Exercices sur l'apprentissage supervisé(Machine Learning)

MOISE EHIMIGAYE SENGHOR

UE : Introduction à l'apprentissage supervisé

In [2]:

```python
#Importons les packages nécessaires .
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from summarytools import dfSummary
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import graphviz
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score, confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from matplotlib.ticker import PercentFormatter
from sklearn.metrics import precision_recall_curve, auc
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_absolute_error as MAE, mean_squared_error as MSE
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')#enlever les warnings
```

Projet 1 : les pingouins

- Commencencons par importer les données :

In [3]:

```python
data = pd.read_csv('penguins_size.csv', sep =',')
```

- Exploration du dataset et recapitulatif

In [4]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

In [5]:

```python
data.head()
```

Out[5]:

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE

In [6]:

```python
data.shape
```

Out[6]:

```
(344, 7)
```

In [7]:

```python
data.groupby('species').size()
```

Out[7]:

```
species
Adelie       152
Chinstrap     68
Gentoo       124
dtype: int64
```

In [8]:

```python
# Utilisons le package summary tools pour faire un récapitulatif du jeu de données .
dfSummary(data)
```

Out[8]:

Data Frame Summary

data

Dimensions: 344 x 7

Duplicates: 0

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**species** [object]	1. Adelie 2. Gentoo 3. Chinstrap	152 (44.2%) 124 (36.0%) 68 (19.8%)		0 (0.0%)
2	**island** [object]	1. Biscoe 2. Dream 3. Torgersen	168 (48.8%) 124 (36.0%) 52 (15.1%)		0 (0.0%)
3	**culmen_length_mm** [float64]	Mean (sd) : 43.9 (5.5) min < med < max: 32.1 < 44.5 < 59.6 IQR (CV) : 9.3 (8.0)	164 distinct values		2 (0.6%)
4	**culmen_depth_mm** [float64]	Mean (sd) : 17.2 (2.0) min < med < max: 13.1 < 17.3 < 21.5 IQR (CV) : 3.1 (8.7)	80 distinct values		2 (0.6%)
5	**flipper_length_mm** [float64]	Mean (sd) : 200.9 (14.1) min < med < max: 172.0 < 197.0 < 231.0 IQR (CV) : 23.0 (14.3)	55 distinct values		2 (0.6%)
6	**body_mass_g** [float64]	Mean (sd) : 4201.8 (802.0) min < med < max: 2700.0 < 4050.0 < 6300.0 IQR (CV) : 1200.0 (5.2)	94 distinct values		2 (0.6%)
7	**sex** [object]	1. MALE 2. FEMALE 3. nan 4. .	168 (48.8%) 165 (48.0%) 10 (2.9%) 1 (0.3%)		10 (2.9%)

- Gestion des données manquantes :

In [9]:

```python
data.isnull().sum()
```

Out[9]:

```
species               0
island                0
culmen_length_mm      2
culmen_depth_mm       2
flipper_length_mm     2
body_mass_g           2
sex                  10
dtype: int64
```

In [10]:

```python
data = data.dropna()
data.shape
```

Out[10]:

```
(334, 7)
```

In [11]:

```python
data.isnull().sum() # verification de l'épuration des données manquantes .
```

Out[11]:

```
species              0
island               0
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

Il nous est demandé d'essayer de prédire l'espèce des pingouins en fonction des caractéristiques de leurs culmens. Les variables explicatives seront le bill depth(profondeur du bec) et le bill lenght(longueur du bec). Nous allons commencer par extraire les données que nous utiliserons pour notre modèle.

In [12]:

```python
pingouins = data.iloc[:, [0, 2, 3]]
pingouins.head()#verification
```

Out[12]:

	species	culmen_length_mm	culmen_depth_mm
0	Adelie	39.1	18.7
1	Adelie	39.5	17.4
2	Adelie	40.3	18.0
4	Adelie	36.7	19.3
5	Adelie	39.3	20.6

In [13]:

```python
pingouins.shape
```

Out[13]:

```
(334, 3)
```

Divisons le jeu de données en deux entités une partie pour l'entrainement du modele et l'autre pour le tester.

In [14]:

```python
X= pingouins.iloc[:, [1,2]] # variables explicatives
y = pingouins.iloc[:, [0]]# variable cible
```

In [15]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0) # Les paremetres nous on été c
```

1 - Entrainez un arbre de profondeur 1 :

Creeons le modele et entrainons le:

In [16]:

```python
modele = DecisionTreeClassifier( max_depth=1)#modele
modele.fit(X_train, y_train)#entrainement
```

Out[16]:

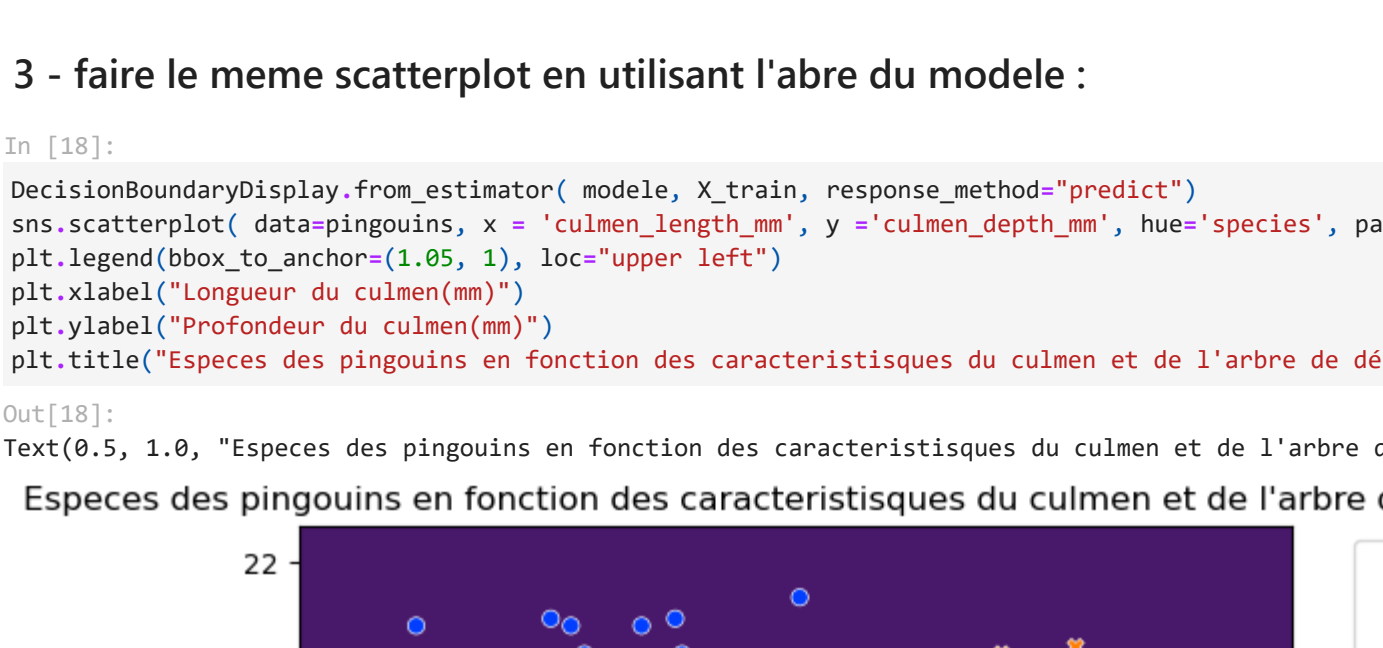
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=1)
```

2- faire une visualisation basée sur les caracteristiques du culmen avec en differenciant les especes des pingouins.

In [17]:

```python
sns.scatterplot(x = 'culmen_length_mm', y ='culmen_depth_mm', hue = 'species', style= 'species', palette='bright', data
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.xlabel("Longueur du culmen(mm)")
plt.ylabel("Profondeur du culmen(mm)")
plt.title("Especes des pingouins en fonction de la longueur et la profondeur du culmen.")
plt.show()
```

Especes des pingouins en fonction de la longueur et la profondeur du culmen.



3 - faire le meme scatterplot en utilisant l'abre du modele :

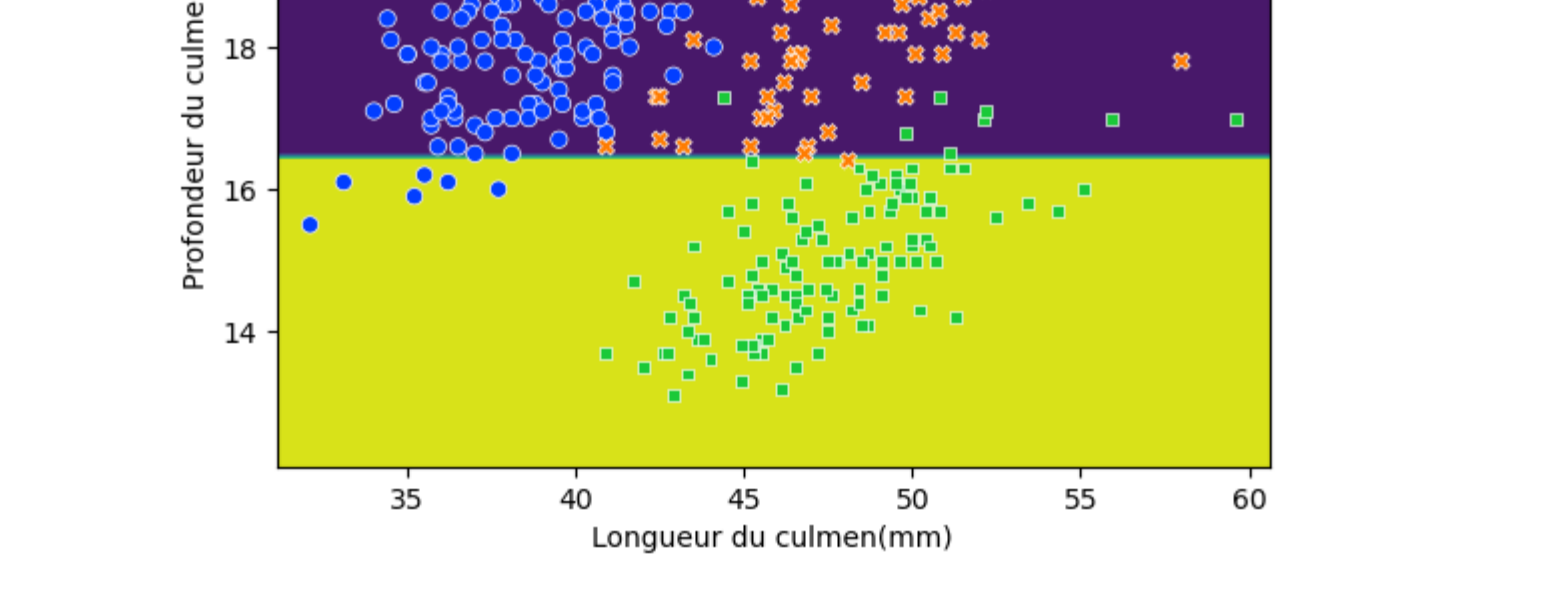
In [18]:

```python
DecisionBoundaryDisplay.from_estimator( modele, X_train, response_method="predict")
sns.scatterplot( data=pingouins, x = 'culmen_length_mm', y ='culmen_depth_mm', hue='species', palette='bright', style='
plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
plt.xlabel("Longueur du culmen(mm)")
plt.ylabel("Profondeur du culmen(mm)")
plt.title("Especes des pingouins en fonction des caracteristisques du culmen et de l'arbre de décision")
```

Out[18]:

```
Text(0.5, 1.0, "Especes des pingouins en fonction des caracteristisques du culmen et de l'arbre de décision")
```

Especes des pingouins en fonction des caracteristisques du culmen et de l'arbre de décision



La variable utilisée pour partitionner le jeu de données est la variable culmen_depth_mm (profondeur du culmen) car on voit que la partition est faite sur l'axe des ordonnées. On peut l'interpreter en disant que la variable profondeur est plus significative et differenciative que la variable longueur.

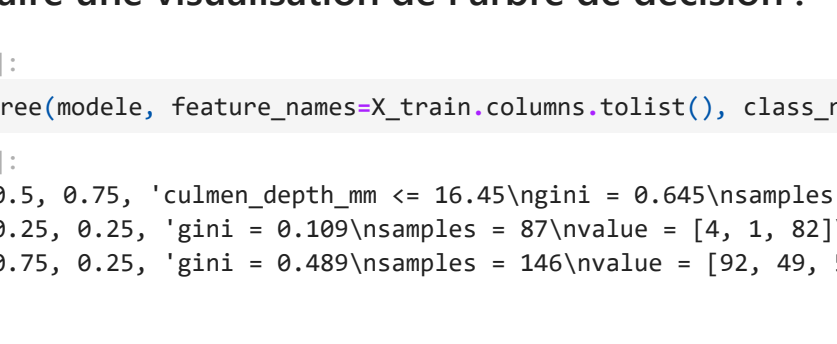
4 - Faire une visualisation de l'arbre de décision :

In [19]:

```python
plot_tree(modele, feature_names=X_train.columns.tolist(), class_names=modele.classes_.tolist())
```

Out[19]:

