MATH5743M: Statistical Learning: Assessed Practical 3 - Mushroom

Anupam Bose, 201570198, School of Mathematics

Semester 2 2022

The libraries to import to do further analysis on the data.

```
library(sf)
library(caret)
library(tidyverse)
library(rpart)
library(rpart.plot)
library(randomForest)
```

Task 1: Decision Tree

The mushroom dataset used in the report was submitted to the Machine Learning Repository a the University of California, Irvine. The data has been imported into a dataframe (df) by using the **read.csv** function.

```
df <- read_csv("mushrooms.csv", col_names = TRUE)
```

```
## Rows: 8124 Columns: 6

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (6): Edible, CapShape, CapSurface, CapColor, Odor, Height

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We can get some idea about the features of our dataset by **glimpse()** function.

```
glimpse(df)
```

```
## Rows: 8,124
## Columns: 6
## $ Edible     <chr> "Poisonous", "Edible", "Edible", "Poisonous", "Edible", "Ed~
## $ CapShape   <chr> "Convex", "Convex", "Bell", "Convex", "Convex", "Convex", "~
## $ CapSurface <chr> "Smooth", "Smooth", "Smooth", "Scaly", "Smooth", "Scaly", "~
## $ CapColor   <chr> "Brown", "Yellow", "White", "White", "Gray", "Yellow", "Whi~
## $ Odor       <chr> "Pungent", "Almond", "Anise", "Pungent", "None", "Almond", ~
## $ Height     <chr> "Tall", "Short", "Tall", "Short", "Short", "Short", "Short"~
```

We have total 8,124 samples in our dataset with 6 features. Edible column is our target variable which tells us if the mushroom is edible or poisonous. Rest of the features are the predictor variables which tells about the physical, olfactory and visual characteristics of the mushroom.

As all the variables are categorical and in character format, we must convert them to factors for the analysis. Converting the variables into factors helps to store the information in levels. Hence, they help in statistical modelling and data analysis.

```
##Making each variable as a factor
df <-df %>%
  map_df(function(.x) as.factor(.x))
```

We can now see that the variables have changed into factors:

```
glimpse(df)
```

```
## Rows: 8,124
## Columns: 6
## $ Edible     <fct> Poisonous, Edible, Edible, Poisonous, Edible, Edible, Edibl~
## $ CapShape   <fct> Convex, Convex, Bell, Convex, Convex, Convex, Bell, Bell, C~
## $ CapSurface <fct> Smooth, Smooth, Smooth, Scaly, Smooth, Scaly, Smooth, Scaly~
## $ CapColor   <fct> Brown, Yellow, White, White, Gray, Yellow, White, White, Wh~
## $ Odor       <fct> Pungent, Almond, Anise, Pungent, None, Almond, Almond, Anis~
## $ Height     <fct> Tall, Short, Tall, Short, Short, Short, Short, Tall, Tall, ~
```

```
number_class <- function(x){
  x <- length(levels(x))
}

#levels of each column
x <- df %>% map_dbl(function(.x) number_class(.x)) %>% as_tibble() %>%
  rownames_to_column() %>% arrange(desc(value))
colnames(x) <- c("Variable name", "Number of levels")

x <-x %>%
  mutate(Variable_Name=colnames(df[as.numeric(x$`Variable name`)])) %>%
  select(Variable_Name,`Number of levels`)
x
```

```
## # A tibble: 6 x 2
##   Variable_Name ‘Number of levels‘
##   <chr>                     <dbl>
## 1 CapColor                     10
## 2 Odor                          9
## 3 CapShape                      6
## 4 CapSurface                    4
## 5 Edible                        2
## 6 Height                        2
```

The table above shows the number of levels in each of our features of our data set. For an example, there are 10 different cap colors and 9 different odors of the mushroom. The target variable tells whether the mushroom is poisonous or edible.

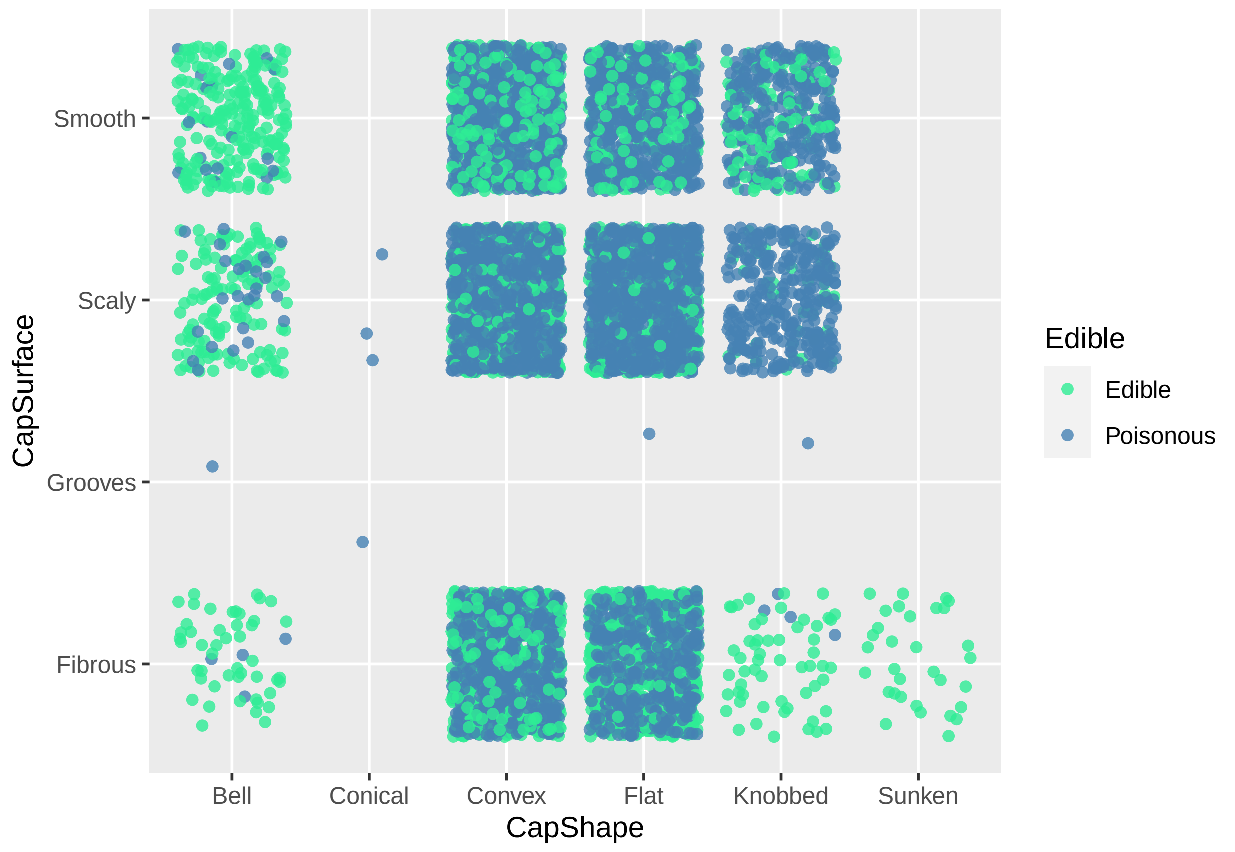
```r
#checking missing data
map_dbl(df, function(.x) {sum(is.na(.x))})
```

