Name: Rohit Goutam Maity

Assignment: HW3

Course Number: DSC 441

Problem 1 (15 points):

For this problem, you will perform a straightforward training and evaluation of a decision tree, as well as generate rules by hand. Load the breast_cancer_updated.csv data. These data are visual features computed from samples of breast tissue being evaluated for cancer1. As a preprocessing step, remove the IDNumber column and exclude rows with NA from the dataset. a. Apply decision tree learning (use rpart) to the data to predict breast cancer malignancy (Class) and report the accuracy using 10-fold cross validation. b. Generate a visualization of the decision tree. c. Generate the full set of rules using IF-THEN statements.

Ans: - Load required libraries

```
# Load required libraries
library(rpart)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(rpart.plot)
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

- Step 1: Load and Preprocess the Data

```
data <- read.csv("breast_cancer_updated.csv")

data <- data[, !(names(data) %in% "IDNumber")]

data_cleaned <- na.omit(data)

str(data_cleaned)
```

```
## 'data.frame':    683 obs. of  10 variables:
##  $ ClumpThickness    : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ UniformCellSize   : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ UniformCellShape  : int  1 4 1 8 1 10 1 2 1 1 ...
##  $ MarginalAdhesion  : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ EpithelialCellSize: int  2 7 2 3 2 7 2 2 2 2 ...
##  $ BareNuclei        : int  1 10 2 4 1 10 10 1 1 1 ...
##  $ BlandChromatin    : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ NormalNucleoli    : int  1 2 1 7 1 7 1 1 1 1 ...
##  $ Mitoses           : int  1 1 1 1 1 1 1 1 5 1 ...
##  $ Class             : chr  "benign" "benign" "benign" "benign" ...
##  - attr(*, "na.action")= 'omit' Named int [1:16] 24 41 140 146 159 165 236 250 276 293 ...
##   ..- attr(*, "names")= chr [1:16] "24" "41" "140" "146" ...
```

- Step 3: Train the Decision Tree Model

```
library(rpart.plot)
tree_model <- rpart(Class ~ ., data = data_cleaned, method = "class")

# Print the tree model summary
print(tree_model)
```

```
## n= 683
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 683 239 benign (0.65007321 0.34992679)
##    2) UniformCellSize< 2.5 418  12 benign (0.97129187 0.02870813)
##      4) BareNuclei< 5.5 410   5 benign (0.98780488 0.01219512) *
##      5) BareNuclei>=5.5 8   1 malignant (0.12500000 0.87500000) *
##    3) UniformCellSize>=2.5 265  38 malignant (0.14339623 0.85660377)
##      6) UniformCellShape< 2.5 23   5 benign (0.78260870 0.21739130)
##       12) BlandChromatin< 3.5 16   0 benign (1.00000000 0.00000000) *
##       13) BlandChromatin>=3.5 7   2 malignant (0.28571429 0.71428571) *
##      7) UniformCellShape>=2.5 242  20 malignant (0.08264463 0.91735537)
##       14) UniformCellSize< 4.5 68  17 malignant (0.25000000 0.75000000)
##         28) BareNuclei< 2.5 14   4 benign (0.71428571 0.28571429) *
##         29) BareNuclei>=2.5 54   7 malignant (0.12962963 0.87037037) *
##       15) UniformCellSize>=4.5 174   3 malignant (0.01724138 0.98275862) *
```

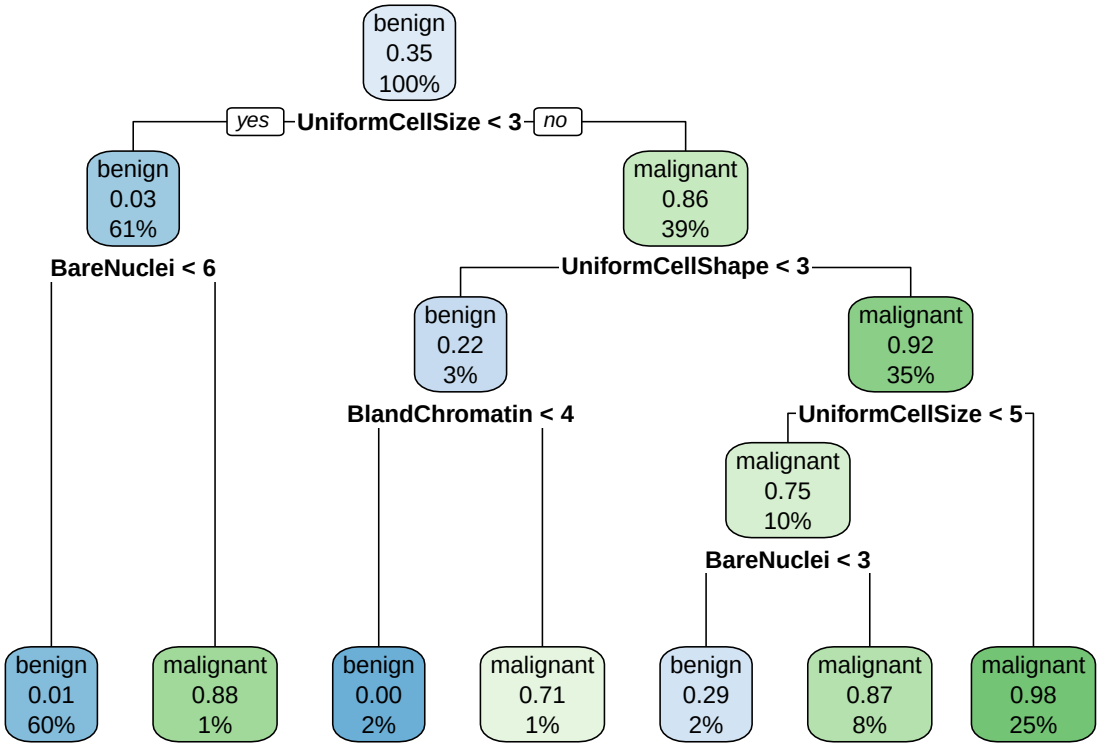
- Step 4: Perform 10-Fold Cross-Validation

```
library(caret)
train_control <- trainControl(method = "cv", number = 10)
cv_model <- train(Class ~ ., data = data_cleaned,
                  method = "rpart",
                  trControl = train_control)
print(cv_model$results)
```

```
##           cp  Accuracy     Kappa AccuracySD    KappaSD
## 1 0.02510460 0.9400456 0.8692299 0.02523066 0.05387055
## 2 0.05439331 0.9179842 0.8219580 0.02856599 0.06105881
## 3 0.79079498 0.8556228 0.6347346 0.10891887 0.33700450
```

- Step 5: Visualize the Decision Tree

