IDS572 Assignment2

Yaze Gao, Shixun Jiang, Kritika Raghuwanshi

2023-02-18

Problem 1

Do not use R to answer this question. The following table contains data from an employee database. The database includes the status, department, age range and salary of each employee. This problem asks you to learn a Näıve Bayes classifier for predicting the employee status.

(a) Use your näıve Bayes classifier, predict the status for two instances A={Marketing, 31-35, 46K-50K} and B={Sales, 31-35, 66K-70K}.

A={Marketing, 31-35, 46K-50}

P status=Senior|A)=P(A|status=Senior)*P(status=Senior)/P(A) P status=Junior|A)=P(A|status=Junior)*P(status=Junior)

P status=Senior)=5/11=0.455 P status=Junior)=6/11=0.545

Calculating conditional probabilities P Marketing status=Senior =1/5=0.2 P Marketing status=Junior =1/6=0.167

P 31-35 status=Senior =1/5=0.2 P 31-35 status=Junior =2/6=0.333

P 46K-50K status=Senior =2/5=0.4 P 46K-50K status=Junior =1/6=0.167

P(A|status=Senior)=0.2* 0.2* 0.4=0.016 P(A|status=Junior)=0.167* 0.333* 0.167=0.009

P(A|status=Senior)* P status=Senior)=0.016* 0.455=0.00782 P(A|status=Junior)* P(status=Junior)=0.009* 0.545=0.0049

```
0.00782  >0.0049
```

Thus for A={Marketing, 31-35, 46K-50}, according to the naive Bayesian classifier, status=Senior.

B={Sales, 31-35, 66K-70K}

P status=Senior|B)=P(B|status=Senior)*P(status=Senior)/P(B) P status=Junior|B)=P(B|status=Junior)*P(status=Junior)/

P status=Senior)=5/11=0.455 P status=Junior)=6/11=0.545

Calculating conditional probabilities P Sales status=Senior =1/5=0.2 P Sales status=Junior =2/6=0.333

P 31-35 status=Senior =1/5=0.2 P 31-35 status=Junior =2/6=0.333

P 66K-70K status=Senior =2/5=0.4 P 66K-70K status=Junior =1/6=0.167

P(B|status=Senior)=0.2* 0.2* 0.4=0.016 P(B|status=Junior)=0.333* 0.333* 0.167=0.019

P(B|status=Senior)* P status=Senior)=0.016* 0.455=0.00782 P(B|status=Junior)* P(status=Junior)=0.019* 0.545=0.010

```
0.00782   <0.01
```

Thus for B={Sales, 31-35, 66K-70K}, according to the naive Bayesian classifier, status=Junior.

(b) Suppose we add another feature called “SalaryDuplicate”, which takes on the same value as “Salary” for all training examples. What are the prediction results for the above two instances, if we train a na¨ıve Bayes classifier on the same dataset with this extra feature? Justify your observations.

ForA P(A|status=Senior)=0.2* 0.2* 0.4* 0.4=0.0064 P(A|status=Junior)=0.167* 0.333* 0.167* 0.167=0.0015

P(A|status=Senior)* P status=Senior)=0.0064* 0.455=0.0029 P(A|status=Junior)* P(status=Junior)=0.0015* 0.545=0.0008

After we add another feature called “SalaryDuplicate”,for A status=senior

For B P(B|status=Senior)=0.2* 0.2* 0.4* 0.4=0.0064 P(B|status=Junior)=0.333* 0.333* 0.167* 0.167=0.003

P(B|status=Senior)* P status=Senior)=0.0064* 0.455=0.0029 P(B|status=Junior)* P(status=Junior)=0.003* 0.545=0.0016

After we add another feature called “SalaryDuplicate”,for B status=senior

I think the calculation needs to be done from new after adding a feature.

Problem 2

(Decision tree in R) One of the challenges in marketing is identifying a set of customers who are most likely to respond to the marketing campaigns. Data science and machine learning systems can help companies to identify such customers. In this question, we will analyze a bank data and build a decision tree model to predict the outcome of the campaign held by the bank, namely whether the client has subscribed to a term deposit. To do so, download the "Bank" data set from Blackboard. This data set contains information about over 41000 observations, including variables about a bank's clients, data related to the previous and current campaigns held by the bank, and social and economic context attributes present at a particular time.

Import data through "Environment-Import Dataset-From Text(base)"

```r
Bank <- read.csv("C:/Bank.csv", sep=";")
View(Bank)
```

(a) Before thinking about modeling, have a look at your data. Try to understand the variables' distributions and their relationships with the target variable. Which variables do you think could be major predictors of the target variable? Also, clean your data appropriately: Are there highly correlated variables? Are there any missing values or outliers? If yes, how do you handle them?

First we need change character variable to factor.

```
fac <- c('y', 'poutcome', 'default', 'month')
Bank[, fac] <- lapply(Bank[, fac],factor)
summary(Bank)
```

```
##       age            job              marital            education
##  Min.   :18.00   Length:45211       Length:45211       Length:45211
##  1st Qu.:33.00   Class :character   Class :character   Class :character
##  Median :39.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   :40.94
##  3rd Qu.:48.00
##  Max.   :95.00
##
##  default        balance          housing              loan
##  no :44396   Min.   : -8019   Length:45211       Length:45211
##  yes:  815   1st Qu.:    72   Class :character   Class :character
##              Median :   448   Mode  :character   Mode  :character
##              Mean   :  1362
##              3rd Qu.:  1428
##              Max.   :102127
##
##    contact               day            month          duration
##  Length:45211       Min.   : 1.00   may    :13766   Min.   :   0.0
##  Class :character   1st Qu.: 8.00   jul    : 6895   1st Qu.: 103.0
##  Mode  :character   Median :16.00   aug    : 6247   Median : 180.0
##                     Mean   :15.81   jun    : 5341   Mean   : 258.2
##                     3rd Qu.:21.00   nov    : 3970   3rd Qu.: 319.0
##                     Max.   :31.00   apr    : 2932   Max.   :4918.0
##                                     (Other): 6060
##     campaign          pdays          previous           poutcome
##  Min.   : 1.000   Min.   : -1.0   Min.   :  0.0000   failure: 4901
##  1st Qu.: 1.000   1st Qu.: -1.0   1st Qu.:  0.0000   other  : 1840
##  Median : 2.000   Median : -1.0   Median :  0.0000   success: 1511
##  Mean   : 2.764   Mean   : 40.2   Mean   :  0.5803   unknown:36959
##  3rd Qu.: 3.000   3rd Qu.: -1.0   3rd Qu.:  0.0000
##  Max.   :63.000   Max.   :871.0   Max.   :275.0000
##
##    y
##  no :39922
##  yes: 5289
##
##
##
##
##
```

Then we checking the missing values. The result shows there is no missing values.

```
sum(is.na(Bank))
```

```
## [1] 0
```

The first variable that we believe is most likely to be associated with whether or not to purchase a product is balance, where customers with more balance have more money at their disposal and therefore are likely to be associated with whether or not to purchase a deposit. The second variable is duration, where the longer the conversation with the bank staff, the more interested the customer is in the product. The third variable is campign, the more contact with the customer and the more opportunities to communicate with the customer, the more likely to impress the customer. The last variable is poutcome. the results of the last marketing campaign can easily influence the customer's trust in the product.

(b) Create a decision tree (using "information" for splits) to its full depth. How many leaves are in this tree?

First using sample() function separate the training data and testing data. We chose 70% training values and 30% testing values.

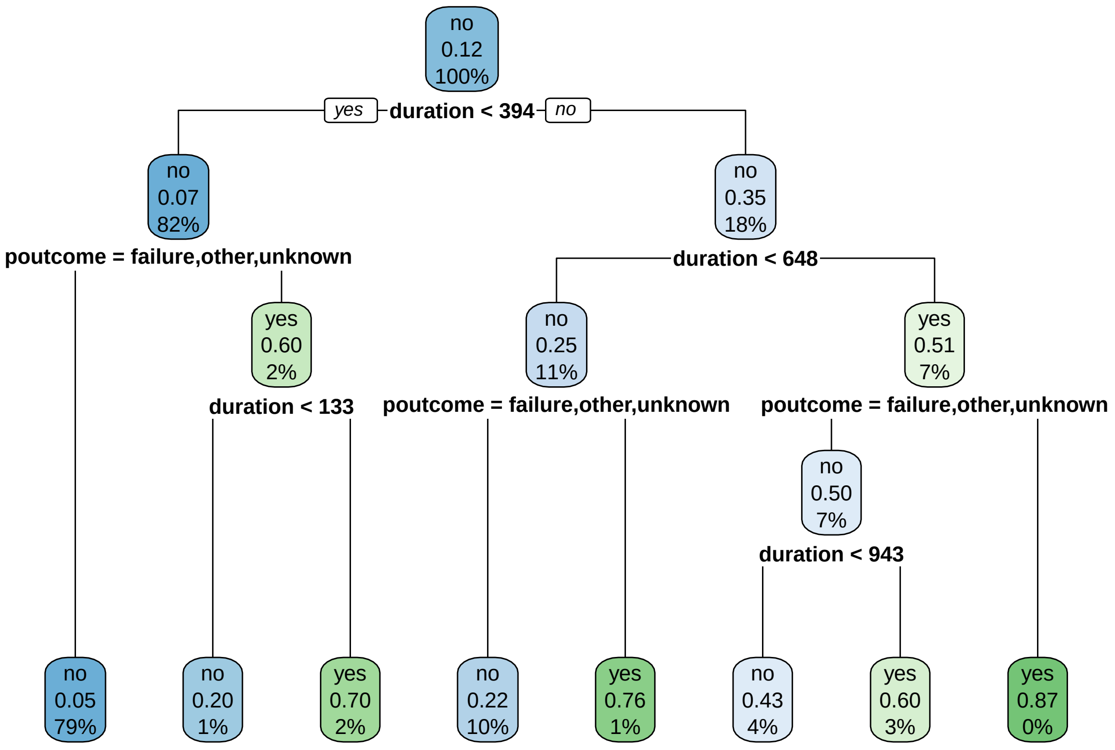
```r
indx<-sample(2,45211,replace = T ,prob = c(0.7,0.3))
train_Bank<-Bank[indx==1,]
test_Bank<-Bank[indx==2,]
```

Then use 'rpart' package to creat decision tree.

```r
library(rpart)
tree_model1<-rpart(formula = y~.,data = train_Bank,
          parms = list(split = "information"))
```

Use 'rpart.plot' package to plot the tree.

```r
library(rpart.plot)
rpart.plot(tree_model1,type = 2)
```



```r
print(tree_model1)
```

```
## n= 31706
```

```
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 31706 3716 no (0.88279821 0.11720179)
##    2) duration< 393.5 25968 1689 no (0.93495841 0.06504159)
##      4) poutcome=failure,other,unknown 25180 1220 no (0.95154885 0.04845115) *
##      5) poutcome=success 788  319 yes (0.40482234 0.59517766)
##       10) duration< 132.5 165   33 no (0.80000000 0.20000000) *
##       11) duration>=132.5 623  187 yes (0.30016051 0.69983949) *
##    3) duration>=393.5 5738 2027 no (0.64674102 0.35325898)
##      6) duration< 647.5 3485  871 no (0.75007174 0.24992826)
##       12) poutcome=failure,other,unknown 3313  740 no (0.77663749 0.22336251) *
##       13) poutcome=success 172   41 yes (0.23837209 0.76162791) *
##      7) duration>=647.5 2253 1097 yes (0.48690635 0.51309365)
##       14) poutcome=failure,other,unknown 2166 1080 no (0.50138504 0.49861496)
##         28) duration< 942.5 1299  564 no (0.56581986 0.43418014) *
##         29) duration>=942.5 867  351 yes (0.40484429 0.59515571) *
##       15) poutcome=success 87   11 yes (0.12643678 0.87356322) *
```

The tree has a total of 13 nodes, of which there are 7 leaf nodes and 5 child nodes.

