Data Mining Assignment -2

Sai Hruthik Reddy Varakantham-Dorababu Chintala-Radhika Girish Gupta

2/22/2022

We want our working directory

```
setwd("C:/Users/vsaih/OneDrive/Desktop/Data_Mining/Assignement-2")
```

Loading our packages and adding German Credit to the Assignment_2 variable.

```
setwd("C:/Users/vsaih/OneDrive/Desktop/Data_Mining/Assignement-2")
pacman::p_load("pacman","tidyverse","rio","readxl")
Assignment_2<-read_xls("German Credit.xls")
```

Let us now sort CHK_ACCT as our predictor variable and RESPONSE as our target variable

```
pacman::p_load("tidyverse","rio","pacman","readxl")
Assignment_2 %>%
select(CHK_ACCT,RESPONSE)%>%
  filter(CHK_ACCT==3,RESPONSE==0)%>%
count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    46
```

The other case when RESPONSE==1 would be

```
Assignment_2 %>%
 select(CHK_ACCT,RESPONSE)%>%
  filter(CHK_ACCT==3,RESPONSE==1)%>%
 count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   348
```

Assessing our data we assumed RESPONSE to be our target variable and all other variables to be predictor variables. When we consider the CHK_ACCT as our predictor variable and RESPONSE as our target variable we consider 2 cases if(CHK_ACCT == 3 & RESPONSE ==0) we get 46 instances of that happening and if(CHK_ACCT == 3 & RESPONSE ==1) we get 348 instances of that happening. So according to the given data we say that 89 percent of the time (9/10 in proportion approx), if there is no checking account, the credit score would be good (348/394*100 = 88.32).

Considering the HISTORY categorical variable we have

Two cases HISTORY=4 & RESPONSE =0

```
Assignment_2 %>%
  select(HISTORY,RESPONSE)%>%
  filter(HISTORY==4,RESPONSE==1)%>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   243
```

So if there is a critical account in HISTORY then we can say about 8/10 i.e 83 percent of them have a good credit score.

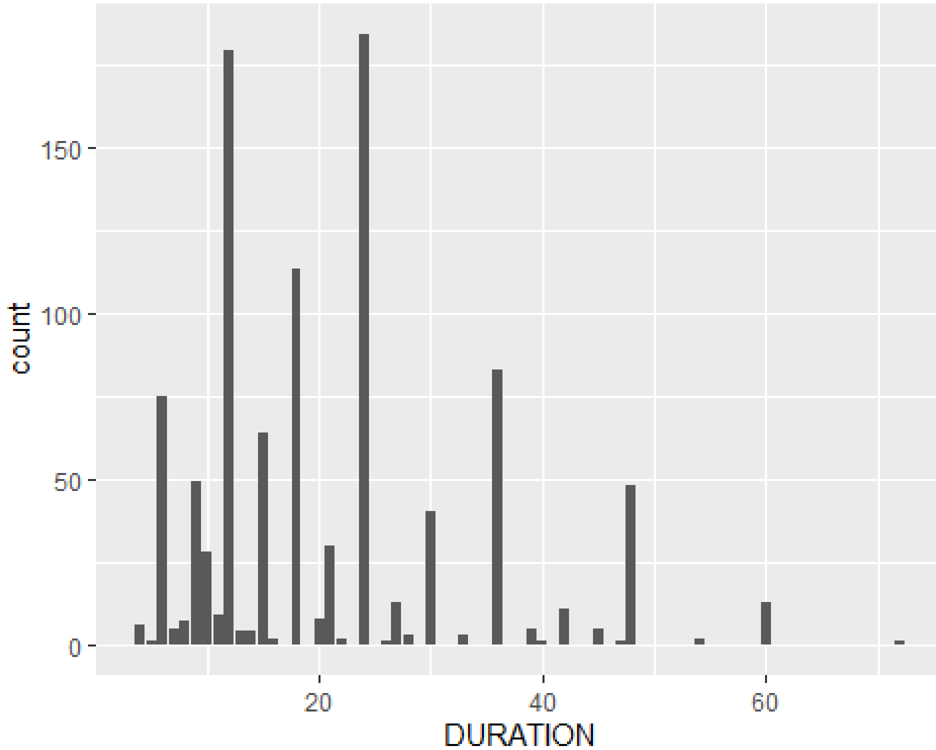
```
summary(Assignment_2)
```

```
##       OBS#          CHK_ACCT        DURATION        HISTORY     
##  Min.   :   1.0   Min.   :0.000   Min.   : 4.0   Min.   :0.000  
##  1st Qu.: 250.8   1st Qu.:0.000   1st Qu.:12.0   1st Qu.:2.000  
##  Median : 500.5   Median :1.000   Median :18.0   Median :2.000  
##  Mean   : 500.5   Mean   :1.577   Mean   :20.9   Mean   :2.545  
##  3rd Qu.: 750.2   3rd Qu.:3.000   3rd Qu.:24.0   3rd Qu.:4.000  
##  Max.   :1000.0   Max.   :3.000   Max.   :72.0   Max.   :4.000  
##     NEW_CAR         USED_CAR       FURNITURE        RADIO/TV       E
DUCATION   
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.00   Min
.   :0.00  
##  1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.00   1st 
Qu.:0.00  
##  Median :0.000   Median :0.000   Median :0.000   Median :0.00   Med
ian :0.00  
##  Mean   :0.234   Mean   :0.103   Mean   :0.181   Mean   :0.28   Mea
n   :0.05  
##  3rd Qu.:0.000   3rd Qu.:0.000   3rd Qu.:0.000   3rd Qu.:1.00   3rd 
Qu.:0.00  
##  Max.   :1.000   Max.   :1.000   Max.   :1.000   Max.   :1.00   Max
.   :1.00  
##    RETRAINING         AMOUNT         SAV_ACCT       EMPLOYMENT   
##  Min.   :0.000   Min.   :  250   Min.   :0.000   Min.   :0.000  
##  1st Qu.:0.000   1st Qu.: 1366   1st Qu.:0.000   1st Qu.:2.000  
##  Median :0.000   Median : 2320   Median :0.000   Median :2.000  
##  Mean   :0.097   Mean   : 3271   Mean   :1.105   Mean   :2.384  
```

```
##  3rd Qu.:0.000   3rd Qu.: 3972   3rd Qu.:2.000   3rd Qu.:4.000
##  Max.   :1.000   Max.   :18424   Max.   :4.000   Max.   :4.000
##   INSTALL_RATE      MALE_DIV     MALE_SINGLE    MALE_MAR_or_WID  CO
-APPLICANT
##  Min.   :1.000   Min.   :0.00   Min.   :0.000   Min.   :0.000   Min
.   :0.000
##  1st Qu.:2.000   1st Qu.:0.00   1st Qu.:0.000   1st Qu.:0.000   1st
Qu.:0.000
##  Median :3.000   Median :0.00   Median :1.000   Median :0.000   Med
ian :0.000
##  Mean   :2.973   Mean   :0.05   Mean   :0.548   Mean   :0.092   Mea
n   :0.041
##  3rd Qu.:4.000   3rd Qu.:0.00   3rd Qu.:1.000   3rd Qu.:0.000   3rd
Qu.:0.000
##  Max.   :4.000   Max.   :1.00   Max.   :1.000   Max.   :1.000   Max
.   :1.000
##    GUARANTOR     PRESENT_RESIDENT  REAL_ESTATE    PROP_UNKN_NONE
##  Min.   :0.000   Min.   :1.000    Min.   :0.000   Min.   :0.000
##  1st Qu.:0.000   1st Qu.:2.000    1st Qu.:0.000   1st Qu.:0.000
##  Median :0.000   Median :3.000    Median :0.000   Median :0.000
##  Mean   :0.052   Mean   :2.845    Mean   :0.282   Mean   :0.154
##  3rd Qu.:0.000   3rd Qu.:4.000    3rd Qu.:1.000   3rd Qu.:0.000
##  Max.   :1.000   Max.   :4.000    Max.   :1.000   Max.   :1.000
##       AGE        OTHER_INSTALL        RENT           OWN_RES
##  Min.   :19.00   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:27.00   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000
##  Median :33.00   Median :0.000   Median :0.000   Median :1.000
##  Mean   :35.55   Mean   :0.186   Mean   :0.179   Mean   :0.713
##  3rd Qu.:42.00   3rd Qu.:0.000   3rd Qu.:0.000   3rd Qu.:1.000
##  Max.   :75.00   Max.   :1.000   Max.   :1.000   Max.   :1.000
##   NUM_CREDITS         JOB        NUM_DEPENDENTS    TELEPHONE
##  Min.   :1.000   Min.   :0.000   Min.   :1.000   Min.   :0.000
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:0.000
##  Median :1.000   Median :2.000   Median :1.000   Median :0.000
##  Mean   :1.407   Mean   :1.904   Mean   :1.155   Mean   :0.404
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :4.000   Max.   :3.000   Max.   :2.000   Max.   :1.000
##     FOREIGN         RESPONSE
##  Min.   :0.000   Min.   :0.0
##  1st Qu.:0.000   1st Qu.:0.0
##  Median :0.000   Median :1.0
##  Mean   :0.037   Mean   :0.7
##  3rd Qu.:0.000   3rd Qu.:1.0
##  Max.   :1.000   Max.   :1.0
```

