Meghashree Maddihally Nagoji

2/22/2022

Problem a)

Reading data from the excel.

```
data <- read_excel("German Credit.xls")
```

Exploring the data.

```
str(data)

## tibble [1,000 x 32] (S3: tbl_df/tbl/data.frame)
##  $ OBS#           : num [1:1000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ CHK_ACCT       : num [1:1000] 0 1 3 0 0 3 3 1 3 1 ...
##  $ DURATION       : num [1:1000] 6 48 12 42 24 36 24 36 12 30 ...
##  $ HISTORY        : num [1:1000] 4 2 4 2 3 2 2 2 2 4 ...
##  $ NEW_CAR        : num [1:1000] 0 0 0 0 1 0 0 0 0 1 ...
##  $ USED_CAR       : num [1:1000] 0 0 0 0 0 0 0 1 0 0 ...
##  $ FURNITURE      : num [1:1000] 0 0 0 1 0 0 1 0 0 0 ...
##  $ RADIO/TV       : num [1:1000] 1 1 0 0 0 0 0 0 1 0 ...
##  $ EDUCATION      : num [1:1000] 0 0 1 0 0 1 0 0 0 0 ...
##  $ RETRAINING     : num [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ AMOUNT         : num [1:1000] 1169 5951 2096 7882 4870 ...
##  $ SAV_ACCT       : num [1:1000] 4 0 0 0 0 4 2 0 3 0 ...
##  $ EMPLOYMENT     : num [1:1000] 4 2 3 3 2 2 4 2 3 0 ...
##  $ INSTALL_RATE   : num [1:1000] 4 2 2 2 3 2 3 2 2 4 ...
##  $ MALE_DIV       : num [1:1000] 0 0 0 0 0 0 0 0 1 0 ...
##  $ MALE_SINGLE    : num [1:1000] 1 0 1 1 1 1 1 1 0 0 ...
##  $ MALE_MAR_or_WID : num [1:1000] 0 0 0 0 0 0 0 0 0 1 ...
##  $ CO-APPLICANT   : num [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ GUARANTOR      : num [1:1000] 0 0 0 1 0 0 0 0 0 0 ...
##  $ PRESENT_RESIDENT: num [1:1000] 4 2 3 4 4 4 4 2 4 2 ...
##  $ REAL_ESTATE    : num [1:1000] 1 1 1 0 0 0 0 0 1 0 ...
##  $ PROP_UNKN_NONE : num [1:1000] 0 0 0 0 1 1 0 0 0 0 ...
##  $ AGE            : num [1:1000] 67 22 49 45 53 35 53 35 61 28 ...
##  $ OTHER_INSTALL  : num [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ RENT           : num [1:1000] 0 0 0 0 0 0 0 1 0 0 ...
##  $ OWN_RES        : num [1:1000] 1 1 1 0 0 0 1 0 1 1 ...
##  $ NUM_CREDITS    : num [1:1000] 2 1 1 1 2 1 1 1 1 2 ...
##  $ JOB            : num [1:1000] 2 2 1 2 2 1 2 3 1 3 ...
##  $ NUM_DEPENDENTS : num [1:1000] 1 1 2 2 2 2 1 1 1 1 ...
##  $ TELEPHONE      : num [1:1000] 1 0 0 0 0 1 0 1 0 0 ...
##  $ FOREIGN        : num [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ RESPONSE       : num [1:1000] 1 0 1 1 0 1 1 1 1 0 ...

dim(data)

## [1] 1000   32
```

```
names(data)

##  [1] "OBS#"              "CHK_ACCT"         "DURATION"         "HISTORY"
##  [5] "NEW_CAR"           "USED_CAR"         "FURNITURE"        "RADIO/TV"
##  [9] "EDUCATION"         "RETRAINING"       "AMOUNT"           "SAV_ACCT"
## [13] "EMPLOYMENT"        "INSTALL_RATE"     "MALE_DIV"
"MALE_SINGLE"
## [17] "MALE_MAR_or_WID"  "CO-APPLICANT"     "GUARANTOR"
"PRESENT_RESIDENT"
## [21] "REAL_ESTATE"       "PROP_UNKN_NONE"   "AGE"
"OTHER_INSTALL"
## [25] "RENT"              "OWN_RES"          "NUM_CREDITS"      "JOB"
## [29] "NUM_DEPENDENTS"    "TELEPHONE"        "FOREIGN"          "RESPONSE"

summary(data)

##       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
EDUCATION
##  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   Median
:0.00
##  Mean   :0.234   Mean   :0.103   Mean   :0.181   Mean   :0.28   Mean
: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   Median
:0.000
##  Mean   :2.973   Mean   :0.05   Mean   :0.548   Mean   :0.092   Mean
: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
```

```
head(data)
```

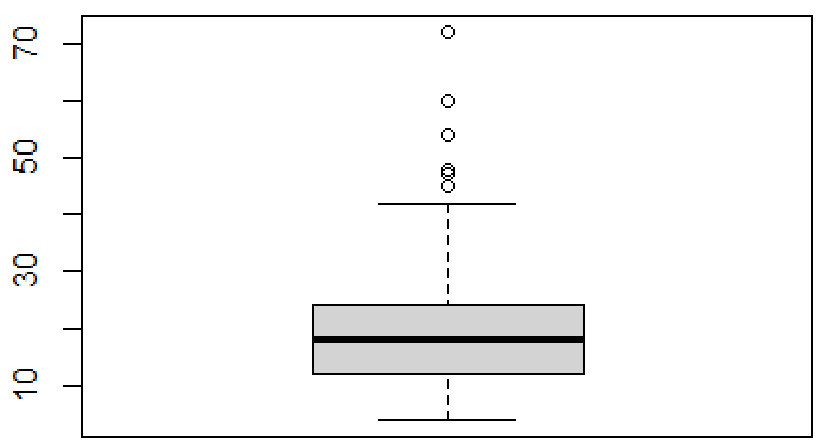
```
## # A tibble: 6 x 32
##   `OBS#` CHK_ACCT DURATION HISTORY NEW_CAR USED_CAR FURNITURE `RADIO/TV`
##    <dbl>    <dbl>    <dbl>   <dbl>   <dbl>    <dbl>     <dbl>      <dbl>
## 1      1        0        6       4       0        0         0          1
## 2      2        1       48       2       0        0         0          1
## 3      3        3       12       4       0        0         0          0
## 4      4        0       42       2       0        0         1          0
## 5      5        0       24       3       1        0         0          0
## 6      6        3       36       2       0        0         0          0
## # ... with 24 more variables: EDUCATION <dbl>, RETRAINING <dbl>, AMOUNT
<dbl>,
```

```
## #   SAV_ACCT <dbl>, EMPLOYMENT <dbl>, INSTALL_RATE <dbl>, MALE_DIV <dbl>,
## #   MALE_SINGLE <dbl>, MALE_MAR_or_WID <dbl>, CO-APPLICANT <dbl>,
## #   GUARANTOR <dbl>, PRESENT_RESIDENT <dbl>, REAL_ESTATE <dbl>,
## #   PROP_UNKN_NONE <dbl>, AGE <dbl>, OTHER_INSTALL <dbl>, RENT <dbl>,
## #   OWN_RES <dbl>, NUM_CREDITS <dbl>, JOB <dbl>, NUM_DEPENDENTS <dbl>,
## #   TELEPHONE <dbl>, FOREIGN <dbl>, RESPONSE <dbl>
```

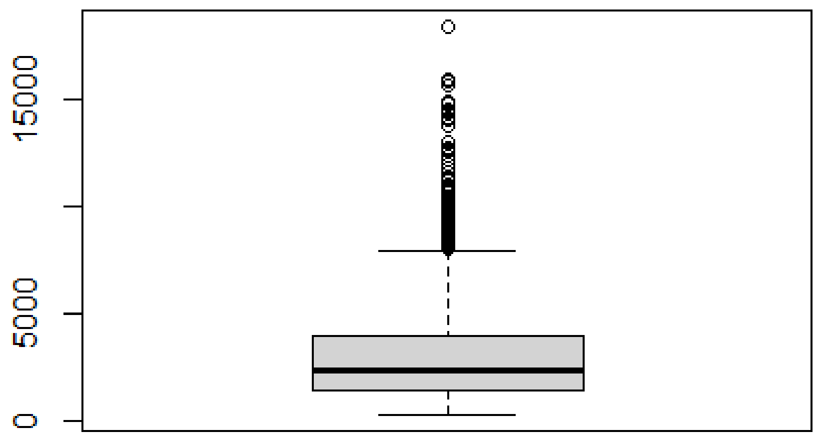
Finding the outliers in the numerical attributes.

