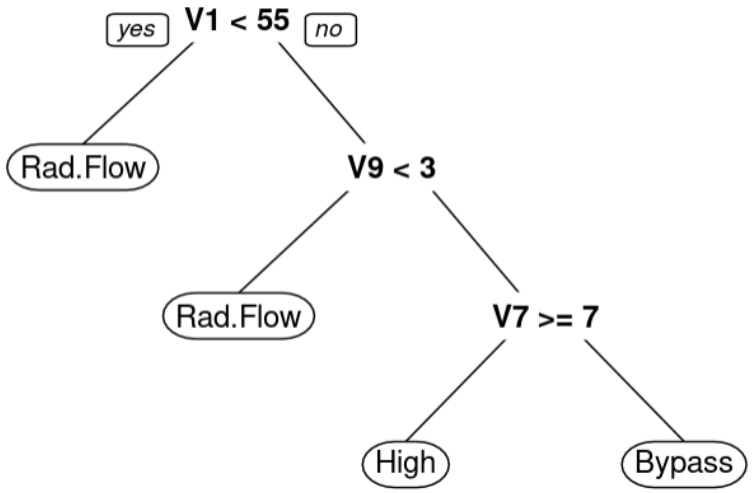
Série apprentissage supervisé
Mme Hamdad Leila
BDM. 2CSSIT; 2CSSIL G1
Amriou/Chikouche

Arbre de décision: Script avec Shuttle ensuite Glass.

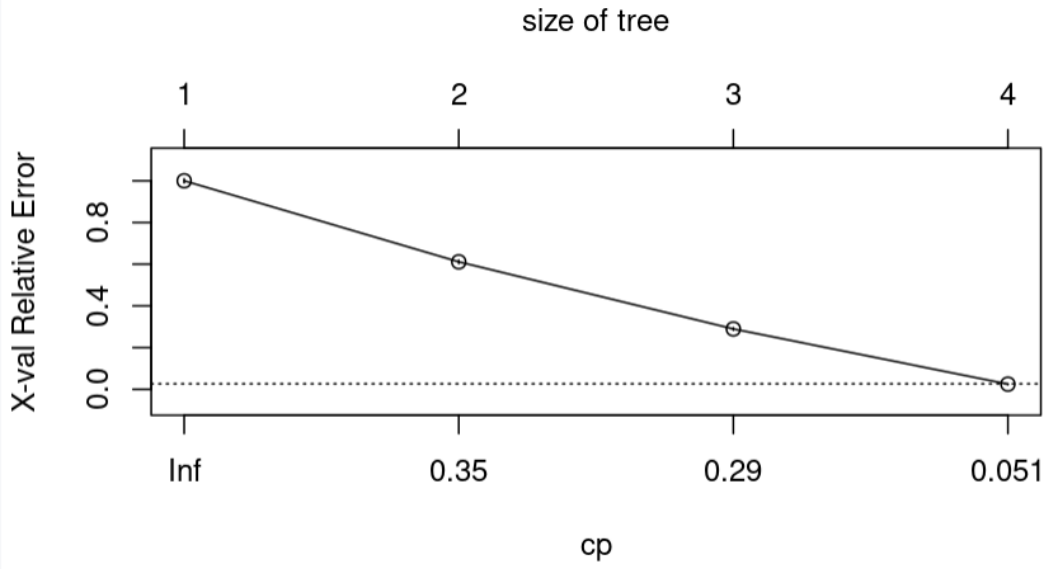
Remarque: Vous trouverez le code dans les fichiers .R

Résultats Shuttle:

- Après avoir partitionner les données en Training-set et Test-set, on obtient un arbre de décision simple et facilement interprétable de 4 niveaux comme suit:



- On remarque que dans un cas pareil l'élagage (simplification de l'arbre) ne servira pas à grand chose car en prenant un `cp = 0.051` minimisant l'erreur de la validation croisée, la taille de l'arbre restera constante i.e 4 niveaux.