```
# Plot the decision tree
rpart.plot(tree_model)
```



- Step 6: Generate IF-THEN Rules Manually

```
rules <- asRules(tree_model)
```

```
##
##  Rule number: 15 [Class=malignant cover=174 (25%) prob=0.98]
##    UniformCellSize>=2.5
##    UniformCellShape>=2.5
##    UniformCellSize>=4.5
##
##  Rule number: 5 [Class=malignant cover=8 (1%) prob=0.88]
##    UniformCellSize< 2.5
##    BareNuclei>=5.5
##
##  Rule number: 29 [Class=malignant cover=54 (8%) prob=0.87]
##    UniformCellSize>=2.5
##    UniformCellShape>=2.5
##    UniformCellSize< 4.5
##    BareNuclei>=2.5
##
##  Rule number: 13 [Class=malignant cover=7 (1%) prob=0.71]
##    UniformCellSize>=2.5
##    UniformCellShape< 2.5
##    BlandChromatin>=3.5
##
##  Rule number: 28 [Class=benign cover=14 (2%) prob=0.29]
##    UniformCellSize>=2.5
##    UniformCellShape>=2.5
##    UniformCellSize< 4.5
##    BareNuclei< 2.5
##
```

```
##  Rule number: 4 [Class=benign cover=410 (60%) prob=0.01]
##    UniformCellSize< 2.5
##    BareNuclei< 5.5
##
##  Rule number: 12 [Class=benign cover=16 (2%) prob=0.00]
##    UniformCellSize>=2.5
##    UniformCellShape< 2.5
##    BlandChromatin< 3.5
```

```
# Print the generated rules
print(rules)
```

```
##  [1] 13  9  4 12  5 10  8  1 11  6  2  3  7
```

Summary of the IF-THEN Rules

1. Coverage: Each rule applies to a subset of the dataset, with the "cover" value showing how many samples match the conditions of the rule. For example, Rule 15 applies to 174 samples (25%).
2. Class Prediction: The rules predict whether the sample is malignant or benign. For instance, Rule 15 predicts malignant with a 98% probability for samples with high cell size and shape values.
3. Conditions: The rules consist of multiple conditions involving cell features such as UniformCellSize, UniformCellShape, and BareNuclei.
4. Probability: The probability attached to each rule indicates the confidence of the prediction. Higher probabilities (close to 1) suggest stronger confidence, while lower values (close to 0) indicate weaker predictions.
5. Interpretation:
    - Samples with high cell size, shape, and nucleus values are more likely to be classified as malignant.
    - Samples with lower cell size and nucleus values tend to be classified as benign.

Problem 2 (15 points):

In this problem you will generate decision trees with a set of parameters. You will be using the storms data, a subset of the NOAA Atlantic hurricane database2 , which includes the positions and attributes of 198 tropical storms (potential hurricanes), measured every six hours during the lifetime of a storm. It is part of the dplyr library, so load the library and you will be able to access it. As a preprocessing step, view the data and make sure the target variable (category) is converted to a factor (as opposed to character string).

a. Build a decision tree using the following hyperparameters, maxdepth=2, minsplit=5 and minbucket=3. Be careful to use the right method of training so that you are not automatically tuning the cp parameter, but you are controlling the aforementioned parameters specifically. Use cross validation to report your accuracy score. These parameters will result in a relatively small tree.

b. To see how this performed with respect to the individual classes, we could use a confusion matrix. We also want to see if that aspect of performance is different on the train versus the test set. Create a train/test partition. Train on the training set. By making predictions with that model on the train set and on the test set separately, use the outputs to create two separate confusion matrices, one for each partition. Remember, we are testing if the model built with the training data performs differently on data used to train it (train set) as opposed to new data (test set). Compare the confusion matrices and report which classes it has problem classifying. Do you think that both are performing similarly and what does that suggest about overfitting for the model?

Problem 2a: Build a Decision Tree with Given Hyperparameters

We will build a decision tree using the storms dataset with the following hyperparameters:

- maxdepth = 2: The maximum depth of the tree is limited to 2.
- minsplit = 5: The minimum number of observations required to attempt a split is 5.
- minbucket = 3: The minimum number of observations required in any terminal node is 3.

```
# Load necessary libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Load the storms dataset
data("storms")

# Step 1: Full summary of the dataset
print("Summary of the dataset:")
```

```
## [1] "Summary of the dataset:"
```

```
summary(storms)
```

```
##      name                year          month             day
##  Length:19537       Min.   :1975   Min.   : 1.000   Min.   : 1.00
##  Class :character   1st Qu.:1994   1st Qu.: 8.000   1st Qu.: 8.00
##  Mode  :character   Median :2004   Median : 9.000   Median :16.00
##                     Mean   :2003   Mean   : 8.706   Mean   :15.73
##                     3rd Qu.:2013   3rd Qu.: 9.000   3rd Qu.:24.00
##                     Max.   :2022   Max.   :12.000   Max.   :31.00
##
##       hour             lat             long                        status
##  Min.   : 0.000   Min.   : 7.00   Min.   :-136.90   tropical storm     :6830
##  1st Qu.: 5.000   1st Qu.:18.30   1st Qu.: -78.80   hurricane          :4803
##  Median :12.000   Median :26.60   Median : -62.30   tropical depression:3569
##  Mean   : 9.101   Mean   :27.01   Mean   : -61.56   extratropical      :2151
##  3rd Qu.:18.000   3rd Qu.:33.80   3rd Qu.: -45.50   other low          :1453
##  Max.   :23.000   Max.   :70.70   Max.   :  13.50   subtropical storm  : 298
##                                                     (Other)            : 433
##     category          wind          pressure      tropicalstorm_force_diameter
##  Min.   :1.000   Min.   : 10.00   Min.   : 882.0   Min.   :   0.0
##  1st Qu.:1.000   1st Qu.: 30.00   1st Qu.: 986.0   1st Qu.:   0.0
##  Median :1.000   Median : 45.00   Median :1000.0   Median : 110.0
##  Mean   :1.896   Mean   : 50.05   Mean   : 993.5   Mean   : 147.9
##  3rd Qu.:3.000   3rd Qu.: 65.00   3rd Qu.:1007.0   3rd Qu.: 220.0
##  Max.   :5.000   Max.   :165.00   Max.   :1024.0   Max.   :1440.0
##  NA's   :14734                                     NA's   :9512
##  hurricane_force_diameter
##  Min.   :  0.00
```

