EXPERIMENT NO:6

AIM:

Write R program to Implement decision trees using mushrooms dataset.

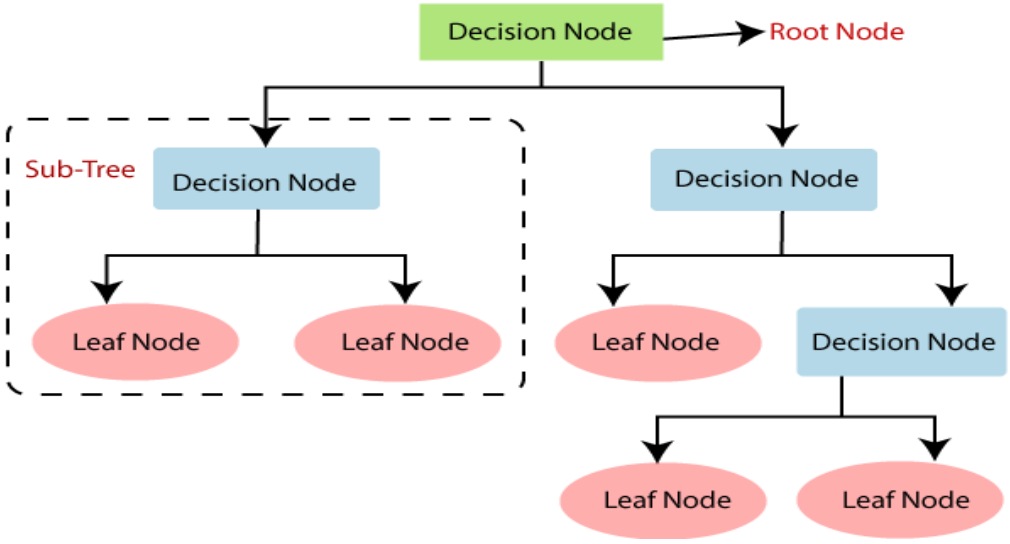
DESCRIPTION:

Decision Tree Classification Algorithm:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.

- Below diagram explains the general structure of a decision tree.



Decision Tree Terminologies:

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

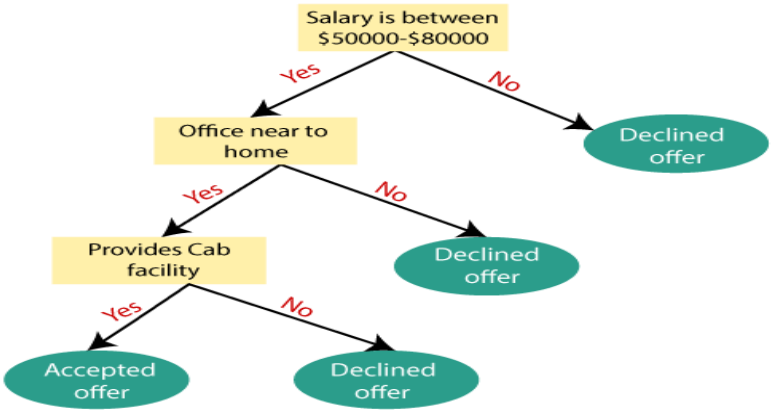
How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures:

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.

1. Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$Entropy(s) = -P(yes)\log_2 P(yes) - P(no)\log_2 P(no)$$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- Gini index can be calculated using the below formula:

$$Gini\ Index = 1 - \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree:

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Steps involved in Implementation of Decision Tree:

For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as KNN SVM, LogisticRegression, etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

CODE:

```
library(rpart)
library(rpart.plot)
library(partykit)
library(caret)
library(pROC)
library(ROCR)
```

