?

```
# print nb of instances and features
print(fruits.shape)
```

```
(59, 7)
```

j'ai 59 instatnces (exemples)

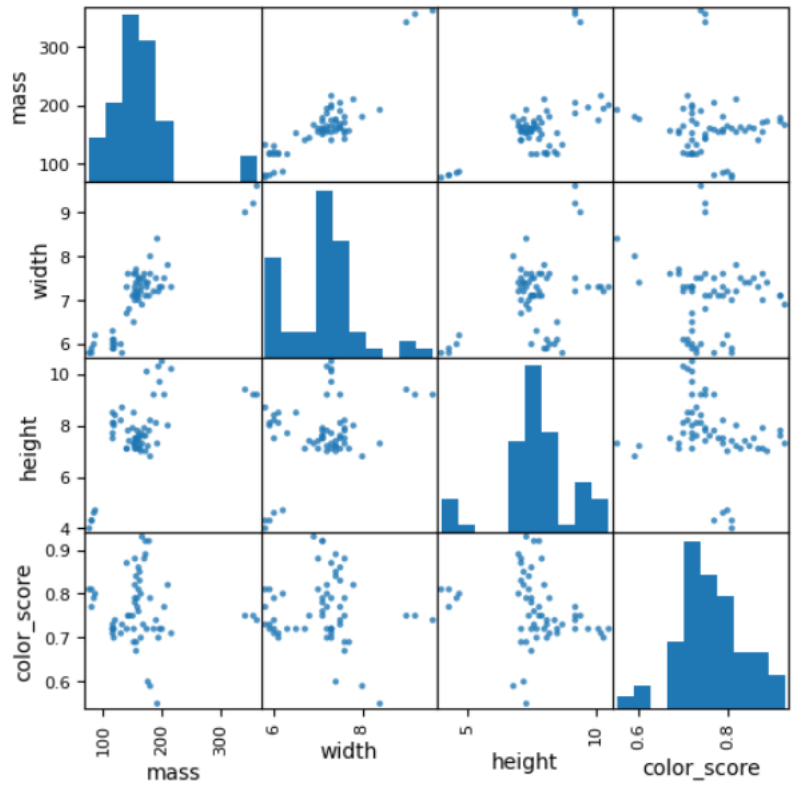
g. Les classes sont-elles équilibrées quant au nombre d'instances ?

```
# print balance between classes
print(fruits.groupby('fruit_name').size())
```

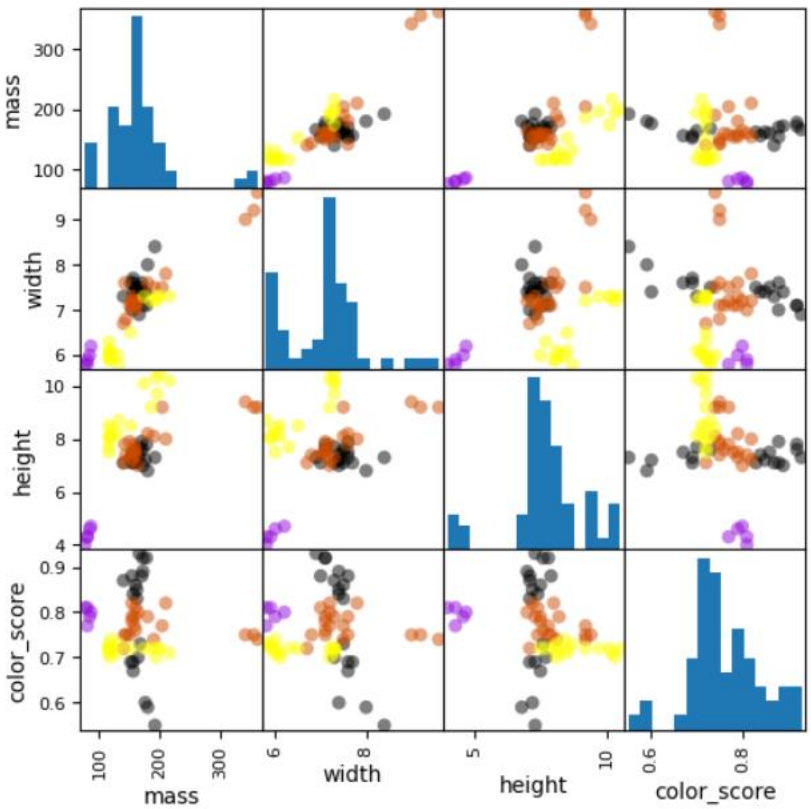
```
fruit_name
apple       19
lemon       16
mandarin     5
orange      19
dtype: int64
```

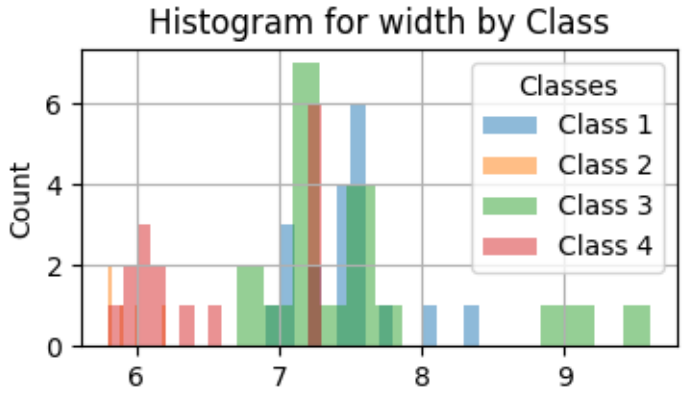
Donc les classes ne sont pas équilibrées au nombre d'instances; le mandarin et limon ont mois d'instances par rapport aux autre

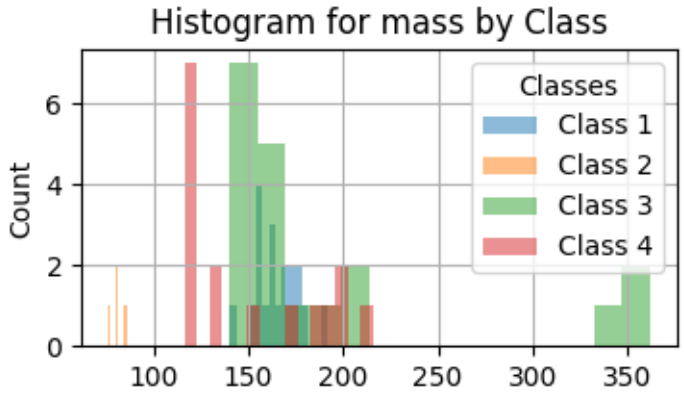
h. . Examinez les données de manière graphique à l'aide d'un diagramme de dispersions (fonction plotting.scatter_matrix() de pandas)