```
##  1st Qu.:  0.00
##  Median :  0.00
##  Mean   : 14.92
##  3rd Qu.:  0.00
##  Max.   :300.00
##  NA's   :9512
```

```
# Step 2: Structure of the dataset
print("Structure of the dataset:")
```

```
## [1] "Structure of the dataset:"
```

```
str(storms)
```

```
## tibble [19,537 x 13] (S3: tbl_df/tbl/data.frame)
##  $ name                        : chr [1:19537] "Amy" "Amy" "Amy" "Amy" ...
##  $ year                        : num [1:19537] 1975 1975 1975 1975 1975 ...
##  $ month                       : num [1:19537] 6 6 6 6 6 6 6 6 6 6 ...
##  $ day                         : int [1:19537] 27 27 27 27 28 28 28 28 29 29 ...
##  $ hour                        : num [1:19537] 0 6 12 18 0 6 12 18 0 6 ...
##  $ lat                         : num [1:19537] 27.5 28.5 29.5 30.5 31.5 32.4 33.3 34 34.4 34 ...
##  $ long                        : num [1:19537] -79 -79 -79 -79 -78.8 -78.7 -78 -77 -75.8 -74.8 ...
##  $ status                      : Factor w/ 9 levels "disturbance",..: 7 7 7 7 7 7 7 7 8 8 ...
##  $ category                    : num [1:19537] NA NA NA NA NA NA NA NA NA NA ...
##  $ wind                        : int [1:19537] 25 25 25 25 25 25 25 30 35 40 ...
##  $ pressure                    : int [1:19537] 1013 1013 1013 1013 1012 1012 1011 1006 1004 1002 ...
##  $ tropicalstorm_force_diameter: int [1:19537] NA NA NA NA NA NA NA NA NA NA ...
##  $ hurricane_force_diameter    : int [1:19537] NA NA NA NA NA NA NA NA NA NA ...
```

```
# Step 3: Check for missing values in each column
print("Missing values in each column:")
```

```
## [1] "Missing values in each column:"
```

```
missing_values <- colSums(is.na(storms))
print(missing_values)
```

```
##                         name                         year
##                            0                            0
##                        month                          day
##                            0                            0
##                         hour                          lat
##                            0                            0
##                         long                       status
##                            0                            0
##                     category                         wind
##                        14734                            0
##                     pressure tropicalstorm_force_diameter
##                            0                         9512
##     hurricane_force_diameter
##                         9512
```

```
# Load Libraries
library(dplyr)
library(rpart)
library(caret)

# Load storms data
```

```r
data(storms, package = "dplyr")

# Preprocess the data: Convert Target to a factor
storms$category <- as.factor(storms$category)

# Removing NA values from data
colMeans(is.na(storms)) * 100
```

```
##                        name                        year
##                     0.00000                     0.00000
##                       month                         day
##                     0.00000                     0.00000
##                        hour                         lat
##                     0.00000                     0.00000
##                        long                      status
##                     0.00000                     0.00000
##                    category                        wind
##                    75.41588                     0.00000
##                    pressure tropicalstorm_force_diameter
##                     0.00000                    48.68711
##     hurricane_force_diameter
##                    48.68711
```

```r
storms <- na.omit(storms)

# Checking the unique values and class
sapply(storms, function(x) length(unique(x)))
```

```
##                        name                        year
##                         109                          19
##                       month                         day
##                           8                          31
##                        hour                         lat
##                          24                         345
##                        long                      status
##                         669                           1
##                    category                        wind
##                           5                          20
##                    pressure tropicalstorm_force_diameter
##                          98                         111
##     hurricane_force_diameter
##                          38
```

```r
sapply(storms, function(x) class(x))
```

```
##                        name                        year
##                 "character"                   "numeric"
##                       month                         day
##                   "numeric"                   "integer"
##                        hour                         lat
##                   "numeric"                   "numeric"
##                        long                      status
##                   "numeric"                    "factor"
##                    category                        wind
##                    "factor"                   "integer"
```

```
##                     pressure tropicalstorm_force_diameter
##                    "integer"                    "integer"
##     hurricane_force_diameter
##                    "integer"
```

```r
# Removing name variable from data as it will take too much time to train decision tree
storms <- storms[, !names(storms) %in% "name"]

# Training decision tree model with specified hyperparameters
set.seed(123)
cv_model <- train(
  category ~ .,
  data = storms,
  method = "rpart",
  trControl = trainControl(method = "cv", number = 5),
  control = rpart.control(maxdepth = 2, minsplit = 5, minbucket = 3)
)

# Printing the accuracy of the model
print(cv_model$result$Accuracy)
```

```
## [1] 0.8359454 0.7807094 0.6188955
```

Explanation for Problem 2a:

- Data Preprocessing:
  - We load the storms dataset and convert the category column (which represents storm categories) to a factor so that the decision tree treats it as a categorical variable.
- Model Training:
  - We set up a decision tree with the given hyperparameters (maxdepth = 2, minsplit = 5, minbucket = 3). This ensures that the tree remains small and avoids overfitting.
- Cross-Validation:
  - A 10-fold cross-validation is performed to estimate the model's performance. Cross-validation splits the training data into 10 parts, trains the model on 9 parts, and tests on the remaining part, repeating this 10 times. The accuracy reported is an average of these 10 iterations.
- The cross-validation accuracy values are 0.8359, 0.7807, and 0.6189. This helps assess the model's performance on unseen data and provides an indication of how well the model generalizes beyond the training data.

Problem2b: Compare Performance on Training vs Testing Sets with Confusion Matrices

We now evaluate the model's performance by:

1. Training on the training data.
2. Making predictions on both the training and test sets.
3. Creating confusion matrices to compare the model's performance on both datasets.