OUTPUT:

```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> library(rpart)
> library(rpart.plot)
Warning message:
package 'rpart.plot' was built under R version 4.2.3
> library(partykit)
Loading required package: grid
Loading required package: libcoin
Loading required package: mvtnorm
Warning messages:
1: package 'partykit' was built under R version 4.2.3
2: package 'libcoin' was built under R version 4.2.3
3: package 'mvtnorm' was built under R version 4.2.3
> library(caret)
Loading required package: ggplot2
Loading required package: lattice
Warning messages:
1: package 'caret' was built under R version 4.2.3
2: package 'ggplot2' was built under R version 4.2.3
> library(pROC)
Type 'citation("pROC")' for a citation.

Attaching package: 'pROC'

The following objects are masked from 'package:stats':

    cov, smooth, var

Warning message:
package 'pROC' was built under R version 4.2.3
> library(ROCR)
Warning message:
package 'ROCR' was built under R version 4.2.3
>
```

CODE:

#Load the mushrooms data set

```
dataset<-read.csv("D:\VVIT.BTECH\3_1\ML_lab\exp-6\mushrooms.csv")
#Convert class labels to factors
dataset$class<-as.factor(dataset$class)
set.seed(123)
sample_indices<-sample(nrow(dataset),0.8*nrow(dataset))
train_data<-dataset[sample_indices,]
test_data<-dataset[-sample_indices,]
```

OUTPUT:

```
> dataset<-read.csv("C:\\Users\\DELL\\Downloads\\mushrooms.csv")
> #Convert class labels to factors
> dataset$class<-as.factor(dataset$class)
> set.seed(123)
> sample_indices<-sample(nrow(dataset),0.8*nrow(dataset))
> train_data<-dataset[sample_indices,]
>
```

CODE:

#Build the model

```
tree_model<-rpart(class~.,data=train_data,method="class")
print(tree_model)
#Make predictions on test set
predictions=predict(tree_model,test_data,type = "class")
#Accuracy score
accuracy=sum(predictions == test_data$class)/nrow(test_data)
cat("Accuracy:",accuracy,"\n")
```

OUTPUT:

```
> #Build the model
> tree_model<-rpart(class~.,data=train_data,method="class")
> print(tree_model)
n= 6499

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 6499 3155 e (0.51454070 0.48545930)
  2) odor=a,l,n 3440   96 e (0.97209302 0.02790698)
    4) spore.print.color=b,h,k,n,o,u,w,y 3380   36 e (0.98934911 0.01065089) *
    5) spore.print.color=r 60    0 p (0.00000000 1.00000000) *
  3) odor=c,f,m,p,s,y 3059    0 p (0.00000000 1.00000000) *
> #Make predictions on test set
> predictions=predict(tree_model,test_data,type = "class")
> #Accuracy score
> accuracy=sum(predictions == test_data$class)/nrow(test_data)
> cat("Accuracy:",accuracy,"\n")
Accuracy: 0.9926154
>
```

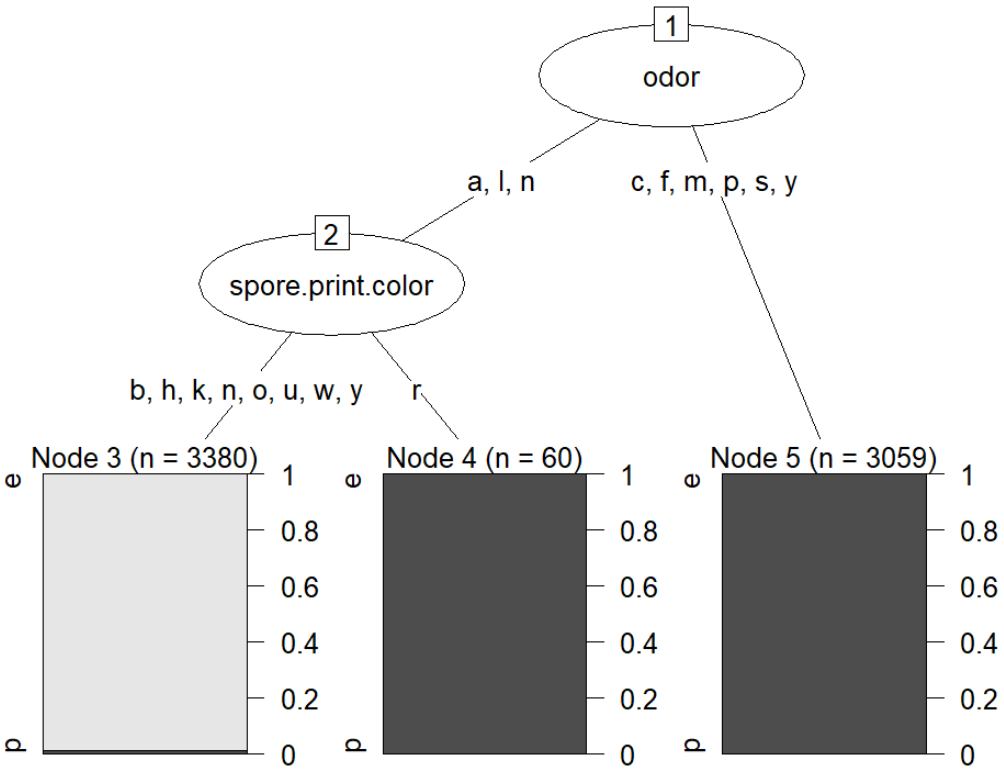
CODE:

#Visualize the tree

```
party_tree<-as.party(tree_model)
plot(party_tree)
```

OUTPUT:

```
> #Visualize the tree
> party_tree<-as.party(tree_model)
> plot(party_tree)
>
```



CODE:

```
#Display the few predictions
predictions=predict(tree_model,newdata=test_data,type = "class")
head(predictions)

#Calculate the precision,recall and F1-score.
actual_labels<-test_data$class
confusion_matrix<-table(actual_labels,predictions)
print(confusion_matrix)
```

OUTPUT:

```
> #Display the few predictions
> predictions=predict(tree_model,newdata=test_data,type = "class")
> head(predictions)
 3  8 10 12 14 15
 e  e  e  e  p  e
Levels: e p
> actual_labels<-test_data$class
> confusion_matrix<-table(actual_labels,predictions)
> print(confusion_matrix)
             predictions
actual_labels   e   p
            e 864   0
            p  12 749
>
```

CODE:

```
precision<-diag(confusion_matrix)/rowSums(confusion_matrix)

recall<-diag(confusion_matrix)/colSums(confusion_matrix)

f1_score<-2*(precision*recall)/(precision+recall)

result_df<-data.frame(Precision=precision,Recall=recall,F1_Score=f1_score)

rownames(result_df)<-levels(predictions)

print(result_df)
```

OUTPUT:

```
> precision<-diag(confusion_matrix)/rowSums(confusion_matrix)
> recall<-diag(confusion_matrix)/colSums(confusion_matrix)
> f1_score<-2*(precision*recall)/(precision+recall)
> result_df<-data.frame(Precision=precision,Recall=recall,F1_Score=f1_score)
> rownames(result_df)<-levels(predictions)
> print(result_df)
  Precision    Recall  F1_Score
e 1.0000000 0.9863014 0.9931034
p 0.9842313 1.0000000 0.9920530
```

CODE:

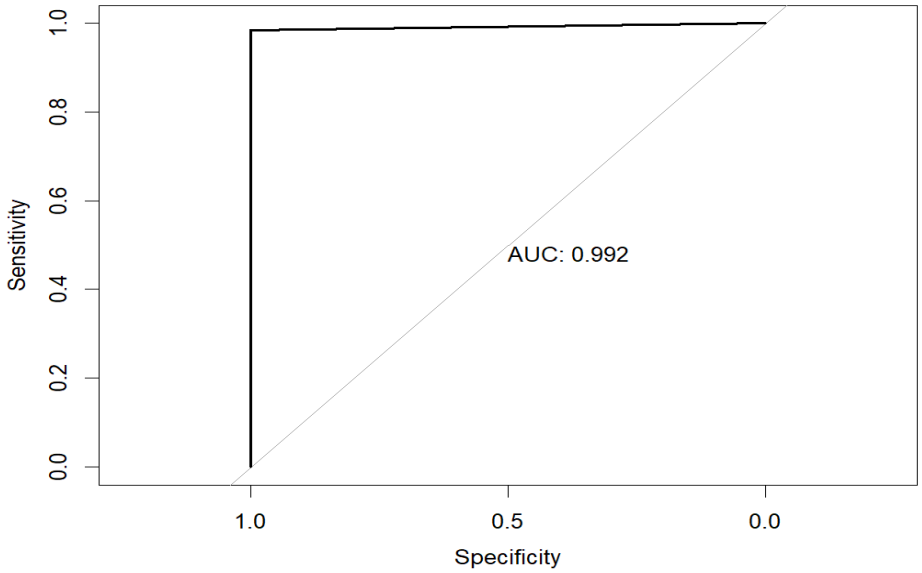
```
binary_predictions<-ifelse(predictions=="p",1,0)

roc_obj<-roc(as.numeric(test_data$class == "p"),binary_predictions)

plot(roc_obj,print.auc=TRUE)
```