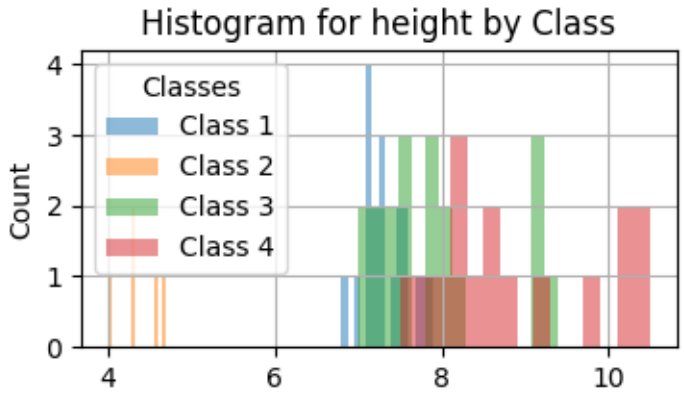


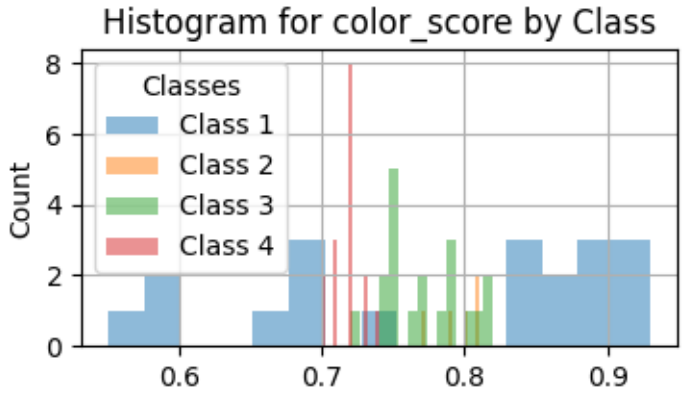
i. Examinez les données de manière graphique à l'aide d'histogrammes montrant la répartition selon un attribut et la classe.




Histogram for width by Class
Classes
Class 1
Class 2
Class 3
Class 4
Count
0
2
4
6
6
7
8
9


Histogram for mass by Class
Classes
Class 1
Class 2
Class 3
Class 4
Count
0
2
4
6
100
150
200
250
300
350

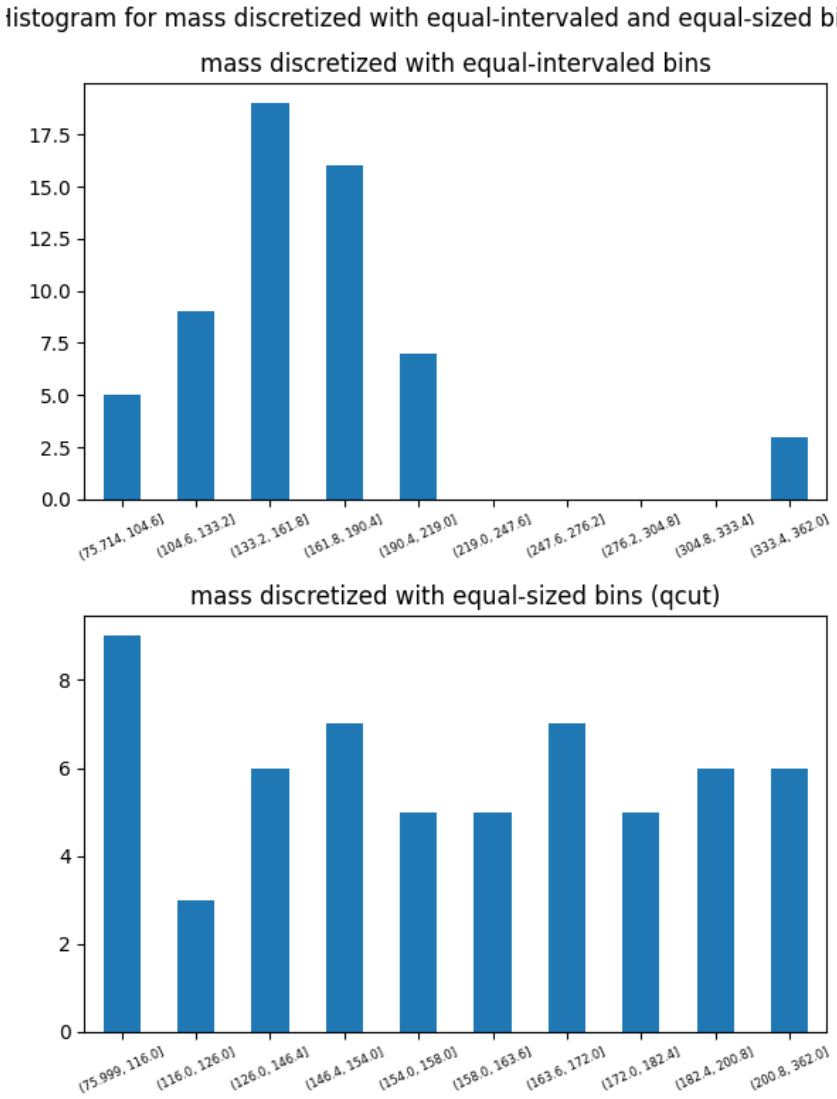

Histogram for height by Class
Classes
Class 1
Class 2
Class 3
Class 4
Count
0
1
2
3
4
4
6
8
10


Histogram for color_score by Class
Classes
Class 1
Class 2
Class 3
Class 4
Count
0
2
4
6
8
0.6
0.7
0.8
0.9

3 Prétraitement

Appliquez à partir des données originelles deux variantes de discrétisation : une première reposant sur des intervalles de même taille (equal-intervaled avec la fonction cut() de pandas) ou de mêmes effectifs (equal-sized bins avec qcut()). En affichant les historiques correspondant à chacune des deux configurations à l'aide de plot.bar(), comparez les histogrammes obtenus pour l'attribut « mass »

a- Discrétisation : equal-intervaled et equal-sized



b- L’effet du filter de normalisation processing.MinMaxScaler de Scikit-learn :

Le filtre de normalisation MinMaxScaler de scikit-learn est utilisé pour mettre à l'échelle les valeurs d'un ensemble de données dans un intervalle spécifié (par défaut, [0, 1]) en préservant la distribution relative des valeurs. Cet objet est couramment utilisé pour normaliser les caractéristiques des données, ce qui peut être utile pour de nombreux algorithmes d'apprentissage automatique, en particulier ceux qui sont sensibles à l'échelle des caractéristiques, tels que les algorithmes basés sur la distance (par exemple, les algorithmes de clustering ou de régression linéaire).

