République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الاعلام الالي سابقا)
École nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

TP BDM

Module :

Big Data Mining

Groupe :

2CS SIL1

Membres de l'équipe :

BENKHELIFA Bouchra
YOUSFI Sarah

Année universitaire 2024-2025

Contents

Chapter 1

Introduction

Dans ce rapport, nous étudions l'utilisation des arbres de décision, une méthode d'apprentissage supervisé largement utilisée à la fois pour la classification et la régression. Les arbres de décision sont des modèles intuitifs qui divisent les données en sous-groupes en fonction des valeurs des variables explicatives, selon une structure arborescente. En classification, ils permettent de prédire des variables qualitatives, tandis qu'en régression, ils estiment des valeurs numériques continues. Cette double application nous permet d'observer la capacité des arbres à s'adapter à différents types de problèmes. Nous analysons les performances du modèle, l'impact de l'élagage, et optimisons les hyperparamètres pour de meilleurs résultats.

Chapter 2

Arbre de decision dans la classification

2.1 Dataset:

Le jeu de données utilisé dans ce travail pratique est PimaIndiansDiabetes, issu du package mlbench en R. Il contient **768 observations** décrites par **8 variables** prédictives numériques, telles que le nombre de grossesses, le taux de glucose, la pression artérielle ou encore l'indice de masse corporelle. La variable cible, nommée **diabetes**, est binaire et indique si une patiente est atteinte ou non de diabète (pos ou neg).

2.2 Trainning:

On commence par charger les bibliothèques nécessaires : rpart pour la création d'arbres de décision, rpart.plot pour leur visualisation, et mlbench pour accéder à des jeux de données intégrés. Le jeu de données utilisé est PimaIndiansDiabetes, qui est stocké dans l'objet data.Un arbre de décision est construit à l'aide de la fonction rpart(), en prenant diabetes comme variable cible et toutes les autres variables comme prédicteurs. L'arbre est visualisé à l'aide des fonctions plot() et text() pour représenter la structure de décision.

```
library(rpart)
library(rpart.plot)
library(mlbench)
data<-PimaIndiansDiabetes
#nature des données
lapply(data,class)
#arbre de decision
Tree <- rpart(diabetes~.,data)
Tree
plot(Tree)
text(Tree)
plotcp(Tree)
```

Figure 2.1: Code R avant élagage

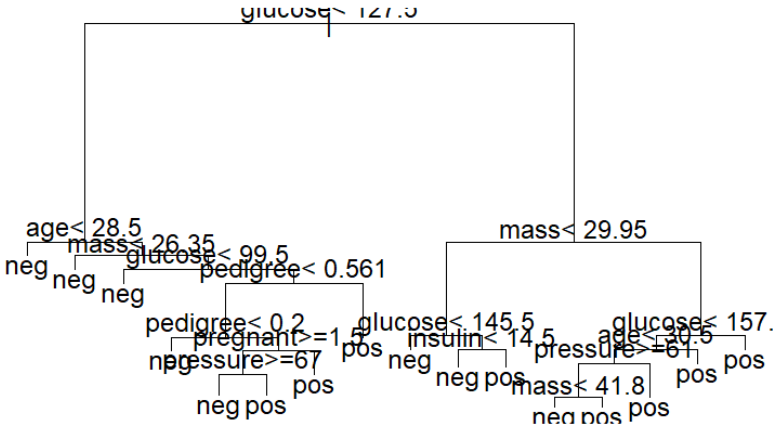


Figure 2.2: Arbre de decision avant élagage

2.3 Evaluation avant élagage

2.3.1 Table de confusion:

La table de confusion ci-dessous permet d'évaluer la performance du modèle en comparant les prédictions aux observations réelles, mettant en évidence les vrais positifs (VP), faux positifs (FP), vrais négatifs (VN) et faux négatifs (FN) pour chaque classe.

```
> Confusion
     pred
      neg pos
  neg 449  51
  pos  72 196
```