OUTPUT:

```
> binary_predictions<-ifelse(predictions=="p",1,0)
> roc_obj<-roc(as.numeric(test_data$class == "p"),binary_predictions)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_obj,print.auc=TRUE)
>
```



CODE:

```
auc_score<-auc(roc_obj)
cat("AUC-ROC:",auc_score,"\n")
summary(tree_model)
```

OUTPUT:

```
> auc_score<-auc(roc_obj)
> cat("AUC-ROC:",auc_score,"\n")
AUC-ROC: 0.9921156
> summary(tree_model)
Call:
rpart(formula = class ~ ., data = train_data, method = "class")
  n= 6499

          CP nsplit  rel error     xerror        xstd
1 0.96957211      0 1.00000000 1.00000000 0.012770567
2 0.01901743      1 0.03042789 0.03042789 0.003082512
3 0.01000000      2 0.01141046 0.01141046 0.001896469

Variable importance
                  odor        spore.print.color                gill.color
                    25                       19                       15
stalk.surface.above.ring                ring.type stalk.surface.below.ring
                    14                       13                       13

Node number 1: 6499 observations,    complexity param=0.9695721
  predicted class=e  expected loss=0.4854593  P(node) =1
    class counts:  3344  3155
   probabilities: 0.515 0.485
  left son=2 (3440 obs) right son=3 (3059 obs)
  Primary splits:
      odor                     splits as  LRRLRLRRR,    improve=3060.110, (0 missing)
      spore.print.color        splits as  LRLLLRLRL,    improve=1757.867, (0 missing)
      gill.color               splits as  RLRRLLLLRLLL, improve=1247.353, (0 missing)
      stalk.surface.above.ring splits as  LRLL,         improve=1120.376, (0 missing)
      stalk.surface.below.ring splits as  LRLL,         improve=1054.844, (0 missing)


  Surrogate splits:
      spore.print.color        splits as  LRLLLLLRL,    agree=0.862, adj=0.707, (0 split)
      gill.color               splits as  RLRRLLLLLLLL, agree=0.812, adj=0.600, (0 split)
      stalk.surface.above.ring splits as  LRLL,         agree=0.781, adj=0.535, (0 split)
      ring.type                splits as  RLRRL,        agree=0.780, adj=0.533, (0 split)
      stalk.surface.below.ring splits as  LRLL,         agree=0.779, adj=0.530, (0 split)

Node number 2: 3440 observations,    complexity param=0.01901743
  predicted class=e  expected loss=0.02790698  P(node) =0.5293122
    class counts:  3344    96
   probabilities: 0.972 0.028
  left son=4 (3380 obs) right son=5 (60 obs)
  Primary splits:
      spore.print.color       splits as  LLLLLRLLL,    improve=115.40870, (0 missing)
      gill.color              splits as  -LLLLLLLRLLL, improve= 38.01964, (0 missing)
      stalk.color.below.ring splits as  --LLLLLLR,    improve= 34.19768, (0 missing)
      cap.color               splits as  RLLLLRLLLL,   improve= 21.94909, (0 missing)
      ring.number             splits as  -LR,          improve= 10.39475, (0 missing)
  Surrogate splits:
      gill.color splits as  -LLLLLLLRLLL, agree=0.988, adj=0.333, (0 split)

Node number 3: 3059 observations
  predicted class=p  expected loss=0  P(node) =0.4706878
    class counts:     0  3059
   probabilities: 0.000 1.000

Node number 4: 3380 observations
  predicted class=e  expected loss=0.01065089  P(node) =0.52008
    class counts:  3344    36
   probabilities: 0.989 0.011
```

```
Node number 5: 60 observations
  predicted class=p  expected loss=0  P(node) =0.00923219
    class counts:     0    60
   probabilities: 0.000 1.000
```