Creating boxplots to identify the outliers.

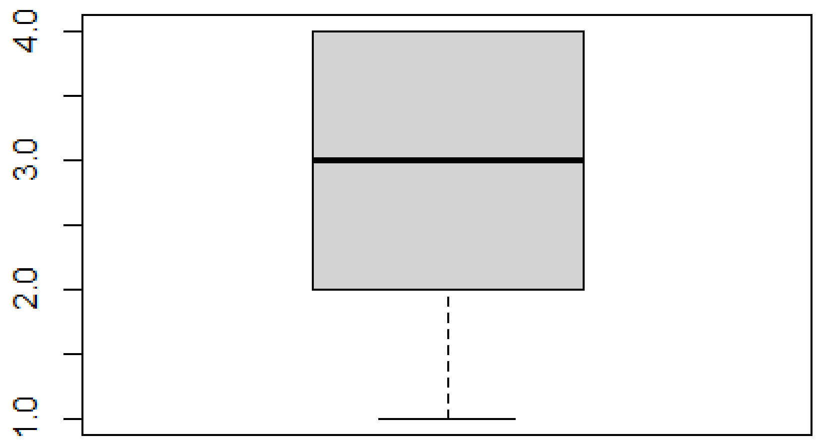
```
boxplot(data$DURATION)
boxplot(data$DURATION)$out
```



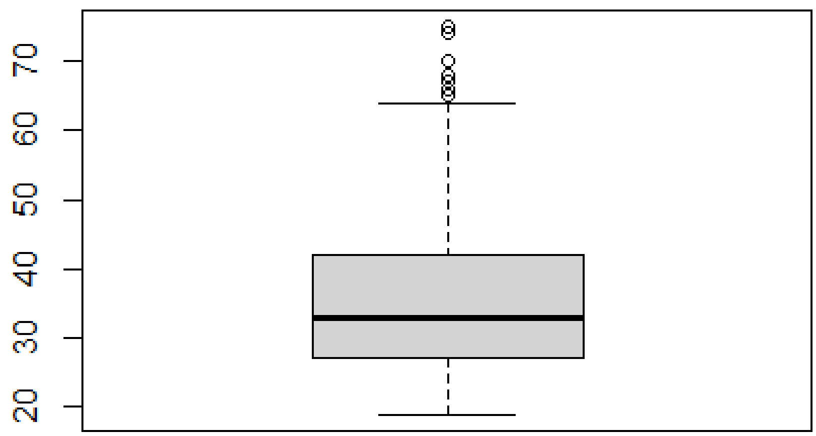

```
##  [1] 48 48 60 45 48 48 48 54 54 48 48 60 48 48 45 48 48 60 48 48 47 48 48
48 48
## [26] 48 48 60 48 60 60 48 48 48 48 48 48 48 48 48 48 60 48 60 48 48 48 60
72 60
## [51] 48 48 60 48 48 48 48 48 48 45 48 48 48 48 60 48 60 48 45 45

boxplot(data$AMOUNT)
boxplot(data$AMOUNT)$out
```



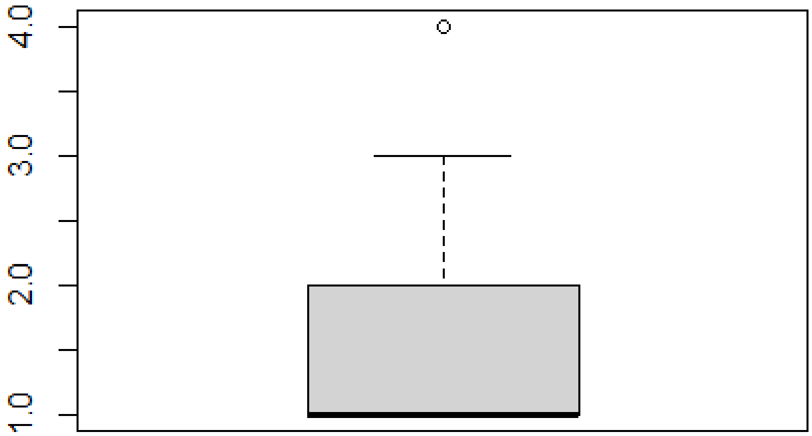

```
##  [1]  9055  8072 12579  9566 14421  8133  9436 12612 15945 11938  8487
10144
## [13]  8613  9572 10623 10961 14555  8978 12169 11998 10722  9398  9960
10127
## [25] 11590 13756 14782 14318 12976 11760  8648  8471 11328 11054  8318
9034
## [37]  8588  7966  8858 12389 12204  9157 15653  7980  8086 10222 10366
9857
## [49] 14027 11560 14179 12680  8065  9271  9283  9629 15857  8335 11816
10875
## [61]  9277 15672  8947 10477 18424 14896 12749 10297  8358 10974  8386
8229

boxplot(data$INSTALL_RATE)
boxplot(data$INSTALL_RATE)$out
```



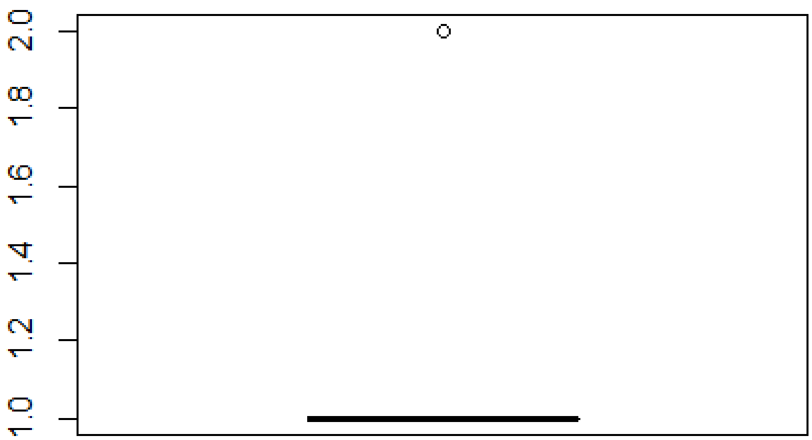

```
## numeric(0)

boxplot(data$AGE)
boxplot(data$AGE)$out
```




```
##  [1] 67 66 66 70 65 74 68 66 75 74 65 75 67 74 65 66 74 66 67 65 68 65 68

boxplot(data$NUM_CREDITS)
boxplot(data$NUM_CREDITS)$out
```




```
## [1] 4 4 4 4 4 4

boxplot(data$NUM_DEPENDENTS)
boxplot(data$NUM_DEPENDENTS)$out
```




```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [149] 2 2 2 2 2 2 2
```

There are outliers observed. There are only two values in the 'NUM_DEPENDENTS' variable . Hence, ignoring this variable for the boxplot.

Proportion of good to bad cases.

```
Good_Bad <-as.factor(ifelse(data$RESPONSE == 1, "Good", "Bad"))
tbl <- table(Good_Bad)
tbl_pct <- cbind(tbl,round(prop.table(tbl)*100,2))
colnames(tbl_pct) <- c('Count','Percentage')
knitr::kable(tbl_pct)
```