Figure 2.3: Table de confusion avant élagage

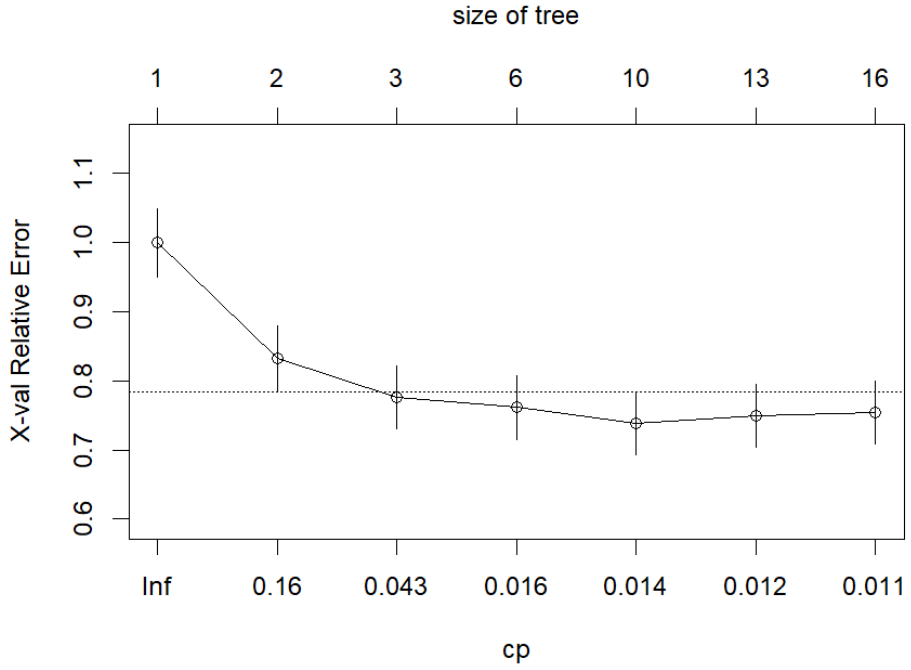
2.3.2 Evaluation:

Afin d'évaluer les performances de notre modèle de classification, nous avons recours à plusieurs métriques couramment utilisées en apprentissage supervisé, notamment la précision, le rappel et accuracy.

- Accuracy : 0.8398438
- Précision : 0.7935223
- Recall : 0.7313433

2.4 Arbre

Pour obtenir un arbre de décision optimal, nous cherchons la valeur du paramètre de complexité cp qui minimise l'erreur de validation croisée, visualisée avec la fonction plotcp().



Cette valeur permet ensuite de générer un arbre élagué, plus simple et plus performant, à l'aide de la fonction prune().

```r
#obtention de l'rbre simplifié
TreeSimple <- prune(Tree,cp=0.014)
#arbre optimal
prp(TreeSimple,extra=1)
#prevision
predict(TreeSimple)
#par classe
pred=predict(TreeSimple,data, type="class")
#Performance: table de confusion
Confusion=table(data$diabetes, pred)
Confusion
accuracy <- sum(diag(Confusion)) / sum(Confusion)
TP <- Confusion["pos", "pos"]
FP <- Confusion["neg", "pos"]
FN <- Confusion["pos", "neg"]
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
print(accuracy)
print(precision)
print(recall)
```

Figure 2.4: Code R après élagage

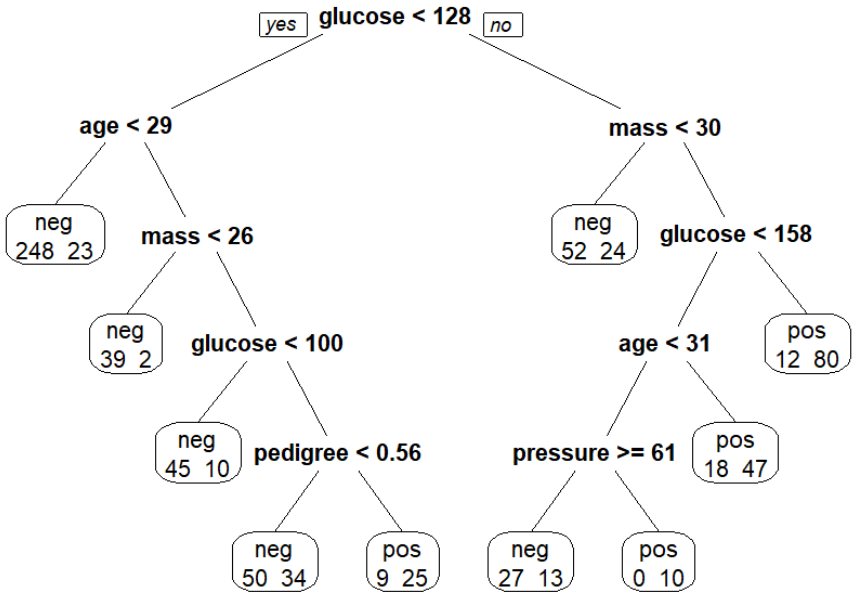


Figure 2.5: Arbre de decision après élagage

2.5 Evaluation après élagage:

2.5.1 Table de confusion

```
> Confusion
     pred
      neg pos
neg 461  39
pos 106 162
```

Figure 2.6: Table de confusion après élagage

2.5.2 Evaluation:

Les métriques d'évaluation (accuracy, précision et rappel) sont recalculées afin de mesurer les performances du modèle élagué.