```
##     Edible   CapShape CapSurface   CapColor       Odor     Height
##          0          0          0          0          0          0
```

We don't have any missing values in our dataset. Hence, we don't need further cleaning of the data.

Visualizing our data will give us more information about our dataset with regards to the edibility of the mushroom.

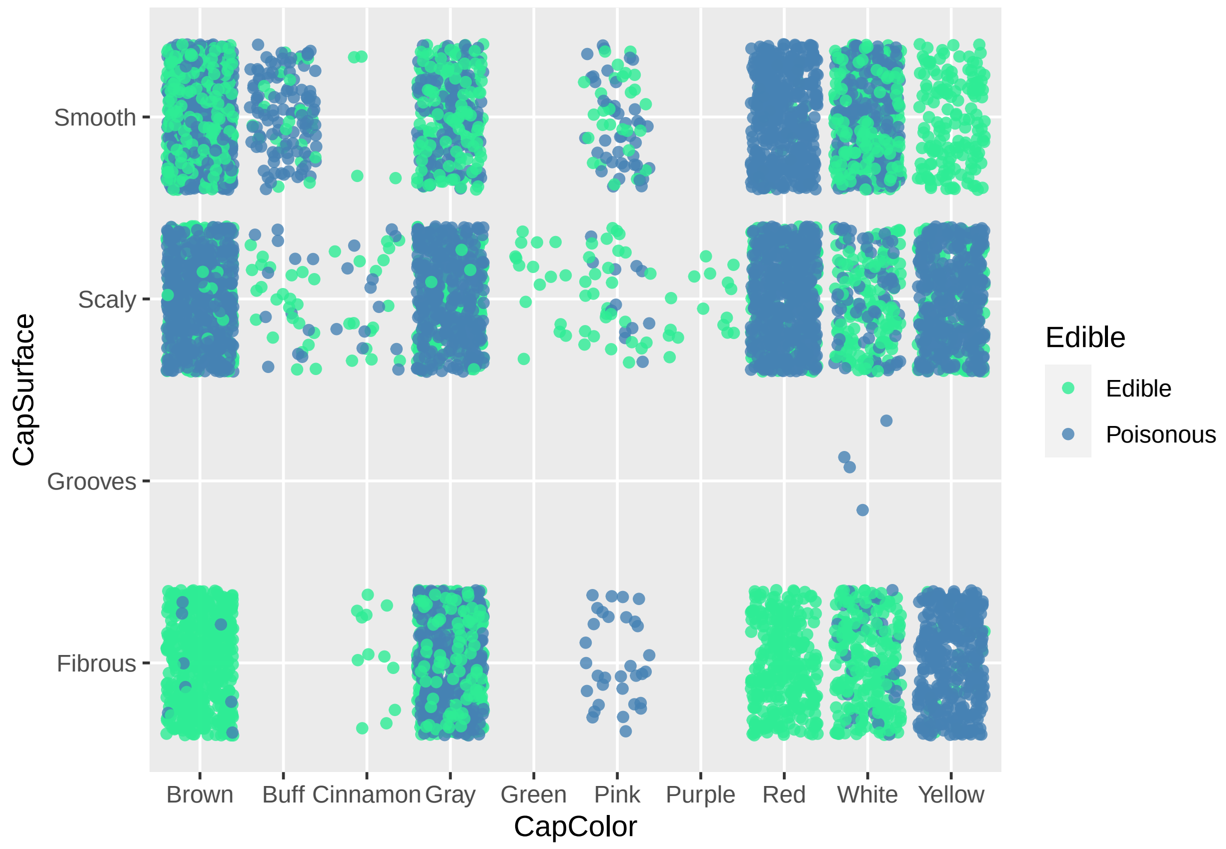
```r
#visualisations
ggplot(df, aes(x = CapShape, y = CapSurface, col = Edible)) +
  geom_jitter(alpha = 0.8) +
  scale_color_manual(breaks = c("Edible", "Poisonous"),
                     values = c("#2eed95", "steelblue"))
```



The plot tells us how the shape of the mushroom and its shape affect edibility. The following observations could be made from the plot:

- It is very risky to consume mushrooms whose shape is convex or flat.
- The fibrous mushrooms of sunken shape are extremely safe to consume.
- It is again risky to consume mushrooms of knobbed shape unless the surface of it is fibrous.
- Bell shaped mushrooms are mostly safe to consume.

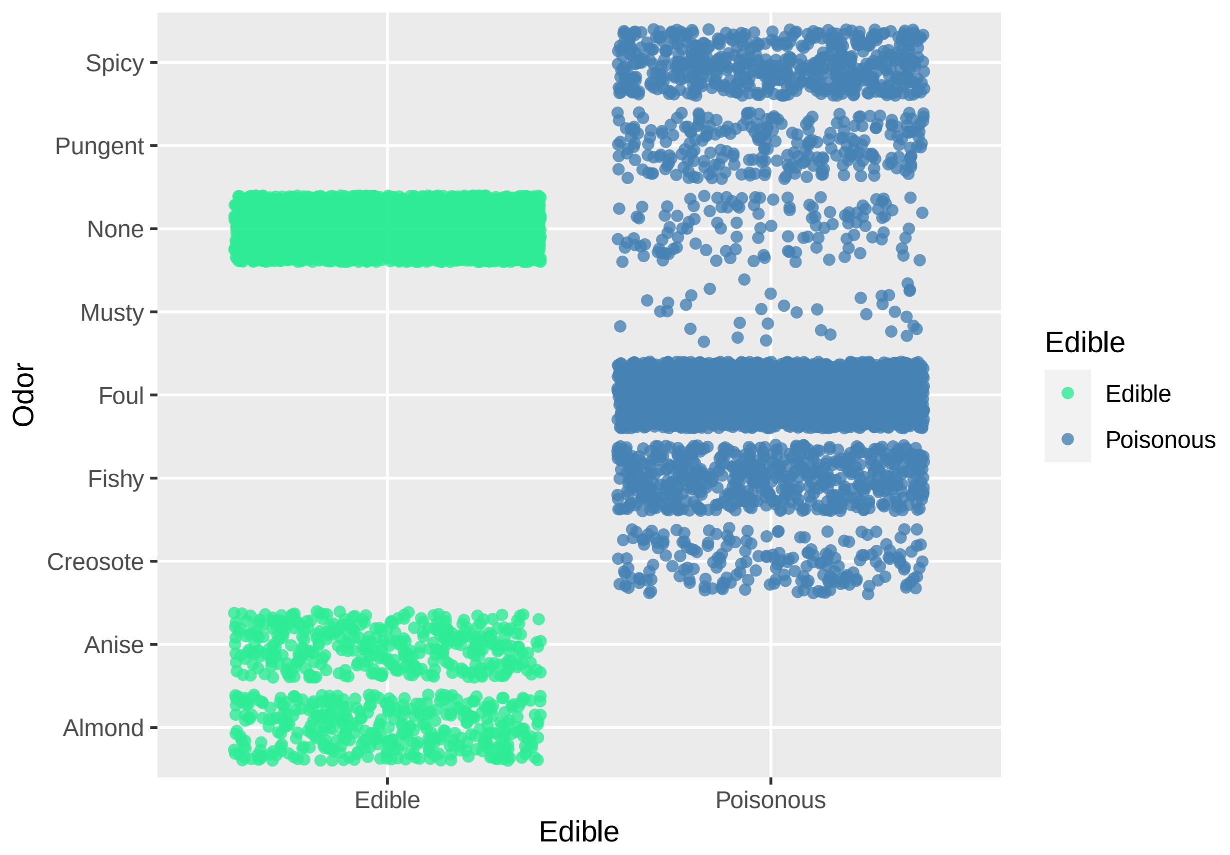
```
ggplot(df, aes(x = CapColor, y = CapSurface, col = Edible)) +
  geom_jitter(alpha = 0.8) +
  scale_color_manual(breaks = c("Edible", "Poisonous"),
                     values = c("#2eed95", "steelblue"))
```



The plot tells us how the surface of the mushroom and its color affect edibility. The following observations could be made from the plot:

- The mushroom having scaly surface have more chances of being poisonous unless they are of green or purple color.
- If the surface of the mushroom is fibrous, it mostly safe to eat them unless they of pink, grey or yellow color.
- It is extremely safe to eat mushroom whose surface is smooth and yellow in color. But if the color is red, it will be definitely poisonous.