	Count	Percentage
Bad	300	30
Good	700	70

convert the required variables to categorical values.

```
data$`OBS#` <- as.factor(data$`OBS#`)
data$CHK_ACCT <- as.factor(data$CHK_ACCT)
data$HISTORY <- as.factor(data$HISTORY)
data$NEW_CAR <- as.factor(data$NEW_CAR)
data$USED_CAR <- as.factor(data$USED_CAR)
data$FURNITURE <- as.factor(data$FURNITURE)
data$`RADIO/TV` <- as.factor(data$`RADIO/TV`)
data$EDUCATION <- as.factor(data$EDUCATION)
data$RETRAINING <- as.factor(data$RETRAINING)
data$SAV_ACCT <- as.factor(data$SAV_ACCT)
data$EMPLOYMENT <- as.factor(data$EMPLOYMENT)
data$MALE_DIV <- as.factor(data$MALE_DIV)
data$MALE_SINGLE <- as.factor(data$MALE_SINGLE)
data$MALE_MAR_or_WID <- as.factor(data$MALE_MAR_or_WID)
data$`CO-APPLICANT` <- as.factor(data$`CO-APPLICANT`)
data$GUARANTOR <- as.factor(data$GUARANTOR)
data$PRESENT_RESIDENT <- as.factor(data$PRESENT_RESIDENT)
data$REAL_ESTATE <- as.factor(data$REAL_ESTATE)
data$PROP_UNKN_NONE <- as.factor(data$PROP_UNKN_NONE)
data$OTHER_INSTALL <- as.factor(data$OTHER_INSTALL)
data$RENT <- as.factor(data$RENT)
data$OWN_RES <- as.factor(data$OWN_RES)
data$JOB <- as.factor(data$JOB)
data$TELEPHONE <- as.factor(data$TELEPHONE)
data$FOREIGN <- as.factor(data$FOREIGN)
```

Descriptions of the predictor (independent) variables - mean of numerical attributes and frequencies of categorical attributes.

```
describe(data)

## data
##
##  32  Variables      1000  Observations
## ---------------------------------------------------------------------------
------
## OBS#
##        n  missing distinct
##     1000        0     1000
##
## lowest : 1    2    3    4    5   , highest: 996  997  998  999  1000
## ---------------------------------------------------------------------------
------
## CHK_ACCT
##        n  missing distinct
##     1000        0        4
##
## Value          0     1     2     3
## Frequency    274   269    63   394
## Proportion 0.274 0.269 0.063 0.394
```

```
## ---------------------------------------------------------------------------
------
## DURATION
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1000        0       33    0.985     20.9    12.98        6        9
##      .25      .50      .75      .90      .95
##       12       18       24       36       48
##
## lowest :  4  5  6  7  8, highest: 47 48 54 60 72
## ---------------------------------------------------------------------------
------
## HISTORY
##        n  missing distinct
##     1000        0        5
##
## lowest : 0 1 2 3 4, highest: 0 1 2 3 4
##
## Value          0     1     2     3     4
## Frequency     40    49   530    88   293
## Proportion 0.040 0.049 0.530 0.088 0.293
## ---------------------------------------------------------------------------
------
## NEW_CAR
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    766   234
## Proportion 0.766 0.234
## ---------------------------------------------------------------------------
------
## USED_CAR
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    897   103
## Proportion 0.897 0.103
## ---------------------------------------------------------------------------
------
## FURNITURE
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    819   181
## Proportion 0.819 0.181
## ---------------------------------------------------------------------------
------
## RADIO/TV
```

```
##        n  missing distinct 
##     1000        0        2 
## 
## Value         0    1
## Frequency   720  280
## Proportion 0.72 0.28
## ---------------------------------------------------------------------------
------
## EDUCATION 
##        n  missing distinct 
##     1000        0        2 
## 
## Value         0    1
## Frequency   950   50
## Proportion 0.95 0.05
## ---------------------------------------------------------------------------
------
## RETRAINING 
##        n  missing distinct 
##     1000        0        2 
## 
## Value          0     1
## Frequency    903    97
## Proportion 0.903 0.097
## ---------------------------------------------------------------------------
------
## AMOUNT 
##        n  missing distinct     Info     Mean      Gmd      .05      .10 
##     1000        0      921        1     3271     2773      709      932 
##      .25      .50      .75      .90      .95 
##     1366     2320     3972     7179     9163 
## 
## lowest :   250   276   338   339   343, highest: 15653 15672 15857 15945 
18424
## ---------------------------------------------------------------------------
------
## SAV_ACCT 
##        n  missing distinct 
##     1000        0        5 
## 
## lowest : 0 1 2 3 4, highest: 0 1 2 3 4
## 
## Value          0     1     2     3     4
## Frequency    603   103    63    48   183
## Proportion 0.603 0.103 0.063 0.048 0.183
## ---------------------------------------------------------------------------
------
## EMPLOYMENT 
##        n  missing distinct 
##     1000        0        5 
```

```
##
## lowest : 0 1 2 3 4, highest: 0 1 2 3 4
##
## Value          0     1     2     3     4
## Frequency     62   172   339   174   253
## Proportion 0.062 0.172 0.339 0.174 0.253
## ---------------------------------------------------------------------------
------
## INSTALL_RATE
##        n  missing distinct     Info     Mean      Gmd
##     1000        0        4    0.873    2.973      1.2
##
## Value          1     2     3     4
## Frequency    136   231   157   476
## Proportion 0.136 0.231 0.157 0.476
## ---------------------------------------------------------------------------
------
## MALE_DIV
##        n  missing distinct
##     1000        0        2
##
## Value         0    1
## Frequency   950   50
## Proportion 0.95 0.05
## ---------------------------------------------------------------------------
------
## MALE_SINGLE
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    452   548
## Proportion 0.452 0.548
## ---------------------------------------------------------------------------
------
## MALE_MAR_or_WID
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    908    92
## Proportion 0.908 0.092
## ---------------------------------------------------------------------------
------
## CO-APPLICANT
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    959    41
```

```
## Proportion 0.959 0.041
## ---------------------------------------------------------------------------
------
## GUARANTOR
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    948    52
## Proportion 0.948 0.052
## ---------------------------------------------------------------------------
------
## PRESENT_RESIDENT
##        n  missing distinct
##     1000        0        4
##
## Value          1     2     3     4
## Frequency    130   308   149   413
## Proportion 0.130 0.308 0.149 0.413
## ---------------------------------------------------------------------------
------
## REAL_ESTATE
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    718   282
## Proportion 0.718 0.282
## ---------------------------------------------------------------------------
------
## PROP_UNKN_NONE
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    846   154
## Proportion 0.846 0.154
## ---------------------------------------------------------------------------
------
## AGE
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     1000        0       53    0.999    35.55    12.41       22       23
##      .25      .50      .75      .90      .95
##       27       33       42       52       60
##
## lowest : 19 20 21 22 23, highest: 67 68 70 74 75
## ---------------------------------------------------------------------------
------
## OTHER_INSTALL
##        n  missing distinct
```

```
##     1000        0        2
##
## Value          0     1
## Frequency    814   186
## Proportion 0.814 0.186
## -----------------------------------------------------------------------------
------
## RENT
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    821   179
## Proportion 0.821 0.179
## -----------------------------------------------------------------------------
------
## OWN_RES
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    287   713
## Proportion 0.287 0.713
## -----------------------------------------------------------------------------
------
## NUM_CREDITS
##        n  missing distinct     Info     Mean      Gmd
##     1000        0        4    0.709    1.407   0.5428
##
## Value          1     2     3     4
## Frequency    633   333    28     6
## Proportion 0.633 0.333 0.028 0.006
## -----------------------------------------------------------------------------
------
## JOB
##        n  missing distinct
##     1000        0        4
##
## Value          0     1     2     3
## Frequency     22   200   630   148
## Proportion 0.022 0.200 0.630 0.148
## -----------------------------------------------------------------------------
------
## NUM_DEPENDENTS
##        n  missing distinct     Info     Mean      Gmd
##     1000        0        2    0.393    1.155   0.2622
##
## Value          1     2
## Frequency    845   155
## Proportion 0.845 0.155
```

```
## ------------------------------------------------------------------------------
------
## TELEPHONE
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    596   404
## Proportion 0.596 0.404
## ------------------------------------------------------------------------------
------
## FOREIGN
##        n  missing distinct
##     1000        0        2
##
## Value          0     1
## Frequency    963    37
## Proportion 0.963 0.037
## ------------------------------------------------------------------------------
------
## RESPONSE
##        n  missing distinct     Info      Sum     Mean      Gmd
##     1000        0        2     0.63      700      0.7   0.4204
##
## ------------------------------------------------------------------------------
------
```

Standard deviation of numerical attributes.