- Accuracy : 0.8111979
- Précision : 0.8059701
- Recall : 0.6044776

2.6 Tuning

Afin d'améliorer les performances du modèle et de limiter le surapprentissage, nous procédons à une étape de tuning consistant à ajuster les hyperparamètres de l'arbre de décision. L'objectif est de trouver la meilleure combinaison de paramètres (profondeur

maximale de l'arbre (maxdepth) ou le paramètre de complexité cp) qui permet d'obtenir un modèle précis.

```r
tuned_model <- tune.rpart(
  diabetes~.,data,
  maxdepth = c(2, 3, 4, 5)
)

# View the best model
summary(tuned_model)

# Access the best tree
best_tree <- tuned_model$best.model
predict(best_tree,data)
#par classe
pred=predict(best_tree,data, type="class")
#Performance: table de confusion
Confusion=table(data$diabetes, pred)
Confusion
accuracy <- sum(diag(Confusion)) / sum(Confusion)
TP <- Confusion["pos", "pos"]
FP <- Confusion["neg", "pos"]
FN <- Confusion["pos", "neg"]
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
print(accuracy)
print(precision)
print(recall)
```

Figure 2.7: Code R: Utilisation du Tuning

Le meilleur modèle a été obtenu avec une profondeur maximale de 5, correspondant à la meilleure performance (taux d'erreur moyen le plus bas) de 0.243.

2.6.1 Table de confusion:

```
> Confusion
     pred
      neg pos
  neg 457  43
  pos  96 172
```

Figure 2.8: Table de confusion Tuning

2.6.2 Evaluation:

- Accuracy : 0.8190104
- Précision : 0.8
- Recall : 0.641791

Chapter 3

Arbre de decision dans la régression

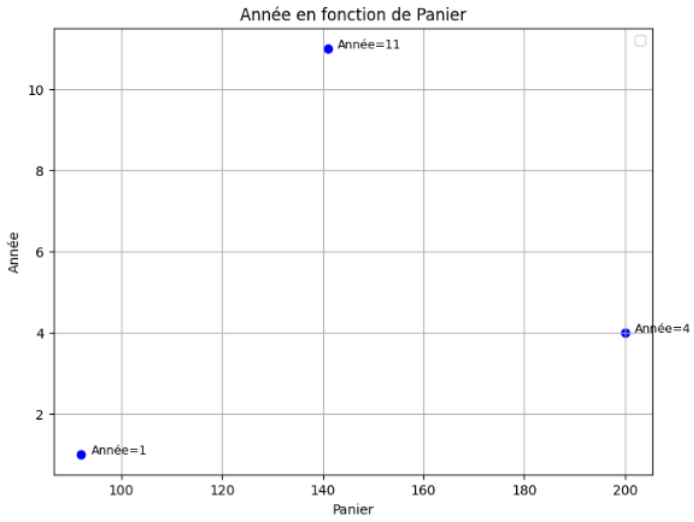
3.1 Description

Ce dataset contient des données sur les paniers annuels et les salaires correspondants de 3 joueurs. Les valeurs de l'année et du panier sont stockées dans une liste X, tandis que les salaires associés à chaque entrée sont dans la liste y. Chaque ligne de X représente une année spécifique et le montant total du panier pour cette année, et chaque valeur dans y représente le salaire associé

	Année	Panier	Yi=Salaire
1	1	92	100
3	4	200	250
2	11	141	500

Pour consulter l'implémentation complète de l'arbre de décision, vous pouvez accéder au notebook via le lien suivant
`https://colab.research.google.com/drive/136rfHDCMlTU9hTe7B84PZSwEG6Y22S33#scrollTo=ZFWhlRdUN6r4`