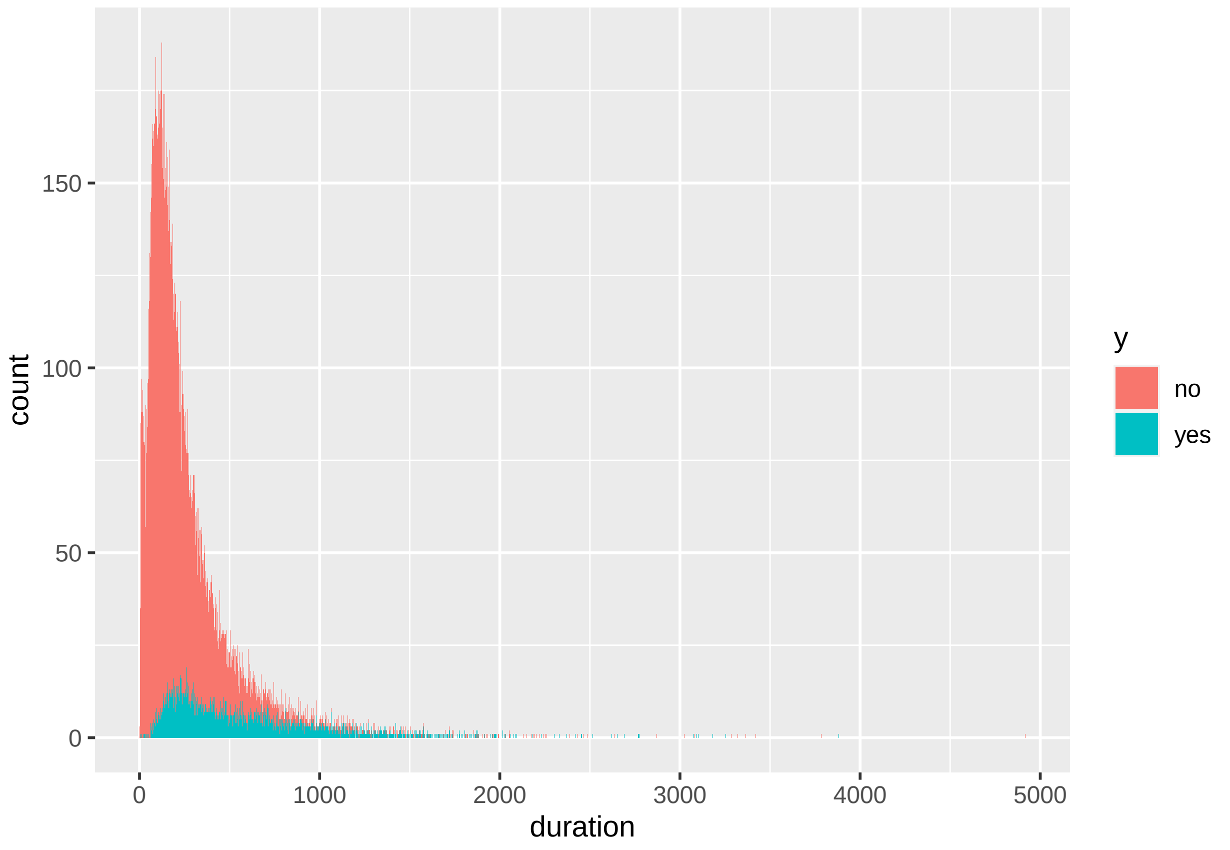
(c) What are the major predictors of diagnosis suggested by your tree? Please justify your reasoning. Do these major predictors are the same as the ones you observed in part (a)?

The main segmentation is based on duration. duration of the last contact, and poutcome. outcome of the previous marketing campaign. this is similar to our prediction in (a), most people who have a short contact time prove that they are not interested in the product, so most of the people with little contact time People are also judged based on the outcome of their last purchase, so a small percentage of people with short contact times end up choosing yes because they have bought the product before.

(d) Give two strong rules describing who will likely subscribe to a term deposit. Please justify your choices.

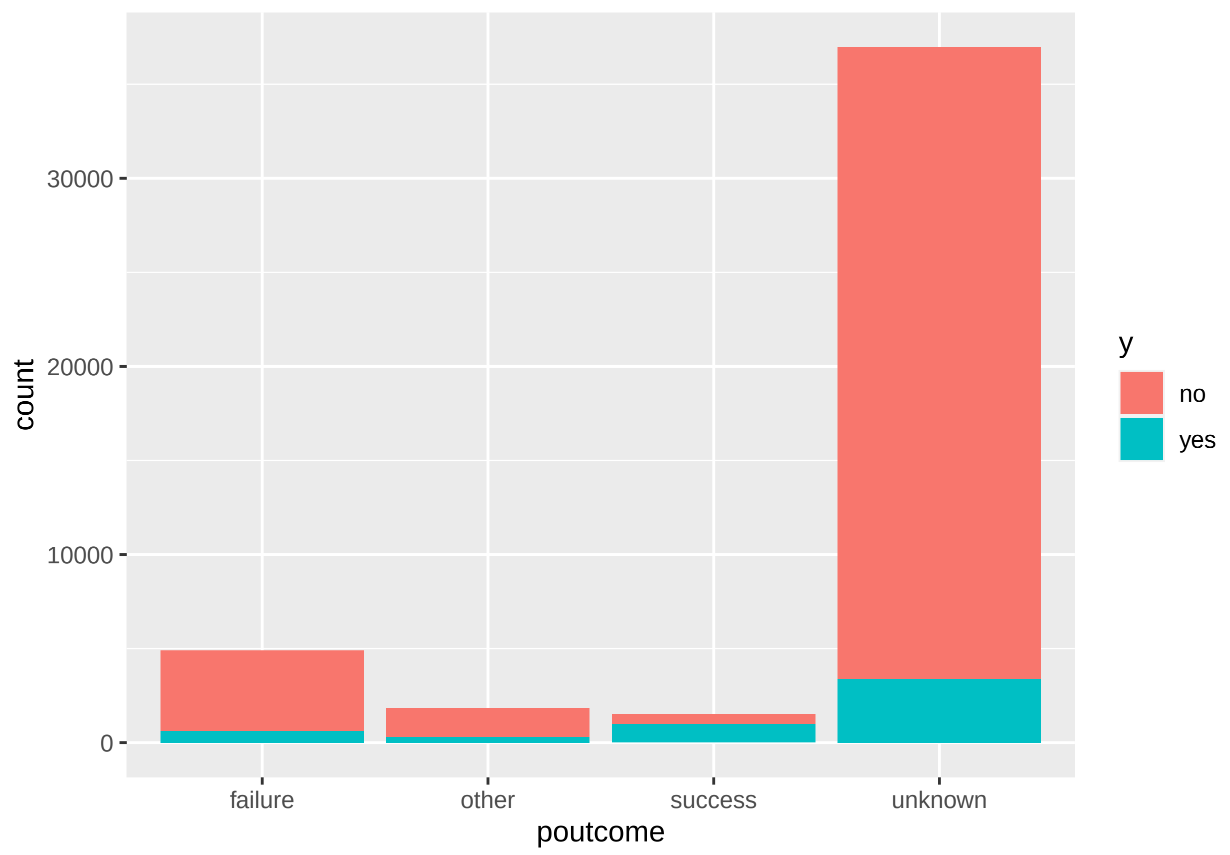
1.People who have been in contact for a long time are more likely to buy deposits.I creat a bar chart to prove it.

```r
library(ggplot2)
ggplot(data = Bank) +
  geom_bar(mapping = aes(x= duration, fill= y))
```



Customers who were successful as a result of the last marketing campaign were more inclined to buy the product.The histogram below shows that the percentage of customers who chose yes for the last influence campaign as a success was much higher than in other cases.

```r
ggplot(data = Bank) +
  geom_bar(mapping = aes(x=  poutcome, fill= y))
```



count
30000
20000
10000
0
failure
other
success
unknown
poutcome
y
no
yes

(e) What is the accuracy of your decision tree model on the training data? What is the accuracy of this model on the test data?

```
tree_pred_prob1 <- predict(tree_model1, train_Bank, type = "prob")
tree_pred_class1 <- predict(tree_model1, train_Bank, type = "class")
```

```
testerror_train <- mean(tree_pred_class1 != train_Bank$y)
print(testerror_train)
```

```
## [1] 0.09925566
```

Error rate of our decision tree model on train data is 0.0994128

```
tree_pred_test1 <- predict(tree_model1, test_Bank, type = "class")
testerror_test1 <- mean(tree_pred_test1 != test_Bank$y)
print(testerror_test1)
```

```
## [1] 0.09944465
```

Error rate of our decision tree model on test data is 0.1004802

```
t1 <- table(tree_pred_test1, test_Bank$y)
acc1 <- sum(diag(t1))/nrow(test_Bank)*100
print(acc1)
```

```
## [1] 90.05553
```

We calculated the accuracy of model one is 89.95198.

(f) Construct the "best possible" decision tree to predict the Y labels. Explain how you construct such a tree and how you evaluate its performance.

Tree model2

```r
tree_model2 <- rpart(y ~ ., train_Bank,
                     parms = list(split = "gini"))
```

```r
tree_pred_prob2 <- predict(tree_model2, train_Bank, type = "prob")
tree_pred_class2 <- predict(tree_model2, train_Bank, type = "class")

testerror_train2 <- mean(tree_pred_class2 != train_Bank$y)
print(testerror_train2)
```

```
## [1] 0.0996026
```

```r
tree_pred_test2 <- predict(tree_model2, test_Bank, type = "class")
testerror_test2 <- mean(tree_pred_test2 != test_Bank$y)
print(testerror_test2)
```

```
## [1] 0.098408
```

```r
t2 <- table(tree_pred_test2, test_Bank$y)
acc2 <- sum(diag(t2))/nrow(test_Bank)*100
print(acc2)
```

```
## [1] 90.1592
```

We calculated the accuracy of model two is 90.00369

Tree model3

```
 tree_model3<- rpart(formula = y~., data = test_Bank,
parms =list(split="information"),
control = rpart.control(minbucket = 3, minsplit = 5))
```

```
tree_pred_prob3 <- predict(tree_model3, train_Bank, type = "prob")
tree_pred_class3 <- predict(tree_model3, train_Bank, type = "class")

testerror_train3 <- mean(tree_pred_class3 != train_Bank$y)
print(testerror_train3)
```

```
## [1] 0.103892
```

```
tree_pred_test3 <- predict(tree_model3, test_Bank, type = "class")
testerror_test3 <- mean(tree_pred_test3 != test_Bank$y)
print(testerror_test3)
```

```
## [1] 0.09959274
```

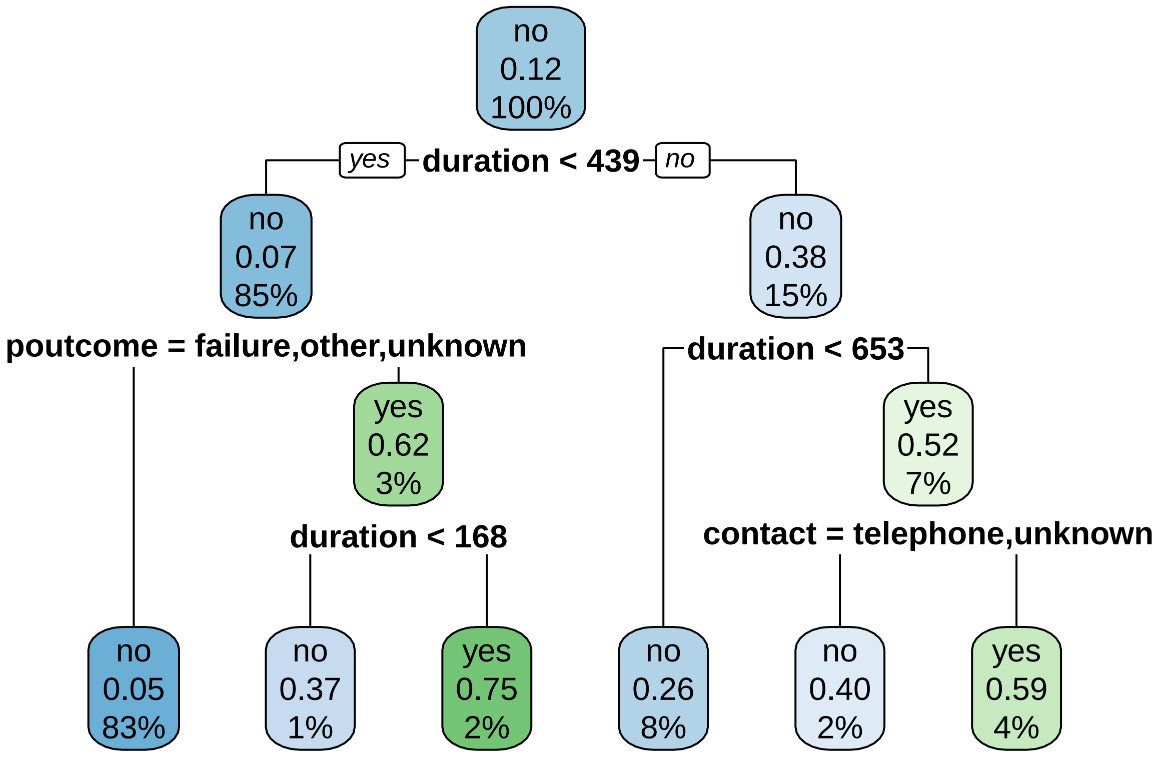
```
t3 <- table(tree_pred_test3, test_Bank$y)
acc3 <- sum(diag(t3))/nrow(test_Bank)*100
print(acc3)
```

```
## [1] 90.04073
```

We calculated the accuracy of model three is 90.43221 After calculating the accuracy rate model 3 has the highest accuracy rate, so I choose model 3 as my best model.