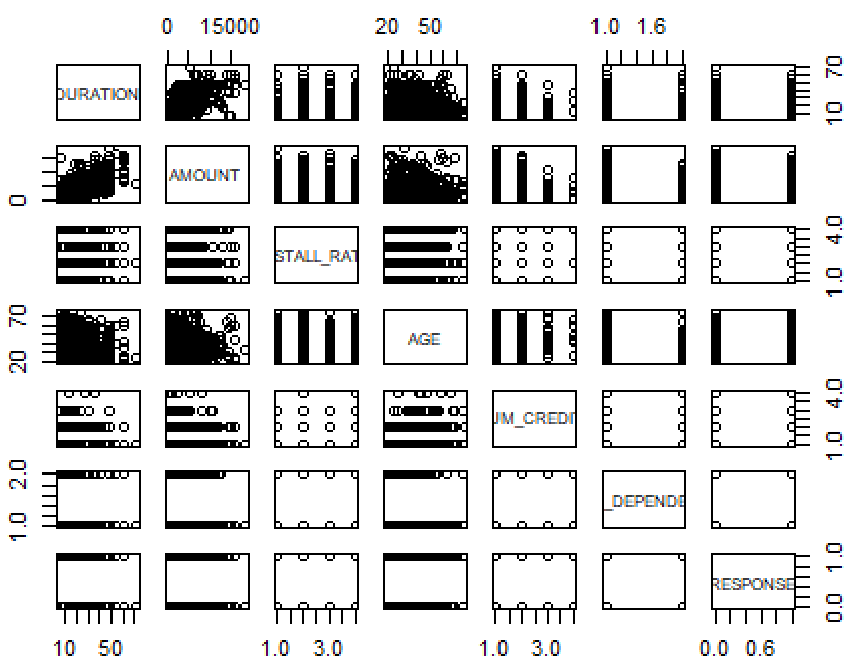
```
sd(data$DURATION)

## [1] 12.05881

sd(data$AMOUNT)

## [1] 2822.737

sd(data$INSTALL_RATE)

## [1] 1.118715

sd(data$AGE)

## [1] 11.37547

sd(data$NUM_CREDITS)

## [1] 0.5776545

sd(data$NUM_DEPENDENTS)

## [1] 0.3620858
```

Finding correlation between numerical variables and target variable - RESPONSE.

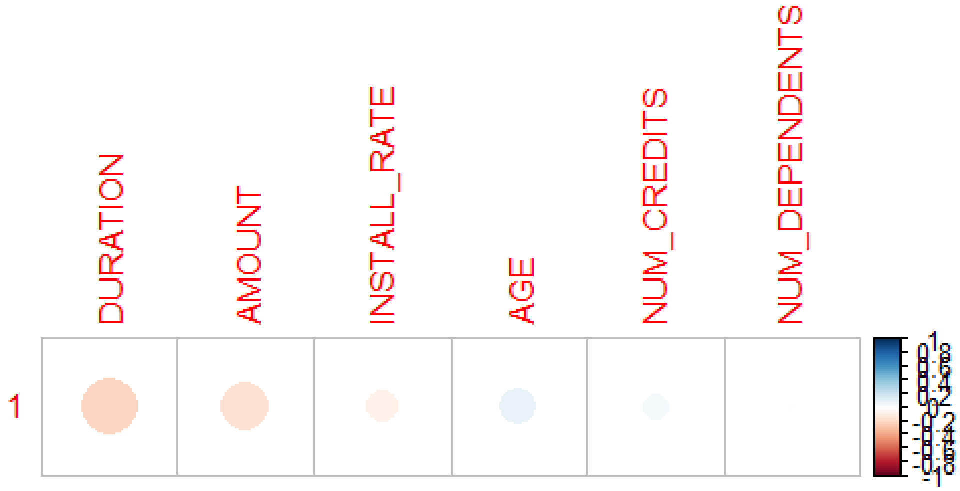
```
cors <- cor(data[,(colnames(data) %in% c('DURATION',
'AMOUNT','INSTALL_RATE','AGE','NUM_CREDITS','NUM_DEPENDENTS'))],
data[,ncol(data)])
kable(cors,col.names=c('RESPONSE'))
```

	RESPONSE
DURATION	-0.2149267
AMOUNT	-0.1547386
INSTALL_RATE	-0.0724039
AGE	0.0911274
NUM_CREDITS	0.0457325
NUM_DEPENDENTS	0.0030149

```
plot(data[,(colnames(data) %in% c('DURATION',
'AMOUNT','INSTALL_RATE','AGE','NUM_CREDITS','NUM_DEPENDENTS', 'RESPONSE'))])
```

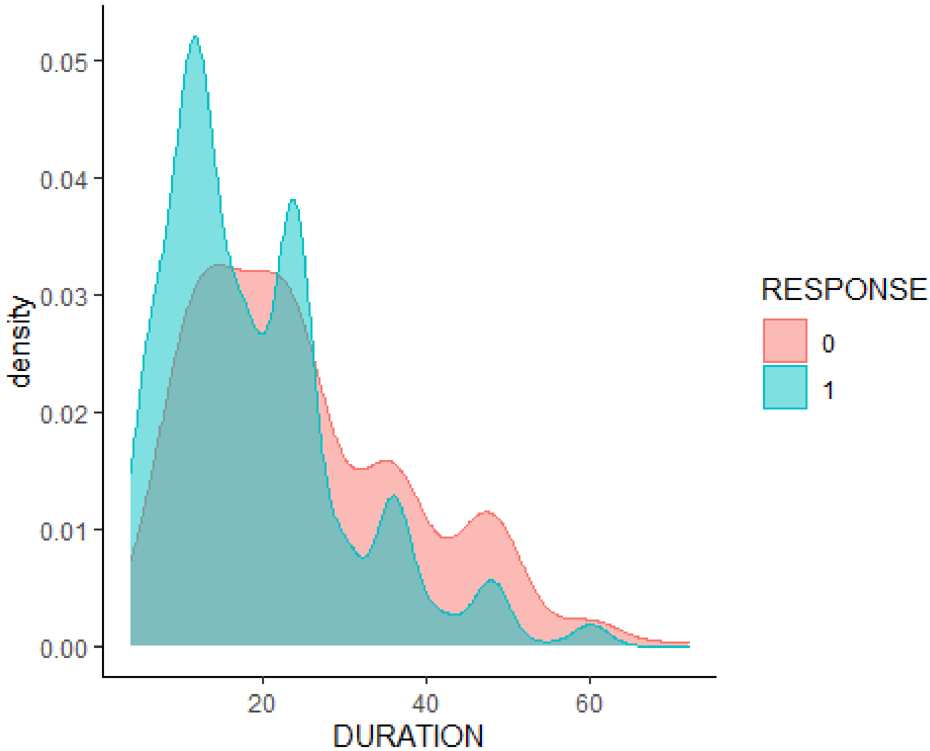



```
corrplot::corrplot(cor(data$RESPONSE,data[,(colnames(data) %in% c('DURATION',
'AMOUNT','INSTALL_RATE','AGE','NUM_CREDITS','NUM_DEPENDENTS'))]))
```



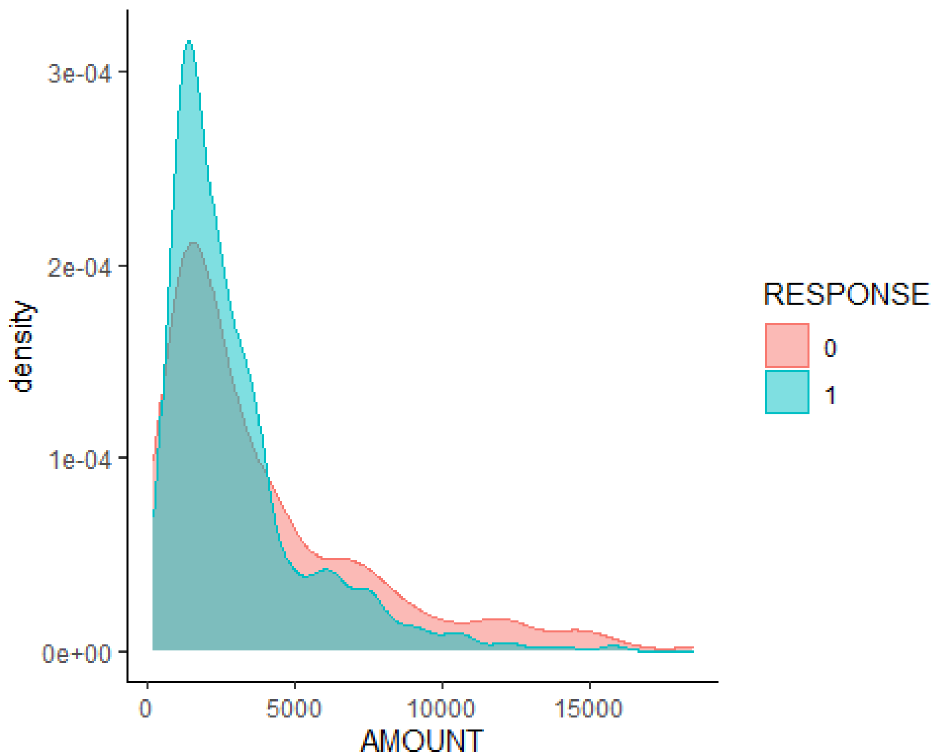
Density Graphs - Finding correlation between numerical variables and target variable - RESPONSE.

```
data$RESPONSE <- as.factor(data$RESPONSE)

ggplot(data)+
  geom_density(aes(x=DURATION,color=RESPONSE,fill=RESPONSE),alpha=0.5) +
theme_classic()
```