```
ggplot(df, aes(x = Edible, y = Odor, col = Edible)) +
  geom_jitter(alpha = 0.8) +
  scale_color_manual(breaks = c("Edible", "Poisonous"),
                     values = c("#2eed95", "steelblue"))
```



The plot tells us how odor of the mushroom affect its edibility. The following observations could be made from the plot:

- It is 100 percent safe to consume the mushrooms which are odorless or if it smells like almond or anise.
- If it smells like anything else, it will be definitely poisonous. Basically, if it smells fishy, pungent or foul, it is good to stay away from it.
- Odorless mushrooms are always edible.

We divided the data into 70 percent training and 30 percent testing partitions.

To perform cross-validation, we will split the data into training and testing set. We divided the data into 70 percent training and 30 percent testing partitions.

```
set.seed(1800) #For reproducibility
#creating partition for training data
df_sample = createDataPartition(df$Edible, p=0.7, list= F)
df_train <- df[df_sample, ]
df_test <- df[-df_sample, ]
```

```
#checking quality of splits
round(prop.table(table(df$Edible)), 2)
```

```
##
##    Edible Poisonous
##      0.52      0.48
```

```r
round(prop.table(table(df_train$Edible)), 2)
```

```
##
##    Edible Poisonous
##      0.52      0.48
```

```r
round(prop.table(table(df_test$Edible)), 2)
```

```
##
##    Edible Poisonous
##      0.52      0.48
```

The quality of the splits can be seen here. The distribution of our target variable factors is the same in both the main dataset and the splitted dataset.

The complexity parameter (cp) is used to determine the optimal tree size and control the size of the decision tree. Tree building terminates if the cost of adding another variable to the decision tree from the current node exceeds the value of cp. We could alternatively argue that tree construction will stop unless the overall lack of fit is reduced by a factor of cp.

Putting CP value as zero will build a tree with its maximum depth. Hence, it will build a very large tree.

```r
TreeModel1 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.002)
TreeModel2 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.003)
TreeModel3 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.004)
TreeModel4 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.00001)

cfmat1=confusionMatrix(data=predict(TreeModel1, type = "class"),
                      reference = df_train$Edible,
                      positive="Edible")
cfmat2=confusionMatrix(data=predict(TreeModel2, type = "class"),
                       reference = df_train$Edible,
                       positive="Edible")
cfmat3=confusionMatrix(data=predict(TreeModel3, type = "class"),
                       reference = df_train$Edible,
                       positive="Edible")
cfmat4=confusionMatrix(data=predict(TreeModel4, type = "class"),
                       reference = df_train$Edible,
                       positive="Edible")
#model accuracy with c.p = 0.002
cfmat1$overall[1]
```