(g) Plot your final decision tree model and write down all decision rules that you will consider for predictions.

```
rpart.plot(tree_model3)
```



A small percentage of users whose last contact was shorter than 467 would simply choose no. A small percentage of users whose last marketing was successful would choose yes, which is 2%. There are 4% of people who have had a successful last marketing campaign who will choose no. Those with a contact time greater than 467 will be divided into two categories. Those with contact time greater than 802 all chose yes. those with contact time between 467 and 802 would make a second choice based on the results of their last campaign. 9% would choose no.

Problem 3

Importing the Data Set

A. In order to do explanatory analysis and construct a decision tree for the Heart data set, I first imported the .csv file into a vector / data frame called heart. After that, I converted the data frame into a tibble for better representation

```
library(readr)
library(tibble)
heart <- read.csv("C:/Heart.csv")
attach(heart)
heart <- as_tibble(heart)
print(heart, n=2)
```

```
## # A tibble: 920 x 16
##      id   age sex   dataset   cp        trest~1  chol fbs   restecg thalch exang
##   <int> <int> <chr> <chr>     <chr>       <int> <int> <lgl> <chr>    <int> <lgl>
## 1     1    63 Male  Cleveland typical ~     145   233 TRUE  lv hyp~    150 FALSE
## 2     2    67 Male  Cleveland asymptom~     160   286 FALSE lv hyp~    108 TRUE
## # ... with 918 more rows, 5 more variables: oldpeak <dbl>, slope <chr>,
## #   ca <int>, thal <chr>, num <int>, and abbreviated variable name 1: trestbps
```

Understanding the Data and Data Manipulation

Apart from the two variables "id" and "dataset", which are not contributing to our overall analysis, this data set has both *numeric* and *non-numeric* variables as follows:

```r
library(knitr)
tbl <- data.frame(
  VariableName = c("age", "sex", "cp", "trestbps", "chol", "fbs", "restecg",
                     "thalch", "exang", "oldpeak", "slope", "ca", "thal",
                    "num"),
  VariableType = c("Numeric", "Non-Numeric", "Non-Numeric", "Numeric"
                   , "Numeric", "Non-Numeric", "Non-Numeric"
                   ,"Numeric", "Non-Numeric", "Numeric"
                   , "Non-Numeric", "Numeric", "Non-Numeric", "Numeric")
)
kable(tbl)
```

VariableName	VariableType
age	Numeric
sex	Non-Numeric
cp	Non-Numeric
trestbps	Numeric
chol	Numeric
fbs	Non-Numeric
restecg	Non-Numeric
thalch	Numeric
exang	Non-Numeric
oldpeak	Numeric
slope	Non-Numeric
ca	Numeric
thal	Non-Numeric
num	Numeric

The **“num”** variable is actually our target variable, which we need to use for classification further. However, since this is numerical, with 5 different values from 0 to 4. We introduced a new *binary* variable called **“target”**, which will have 0 for num = 0 and 1 for num = 1, 2, 3, or 4. While introducing the **“target”** variable, we also give it labels for detection of heart disease. For example, if the value in this variable is 0, we say there’s **“Absence”** of heart disease, whereas if it’s 1, we say there’s **“Presence”** of heart disease. Once we have a new variable called “target”, we need to remove the **“num”** variable from the data set, to keep it to 14 variables. To do this, we simply remove the variable at the 14th index, as shown in the code chunk below:

```r
heart$target <- as.factor(ifelse(num > 0, "Presence", "Absence"))
heart <- heart[,-16]
print.data.frame(head(heart, n=3))
```

```
##   id age  sex  dataset             cp trestbps chol   fbs        restecg
## 1  1  63 Male Cleveland typical angina      145  233  TRUE lv hypertrophy
## 2  2  67 Male Cleveland   asymptomatic      160  286 FALSE lv hypertrophy
## 3  3  67 Male Cleveland   asymptomatic      120  229 FALSE lv hypertrophy
##   thalch exang oldpeak       slope ca              thal   target
## 1    150 FALSE     2.3 downsloping  0      fixed defect  Absence
## 2    108  TRUE     1.5        flat  3            normal Presence
## 3    129  TRUE     2.6        flat  2 reversable defect Presence
```

Some other variables can also be converted to factor for ease, for which we have used the lapply function:

```r
heart[c("sex", "cp", "fbs", "restecg", "exang", "slope", "ca",
"thal")] <- lapply(heart[c("sex", "cp", "fbs", "restecg", "exang"
, "slope", "ca", "thal")], factor)
print(heart, n=2)
```

```
## # A tibble: 920 x 16
##      id   age sex   dataset   cp        trest~1  chol fbs   restecg thalch exang
##   <int> <int> <fct> <chr>     <fct>       <int> <int> <fct> <fct>    <int> <fct>
## 1     1    63 Male  Cleveland typical ~     145   233 TRUE  lv hyp~    150 FALSE
## 2     2    67 Male  Cleveland asymptom~     160   286 FALSE lv hyp~    108 TRUE
## # ... with 918 more rows, 5 more variables: oldpeak <dbl>, slope <fct>,
## #   ca <fct>, thal <fct>, target <fct>, and abbreviated variable name
## #   1: trestbps
```

Cleaning the Data Set

We notice that while the problem statement above suggests that this data set has **303 observations**, we're actually seeing **920 observations** overall, so in order to check why that is the case, we look at the summary of this data set:

```
summary(heart)
```

```
##        id              age             sex          dataset         
##  Min.   :  1.0   Min.   :28.00   Female:194   Length:920        
##  1st Qu.:230.8   1st Qu.:47.00   Male  :726   Class :character  
##  Median :460.5   Median :54.00                Mode  :character  
##  Mean   :460.5   Mean   :53.51                                  
##  3rd Qu.:690.2   3rd Qu.:60.00                                  
##  Max.   :920.0   Max.   :77.00                                  
##                                                                 
##                cp         trestbps          chol          fbs         
##  asymptomatic   :496   Min.   :  0.0   Min.   :  0.0   FALSE:692  
##  atypical angina:174   1st Qu.:120.0   1st Qu.:175.0   TRUE :138  
##  non-anginal    :204   Median :130.0   Median :223.0   NA's : 90  
##  typical angina : 46   Mean   :132.1   Mean   :199.1              
##                        3rd Qu.:140.0   3rd Qu.:268.0              
##                        Max.   :200.0   Max.   :603.0              
##                        NA's   :59      NA's   :30                 
##             restecg        thalch        exang        oldpeak       
##                  :  2   Min.   : 60.0   FALSE:528   Min.   :-2.6000  
##  lv hypertrophy  :188   1st Qu.:120.0   TRUE :337   1st Qu.: 0.0000  
##  normal          :551   Median :140.0   NA's : 55   Median : 0.5000  
##  st-t abnormality:179   Mean   :137.5               Mean   : 0.8788  
##                         3rd Qu.:157.0               3rd Qu.: 1.5000  
##                         Max.   :202.0               Max.   : 6.2000  
##                         NA's   :55                  NA's   :62       
##          slope        ca                      thal          target    
##             :309   0   :181                     :486   Absence :411  
##  downsloping: 63   1   : 67   fixed defect     : 46   Presence:509  
##  flat       :345   2   : 41   normal           :196                 
##  upsloping  :203   3   : 20   reversable defect:192                 
##                    NA's:611                                         
##                                                                     
##
```

Upon closer inspection, we can see that many variables have “NA” values within them, which is being displayed by the summary function above. This means we have a lot of data which is either missing or doesn’t pertain to the overall evaluation. To clean the data, we need to remove the “NA” values from the data set, for which we can use the omit function as follows:

```
heart <- na.omit(heart)
dim(heart)
```

```
## [1] 303  16
```

As mentioned above, out of the 16 variables, the 2 variables called “id” and “dataset” don’t contribute anything to our overall study. In order to remove these columns, we use their indices, so 1 for “id” and 3 for “dataset”. The following commands were used to remove them:

```
heart <- heart[, -1]
heart <- heart[, -3]
```

After the above step, our working data set is as follows:

```
str(heart)
```

```
## tibble [303 x 14] (S3: tbl_df/tbl/data.frame)
##  $ age     : int [1:303] 63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 2 1 1 2 2 ...
##  $ cp      : Factor w/ 4 levels "asymptomatic",..: 4 1 1 3 2 2 1 1 1 1 ...
##  $ trestbps: int [1:303] 145 160 120 130 130 120 140 120 130 140 ...
##  $ chol    : int [1:303] 233 286 229 250 204 236 268 354 254 203 ...
##  $ fbs     : Factor w/ 2 levels "FALSE","TRUE": 2 1 1 1 1 1 1 1 1 2 ...
##  $ restecg : Factor w/ 4 levels "","lv hypertrophy",..: 2 2 2 3 2 3 2 3 2 2 ...
##  $ thalch  : int [1:303] 150 108 129 187 172 178 160 163 147 155 ...
##  $ exang   : Factor w/ 2 levels "FALSE","TRUE": 1 2 2 1 1 1 1 2 1 2 ...
##  $ oldpeak : num [1:303] 2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ slope   : Factor w/ 4 levels "","downsloping",..: 2 3 3 2 4 4 2 4 3 2 ...
##  $ ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
##  $ thal    : Factor w/ 4 levels "","fixed defect",..: 2 3 4 3 3 3 3 3 4 4 ...
##  $ target  : Factor w/ 2 levels "Absence","Presence": 1 2 2 1 1 1 2 1 2 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:617] 167 193 288 303 304 305 306 307 308 309 ...
##   ..- attr(*, "names")= chr [1:617] "167" "193" "288" "303" ...
```

Explanatory Analysis

Now that we have imported, cleaned, and manipulated the data, we can dive further into some analysis of the data set and how certain variables stack up against each other, for example, what are the chances of heart disease with respect to age etc.