X_scaled = (X - X_min) / (X_max - X_min)

c- Cas d’Utilisation de transform() au lieu de fit_transform()

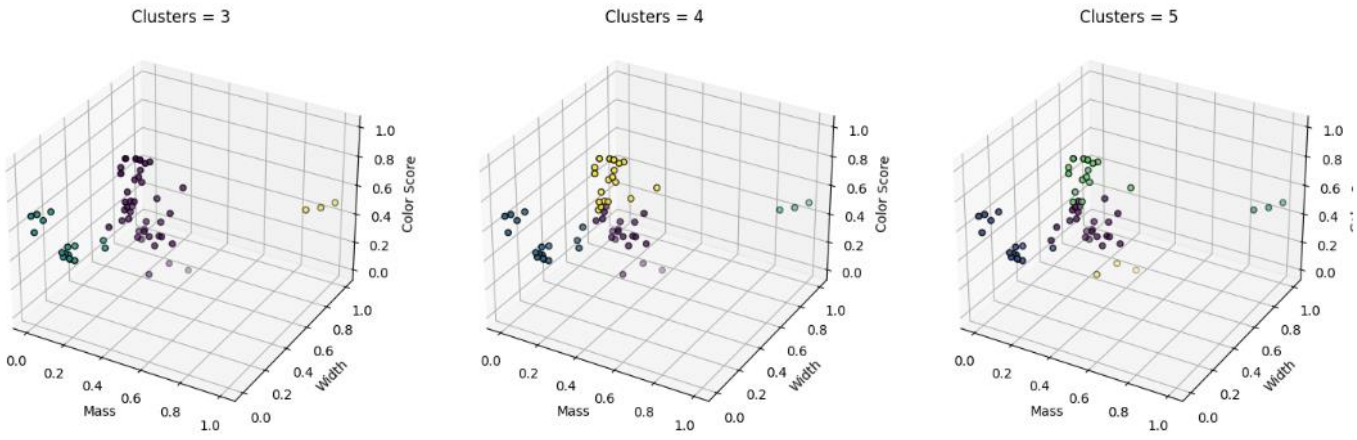
On peut utiliser fit_transform() pour la première transformation, généralement sur l'ensemble de données d'entraînement, pour apprendre les paramètres. Ensuite, on peut utiliser transform() pour appliquer la même transformation à d'autres ensembles de données en utilisant les paramètres appris.

Il est préférable d'utiliser transform() lorsque on a déjà appris les paramètres de mise à l'échelle et qu'on souhaite simplement les appliquer à de nouvelles données.

4 Clustering

a-

Ouvrez le fichier de données en appliquant un filtre de normalisation. Appliquez KMeans sur ces données en fixant successivement le nombre de clusters à 3, 4 et 5. Examinez manuellement les résultats du clustering en traçant les instances dans une espace en 3 dimensions correspondant aux attributs « mass », « width » et « color_score ». Quelle(s) classe(s) représente chacun des clusters ?



a.a Quelle(s) classe(s) représente chacun des clusters ?

- pour le nbre de clusters = 3 :

0n peut faire des requettes pour filtrer les données suivants les valeurs des attributs mass, width et color_score observées sur les graphes comme suit :

```
classe_2 = fruits[(fruits['color_score'] >= 0.70) & (fruits['width'] >= 9.5)]
classe_2
```

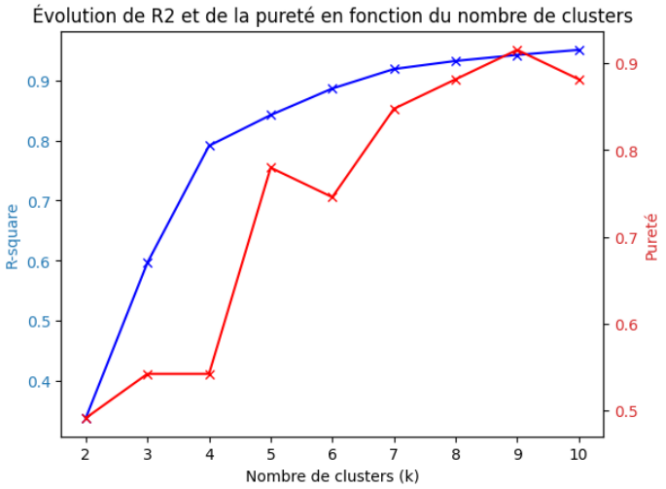
	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
26	2	orange	spanish_jumbo	362	9.6	9.2	0.74

On remarque bien que le groupe ayant le couleur violet correspond bien à la classe 2 calculer par Kmeans

On fait la même chose pour les autres, et c'est pareil pour les autres nombres de clusters (4 et 5)

b-

Pour fixer le nombre de clusters, on propose d'utiliser la métrique R2 calculée suivant les variances intra-classe et inter-classe. Cette métrique permet d'avoir une évaluation objective de l'homogénéité des groupes construits artificiellement, sans avoir étiqueté manuellement chaque instance. Un module R_square_clustering.py effectuant le calcul de R2 vous est fourni sur le site du cours. Dessinez le graphe montrant l'évolution de cette métrique en faisant varier le nombre de clusters entre 2 et 10. Comment varie R2 ? Quel est le nombre de clusters que vous retiendriez ?



b.b Comment Varier R_Square?:

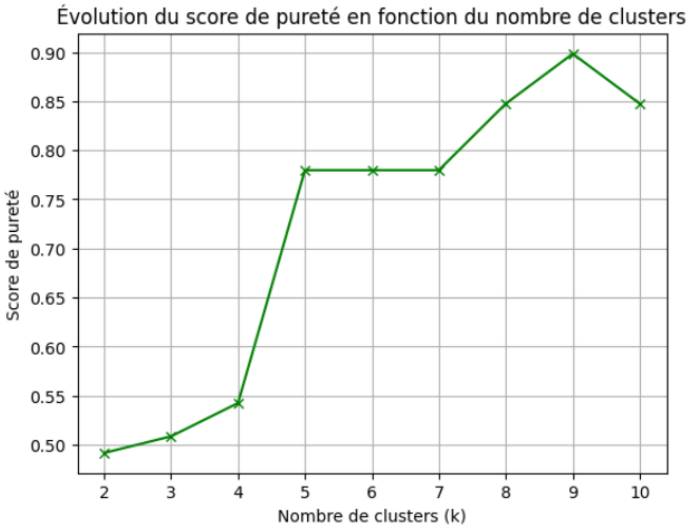
En examinant le graphe de R2 en fonction de k, on observe que à partir de K=4, R2 commence à se stabiliser. C'est généralement le nombre de clusters que nous retiendrions comme optimal pour notre modèle de clustering **kmeans**.

b .c Quel est le nombre de clusters que nous retiendrions ?

En examinant le graphe de l'évolution de K en fct de R_square on voit que le graphe stabilise exactement en K=4, ce qui est exactement le nbre de classes à retenir.

c-

Il est possible d'obtenir une évaluation quantitative plus complète du clustering lorsque l'on dispose des classes à obtenir dans les données (c'est un cas peu réaliste de l'apprentissage non supervisé mais ici nous sommes dans la situation luxueuse où chaque instance est étiquetée). Un module purity.py disponible sur le site du cours détermine un score de pureté en attribuant chaque cluster à la classe majoritaire. Utilisez-le pour tracer un graphe montrant l'évolution de ce score en fonction du nombre de clusters variant entre 2 et 10.

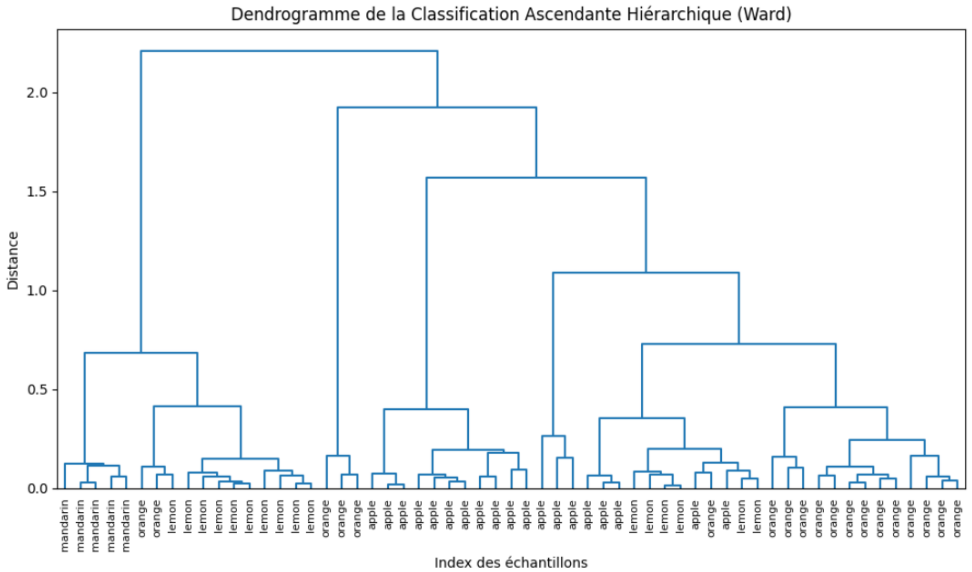


Quel est le nombre de groupes que vous retiendriez suivant ce critère ?

En observant le graphe, le nbre de groupes que je peux retenir c'est 4(là où la pureté commence à augmenter d'une façon remarquable)

d- Remplacement k-means par une classification ascendante hiérarchique :

Remplacez l'utilisation du calcul des k-moyennes par une classification ascendante hiérarchique en établissant le dendogramme des données normalisées à l'aide de la méthode de Ward et de la bibliothèque scipy.cluster.hierarchy. Commentez l'arbre obtenu et proposez une partition en sélectionnant une hauteur appropriée de coupure.



d-a) Commentez l’arbre obtenu et proposez une partition en sélectionnant une hauteur appropriée de coupure:

L'arbre obtenu à partir du dendrogramme représente la structure hiérarchique des données après avoir effectué une classification ascendante hiérarchique (CAH) en utilisant la méthode de Ward. Chaque feuille de l'arbre représente une instance ou un groupe d'instances (selon le niveau de découpage), et les branches représentent les regroupements successifs.

Au début (en bat du dendrogramme) tous les enregistrements sont considérés comme des clusters après le modèle commence à regrouper des éléments les uns avec les autres en fonction de distances jusqu’à avoir un seul cluster

Pour notre choix de cluster en peut faire une coupure horizontale a partir de la distance 1,2 la ou on a des clusters très claires distants les l’un des autre donc on aura 4 clusters.

5 Classement

Un des intérêts de scikit-learn est d'incorporer un grand nombre d'algorithmes d'apprentissage automatique permettant de prédire la classe des individus. On se propose ici de tester quelques méthodes simples de classement selon deux protocoles d'évaluation.

a- Découpage des données

```
from sklearn.model_selection import train_test_split
# Divisez les données en ensembles d'entraînement (75%) et de test (25%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

b- Comparaison de performances de méthodes de classement