Let us observe how values are spread out in Duration

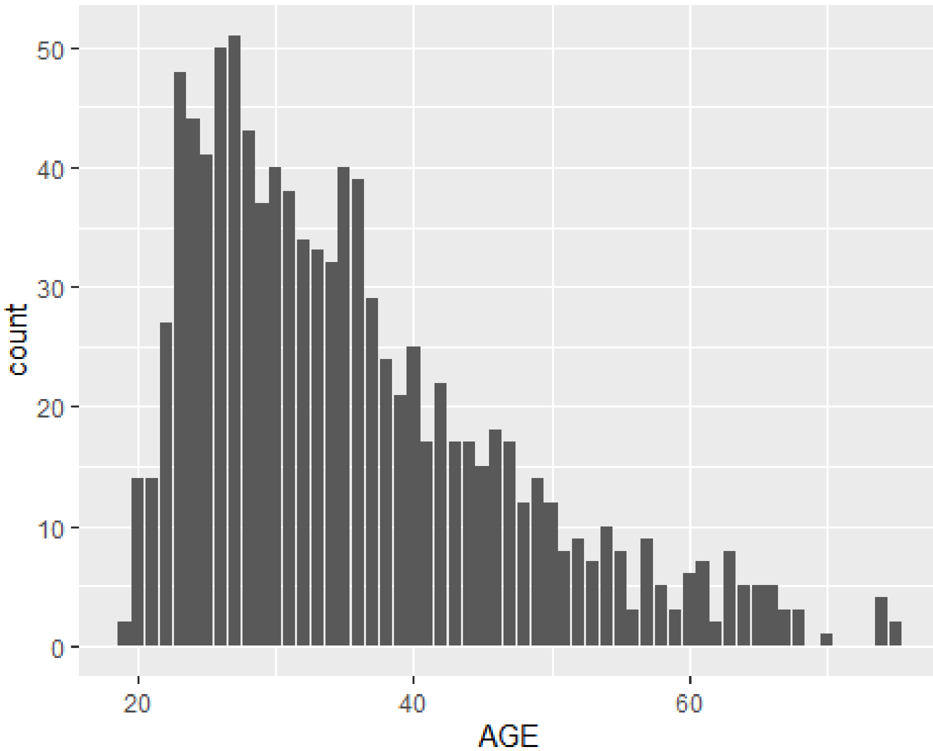
```
ggplot(Assignment_2,aes(DURATION))+geom_bar()
```



We can see from the above bar chart that there are outliers for duration greater than 24 months and we can make this numerical variable to a categorical variable with ranges like 0-10(0),11-20(1),etc.

For AGE variable we have,

```
ggplot(Assignment_2,aes(AGE))+geom_bar()
```



The maximum number of the people in the dataset are people in ages less than 45 about 799 of them and the remaining after 45.

```
Assignment_2 %>%
  select(AGE) %>%
  filter(AGE<45) %>%
  count()

## # A tibble: 1 x 1
##       n
##   <int>
## 1   799
```

(b)To avoid poor prediction, we use sample function. It generates a random list of index and we use this index to shuffle our dataset Assignment_2. The rpart function provides us the implementation of a decision tree and rpart.plot function visualizes the tree for us. RESPONSE ~.is Formula of the Decision Trees. We use the method “class” for a classification tree. By default, rpart() function uses the Gini impurity measure to split the note. The higher the Gini coefficient, the more different instances within the node.To make a prediction,the predict() function is used.Finally, we can compute an accuracy measure for classification task with the confusion matrix on train data for 70-30 split.

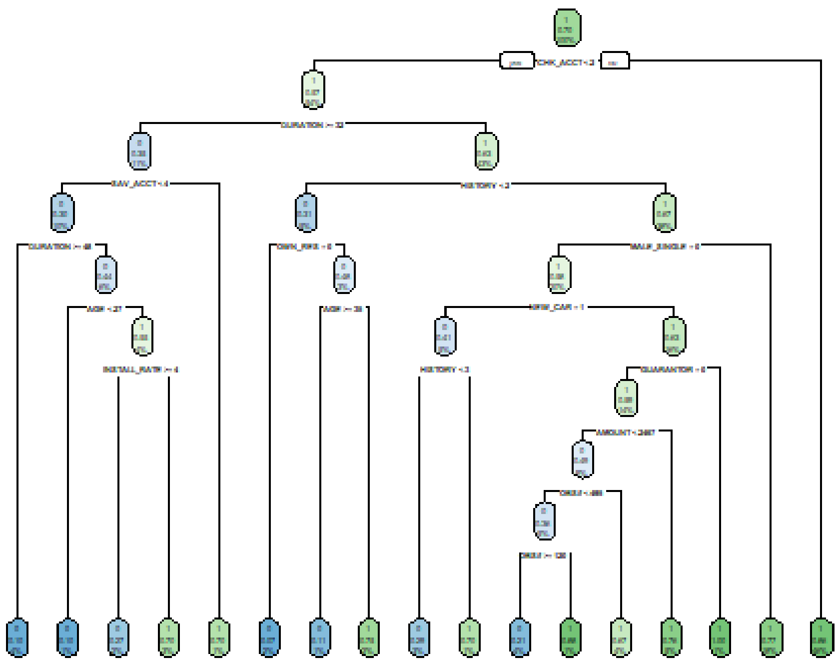
```
library(rpart)
library(rpart.plot)
set.seed(735)
pacman::p_load("tidyverse","rio","pacman","readxl")
```

```
library(ISLR)
shuffle_index <- sample(1:nrow(Assignment_2))
Assignment_2 <- Assignment_2[shuffle_index, ]
indx <- sample(2, nrow(Assignment_2), replace= TRUE, prob = c(0.7, 0.3
))
train <- Assignment_2[indx == 1,  ]
test <- Assignment_2[indx == 2, ]
tree <- rpart(RESPONSE ~ ., train)
print(tree)
```

```
## n= 713
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 713 148.9649000 0.70266480
##     2) CHK_ACCT< 1.5 388  95.1211300 0.56958760
##       4) DURATION>=31.5 80  18.2000000 0.35000000
##         8) SAV_ACCT< 3.5 70  14.7000000 0.30000000
##          16) DURATION>=47.5 29   2.6896550 0.10344830 *
##          17) DURATION< 47.5 41  10.0975600 0.43902440
##            34) AGE< 26.5 10   0.9000000 0.10000000 *
##            35) AGE>=26.5 31   7.6774190 0.54838710 *
##         9) SAV_ACCT>=3.5 10   2.1000000 0.70000000 *
##       5) DURATION< 31.5 308  72.0616900 0.62662340
##        10) HISTORY< 1.5 36   7.6388890 0.30555560
##          20) OWN_RES< 0.5 15   0.9333333 0.06666667 *
##          21) OWN_RES>=0.5 21   5.2380950 0.47619050
##            42) AGE>=35 9   0.8888889 0.11111110 *
##            43) AGE< 35 12   2.2500000 0.75000000 *
##        11) HISTORY>=1.5 272  60.2205900 0.66911760
##          22) MALE_SINGLE< 0.5 145  35.3379300 0.57931030
##            44) NEW_CAR>=0.5 34   8.2352940 0.41176470 *
##            45) NEW_CAR< 0.5 111  25.8558600 0.63063060
##              90) GUARANTOR< 0.5 101  24.3564400 0.59405940
##               180) AMOUNT< 2467 63  15.7460300 0.49206350
##                 360) OBS#< 494.5 36   8.3055560 0.36111110
##                   720) OBS#>=119.5 28   4.7142860 0.21428570 *
##                   721) OBS#< 119.5 8   0.8750000 0.87500000 *
##                 361) OBS#>=494.5 27   6.0000000 0.66666670 *
##               181) AMOUNT>=2467 38   6.8684210 0.76315790 *
##              91) GUARANTOR>=0.5 10   0.0000000 1.00000000 *
##          23) MALE_SINGLE>=0.5 127  22.3779500 0.77165350 *
##     3) CHK_ACCT>=1.5 325  38.7692300 0.86153850
##       6) OTHER_INSTALL>=0.5 58  12.4137900 0.68965520
##        12) NEW_CAR>=0.5 12   2.6666670 0.33333330 *
```

```
##        13) NEW_CAR< 0.5 46   7.8260870 0.78260870 *
##       7) OTHER_INSTALL< 0.5 267  24.2696600 0.89887640 *
```

```
library(rpart)
library(rpart.plot)
fit <- rpart(RESPONSE~., data = train, method = 'class', parms = list(
split ="gini"), control = rpart.control())
rpart.plot(fit)
```



```
predict_unseen <-predict(fit, train, type = 'class')
table_mat <- table(train$RESPONSE, predict_unseen)
table_mat
```

```
##    predict_unseen
##       0   1
##   0 104 108
##   1  22 479
```

```
accuracy_Train <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for train is', accuracy_Train))
```

```
## [1] "Accuracy for train is 0.817671809256662"
```