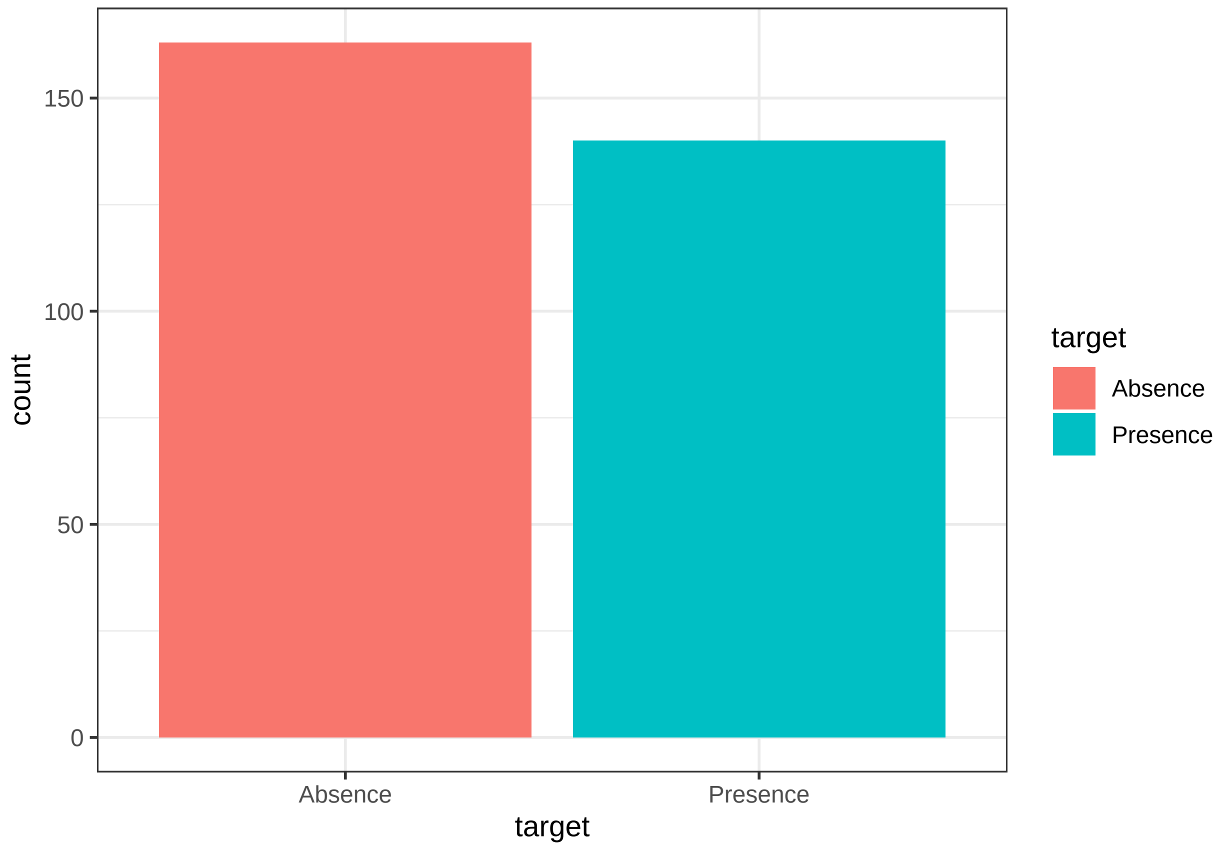
First up, we can check the distribution of variable **"target"**, which tells us about the presence or absence of heart disease in an individual. For this, we make use of the table function and then plot the same using ggplot histogram:

```
library(ggplot2)
table(heart$target)
```

```
##
##  Absence Presence
##      163      140
```

```
ggplot(heart, aes(target, fill = target)) + geom_histogram(stat = "count") + theme_bw()
```

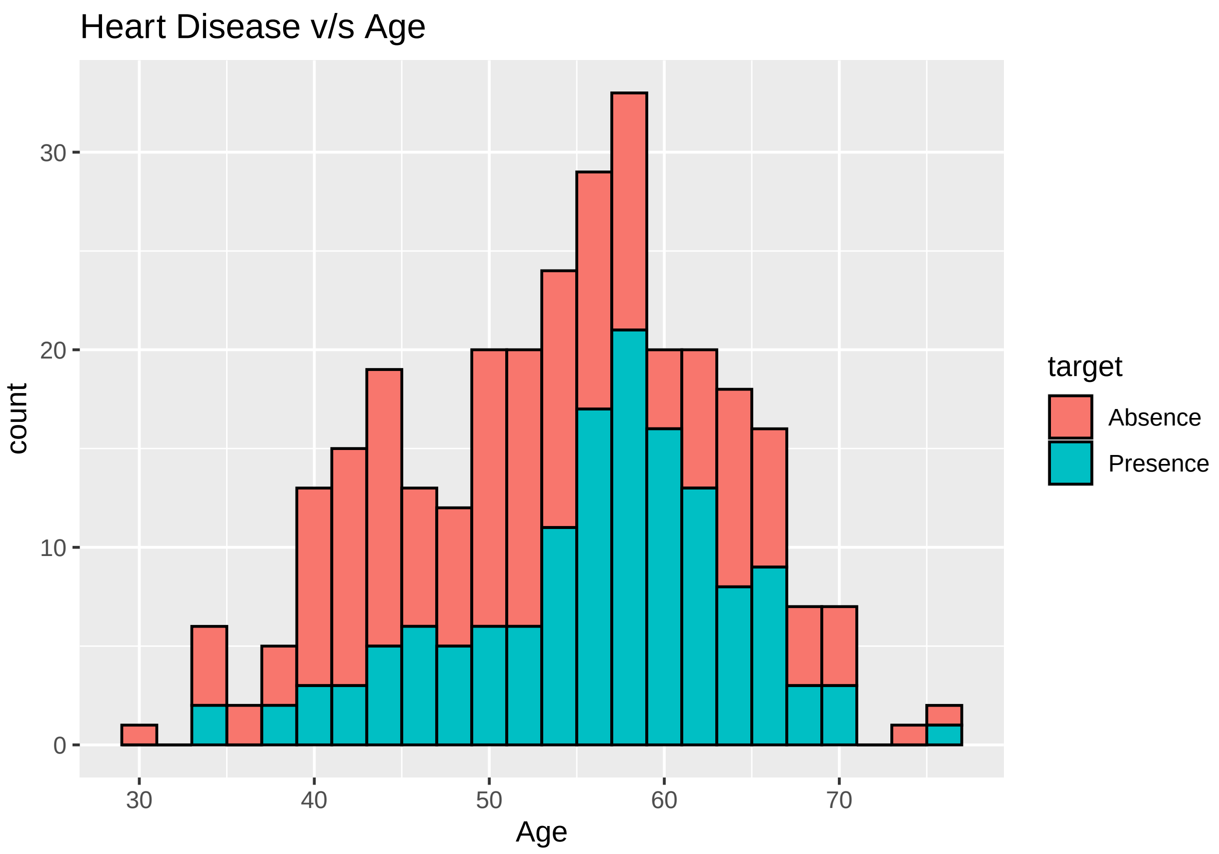
```
## Warning in geom_histogram(stat = "count"): Ignoring unknown parameters:
## `binwidth`, `bins`, and `pad`
```



Based on the above histogram and frequency distribution, we see that there's an almost equal split between individuals who have heart disease and those who don't, with the absence of heart disease being slightly higher at **163** compared to presence at **140**

Now that we've seen the distribution of the target variable, we can analyze it against factors like age, where it would be good to know what age groups are likely to get heart disease and vice versa. For that, we will use a ggplot histogram of target with respect to (w.r.t) age variable

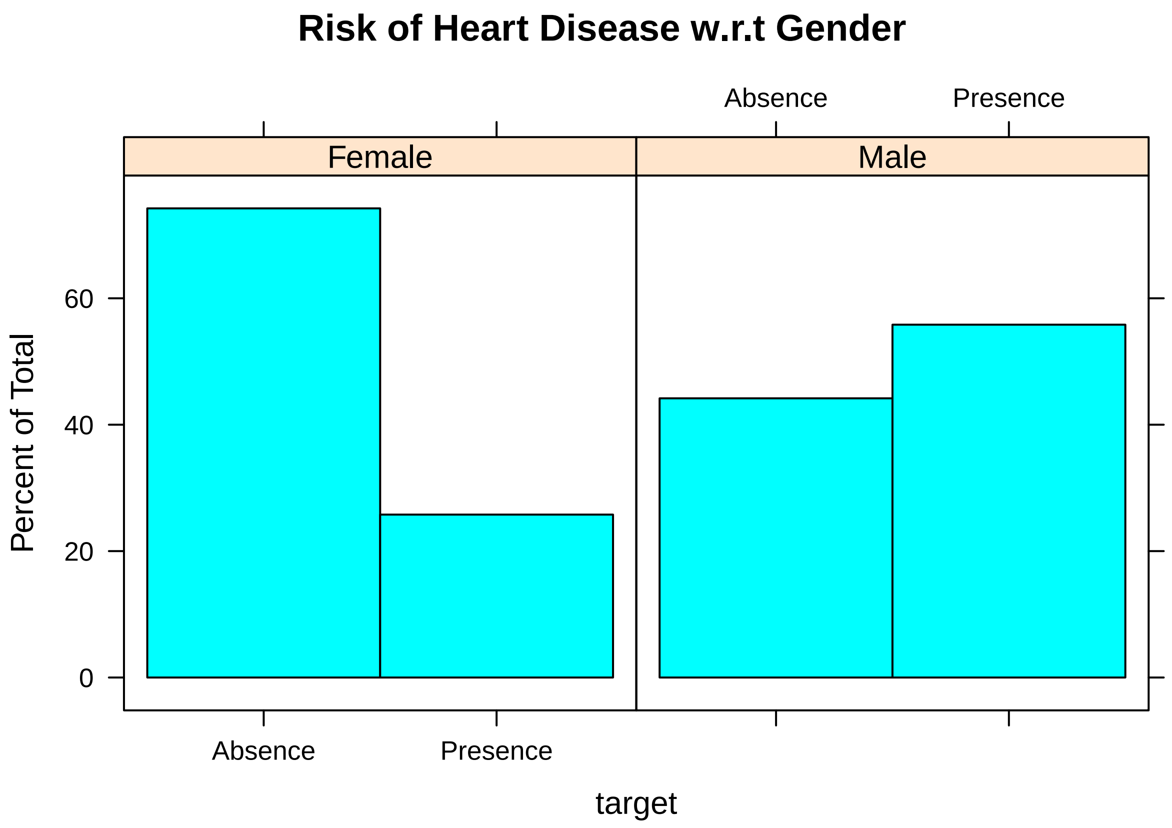
```
ggplot(heart, aes(x = age, fill = target, color = target)) +
geom_histogram(binwidth = 2, color="black") +
labs(x = "Age", title = "Heart Disease v/s Age")
```



This distributions shows us that broader *age groups 55 to 65 have higher chances* of getting heart disease compared to age groups above age 68.

Similarly, if we want to understand the risk of heart disease w.r.t gender (Male / Female), we can look at the following distribution:

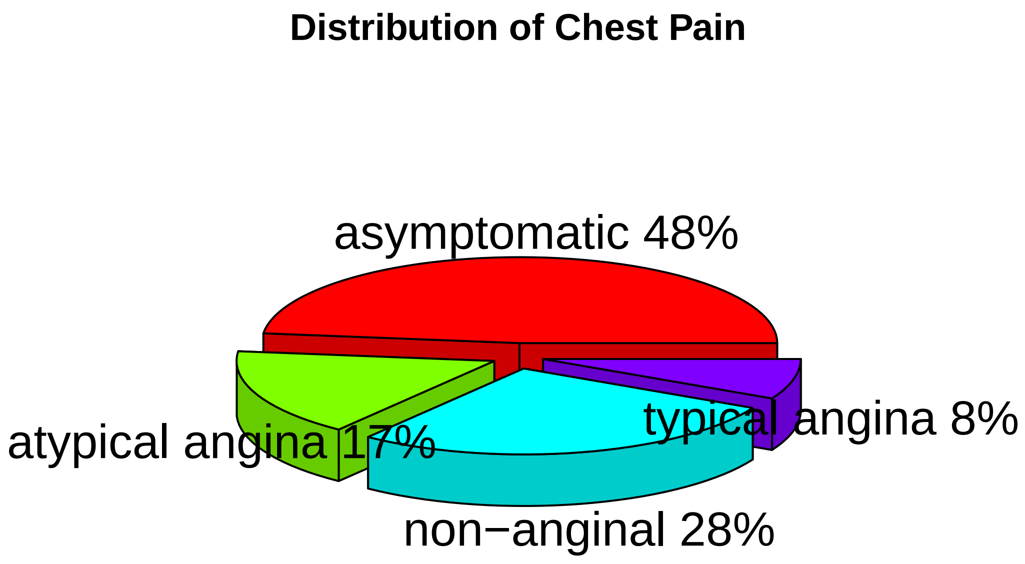
```
library(lattice)
heart %>% histogram(~target | sex, data= ., main = "Risk of Heart Disease w.r.t Gender")
```



Here we can see the proportion of presence of heart disease in females is much lower (~25%) compared to proportion of presence of heart disease among males (approximately 50%).