```
[Text(0.5, 0.75, 'culmen_depth_mm <= 16.45\ngini = 0.645\nsamples = 233\nvalue = [96, 50, 87]\nclass = Adelie'),
 Text(0.25, 0.25, 'gini = 0.109\nsamples = 87\nvalue = [4, 1, 82]\nclass = Gentoo'),
 Text(0.75, 0.25, 'gini = 0.489\nsamples = 146\nvalue = [92, 49, 5]\nclass = Adelie')]
```



Les labels obtenus semblent cohérents, car on observe que, pour les feuilles avec un indice de Gini faible, la différence de population entre l'espèce majoritaire et les autres est nettement marquée. Par exemple, dans la feuille numéro deux, qui présente un indice de Gini de 0.1, la classe Gentoo compte 82 individus sur un total de 87, témoignant d'une forte pureté de la feuille.

En revanche, pour les feuilles où l'indice de Gini est plus élevé, les proportions entre les différentes espèces tendent à s'équilibrer, indiquant une diminution de la pureté. Cela reflète la difficulté du modèle à distinguer clairement les classes dans ces cas, ce qui peut impacter la précision des prédictions.

Par ailleurs, on note que la classe Chinstrap n'est pas représentée dans les feuilles obtenues, alors qu'elle constitue 20 % du jeu de données initial. Cela suggère que cette classe est sous-représentée dans le modèle ou qu'elle n'a pas été correctement prise en compte lors de l'apprentissage. Cette observation pourrait être due à un déséquilibre dans les données ou à un choix de paramètres du modèle favorisant les classes majoritaires.

Ces résultats mettent néanmoins en évidence une certaine efficacité prédictive du modèle, particulièrement pour la classe dominante (Gentoo) dans certaines feuilles. Une analyse complémentaire serait utile pour mieux comprendre les limites du modèle et envisager des ajustements visant à mieux intégrer les classes minoritaires, comme Chinstrap.

5 - Faisons une prédiction à partir du modèle pour un pingouin dont la longueur du bec est 35 mm et dont la profondeur est de 17 mm.

In [20]:

```python
print("Le modèle prédit qu'un pingouin dont la longueur de culmen est de 35 mm et dont la profondeur est de 17 mm est d
```

```
Le modèle prédit qu'un pingouin dont la longueur de culmen est de 35 mm et dont la profondeur est de 17 mm est de l'esp
èce :  ['Adelie']
```

- Visualisation de la probabilité d'appartenance à une espece de l'oiseau cité plus haut :

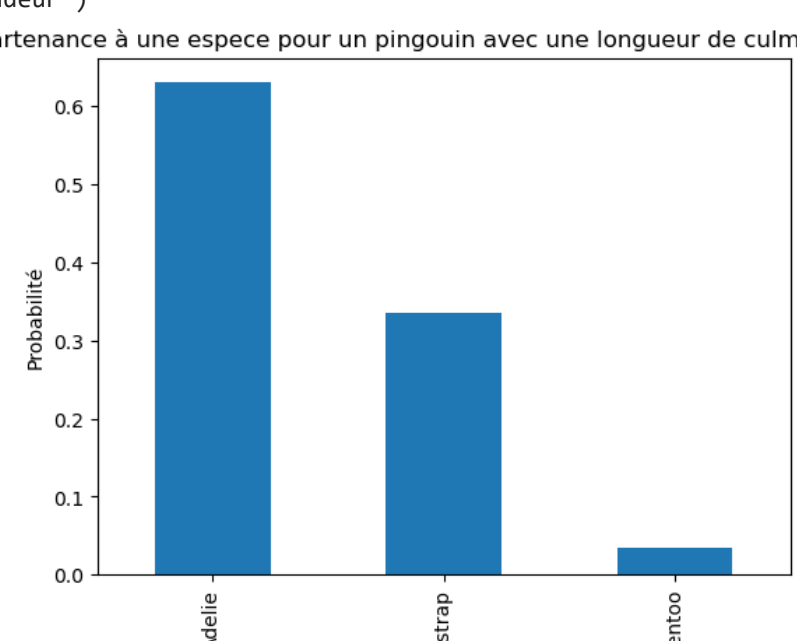
In [21]:

```python
proba = modele.predict_proba(pd.DataFrame({"culmen_length_mm": [35], "culmen_depth_mm": [17]}))#probabilité
Proba = pd.Series(proba[0], index=modele.classes_)#serie pour la visualisation
Proba.plot.bar()#viz
plt.ylabel("Probabilité")
plt.title("Prediction de la probabilité d'appartenance à une espece pour un pingouin avec une longueur de culmen de 35
```

Out[21]:

```
Text(0.5, 1.0, "Prediction de la probabilité d'appartenance à une espece pour un pingouin avec une longueur de culmen d
e 35 mm et de 17 mm de profondeur")
```

Prediction de la probabilité d'appartenance à une espece pour un pingouin avec une longueur de culmen de 35 mm et de 17 mm de profondeur



La prévision correspond à une feuille de l'arbre de décision. En effet, le modèle prédit que si la profondeur du bec est supérieure à 16,45 mm, l'espèce associée à cette feuille est Adélie.

6 - Refaisons la meme chose en faisant un autre arbre de décision avec une profondeur de deux.

In [22]:

```python
# Commencons par créer le modele et par l'entrainer :
modele_2 = DecisionTreeClassifier( max_depth=2)#modele
modele_2.fit(X_train, y_train)#entrainement
```

Out[22]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

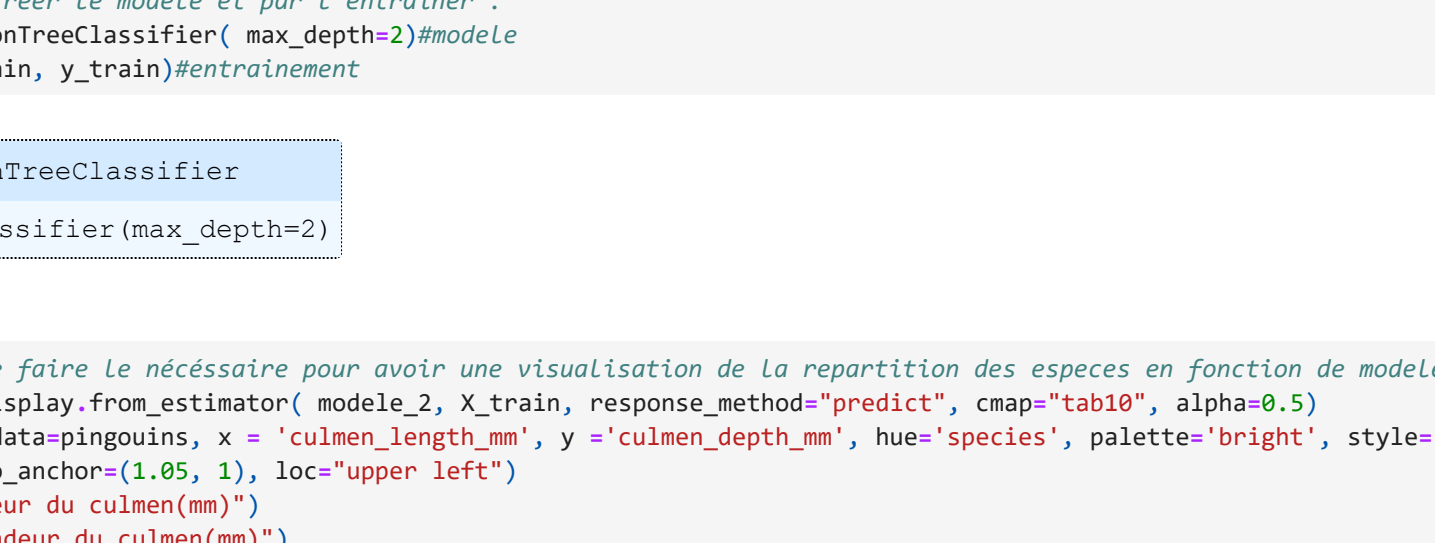
In [23]:

```python
#Essayons aussi de faire le nécéssaire pour avoir une visualisation de la repartition des especes en fonction de modele
DecisionBoundaryDisplay.from_estimator( modele_2, X_train, response_method="predict", cmap="tab10", alpha=0.5)
sns.scatterplot( data=pingouins, x = 'culmen_length_mm', y ='culmen_depth_mm', hue='species', palette='bright', style='
plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
plt.xlabel("Longueur du culmen(mm)")
plt.ylabel("Profondeur du culmen(mm)")
plt.title("Especes des pingouins en fonction des caracteristisques du culmen et de l'arbre de décision (de profondeur 2
```

Out[23]:

```
Text(0.5, 1.0, "Especes des pingouins en fonction des caracteristisques du culmen et de l'arbre de décision (de profond
eur 2)")
```

Especes des pingouins en fonction des caracteristisques du culmen et de l'arbre de décision (de profondeur 2)



In [24]:

```python
# Essayons de déterminer comment l'arbre de décision à fait les dichotomies :
importances = modele_2.feature_importances_
for feature, importance in zip(X_train.columns, importances):
    print(f"{feature}: {importance}")
```

```
culmen_length_mm: 0.46741020473018674
culmen_depth_mm: 0.5325897952698132
```

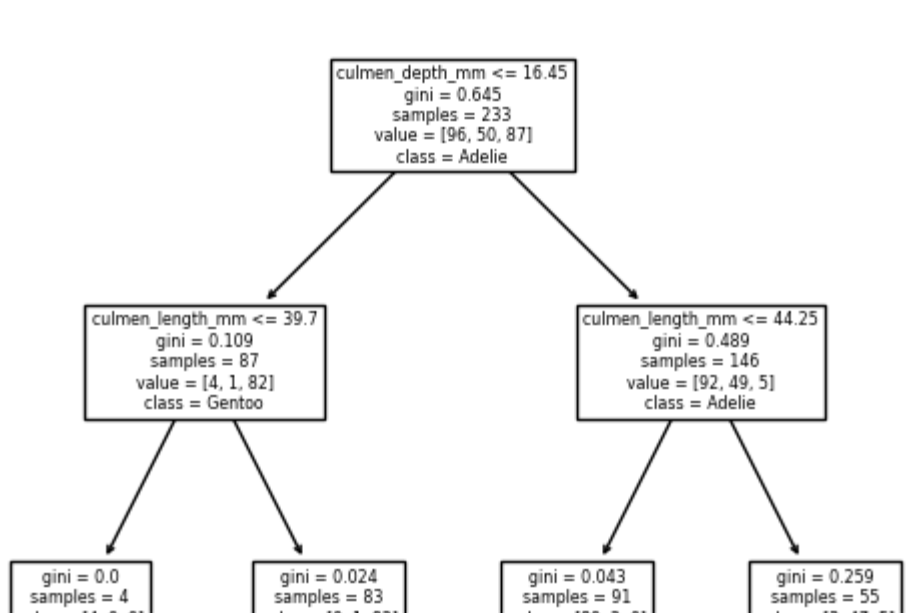
La visualisation met en évidence deux dichotomies successives. L'analyse des feature_importances révèle que la première scission est effectuée sur la profondeur du culmen (culmen_depth_mm), suivie de la longueur du culmen (culmen_length_mm).

In [25]:

```python
# Essayons de visualiser l'arbre de décision :
plot_tree(modele_2, feature_names=X_train.columns.tolist(), class_names=modele_2.classes_.tolist())
```

Out[25]:

```
[Text(0.5, 0.8333333333333334, 'culmen_depth_mm <= 16.45\ngini = 0.645\nsamples = 233\nvalue = [96, 50, 87]\nclass = Ad
elie'),
 Text(0.25, 0.5, 'culmen_length_mm <= 39.7\ngini = 0.109\nsamples = 87\nvalue = [4, 1, 82]\nclass = Gentoo'),
 Text(0.125, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4, 0, 0]\nclass = Adelie'),
 Text(0.375, 0.16666666666666666, 'gini = 0.024\nsamples = 83\nvalue = [0, 1, 82]\nclass = Gentoo'),
 Text(0.75, 0.5, 'culmen_length_mm <= 44.25\ngini = 0.489\nsamples = 146\nvalue = [92, 49, 5]\nclass = Adelie'),
 Text(0.625, 0.16666666666666666, 'gini = 0.043\nsamples = 91\nvalue = [89, 2, 0]\nclass = Adelie'),
 Text(0.875, 0.16666666666666666, 'gini = 0.259\nsamples = 55\nvalue = [3, 47, 5]\nclass = Chinstrap')]
```



On constate une nette amélioration des performances du modèle. En effet, l'indice de Gini de trois des quatre feuilles est inférieur à 0,1, ce qui indique une amélioration significative de la pureté des feuilles par rapport au modèle avec une profondeur limitée à 1.

De plus, on remarque que la classe Chinstrap, auparavant absente, est désormais représentée et prise en compte dans les prédictions, ce qui témoigne d'une meilleure capacité du modèle à intégrer toutes les classes présentes dans le jeu de données.

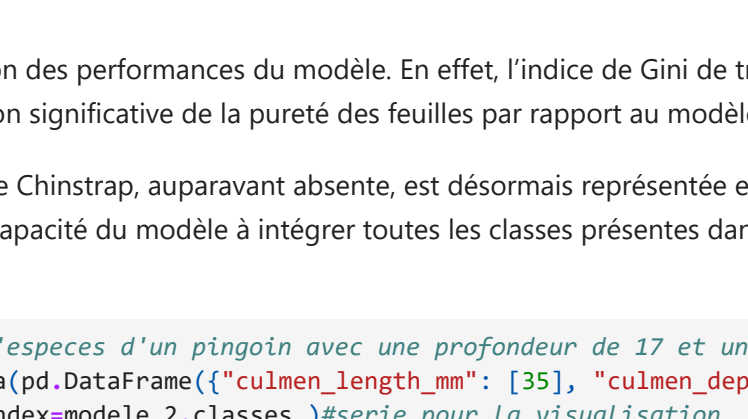
In [26]:

```python
# Faisons une prédiction de l'especes d'un pingoin avec une profondeur de 17 et un longeur de 35 :
proba = modele_2.predict_proba(pd.DataFrame({"culmen_length_mm": [35], "culmen_depth_mm": [17]}))#probabilité
Proba = pd.Series(proba[0], index=modele_2.classes_)#serie pour la visualisation
Proba.plot.bar()#viz
plt.ylabel("Probabilité")
plt.title("Prediction de la probabilité d'appartenance à une espece pour un pingouin avec une longueur de culmen de 35
```

Out[26]:

```
Text(0.5, 1.0, "Prediction de la probabilité d'appartenance à une espece pour un pingouin avec une longueur de culmen d
e 35 mm et de 17 mm de profondeur(modele 2)")
```

Prediction de la probabilité d'appartenance à une espece pour un pingouin avec une longueur de culmen de 35 mm et de 17 mm de profondeur(modele 2)



In [27]:

```python
modele_2.predict_proba(pd.DataFrame({"culmen_length_mm": [35], "culmen_depth_mm": [17]}))
```

Out[27]:

```
array([[0.97802198, 0.02197802, 0.        ]])
```

Projet 2 : Cancer du sein

Exercice 1 : méthode du Hold Out

- Importons le dataset et faire une analyse exploratoire :

In [28]:

```python
cancer = pd.read_csv('cancer.csv')
cancer.head()
```

Out[28]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavit
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

In [29]:

```python
cancer.shape
```

Out[29]:

```
(569, 33)
```

In [30]:

```python
cancer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [31]:

```python
# Résumé statistique du jeu de données :
dfSummary(cancer)
```

Out[31]:

Data Frame Summary

cancer

Dimensions: 569 x 33

Duplicates: 0

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**id** [int64]	Mean (sd) : 30371831.4 (125020585.6) min < med < max: 8670.0 < 906024.0 < 911320502.0 IQR (CV) : 7943911.0 (0.2)	569 distinct values		0 (0.0%)
2	**diagnosis** [object]	1. B 2. M	357 (62.7%) 212 (37.3%)		0 (0.0%)
3	**radius_mean** [float64]	Mean (sd) : 14.1 (3.5) min < med < max: 7.0 < 13.4 < 28.1 IQR (CV) : 4.1 (4.0)	456 distinct values		0 (0.0%)
4	**texture_mean** [float64]	Mean (sd) : 19.3 (4.3) min < med < max: 9.7 < 18.8 < 39.3 IQR (CV) : 5.6 (4.5)	479 distinct values		0 (0.0%)
5	**perimeter_mean** [float64]	Mean (sd) : 92.0 (24.3) min < med < max: 43.8 < 86.2 < 188.5 IQR (CV) : 28.9 (3.8)	522 distinct values		0 (0.0%)
6	**area_mean** [float64]	Mean (sd) : 654.9 (351.9) min < med < max: 143.5 < 551.1 < 2501.0 IQR (CV) : 362.4 (1.9)	539 distinct values		0 (0.0%)
7	**smoothness_mean** [float64]	Mean (sd) : 0.1 (0.0) min < med < max: 0.1 < 0.1 < 0.2 IQR (CV) : 0.0 (6.9)	474 distinct values		0 (0.0%)
8	**compactness_mean** [float64]	Mean (sd) : 0.1 (0.1) min < med < max: 0.0 < 0.1 < 0.3 IQR (CV) : 0.1 (2.0)	537 distinct values		0 (0.0%)
9	**concavity_mean** [float64]	Mean (sd) : 0.1 (0.1) min < med < max: 0.0 < 0.1 < 0.4 IQR (CV) : 0.1 (1.1)	537 distinct values		0 (0.0%)
10	**concave points_mean** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.2 IQR (CV) : 0.1 (1.3)	542 distinct values		0 (0.0%)
11	**symmetry_mean** [float64]	Mean (sd) : 0.2 (0.0) min < med < max: 0.1 < 0.2 < 0.3 IQR (CV) : 0.0 (6.6)	432 distinct values		0 (0.0%)
12	**fractal_dimension_mean** [float64]	Mean (sd) : 0.1 (0.0) min < med < max: 0.0 < 0.1 < 0.1 IQR (CV) : 0.0 (8.9)	499 distinct values		0 (0.0%)
13	**radius_se** [float64]	Mean (sd) : 0.4 (0.3) min < med < max: 0.1 < 0.3 < 2.9 IQR (CV) : 0.2 (1.5)	540 distinct values		0 (0.0%)
14	**texture_se** [float64]	Mean (sd) : 1.2 (0.6) min < med < max: 0.4 < 1.1 < 4.9 IQR (CV) : 0.6 (2.2)	519 distinct values		0 (0.0%)
15	**perimeter_se** [float64]	Mean (sd) : 2.9 (2.0) min < med < max: 0.8 < 2.3 < 22.0 IQR (CV) : 1.8 (1.4)	533 distinct values		0 (0.0%)
16	**area_se** [float64]	Mean (sd) : 40.3 (45.5) min < med < max: 6.8 < 24.5 < 542.2 IQR (CV) : 27.3 (0.9)	528 distinct values		0 (0.0%)
17	**smoothness_se** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.0 IQR (CV) : 0.0 (2.3)	547 distinct values		0 (0.0%)
18	**compactness_se** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.1 IQR (CV) : 0.0 (1.4)	541 distinct values		0 (0.0%)
19	**concavity_se** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.4 IQR (CV) : 0.0 (1.1)	533 distinct values		0 (0.0%)
20	**concave points_se** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.1 IQR (CV) : 0.0 (1.9)	507 distinct values		0 (0.0%)
21	**symmetry_se** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.1 IQR (CV) : 0.0 (2.5)	498 distinct values		0 (0.0%)
22	**fractal_dimension_se** [float64]	Mean (sd) : 0.0 (0.0) min < med < max: 0.0 < 0.0 < 0.0 IQR (CV) : 0.0 (1.4)	545 distinct values		0 (0.0%)
23	**radius_worst** [float64]	Mean (sd) : 16.3 (4.8) min < med < max: 7.9 < 15.0 < 36.0 IQR (CV) : 5.8 (3.4)	457 distinct values		0 (0.0%)
24	**texture_worst** [float64]	Mean (sd) : 25.7 (6.1) min < med < max: 12.0 < 25.4 < 49.5 IQR (CV) : 8.6 (4.2)	511 distinct values		0 (0.0%)
25	**perimeter_worst** [float64]	Mean (sd) : 107.3 (33.6) min < med < max: 50.4 < 97.7 < 251.2 IQR (CV) : 41.3 (3.2)	514 distinct values		0 (0.0%)
26	**area_worst** [float64]	Mean (sd) : 880.6 (569.4) min < med < max: 185.2 < 686.5 < 4254.0 IQR (CV) : 568.7 (1.5)	544 distinct values		0 (0.0%)
27	**smoothness_worst** [float64]	Mean (sd) : 0.1 (0.0) min < med < max: 0.1 < 0.1 < 0.2 IQR (CV) : 0.0 (5.8)	411 distinct values		0 (0.0%)
28	**compactness_worst** [float64]	Mean (sd) : 0.3 (0.2) min < med < max: 0.0 < 0.2 < 1.1 IQR (CV) : 0.2 (1.6)	529 distinct values		0 (0.0%)
29	**concavity_worst** [float64]	Mean (sd) : 0.3 (0.2) min < med < max: 0.0 < 0.2 < 1.3 IQR (CV) : 0.3 (1.3)	539 distinct values		0 (0.0%)
30	**concave points_worst** [float64]	Mean (sd) : 0.1 (0.1) min < med < max: 0.0 < 0.1 < 0.3 IQR (CV) : 0.1 (1.7)	492 distinct values		0 (0.0%)
31	**symmetry_worst** [float64]	Mean (sd) : 0.3 (0.1) min < med < max: 0.2 < 0.3 < 0.7 IQR (CV) : 0.1 (4.7)	500 distinct values		0 (0.0%)
32	**fractal_dimension_worst** [float64]	Mean (sd) : 0.1 (0.0) min < med < max: 0.1 < 0.1 < 0.2 IQR (CV) : 0.0 (4.6)	535 distinct values		0 (0.0%)
33	**Unnamed: 32** [float64]	1. nan	569 (100.0%)		569 (100.0%)

In [32]:

```python
#Le dataset semble correct avec des données manquantes sur une seule variable et des données qui semblent vraisembles.
# Nous avons choisi deux variables non pertinentes la variable 'id' et la 'Unnamed :32'(totalement vide) à élaguer pour
cancer_sein= cancer.drop(['id','Unnamed: 32'], axis =1)
```

In [33]:

```python
cancer_sein.head()
```

Out[33]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 31 columns

1. Divisez aléatoirement le jeu de données en partie train et partie test :

- Choisissons les données prevues pour notre modele en selectionnant certaines variables du jeu de données :

In [34]:

```python
cancer_sein_df = cancer_sein[["diagnosis", "radius_mean","concave points_mean"]]#variables selectionnées
cancer_sein_df['diagnosis'] = cancer_sein_df['diagnosis'].replace({'M': 1, 'B': 0})# labelisées la variable diagnosis
cancer_sein_df.head()#verification
```

Out[34]:

	diagnosis	radius_mean	concave points_mean
0	1	17.99	0.14710
1	1	20.57	0.07017
2	1	19.69	0.12790
3	1	11.42	0.10520
4	1	20.29	0.10430

- divisons le jeu données en features et target :

In [35]:

```python
y = cancer_sein_df['diagnosis'] #variable expliquée
X= cancer_sein_df.drop('diagnosis', axis=1)#variable explicatives
```

- Nous pouvons faire la dichotomie en partie test(20%) et partie entrainement :

In [36]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. Importer DecisionTreeClassifier :

from sklearn.tree import DecisionTreeClassifier

3. Instancier un classifieur :

In [37]:

```python
seed = 1
modele = DecisionTreeClassifier(max_depth=6, random_state=seed)
```

4. Entrainer le modele sur le jeu de données d'entrainement (X train et y train) :

In [38]:

```python
modele.fit(X_train, y_train)
```

Out[38]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=1)
```

5. Predire les cinq premieres instanciations du jeu de données test.

In [39]:

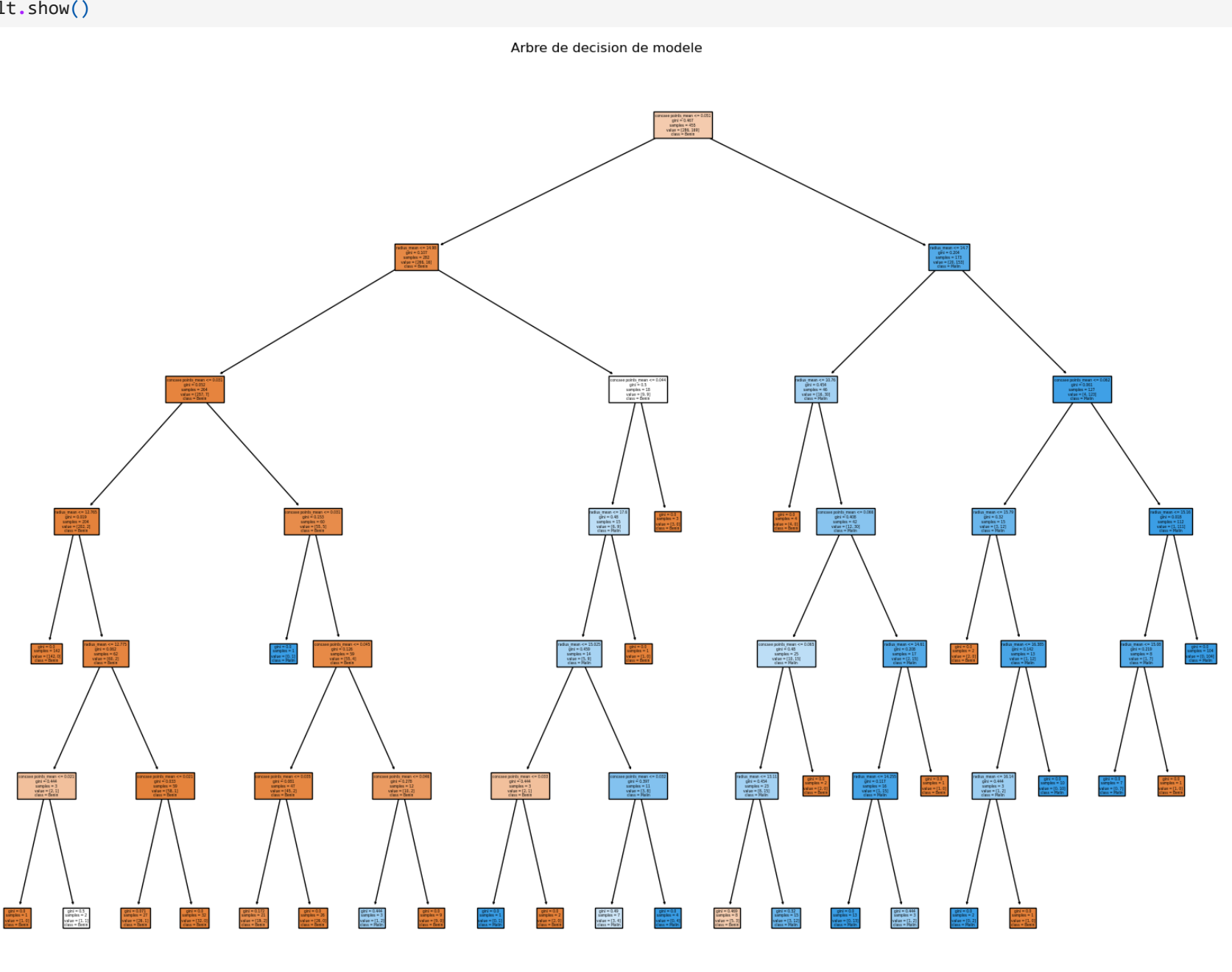
```python
pred = modele.predict(X_test[:5])
print("comparaison entre les cinqs premieres valeurs prédites et les valeurs réels :")
print("Cinq premieres valeurs predites  :", pred)
print("Cinq premieres valeurs reelles :", y_test[:5].values)
```