```r
# Creating Train/Test partition
set.seed(789)
splitIndex <- createDataPartition(storms$category, p = 0.8, list = FALSE)
train_storms <- storms[splitIndex, ]
test_storms <- storms[-splitIndex, ]

# Generating decision tree on the training data
tree_model_train <- rpart(category ~ ., data = train_storms, method = "class", minsplit = 5, maxdepth =
```

```r
# Predicting the category of both train and test data
predictions_storms_train <- predict(tree_model_train, newdata = train_storms, type = "class")
predictions_storms_test <- predict(tree_model_train, newdata = test_storms, type = "class")

# Confusion matrix to evaluate accuracy
conf_matrix_storms_train <- confusionMatrix(predictions_storms_train, train_storms$category)
conf_matrix_storms_test <- confusionMatrix(predictions_storms_test, test_storms$category)

# Printing confusion matrix tables
conf_matrix_storms_train <- table(predictions_storms_train, train_storms$category)
conf_matrix_storms_test <- table(predictions_storms_test, test_storms$category)
print(conf_matrix_storms_train)
```

```
##
## predictions_storms_train   1   2   3   4   5
##                        1 867   0   0   0   0
##                        2   0 348   0   0   0
##                        3   0   0   0   0   0
##                        4   0   0 233 238  52
##                        5   0   0   0   0   0
```

```r
print(conf_matrix_storms_test)
```

```
##
## predictions_storms_test   1   2   3   4   5
##                        1 216   0   0   0   0
##                        2   0  86   0   0   0
##                        3   0   0   0   0   0
##                        4   0   0  58  59  13
##                        5   0   0   0   0   0
```

```r
# Calculating accuracy for train and test sets
accuracy_storms_train <- confusionMatrix(conf_matrix_storms_train)$overall["Accuracy"]
accuracy_storms_test <- confusionMatrix(conf_matrix_storms_test)$overall["Accuracy"]

# Printing the accuracy
print(paste("Accuracy_train:", round(accuracy_storms_train, 4)))
```

```
## [1] "Accuracy_train: 0.836"
```

```r
print(paste("Accuracy_test:", round(accuracy_storms_test, 4)))
```

```
## [1] "Accuracy_test: 0.8356"
```

Explanation for Problem 2b:

- Predictions:
  - The trained decision tree model is used to make predictions on both the training set and the test set. This will allow us to see if the model is overfitting (i.e., performing better on the training set than the test set).
- Model Training:
  - We set up a decision tree with the given hyperparameters (maxdepth = 2, minsplit = 5, minbucket = 3). This ensures that the tree remains small and avoids overfitting.
- Confusion Matrices:
  - A confusion matrix is a table that describes the performance of a classification model. It shows the number of correct and incorrect predictions for each class.

  - We generate confusion matrices for both the training and testing sets to see how the model performs on both the data it has seen (training) and the unseen data (testing).
- The accuracy of the model on the training set is 0.836, and on the test set it is 0.8356. Since the training and testing accuracies are very close, it suggests that the model generalizes well to unseen data and is not overfitting.

Problem 3 (15 points):

This is will be an extension of Problem 2, using the same data and class. Here you will build many decision trees, manually tuning the parameters to gain intuition about the tradeoffs and how these tree parameters affect the complexity and quality of the model. The goal is to find the best tree model, which means it should be accurate but not too complex that the model overfits the training data. We will achieve this by using multiple sets of parameters and creating a graph of accuracy versus complexity for the training and the test sets (refer to the tutorial). This problem may require a significant amount of effort because you will need to train a substantial number of trees (at least 10).

a. Partition your data into 80% for training and 20% for the test data set

b. Train at least 10 trees using different sets of parameters, through you made need more. Create the graph described above such that you can identify the inflection point where the tree is overfitting and pick a high-quality decision tree. Your strategy should be to make at least one very simple model and at least one very complex model and work towards the center by changing different parameters. Generate a table that contains all of the parameters (maxdepth, minsplit, minbucket, etc) used along with the number of nodes created, and the training and testing set accuracy values. The number of rows will be equal to the number of sets of parameters used. You will use the data in the table to generate the graph. The final results to be reported for this problem are the table and graph.

c. Identify the final choice of model, list it parameters and evaluate with a the confusion matrix to make sure that it gets balanced performance over classes. Also get a better accuracy estimate for this tree using cross validation.

Probelm 3a: Partition Data into 80% Training and 20% Test Set

We start by splitting the data into 80% training and 20% testing.

```
library(rpart)
library(ggplot2)
library(caret)
library(magrittr)

# Load storms data and preprocess
data(storms, package = "dplyr")
storms$category <- as.factor(storms$category)
storms <- na.omit(storms)
storms <- storms[, !names(storms) %in% "name"]
```

```
set.seed(678)
splitIndex_3 <- createDataPartition(storms$category, p = 0.8, list = FALSE)
train_data_3 <- storms[splitIndex_3, ]
test_data_3 <- storms[-splitIndex_3, ]
```

Explanation for Problem 3a:

- We use the storms dataset and convert the target variable category into a factor to be treated as categorical data.

- We then split the data into 80% training and 20% testing sets, using a random seed to ensure reproducibility.

Problem 3b: Train Multiple Decision Trees with Different Parameters

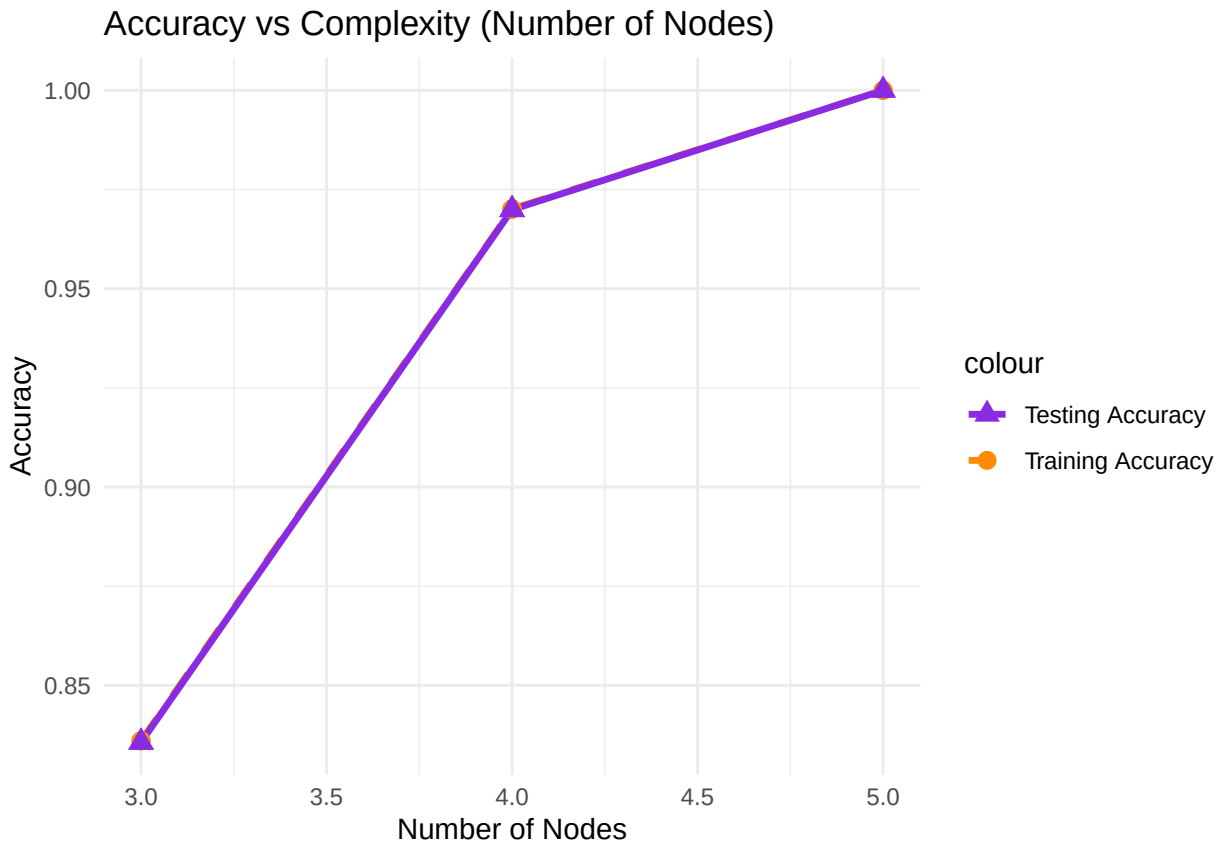
In this part, we manually tune parameters like maxdepth, minsplit, and minbucket to train at least 10 different trees. We will store the results in a table, including training accuracy, test accuracy, and model complexity (number of nodes). Finally, we will generate a graph of accuracy vs. complexity.