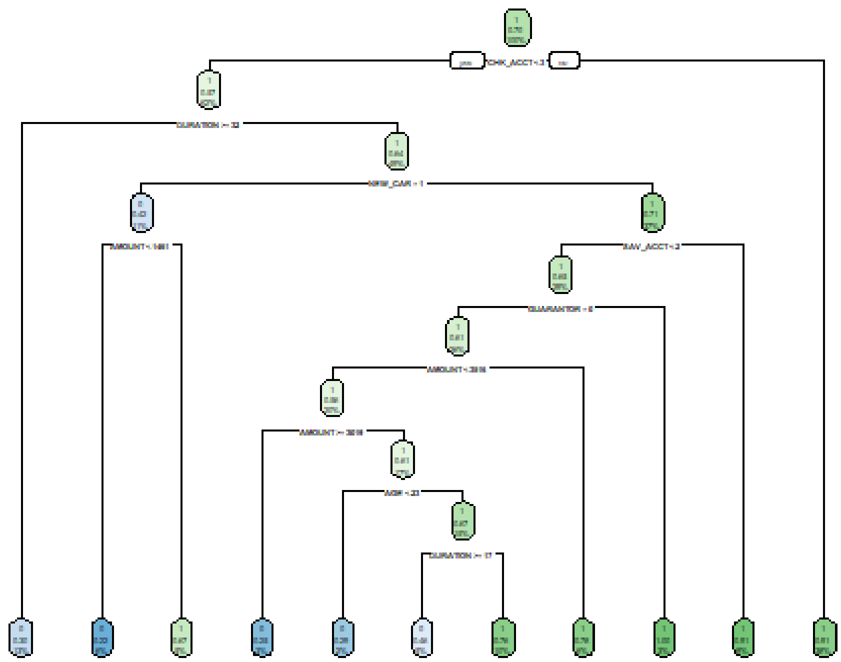
Simailarly, we calculate the accuracy for test data for 70-30 split

```r
set.seed(735)
library(rpart)
library(rpart.plot)
pacman::p_load("tidyverse","rio","pacman","readxl")
library(ISLR)
shuffle_index <- sample(1:nrow(Assignment_2))
Assignment_2 <- Assignment_2[shuffle_index, ]
indx <- sample(2, nrow(Assignment_2), replace= TRUE, prob = c(0.7, 0.3
))
train <- Assignment_2[indx == 1,  ]
test <- Assignment_2[indx == 2, ]
tree <- rpart(RESPONSE ~ ., train)
print(tree)
```

```
## n= 713
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 713 149.7700000 0.6998597
##     2) CHK_ACCT< 1.5 384  94.6224000 0.5598958
##       4) DURATION>=22.5 162  39.6111100 0.4259259
##         8) SAV_ACCT< 0.5 108  23.6574100 0.3240741
##          16) USED_CAR< 0.5 92  18.2065200 0.2717391 *
##          17) USED_CAR>=0.5 16   3.7500000 0.6250000 *
##         9) SAV_ACCT>=0.5 54  12.5925900 0.6296296 *
##       5) DURATION< 22.5 222  49.9819800 0.6576577
##        10) HISTORY< 1.5 22   4.7727270 0.3181818
##          20) NEW_CAR>=0.5 9   0.0000000 0.0000000 *
##          21) NEW_CAR< 0.5 13   3.2307690 0.5384615 *
##        11) HISTORY>=1.5 200  42.3950000 0.6950000
##          22) AMOUNT>=7442.5 7   0.8571429 0.1428571 *
##          23) AMOUNT< 7442.5 193  39.3264200 0.7150259
##            46) EDUCATION>=0.5 8   0.8750000 0.1250000 *
##            47) EDUCATION< 0.5 185  35.5459500 0.7405405
##              94) DURATION>=11.5 138  30.3260900 0.6739130
##               188) OBS#< 903.5 124  28.6693500 0.6370968 *
##               189) OBS#>=903.5 14   0.0000000 1.0000000 *
##              95) DURATION< 11.5 47   2.8085110 0.9361702 *
##     3) CHK_ACCT>=1.5 329  38.8449800 0.8632219
##       6) OTHER_INSTALL>=0.5 54  12.3148100 0.6481481 *
##       7) OTHER_INSTALL< 0.5 275  23.5418200 0.9054545 *
```

```r
library(rpart)
library(rpart.plot)
fit <- rpart(RESPONSE~., data = test, method = 'class', parms = list(s
```

```
plit ="gini"), control = rpart.control())
rpart.plot(fit)
```



```
predict_unseen <-predict(fit, test, type = 'class')
table_mat <- table(test$RESPONSE, predict_unseen)
table_mat

##    predict_unseen
##       0   1
##   0  58  28
##   1  25 176

accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test is', accuracy_Test))

## [1] "Accuracy for test is 0.815331010452962"
```

Now, for the split of 50-50 on train data;