```
comparaison entre les cinqs premieres valeurs prédites et les valeurs réels :
Cinq premieres valeurs predites  : [0 1 1 0 0]
Cinq premieres valeurs reelles : [0 1 1 0 0]
```

6. Afficher l'arbre de classification correspondant.

In [40]:

```python
plt.figure(figsize=(20, 15))
plot_tree(modele, feature_names=X.columns.tolist(), class_names=['Benin', 'Malin'], filled=True)
plt.title("Arbre de decision de modele ")
plt.show()
```



In [41]:

```python
# La qualité du rendu ne nous satisfait pas nous avons donc décidé de l'améliorer avec le package graphviz :
# fichier dot arbre
dot_data = export_graphviz(modele,
                           out_file=None,
                           feature_names=X.columns.tolist(),
                           class_names=['Benin', 'Malin'],
                           filled=True,
                           rounded=True,
                           special_characters=True)

# faire le graphique
graph = graphviz.Source(dot_data)
graph.render("tree", format="png", cleanup=True)
graph.view()  #afficher
```

Out[41]:

'tree.pdf'

7. Quels problèmes posent la méthode du hold out .

La méthode hold-out, bien qu'appréciée pour sa simplicité, présente plusieurs inconvénients. Tout d'abord, elle est très sensible à la séparation des données : la manière dont les données sont divisées influence directement la qualité du modèle et la représentativité des ensembles d'entraînement et de test.

Ensuite, en réservant une partie des données pour le test, on réduit la quantité d'informations disponibles pour l'entraînement, ce qui peut limiter l'exhaustivité et l'efficacité du modèle, en particulier sur des jeux de données de petite taille.

De plus, cette méthode est sensible à des problèmes comme le sous-apprentissage si l'ensemble d'entraînement est insuffisant, ou au surapprentissage (si le modèle s'ajuste trop spécifiquement à un petit échantillon non représentatif.

Exercice 2 : Choix du critère d'information

1. Importez accuracy score depuis sklearn.metrics from sklearn.metrics import accuracy_score

2. Nommer y pred la prédiction réalisée à partir de X_test :

In [42]:

```python
y_pred = modele.predict(X_test)
```

3. Affichez la valeur de la métrique de performance :

In [43]:

```python
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[43]:

0.8947368421052632

4. Construisez maintenant deux arbres en vous basant sur l'ensemble des features disponibles dans le jeu de données, l'un mobilisant l'entropie comme

critère d'information, et le second mobilisant l'indice de Gini :

- Dataset en tenant compte de toutes les variables explicatives :

In [44]:

```python
cancer_sein1 = cancer_sein
cancer_sein1['diagnosis']= cancer_sein1['diagnosis'].replace({'M': 1, 'B': 0})
cancer_sein1.head()
```

Out[44]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 31 columns

- Division du jeu de données en features et target :

In [45]:

```python
y = cancer_sein1['diagnosis']
X= cancer_sein1.drop('diagnosis', axis=1)
```

- Division du jeu données en partie test et entrainement :

In [46]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Modele avec comme critere l'indice de Gini :

In [47]:

```python
seed = 1
modele_gini= DecisionTreeClassifier(max_depth=6, criterion = 'gini',random_state=seed)
modele_gini.fit(X_train, y_train)
```

Out[47]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=1)
```

- Modele avec comme critere l'entropie :

In [48]:

```python
modele_entropy= DecisionTreeClassifier(max_depth=6, criterion = 'entropy',random_state=seed)
modele_entropy.fit(X_train, y_train)
```

Out[48]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=1)
```

5. Expliquez les principales différences entre les deux métriques

Ces deux métriques sont utilisées pour mesurer l'impureté dans les arbres de décision. Elles reposent sur des concepts différents : l'indice de Gini calcule la probabilité d'une erreur dans la classification, tandis que l'entropie mesure le désordre (ou l'incertitude) dans une feuille ou un nœud.

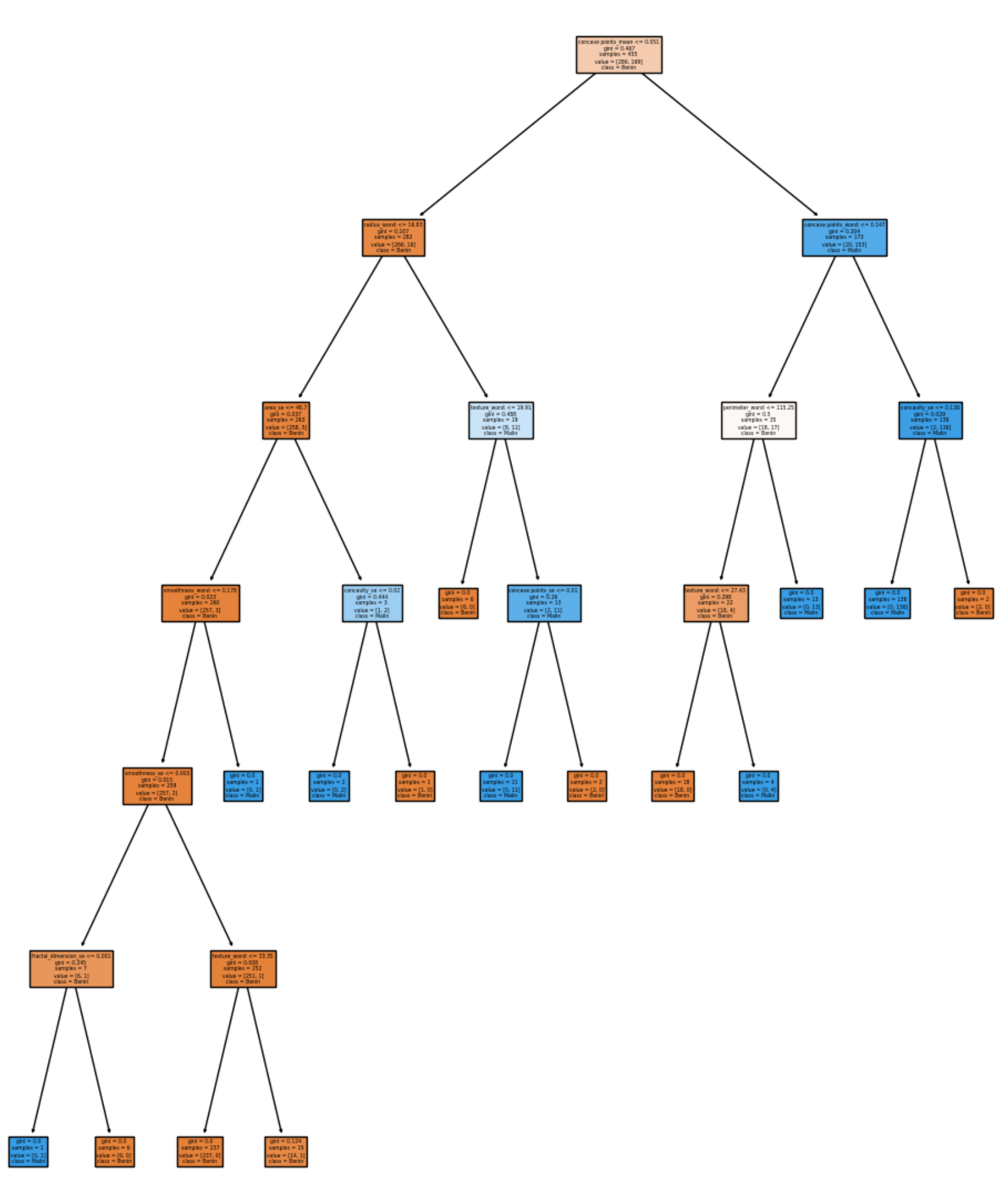
Cependant, la première est plus facile à calculer et plus simple à appréhender que la seconde, qui, elle, est plus intéressante d'un point de vue statistique.

6. Affichez les arbres de classification correspondants

- Plot tree modele indice de gini :

In [49]:

```python
# Visualisation de l'arbre avec le critère Gini
plt.figure(figsize=(12, 15))
plot_tree(modele_gini, feature_names=X.columns.tolist(), class_names=['Benin', 'Malin'], filled=True)
plt.show()

#avec graphviz :
dot_data = export_graphviz(modele_gini,
                           out_file=None,
                           feature_names=X.columns.tolist(),
                           class_names=['Benin', 'Malin'],
                           filled=True,
                           rounded=True,
                           special_characters=True)


graph = graphviz.Source(dot_data)
graph.render("tree_gini", format="png", cleanup=True)
graph.view()
```



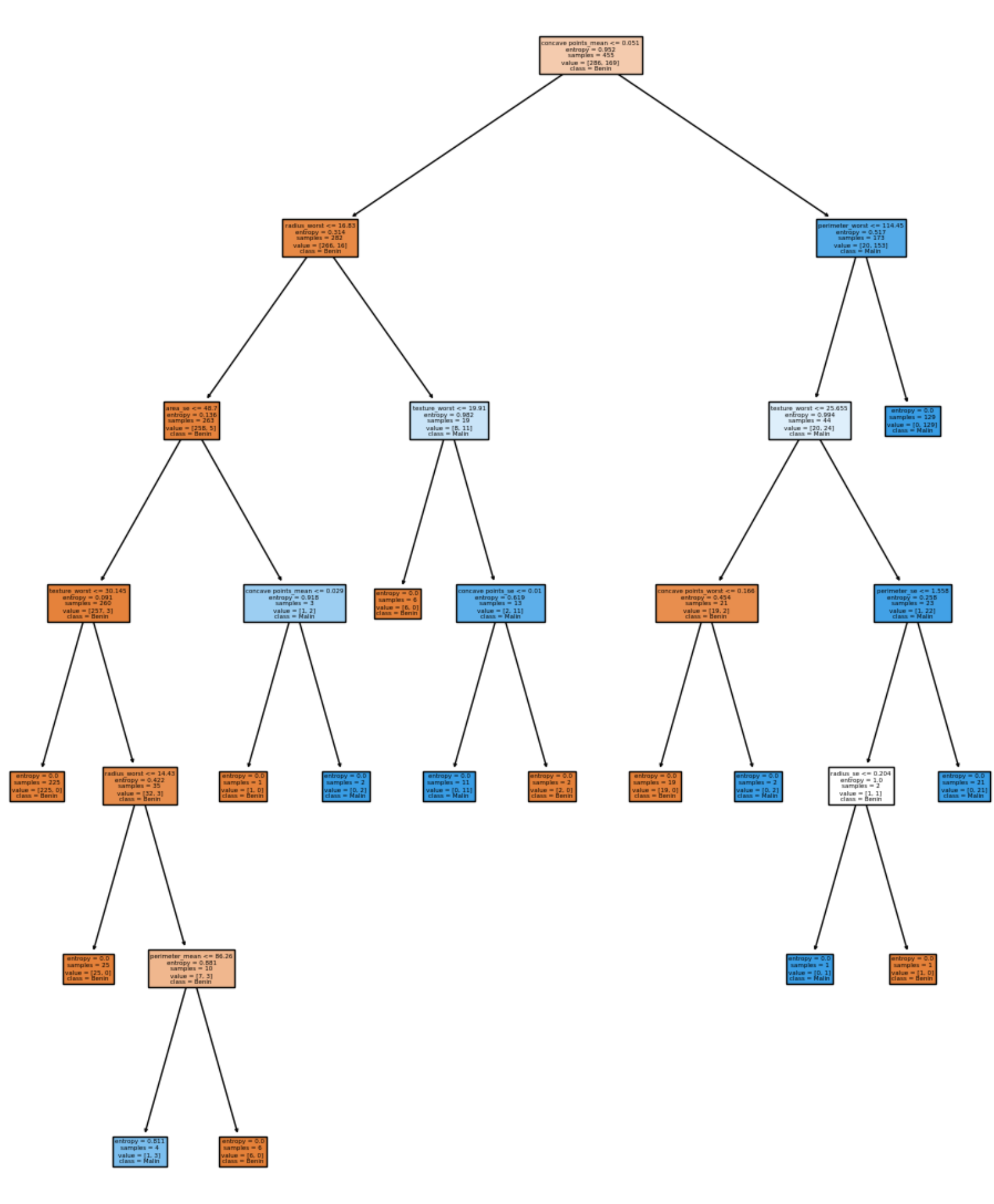
Out[49]:

'tree_gini.pdf'

- Plot tree modele indice de Entropy :

In [50]:

```python
# Visualisation de l'arbre avec le critère Entropy
plt.figure(figsize=(12, 15))
plot_tree(modele_entropy, feature_names=X.columns.tolist(), class_names=['Benin', 'Malin'], filled=True)
plt.show()

#Avec graphviz
dot_data = export_graphviz(modele_entropy,
                           out_file=None,
                           feature_names=X.columns.tolist(),
                           class_names=['Benin', 'Malin'],
                           filled=True,
                           rounded=True,
                           special_characters=True)


graph = graphviz.Source(dot_data)
graph.render("tree_entropy", format="png", cleanup=True)
graph.view()
```



Out[50]:

'tree_entropy.pdf'

7. Comparez les deux approches en utilisant la métrique de performance

précédente

- Prediction des deux modeles :

In [51]:

```python
y_pred_gini = modele_gini.predict(X_test) #indice de gini
y_pred_entropy = modele_entropy.predict(X_test) #entropy
```

- Accuracy modele avec indice de Gini :

In [52]:

```python
accuracy_gini = accuracy_score(y_test, y_pred_gini)
round(accuracy_gini, 3)
```

Out[52]:

0.939

- Accuracy modele Entropie :

In [53]:

```python
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
round(accuracy_entropy,3)
```

Out[53]:

0.947

Les résultats sont sensiblement similaires, avec un léger avantage pour l'entropie. Cela pourrait indiquer que l'entropie permet d'affiner davantage les prédictions par rapport à l'indice de Gini.