```
Accuracy Dummy Classifier: 0.26666666666666666
Accuracy Gaussian Naive Bayes: 0.8666666666666667
Accuracy Decision Tree Classifier: 0.8
Accuracy Random Forest Classifier: 0.8666666666666667
Accuracy Logistic Regression: 0.6666666666666666
```

c- Quelle est la classe pour laquelle la prédiction fait le plus d'erreurs ?

D'après les matrices de confusion que vous avez fournies, voici le nombre d'erreurs de classification pour chaque classe pour chaque méthode de classification :

Pour le "Dummy Classifier" (classifieur élémentaire) :

- Classe 0 : 0 erreurs
- Classe 1 : 2 erreurs
- Classe 2 : 4 erreurs
- Classe 3 : 5 erreurs

Pour le "Gaussian Naive Bayes" :

- Classe 0 : 0 erreurs
- Classe 1 : 1 erreur
- Classe 2 : 4 erreurs
- Classe 3 : 0 erreurs

Pour le "Decision Tree Classifier" :

- Classe 0 : 1 erreur
- Classe 1 : 0 erreurs
- Classe 2 : 0 erreurs
- Classe 3 : 2 erreurs

Pour le "Random Forest Classifier" :

- Classe 0 : 1 erreur
- Classe 1 : 0 erreurs
- Classe 2 : 0 erreurs
- Classe 3 : 1 erreur

Pour la "Logistic Regression" (régression logistique) :

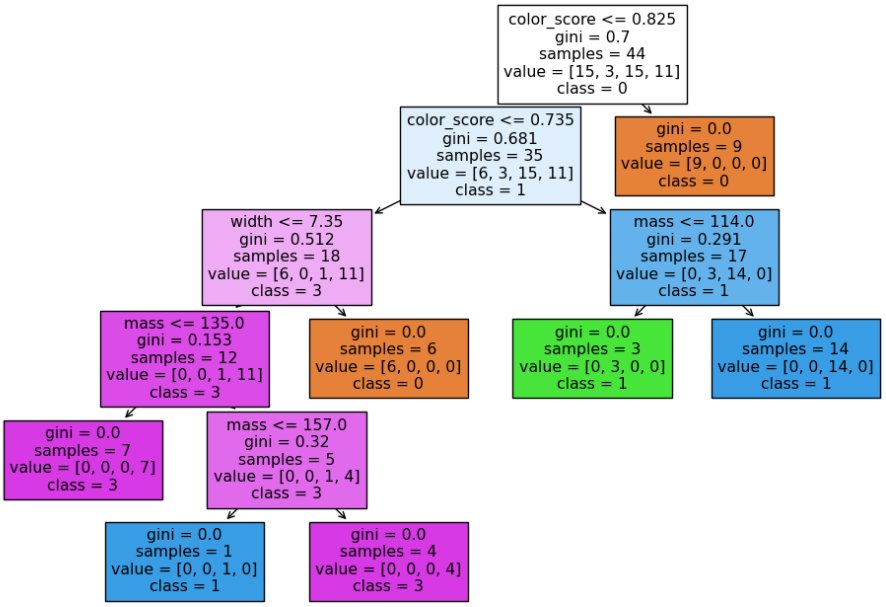
- Classe 0 : 4 erreurs
- Classe 1 : 0 erreurs
- Classe 2 : 1 erreur
- Classe 3 : 2 erreurs

En analysant ces résultats, on peut voir que chaque méthode de classification fait le plus d'erreurs pour différentes classes :

- Le "Dummy Classifier" fait le plus d'erreurs pour la Classe 3.
- Le "Gaussian Naive Bayes" fait le plus d'erreurs pour la Classe 2.
- Le "Decision Tree Classifier" fait le plus d'erreurs pour la Classe 3.
- Le "Random Forest Classifier" fait le plus d'erreurs pour la Classe 0.
- La "Logistic Regression" fait le plus d'erreurs pour la Classe 0.

Cela signifie que chaque méthode a des performances variables en fonction de la classe à prédire, et la classe pour laquelle il y a le plus d'erreurs dépend de la méthode de classification utilisée.

d- Dans le cas de l'arbre de décision, visualisez avec la fonction tree.plot_tree et commentez l'arbre construit par le modèle.



e- Commentez l'arbre construit par le modèle.

f- Validation croisée en 5 strates

```
Dummy Classifier: Mean Accuracy = 0.34, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.70, Std Deviation = 0.12
Decision Tree Classifier: Mean Accuracy = 0.89, Std Deviation = 0.07
Random Forest Classifier: Mean Accuracy = 0.91, Std Deviation = 0.05
Logistic Regression: Mean Accuracy = 0.64, Std Deviation = 0.22
```

g- parmi les 5 méthodes précédement testés, quelle est la plus performante ?

Sur la base de ces moyennes, on peut dire que le modèle "Random Forest Classifier" est le plus performant, car il a la précision moyenne la plus élevée (0.91). Il est suivi du modèle "Decision Tree Classifier" avec une précision moyenne de 0.89. Le modèle "Gaussian Naive Bayes" est également assez performant avec une précision moyenne de 0.70.

Cependant, il est important de noter que la variabilité des performances (indiquée par l'écart-type, Std Deviation) varie également. Un écart-type plus faible signifie une plus grande stabilité des performances du modèle. Dans ce cas, le "Random Forest Classifier" a un écart-type de 0.05, ce qui suggère une stabilité relativement élevée de ses performances. En revanche, le modèle "Logistic Regression" a un écart-type élevé de 0.22, ce qui indique une variabilité significative de ses performances entre les itérations de la validation croisée.

Donc, bien que le "Random Forest Classifier" soit le plus performant en termes de précision moyenne, il est également important de prendre en compte la stabilité des performances lors de la sélection du modèle final. Dans ce cas, le "Random Forest Classifier" semble être un choix solide.

h- Le classement des méthodes est-il le même que lors du protocole d'évaluation précédent ?

Avant la validation croisée :

- Le "Gaussian Naive Bayes" et le "Random Forest Classifier" avaient des performances similaires, avec une accuracy d'environ 0.87.
- Le "Decision Tree Classifier" avait une accuracy de 0.80, ce qui était également assez élevé.
- Le "Logistic Regression" avait une accuracy de 0.67, ce qui était légèrement inférieur, mais toujours respectable.
- Le Dummy Classifier" avait la performance la plus faible avec une accuracy d'environ 0.27, ce qui est attendu car il effectue des prédictions aléatoires.

Après la validation croisée :

- Le "Random Forest Classifier" est le modèle le plus performant, avec une accuracy moyenne de 0.91 et une faible variance (écart-type de 0.05). Cela signifie qu'il a généralement de très bonnes performances et est stable.
- Le "Decision Tree Classifier" suit de près, avec une accuracy moyenne de 0.89 et une variance modérément faible (écart-type de 0.07).
- Le "Gaussian Naive Bayes" a une accuracy moyenne de 0.70, ce qui est inférieur aux modèles d'arbres, mais il présente une plus grande variabilité (écart-type de 0.12).
- Le "Logistic Regression" a la performance la moins élevée avec une accuracy moyenne de 0.64 et la plus grande variabilité (écart-type de 0.22).
- Le "Dummy Classifier" a montré une légère amélioration par rapport à la précédente évaluation, mais reste le modèle le moins performant.

En résumé, la validation croisée a permis d'obtenir une évaluation plus robuste des performances des modèles, montrant que le "Random Forest Classifier" est le modèle le plus performant, suivi du "Decision Tree Classifier." Ces résultats sont plus fiables que les évaluations basées sur une seule division des données en entraînement et test.

Sélection des paramètres pour le modèle de régression logistique