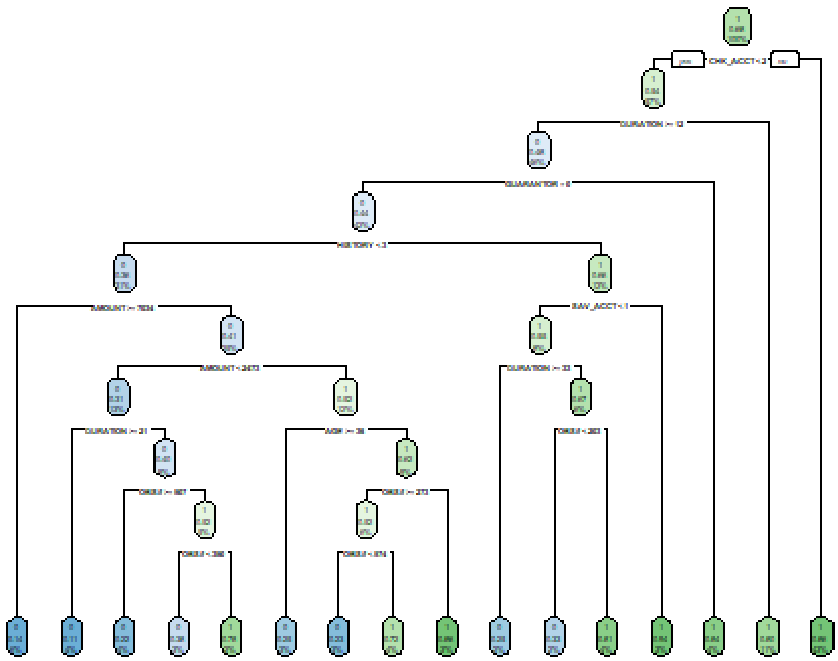
```
set.seed(735)
library(rpart)
library(rpart.plot)
pacman::p_load("tidyverse","rio","pacman","readxl")
library(ISLR)
shuffle_index <- sample(1:nrow(Assignment_2))
Assignment_2 <- Assignment_2[shuffle_index, ]
```

```
indx <- sample(2, nrow(Assignment_2), replace= TRUE, prob = c(0.5, 0.5
))
train <- Assignment_2[indx == 1,  ]
test <- Assignment_2[indx == 2, ]
tree <- rpart(RESPONSE ~ ., train)
print(tree)
```

```
## n= 495
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 495 107.927300 0.6787879
##     2) CHK_ACCT< 1.5 282  70.070920 0.5390071
##       4) DURATION>=11.5 228  56.890350 0.4780702
##         8) GUARANTOR< 0.5 209  51.617220 0.4449761
##          16) HISTORY< 2.5 151  34.966890 0.3642384
##            32) AMOUNT>=7033.5 28   3.428571 0.1428571 *
##            33) AMOUNT< 7033.5 123  29.853660 0.4146341
##              66) AMOUNT< 2472.5 62  13.177420 0.3064516
##               132) DURATION>=20.5 19   1.789474 0.1052632 *
##               133) DURATION< 20.5 43  10.279070 0.3953488 *
##              67) AMOUNT>=2472.5 61  15.213110 0.5245902
##               134) AGE>=35.5 16   3.000000 0.2500000 *
##               135) AGE< 35.5 45  10.577780 0.6222222
##                 270) OBS#>=273 31   7.741935 0.5161290
##                   540) OBS#< 574 13   2.307692 0.2307692 *
##                   541) OBS#>=574 18   3.611111 0.7222222 *
##                 271) OBS#< 273 14   1.714286 0.8571429 *
##          17) HISTORY>=2.5 58  13.103450 0.6551724
##            34) SAV_ACCT< 0.5 42  10.404760 0.5476190
##              68) DURATION>=33 12   2.250000 0.2500000 *
##              69) DURATION< 33 30   6.666667 0.6666667
##               138) OBS#< 203 9   2.000000 0.3333333 *
##               139) OBS#>=203 21   3.238095 0.8095238 *
##            35) SAV_ACCT>=0.5 16   0.937500 0.9375000 *
##         9) GUARANTOR>=0.5 19   2.526316 0.8421053 *
##       5) DURATION< 11.5 54   8.759259 0.7962963
##        10) REAL_ESTATE< 0.5 28   6.678571 0.6071429
##          20) NUM_CREDITS< 1.5 18   4.444444 0.4444444 *
##          21) NUM_CREDITS>=1.5 10   0.900000 0.9000000 *
##        11) REAL_ESTATE>=0.5 26   0.000000 1.0000000 *
##     3) CHK_ACCT>=1.5 213  25.051640 0.8638498
##       6) EMPLOYMENT< 1.5 37   7.729730 0.7027027
##        12) AMOUNT>=5309 7   1.428571 0.2857143 *
```

```
##         13) AMOUNT< 5309 30   4.800000 0.8000000 *
##        7) EMPLOYMENT>=1.5 176  16.159090 0.8977273 *
```

```
library(rpart)
library(rpart.plot)
fit <- rpart(RESPONSE~., data = train, method = 'class')
rpart.plot(fit)
```



```
predict_unseen <-predict(fit, train, type = 'class')
table_mat <- table(train$RESPONSE, predict_unseen)
table_mat
```

```
##    predict_unseen
##        0   1
##   0 102  57
##   1  29 307
```

```
accuracy_Train <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for train is', accuracy_Train))
```

```
## [1] "Accuracy for train is 0.826262626262626"
```

Now, for the split of 50-50 on test data;

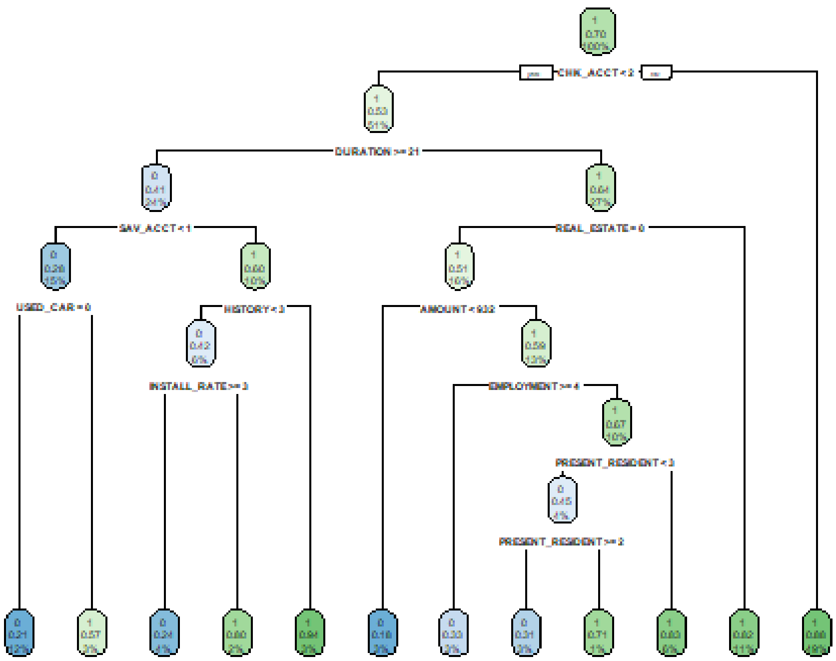
```
set.seed(735)
library(rpart)
```

```
library(rpart.plot)
pacman::p_load("tidyverse","rio","pacman","readxl")
library(ISLR)
shuffle_index <- sample(1:nrow(Assignment_2))
Assignment_2 <- Assignment_2[shuffle_index, ]
indx <- sample(2, nrow(Assignment_2), replace= TRUE, prob = c(0.5, 0.5
))
train <- Assignment_2[indx == 1,  ]
test <- Assignment_2[indx == 2, ]
tree <- rpart(RESPONSE ~ ., train)
print(tree)
```

```
## n= 495
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 495 103.3455000 0.7030303
##     2) CHK_ACCT< 2.5 308  74.6331200 0.5876623
##       4) HISTORY< 1.5 43   9.0697670 0.3023256
##         8) SAV_ACCT< 1.5 36   5.6388890 0.1944444 *
##         9) SAV_ACCT>=1.5 7   0.8571429 0.8571429 *
##       5) HISTORY>=1.5 265  61.4943400 0.6339623
##        10) AMOUNT>=8015.5 18   3.1111110 0.2222222 *
##        11) AMOUNT< 8015.5 247  55.1093100 0.6639676
##          22) OBS#>=405.5 154  37.2272700 0.5909091
##            44) DURATION>=15.5 78  19.3846200 0.4615385
##              88) OBS#< 868 57  12.9824600 0.3508772
##               176) SAV_ACCT< 2 47   9.4042550 0.2765957 *
##               177) SAV_ACCT>=2 10   2.1000000 0.7000000 *
##              89) OBS#>=868 21   3.8095240 0.7619048 *
##            45) DURATION< 15.5 76  15.1973700 0.7236842
##              90) AMOUNT< 1541.5 48  11.2500000 0.6250000
##               180) NEW_CAR>=0.5 18   4.2777780 0.3888889 *
##               181) NEW_CAR< 0.5 30   5.3666670 0.7666667
##                 362) INSTALL_RATE< 3.5 13   3.2307690 0.5384615 *
##                 363) INSTALL_RATE>=3.5 17   0.9411765 0.9411765 *
##              91) AMOUNT>=1541.5 28   2.6785710 0.8928571 *
##          23) OBS#< 405.5 93  15.6989200 0.7849462
##            46) INSTALL_RATE>=2.5 57  11.9298200 0.7017544
##              92) DURATION>=27 11   2.7272730 0.4545455 *
##              93) DURATION< 27 46   8.3695650 0.7608696
##               186) AMOUNT< 1384 20   4.9500000 0.5500000 *
##               187) AMOUNT>=1384 26   1.8461540 0.9230769 *
##            47) INSTALL_RATE< 2.5 36   2.7500000 0.9166667 *
##     3) CHK_ACCT>=2.5 187  17.8609600 0.8930481
```

```
##         6) OTHER_INSTALL>=0.5 35    7.5428570 0.6857143 *
##         7) OTHER_INSTALL< 0.5 152    8.4671050 0.9407895 *
```

```
library(rpart)
library(rpart.plot)
fit <- rpart(RESPONSE~., data = test, method = 'class')
rpart.plot(fit)
```




```
predict_unseen <-predict(fit, test, type = 'class')
table_mat <- table(test$RESPONSE, predict_unseen)
table_mat
```

```
##    predict_unseen
##       0   1
##   0  97  56
##   1  30 322
```

```
accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test is', accuracy_Test))
```

```
## [1] "Accuracy for test is 0.82970297029703"
```