Compte tenu de leurs spécificités, on peut émettre l'hypothèse que le choix de l'indice de Gini reste judicieux lorsque la rapidité est prioritaire, tandis que l'entropie serait plus adaptée si l'exhaustivité et la précision des résultats sont privilégiées.

Exercice 3 : métriques de performance

Importer le package necessaire : from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score, confusion_matrix

-Commencons par generer une matrice de confusion comme nous sommes sur un modéle de classification binaire :

In [54]:

```python
from sklearn.metrics import confusion_matrix
matrice = confusion_matrix(y_test, y_pred_gini)
tn_gini, fp_gini, fn_gini, tp_gini = matrice.ravel()
print("Vrais Négatifs (TN):", tn_gini)
print("Faux positifs (FP):", fp_gini)
print("Faux Negatifs (FN):", fn_gini)
print("Vrais Positfs (TP):", tp_gini)
```

```
Vrais Négatifs (TN): 68
Faux positifs (FP): 3
Faux Negatifs (FN): 4
Vrais Positfs (TP): 39
```

1. Accuracy

In [55]:

```python
print('Accuracy :',round(accuracy_score(y_test, y_pred_gini), 4))
```

```
Accuracy : 0.9386
```

- Definition : L'accuracy mesure l'efficacité d'un modèle à prédire correctement à la fois les individus positifs et négatifs.
- Formule : (TP + FP)/(TN+FP+FN+TP)
- Avantages : simplicité, moindre cout, mesures globales et intuitives
- Inconvenients : Innefficace lorsque les classes sont désiquilibrées, pas de prise en compte des erreurs

2. Rappel (Recall-sensibilité)

In [56]:

```python
print('Rappel (Recall-sensibilité) :',round(recall_score(y_test, y_pred_gini), 4))
```

```
Rappel (Recall-sensibilité) : 0.907
```

- Definition : Le rappel est une mesure de la sensibilité d'un classificateur ou d'un prédicteur dans l'apprentissage automatique (ML). Il est calculé comme la proportion du nombre total de cas positifs réels par rapport au nombre de prédictions positives réelles faites par le classificateur.
- Formule : TP/(TP+FN)
- Avantages : Utiles pour la distinction des cas positifs et minimise les Faux négatifs
- Inconvenients : Pas prise en compte des erreurs et aussi risques de taux de faux positifs non detecté élévés

3. Precision

In [57]:

```python
print(' Precision :', round(precision_score(y_test, y_pred_gini),4))
```

```
 Precision : 0.9286
```

- Definition : La precision est également appelée Positive Predictive Value. Elle correspond au taux de prédictions correctes parmi les prédictions positives
- Formule : TP/(TP+FP)
- Avantages : Baisses des faux positifs, garantie sur les predictions
- Inconvenients : Non prise en compte des faux négatifs, sensibles aux classes déséquilibrées

4. Spécificité

In [58]:

```python
print("Spécificité :",round(tn_gini / (tn_gini + fp_gini),4))
```

```
Spécificité : 0.9577
```

- Definition :a spécificité mesure la proportion des vrais négatifs parmi tous les exemples réellement négatifs. En d'autres termes, elle indique à quel point un modèle est capable de réduire les faux positifs.
- Formule : TN/(TN+FP)
- Avantages : Baisse du taux faux positifs, complementarité avec la sensibilite
- Inconvenients : sensible aux classes déséquilibrées et non prise encombre des faux négatifs

5. F1-Score

In [59]:

```python
print( 'F1-Score ', round( f1_score(y_test, y_pred_gini), 4))
```

```
F1-Score  0.9176
```

- Definition : Metrique combinant précision et sensibilité utile pour les classes déséquilibrées
- Formule : (2× précision × sensibilité) / (précision + sensibilité)
- Avantages : combine précision et sensibilité et leur avantages, prises en charge du biais des classes désiquilibrées, Réduction de l'Impact des Faux Positifs et Faux Négatifs
- Inconvenients :Ne Prend Pas en Compte les Vrais Négatifs, Ne Fournit Pas d'Information Complète, Sensibilité aux Variations de Précision et de Rappel ,Pas Toujours Représentatif de l'Impact des Erreurs

6. AUC

In [60]:

```python
print('AUC :', round(roc_auc_score(y_test, y_pred_gini),4))
```

```
AUC : 0.9324
```

- Definition : Le terme AUC signifie Area Under the Curve, aire sous la courbe en français. La métrique AUC PR consiste donc à calculer l'aire sous la courbe Precision Recall.
- Formule : [

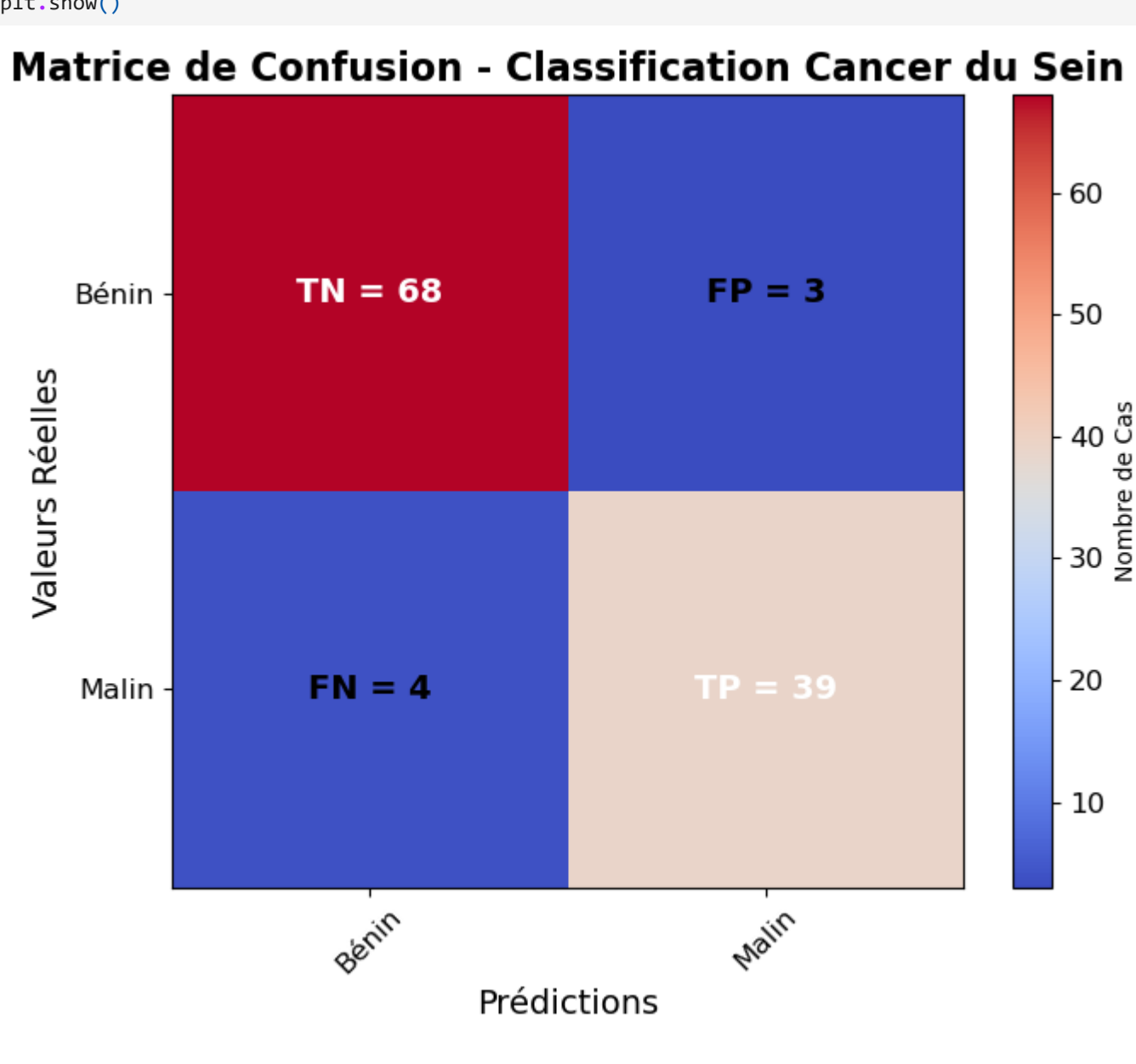
\text{AUC} = \sum_{i=1}^{n-1} \frac{(FPR_{i+1} - FPR_i) \times (TPR_{i+1} + TPR_i)}{2}]

- Avantages :Facilité d'Interprétation, utilité dans les Cas de Classes Déséquilibrées
- Inconvenients : peu representativite des classes minoritaires, non prises en compte des erreurs, sensibles aux données d'évaluation

1. Trouvez une méthode pour afficher la matrice de confusion du modéle

In [61]:

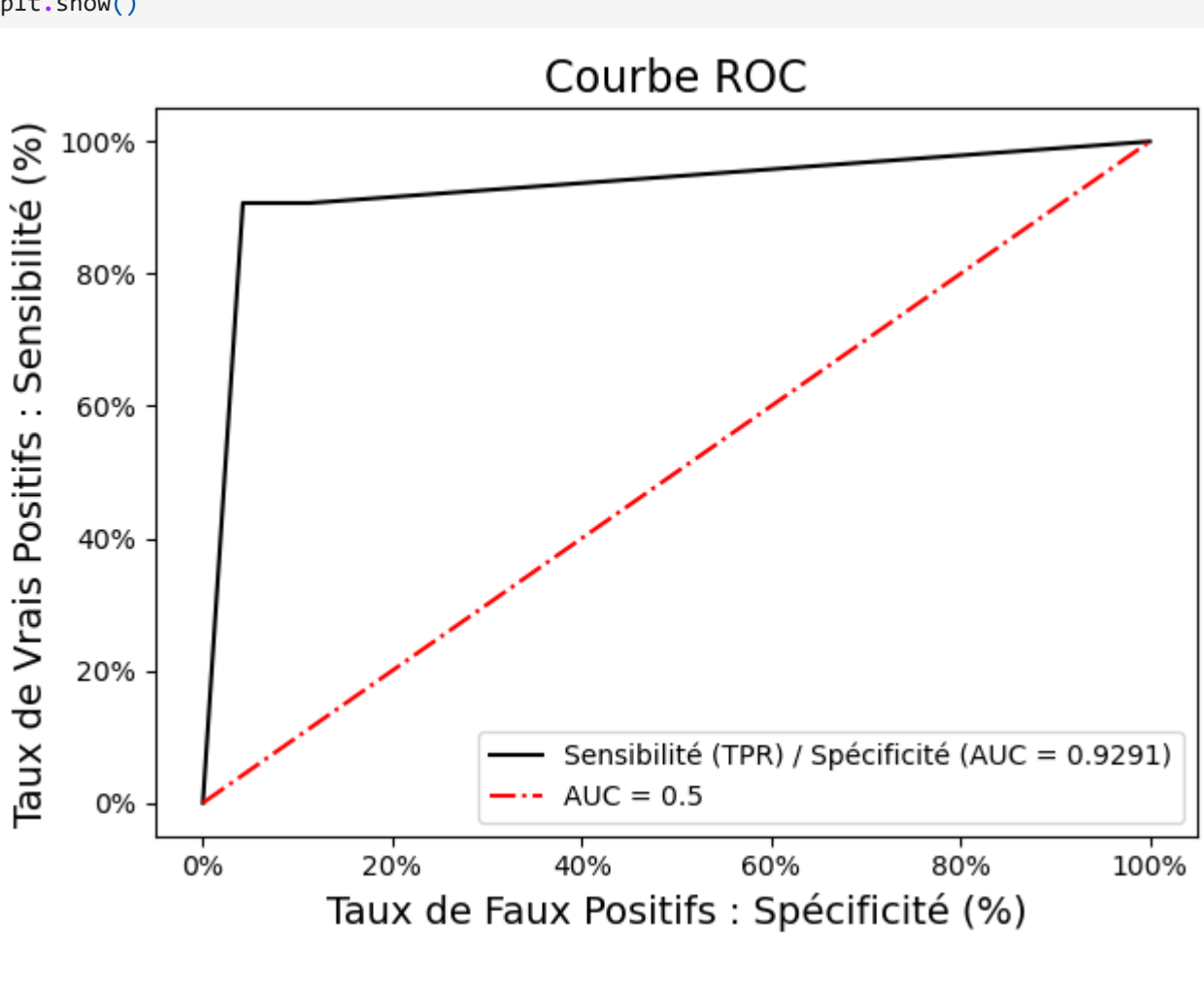
```python
plt.figure(figsize=(8, 6))
plt.imshow(matrice, interpolation='nearest', cmap='coolwarm')
classNames = ['Bénin', 'Malin']
tick_marks = np.arange(len(classNames))

# Legendes ordonnées et abscisses
plt.xticks(tick_marks, classNames, rotation=45, fontsize=12)
plt.yticks(tick_marks, classNames, fontsize=12)

#valeurs des labels
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        color = "white" if matrice[i][j] > matrice.max() / 2 else "black"
        plt.text(j, i, f"{s[i][j]} = {matrice[i][j]}", ha='center', va='center', color=color, fontsize=14, weight='bold

# Customiser
plt.title("Matrice de Confusion - Classification Cancer du Sein", fontsize=16, weight='bold')
plt.xlabel('Prédictions', fontsize=14)
plt.ylabel('Valeurs Réelles', fontsize=14)
cbar = plt.colorbar(label='Nombre de Cas', fraction=0.046, pad=0.04)
cbar.ax.tick_params(labelsize=12)
plt.tight_layout()
plt.show()
```