We observe that there's a variable called **"cp"**, which refers to chest pain, so in order to see the distribution of chest pain, we can use the table function and then display it in form of a pie chart, as follows:

```
library(ggplot2)
library(plotrix)
pietable <- table(heart$cp)
percent <- round(pietable/sum(pietable)*100)
label1 <- paste(names(pietable),percent)
label2 <- paste(label1, "%", sep="")
pie3D(pietable, labels = label2, explode = 0.1,
main="Distribution of Chest Pain", radius = 1)
```



Here we can see that **48%** of the chest pain cases are asymptomatic, but out of the ones with symptoms, the highest percentage is for non-anginal type chest pains (**28%**), where there's no angina presence.

Decision Tree Model 1

Partition Data into Train and Test Data

In order to build a classification / decision tree, we need to first split our data into train and test data, for which we can take a **70-30** approach, where **70%** of heart data set will be train data and **30%** will be the test data.

```
set.seed(1) # to ensure we're always getting the same split
index_heart_model1 <- sample(2, nrow(heart), replace= TRUE, prob = c(0.7, 0.3))
train_heart_model1 <- heart[index_heart_model1 == 1,  ]
test_heart_model1 <- heart[index_heart_model1 == 2, ]
```

Classification Tree using default "cp"

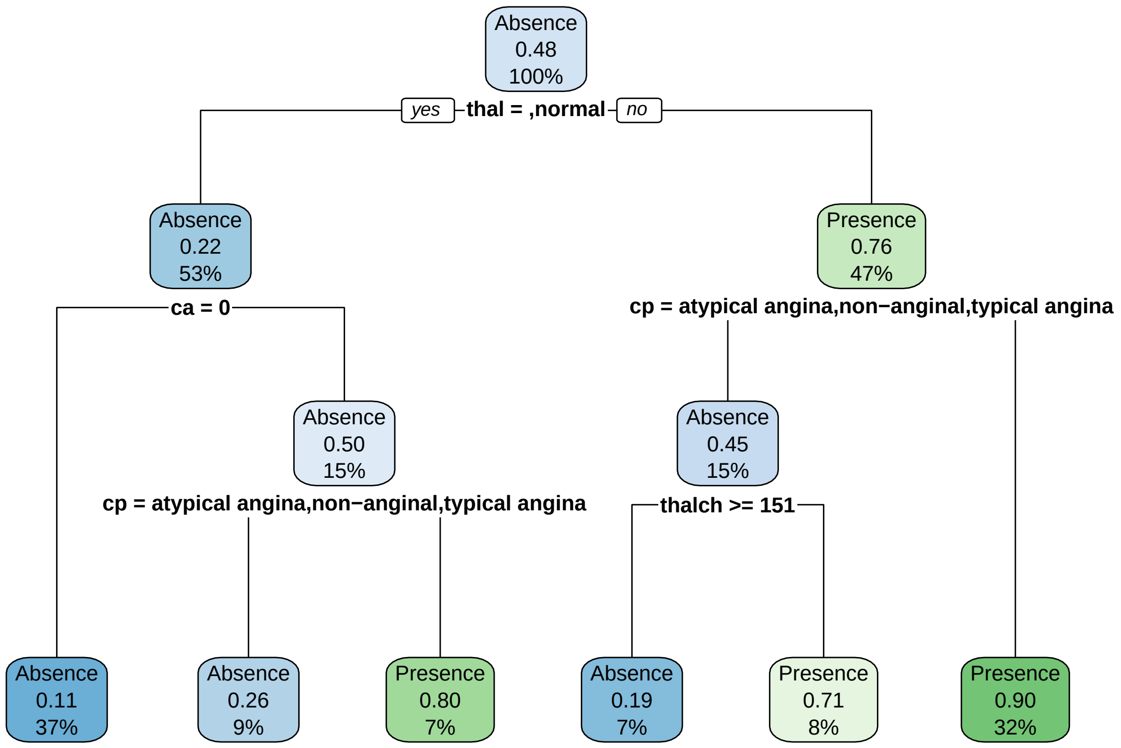
We now make use of the rpart package to build the decision tree model, where we first start with the **"target"** variable and compare it against all predictors, by using ~., which refers to input variables. We're also using default cp of 0.01, which means any split that does not reduce the tree's overall complexity by a factor of 0.01, is not attempted. The code for that is as follows:

```
library(rpart)
tree_heart_model1_train <- rpart(target ~., train_heart_model1)
print(tree_heart_model1_train) # to print the decision rules
```

```
## n= 220
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 220 105 Absence (0.52272727 0.47727273)
##    2) thal=,normal 116  26 Absence (0.77586207 0.22413793)
##      4) ca=0 82   9 Absence (0.89024390 0.10975610) *
##      5) ca=1,2,3 34  17 Absence (0.50000000 0.50000000)
##       10) cp=atypical angina,non-anginal,typical angina 19   5 Absence (0.73684211 0.26315789) *
##       11) cp=asymptomatic 15   3 Presence (0.20000000 0.80000000) *
##    3) thal=fixed defect,reversable defect 104  25 Presence (0.24038462 0.75961538)
##      6) cp=atypical angina,non-anginal,typical angina 33  15 Absence (0.54545455 0.45454545)
##       12) thalch>=150.5 16   3 Absence (0.81250000 0.18750000) *
##       13) thalch< 150.5 17   5 Presence (0.29411765 0.70588235) *
##      7) cp=asymptomatic 71   7 Presence (0.09859155 0.90140845) *
```

To plot an rpart decision tree we can use the “rpart.plot()” function from “rpart.plot” package:

```
library(rpart.plot)
rpart.plot(tree_heart_model1_train)
```



```
rpart.rules(tree_heart_model1_train)
```

target
0.11 when thal is or normal & ca is 0
0.19 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & thalch >= 151 0.26 when thal is or normal & cp is atypical angina or non-anginal or typical angina & ca is 1 or 2 or 3
0.71 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & thalch < 151 0.80 when thal is or normal & cp is asymptomatic & ca is 1 or 2 or 3
0.90 when thal is fixed defect or reversable defect & cp is asymptomatic

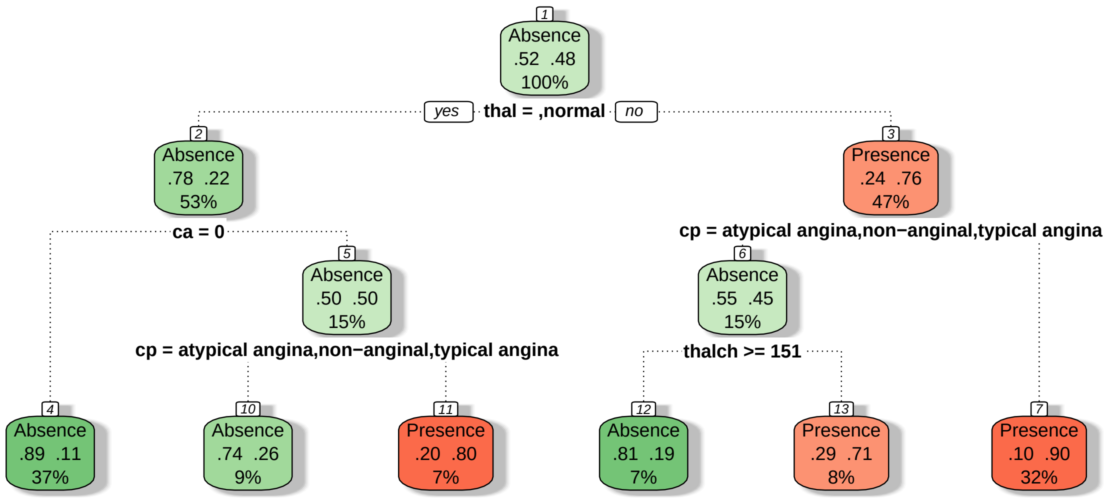
In order to see a more fancier version of rpart.plot, we also have the option of fancyRpartPlot() function, which is part of the rattle library. It can be run as follows:

```
library(rattle)
```

```
## 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.
```

```
fancyRpartPlot(tree_heart_model1_train, palettes=c("Greens", "Reds"), sub="")
```



To obtain the predicted classes or predicted probabilities we can use the "predict" function.

```
tree_heart_pred_prob_model1 <-
predict(tree_heart_model1_train, train_heart_model1)

tree_heart_pred_prob_model1 <-
predict(tree_heart_model1_train, train_heart_model1, type = "prob")

tree_heart_pred_class_model1 <-
predict(tree_heart_model1_train, train_heart_model1, type = "class")
```

The error rate of the decision tree model on training data:

```
error_rate_heart_train_model1 <-
mean(tree_heart_pred_class_model1 != train_heart_model1$target)

print(error_rate_heart_train_model1)
```

```
## [1] 0.1454545
```

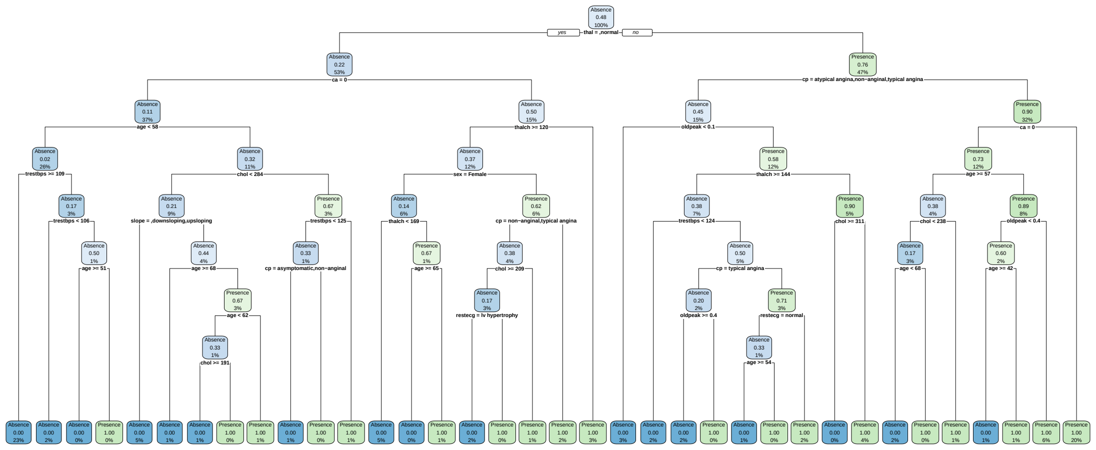
The error rate of the decision tree model on test data is:

```
tree_heart_pred_test_model1 <-
predict(tree_heart_model1_train, test_heart_model1, type = "class")

base_error_heart_model1 <-
mean(tree_heart_pred_test_model1 != test_heart_model1$target)

print(base_error_heart_model1)
```

```
## [1] 0.1807229
```

Fully Grown Decision Tree (cp=0, split = “information”)

```
tree_heart_model1 <- rpart(target ~ ., train_heart_model1,
parms = list(split = "information"),
control = rpart.control(minbucket = 0, minsplit = 0, cp = 0))
rpart.plot(tree_heart_model1)
```



```
pred_heart_test_model1 <-
predict(tree_heart_model1, test_heart_model1, type = "class")

error_preprun_heart_model1 <-
mean(pred_heart_test_model1 != test_heart_model1$target)
```