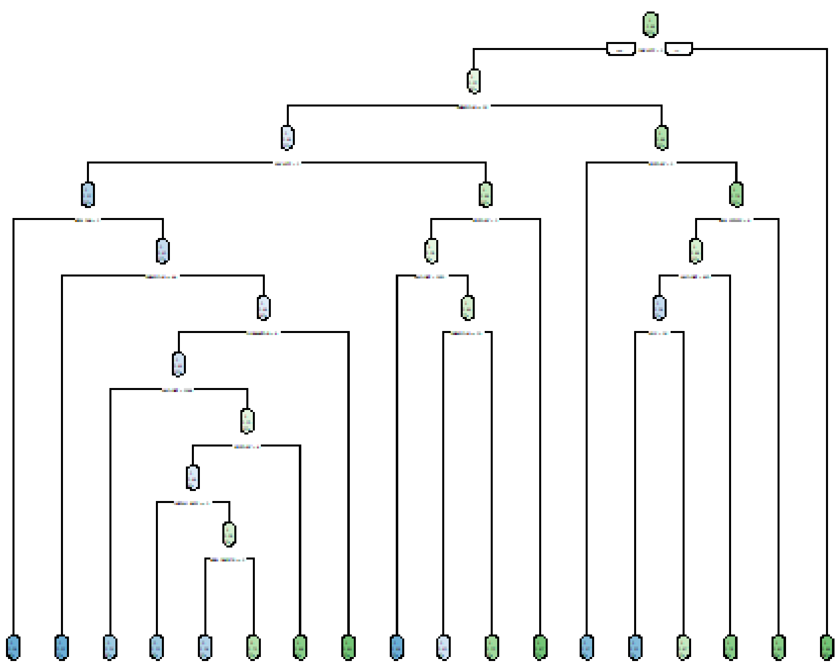
Simailarly, we calculate the accuracy for train data for 80-20 split

```
set.seed(735)
library(rpart)
```

```
library(rpart.plot)
pacman::p_load("tidyverse","rio","pacman","readxl")
library(ISLR)
shuffle_index <- sample(1:nrow(Assignment_2))
Assignment_2 <- Assignment_2[shuffle_index, ]
indx <- sample(2, nrow(Assignment_2), replace= TRUE, prob = c(0.8, 0.2
))
train <- Assignment_2[indx == 1,  ]
test <- Assignment_2[indx == 2, ]
tree <- rpart(RESPONSE ~ ., train)
print(tree)
```

```
## n= 810
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 810 172.4556000 0.69259260
##     2) CHK_ACCT< 1.5 438 108.3950000 0.55022830
##       4) DURATION>=15.5 257  63.7354100 0.45525290
##         8) SAV_ACCT< 0.5 171  38.9473700 0.35087720
##          16) NEW_CAR>=0.5 32   1.8750000 0.06250000 *
##          17) NEW_CAR< 0.5 139  33.7985600 0.41726620
##            34) DURATION>=43.5 22   0.9545455 0.04545455 *
##            35) DURATION< 43.5 117  29.2307700 0.48717950
##              70) GUARANTOR< 0.5 106  26.1603800 0.44339620
##               140) AMOUNT< 2209 29   4.7586210 0.20689660 *
##               141) AMOUNT>=2209 77  19.1688300 0.53246750
##                 282) HISTORY< 3.5 63  15.6507900 0.46031750 *
##                 283) HISTORY>=3.5 14   1.7142860 0.85714290 *
##              71) GUARANTOR>=0.5 11   0.9090909 0.90909090 *
##         9) SAV_ACCT>=0.5 86  19.2209300 0.66279070 *
##       5) DURATION< 15.5 181  39.0497200 0.68508290
##        10) HISTORY< 1.5 18   2.5000000 0.16666670 *
##        11) HISTORY>=1.5 163  31.1779100 0.74233130
##          22) REAL_ESTATE< 0.5 90  20.3222200 0.65555560
##            44) AMOUNT< 963 24   5.6250000 0.37500000 *
##            45) AMOUNT>=963 66  12.1212100 0.75757580 *
##          23) REAL_ESTATE>=0.5 73   9.3424660 0.84931510 *
##     3) CHK_ACCT>=1.5 372  44.7311800 0.86021510
##       6) EMPLOYMENT< 2.5 193  31.1191700 0.79792750
##        12) AMOUNT>=4558.5 36   8.7500000 0.58333330 *
##        13) AMOUNT< 4558.5 157  20.3312100 0.84713380 *
##       7) EMPLOYMENT>=2.5 179  12.0558700 0.92737430 *
```

```
library(rpart)
library(rpart.plot)
fit <- rpart(RESPONSE~., data = train, method = 'class')
rpart.plot(fit)
```



```
predict_unseen <-predict(fit, train, type = 'class')
table_mat <- table(train$RESPONSE, predict_unseen)
table_mat

##    predict_unseen
##       0   1
##   0 146 103
##   1  38 523

accuracy_Train <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for train is', accuracy_Train))

## [1] "Accuracy for train is 0.825925925925926"
```