```
Best score: 0.681
Best parameters set:
        C: 0.01
        penalty: None
Accuracy on test data: 0.733
```

Best score: 0.681 : C'est le score de performance optimal obtenu lors de la recherche par grille (GridSearchCV). Ce score est généralement basé sur une métrique telle que la précision (accuracy) ou une autre métrique appropriée à notre problème. Dans ce cas, il s'agit de 0.681.

Best parameters set : Ces lignes indiquent les meilleures valeurs des hyperparamètres qui ont conduit au meilleur score. Dans votre cas, les meilleures valeurs sont :

- C (force de régularisation**)** : 0.01
- Penalty (type de régularisation) : None (ce qui signifie aucune régularisation)

Cela signifie que, selon la recherche par grille, le modèle de régression logistique performe le mieux lorsque la force de régularisation (C) est fixée à 0.01 et qu'aucune régularisation (penalty) n'est appliquée aux coefficients.

Accuracy on test data: 0.733 : C'est le score de précision du modèle optimisé évalué sur des données de test indépendantes. Il s'agit de la précision du modèle (le nombre de prédictions correctes divisé par le nombre total de prédictions) sur ces données de test. Dans notre cas, le modèle a une précision de 0.733 sur les données de test.

6 Classement et discrétisation

a- Discrétisez les attributs numériques à l'aide d'intervalles de mêmes tailles

La discrétisation des attributs numériques semble avoir un impact sur les performances des modèles de classification. Voici une comparaison des performances avant et après la discrétisation :

Avant la discrétisation :

- Dummy Classifier avait une précision moyenne de 0.34.
- Gaussian Naive Bayes avait une précision moyenne de 0.70.
- Decision Tree Classifier avait une précision moyenne de 0.89.
- Random Forest Classifier avait une précision moyenne de 0.91.

Après la discrétisation :

- Dummy Classifier a maintenu une précision moyenne de 0.34, ce qui n'est pas surprenant car il se base sur la classe majoritaire.
- Gaussian Naive Bayes a vu sa précision moyenne baisser à 0.50. La discrétisation a probablement réduit la capacité du modèle à bien gérer les distributions continues.
- Decision Tree Classifier a également vu sa précision moyenne baisser à 0.84, mais reste un bon modèle.
- Random Forest Classifier a maintenu une précision moyenne de 0.84, ce qui est cohérent avec l'arbre de décision seul.
- Logistic Regression a vu sa précision moyenne passer à 0.69, ce qui indique que la discrétisation peut ne pas être bénéfique pour ce modèle.

b- Discrétisez les attributs numériques à l'aide d'intervalles de mêmes effectifs

Avant la discrétisation :

-Dummy Classifier : Mean Accuracy = 0.34

-Gaussian Naive Bayes : Mean Accuracy = 0.70

-Decision Tree Classifier : Mean Accuracy = 0.89

-Random Forest Classifier : Mean Accuracy = 0.91

-Logistic Regression : Mean Accuracy = 0.64

-

Après la discrétisation (intervalles de mêmes effectifs) :

-Dummy Classifier : Mean Accuracy = 0.34

-Gaussian Naive Bayes : Mean Accuracy = 0.78

-Decision Tree Classifier : Mean Accuracy = 0.86

-Random Forest Classifier : Mean Accuracy = 0.89

-Logistic Regression : Mean Accuracy = 0.66

On peut observer que la discrétisation a eu un effet différent sur chaque modèle. Le modèle Gaussian Naive Bayes a montré une amélioration significative de ses performances après la discrétisation, tandis que le modèle Logistic Regression a montré une légère amélioration. Les modèles Decision Tree Classifier et Random Forest Classifier maintiennent des performances élevées, bien que légèrement inférieures à celles avant la discrétisation. Le Dummy Classifier, quant à lui, conserve des performances similaires.

En général, la discrétisation peut être utile pour certains types de données, mais elle peut aussi avoir un impact sur les performances des modèles. Il est important de choisir judicieusement la méthode de discrétisation en fonction de la nature des données et des objectifs du modèle.

7 ACP et sélection des variables

Quelles sont les valeurs propres des facteurs de l'ACP ?

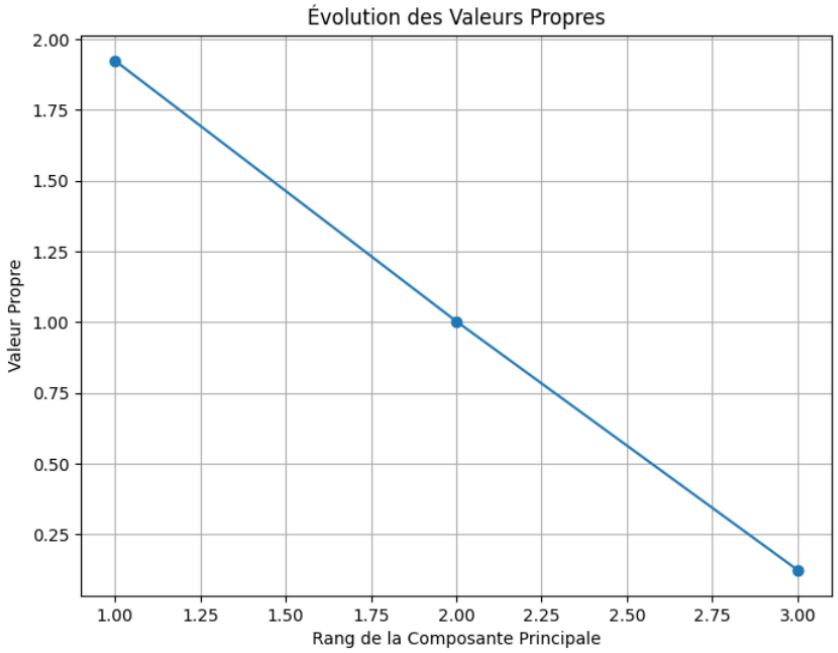
```
# Valeurs propres des facteurs de l'ACP
eigenvalues = acp.explained_variance_
print("valeurs propres: ",eigenvalues)
# Vous pouvez également obtenir le pourcentage d'explication de chaque composante
explained_variance_ratio = acp.explained_variance_ratio_
print("\nle pourcentage d'explication de chaque composante",explained_variance_ratio)
```