Selecting the Best CP

The CP table allows you to see what's the best decision tree that would help you minimize the misclassification error.

```r
printcp(tree_heart_model1)
```

```
##
## Classification tree:
## rpart(formula = target ~ ., data = train_heart_model1, parms = list(split = "information"),
##     control = rpart.control(minbucket = 0, minsplit = 0, cp = 0))
##
## Variables actually used in tree construction:
##  [1] age      ca       chol     cp       oldpeak  restecg  sex      slope
##  [9] thal     thalch   trestbps
##
## Root node error: 105/220 = 0.47727
##
## n= 220
##
##           CP nsplit rel error  xerror     xstd
## 1  0.5142857      0 1.0000000 1.00000 0.070557
## 2  0.0349206      1 0.4857143 0.60000 0.063859
## 3  0.0333333      4 0.3809524 0.50476 0.060408
## 4  0.0285714      6 0.3142857 0.50476 0.060408
## 5  0.0190476      7 0.2857143 0.49524 0.060014
## 6  0.0142857      9 0.2476190 0.50476 0.060408
## 7  0.0126984     11 0.2190476 0.58095 0.063236
## 8  0.0095238     14 0.1809524 0.58095 0.063236
## 9  0.0031746     32 0.0095238 0.58095 0.063236
## 10 0.0000000     35 0.0000000 0.60952 0.064158
```

```r
mincp_i_heart_model1 <- which.min(tree_heart_model1$cptable[, 'xerror'])
```

To get the best cp, we can use two approaches:

Approach 1

Here we use the above calculated mincp_i value and find the row (index) corresponding to the min xerror:

```
optCP_heart_model1 <- tree_heart_model1$cptable[mincp_i_heart_model1, "CP"]
```

Approach 2

We calculate the optimal xerror by adding min_xerror + min_xstd, which we do as follows:

```
optError_heart_model1 <-
tree_heart_model1$cptable[mincp_i_heart_model1, "xerror"]
+ tree_heart_model1$cptable[mincp_i_heart_model1, "xstd"]
```

```
## [1] 0.06001443
```

After this, we find the row(index) of the xerror value which is closest to optError calculated above, using the following code:

```
optCP_i_heart_model1 <-
which.min(abs( tree_heart_model1$cptable[,"xerror"] - optError_heart_model1))
```

Finally, to get the best CP, we find the cp value corresponding to optCP_i calculated above:

```
optCP_heart_model1 <- tree_heart_model1$cptable[optCP_i_heart_model1, "CP"]
print(optCP_heart_model1)
```

```
## [1] 0.01904762
```

Now that we've gotten the best cp value, we can proceed with pruning our decision tree and calculate the accuracy of the decision tree, as follows:

```r
model1_heart_pruned  <- prune(tree_heart_model1, cp = optCP_heart_model1)

test_heart_model1$pred <-
predict(model1_heart_pruned, test_heart_model1, type = "class")

error_postprun_heart_model1 <-
mean(test_heart_model1$pred != test_heart_model1$target)

df_heart_model1 <-
data.frame(base_error_heart_model1,
error_preprun_heart_model1, error_postprun_heart_model1)

base_error_pct_heart_model1 <-
paste(round(base_error_heart_model1*100, 3), "%", sep = "")

error_preprun_pct_heart_model1 <-
paste(round(error_preprun_heart_model1*100, 3), "%", sep = "")

error_postprun_pct_heart_model1 <-
paste(round(error_postprun_heart_model1*100, 3), "%", sep = "")

df_percent_heart_model1 <-
data.frame(base_error_pct_heart_model1,
error_preprun_pct_heart_model1, error_postprun_pct_heart_model1)
```

Error rate and error percentage for base_error, error before pruning, and error after pruning are as follows:

```r
kable(df_heart_model1)
```

base_error_heart_model1	error_preprun_heart_model1	error_postprun_heart_model1
0.1807229	0.313253	0.2289157

```r
kable(df_percent_heart_model1)
```

base_error_pct_heart_model1	error_preprun_pct_heart_model1	error_postprun_pct_heart_model1
18.072%	31.325%	22.892%

Decision Tree Model 2

Partition Data into Train and Test Data

In order to build a classification / decision tree, we need to first split our data into train and test data, for which we can take a **80-20** approach, where **80%** of heart data set will be train data and **20%** will be the test data.

```
set.seed(1) # to ensure we're always getting the same split
index_heart_model2 <- sample(2, nrow(heart), replace= TRUE, prob = c(0.8, 0.2))
train_heart_model2 <- heart[index_heart_model2 == 1,  ]
test_heart_model2 <- heart[index_heart_model2 == 2, ]
```

Classification Tree using default "cp"

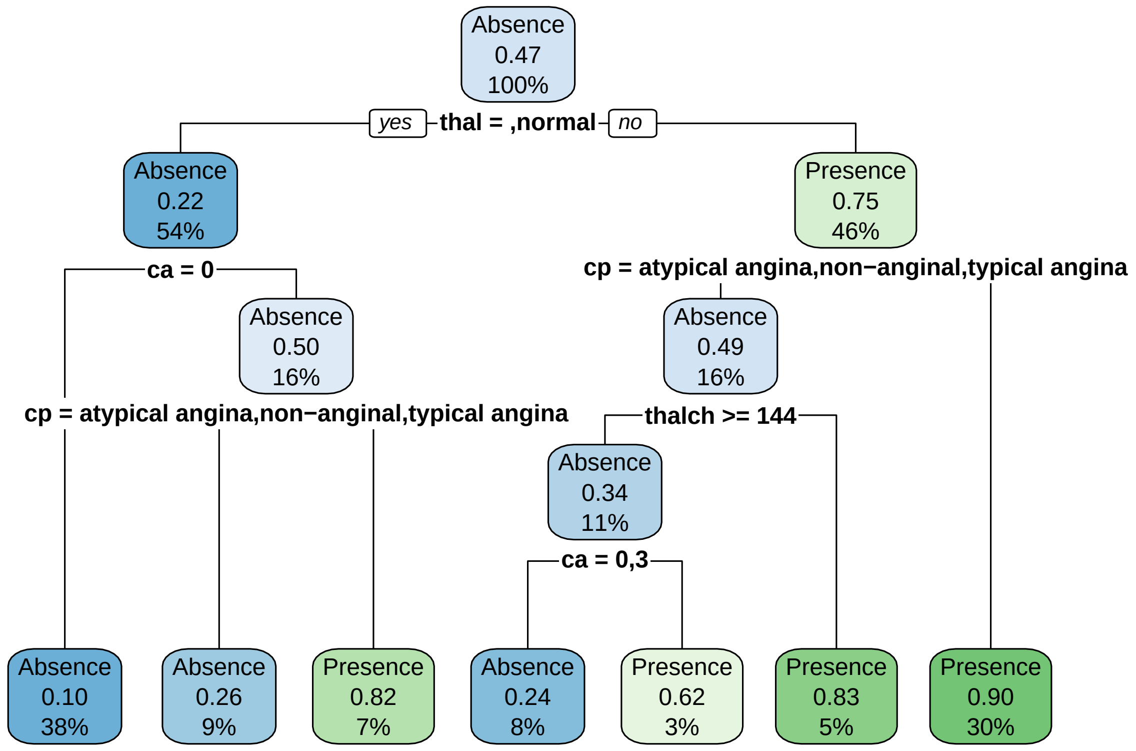
We now make use of the rpart package to build the decision tree model, where we first start with the **"target"** variable and compare it against all predictors, by using ~., which refers to input variables. We're also using default cp of 0.01, which means any split that does not reduce the tree's overall complexity by a factor of 0.01, is not attempted. The code for that is as follows:

```
library(rpart)
tree_heart_model2_train <- rpart(target ~., train_heart_model2)
print(tree_heart_model2_train) # to print the decision rules
```

```
## n= 254
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 254 119 Absence (0.5314961 0.4685039)
##    2) thal=,normal 136  30 Absence (0.7794118 0.2205882)
##      4) ca=0 96  10 Absence (0.8958333 0.1041667) *
##      5) ca=1,2,3 40  20 Absence (0.5000000 0.5000000)
##       10) cp=atypical angina,non-anginal,typical angina 23   6 Absence (0.7391304 0.2608696) *
##       11) cp=asymptomatic 17   3 Presence (0.1764706 0.8235294) *
##    3) thal=fixed defect,reversable defect 118  29 Presence (0.2457627 0.7542373)
##      6) cp=atypical angina,non-anginal,typical angina 41  20 Absence (0.5121951 0.4878049)
##       12) thalch>=144 29  10 Absence (0.6551724 0.3448276)
##         24) ca=0,3 21   5 Absence (0.7619048 0.2380952) *
##         25) ca=1,2 8   3 Presence (0.3750000 0.6250000) *
##       13) thalch< 144 12   2 Presence (0.1666667 0.8333333) *
##      7) cp=asymptomatic 77   8 Presence (0.1038961 0.8961039) *
```

To plot an rpart decision tree we can use the “rpart.plot()” function from “rpart.plot” package:

```
library(rpart.plot)
rpart.plot(tree_heart_model2_train)
```

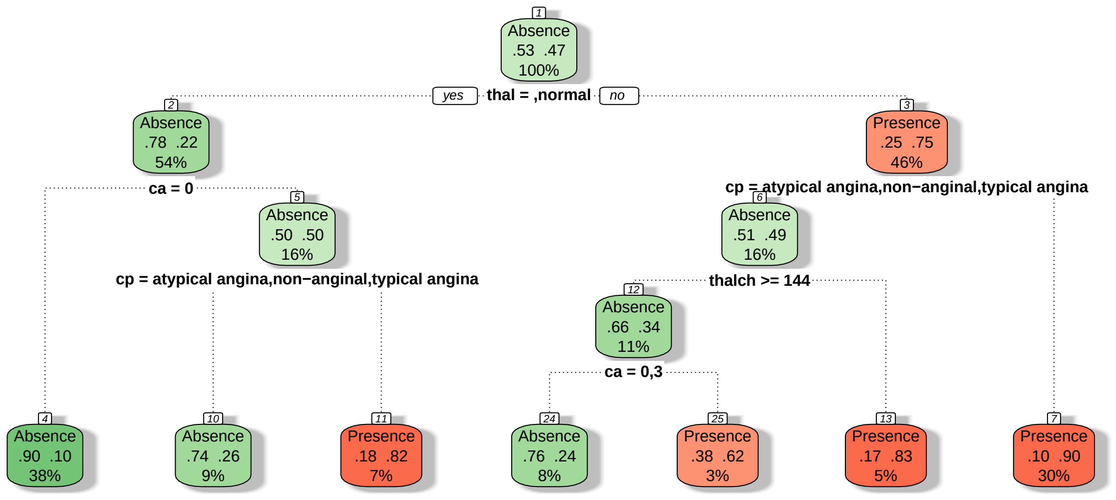


```
rpart.rules(tree_heart_model2_train)
```

target
0.10 when thal is or normal & ca is 0
0.24 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & ca is 0 or 3 & thalch >= 144 0.26 when thal is or normal & cp is atypical angina or non-anginal or typical angina & ca is 1 or 2 or 3
0.62 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & ca is 1 or 2 & thalch >= 144 0.82 when thal is or normal & cp is asymptomatic & ca is 1 or 2 or 3
0.83 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & thalch < 144 0.90 when thal is fixed defect or reversable defect & cp is asymptomatic

In order to see a more fancier version of rpart.plot, we also have the option of fancyRpartPlot() function, which is part of the rattle library. It can be run as follows:

```r
library(rattle)
fancyRpartPlot(tree_heart_model2_train, palettes=c("Greens", "Reds"), sub="")
```



To obtain the predicted classes or predicted probabilities we can use the "predict" function.

```
tree_heart_pred_prob_model2 <-
predict(tree_heart_model2_train, train_heart_model2)

tree_heart_pred_prob_model2 <- predict(tree_heart_model2_train,
train_heart_model2, type = "prob")

tree_heart_pred_class_model2 <- predict(tree_heart_model2_train,
train_heart_model2, type = "class")
```