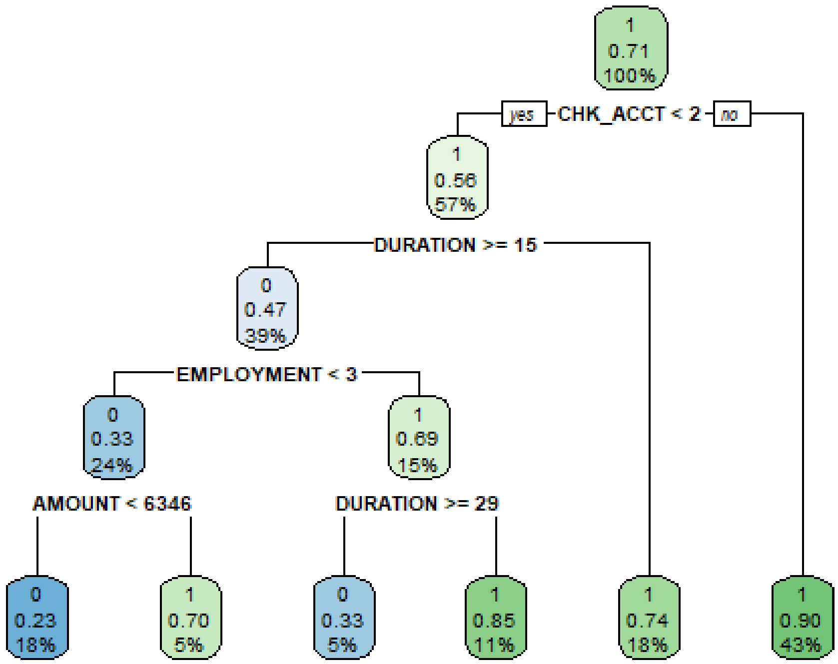
Simailarly, we calculate the accuracy for test data for 80-20 split

```
set.seed(735)
library(rpart)
library(rpart.plot)
pacman::p_load("tidyverse","rio","pacman","readxl")
library(ISLR)
```

```
shuffle_index <- sample(1:nrow(Assignment_2))
Assignment_2 <- Assignment_2[shuffle_index, ]
indx <- sample(2, nrow(Assignment_2), replace= TRUE, prob = c(0.8, 0.2
))
train <- Assignment_2[indx == 1,  ]
test <- Assignment_2[indx == 2, ]
tree <- rpart(RESPONSE ~ ., train)
print(tree)
```

```
## n= 810
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 810 170.498800 0.6987654
##     2) CHK_ACCT< 1.5 434 107.059900 0.5576037
##       4) DURATION>=22.5 188  45.957450 0.4255319
##         8) SAV_ACCT< 2.5 157  36.025480 0.3566879 *
##         9) SAV_ACCT>=2.5 31   5.419355 0.7741935
##          18) CHK_ACCT< 0.5 11   2.727273 0.4545455 *
##          19) CHK_ACCT>=0.5 20   0.950000 0.9500000 *
##       5) DURATION< 22.5 246  55.317070 0.6585366
##        10) HISTORY< 1.5 20   3.750000 0.2500000 *
##        11) HISTORY>=1.5 226  47.933630 0.6946903
##          22) OBS#>=120.5 197  44.213200 0.6598985
##            44) GUARANTOR< 0.5 172  40.436050 0.6220930
##              88) HISTORY< 2.5 115  28.643480 0.5304348
##               176) OBS#< 309.5 20   3.750000 0.2500000 *
##               177) OBS#>=309.5 95  22.989470 0.5894737
##                 354) OTHER_INSTALL>=0.5 13   2.307692 0.2307692 *
##                 355) OTHER_INSTALL< 0.5 82  18.743900 0.6463415 *
##              89) HISTORY>=2.5 57   8.877193 0.8070175 *
##            45) GUARANTOR>=0.5 25   1.840000 0.9200000 *
##          23) OBS#< 120.5 29   1.862069 0.9310345 *
##     3) CHK_ACCT>=1.5 376  44.808510 0.8617021 *
```

```
library(rpart)
library(rpart.plot)
fit <- rpart(RESPONSE~., data = test, method = 'class')
rpart.plot(fit)
```




```
predict_unseen <-predict(fit, test, type = 'class')
table_mat <- table(test$RESPONSE, predict_unseen)
table_mat

##    predict_unseen
##       0   1
##   0  33  23
##   1  11 123

accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test is', accuracy_Test))

## [1] "Accuracy for test is 0.821052631578947"
```