2. Affichez la courbe ROC et l'AUC correspondant.

In [62]:

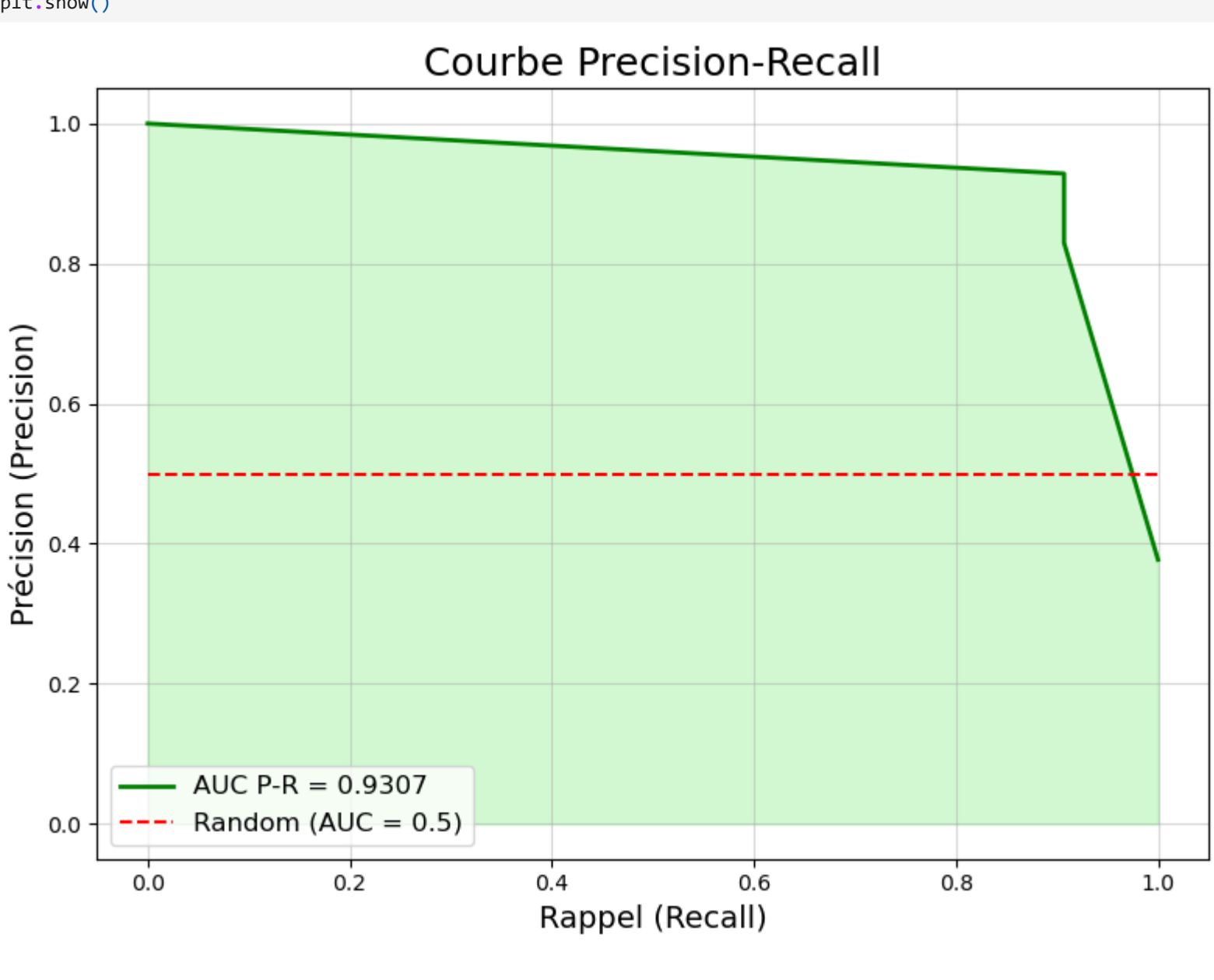
```python
y_pred_proba = modele_gini.predict_proba(X_test)[:, 1]
#ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
#AUC
auc = roc_auc_score(y_test, y_pred_proba)
#  courbe ROC
plt.plot(fpr, tpr, color='black', label=f'Sensibilité (TPR) / Spécificité (AUC = {auc:.4f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='-.', label='AUC = 0.5')

# Customiser
plt.title("Courbe ROC", fontsize=16)
plt.xlabel("Taux de Faux Positifs : Spécificité (%)", fontsize=14)
plt.ylabel("Taux de Vrais Positifs : Sensibilité (%)", fontsize=14)
plt.legend(loc='lower right')
plt.gca().xaxis.set_major_formatter(PercentFormatter(1))
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))
plt.tight_layout()
plt.show()
```



3. Affichez la courbe Precision-Recall et l'AUC P-R correspondant.

In [63]:

```python
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt


# Calcul des valeurs pour la courbe Precision-Recall
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

# Calcul de l'AUC Precision-Recall
auc_pr = auc(recall, precision)

# Affichage de la courbe Precision-Recall
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='green', linewidth=2, label=f'AUC P-R = {auc_pr:.4f}')
plt.fill_between(recall, precision, color='lightgreen', alpha=0.4)  # Remplissage sous la courbe

# Ligne aléatoire de référence
plt.plot([0, 1], [0.5, 0.5], color='red', linestyle='--', label='Random (AUC = 0.5)')

# Titre et labels des axes
plt.title("Courbe Precision-Recall", fontsize=18)
plt.xlabel("Rappel (Recall)", fontsize=14)
plt.ylabel("Précision (Precision)", fontsize=14)

# Légende et autres personnalisations
plt.legend(loc="lower left", fontsize=12)
plt.grid(True, linestyle='-', alpha=0.5)  # Grille avec plus de visibilité

# Améliorer la mise en page
plt.tight_layout()

# Affichage de la courbe
plt.show()
```



Projet 3 : Consommation d'essence

Exercice 1 : Arbre de régression et RMSE

- Essayons d'abord de faire une petite analyse exploratoire :

In [64]:

```python
df = pd.read_csv('auto-mpg.csv')
```

In [65]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car name      398 non-null    object
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

In [66]:

```python
df.head()
```

Out[66]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [67]:

```python
df.isnull().sum()
```

Out[67]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

In [68]:
```
dfSummary(df)
```

Out[68]:

Data Frame Summary
df
Dimensions: 398 x 9
Duplicates: 0

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**mpg** [float64]	Mean (sd) : 23.5 (7.8) min < med < max: 9.0 < 23.0 < 46.6 IQR (CV) : 11.5 (3.0)	129 distinct values		0 (0.0%)
2	**cylinders** [int64]	Mean (sd) : 5.5 (1.7) min < med < max: 3.0 < 4.0 < 8.0 IQR (CV) : 4.0 (3.2)	5 distinct values		0 (0.0%)
3	**displacement** [float64]	Mean (sd) : 193.4 (104.3) min < med < max: 68.0 < 148.5 < 455.0 IQR (CV) : 157.8 (1.9)	82 distinct values		0 (0.0%)
4	**horsepower** [object]	1. 150 2. 90 3. 88 4. 110 5. 100 6. 75 7. 95 8. 105 9. 70 10. 67 11. other	22 (5.5%) 20 (5.0%) 19 (4.8%) 18 (4.5%) 17 (4.3%) 14 (3.5%) 14 (3.5%) 12 (3.0%) 12 (3.0%) 12 (3.0%) 238 (59.8%)		0 (0.0%)
5	**weight** [int64]	Mean (sd) : 2970.4 (846.8) min < med < max: 1613.0 < 2803.5 < 5140.0 IQR (CV) : 1384.2 (3.5)	351 distinct values		0 (0.0%)
6	**acceleration** [float64]	Mean (sd) : 15.6 (2.8) min < med < max: 8.0 < 15.5 < 24.8 IQR (CV) : 3.3 (5.6)	95 distinct values		0 (0.0%)
7	**model year** [int64]	Mean (sd) : 76.0 (3.7) min < med < max: 70.0 < 76.0 < 82.0 IQR (CV) : 6.0 (20.6)	13 distinct values		0 (0.0%)
8	**origin** [int64]	Mean (sd) : 1.6 (0.8) min < med < max: 1.0 < 1.0 < 3.0 IQR (CV) : 1.0 (2.0)	3 distinct values		0 (0.0%)
9	**car name** [object]	1. ford pinto 2. toyota corolla 3. amc matador 4. ford maverick 5. chevrolet chevette 6. amc gremlin 7. chevrolet impala 8. peugeot 504 9. amc hornet 10. toyota corona 11. other	6 (1.5%) 5 (1.3%) 5 (1.3%) 5 (1.3%) 4 (1.0%) 4 (1.0%) 4 (1.0%) 4 (1.0%) 4 (1.0%) 4 (1.0%) 353 (88.7%)		0 (0.0%)

Les variables objet sont les variables 'horsepower' et 'car name' :