The error rate of the decision tree model on training data:

```
error_rate_heart_train_model2 <-
mean(tree_heart_pred_class_model2 != train_heart_model2$target)

print(error_rate_heart_train_model2)
```

```
## [1] 0.1456693
```

The error rate of the decision tree model on test data is:

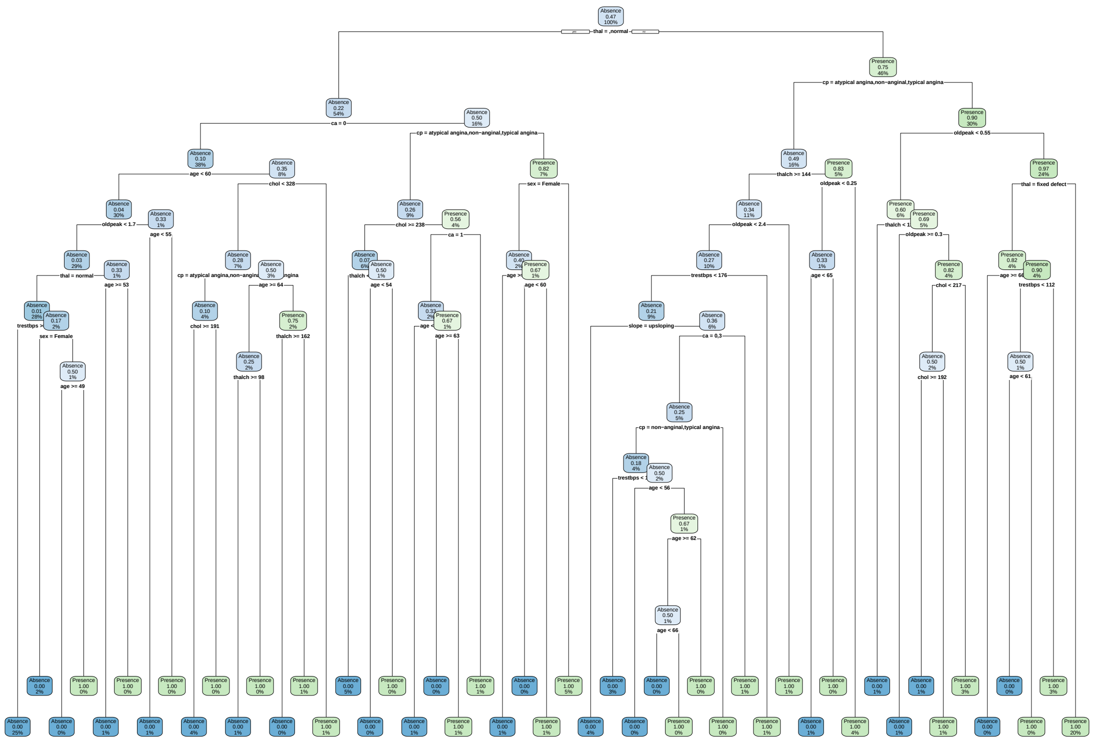
```
tree_heart_pred_test_model2 <-
predict(tree_heart_model2_train, test_heart_model2, type = "class")

base_error_heart_model2 <-
mean(tree_heart_pred_test_model2 != test_heart_model2$target)

print(base_error_heart_model2)
```

```
## [1] 0.1632653
```

Fully Grown Decision Tree (cp=0, split = "gini")

```
tree_heart_model2 <-
rpart(target ~ ., train_heart_model2, parms = list(split = "gini")
, control = rpart.control(minbucket = 0, minsplit = 0, cp = 0))
rpart.plot(tree_heart_model2)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
pred_heart_test_model2 <-
predict(tree_heart_model2, test_heart_model2, type = "class")

error_preprun_heart_model2 <-
mean(pred_heart_test_model2 != test_heart_model2$target)
```

Selecting the Best CP

The CP table allows you to see what's the best decision tree that would help you minimize the misclassification error.

```
printcp(tree_heart_model2)
```

```
##
## Classification tree:
## rpart(formula = target ~ ., data = train_heart_model2, parms = list(split = "gini"),
##     control = rpart.control(minbucket = 0, minsplit = 0, cp = 0))
##
## Variables actually used in tree construction:
##  [1] age      ca       chol     cp       oldpeak  sex      slope    thal
##  [9] thalch   trestbps
##
## Root node error: 119/254 = 0.4685
##
## n= 254
##
##           CP nsplit rel error  xerror     xstd
## 1  0.5042017      0 1.0000000 1.00000 0.066831
## 2  0.0462185      1 0.4957983 0.57983 0.059573
## 3  0.0378151      3 0.4033613 0.52101 0.057528
## 4  0.0252101      5 0.3277311 0.51261 0.057211
## 5  0.0168067      6 0.3025210 0.50420 0.056887
## 6  0.0126050      7 0.2857143 0.53782 0.058144
## 7  0.0112045      9 0.2605042 0.55462 0.058734
## 8  0.0084034     12 0.2268908 0.55462 0.058734
## 9  0.0042017     31 0.0672269 0.58824 0.059840
## 10 0.0028011     45 0.0084034 0.59664 0.060102
## 11 0.0000000     48 0.0000000 0.59664 0.060102
```

```
mincp_i_heart_model2 <- which.min(tree_heart_model2$cptable[, 'xerror'])
```

To get the best cp, we can use two approaches:

Approach 1

Here we use the above calculated mincp_i value and find the row (index) corresponding to the min xerror:

```
optCP_heart_model2 <- tree_heart_model2$cptable[mincp_i_heart_model2, "CP"]
```

Approach 2

We calculate the optimal xerror by adding min_xerror + min_xstd, which we do as follows:

```
optError_heart_model2 <-
tree_heart_model2$cptable[mincp_i_heart_model2, "xerror"]
+ tree_heart_model2$cptable[mincp_i_heart_model2, "xstd"]
```

```
## [1] 0.05688695
```

After this, we find the row(index) of the xerror value which is closest to optError calculated above, using the following code:

```
optCP_i_heart_model2 <-
which.min(abs(tree_heart_model2$cptable[,"xerror"] - optError_heart_model2))
```

Finally, to get the best CP, we find the cp value corresponding to optCP_i calculated above:

```
optCP_heart_model2 <- tree_heart_model2$cptable[optCP_i_heart_model2, "CP"]
print(optCP_heart_model2)
```

```
## [1] 0.01680672
```

Now that we've gotten the best cp value, we can proceed with pruning our decision tree and calculate the accuracy of the decision tree, as follows:

```r
model2_heart_pruned  <- prune(tree_heart_model2, cp = optCP_heart_model2)

test_heart_model2$pred <-
predict(model2_heart_pruned, test_heart_model2, type = "class")

error_postprun_heart_model2 <-
mean(test_heart_model2$pred != test_heart_model2$target)

df_heart_model2 <-
data.frame(base_error_heart_model2, error_preprun_heart_model2,
error_postprun_heart_model2)

base_error_heart_pct_model2 <-
paste(round(base_error_heart_model2*100, 3), "%", sep = "")

error_preprun_heart_pct_model2 <-
paste(round(error_preprun_heart_model2*100, 3),
"%", sep = "")

error_postprun_heart_pct_model2 <-
paste(round(error_postprun_heart_model2*100, 3),
"%", sep = "")

df_percent_heart_model2 <- data.frame(base_error_heart_pct_model2,
error_preprun_heart_pct_model2, error_postprun_heart_pct_model2)
```

Error rate and error percentage for base_error, error before pruning, and error after pruning are as follows:

```r
kable(df_heart_model2)
```

base_error_heart_model2	error_preprun_heart_model2	error_postprun_heart_model2
0.1632653	0.2857143	0.1836735

```r
kable(df_percent_heart_model2)
```

base_error_heart_pct_model2	error_preprun_heart_pct_model2	error_postprun_heart_pct_model2
16.327%	28.571%	18.367%

Decision Tree Model 3

Partition Data into Train and Test Data

In order to build a classification / decision tree, we need to first split our data into train and test data, for which we can take a **80-20** approach, where **80%** of heart data set will be train data and **20%** will be the test data.

```
set.seed(1) # to ensure we're always getting the same split
index_heart_model3 <- sample(2, nrow(heart), replace= TRUE, prob = c(0.8, 0.2))
train_heart_model3 <- heart[index_heart_model3 == 1,  ]
test_heart_model3 <- heart[index_heart_model3 == 2, ]
```

Classification Tree using default "cp"

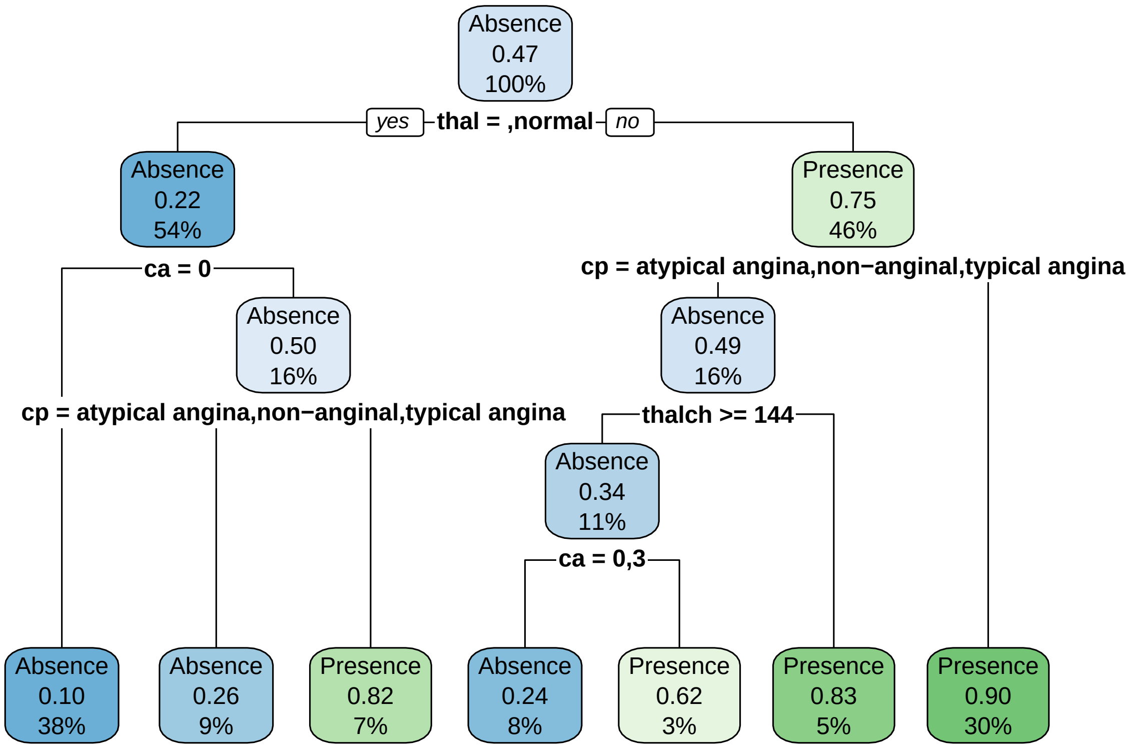
We now make use of the rpart package to build the decision tree model, where we first start with the **"target"** variable and compare it against all predictors, by using ~., which refers to input variables. We're also using default cp of 0.01, which means any split that does not reduce the tree's overall complexity by a factor of 0.01, is not attempted. The code for that is as follows:

```
library(rpart)
tree_heart_model3_train <- rpart(target ~., train_heart_model3)
print(tree_heart_model3_train) # to print the decision rules
```

```
## n= 254
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 254 119 Absence (0.5314961 0.4685039)
##    2) thal=,normal 136  30 Absence (0.7794118 0.2205882)
##      4) ca=0 96  10 Absence (0.8958333 0.1041667) *
##      5) ca=1,2,3 40  20 Absence (0.5000000 0.5000000)
##       10) cp=atypical angina,non-anginal,typical angina 23   6 Absence (0.7391304 0.2608696) *
##       11) cp=asymptomatic 17   3 Presence (0.1764706 0.8235294) *
##    3) thal=fixed defect,reversable defect 118  29 Presence (0.2457627 0.7542373)
##      6) cp=atypical angina,non-anginal,typical angina 41  20 Absence (0.5121951 0.4878049)
##       12) thalch>=144 29  10 Absence (0.6551724 0.3448276)
##         24) ca=0,3 21   5 Absence (0.7619048 0.2380952) *
##         25) ca=1,2 8   3 Presence (0.3750000 0.6250000) *
##       13) thalch< 144 12   2 Presence (0.1666667 0.8333333) *
##      7) cp=asymptomatic 77   8 Presence (0.1038961 0.8961039) *
```