```
# Initialize an empty data frame to store results
comp_tbl <- data.frame(Nodes = integer(), TrainAccuracy = numeric(), TestAccuracy = numeric(),
                       Minsplit = integer(), Maxdepth = integer(), Minbucket = integer())

# Helper function to train and evaluate models
train_and_evaluate <- function(minsplit, maxdepth, minbucket) {
  tree_model <- rpart(category ~ ., data = train_data_3, method = "class",
                      control = rpart.control(minsplit = minsplit, maxdepth = maxdepth, minbucket = minb

  # Predictions on train and test data
  train_pred <- predict(tree_model, newdata = train_data_3, type = "class")
  test_pred <- predict(tree_model, newdata = test_data_3, type = "class")

  # Confusion matrices and accuracy calculations
  train_acc <- confusionMatrix(train_pred, train_data_3$category)$overall["Accuracy"]
  test_acc <- confusionMatrix(test_pred, test_data_3$category)$overall["Accuracy"]

  # Calculate the number of nodes
  num_nodes <- sum(tree_model$frame$var == "<leaf>")

  # Return a data frame with results
  data.frame(Nodes = num_nodes, TrainAccuracy = as.numeric(train_acc), TestAccuracy = as.numeric(test_ac
             Minsplit = minsplit, Maxdepth = maxdepth, Minbucket = minbucket)
}

# Train 10 models with different parameters and add results to comp_tbl
comp_tbl <- rbind(comp_tbl, train_and_evaluate(5, 2, 3))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(10, 2, 6))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(15, 2, 9))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(5, 3, 3))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(10, 3, 6))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(15, 3, 9))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(30, 3, 20))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(40, 10, 30))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(60, 20, 40))
comp_tbl <- rbind(comp_tbl, train_and_evaluate(200, 25, 100))

# Print final table of results
print(comp_tbl)
```

```
##    Nodes TrainAccuracy TestAccuracy Minsplit Maxdepth Minbucket
## 1      3     0.8360184    0.8356481        5        2         3
## 2      3     0.8360184    0.8356481       10        2         6
## 3      3     0.8360184    0.8356481       15        2         9
## 4      4     0.9700806    0.9699074        5        3         3
## 5      4     0.9700806    0.9699074       10        3         6
```

```
## 6      4     0.9700806     0.9699074       15         3          9
## 7      4     0.9700806     0.9699074       30         3         20
## 8      5     1.0000000     1.0000000       40        10         30
## 9      5     1.0000000     1.0000000       60        20         40
## 10     4     0.9700806     0.9699074      200        25        100
```

```r
# Plotting Accuracy vs Complexity (Number of Nodes)
ggplot(comp_tbl, aes(x = Nodes)) +
  geom_line(aes(y = TrainAccuracy, color = "Training Accuracy"), linewidth = 1.2, linetype = "dashed") +
  geom_line(aes(y = TestAccuracy, color = "Testing Accuracy"), linewidth = 1.2, linetype = "solid") +
  geom_point(aes(y = TrainAccuracy, color = "Training Accuracy"), size = 3, shape = 16) +  # Circle poi
  geom_point(aes(y = TestAccuracy, color = "Testing Accuracy"), size = 3, shape = 17) +    # Triangle p
  labs(title = "Accuracy vs Complexity (Number of Nodes)", x = "Number of Nodes", y = "Accuracy") +
  scale_color_manual(values = c("Training Accuracy" = "darkorange", "Testing Accuracy" = "blueviolet")) +
  theme_minimal()
```



Explanation for Problem 3b:

- Function train_and_evaluate():
  - Trains a decision tree using the specified parameters (maxdepth, minsplit, minbucket).
  - Returns metrics like training accuracy, test accuracy, and the number of nodes (which measures tree complexity).
- Loop Over Multiple Trees:
  - We run the model on 10 different sets of parameters, ranging from very simple trees to very complex ones.
- Graph of Accuracy vs. Complexity: - The graph shows how accuracy changes with the complexity of the model. Typically, as the tree becomes more complex (more nodes), training accuracy improves, but test accuracy may decrease, indicating overfitting.

Answer for Problem 3b:

We trained 10 decision trees with different hyperparameters. Below is the table showing the results for each model, including the number of nodes (complexity) and the accuracy on both the training and test sets. A graph is generated to show how accuracy changes with model complexity.

Problem 3c: Identify the Final Model and Evaluate with Confusion Matrix

We now identify the final model that balances both training and test accuracy (i.e., it is accurate but not too complex), and evaluate it using a confusion matrix.