```
##  Accuracy
## 0.9868143
```

```r
#model accuracy with c.p = 0.003
cfmat2$overall[1]
```

```
##  Accuracy
## 0.9868143
```

```r
#model accuracy with c.p = 0.004
cfmat3$overall[1]
```

```
##  Accuracy
## 0.9832982
```

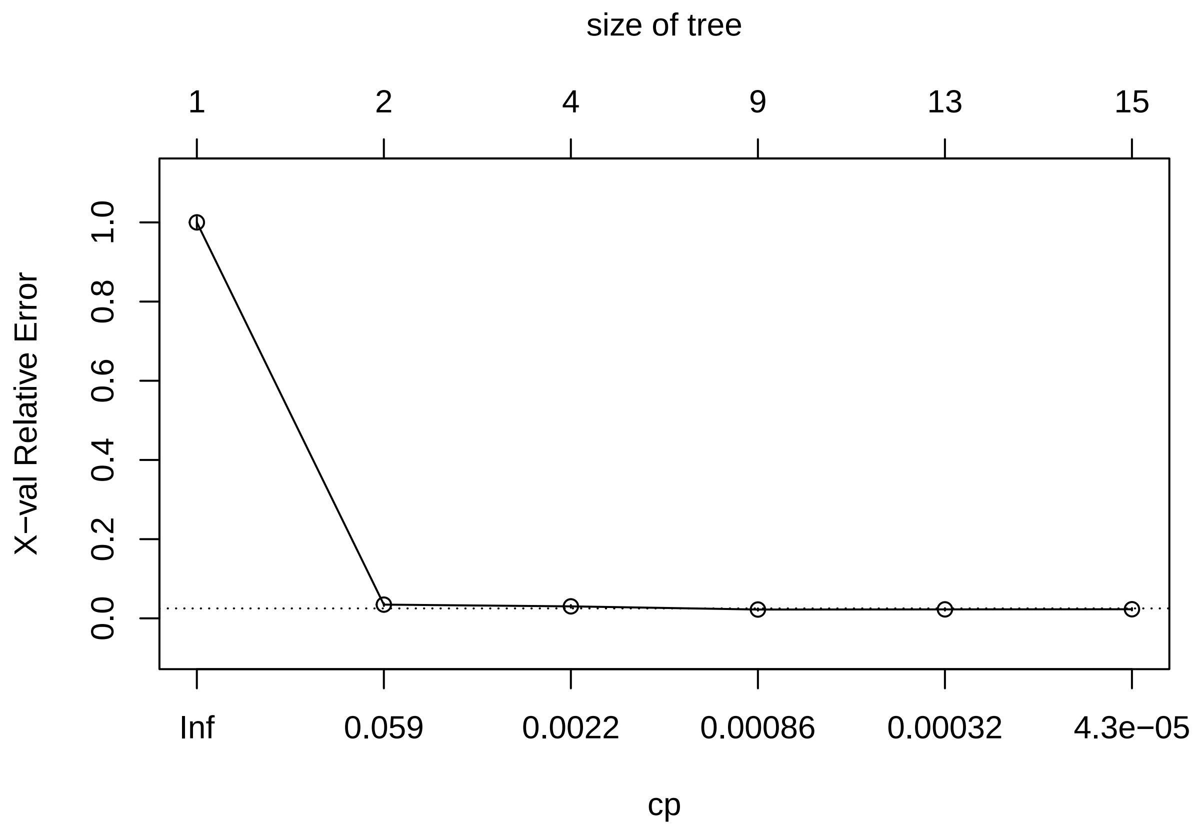
```r
#model accuracy with c.p = 0.00001
cfmat4$overall[1]
```

```
## Accuracy
## 0.991737
```

We used four randomly chosen cp values to train the decision tree model. Then we examine how well each model predicts.

- Model 1 and model 2 with cp value of 0.002 and 0.003 respectively have same accuracy which is 98.68%.
- Model 4 predicts most accurately with the accuracy of 99.17%.
- It's also worth noting that there's not much of a difference in accuracy between all of the models when comparing tree size, as larger cp values indicate a less complex tree that's easier to grasp and more computationally efficient.
- But for now we will choose cp value of 0.00001 for further analysis.

```r
plotcp(TreeModel4)
```

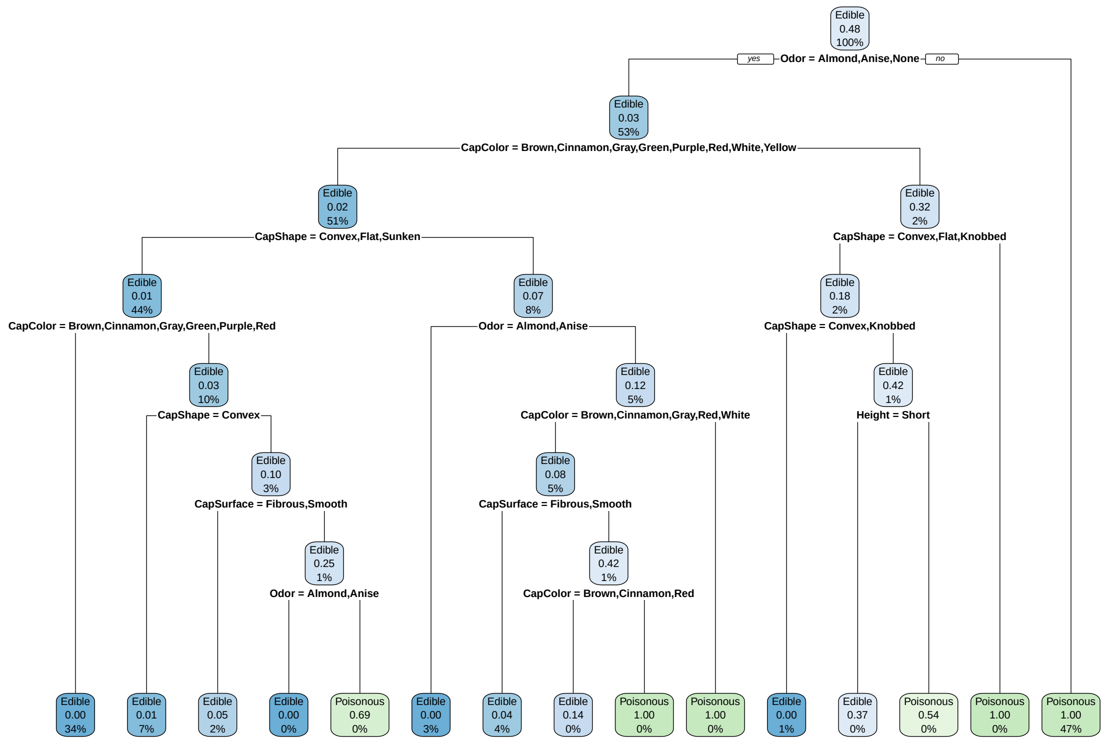


```
printcp(TreeModel4)
```

```
##
## Classification tree:
## rpart(formula = Edible ~ ., data = df_train, method = "class",
##     cp = 1e-05)
##
## Variables actually used in tree construction:
## [1] CapColor   CapShape   CapSurface Height     Odor
##
## Root node error: 2742/5688 = 0.48207
##
## n= 5688
##
##           CP nsplit rel error   xerror      xstd
## 1 0.96535376      0  1.000000 1.000000 0.0137437
## 2 0.00364697      1  0.034646 0.034646 0.0035248
## 3 0.00133722      3  0.027352 0.030270 0.0032982
## 4 0.00054705      8  0.019694 0.022247 0.0028331
## 5 0.00018235     12  0.017505 0.022611 0.0028559
## 6 0.00001000     14  0.017141 0.022976 0.0028786
```

- We can see from the plot and table that the cross validation error decreases as the cp value drops until the number of splits reaches 8, at which point it begins to increase. As a result, the cp value for it is 0.00054705, and the tree can be pruned with a cross validation error of 0.022247.
- Choosing a very small value of cp is very helpful to fit the model as we can see various values of cp for various tree sizes. The optimal value of cp can be chosen with lowest cross validation error. In our case the optimal number of splits is 8.
- We can now check the accuracy of the model by using optimum value of the cp.