CODE:

```
pruned_tree<-prune(tree_model,cp=0.01)
summary(pruned_tree)
```

OUTPUT:

```
> pruned_tree<-prune(tree_model,cp=0.01)
> summary(pruned_tree)
Call:
rpart(formula = class ~ ., data = train_data, method = "class")
  n= 6499

          CP nsplit  rel error     xerror        xstd
1 0.96957211      0 1.00000000 1.00000000 0.012770567
2 0.01901743      1 0.03042789 0.03042789 0.003082512
3 0.01000000      2 0.01141046 0.01141046 0.001896469

Variable importance
                     odor        spore.print.color                gill.color
                       25                       19                       15
stalk.surface.above.ring                ring.type stalk.surface.below.ring
                       14                       13                       13

Node number 1: 6499 observations,    complexity param=0.9695721
  predicted class=e  expected loss=0.4854593  P(node) =1
    class counts:  3344  3155
   probabilities: 0.515 0.485
  left son=2 (3440 obs) right son=3 (3059 obs)
  Primary splits:
      odor                     splits as  LRRLRLRRR,    improve=3060.110, (0 missing)
      spore.print.color        splits as  LRLLLRLRL,    improve=1757.867, (0 missing)
      gill.color               splits as  RLRRLLLLRLLL, improve=1247.353, (0 missing)
      stalk.surface.above.ring splits as  LRLL,         improve=1120.376, (0 missing)
      stalk.surface.below.ring splits as  LRLL,         improve=1054.844, (0 missing)

  Surrogate splits:
      spore.print.color        splits as  LRLLLLLRL,    agree=0.862, adj=0.707, (0 split)
      gill.color               splits as  RLRRLLLLLLLL, agree=0.812, adj=0.600, (0 split)
      stalk.surface.above.ring splits as  LRLL,         agree=0.781, adj=0.535, (0 split)
      ring.type                splits as  RLRRL,        agree=0.780, adj=0.533, (0 split)
      stalk.surface.below.ring splits as  LRLL,         agree=0.779, adj=0.530, (0 split)

Node number 2: 3440 observations,    complexity param=0.01901743
  predicted class=e  expected loss=0.02790698  P(node) =0.5293122
    class counts:  3344    96
   probabilities: 0.972 0.028
  left son=4 (3380 obs) right son=5 (60 obs)
  Primary splits:
      spore.print.color      splits as  LLLLLRLLL,    improve=115.40870, (0 missing)
      gill.color             splits as  -LLLLLLLRLLL, improve= 38.01964, (0 missing)
      stalk.color.below.ring splits as  --LLLLLLR,    improve= 34.19768, (0 missing)
      cap.color              splits as  RLLLLRLLLL,   improve= 21.94909, (0 missing)
      ring.number            splits as  -LR,          improve= 10.39475, (0 missing)
  Surrogate splits:
      gill.color splits as  -LLLLLLLRLLL, agree=0.988, adj=0.333, (0 split)

Node number 3: 3059 observations
  predicted class=p  expected loss=0  P(node) =0.4706878
    class counts:     0  3059
   probabilities: 0.000 1.000

Node number 4: 3380 observations
  predicted class=e  expected loss=0.01065089  P(node) =0.52008
    class counts:  3344    36
   probabilities: 0.989 0.011
```