The accuracy for all the split i.e 70-30, 50-50 and 80-20 is approximately equal for training and test data. Therefore we can conclude that the model performance of all three is equal because we look for minimum difference of accuracy between the two dataset.

```
getwd()

## [1] "C:/Users/vsaih/OneDrive/Desktop/Data_Mining/Assignement-2/R-Se
ssions"

library(rpart)
 # for supervised models it is basically the same. We are just specify
```

```
ing the response and predict variable.
library(rattle)

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

#Created indexes based on the data imported and saved as data, this da
ta is divided into training and testing sets and analyzed.

data<-Assignment_2

#In order to calculate misclassification and performance,Created a mat
rix based on a question.

loss_mat<-matrix(c(0,100,500,0), byrow=TRUE, ncol=2)
View(loss_mat)


#In myFormula, assign RESPONSE~., which is the target variable.
myFormula<-RESPONSE~.

#Created indexes based on the data imported and saved as data, this da
ta is divided into training and testing sets and analyzed.

indx<-sample(2,nrow(data),replace=TRUE, prob=c(0.8,0.2))
indx

##    [1] 1 1 1 1 2 1 1 1 2 1 1 2 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1
1 2 1 1 2 1 2
##   [38] 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1
1 1 1 1 2 1 1
##   [75] 1 1 2 1 2 1 1 2 1 1 1 1 2 1 2 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1
1 1 1 1 1 1 1
##  [112] 2 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
1 1 1 1 1 1 1
##  [149] 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 1 2 1 1 1 1
##  [186] 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1
1 2 1 2 1 1 1
##  [223] 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
1 1 1 1 1 1 2
##  [260] 2 2 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 1 1 2 2 1 1 2 2 2 1 1 2 1
1 1 1 1 1 1 1
##  [297] 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 2 1
```

```
1 1 2 1 1 1 1
##  [334] 1 2 1 1 1 2 1 1 1 2 1 1 2 1 1 2 2 1 2 1 2 2 1 2 1 1 1 1 1 1
1 1 1 1 1 1 1
##  [371] 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1 2 1
2 1 1 1 1 1 1
##  [408] 1 1 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 2
2 1 1 1 2 2 1
##  [445] 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1
2 1 2 1 1 1 1
##  [482] 1 1 1 1 2 2 2 2 1 1 2 2 1 1 1 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1
1 1 1 1 1 2 1
##  [519] 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 2 1 1 1 2 2 1 2 1 1 1 1
1 2 1 1 1 1 1
##  [556] 1 2 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 1 1 1 2 1 1
##  [593] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 2 1
##  [630] 1 1 1 2 1 1 1 1 1 1 2 2 1 2 2 2 1 1 1 1 2 1 1 1 1 1 2 1 2 1
1 1 1 1 1 1 1
##  [667] 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1
1 1 1 1 1 1 1
##  [704] 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
1 1 1 2 1 2 2
##  [741] 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1
2 1 1 1 1 1 1
##  [778] 2 1 1 1 1 1 2 1 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1
1 1 1 1 1 1 1
##  [815] 1 1 2 1 2 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
2 1 1 1 1 2 1
##  [852] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 2
1 1 1 1 1 1 1
##  [889] 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 2 1 2 1
##  [926] 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 2 1 1 1 1
1 1 1 1 1 1 2
##  [963] 2 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 2 1 2 2 2 1 1 1 1
1 2 1 2 1 1 2
## [1000] 1
```

```
nrow(data)
```

```
## [1] 1000
```

```
train<-data[indx==1,]
test<-data[indx==2,]
view(train)
```

```
##Mytree _loss was developed in order to allocate what data to train a
nd to calculate information gain.

mytree_loss<-rpart(myFormula, data = train, method = "class", parms =
list(loss=loss_mat))

summary(mytree_loss)
```

```
## Call:
## rpart(formula = myFormula, data = train, method = "class", parms =
list(loss = loss_mat))
##   n= 799
##
##           CP nsplit rel error   xerror      xstd
## 1 0.0106383      0 1.0000000 5.000000 0.2740326
## 2 0.0100000      7 0.9021277 4.787234 0.2678144
##
## Variable importance
##         DURATION          CHK_ACCT          SAV_ACCT            AMOUNT
##               20                15                10                10
##          HISTORY       NUM_CREDITS PRESENT_RESIDENT           NEW_CAR
##                7                 6                 6                 5
##       EMPLOYMENT              OBS#               AGE      INSTALL_RATE
##                3                 3                 3                 2
##          OWN_RES    PROP_UNKN_NONE       MALE_SINGLE   MALE_MAR_or_WID
##                2                 2                 1                 1
##      REAL_ESTATE               JOB
##                1                 1
##
## Node number 1: 799 observations,    complexity param=0.0106383
##   predicted class=1  expected loss=29.41176  P(node) =1
##     class counts:   235   564
##    probabilities: 0.294 0.706
##   left son=2 (429 obs) right son=3 (370 obs)
##   Primary splits:
##       CHK_ACCT < 1.5     to the left,  improve=4.078775, (0 missing
)
##       HISTORY  < 1.5     to the left,  improve=2.233046, (0 missing
)
##       DURATION < 34.5    to the right, improve=1.771703, (0 missing
)
##       AMOUNT   < 10841.5 to the right, improve=1.643865, (0 missing
)
##       SAV_ACCT < 1.5     to the left,  improve=1.544164, (0 missing
)
```

```
##   Surrogate splits:
##       SAV_ACCT   < 1.5     to the left,  agree=0.620, adj=0.178, (0
split)
##       HISTORY    < 3.5     to the left,  agree=0.586, adj=0.105, (0
split)
##       DURATION   < 15.5    to the right, agree=0.559, adj=0.049, (0
split)
##       AGE        < 30.5    to the left,  agree=0.558, adj=0.046, (0
split)
##       EMPLOYMENT < 3.5     to the left,  agree=0.553, adj=0.035, (0
split)
##
## Node number 2: 429 observations,    complexity param=0.0106383
##   predicted class=1  expected loss=43.12354  P(node) =0.5369212
##     class counts:   185   244
##    probabilities: 0.431 0.569
##   left son=4 (194 obs) right son=5 (235 obs)
##   Primary splits:
##       DURATION       < 22.5    to the right, improve=2.364878, (0 m
issing)
##       HISTORY        < 1.5     to the left,  improve=2.008322, (0 m
issing)
##       AMOUNT         < 12296.5 to the right, improve=1.605905, (0 m
issing)
##       SAV_ACCT       < 1.5     to the left,  improve=1.605256, (0 m
issing)
##       PROP_UNKN_NONE < 0.5     to the right, improve=1.397851, (0 m
issing)
##   Surrogate splits:
##       AMOUNT         < 2665    to the right, agree=0.739, adj=0.423
, (0 split)
##       PROP_UNKN_NONE < 0.5     to the right, agree=0.639, adj=0.201
, (0 split)
##       REAL_ESTATE    < 0.5     to the left,  agree=0.604, adj=0.124
, (0 split)
##       HISTORY        < 1.5     to the left,  agree=0.592, adj=0.098
, (0 split)
##       JOB            < 2.5     to the right, agree=0.592, adj=0.098
, (0 split)
##
## Node number 3: 370 observations
##   predicted class=1  expected loss=13.51351  P(node) =0.4630788
##     class counts:    50   320
##    probabilities: 0.135 0.865
##
## Node number 4: 194 observations,    complexity param=0.0106383
```

```
##   predicted class=1  expected loss=56.18557  P(node) =0.2428035
##     class counts:   109    85
##    probabilities: 0.562 0.438
##   left son=8 (130 obs) right son=9 (64 obs)
##   Primary splits:
##       SAV_ACCT     < 0.5     to the left,  improve=1.8981500, (0 mi
ssing)
##       AMOUNT       < 1381.5  to the left,  improve=1.6393170, (0 mi
ssing)
##       USED_CAR     < 0.5     to the left,  improve=0.9419499, (0 mi
ssing)
##       INSTALL_RATE < 2.5     to the right, improve=0.9400633, (0 mi
ssing)
##       AGE          < 25.5    to the left,  improve=0.8639996, (0 mi
ssing)
##   Surrogate splits:
##       AMOUNT < 1412    to the right, agree=0.68, adj=0.031, (0 spli
t)
##
## Node number 5: 235 observations
##   predicted class=1  expected loss=32.34043  P(node) =0.2941176
##     class counts:    76   159
##    probabilities: 0.323 0.677
##
## Node number 8: 130 observations,    complexity param=0.0106383
##   predicted class=1  expected loss=65.38462  P(node) =0.1627034
##     class counts:    85    45
##    probabilities: 0.654 0.346
##   left son=16 (22 obs) right son=17 (108 obs)
##   Primary splits:
##       DURATION < 47.5    to the right, improve=2.676149, (0 missing
)
##       AMOUNT   < 11141   to the right, improve=2.554141, (0 missing
)
##       HISTORY  < 1.5     to the left,  improve=1.823319, (0 missing
)
##       NEW_CAR  < 0.5     to the right, improve=1.399969, (0 missing
)
##       USED_CAR < 0.5     to the left,  improve=1.353272, (0 missing
)
##   Surrogate splits:
##       AMOUNT < 13303   to the right, agree=0.854, adj=0.136, (0 spl
it)
##
## Node number 9: 64 observations
##   predicted class=1  expected loss=37.5  P(node) =0.08010013
```

```
##     class counts:    24    40
##    probabilities: 0.375 0.625
##
## Node number 16: 22 observations,    complexity param=0.0106383
##   predicted class=0  expected loss=45.45455  P(node) =0.02753442
##     class counts:    20     2
##    probabilities: 0.909 0.091
##   left son=32 (14 obs) right son=33 (8 obs)
##   Primary splits:
##       PRESENT_RESIDENT < 3.5     to the right, improve=1.5256410, (
0 missing)
##       OBS#             < 676.5   to the right, improve=0.8717949, (
0 missing)
##       INSTALL_RATE     < 3.5     to the left,  improve=0.8717949, (
0 missing)
##       CHK_ACCT         < 0.5     to the right, improve=0.7472527, (
0 missing)
##       AMOUNT           < 7580.5  to the right, improve=0.6340326, (
0 missing)
##   Surrogate splits:
##       INSTALL_RATE < 1.5     to the right, agree=0.773, adj=0.375,
(0 split)
##       AGE          < 35      to the right, agree=0.773, adj=0.375,
(0 split)
##       OWN_RES      < 0.5     to the left,  agree=0.773, adj=0.375,
(0 split)
##       AMOUNT       < 8861.5  to the left,  agree=0.727, adj=0.250,
(0 split)
##       MALE_SINGLE  < 0.5     to the right, agree=0.727, adj=0.250,
(0 split)
##
## Node number 17: 108 observations,    complexity param=0.0106383
##   predicted class=1  expected loss=60.18519  P(node) =0.135169
##     class counts:    65    43
##    probabilities: 0.602 0.398
##   left son=34 (22 obs) right son=35 (86 obs)
##   Primary splits:
##       NEW_CAR      < 0.5     to the right, improve=1.3417600, (0 mi
ssing)
##       AMOUNT       < 2249    to the left,  improve=1.2640800, (0 mi
ssing)
##       USED_CAR     < 0.5     to the left,  improve=1.0668240, (0 mi
ssing)
##       HISTORY      < 1.5     to the left,  improve=1.0628320, (0 mi
ssing)
##       INSTALL_RATE < 2.5     to the right, improve=0.9620402, (0 mi
```

```
ssing)
##   Surrogate splits:
##       AMOUNT       < 1494    to the left,  agree=0.852, adj=0.273,
(0 split)
##       CO-APPLICANT < 0.5     to the right, agree=0.815, adj=0.091,
(0 split)
##       OBS#         < 16.5    to the left,  agree=0.806, adj=0.045,
(0 split)
##
## Node number 32: 14 observations
##   predicted class=0  expected loss=0  P(node) =0.0175219
##     class counts:    14     0
##    probabilities: 1.000 0.000
##
## Node number 33: 8 observations
##   predicted class=1  expected loss=75  P(node) =0.01001252
##     class counts:     6     2
##    probabilities: 0.750 0.250
##
## Node number 34: 22 observations,    complexity param=0.0106383
##   predicted class=1  expected loss=81.81818  P(node) =0.02753442
##     class counts:    18     4
##    probabilities: 0.818 0.182
##   left son=68 (9 obs) right son=69 (13 obs)
##   Primary splits:
##       NUM_CREDITS  < 1.5     to the right, improve=1.7087810, (0 mi
ssing)
##       DURATION     < 33      to the left,  improve=1.1050380, (0 mi
ssing)
##       INSTALL_RATE < 3.5     to the right, improve=0.8675535, (0 mi
ssing)
##       AGE          < 34.5    to the left,  improve=0.6608852, (0 mi
ssing)
##       AMOUNT       < 3034.5  to the left,  improve=0.6608852, (0 mi
ssing)
##   Surrogate splits:
##       HISTORY         < 2.5     to the right, agree=0.864, adj=0.66
7, (0 split)
##       OBS#            < 170.5   to the left,  agree=0.773, adj=0.44
4, (0 split)
##       EMPLOYMENT      < 0.5     to the left,  agree=0.773, adj=0.44
4, (0 split)
##       AMOUNT          < 1200    to the left,  agree=0.682, adj=0.22
2, (0 split)
##       MALE_MAR_or_WID < 0.5     to the right, agree=0.682, adj=0.22
2, (0 split)
```

```
##
## Node number 35: 86 observations
##   predicted class=1  expected loss=54.65116  P(node) =0.1076345
##     class counts:    47    39
##    probabilities: 0.547 0.453
##
## Node number 68: 9 observations
##   predicted class=0  expected loss=0  P(node) =0.01126408
##     class counts:     9     0
##    probabilities: 1.000 0.000
##
## Node number 69: 13 observations
##   predicted class=1  expected loss=69.23077  P(node) =0.01627034
##     class counts:     9     4
##    probabilities: 0.692 0.308
```

```
# The predict() function can be used to predicted values, obtained by
evaluating the regression function in the frame newdata.
pred_train_loss<-predict(mytree_loss,data=train,type = "class")
table(pred_train_loss)
```

```
## pred_train_loss
##   0   1
##  23 776
```

```
# mean() is used to calculate the arithmetic mean of the elements of t
he numeric vector passed to it as argument.
mean(train$RESPONSE!=pred_train_loss)
```

```
## [1] 0.2653317
```

```
pred_test_loss<-predict(mytree_loss,newdata = test,type="class")
table(pred_test_loss)
```

```
## pred_test_loss
##   0   1
##   2 199
```

```
mean(test$RESPONSE!=pred_test_loss)
```