```
rpart.plot(TreeModel4)
```



The plot here shows the the decision tree with cp value of 0.00001.

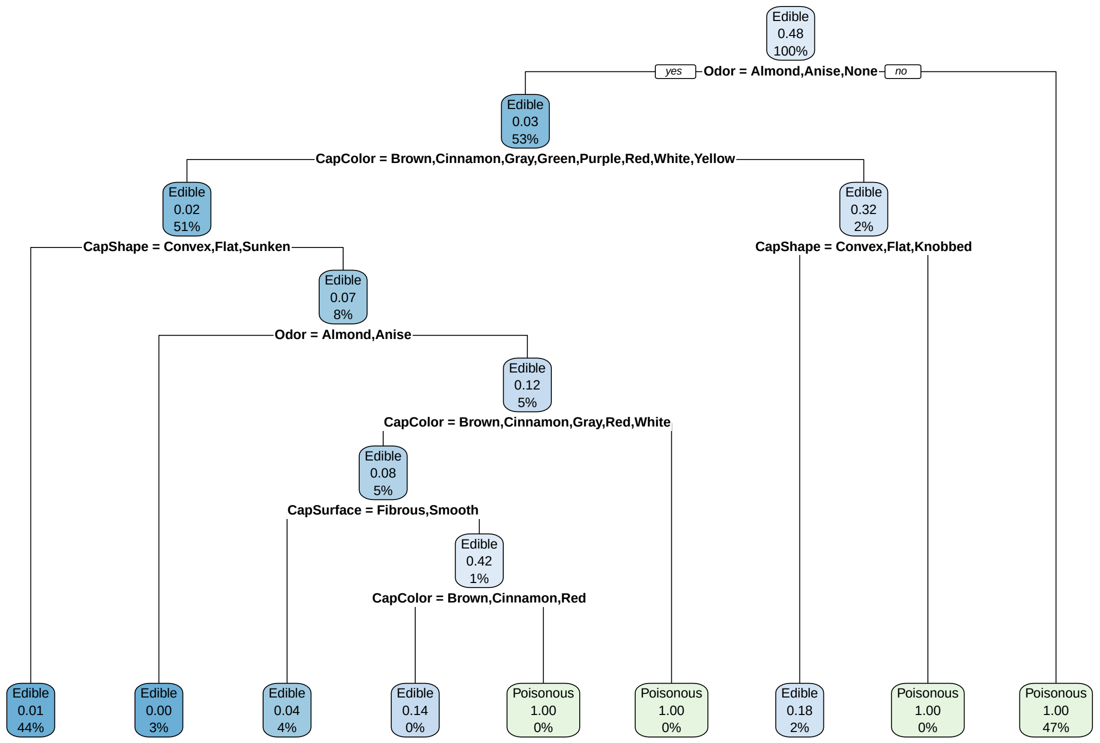
Now we will train the model with optimal value of cp with 8 splits and check the accuracy.

```
TreeModel5 <- rpart(Edible ~ ., data = df_train, method = "class",cp = 0.00054705)
cfmat5=confusionMatrix(data=predict(TreeModel5, type = "class"),
                       reference = df_train$Edible,
                       positive="Edible")
cfmat5$overall[1]
```

```
##  Accuracy
## 0.9905063
```

Here we see that the accuracy is 99.05% which is very close to 99.17%. Let's also look at the size of the decision tree.

```
rpart.plot(TreeModel5)
```



It's interesting to see from the plot that the accuracy doesn't seem to alter much as the tree gets smaller. As a result, it makes sense to use this model, which are both computationally efficient and easy to understand while maintaining the same level of accuracy.

Then, when the training data changes, we'll see how much the accuracy changes. As a result, we'll randomly split the data into testing and training sets fifty times with a cp value of 0.00054705.

```
#change in accuracy with variation in dataset
#Optimal cp value is chosen in every loop with lowest cross validation error and highest accuracy.
winner = rep(NA, 50)
for (iteration in 1:50){
  #Make a new random training data - test data split
  idx=createDataPartition(df$Edible, p=0.7, list= F)
  #idx = sample(1:8124, 5687)
  train_data = df[idx, ]
  test_data = df[-idx, ]
  cpval = rep(NA, 50)
  BestAcc = rep(NA, 50)

  for (i in 1:50){
    #Fit model with the training data
    TreeModelCV <- rpart(Edible ~ ., data = train_data,
                         method = "class", cp = 0.00054705)

    cpval[i]=TreeModelCV$cptable[which.min(TreeModelCV$cptable[, "xerror"]), "CP"]
    TreeModelCVPruned=prune(TreeModelCV, cp = cpval[i])
    cfmat=confusionMatrix(data=predict(TreeModelCVPruned, type = "class"),
                           reference = train_data$Edible,
                           positive="Edible")

    BestAcc[i]=cfmat$overall[1]
  }
  #Winning accuracy for this iteration is stored
  winner[iteration] = max(BestAcc)
}
```

```
#accuracy of 50 randomly chosen training dataset with
#cp value = 0.00054705
winner
```

```
##  [1] 0.9924402 0.9927918 0.9913854 0.9919128 0.9917370 0.9919128 0.9938467
##  [8] 0.9920886 0.9910338 0.9926160 0.9934951 0.9906821 0.9922644 0.9924402
## [15] 0.9905063 0.9908579 0.9912096 0.9910338 0.9933193 0.9915612 0.9912096
## [22] 0.9913854 0.9910338 0.9912096 0.9922644 0.9912096 0.9905063 0.9906821
## [29] 0.9912096 0.9931435 0.9906821 0.9910338 0.9920886 0.9931435 0.9931435
## [36] 0.9929677 0.9901547 0.9920886 0.9912096 0.9917370 0.9905063 0.9913854
## [43] 0.9922644 0.9913854 0.9906821 0.9912096 0.9926160 0.9908579 0.9912096
## [50] 0.9926160
```

The results above show that even when the training dataset varies, the accuracy does not change significantly. We acquire a maximum accuracy of 99.38%.

```
#cp value corresponding to max accuracy
MostAccModel=which.max(winner)
bestcpval=cpval[MostAccModel]
bestcpval
```

```
## [1] 0.00054705
```

The best cp value with lowest cross validation error and highest accuracy(99.38%) is still the same. Hence, we choose our final CP value as 0.00054705.

Let's check the performance of the final model on the entire dataset.