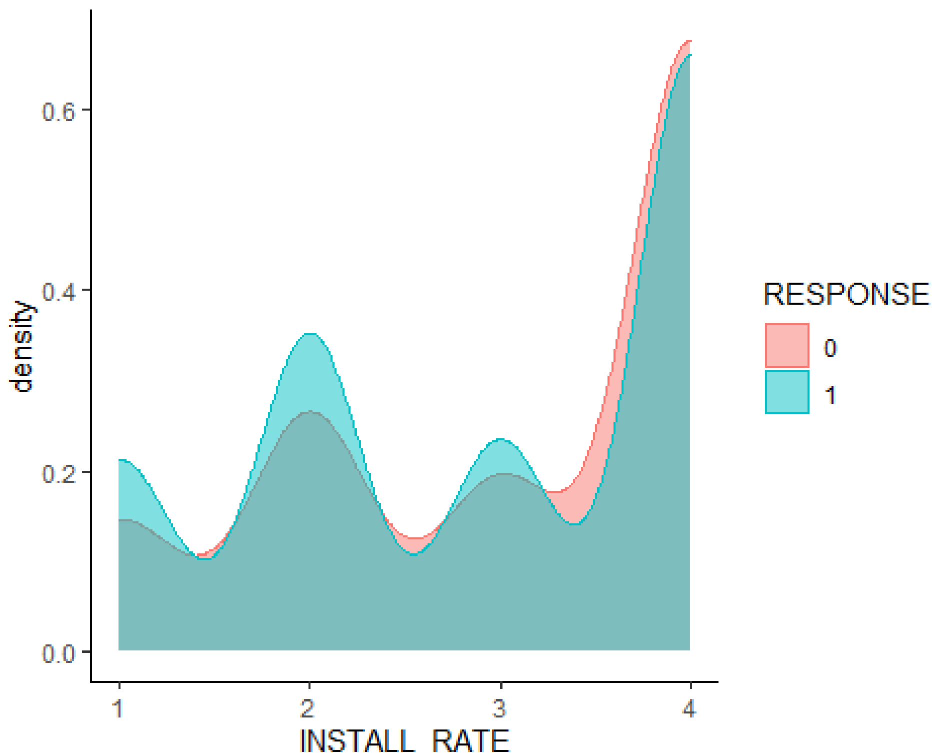
The plot shows higher density of Good and Bad Response below 30 months, with a higher ratio of Good Response. The plot gradually reduced after 30 months, with a higher ratio of Bad Response.

```
ggplot(data)+
  geom_density(aes(x=AMOUNT, color=RESPONSE,fill=RESPONSE),alpha=0.5) +
theme_classic()
```



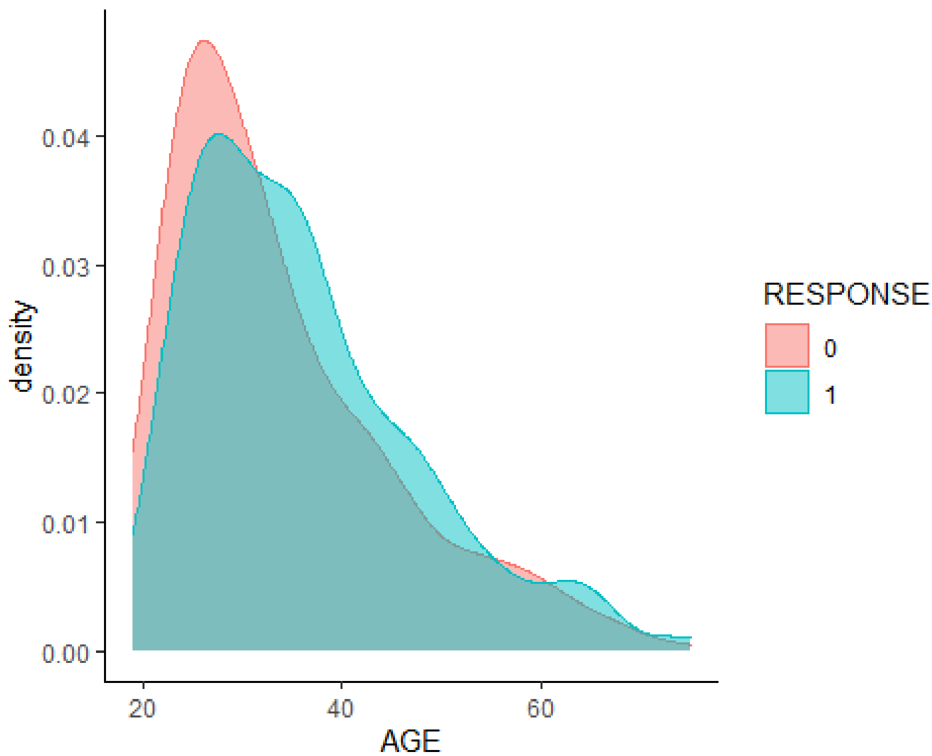
The plot shows higher density of Good and Bad Response below $5000, with a higher ratio of Good Response. The plot gradually reduces after $5000, with a higher ratio of Bad Response.

```
ggplot(data)+
  geom_density(aes(x=INSTALL_RATE, color=RESPONSE,fill=RESPONSE),alpha=0.5) +
theme_classic()
```



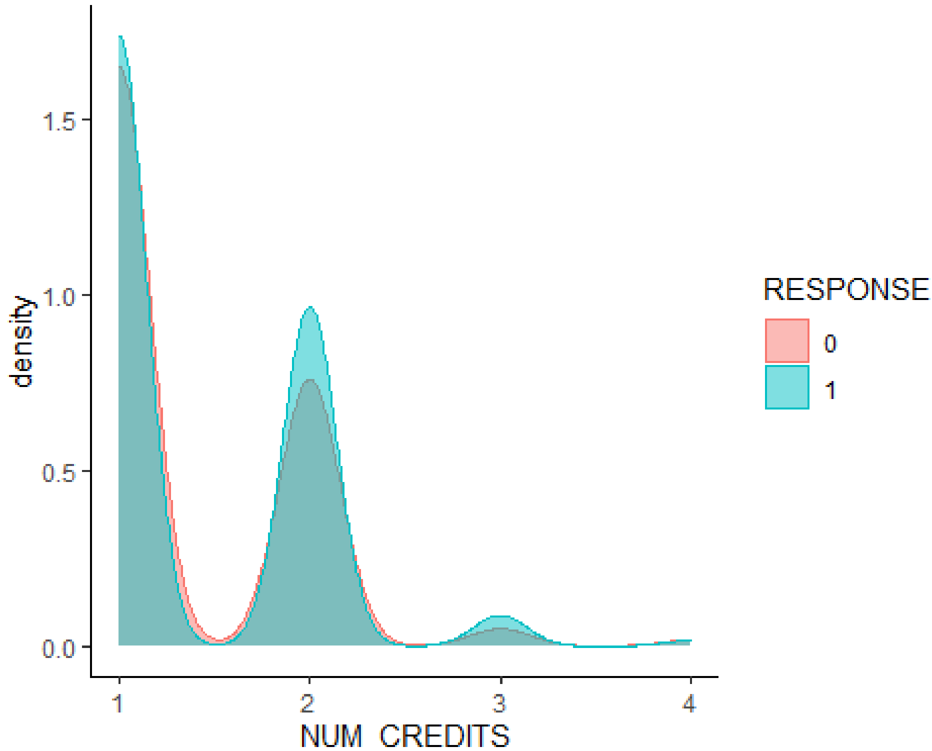
The plot shows a similar trend between Good and Bad Responses.

```
ggplot(data)+
  geom_density(aes(x=AGE, color=RESPONSE,fill=RESPONSE),alpha=0.5) +
theme_classic()
```