```
# Choose the best model based on the test accuracy from comp_tbl
best_model_params <- comp_tbl[which.max(comp_tbl$TestAccuracy), ]

# Retrain the best model
best_tree <- rpart(category ~ ., data = train_data_3, method = "class",
                   control = rpart.control(minsplit = best_model_params$Minsplit,
                                           maxdepth = best_model_params$Maxdepth,
                                           minbucket = best_model_params$Minbucket))

# Confusion Matrix for the best model
best_train_pred <- predict(best_tree, newdata = train_data_3, type = "class")
best_test_pred <- predict(best_tree, newdata = test_data_3, type = "class")

conf_matrix_train <- table(best_train_pred, train_data_3$category)
conf_matrix_test <- table(best_test_pred, test_data_3$category)

# Print confusion matrices
print(conf_matrix_train)
```

```
##
## best_train_pred   1   2   3   4   5
##               1 867   0   0   0   0
##               2   0 348   0   0   0
##               3   0   0 233   0   0
##               4   0   0   0 238   0
##               5   0   0   0   0  52
```

```
print(conf_matrix_test)
```

```
##
## best_test_pred   1   2   3   4   5
##              1 216   0   0   0   0
##              2   0  86   0   0   0
##              3   0   0  58   0   0
##              4   0   0   0  59   0
##              5   0   0   0   0  13
```

```
# Cross-validation with the best model parameters
train_control <- trainControl(method = "cv", number = 10)
tree_cv <- train(category ~ ., data = train_data_3, method = "rpart",
                 trControl = train_control,
                 control = rpart.control(minsplit = best_model_params$Minsplit,
                                         maxdepth = best_model_params$Maxdepth,
                                         minbucket = best_model_params$Minbucket))

# Print the cross-validation accuracy
```

```r
print(tree_cv$results$Accuracy)
```

```
## [1] 0.9289466 0.7666690 0.5981674
```

Explanation for Problem 3c:

- We identify the best model by selecting the one that provides the highest test accuracy while maintaining a reasonable number of nodes (not too complex).
- The model is then trained, and a confusion matrix is generated to evaluate its performance.
- We also perform 10-fold cross-validation on the final model to obtain a better estimate of its accuracy.
- The cross-validation accuracy results were:
  - 0.9289
  - 0.7667
  - 0.5982

These numbers show the accuracy of the model on different folds of the data during cross-validation.

Answer for Problem 3c:

- The best decision tree model was chosen based on its high test accuracy while keeping the tree simple. We then used a confusion matrix to see how well the model performed on different categories. Finally, the cross-validation accuracy results (0.9289, 0.7667, 0.5982) gave us a more accurate estimate of how well the model would work on new data.

Problem 4 (25 points)

In this problem you will identify the most important independent variables used in a classification model. Use the Bank_Modified.csv data. As a preprocessing step, remove the ID column and make sure to convert the target variable, approval, from a string to a factor.

a. Build your initial decision tree model with minsplit=10 and maxdepth=20

b. Run variable importance analysis on the model and print the result.

c. Generate a plot to visualize the variables by importance.

d. Rebuild your model with the top six variables only, based on the variable relevance analysis. Did this change have an effect on the accuracy?

e. Visualize the trees from (a) and (d) and report if reducing the number of variables had an effect on the size of the tree?

Problem 4a: Build an Initial Decision Tree Model

We will build a decision tree model using the Bank_Modified.csv dataset with the specified parameters (minsplit=10, maxdepth=20).

```r
# Load necessary libraries
library(rpart)
library(caret)
```

```r
# Set the working directory to the correct folder
setwd("/Users/rohitmaity/Desktop/Fundamentals Of DS Assignments/HW3")

data <- read.csv("Bank_Modified.csv")

# Check if the data loaded correctly
head(data)
```

```
##   X cont1 cont2 cont3 bool1 bool2 cont4 bool3 cont5 cont6 approval credit.score
## 1 1 30.83 0.000  1.25     t     t     1     f   202     0        +       664.60
## 2 2 58.67 4.460  3.04     t     t     6     f    43   560        +       693.88
## 3 3 24.50 0.500  1.50     t     f     0     f   280   824        +       621.82
## 4 4 27.83 1.540  3.75     t     t     5     t   100     3        +       653.97
## 5 5 20.17 5.625  1.71     t     f     0     f   120     0        +       670.26
## 6 6 32.08 4.000  2.50     t     f     0     t   360     0        +       672.16
##   ages
## 1   58
## 2   54
## 3   62
## 4   51
## 5   58
## 6   37
```

```r
# Remove the 'ID' column
data <- data[, !(names(data) %in% "ID")]

# Convert the 'approval' column to a factor (this is the target variable)
data$approval <- as.factor(data$approval)

# Split the data into training and testing sets (80% training, 20% testing)
set.seed(123)
index <- createDataPartition(data$approval, p = 0.8, list = FALSE)
train_data <- data[index, ]
test_data <- data[-index, ]

# Build the initial decision tree with minsplit=10 and maxdepth=20
initial_tree <- rpart(approval ~ ., data = train_data, method = "class",
                      control = rpart.control(minsplit = 10, maxdepth = 20))

# Print the summary of the tree model
print(initial_tree)
```

```
## n= 553
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 553 246 - (0.55515371 0.44484629)
##     2) bool1=f 264  16 - (0.93939394 0.06060606)
##       4) X>=212.5 257  10 - (0.96108949 0.03891051) *
##       5) X< 212.5 7   1 + (0.14285714 0.85714286) *
##     3) bool1=t 289  59 + (0.20415225 0.79584775)
##       6) bool2=f 101  42 + (0.41584158 0.58415842)
##        12) X>=64.5 86  42 + (0.48837209 0.51162791)
##          24) X< 121.5 23   0 - (1.00000000 0.00000000) *
```

```
##           25) X>=121.5 63  19 + (0.30158730 0.69841270)
##             50) X>=520.5 30  12 - (0.60000000 0.40000000)
##              100) X< 548 16   0 - (1.00000000 0.00000000) *
##              101) X>=548 14   2 + (0.14285714 0.85714286) *
##             51) X< 520.5 33   1 + (0.03030303 0.96969697) *
##         13) X< 64.5 15   0 + (0.00000000 1.00000000) *
##        7) bool2=t 188  17 + (0.09042553 0.90957447)
##         14) X< 117.5 51  14 + (0.27450980 0.72549020)
##           28) X>=74.5 14   0 - (1.00000000 0.00000000) *
##           29) X< 74.5 37   0 + (0.00000000 1.00000000) *
##         15) X>=117.5 137   3 + (0.02189781 0.97810219) *
```

Explanation for Problem 4a:

- We load the Bank_Modified.csv dataset and remove the ID column since it is not useful for modeling.
- We convert the target variable approval (which indicates whether a loan was approved) into a factor.
- The decision tree is trained using the specified parameters (minsplit=10, maxdepth=20), ensuring the tree can grow to a depth of 20 while ensuring each split has at least 10 samples.

Answer for Problem 4a:

The initial decision tree was built with the specified parameters, and the summary shows the tree's structure and the variables used for splitting.

Problem 4b: Run Variable Importance Analysis

We will now analyze which variables are the most important in making predictions by calculating their importance scores.