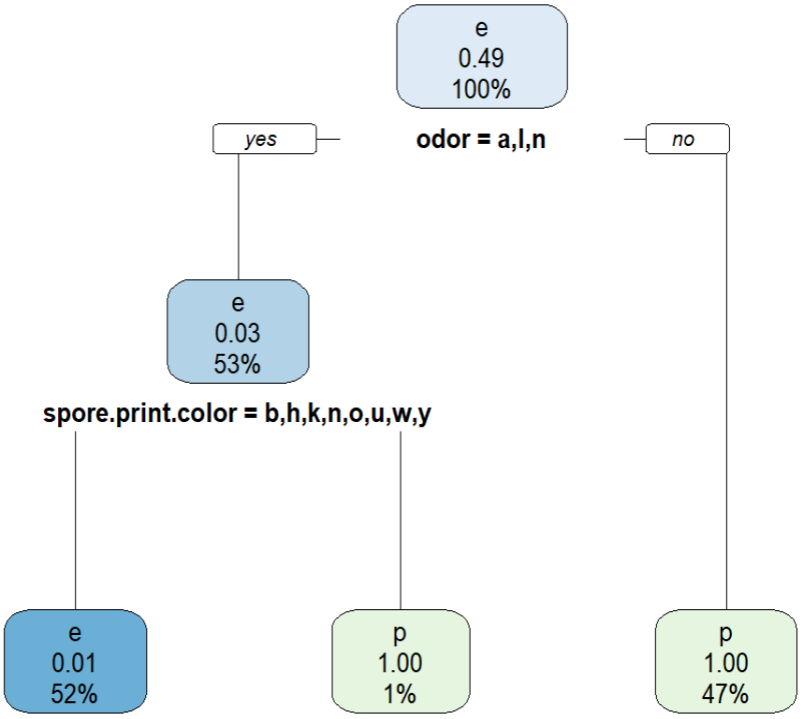

```
Node number 5: 60 observations
  predicted class=p  expected loss=0  P(node) =0.00923219
    class counts:     0    60
   probabilities: 0.000 1.000
```

CODE:

rpart.plot(pruned_tree)

OUTPUT:

```
> rpart.plot(pruned_tree)
>
```



CODE:

#make predictions on the test set using pruned_tree.

pruned_predictions<-predict(pruned_tree,newdata = test_data,type = "class")

pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)

print(pruned_cm)

OUTPUT:

```
> pruned_predictions<-predict(pruned_tree,newdata = test_data,type = "class")
> pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
> print(pruned_cm)
Confusion Matrix and Statistics

          Reference
Prediction   e   p
         e 864  12
         p   0 749

               Accuracy : 0.9926
                 95% CI : (0.9871, 0.9962)
    No Information Rate : 0.5317
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9852

 Mcnemar's Test P-Value : 0.001496

            Sensitivity : 1.0000
            Specificity : 0.9842
         Pos Pred Value : 0.9863
         Neg Pred Value : 1.0000
             Prevalence : 0.5317
         Detection Rate : 0.5317
   Detection Prevalence : 0.5391
      Balanced Accuracy : 0.9921

       'Positive' Class : e
```

CODE:

```
pruned_accuracy<-sum(pruned_predictions==test_data$class)/length(test_data$class)
cat("Pruned Model Accuracy : ",pruned_accuracy,"\n")
pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
pruned_precision<-pruned_cm$byClass["Positive predictive value"]
pruned_recall<-pruned_cm$byClass["Sensitivity"]
pruned_f1_score<-pruned_cm$byClass["F1"]
cat("Pruned model precision: ",pruned_precision,"\n")
cat("Pruned model recall: ",pruned_recall,"\n")
cat("Pruned model f1-score: ",pruned_f1_score,"\n")
```

OUTPUT:

```
> pruned_accuracy<-sum(pruned_predictions==test_data$class)/length(test_data$class)
> cat("Pruned Model Accuracy : ",pruned_accuracy,"\n")
Pruned Model Accuracy :  0.9926154
> pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
> pruned_precision<-pruned_cm$byClass["Positive predictive value"]
> pruned_recall<-pruned_cm$byClass["Sensitivity"]
> pruned_f1_score<-pruned_cm$byClass["F1"]
> cat("Pruned model precision: ",pruned_precision,"\n")
Pruned model precision:  NA
> cat("Pruned model recall: ",pruned_recall,"\n")
Pruned model recall:  1
> cat("Pruned model f1-score: ",pruned_f1_score,"\n")
Pruned model f1-score:  0.9931034
```

Findings:

Pruned model accuracy: 0.9926154

Pruned model precision: NA

Pruned model recall: 1

Pruned model f1-score: 0.9931034