3.2 Repésentation graphique Années en fonction de Panier



3.3 On commence par l'attribut Panier

3.3.1 Les étpaes

Fonction pour faire la subdivision en 2 régions

```
def diviser_donnees(X, y, seuil, est_annee=False):
    if est_annee:
        region1 = [(x, y_val) for (x, y_val) in zip(X, y) if x[0] <= seuil]
        region2 = [(x, y_val) for (x, y_val) in zip(X, y) if x[0] > seuil]
    else:
        region1 = [(x, y_val) for (x, y_val) in zip(X, y) if x[1] <= seuil]
        region2 = [(x, y_val) for (x, y_val) in zip(X, y) if x[1] > seuil]

    return region1, region2
```

Fonction pour calculer les moyennes de salaires de chaque région et l'erreur

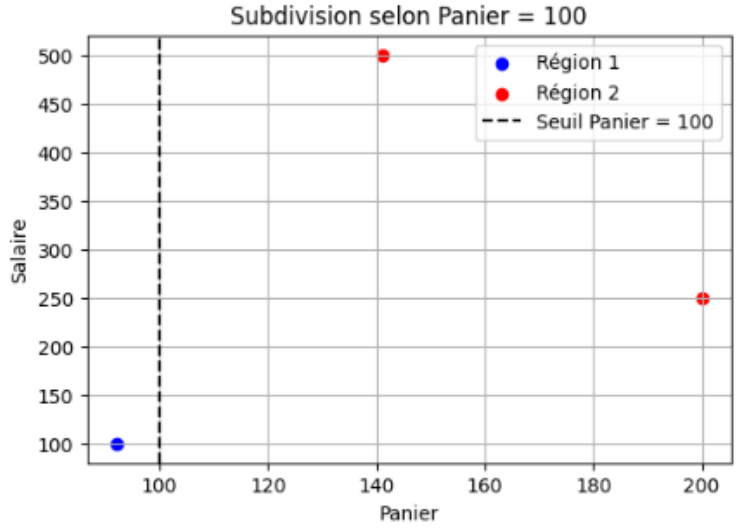
```
def calculer_erreur(region1, region2):
    moy1 = np.mean([y_val for _, y_val in region1])
    moy2 = np.mean([y_val for _, y_val in region2])

    erreur_1 = sum((y_val - moy1)**2 for _, y_val in region1)
    erreur_2 = sum((y_val - moy2)**2 for _, y_val in region2)

    erreur_totale = erreur_1 + erreur_2
    return erreur_totale
```

Fonction pour la subdivision de données

```
region1, region2 = diviser_donnees(X, y, seuil=150, est_annee=False)
erreur, moyenne1, moyenne2 = calculer_erreur(region1, region2)
print("Moyenne des salaires dans Région 1 :", moyenne1)
print("Moyenne des salaires dans Région 2 :", moyenne2)
print("Erreur quadratique totale :", erreur)
plt.figure(figsize=(6, 4))
for (x, y_val) in region1:
    plt.scatter(x[1], y_val, color='blue', label='Région 1' if 'Région 1' not in plt.gca().get_legend_handles_labels()[1] else "")
for (x, y_val) in region2:
    plt.scatter(x[1], y_val, color='red', label='Région 2' if 'Région 2' not in plt.gca().get_legend_handles_labels()[1] else "")
plt.axvline(x=150, color='black', linestyle='--', label='Seuil Panier = 150')
plt.xlabel('Panier')
plt.ylabel('Salaire')
plt.title('Subdivision selon Panier = 150')
plt.legend()
plt.grid(True)
plt.show()
```

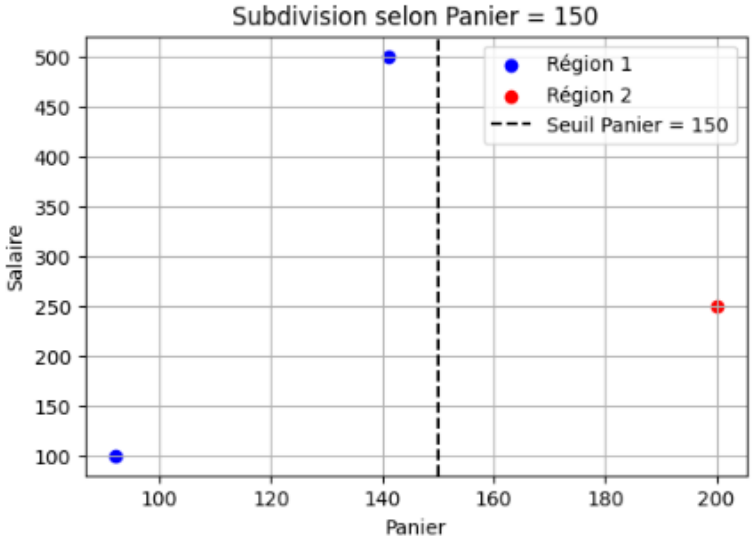
3.3.2 Valeur=1

```
Moyenne des salaires dans Région 1 : 100.0
Moyenne des salaires dans Région 2 : 375.0
Erreur quadratique totale : 31250.0
```