To plot an rpart decision tree we can use the "rpart.plot()" function from "rpart.plot" package:

```
library(rpart.plot)
rpart.plot(tree_heart_model3_train)
```

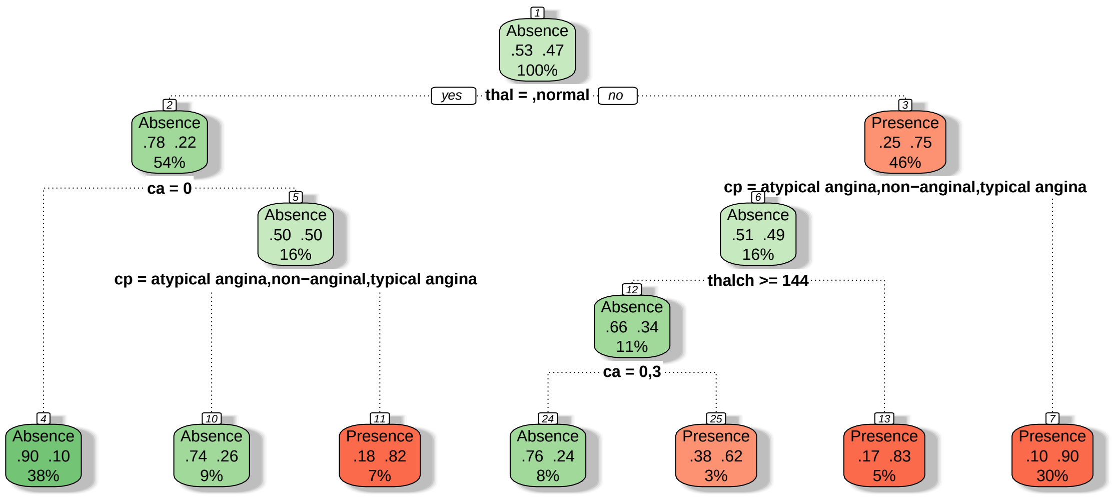


```
rpart.rules(tree_heart_model3_train)
```

target
0.10 when thal is or normal & ca is 0
0.24 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & ca is 0 or 3 & thalch >= 144 0.26 when thal is or normal & cp is atypical angina or non-anginal or typical angina & ca is 1 or 2 or 3
0.62 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & ca is 1 or 2 & thalch >= 144 0.82 when thal is or normal & cp is asymptomatic & ca is 1 or 2 or 3
0.83 when thal is fixed defect or reversable defect & cp is atypical angina or non-anginal or typical angina & thalch < 144 0.90 when thal is fixed defect or reversable defect & cp is asymptomatic

In order to see a more fancier version of rpart.plot, we also have the option of fancyRpartPlot() function, which is part of the rattle library. It can be run as follows:

```r
library(rattle)
fancyRpartPlot(tree_heart_model3_train, palettes=c("Greens", "Reds"), sub="")
```



To obtain the predicted classes or predicted probabilities we can use the “predict” function.

```r
tree_heart_pred_prob_model3 <-
predict(tree_heart_model3_train, train_heart_model3)

tree_heart_pred_prob_model3 <- predict(tree_heart_model3_train,
train_heart_model3, type = "prob")

tree_heart_pred_class_model3 <- predict(tree_heart_model3_train,
train_heart_model3, type = "class")
```

The error rate of the decision tree model on training data:

```r
error_rate_heart_train_model3 <-
mean(tree_heart_pred_class_model3 != train_heart_model3$target)

print(error_rate_heart_train_model3)
```

```
## [1] 0.1456693
```

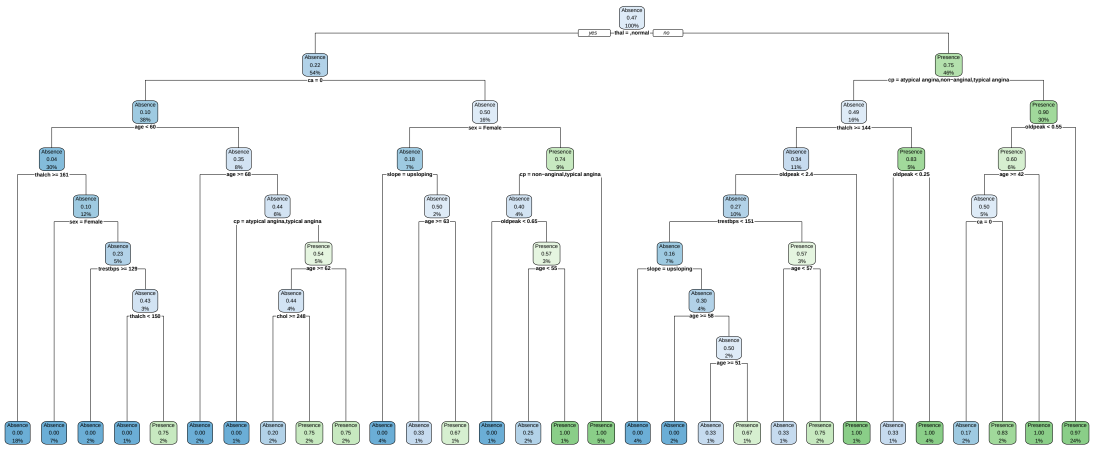
The error rate of the decision tree model on test data is:

```r
tree_heart_pred_test_model3 <-
predict(tree_heart_model3_train, test_heart_model3, type = "class")

base_error_heart_model3 <-
mean(tree_heart_pred_test_model3 != test_heart_model3$target)

print(base_error_heart_model3)
```

```
## [1] 0.1632653
```

Fully Grown Decision Tree (cp=0, split = “information”, minbucket = 3, minsplit = 5)

```r
tree_heart_model3 <-
rpart(target ~ ., train_heart_model3, parms = list(split = "information")
, control = rpart.control(minbucket = 3, minsplit = 5, cp = 0))
rpart.plot(tree_heart_model3)
```



```r
pred_heart_test_model3 <-
predict(tree_heart_model3, test_heart_model3, type = "class")

error_preprun_heart_model3 <-
mean(pred_heart_test_model3 != test_heart_model3$target)
```

Selecting the Best CP

The CP table allows you to see what's the best decision tree that would help you minimize the misclassification error.

```
printcp(tree_heart_model3)
```

```
##
## Classification tree:
## rpart(formula = target ~ ., data = train_heart_model3, parms = list(split = "information"),
##     control = rpart.control(minbucket = 3, minsplit = 5, cp = 0))
##
## Variables actually used in tree construction:
##  [1] age      ca       chol     cp       oldpeak  sex      slope    thal
##  [9] thalch   trestbps
##
## Root node error: 119/254 = 0.4685
##
## n= 254
##
##           CP nsplit rel error  xerror     xstd
## 1  0.5042017      0   1.00000 1.00000 0.066831
## 2  0.0462185      1   0.49580 0.57983 0.059573
## 3  0.0378151      3   0.40336 0.52101 0.057528
## 4  0.0252101      5   0.32773 0.52101 0.057528
## 5  0.0168067      6   0.30252 0.53782 0.058144
## 6  0.0126050      7   0.28571 0.57143 0.059299
## 7  0.0112045      9   0.26050 0.57983 0.059573
## 8  0.0084034     12   0.22689 0.57983 0.059573
## 9  0.0067227     15   0.20168 0.57143 0.059299
## 10 0.0042017     20   0.16807 0.57143 0.059299
## 11 0.0028011     26   0.14286 0.61345 0.060609
## 12 0.0000000     29   0.13445 0.62185 0.060854
```

```
mincp_i_heart_model3 <- which.min(tree_heart_model3$cptable[, 'xerror'])
```

To get the best cp, we can use two approaches:

Approach 1

Here we use the above calculated mincp_i value and find the row (index) corresponding to the min xerror:

```
optCP_heart_model3 <- tree_heart_model3$cptable[mincp_i_heart_model3, "CP"]
```

Approach 2

We calculate the optimal xerror by adding min_xerror + min_xstd, which we do as follows:

```
optError_heart_model3 <-
tree_heart_model3$cptable[mincp_i_heart_model3, "xerror"]
+ tree_heart_model3$cptable[mincp_i_heart_model3, "xstd"]
```

```
## [1] 0.05752845
```

After this, we find the row(index) of the xerror value which is closest to optError calculated above, using the following code:

```
optCP_i_heart_model3 <-
which.min(abs(tree_heart_model3$cptable[,"xerror"] - optError_heart_model3))
```

Finally, to get the best CP, we find the cp value corresponding to optCP_i calculated above:

```
optCP_heart_model3 <- tree_heart_model3$cptable[optCP_i_heart_model3, "CP"]
print(optCP_heart_model3)
```

```
## [1] 0.03781513
```

Now that we've gotten the best cp value, we can proceed with pruning our decision tree and calculate the accuracy of the decision tree, as follows:

```r
model3_heart_pruned  <- prune(tree_heart_model3, cp = optCP_heart_model3)

test_heart_model3$pred <-
predict(model3_heart_pruned, test_heart_model3, type = "class")

error_postprun_heart_model3 <-
mean(test_heart_model3$pred != test_heart_model3$target)

df_heart_model3 <-
data.frame(base_error_heart_model3, error_preprun_heart_model3,
error_postprun_heart_model3)

base_error_heart_pct_model3 <-
paste(round(base_error_heart_model3*100, 3), "%", sep = "")

error_preprun_heart_pct_model3 <-
paste(round(error_preprun_heart_model3*100, 3),
"%", sep = "")

error_postprun_heart_pct_model3 <-
paste(round(error_postprun_heart_model3*100, 3),
"%", sep = "")

df_percent_heart_model3 <- data.frame(base_error_heart_pct_model3,
error_preprun_heart_pct_model3, error_postprun_heart_pct_model3)
```

Error rate and error percentage for base_error, error before pruning, and error after pruning are as follows:

```r
kable(df_heart_model3)
```

base_error_heart_model3	error_preprun_heart_model3	error_postprun_heart_model3
0.1632653	0.244898	0.244898

```r
kable(df_percent_heart_model3)
```

base_error_heart_pct_model3	error_preprun_heart_pct_model3	error_postprun_heart_pct_model3
16.327%	24.49%	24.49%

Summary

Based on the calculations and decision tree models we created, we see the following error %:

```
kable(df_percent_heart_model1)
```

base_error_pct_heart_model1	error_preprun_pct_heart_model1	error_postprun_pct_heart_model1
18.072%	31.325%	22.892%

```
kable(df_percent_heart_model2)
```

base_error_heart_pct_model2	error_preprun_heart_pct_model2	error_postprun_heart_pct_model2
16.327%	28.571%	18.367%

```
kable(df_percent_heart_model3)
```

base_error_heart_pct_model3	error_preprun_heart_pct_model3	error_postprun_heart_pct_model3
16.327%	24.49%	24.49%

This concludes for us that **model 2, with cp=0, and split="gini" is the best model**, as it's error percentage pre and post pruning is **28.6%** and **18.4%**, which means it predicts the presence or absence of heart disease with approximately **82%** accuracy. post pruning. Models 1 and 3 are both predicting with approximately **77%** and **75%** respectively, post pruning.