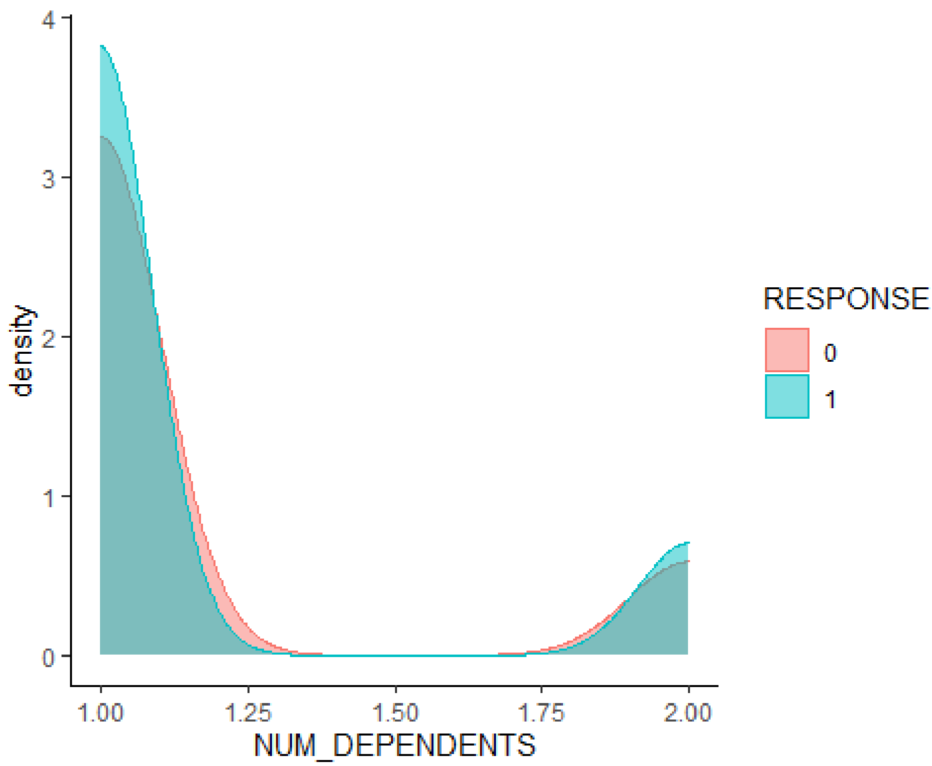
The plot shows a similar trend between Good and Bad Responses. But Bad Responses are higher than Good Responses at the age of 30.

```
ggplot(data)+
  geom_density(aes(x=NUM_CREDITS, color=RESPONSE,fill=RESPONSE),alpha=0.5) +
theme_classic()
```



The plot shows a similar trend between Good and Bad Responses.

```
ggplot(data)+
  geom_density(aes(x=NUM_DEPENDENTS, color=RESPONSE,fill=RESPONSE),alpha=0.5)
+ theme_classic()
```



The plot shows a similar trend between Good and Bad Responses.

Calculate chi-square values for categorical variables (Descending order of X-Squared, Higher = More important)

```
chisq.test(data$CHK_ACCT, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$CHK_ACCT and data$RESPONSE
## X-squared = 123.72, df = 3, p-value < 2.2e-16

chisq.test(data$HISTORY, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$HISTORY and data$RESPONSE
## X-squared = 61.691, df = 4, p-value = 1.279e-12

chisq.test(data$SAV_ACCT, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$SAV_ACCT and data$RESPONSE
## X-squared = 36.099, df = 4, p-value = 2.761e-07
```

```r
chisq.test(data$EMPLOYMENT, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$EMPLOYMENT and data$RESPONSE
## X-squared = 18.368, df = 4, p-value = 0.001045

chisq.test(data$OWN_RES, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$OWN_RES and data$RESPONSE
## X-squared = 18.114, df = 1, p-value = 2.081e-05

chisq.test(data$PROP_UNKN_NONE, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$PROP_UNKN_NONE and data$RESPONSE
## X-squared = 15.813, df = 1, p-value = 6.992e-05

chisq.test(data$REAL_ESTATE, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$REAL_ESTATE and data$RESPONSE
## X-squared = 14.232, df = 1, p-value = 0.0001616

chisq.test(data$OTHER_INSTALL, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$OTHER_INSTALL and data$RESPONSE
## X-squared = 12.834, df = 1, p-value = 0.0003405

chisq.test(data$`RADIO/TV`, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$`RADIO/TV` and data$RESPONSE
## X-squared = 11.432, df = 1, p-value = 0.0007218

chisq.test(data$NEW_CAR, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
```

```
## data:  data$NEW_CAR and data$RESPONSE
## X-squared = 9.3897, df = 1, p-value = 0.002182

chisq.test(data$USED_CAR, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$USED_CAR and data$RESPONSE
## X-squared = 9.9582, df = 1, p-value = 0.001601

chisq.test(data$RENT, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$RENT and data$RESPONSE
## X-squared = 8.6091, df = 1, p-value = 0.003345

chisq.test(data$MALE_SINGLE, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$MALE_SINGLE and data$RESPONSE
## X-squared = 6.5087, df = 1, p-value = 0.01073

chisq.test(data$FOREIGN, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$FOREIGN and data$RESPONSE
## X-squared = 6.737, df = 1, p-value = 0.009443

chisq.test(data$EDUCATION, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$EDUCATION and data$RESPONSE
## X-squared = 4.9123, df = 1, p-value = 0.02667

chisq.test(data$`CO-APPLICANT`, data$RESPONSE, correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  data$`CO-APPLICANT` and data$RESPONSE
## X-squared = 3.9348, df = 1, p-value = 0.0473

chisq.test(data$GUARANTOR, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$GUARANTOR and data$RESPONSE
## X-squared = 3.0293, df = 1, p-value = 0.08177
```

```
chisq.test(data$MALE_DIV, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$MALE_DIV and data$RESPONSE
## X-squared = 2.5063, df = 1, p-value = 0.1134
```

```
chisq.test(data$JOB, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$JOB and data$RESPONSE
## X-squared = 1.8852, df = 3, p-value = 0.5966
```

```
chisq.test(data$RETRAINING, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$RETRAINING and data$RESPONSE
## X-squared = 1.3053, df = 1, p-value = 0.2532
```

```
chisq.test(data$TELEPHONE, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$TELEPHONE and data$RESPONSE
## X-squared = 1.3298, df = 1, p-value = 0.2488
```

```
chisq.test(data$PRESENT_RESIDENT, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$PRESENT_RESIDENT and data$RESPONSE
## X-squared = 0.7493, df = 3, p-value = 0.8616
```

```
chisq.test(data$MALE_MAR_or_WID, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
```