3.3.3 Valeur=1.5

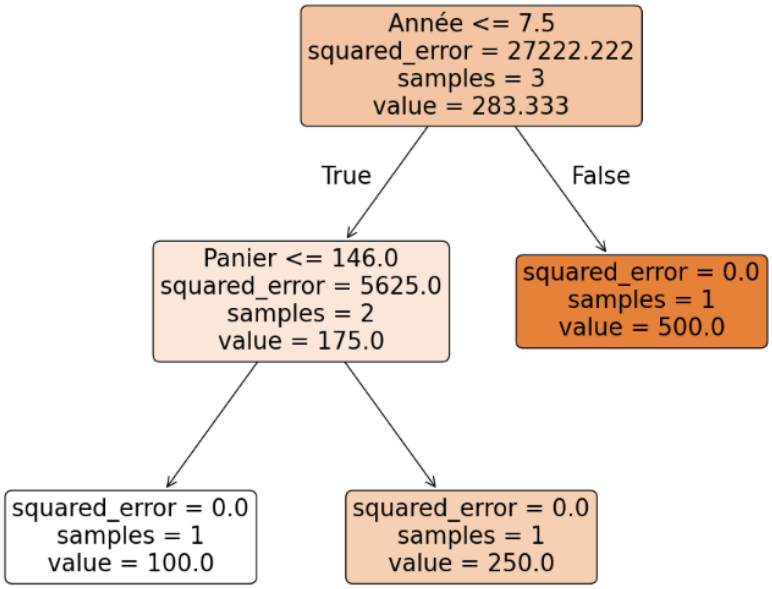
```
Moyenne des salaires dans Région 1 : 300.0
Moyenne des salaires dans Région 2 : 250.0
Erreur quadratique totale : 80000.0
```



3.3.4 Meilleure subdivision

La meilleure subdivision est celle qui minimise l'erreur quadratique totale donc la première avec Erreur minimale : 31250.0