```
valeurs propres:  [1.92401292 1.00329581 0.12441541]

le pourcentage d'explication de chaque composante [0.63046751 0.3287636  0.04076889]
```

Valeurs Propres : Les valeurs propres mesurent la quantité de variance expliquée par chaque composante principale. Dans votre cas, les trois valeurs propres sont approximativement 1.924, 1.003, et 0.124. La première composante principale explique la plus grande quantité de variance (1.924), suivie de la deuxième (1.003), et enfin la troisième (0.124). Cela signifie que la première composante principale capture la variance la plus significative dans vos données, suivie de près par la deuxième composante principale, tandis que la troisième n'explique qu'une petite partie de la variance.

Le Pourcentage d'Explication de Chaque Composante : Ces pourcentages indiquent la proportion de la variance totale des données expliquée par chaque composante principale. Dans votre cas, la première composante principale explique environ 63% de la variance totale, la deuxième environ 33%, et la troisième seulement environ 4%. Cela signifie que les deux premières composantes principales expliquent la grande majorité de la variance (environ 96%), tandis que la troisième n'apporte qu'une petite contribution.

Dessinez le graphe montrant l'évolution des valeurs propres en fonction de leur rang ?



Ce graphique montre comment les valeurs propres évoluent en fonction de leur rang, ce qui nous aidera à décider combien de composantes principales à conserver dans notre analyse en composantes principales (ACP). Les valeurs propres décroissent à mesure que le rang augmente, et nous pouvons choisir de conserver les premières composantes principales qui expliquent la majeure partie de la variance de nos données.

Quelles variables originelles sont corrélées avec chacun des nouveaux facteurs(Composante) ?

Pour déterminer quelles variables originales sont corrélées avec chacune des nouvelles composantes principales (facteurs) dans une analyse en composantes principales (ACP), nous pouvons examiner les charges factorielles. Les charges factorielles représentent les corrélations entre les variables originales et les composantes principales. Les valeurs absolues des charges factorielles les plus élevées indiquent les variables les plus fortement corrélées à chaque composante.

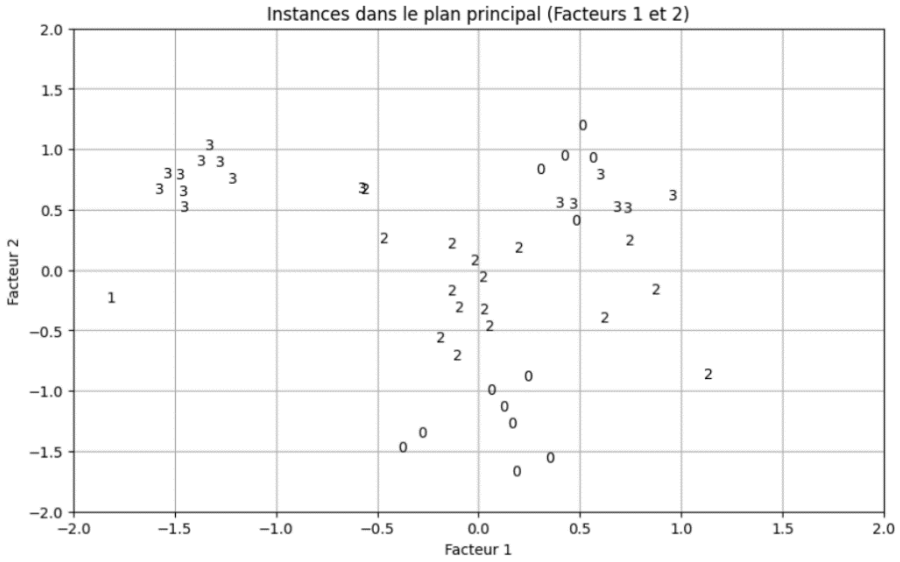
```
Composante Principale 1:
Variable mass : 0.9735
Variable width : -0.0853
Variable color_score : -0.2495

Composante Principale 2:
Variable mass : 0.9732
Variable width : -0.0890
Variable color_score : 0.2494

Composante Principale 3:
Variable mass : -0.1707
Variable width : -0.9940
Variable color_score : -0.0009
```

- **Composante Principale 1** : La variable "mass" a une forte corrélation positive (0.9735) avec cette composante principale, ce qui signifie qu'elle contribue fortement à cette composante. Les variables "width" et "color_score" ont également une corrélation, mais elles sont négatives et moins importantes.

- **Composante Principale 2** : Encore une fois, la variable "mass" a une forte corrélation positive (0.9732) avec cette composante principale. La variable "width" a une corrélation négative, tandis que la variable "color_score" a une corrélation positive.

- **Composante Principale 3** : Dans ce cas, la variable "width" a une corrélation très élevée et négative (-0.9940) avec cette composante principale, ce qui signifie qu'elle est fortement corrélée à cette composante. Les autres variables ont des corrélations proches de zéro.

Tracez les instances dans le plan principal constitué des deux premiers facteurs ?



Les classes des fruits sont-ils bien séparés dans ce nouvel espace ?

- Si les classes de fruits sont bien séparées, cela signifie que l'ACP a permis de réduire la dimensionnalité de manière à ce que les différentes classes soient facilement distinguables.
- En revanche, si les classes se chevauchent, cela peut signifier que la réduction de dimension n'a pas permis une séparation claire des classes.

Calculez les performances en validation croisée en concidérant les deux facteurs principaux de l'ACP

Anciens resultats avec les données originaux