```
#Accuracy on entire dataset with optimal cp val
#C.P = 0.00054705
confusionMatrix(data = predict(TreeModel5, newdata = df, type = "class"),
                reference = df$Edible,
                positive = "Edible")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Edible Poisonous
##   Edible      4208        70
##   Poisonous      0      3846
##
##                Accuracy : 0.9914
##                  95% CI : (0.9891, 0.9933)
##     No Information Rate : 0.518
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9827
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.9821
##          Pos Pred Value : 0.9836
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5180
##          Detection Rate : 0.5180
##    Detection Prevalence : 0.5266
##       Balanced Accuracy : 0.9911
##
##        'Positive' Class : Edible
##
```

```
#Accuracy of model with cp = 0.00001 on testing data
confusionMatrix(data = predict(TreeModel4, newdata = df, type = "class"),
                reference = df$Edible,
                positive = "Edible")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Edible Poisonous
##   Edible      4187        49
##   Poisonous     21      3867
##
##                Accuracy : 0.9914
##                  95% CI : (0.9891, 0.9933)
##     No Information Rate : 0.518
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9827
##
##  Mcnemar's Test P-Value : 0.00125
##
##             Sensitivity : 0.9950
##             Specificity : 0.9875
##          Pos Pred Value : 0.9884
##          Neg Pred Value : 0.9946
##              Prevalence : 0.5180
##          Detection Rate : 0.5154
##    Detection Prevalence : 0.5214
##       Balanced Accuracy : 0.9912
##
##        'Positive' Class : Edible
##
```

The following observations can made from the confusion matrices:

- The accuracy of prediction using the final model on the testing dataset is 99.14%.
- It is fine to eat mushroom which has been predicted as poisonous but when it is edible in reality. On the other hand, if the mushroom is poisonous in reality but the model predicted it as edible is very dangerous.
- From the confusion matrix, we can see that the probability of eating a poisonous mushroom predicted as edible is 0.86%. Hence, the chances of eating a poisonous mushroom is extremely low.
- Odor, CapColor, CapShape, and CapSurface are the four most important characteristics in predicting the target variables.
- When compared to the model with cp = 0.00001%, the model's accuracy in predicting the target variable is the same, at 99.14% percent. However, there is a 0.6% chance of eating a poisonous mushroom that has been predicted as edible. There isn't much difference in the probability too.
- Hence, our final model with cp = 0.00054705 is most optimal model which is relatively less computationally intensive with smaller tree size but same accuracy and the difference in the chance of eating poisonous mushroom predicted as edible is 0.26%

Task 2: Random Forest

We will now use Random Forest to evaluate the model's performance and compare it to decision tree.

A random forest is a machine learning technique for solving regression and classification problems. It makes use of ensemble learning, a technique that combines many classifiers to solve complex problems. A random forest algorithm is made up of numerous decision trees. The outcome is determined by the (random forest) algorithm based on the predictions of the decision trees. It predicts by averaging or averaging the output of various trees. The precision of the outcome improves as the number of trees increases. A random forest algorithm overcomes the limitations of the decision tree algorithm. It reduces dataset overfitting and improves precision.

To fit the model and predict the target variable, the **randomForest()** function is used. It is also used to determine the relative importance of predictor variables.

Decision Tree Drawbacks

- Decision-tree learners who do not adequately generalise the input may create too complicated trees which leads to over-fitting.
- The cost of creating a decision tree is high since each node requires field sorting. Other approaches combine multiple fields at the same time, resulting in even higher prices. Pruning methods are particularly expensive because to the large number of candidate subtrees that must be produced and compared.
- Any change in data causes tree to be formed which makes the decision tree unstable.

We divide the data again into 70 percent training and 30 percent testing partitions.

```r
df1 <- read_csv("mushrooms.csv", col_names = TRUE)
```

```
## Rows: 8124 Columns: 6
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (6): Edible, CapShape, CapSurface, CapColor, Odor, Height
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#cross-validation
set.seed(3) #For reproducibility
#creating partition for training data
df_sample = createDataPartition(df1$Edible, p=0.7, list= F)
df_train_rf <- df1[df_sample, ]
df_test_rf <- df1[-df_sample, ]
```

```
formulas = c("Edible ~ CapShape", "Edible ~ CapSurface", "Edible ~ CapColor","Edible ~ Odor","Edible ~ H
             "Edible ~ CapShape + CapSurface", "Edible ~ CapShape + CapColor", "Edible ~ CapShape + Odor
             "Edible ~ CapSurface + CapColor","Edible ~ CapSurface + Odor","Edible ~ CapSurface + Height
             "Edible ~ CapColor + Height","Edible ~ Odor + Height",
             "Edible ~ CapShape + CapSurface + CapColor","Edible ~ CapSurface + CapColor + Odor","Edible
             "Edible ~ CapShape + CapColor + Odor","Edible ~ CapShape + Odor + Height","Edible ~ CapShap
             "Edible ~ CapShape + CapSurface + Height","Edible ~ CapSurface + CapColor + Height","Edible
             "Edible ~ CapShape + CapColor + Height",
             "Edible ~ CapShape + CapSurface + CapColor + Odor","Edible ~ CapSurface + CapColor + Odor +
             "Edible ~ CapShape + CapColor + Odor + Height","Edible ~ CapShape + CapSurface + Odor + Hei
             "Edible ~ CapShape + CapSurface + CapColor + Height",
             "Edible ~ CapShape + CapSurface + CapColor + Odor + Height")
length(formulas)
```