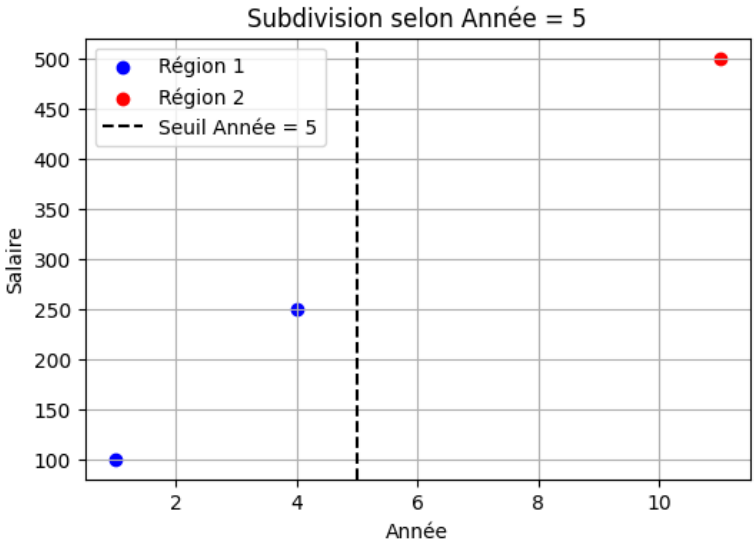
3.3.5 Arbre de decision



3.4 On refait les mêmes étapes avec l'attribut Année

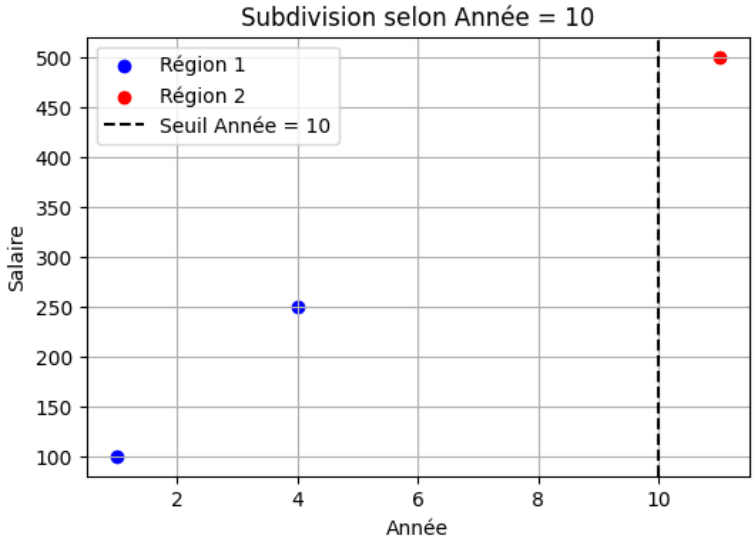
3.4.1 Valeur=0.5

```
Moyenne des salaires dans Région 1 : 175.0
Moyenne des salaires dans Région 2 : 500.0
Erreur quadratique totale : 11250.0
```



3.4.2 Valeur=1

```
Moyenne des salaires dans Région 1 : 175.0
Moyenne des salaires dans Région 2 : 500.0
Erreur quadratique totale : 11250.0
```

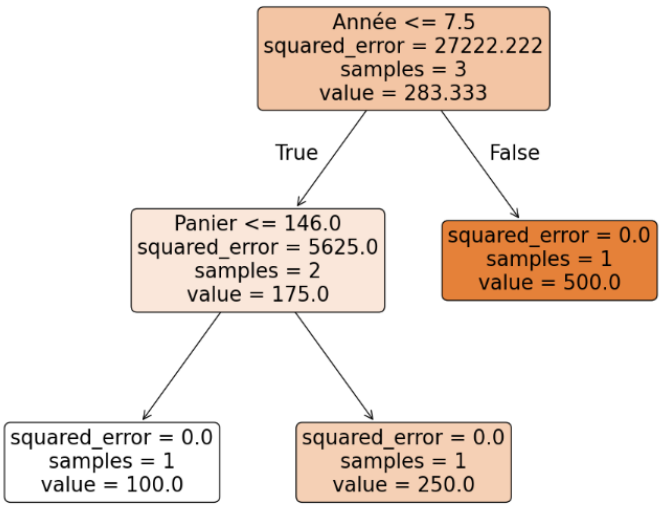


3.4.3 Meilleure subdivision

Les deux subdivisions donnent la même erreur quadratique totale, donc on doit utiliser d'autres critères pour choisir la "meilleure" subdivision :Équilibre des régions , Priorité sur certaines variables dans le contexte metier ... Dans notre cas, bien que les erreurs quadratiques soient identiques pour les deux subdivisions, l'attribut Panier présente une erreur minimale dès la première subdivision. Cela indique que cette première subdivision est plus pertinente pour le modèle global, car elle permet une meilleure séparation initiale des données.

3.4.4 Arbre de decision

La racine de l'arbre est choisie automatiquement par l'algorithme de l'arbre de décision (DecisionTreeRegressor de scikit-learn), en fonction de la meilleure variable de découpage, donc on a obtenu le même arbre



3.4.5 Salaire en moyenne obtiendra un joueur ayant 2 années d'expérience et 13 paniers

Pour estimer le salaire de ce joueur , on suit la branche la plus à gauche de l'arbre, ce qui signifie que son profil correspond à cette feuille salaire=100.

```
joueur = [2, 13]
seuil_panier = meilleur_seuil

if joueur[1] <= seuil_panier:
    region = [(x, val) for (x, val) in zip(X, y) if x[1] <= seuil_panier]
else:
    region = [(x, val) for (x, val) in zip(X, y) if x[1] > seuil_panier]

salaire_moyen = np.mean([val for _, val in region])
print(f"Salaire estimé pour 2 années d'expérience et 13 paniers : {salaire_moyen:.2f} ")
```

```
Salaire estimé pour 2 années d'expérience et 13 paniers : 100.00
```

Chapter 4

Conclusion

En résumé, l'arbre de décision est un modèle puissant et polyvalent, offrant une bonne balance entre précision et explicabilité. Bien qu'il puisse ne pas toujours être le modèle le plus performant seul, son utilité dans des ensembles d'algorithmes et sa capacité à traiter une grande variété de problèmes en font un outil essentiel dans le domaine de l'apprentissage automatique.