```
Dummy Classifier: Mean Accuracy = 0.34, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.70, Std Deviation = 0.12
Decision Tree Classifier: Mean Accuracy = 0.89, Std Deviation = 0.07
Random Forest Classifier: Mean Accuracy = 0.86, Std Deviation = 0.08
Logistic Regression: Mean Accuracy = 0.64, Std Deviation = 0.22
```

Nouveaux resultats avec les 2 Composants principaux

```
Dummy Classifier: Mean Accuracy = 0.32, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.78, Std Deviation = 0.12
Decision Tree Classifier: Mean Accuracy = 0.78, Std Deviation = 0.10
Random Forest Classifier: Mean Accuracy = 0.85, Std Deviation = 0.11
Logistic Regression: Mean Accuracy = 0.47, Std Deviation = 0.18
```

- Dummy Classifier : Les performances du Dummy Classifier sont légèrement moins bonnes avec les données réduites, mais cela est attendu car il ne fait que des prédictions triviales.

- Gaussian Naive Bayes : Les performances de Naive Bayes ont augmenté avec les données réduites. Cela peut s'expliquer par le fait que Naive Bayes est moins sensible à la dimensionnalité réduite.

- Decision Tree : Les performances du Decision Tree Classifier sont restées relativement stables avec les données réduites.

- Random Forest : Le Random Forest Classifier a montré des performances légèrement meilleures avec les données réduites.

- Logistic Regression : Les performances de la régression logistique ont chuté considérablement avec les données réduites. Cela peut être dû à la perte d'informations importantes lors de la réduction de dimensionnalité.

En général, la réduction de la dimensionnalité à l'aide de l'ACP peut être bénéfique pour certains classifieurs, mais peut également entraîner une perte d'informations, comme le montre la régression logistique. Il est important de choisir la méthode de réduction de dimension et le classifieur en fonction des besoins spécifiques de votre problème. Vous pouvez également essayer d'autres méthodes de réduction de dimension, comme la sélection de variables, pour comparer les performances.

Obtenez-nous les mêmes performances qu'en utilisant deux attributs du fichier original normalisé (par exemple les deux premiers attributs) ?

```
*** Résultats avec les 2 premiers facteurs de l'ACP ***
Dummy Classifier: Mean Accuracy = 0.32, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.78, Std Deviation = 0.12
Decision Tree Classifier: Mean Accuracy = 0.78, Std Deviation = 0.10
Random Forest Classifier: Mean Accuracy = 0.89, Std Deviation = 0.06
Logistic Regression: Mean Accuracy = 0.47, Std Deviation = 0.18
```

```
*** Résultats avec les deux premiers attributs du fichier original normalisé ***
Dummy Classifier: Mean Accuracy = 0.32, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.52, Std Deviation = 0.19
Decision Tree Classifier: Mean Accuracy = 0.76, Std Deviation = 0.13
Random Forest Classifier: Mean Accuracy = 0.85, Std Deviation = 0.13
Logistic Regression: Mean Accuracy = 0.59, Std Deviation = 0.17
```

Avec les deux premiers facteurs de l'ACP :

- Le Dummy Classifier a une performance similaire (0.32 vs 0.32) ;
- Le Gaussian Naive Bayes a une meilleure performance (0.78 vs 0.52) ;
- Le Decision Tree Classifier a une performance similaire (0.78 vs 0.76) ;
- Le Random Forest Classifier a une meilleure performance (0.89 vs 0.85) ;
- La Logistic Regression a une performance inférieure (0.47 vs 0.59).

Cela indique que l'utilisation des deux premiers facteurs de l'ACP a des performances différentes par rapport à l'utilisation des deux premiers attributs originaux normalisés. Les performances varient en fonction de l'algorithme de classification.

L'ACP a permis de réduire la dimension des données tout en conservant certaines informations importantes, mais il peut ne pas être nécessairement meilleur dans tous les cas.
La performance dépendra des caractéristiques des données et de la méthode de classification utilisée.

Sélection de variables Utilisant le sélectionneur SelectKBest avec un critère basé sur l'information mutuelle

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# Créez le sélecteur avec le critère basé sur l'information mutuelle
selector = SelectKBest(score_func=mutual_info_classif, k=2)

# Appliquez la sélection sur vos données originales normalisées
X_two_best_attributes = selector.fit_transform(X_standard, y)

# Obtenez les indices des deux attributs sélectionnés
selected_indices = selector.get_support(indices=True)

# Obtenez les noms des attributs sélectionnés
selected_attributes = X.columns[selected_indices]

print("Les deux attributs les plus importants sont :", selected_attributes)
```

```
Les deux attributs les plus importants sont : Index(['mass', 'color_score'], dtype='object')
```

Quels sont les deux attributs jugés les plus importants par cette méthode ?:

Les deux attributs jugés les plus importants par cette méthode sont : **mass** et **color_score**

Calculez les performances en validation croisée des quatre classifieurs utilisés sur les deux meilleures variables retenues et comparez les résultats précédents calculés sur les deux premiers facteurs de l'ACP ?

```
*** Résultats avec les 2 premiers facteurs de l'ACP ***
Dummy Classifier: Mean Accuracy = 0.32, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.78, Std Deviation = 0.12
Decision Tree Classifier: Mean Accuracy = 0.78, Std Deviation = 0.10

Random Forest Classifier: Mean Accuracy = 0.87, Std Deviation = 0.08
Logistic Regression: Mean Accuracy = 0.47, Std Deviation = 0.18
*** Résultats avec les deux Best premiers attributs du fichier original normalisé ***
Dummy Classifier: Mean Accuracy = 0.32, Std Deviation = 0.02
Gaussian Naive Bayes: Mean Accuracy = 0.74, Std Deviation = 0.06
Decision Tree Classifier: Mean Accuracy = 0.87, Std Deviation = 0.08

Random Forest Classifier: Mean Accuracy = 0.91, Std Deviation = 0.08
Logistic Regression: Mean Accuracy = 0.46, Std Deviation = 0.16
```

En interprétant ces résultats, on peut dire que pour ce jeu de données spécifique, les deux meilleurs attributs du fichier original normalisé semblent être plus informatifs que les deux premiers facteurs de l'ACP pour la plupart des classifieurs. Cependant, il est essentiel de noter que ces performances dépendent du classifieur spécifique utilisé, et il peut y avoir des variations en fonction du modèle choisi. En fin de compte, le choix de la méthode de réduction de dimension ou de sélection de variables dépendra des caractéristiques des données et des objectifs spécifiques du problème.