```
## [1] 31
```

We have constructed every possible predictor variable combinations to check if combination gives the best model with highest accuracy in predicting the target variable.

There are total 31 possible combinations of the predictor variable.

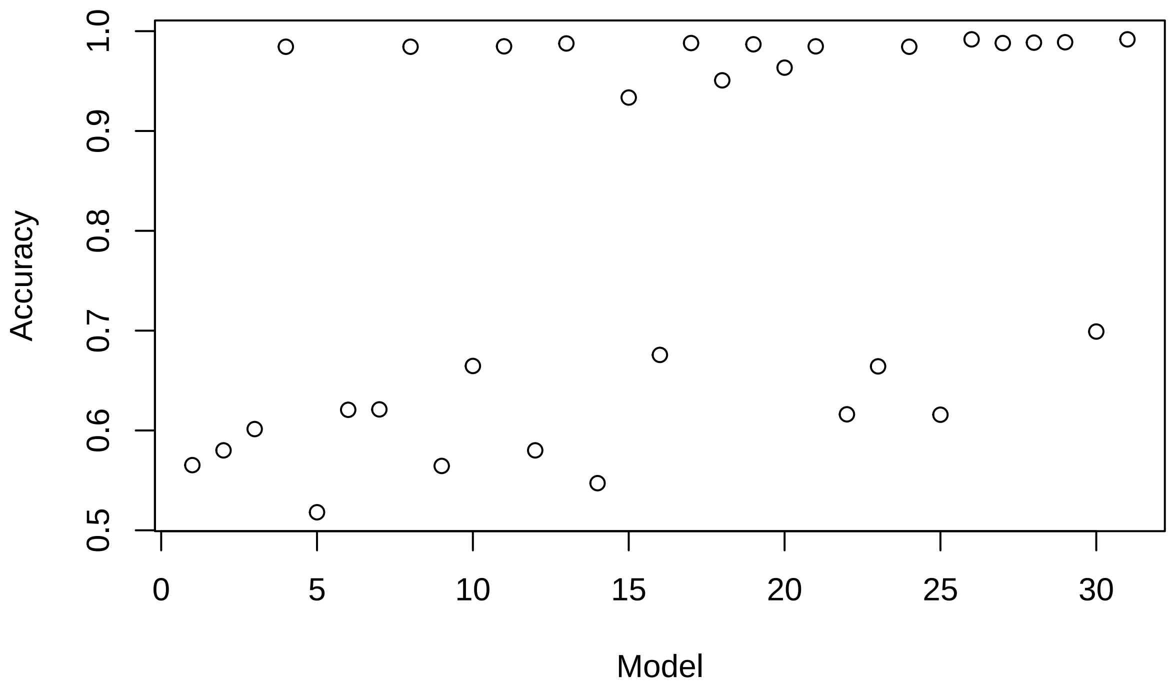
The following code has a loop that uses the random forest fit-predict process for cross-validation and stores the prediction's quality. As a measure of prediction quality, we'll utilise the predicted random forest of the actual test outputs.

```
df_train_rf$Edible = as.factor(df_train_rf$Edible)
accuracy = rep(NA, length(formulas))
for(i in 1:length(formulas)){
  myForest = randomForest(formula=as.formula(formulas[i]),
                          data=df_train_rf)
  prediction = predict(myForest, df_test_rf)
  prediction_df = data.frame(df_test_rf$Edible, prediction)

  cnf_mat = table(prediction_df$df_test_rf.Edible, prediction_df$prediction)

  accuracy[i] = (cnf_mat[1,1] + cnf_mat[2,2])/dim(df_test_rf)[1]
}

formulas[which.max(accuracy)]
```

```
## [1] "Edible ~ CapShape + CapSurface + CapColor + Odor"
```

The mushroom's shape, surface type, colour, and odour are the characteristics that provide the most accurate model. It's interesting to note that not all of the features are required to get the best model. It outperforms the model having all the predictor variables. Hence, the mushroom's height is not required to accurately predict the target variable.

```
plot(1:length(formulas), accuracy, xlab="Model", ylab="Accuracy")
```



From the plot we can conclude that there are many models which gives good accuracy. The only interesting point to note is that if only single predictor variable is considered to check the accuracy, odor of the mushroom alone helps to predict the target variable accurately. Hence, it is most significant feature in predicting the target variable.

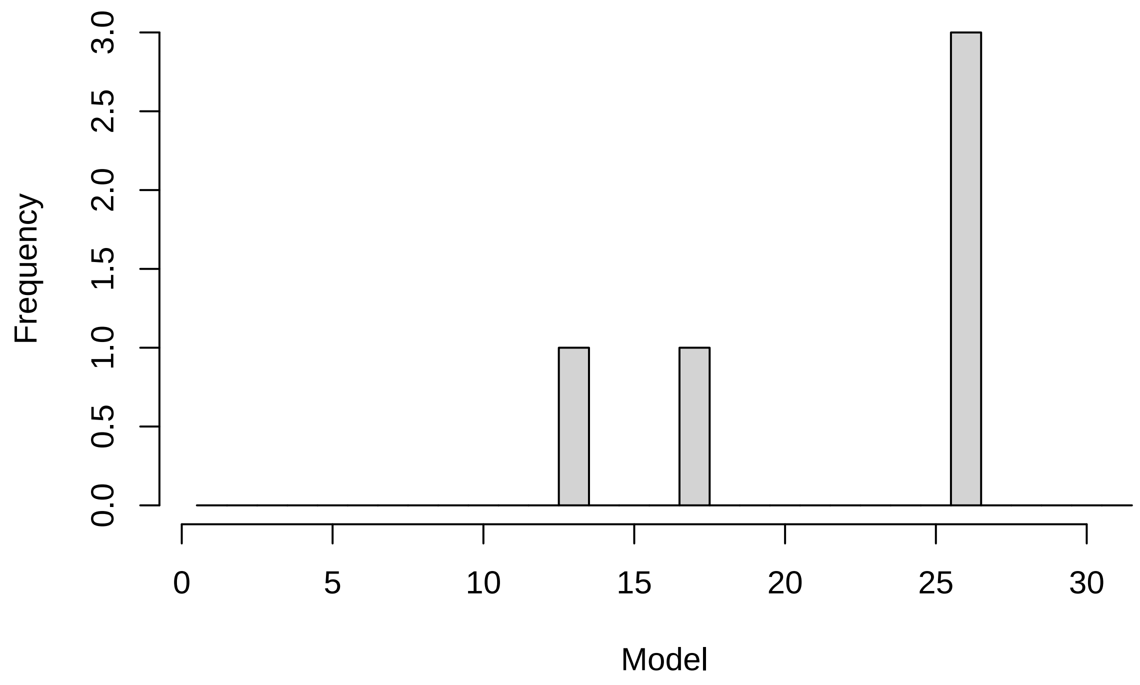
We can now repeat the procedure many times, each time selecting a different random split in the data. Because the number of samples is large (8124 rows), a small number of iterations will enough in this case. When working with a tiny dataset, we should insist on doing more repetitions.The model wins is tracked which gives most accurate result in predicting the target variable.

```
set.seed(3)
rf_winner = rep(NA, 5)
for(iter in 1:5){
  accuracy = rep(NA, length(formulas))
  idx = createDataPartition(df1$Edible, p=0.7, list = F)
  df_train_rf = df1[idx, ]
  df_test_rf = df1[-idx, ]
  df_train_rf$Edible = as.factor(df_train_rf$Edible)
  for(i in 1:length(formulas)){
    myForest = randomForest(formula=as.formula(formulas[i]),
                              data=df_train_rf)
    prediction = predict(myForest, df_test_rf)
    prediction_df = data.frame(df_test_rf$Edible, prediction)

    cnf_mat = table(prediction_df$df_test_rf.Edible, prediction_df$prediction)

    accuracy[i] = (cnf_mat[1,1] + cnf_mat[2,2])/dim(df_test_rf)[1]

  }
  rf_winner[iter] = which.max(accuracy)
}
formulas[which.max(accuracy)]
```

```
## [1] "Edible ~ CapShape + CapSurface + CapColor + Odor"
```

The mushroom's shape, surface type, colour, and odour (model 26) are again the characteristics that provide the most accurate model. Hence, it is the best model after training it multiple times.