In [69]:
```
df['car name'].unique()
```

Out[69]:
```
array(['chevrolet chevelle malibu', 'buick skylark 320',
       'plymouth satellite', 'amc rebel sst', 'ford torino',
       'ford galaxie 500', 'chevrolet impala', 'plymouth fury iii',
       'pontiac catalina', 'amc ambassador dpl', 'dodge challenger se',
       "plymouth 'cuda 340", 'chevrolet monte carlo',
       'buick estate wagon (sw)', 'toyota corona mark ii',
       'plymouth duster', 'amc hornet', 'ford maverick', 'datsun pl510',
       'volkswagen 1131 deluxe sedan', 'peugeot 504', 'audi 100 ls',
       'saab 99e', 'bmw 2002', 'amc gremlin', 'ford f250', 'chevy c20',
       'dodge d200', 'hi 1200d', 'chevrolet vega 2300', 'toyota corona',
       'ford pinto', 'plymouth satellite custom', 'ford torino 500',
       'amc matador', 'pontiac catalina brougham', 'dodge monaco (sw)',
       'ford country squire (sw)', 'pontiac safari (sw)',
       'amc hornet sportabout (sw)', 'chevrolet vega (sw)',
       'pontiac firebird', 'ford mustang', 'mercury capri 2000',
       'opel 1900', 'peugeot 304', 'fiat 124b', 'toyota corolla 1200',
       'datsun 1200', 'volkswagen model 111', 'plymouth cricket',
       'toyota corona hardtop', 'dodge colt hardtop', 'volkswagen type 3',
       'chevrolet vega', 'ford pinto runabout', 'amc ambassador sst',
       'mercury marquis', 'buick lesabre custom',
       'oldsmobile delta 88 royale', 'chrysler newport royal',
       'mazda rx2 coupe', 'amc matador (sw)',
       'chevrolet chevelle concours (sw)', 'ford gran torino (sw)',
       'plymouth satellite custom (sw)', 'volvo 145e (sw)',
       'volkswagen 411 (sw)', 'peugeot 504 (sw)', 'renault 12 (sw)',
       'ford pinto (sw)', 'datsun 510 (sw)',
       'toyouta corona mark ii (sw)', 'dodge colt (sw)',
       'toyota corolla 1600 (sw)', 'buick century 350',
       'chevrolet malibu', 'ford gran torino', 'dodge coronet custom',
       'mercury marquis brougham', 'chevrolet caprice classic',
       'ford ltd', 'plymouth fury gran sedan',
       'chrysler new yorker brougham', 'buick electra 225 custom',
       'amc ambassador brougham', 'plymouth valiant',
       'chevrolet nova custom', 'volkswagen super beetle', 'ford country',
       'plymouth custom suburb', 'oldsmobile vista cruiser',
       'toyota carina', 'datsun 610', 'maxda rx3', 'mercury capri v6',
       'fiat 124 sport coupe', 'chevrolet monte carlo s',
       'pontiac grand prix', 'fiat 128', 'opel manta', 'audi 100ls',
       'volvo 144ea', 'dodge dart custom', 'saab 99le', 'toyota mark ii',
       'oldsmobile omega', 'chevrolet nova', 'datsun b210',
       'chevrolet chevelle malibu classic', 'plymouth satellite sebring',
       'buick century luxus (sw)', 'dodge coronet custom (sw)',
       'audi fox', 'volkswagen dasher', 'datsun 710', 'dodge colt',
       'fiat 124 tc', 'honda civic', 'subaru', 'fiat x1.9',
       'plymouth valiant custom', 'mercury monarch', 'chevrolet bel air',
       'plymouth grand fury', 'buick century',
       'chevroelt chevelle malibu', 'plymouth fury', 'buick skyhawk',
       'chevrolet monza 2+2', 'ford mustang ii', 'toyota corolla',
       'pontiac astro', 'volkswagen rabbit', 'amc pacer', 'volvo 244dl',
       'honda civic cvcc', 'fiat 131', 'capri ii', 'renault 12tl',
       'dodge coronet brougham', 'chevrolet chevette', 'chevrolet woody',
       'vw rabbit', 'dodge aspen se', 'ford granada ghia',
       'pontiac ventura sj', 'amc pacer d/l', 'datsun b-210', 'volvo 245',
       'plymouth volare premier v8', 'mercedes-benz 280s',
       'cadillac seville', 'chevy c10', 'ford f108', 'dodge d100',
       'honda accord cvcc', 'buick opel isuzu deluxe', 'renault 5 gtl',
       'plymouth arrow gs', 'datsun f-10 hatchback',
       'oldsmobile cutlass supreme', 'dodge monaco brougham',
       'mercury cougar brougham', 'chevrolet concours', 'buick skylark',
       'plymouth volare custom', 'ford granada', 'pontiac grand prix lj',
       'chevrolet monte carlo landau', 'chrysler cordoba',
       'ford thunderbird', 'volkswagen rabbit custom',
       'pontiac sunbird coupe', 'toyota corolla liftback',
       'ford mustang ii 2+2', 'dodge colt m/m', 'subaru dl', 'datsun 810',
       'bmw 320i', 'mazda rx-4', 'volkswagen rabbit custom diesel',
       'ford fiesta', 'mazda glc deluxe', 'datsun b210 gx',
       'oldsmobile cutlass salon brougham', 'dodge diplomat',
       'mercury monarch ghia', 'pontiac phoenix lj',
       'ford fairmont (auto)', 'ford fairmont (man)', 'plymouth volare',
       'amc concord', 'buick century special', 'mercury zephyr',
       'dodge aspen', 'amc concord d/l',
       'buick regal sport coupe (turbo)', 'ford futura',
       'dodge magnum xe', 'datsun 510', 'dodge omni',
       'toyota celica gt liftback', 'plymouth sapporo',
       'oldsmobile starfire sx', 'datsun 200-sx', 'audi 5000',
       'volvo 264gl', 'saab 99gle', 'peugeot 604sl',
       'volkswagen scirocco', 'honda accord lx', 'pontiac lemans v6',
       'mercury zephyr 6', 'ford fairmont 4', 'amc concord dl 6',
       'dodge aspen 6', 'ford ltd landau', 'mercury grand marquis',
       'dodge st. regis', 'chevrolet malibu classic (sw)',
       'chrysler lebaron town @ country (sw)', 'vw rabbit custom',
       'maxda glc deluxe', 'dodge colt hatchback custom', 'amc spirit dl',
       'mercedes benz 300d', 'cadillac eldorado', 'plymouth horizon',
       'plymouth horizon tc3', 'datsun 210', 'fiat strada custom',
       'buick skylark limited', 'chevrolet citation',
       'oldsmobile omega brougham', 'pontiac phoenix',
       'toyota corolla tercel', 'datsun 310', 'ford fairmont',
       'audi 4000', 'toyota corona liftback', 'mazda 626',
       'datsun 510 hatchback', 'mazda glc', 'vw rabbit c (diesel)',
       'vw dasher (diesel)', 'audi 5000s (diesel)', 'mercedes-benz 240d',
       'honda civic 1500 gl', 'renault lecar deluxe', 'vokswagen rabbit',
       'datsun 280-zx', 'mazda rx-7 gs', 'triumph tr7 coupe',
       'ford mustang cobra', 'honda accord', 'plymouth reliant',
       'dodge aries wagon (sw)', 'toyota starlet', 'plymouth champ',
       'honda civic 1300', 'datsun 210 mpg', 'toyota tercel',
       'mazda glc 4', 'plymouth horizon 4', 'ford escort 4w',
       'ford escort 2h', 'volkswagen jetta', 'renault 18i',
       'honda prelude', 'datsun 200sx', 'peugeot 505s turbo diesel',
       'volvo diesel', 'toyota cressida', 'datsun 810 maxima',
       'oldsmobile cutlass ls', 'ford granada gl',
       'chrysler lebaron salon', 'chevrolet cavalier',
       'chevrolet cavalier wagon', 'chevrolet cavalier 2-door',
       'pontiac j2000 se hatchback', 'dodge aries se',
       'ford fairmont futura', 'amc concord dl', 'volkswagen rabbit l',
       'mazda glc custom l', 'mazda glc custom', 'plymouth horizon miser',
       'mercury lynx l', 'nissan stanza xe', 'honda civic (auto)',
       'datsun 310 gx', 'buick century limited',
       'oldsmobile cutlass ciera (diesel)', 'chrysler lebaron medallion',
       'ford granada l', 'toyota celica gt', 'dodge charger 2.2',
       'chevrolet camaro', 'ford mustang gl', 'vw pickup',
       'dodge rampage', 'ford ranger', 'chevy s-10'], dtype=object)
```

In [70]:
```
df['horsepower'].unique()
```

Out[70]:
```
array(['130', '165', '150', '140', '198', '220', '215', '225', '190',
       '170', '160', '95', '97', '85', '88', '46', '87', '90', '113',
       '200', '210', '193', '?', '100', '105', '175', '153', '180', '110',
       '72', '86', '70', '76', '65', '69', '60', '80', '54', '208', '155',
       '112', '92', '145', '137', '158', '167', '94', '107', '230', '49',
       '75', '91', '122', '67', '83', '78', '52', '61', '93', '148',
       '129', '96', '71', '98', '115', '53', '81', '79', '120', '152',
       '102', '108', '68', '58', '149', '89', '63', '48', '66', '139',
       '103', '125', '133', '138', '135', '142', '77', '62', '132', '84',
       '64', '74', '116', '82'], dtype=object)
```

In [71]:
```
# un soucis remarqué avec les lignes pour lesquelles la variable horsepower = ?, nous allons essayons d' enlever ses li
#1,5 % de la taille du dataset ce qui est négligeable
df[df['horsepower'] == '?']
```

Out[71]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
32	25.0	4	98.0	?	2046	19.0	71	1	ford pinto
126	21.0	6	200.0	?	2875	17.0	74	1	ford maverick
330	40.9	4	85.0	?	1835	17.3	80	2	renault lecar deluxe
336	23.6	4	140.0	?	2905	14.3	80	1	ford mustang cobra
354	34.5	4	100.0	?	2320	15.8	81	2	renault 18i
374	23.0	4	151.0	?	3035	20.5	82	1	amc concord dl

In [72]:
```
df1 =  df[df['horsepower'] != '?']
df1['horsepower'] = pd.to_numeric(df1['horsepower'], errors='coerce')

df1.shape
```

Out[72]:
```
(392, 9)
```

- Essayons d'avoir une matrice de corrélation afin de voir qu'elles sont les variables à choisir comme variable explicatives.

In [73]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64
 4   weight        392 non-null    int64
 5   acceleration  392 non-null    float64
 6   model year    392 non-null    int64
 7   origin        392 non-null    int64
 8   car name      392 non-null    object
dtypes: float64(3), int64(5), object(1)
memory usage: 30.6+ KB
```

In [74]:
```
df1.drop(columns=['car name']).corr()
```

Out[74]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
mpg	1.000000	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	0.580541	0.565209
cylinders	-0.777618	1.000000	0.950823	0.842983	0.897527	-0.504683	-0.345647	-0.568932
displacement	-0.805127	0.950823	1.000000	0.897257	0.932994	-0.543800	-0.369855	-0.614535
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361	-0.455171
weight	-0.832244	0.897527	0.932994	0.864538	1.000000	-0.416839	-0.309120	-0.585005
acceleration	0.423329	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	0.290316	0.212746
model year	0.580541	-0.345647	-0.369855	-0.416361	-0.309120	0.290316	1.000000	0.181528
origin	0.565209	-0.568932	-0.614535	-0.455171	-0.585005	0.212746	0.181528	1.000000

D'un point de vue logique les variables 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration' sont des variables liées au capacités techniques et mécaniques qui impacte directement sur la consommation du véhicule nous choisissons donc de les garder. Il nous reste à faire un choix entre les variables 'model year' et 'origin' . Dans ce cas nous choisissons la variable 'model year' car sa correlation avec la variable cible est la plus élévée que la variable 'origin'. Nous choisissons donc d'enlever les variables 'car name' et 'origin' du dataset pour avoir notre dataset de départ pour le modele.

In [75]:
```
df_clean = df1.drop(columns=['origin','car name'])
df_clean.head()
```

Out[75]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year
0	18.0	8	307.0	130	3504	12.0	70
1	15.0	8	350.0	165	3693	11.5	70
2	18.0	8	318.0	150	3436	11.0	70
3	16.0	8	304.0	150	3433	12.0	70
4	17.0	8	302.0	140	3449	10.5	70

- Créeons scindons le jeu donnéees pour l'analyse.

In [76]:
```
X = df_clean.drop(columns=['mpg'])
y = df_clean['mpg']
```

- Diviser les données en ensembles d'entraînement et de test (80% pour l'entraînement) :

In [77]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

1- Importez DecisionTreeRegressor depuis sklearn.tree :

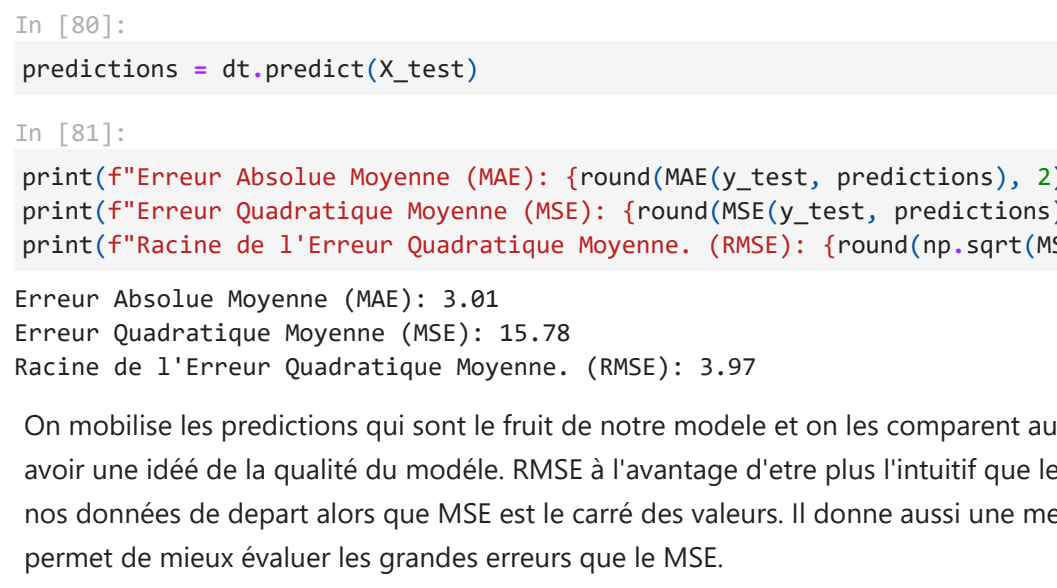
2- Instancions le modèle :

In [78]:
```
dt =  DecisionTreeRegressor(max_depth=8, min_samples_leaf=0.13, random_state=42)
#entrainer le modèle
dt.fit(X_train, y_train)
```

Out[78]:
```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=8, min_samples_leaf=0.13, random_state=42)
```

3. Affichez l'arbre de régression

In [79]:
```
plot_tree(dt, feature_names=X.columns.tolist(), rounded=True, filled=True)
plt.title("Arbre de régression ")
plt.show()
```



5. Utilisez les metriques nécéssaires :

In [80]:
```
predictions = dt.predict(X_test)
```

In [81]:
```
print(f"Erreur Absolue Moyenne (MAE): {round(MAE(y_test, predictions), 2)}")
print(f"Erreur Quadratique Moyenne (MSE): {round(MSE(y_test, predictions),2)}")
print(f"Racine de l'Erreur Quadratique Moyenne. (RMSE): {round(np.sqrt(MSE(y_test, predictions)),2)}")
```

```
Erreur Absolue Moyenne (MAE): 3.01
Erreur Quadratique Moyenne (MSE): 15.78
Racine de l'Erreur Quadratique Moyenne. (RMSE): 3.97
```

On mobilise les predictions qui sont le fruit de notre modele et on les comparent aux valeurs réelles des données d'entrainement pour avoir une idéé de la qualité du modéle. RMSE à l'avantage d'etre plus l'intuitif que le MSE car les valeurs sont rapportées à l'échelle de nos données de depart alors que MSE est le carré des valeurs. Il donne aussi une meilleure idée de la repartition des erreurs et enfin il permet de mieux évaluer les grandes erreurs que le MSE.

6. Evaluez maintenant l'erreur sur le jeu de données d'entraınement :

In [82]:
```
predict_train= dt.predict(X_train)
```

In [83]:
```
print(f"RMSE entrainement : {round(np.sqrt(MSE(y_train, predict_train)),3)}")
```