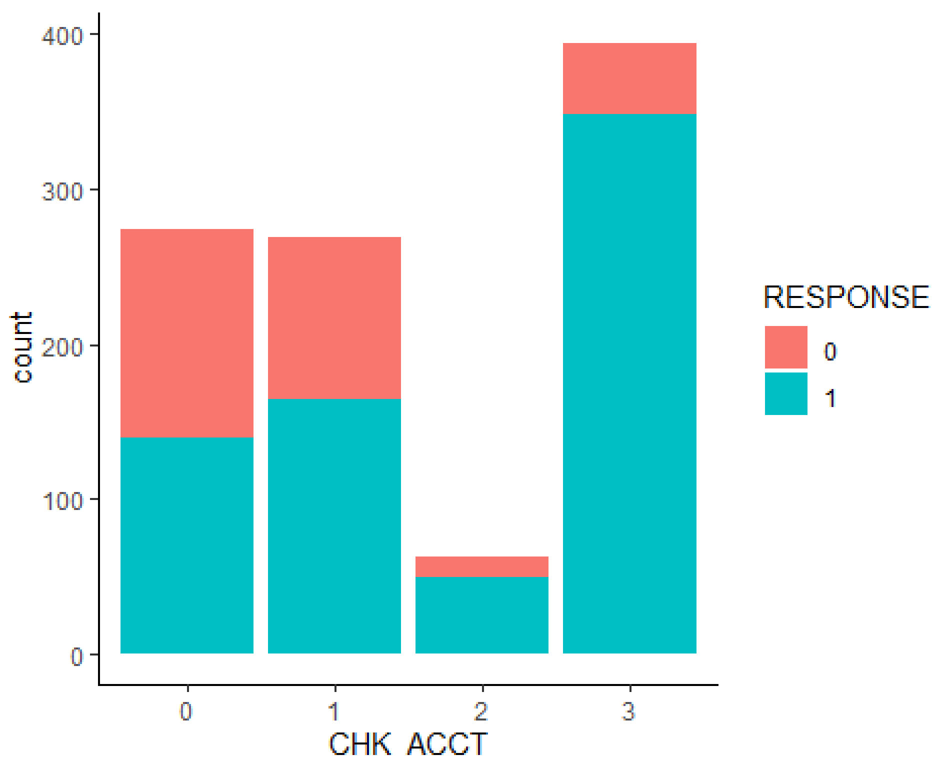
```
## data:  data$MALE_MAR_or_WID and data$RESPONSE
## X-squared = 0.38535, df = 1, p-value = 0.5348
```

```
chisq.test(data$FURNITURE, data$RESPONSE, correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$FURNITURE and data$RESPONSE
## X-squared = 0.43977, df = 1, p-value = 0.5072
```

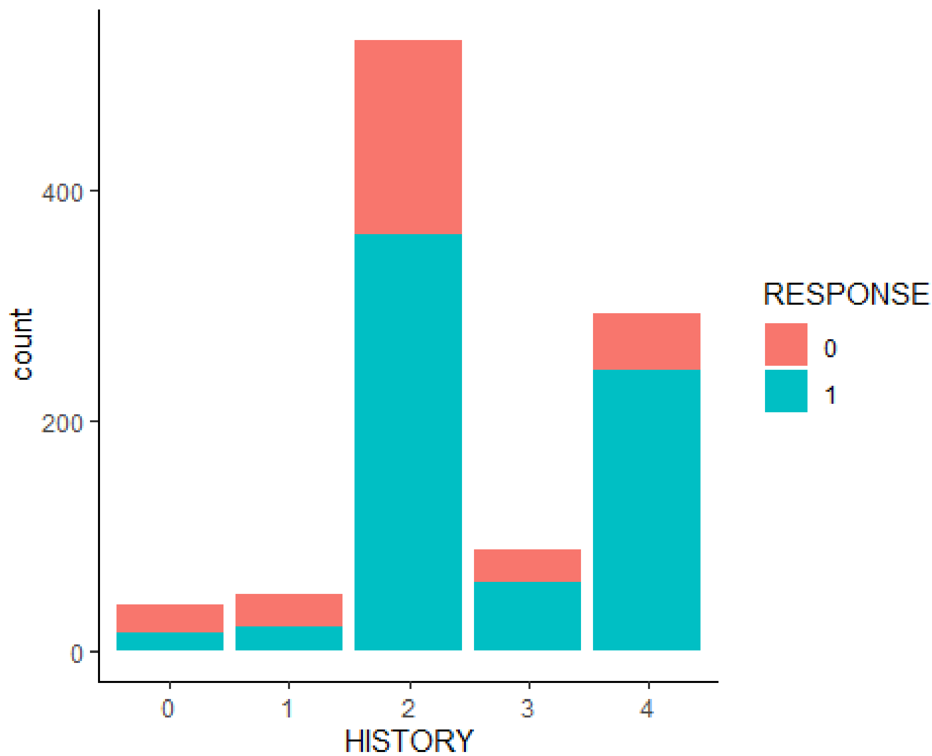
Bar Graphs - Finding correlation between categorical variables and target variable - RESPONSE for important categorical values

```
ggplot(data)+
  geom_bar(aes(x=CHK_ACCT,fill=RESPONSE)) + theme_classic()
```



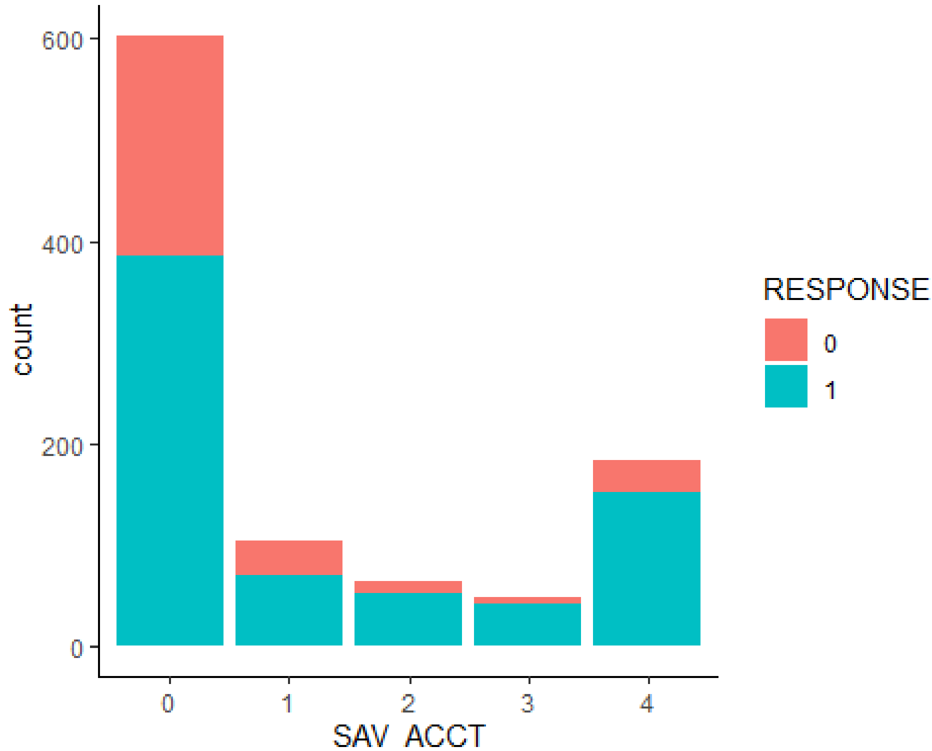
Checking account type 2 (=> 200DM) has the lowest Responses. People with no checking account (type 4) has the highest Response with Higher Good Response.

```
ggplot(data)+
  geom_bar(aes(x=HISTORY,fill=RESPONSE)) + theme_classic()
```



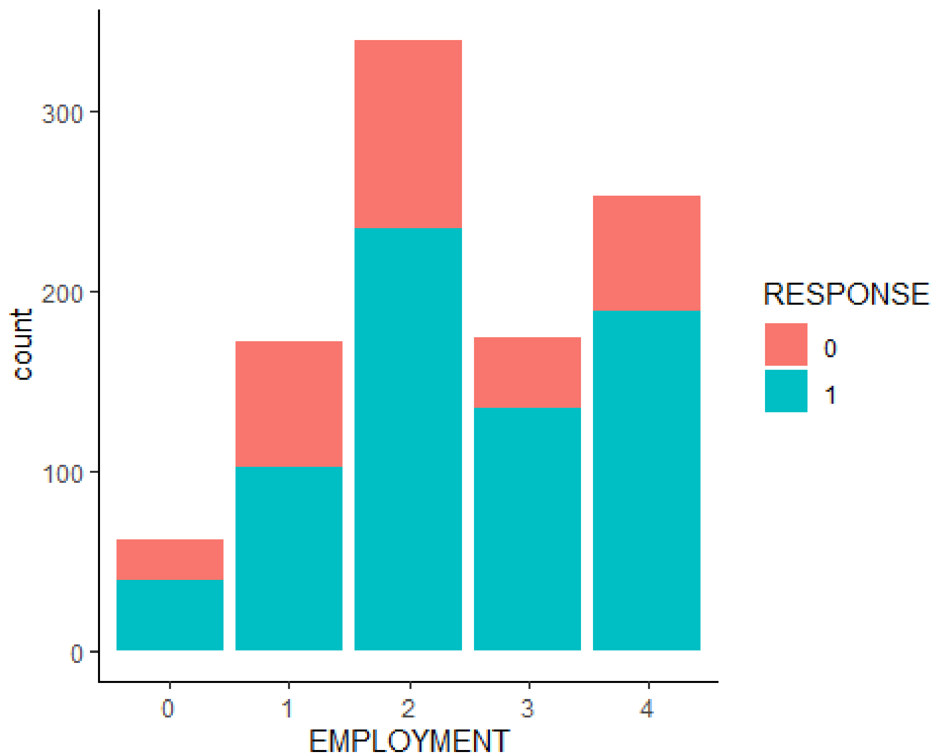
History type 0 (people who have no credits taken) and type 1 (all credits at this bank paid back duly) have equal Good and Bad Responses. History type 2 (existing credits paid back duly till now) have the highest Responses and have more Good Response than Bad.

```
ggplot(data)+
  geom_bar(aes(x=SAV_ACCT,fill=RESPONSE)) + theme_classic()
```