```
hist(rf_winner, breaks = seq(0.5, 31.5, 1), xlab="Model", ylab="Frequency", main="")
```



From the histogram plot, we can conclude again that model 26 is the best model. Also, we see that model 13 and 17 has won many times as mentioned below.

```
#model 13
formulas[13]
```

```
## [1] "Edible ~ CapColor + Odor"
```

```
#model 17
formulas[17]
```

```
## [1] "Edible ~ CapSurface + CapColor + Odor"
```

```
myForest1 = randomForest(formula=as.formula(formulas[26]),
                         data=df_train_rf)
prediction1 = predict(myForest1, df1)
prediction_df1 = data.frame(df1$Edible, prediction1)

cnf_mat1 = table(prediction_df1$df1.Edible, prediction_df1$prediction1)
cnf_mat1
```

```
##
##              Edible Poisonous
##   Edible      4208         0
##   Poisonous     74      3842
```

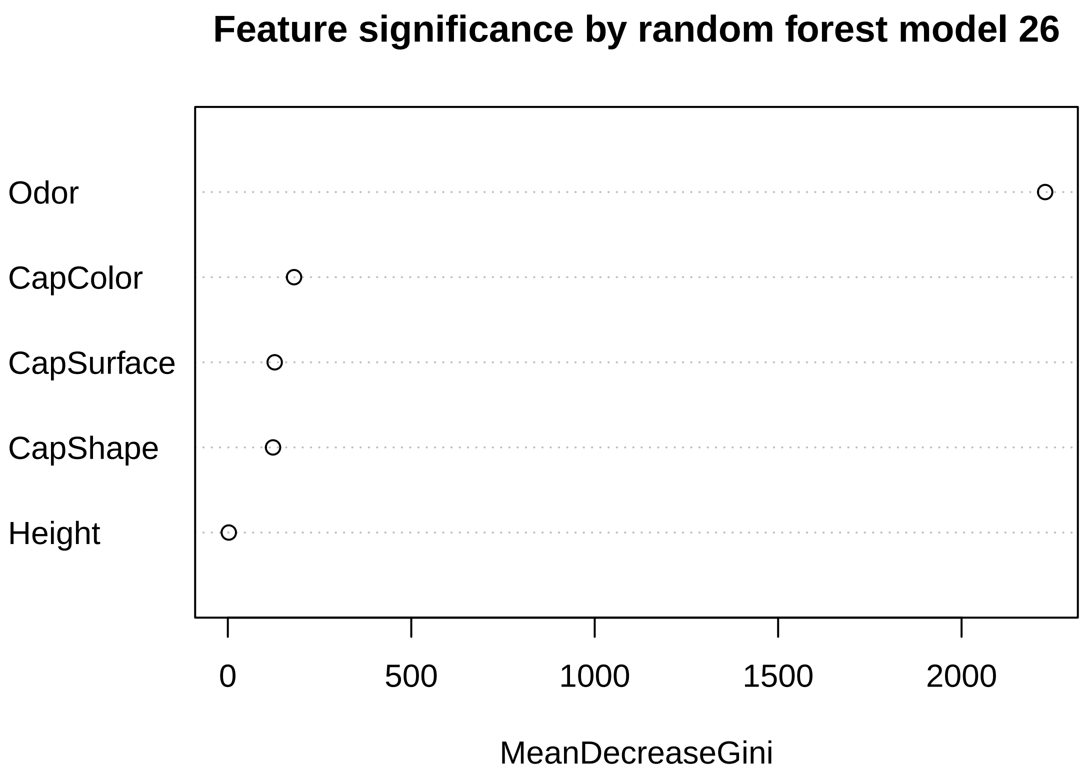
```
accuracy1 = (cnf_mat1[1,1] + cnf_mat1[2,2])/dim(df1)[1]
accuracy1
```

```
## [1] 0.9908912
```

Following observations could be made from the confusion matrix and the accuracy:

- The accuracy of predicting the target variable correctly is 99.09%
- the probability of eating a poisonous mushroom predicted as edible is 0%.
- The probability if eating a edible mushroom predicted as poisonous is 0.91%
- Hence, there is zero percent chance that people consuming mushroom based on our predicted will be poisonous.

```
varImpPlot(myForest,
           sort = T,
           main="Feature significance by random forest model 26")
```



The plot concludes that odor of the mushroom is most significant characteristic in predicting whether the mushroom is edible or poisonous. We have seen this many times in our analysis. The height of the mushroom is least significant. It is evident from the fact that model 26 has won most number of times which does not include height of the mushroom.

Comparing Decision Tree model and Random Forest model:

- The accuracy of predicting edibility of the mushroom by Decision tree is 99.14% and 99.08% for random forest model.
- Both the models chose odor to be the most significant variable in predicting the target variable accurately followed by capColor, capShape and capSurface.
- Decision tree is relatively more accurate than random forest but in the case of random forest, the probability of consuming a poisonous mushroom predicted as edible is 0%. Hence, there is zero percent chance that people consuming mushroom based on the prediction by random forest model will be poisonous.
- Also, considering the drawbacks of decision tree, we will choose random forest as our final model to predict the edibility of the mushroom.

Task 3:

- If I had to forge mushrooms, I would absolutely utilise the classifiers stated by the two models above.
- Both the models chose odor to be the most significant variable in predicting the target variable accurately followed by capColor, capShape and capSurface. Hence, the odor, color, shape and surface of the mushroom can predict the edible mushroom accurately.

- If I were to open a cafe and use decision tree to classify the mushroom. 1 out of every 100 customer will eat a poisonous mushroom. Hence, it is very important to be 100% dure that the customer is eating an edible mushroom.
- The most significant factor to check the edibility of the mushroom is odor. If the mushroom smells fishy, foul, pungent or fishy, we must stay away from it. We must look for the mushroom which is odorless or smells like anise or almond.
- In the case of random forest model, there is 0.91% chance of eating a edible mushroom which we predicted as poisonous. But as we have predicted the mushroom to be poisonous, it makes no sense to consume it. Hence, there is zero percent chance that people consuming mushroom based on the prediction by random forest model will be poisonous. .