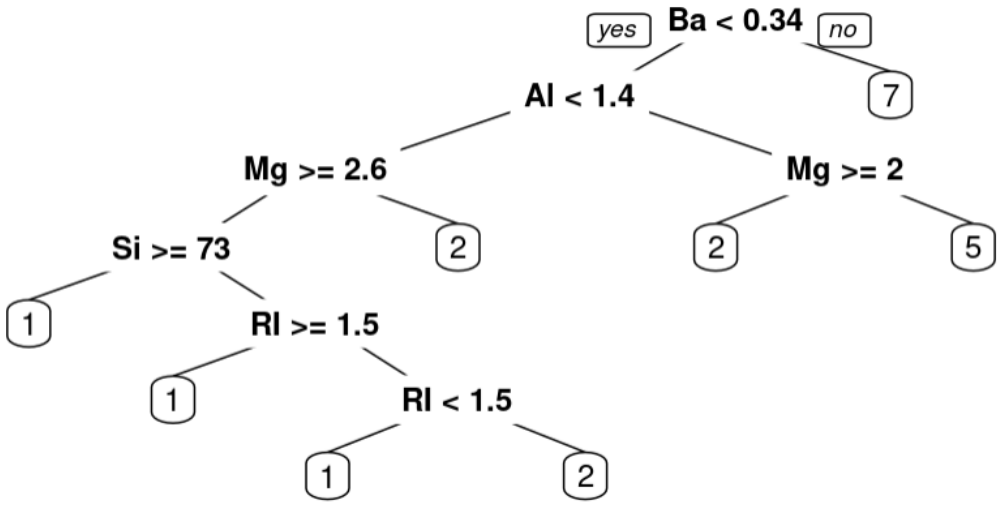
- On compare le temps d'exécution ainsi que les métriques de performances avant et après élagage:

	Avant élagage	Après élagage
Temps d'exécution	0.2731352	0.2639883
Accuracy	0.9944828	0.9944828
Rappel	0.4271551	0.4271551
Précision	0.4241339	0.4241339
Fscore	0.1827631	0.1827631

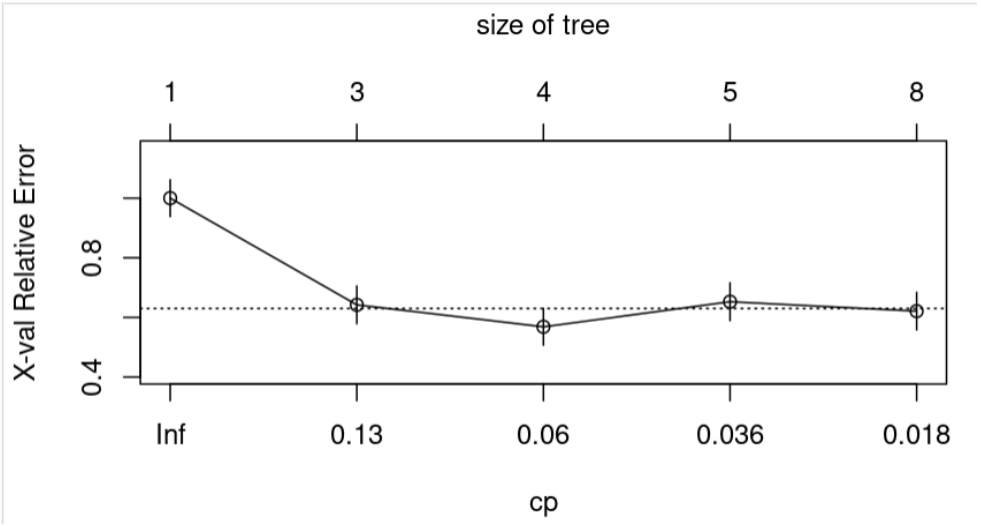
- On en déduit que les résultats sont plus ou moins identiques car l'élagage n'a pas apporté de changement.
- D'après le tableau, on remarque que l'accuracy est très élevée comparée aux autres métriques de performance. Ainsi, la dataset Shuttle est déséquilibrée. Donc, on peut faire face à un problème d'overfitting.

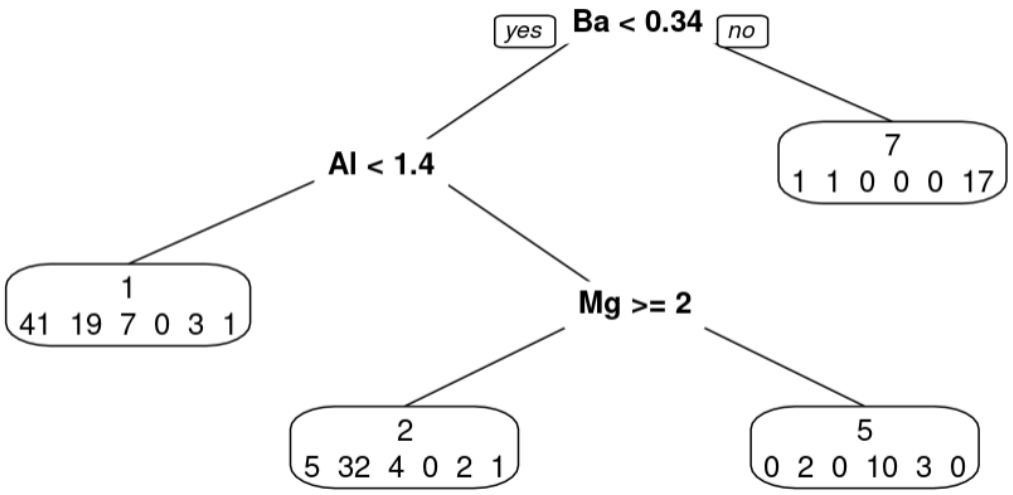
Résultats Glass:

- Après avoir partitionner les données en Training-set et Test-set, on obtient un arbre de décision de 7 niveaux qui peut malgré ça être également interprété comme suit:



- En prenant un `cp = 0.06` minimisant l'erreur de la validation croisée, la taille de l'arbre va se réduire à seulement 4 niveaux. Ici, on peut voir l'effet de l'élagage contrairement a Shuttle





- On compare le temps d'exécution ainsi que les métriques de performances avant et après élagage:

	Avant élagage	**Après élagage**
Temps d'exécution	0.02485895	0.01317787
Accuracy	0.7384615	0.7384615
Rappel	0.5348265	0.5348265
Précision	0.555754	0.555754
Fscore	0.3163852	0.3163852

- En termes de perte d’informations, notre dataset est restée protéger car on remarque que les métriques de performances sont stables. Mais en termes de temps d'exécution, on voit que le modèle s'entraîne plus rapidement sur l’arbre simplifié.

- Glass est plus équilibrée que Shuttle et cette dernière montre de façon plus nette l'effet de l'élagage ainsi que son intérêt sur les données.

Random forest: Script avec Glass.

- En executant `RF=randomForest(Type~., data=Glass)`, on obtient le résultat suivant:

```
Call:
 randomForest(formula = Type ~ ., data = Glass)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3
        OOB estimate of  error rate: 21.03%
Confusion matrix:
   1  2 3  5 6  7 class.error
1 61  7 2  0 0  0   0.1285714
2 10 59 1  3 2  1   0.2236842
3  6  4 7  0 0  0   0.5882353
5  0  2 0 10 0  1   0.2307692
6  0  2 0  0 7  0   0.2222222
7  1  3 0  0 0 25   0.1379310
```

- RF contiendra:
  - le type d'opération effectuée sur la dataset (classification/régression), ici: classification.
  - Le nombre d'arbres générés : 500
  - le nombre de variables testées à chaque division à la construction (split): 3
  - Estimation de l'erreur OOB : 21.03%
  - La matrice de confusion ainsi que l'erreur engendrée par chaque classe.

Random forest avec greed search: Script avec Glass.

- En effectuant un grid search avec cross validation, on cherche le nombre optimal du nombre de variables à prendre en compte dans chaque division. On trouve que c'est 2. Voici les résultats obtenus:

```
Call:
 randomForest(formula = Type ~ ., data = Glass, mtry =
RF_model$bestTune$mtry)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 19.16%
Confusion matrix:
   1  2 3 5 6  7 class.error
1 62  6 2 0 0  0   0.1142857
2  9 64 1 1 1  0   0.1578947
3  7  4 6 0 0  0   0.6470588
5  0  3 0 9 0  1   0.3076923
6  0  2 0 0 7  0   0.2222222
7  1  3 0 0 0 25   0.1379310
```

- On remarque que l’erreur a diminué car cela est peut être dû au fait que le mtry optimal a permis d'améliorer la balance entre le surapprentissage et le sous-ajustement.

Recapitulatif: Random forest VS Decision tree.

	Comparaison
Temps d'exécution	Les Random Forests prennent généralement plus de temps pour s'entraîner et prédire que les arbres de décision simples. Car Random Forest est un ensemble d’arbres de décision.
Précision	Dans la majorité des cas, Random Forest produit des résultats plus précis que les arbres de décision simples. Mais plusieurs facteurs influent sur les performances, notamment le calibre des données, la complexité du modèle et la taille de l'échantillon.
Apprentissage et test	Random Forests ne nécessite pas la division de données en training-set et test-set contrairement aux Decision Trees.En effet, les modèles Random Trees sont basés sur la combinaison de plusieurs arbres de décision, chacun étant construit sur un ensemble unique de données et de facteurs. Les Random Forests ont tendance à mieux généraliser que les arbres de décision simples, ce qui réduit le risque de surapprentissage.
variables d'entrée	Les arbres de décision sont mieux adaptés aux ensembles de données avec un petit nombre de variables d'entrée. Les Random Forest sont plus adaptés aux ensembles de données avec un grand nombre de variables d'entrée. Ceci rend les arbres de décision plus faciles à interpréter que les Random Forests.