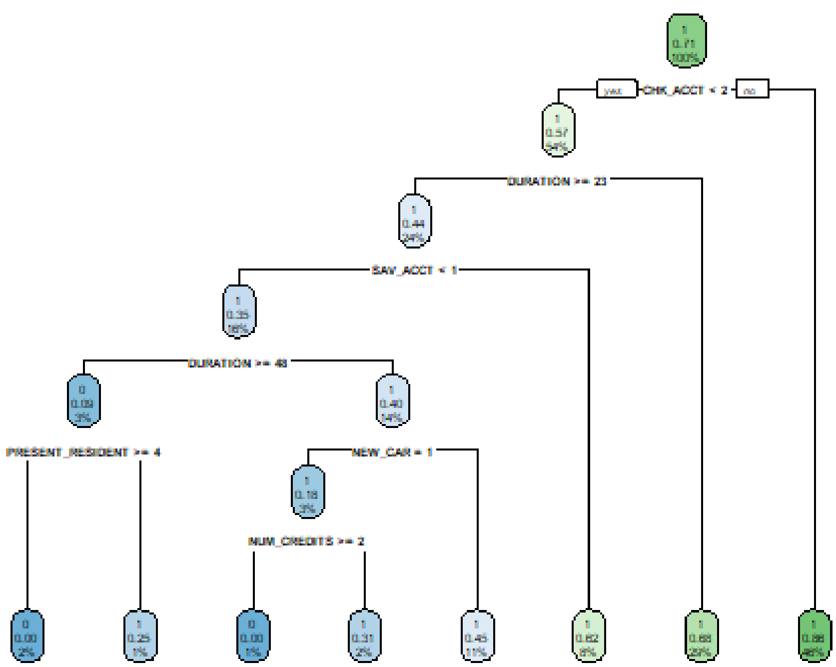
```
## [1] 0.3134328
```

```
print(mytree_loss)
```

```
## n= 799
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
```

```
##  1) root 799 23500 1 (0.29411765 0.70588235)
##    2) CHK_ACCT< 1.5 429 18500 1 (0.43123543 0.56876457)
##      4) DURATION>=22.5 194 10900 1 (0.56185567 0.43814433)
##        8) SAV_ACCT< 0.5 130  8500 1 (0.65384615 0.34615385)
##         16) DURATION>=47.5 22  1000 0 (0.90909091 0.09090909)
##           32) PRESENT_RESIDENT>=3.5 14     0 0 (1.00000000 0.000000
00) *
##           33) PRESENT_RESIDENT< 3.5 8   600 1 (0.75000000 0.2500000
0) *
##         17) DURATION< 47.5 108  6500 1 (0.60185185 0.39814815)
##           34) NEW_CAR>=0.5 22  1800 1 (0.81818182 0.18181818)
##             68) NUM_CREDITS>=1.5 9     0 0 (1.00000000 0.00000000)
*
##             69) NUM_CREDITS< 1.5 13   900 1 (0.69230769 0.30769231)
*
##           35) NEW_CAR< 0.5 86  4700 1 (0.54651163 0.45348837) *
##        9) SAV_ACCT>=0.5 64  2400 1 (0.37500000 0.62500000) *
##      5) DURATION< 22.5 235  7600 1 (0.32340426 0.67659574) *
##    3) CHK_ACCT>=1.5 370  5000 1 (0.13513514 0.86486486) *

library(rpart.plot)
rpart.plot(mytree_loss)
```



d) For CHK_ACCT<3,DURATION>=44,HISTORY<2, AMOUNT>=7974 => Good Credit

CHK_ACCT<3 and DURATION < 44 months => Bad Credit

CHK_ACCT<3 and DURATION >= 44 months => Bad Credit

(e)1) To summarize our findings, we found that when there is no checking account the credit score is good 88 percent of the time(9/10 in proportion) 2) When we use ggplot to plot the DURATION We get a few outliers hence converting it into a categorical variable would be the best option. 3) The maximum number of applicants listed are in ages less than 45.

4)When we split the data into training and test (For (b) question) (i) For an 80-20 split we get 82% Train and 83% test accuracy for them respectively. (ii) For an 70-30 split we get 81% Train and 82% test accuracy for them respectively. (iii) For an 50-50 split we get 84% Train and 83% test accuracy for them respectively.

5)Data is segregated into train and test with probability of 80:20,Based on predicted test case mean calculated error and accuracy around 70%.