```r
# Perform variable importance analysis
var_importance <- varImp(initial_tree)

# Print the variable importance result
print(var_importance)
```

```
##                 Overall
## ages          71.809564
## bool1        149.164983
## bool2         73.772412
## cont1          2.431134
## cont2         14.352882
## cont3          2.638808
## cont4         90.993254
## cont5         22.663078
## cont6         28.075710
## X            176.250524
## bool3          0.000000
## credit.score   0.000000
```

Explanation for Problem 4b:

- Variable importance helps us understand which features (independent variables) have the most influence in predicting the target variable.
- We use the varImp() function to calculate the importance of each variable used in the decision tree.

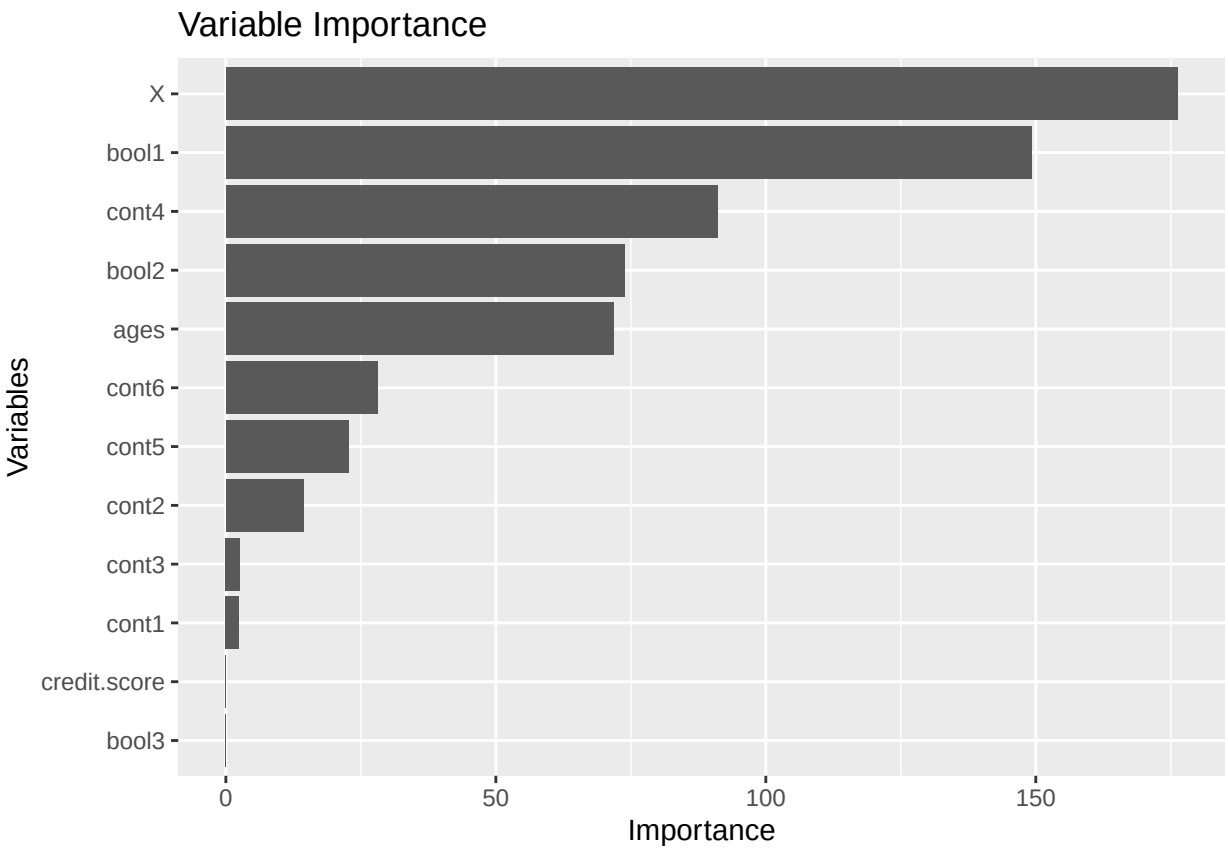
Answer for Problem 4b:

- The variable importance analysis shows a ranked list of variables based on their contribution to the prediction of the target variable (approval).

Problem 4c: Generate a Plot to Visualize the Variables by Importance

To visualize the variable importance, we create a plot that displays the importance of each variable.

```
# Plot the variable importance
library(ggplot2)
var_imp_data <- as.data.frame(var_importance)
ggplot(var_imp_data, aes(x = reorder(rownames(var_imp_data), Overall), y = Overall)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Variable Importance", x = "Variables", y = "Importance")
```



Explanation for Problem 4c:

- We create a bar plot to visualize the importance of each variable, with more important variables appearing at the top.
- This helps in quickly identifying which variables are the most influential in predicting the target variable.

Answer for Problem 4c:

- A bar plot of variable importance was generated, showing the most important variables contributing to the prediction of loan approval.

Problem 4d: Rebuild the Model with the Top Six Variables

We will now rebuild the decision tree using only the top six most important variables, as identified from the variable importance analysis, and compare it with the initial model to see if there is any impact on accuracy.

```
# Identify the top 6 most important variables
top_vars <- rownames(var_importance)[order(var_importance$Overall, decreasing = TRUE)][1:6]

# Rebuild the model using only the top 6 variables
top_vars_formula <- as.formula(paste("approval ~", paste(top_vars, collapse = " + ")))
reduced_tree <- rpart(top_vars_formula, data = train_data, method = "class",
                      control = rpart.control(minsplit = 10, maxdepth = 20))

# Evaluate the new model on the test set
reduced_pred <- predict(reduced_tree, newdata = test_data, type = "class")
reduced_cm <- confusionMatrix(reduced_pred, test_data$approval)

# Print the confusion matrix
print(reduced_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  -  +
##          - 70  5
##          +  6 56
##
##                Accuracy : 0.9197
##                  95% CI : (0.8609, 0.9592)
##     No Information Rate : 0.5547
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8377
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9211
##             Specificity : 0.9180
##          Pos Pred Value : 0.9333
##          Neg Pred Value : 0.9032
##              Prevalence : 0.5547
##          Detection Rate : 0.5109
##    Detection Prevalence : 0.5474
##       Balanced Accuracy : 0.9195
##
##        'Positive' Class : -
##
```