```
RMSE entrainement : 3.703
```

Le calcul du RMSE sur les données d'entrainement permets d'avoir une idée sur la qualité de l'entrainement du modéle en comparant apres entrainement les prédictions sur le jeu de données test et les valeurs réélles . Cela pourrait nous permettre de prevenir le surapprentissage possible si le resultat est relativement faible et se traduit par un modèle fortement corrélé au jeu de données d' entrainement mais tres peu performant avec de nouvelles données. A contrario, il peut aussi en etant tres elevé etre sujet à un sous apprentissage ce qui peut remettre en cause les paramétres du modele ou la qualité des données, ou le choix de l'algorithme utilisé. Logiquement une augmentation du jeu données permettra d'avoir un entrainement de meilleur qualité donc un modéle plus performant. En effet , plus le jeu données est complet meilleure est la prédiction.

7- Generer la courbe d'apprentissage :

Implementer une liste pour le differentes valeurs du pourcentage du jeu de données d'entrainement :

In [84]:
```
train = np.arange(0.1, 0.85, 0.05)
```

J'ai cru comprendre que nous sommes dans un cas ou nous avons un modèle avec une division du jeu de données en partie entrainement et partie test. Ensuite nous avons une deuxième scission de la partie test qui sera divisé en partie entrainement et partie validation.

In [85]:
```
# divisons le jeu de données entrainement en deux parties ;
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [86]:
```
# afin d'avoir les listes de nos deux resultats le rmse pour le jeu de données trai et l 'autre le jeu de données valid
rmse_train = []
rmse_val = []
#la boucle pour les differentes valeurs de la proportion de données d'entrainement :
for size in train:
    #division de la partie entrainement en partie train et validation
    X_train, _, y_train, _ = train_test_split(X_train_full, y_train_full, train_size=size, random_state=42)

    # Instanciation et entrainement
    model = DecisionTreeRegressor(max_depth=8, min_samples_leaf=0.13)
    model.fit(X_train, y_train)

    # Prédictions des train et test
    train_pred = model.predict(X_train)
    val_pred = model.predict(X_test)

    # Calculer les RMSE
    rmse_train.append(np.sqrt(MSE(y_train, train_pred)))
    rmse_val.append(np.sqrt(MSE(y_test, val_pred)))
```

In [87]:
```
#la visualisation
plt.figure(figsize=(10, 6))
plt.plot(train * 100, rmse_train, label="RMSE Entrainement", marker='o')
plt.plot(train * 100, rmse_val, label="RMSE Validation", marker='s')
plt.xlabel("Pourcentage des données utilisées pour l'entrainement")
plt.ylabel("RMSE")
plt.title("Courbes d'apprentissage pour le dataset Auto MPG")
plt.legend()
plt.show()
```

Exercice 2 : La validation croisée

1. La crossvalidation ou validation croisée est une methode de machine learning permettant d'améliorer la robustesse d'un modéle en limitant les risques de surapprentissage. Elle consiste a diviser notre jeu de données en sous parties (les folds) dont certains seront dédiés à l'apprentissage et les autres au test. Le folds seront entrainés de manière itérative et les autres seront utilisée pour tester l'apprentissage .

Les étapes sont les suivantes :

- Divisions du jeu de données en 10 folds(9 folds d'entrainement et 1 fold test) de taille egales
- Entrainement sur les folds d'entrainement et evaluation sur le fold test
- Pour chaque entrainement sur un fold d'entrainement on fait le test sur le fold test et on calcul l'erreur sur le bloc test (soit 10 score)
- Calculer le une erreur totale en faisant une moyenne de toutes les erreurs.

2. Elle a plusieurs avantages :
    - elle permets d'avoir une évaluation du modéle beaucoup plus robuste
    - Tres efficace pour les jeux données peu volumineux et optimise l'utilisation de ces dernières
    - Permets de mieux choisir l'hyperparametrage pertinent
    - elle minore le biais inhérant au split traditionnel

Le cout de cette méthode est lié au surcout computationnel (k calcul , k entrainements , k test) qui alourdit le travail avant d'obtenir les resultats. En dehors de ce travail , il y a aussi le stockage qui est aussi lourd.

Elles luttent contre le surentrainement, en mutlipliant les entrainements sur des blocs disctints, l'itération du calcul des erreurs permets de prévenir les risques d'overfitting et pour finir les subdivisions limitent les risques de biais liés à une division unique.

Le Boostrapping et la cross-validation sont soutenus par le meme paradigme une generation de sous parties des données initiales . La différence réside dans le fait que la cross-validation elle commence par une division franche des données àlors que le bootsrapping fonctionne par echantillonage avec remplacement (une données peut donc se retrouver sur plusieurs partitions ou plusieurs fois sur la meme) Le Boostrapping est donc plus robuste en terme de prise en compte de l'incertitude (ce qui peut avoir des effets positifs sur les statistiques). Le premier est plus utilisé pour trouver des intervalles de confiance alors que la seconde est plus usitée pour l'hyperparametrage et l'évaluation du modéle. Le bootstrapping vise à estimer la variabilité des paramètres ou des performances (distribution des statistiques). La validation croisée vise à évaluer la généralisation du modèle (performance sur des données non vues).

3. Le Bagging ou Boostrapp Aggregating est une methode d' ensemble de machine learning permettant d'obtenir des modeles plus robustes. La logique qu'elle sous tends peut etre divisée en deux phases. La première consiste en la creation d'échantillons de données d'entrainement par Bootstrapping(échantillonage avec remplacement) . La seconde est l'entrainement de chaque échantillon par un modele particulier en ayant pour but une minoration de la variance du modèle par agglomeration des predictions.

Le Random forest est une utilisation améliorée du Bagging, autrement dit c' est un Bagging avec des arbres de décision. La principale amélioration réside dans le fait qu'ici le modele ajoute au bagging une selection aléatoire de variables à chaque noeuds des arbres de la foret.

Les variables à chaque split sont choisies selon deux criteres :

- choix aléatoires de m variables sur le total des variables
- sur les m variables choisies le choix optimal est fait en fonction de l'indice de GINI ou de l'impureté.

L' interet principal est d'éviter que les resultats soit biaisés par une correlation trop forte entre les differents modéles entraines (meme parametres des variables). La prise en compte de l'incertitude liées aux choix aléatoire des variables des modéles d'entrainement rend le modèle plus robuste et permettra une meilleure prédiction en production avec des données inconnues.

4 .Creeons la foret aléatoire :

In [88]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)#split
model_forest = RandomForestRegressor(max_depth=4,min_samples_leaf=0.26,random_state=42)#implementation modele
model_forest.fit(X_train, y_train)#entrainement
```

Out[88]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=4, min_samples_leaf=0.26, random_state=42)
```

In [89]:
```
y_pred_forest = model_forest.predict(X_test)#predictions
print(f"Erreur quadratique moyenne (MSE) : {MSE(y_test, y_pred_forest):.3f}")
```

```
Erreur quadratique moyenne (MSE) : 21.732
```

In [90]:
```
#scoring de tous les folds
scores = cross_val_score(model_forest, X_train, y_train, cv=10, scoring="neg_mean_squared_error")
scores
```

Out[90]:
```
array([-23.60870469, -22.17093883, -28.88682832, -29.64083123,
       -32.51761647, -32.15205478, -15.07867489, -14.36735303,
       -20.4412804 , -26.90177113])
```

In [91]:
```
#calcul Rmse
print('RMSE moyen avec 10 folds: ' , round(np.sqrt(-scores).mean(),3))
```

```
RMSE moyen avec 10 folds:  4.914
```

Projet 4 : Prédire une maladie du foie

Exercice 1 : Bagging contre forets aléatoires

1.Le bootstrapping est une méthode d'échantillonnage qui consiste à diviser les données en sous-groupes formés par un tirage aléatoire avec remplacement. Ces sous-échantillons sont ensuite utilisés pour des études statistiques ou, dans le cadre du machine learning, pour créer des modèles plus robustes.

Le bagging (Bootstrap Aggregating) utilise le bootstrapping pour générer plusieurs sous-ensembles à partir des données d'entraînement. Ensuite, on entraîne des modèles sur ces sous-échantillons, et leurs résultats sont agrégés. Cette agrégation permet de réduire la variance du modèle, ce qui améliore sa robustesse.

En résumé, le bootstrapping est une méthode d'échantillonnage, tandis que le bagging consiste à générer plusieurs modèles à partir des sous-échantillons créés, puis à agréger leurs résultats pour minimiser la variance et renforcer la stabilité du modèle.

- Importons le jeu de données et faisons une AED :

In [92]:
```
foie = pd.read_csv('indian_liver_patient.csv')
foie.head()
```

Out[92]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Tc
0	65	Female	0.7	0.1	187	16	18	
1	62	Male	10.9	5.5	699	64	100	
2	62	Male	7.3	4.1	490	60	68	
3	58	Male	1.0	0.4	182	14	20	
4	72	Male	3.9	2.0	195	27	59	

In [93]:
```
foie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
 5   Alamine_Aminotransferase    583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

In [94]:
```
foie.isnull().sum()
```

Out[94]:
```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphotase          0
Alamine_Aminotransferase      0
Aspartate_Aminotransferase    0
Total_Protiens                0
Albumin                       0
Albumin_and_Globulin_Ratio    4
Dataset                       0
dtype: int64
```

In [95]:
```
dfSummary(foie)
```

Out[95]:

Data Frame Summary
foie
Dimensions: 583 x 11
Duplicates: 13

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**Age** [int64]	Mean (sd) : 44.7 (16.2) min < med < max: 4.0 < 45.0 < 90.0 IQR (CV) : 25.0 (2.8)	72 distinct values		0 (0.0%)
2	**Gender** [object]	1. Male 2. Female	441 (75.6%) 142 (24.4%)		0 (0.0%)
3	**Total_Bilirubin** [float64]	Mean (sd) : 3.3 (6.2) min < med < max: 0.4 < 1.0 < 75.0 IQR (CV) : 1.8 (0.5)	113 distinct values		0 (0.0%)
4	**Direct_Bilirubin** [float64]	Mean (sd) : 1.5 (2.8) min < med < max: 0.1 < 0.3 < 19.7 IQR (CV) : 1.1 (0.5)	80 distinct values		0 (0.0%)
5	**Alkaline_Phosphotase** [int64]	Mean (sd) : 290.6 (242.9) min < med < max: 63.0 < 208.0 < 2110.0 IQR (CV) : 122.5 (1.2)	263 distinct values		0 (0.0%)
6	**Alamine_Aminotransferase** [int64]	Mean (sd) : 80.7 (182.6) min < med < max: 10.0 < 35.0 < 2000.0 IQR (CV) : 37.5 (0.4)	152 distinct values		0 (0.0%)
7	**Aspartate_Aminotransferase** [int64]	Mean (sd) : 109.9 (288.9) min < med < max: 10.0 < 42.0 < 4929.0 IQR (CV) : 62.0 (0.4)	177 distinct values		0 (0.0%)
8	**Total_Protiens** [float64]	Mean (sd) : 6.5 (1.1) min < med < max: 2.7 < 6.6 < 9.6 IQR (CV) : 1.4 (6.0)	58 distinct values		0 (0.0%)
9	**Albumin** [float64]	Mean (sd) : 3.1 (0.8) min < med < max: 0.9 < 3.1 < 5.5 IQR (CV) : 1.2 (3.9)	40 distinct values		0 (0.0%)
10	**Albumin_and_Globulin_Ratio** [float64]	Mean (sd) : 0.9 (0.3) min < med < max: 0.3 < 0.9 < 2.8 IQR (CV) : 0.4 (3.0)	69 distinct values		4 (0.7%)
11	**Dataset** [int64]	Mean (sd) : 1.3 (0.5) min < med < max: 1.0 < 1.0 < 2.0 IQR (CV) : 1.0 (2.8)	2 distinct values		0 (0.0%)

In [96]:
```
foie_clean = foie.dropna()
foie_clean.shape
```

Out[96]:
```
(579, 11)
```

In [97]:
```
la= LabelEncoder()#encodage de la variable genre
foie_clean["Gender"]=la.fit_transform(foie_clean["Gender"])
foie_clean.head()
```

Out[97]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Tc
0	65	0	0.7	0.1	187	16	18	
1	62	1	10.9	5.5	699	64	100	
2	62	1	7.3	4.1	490	60	68	
3	58	1	1.0	0.4	182	14	20	
4	72	1	3.9	2.0	195	27	59	

In [98]:
```
foie_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 579 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         579 non-null    int64
 1   Gender                      579 non-null    int32
 2   Total_Bilirubin             579 non-null    float64
 3   Direct_Bilirubin            579 non-null    float64
 4   Alkaline_Phosphotase        579 non-null    int64
 5   Alamine_Aminotransferase    579 non-null    int64
 6   Aspartate_Aminotransferase  579 non-null    int64
 7   Total_Protiens              579 non-null    float64
 8   Albumin                     579 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     579 non-null    int64
dtypes: float64(5), int32(1), int64(5)
memory usage: 52.0 KB
```

- Indexation de la target des features

In [99]:
```
X = foie_clean.drop('Dataset', axis=1)
y = foie_clean['Dataset']
```

2. Préparer le jeu de données d'entrainement avec un split 70/30

In [100]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3. Utilisation du bagging avec un arbre de décision et trouver les metriques

In [101]:
```
tree_model = DecisionTreeClassifier(random_state=42)# arbre de décision
bagging_model = BaggingClassifier(base_estimator= tree_model, random_state=42) #modele bagging
bagging_model.fit(X_train, y_train)# entrainons le bagging
```

Out[101]:
```
BaggingClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

- Essayons de nous occuper des metriques :

In [102]:
```
predictions =  bagging_model.predict(X_test)# predictions du modele sur les données test
y_prob = bagging_model.predict_proba(X_test)[:, 1] #base de calcul pour AUC pour des probabilité positives

print('Accuracy: ',round(accuracy_score(y_test, predictions),3))
print('Recall: ', round(recall_score(y_test, predictions),3))

print('AUC: ', round(roc_auc_score(y_test, y_prob),3))
```

```
Accuracy:  0.678
Recall:  0.947
AUC:  0.731
```

4. Nous allons essayer de faire la meme chose mais cette fois avec une foret aléatoire

In [103]:
```
# le modele de foret aléatoire
foret = RandomForestClassifier(n_estimators=100, random_state=42)
foret.fit(X_train, y_train)#entrainement
```