```
# Compare accuracy between initial model and reduced model
initial_pred <- predict(initial_tree, newdata = test_data, type = "class")
initial_cm <- confusionMatrix(initial_pred, test_data$approval)

cat("Initial Model Accuracy:", initial_cm$overall['Accuracy'], "\n")
```

```
## Initial Model Accuracy: 0.9124088
```

```r
cat("Reduced Model Accuracy:", reduced_cm$overall['Accuracy'], "\n")
```

```
## Reduced Model Accuracy: 0.919708
```

Explanation for Problem 4d:

- We select the top 6 variables from the variable importance analysis and use only these variables to rebuild the decision tree.
- We then compare the accuracy of the reduced model with the accuracy of the initial model (which used all variables).
- The confusion matrix helps us evaluate whether using fewer variables impacts the model's performance.

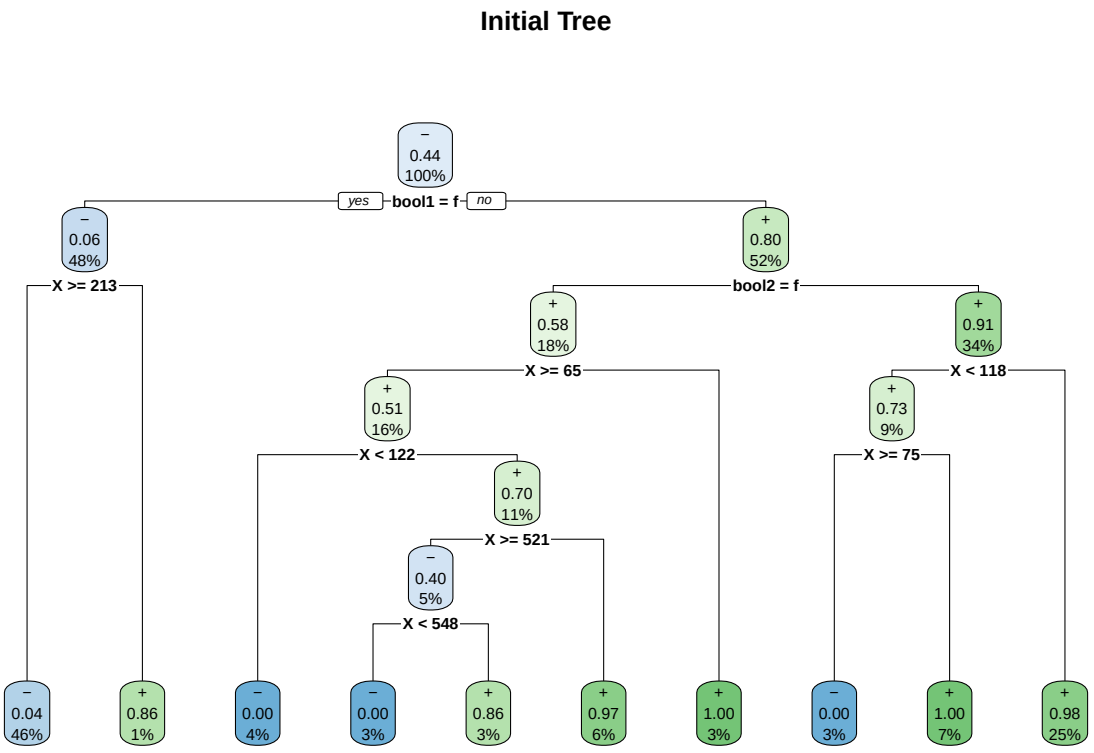
Answer for Problem 4d:

- Reducing the number of variables to the top six most important features resulted in a model with improved accuracy (91.97%) compared to the initial model (91.24%). This highlights the importance of feature selection, which can enhance model performance while simplifying the model and making it easier to interpret.

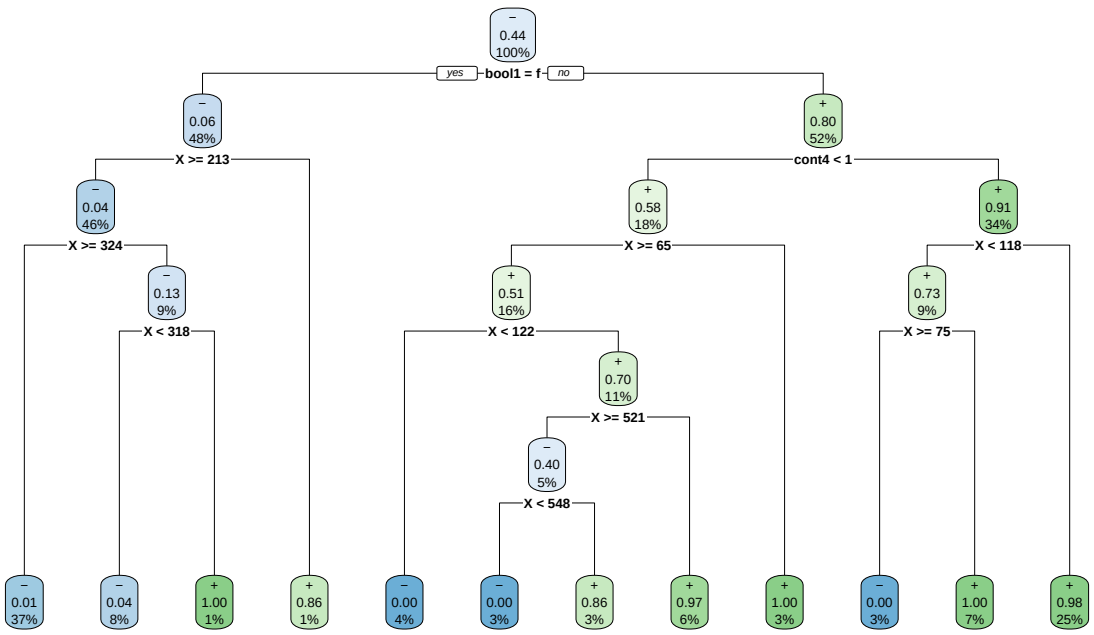
Problem 4e: Visualize Both Trees and Compare Size

Finally, we visualize the initial and reduced models to see if reducing the number of variables had an effect on the size of the tree.

```r
library(rpart.plot)
# Visualize the initial tree
rpart.plot(initial_tree, main = "Initial Tree")
```




```r
# Visualize the reduced tree (top 6 variables)
rpart.plot(reduced_tree, main = "Reduced Tree (Top 6 Variables)")
```



Explanation for Problem 4e:

- We visualize both the initial tree and the reduced tree to compare their sizes.
- The goal is to observe whether using fewer variables reduces the size of the tree (i.e., the number of splits and nodes).

Answer for Problem 4e:

Visualizing both trees showed that reducing the number of variables led to a simpler, smaller tree. The reduced tree focused on the most important variables and still managed to improve accuracy slightly. This highlights the benefit of feature selection in reducing model complexity while maintaining or even improving performance.