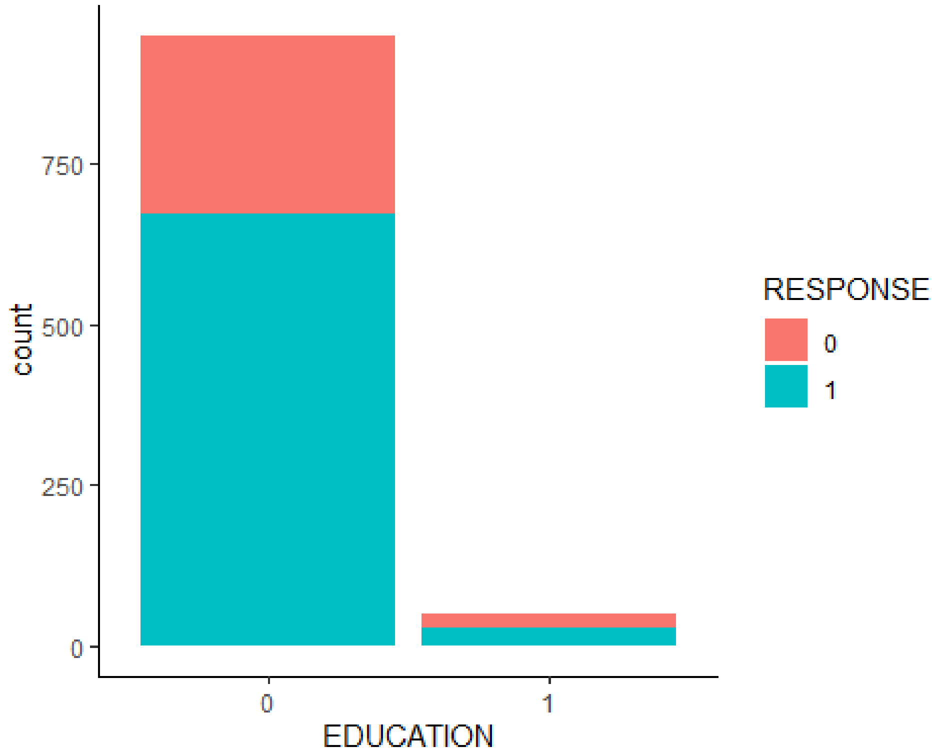
Savings account type 0 (< 100 DM) has the highest Response rate. It also has the highest density of Bad Responses among all other Savings Account Type.

```
ggplot(data)+
  geom_bar(aes(x=EMPLOYMENT,fill=RESPONSE)) + theme_classic()
```



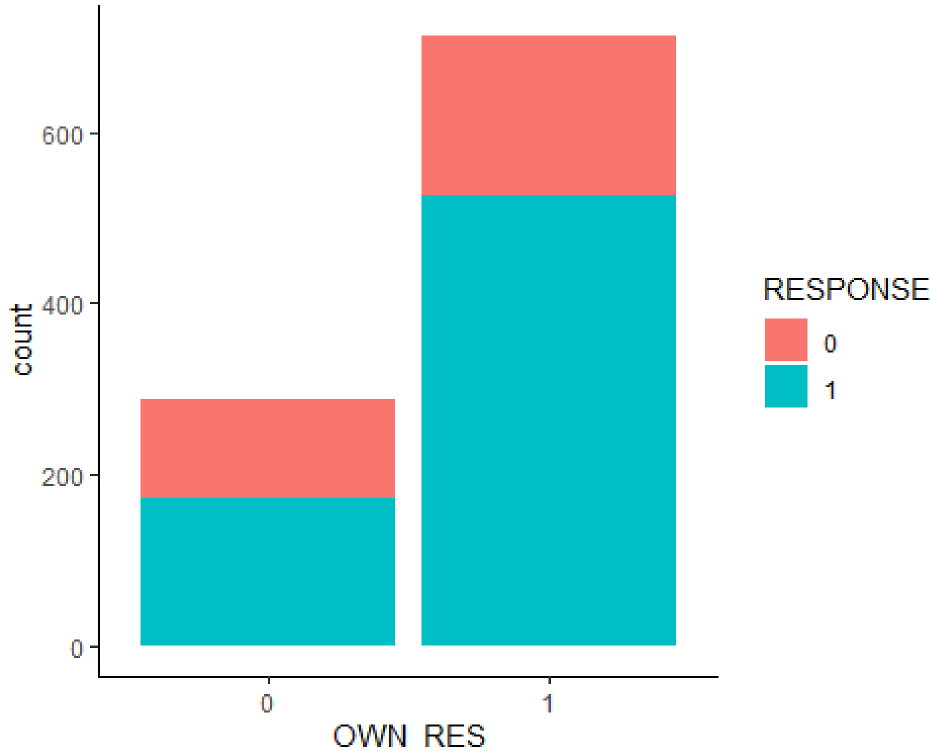
Employment type have similar Response Trends. Employment between 1 and 4 Years (Type 2) has the highest Response Rate.

```
ggplot(data)+
  geom_bar(aes(x=EDUCATION,fill=RESPONSE)) + theme_classic()
```



There are higher Responses for Education = 0 than Education = 1.

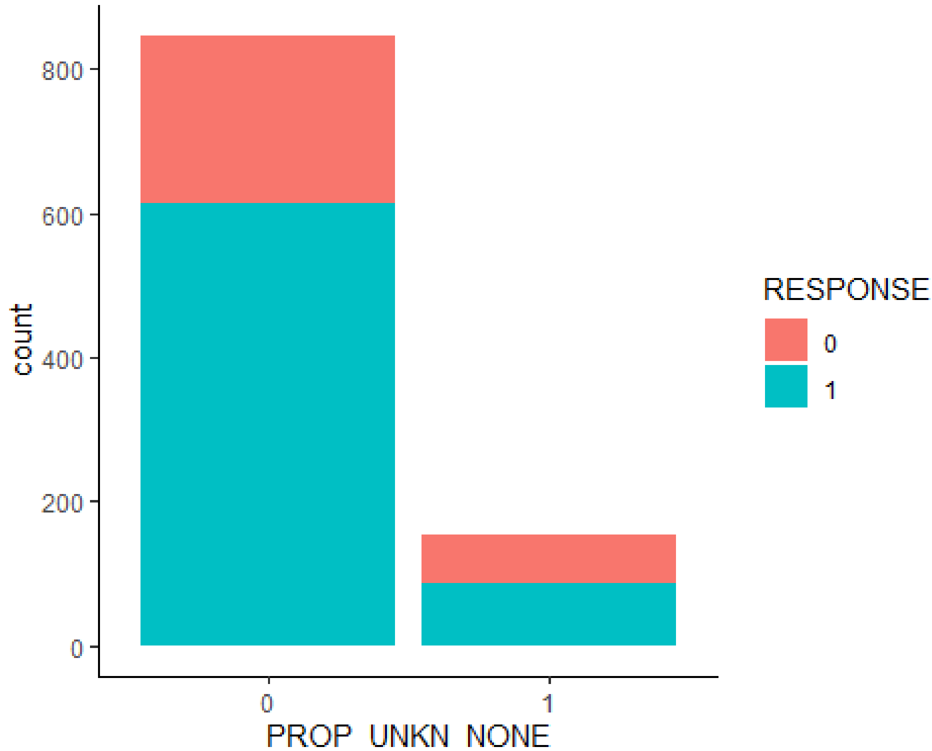
```
ggplot(data)+
  geom_bar(aes(x=OWN_RES,fill=RESPONSE)) + theme_classic()
```



There is a higher

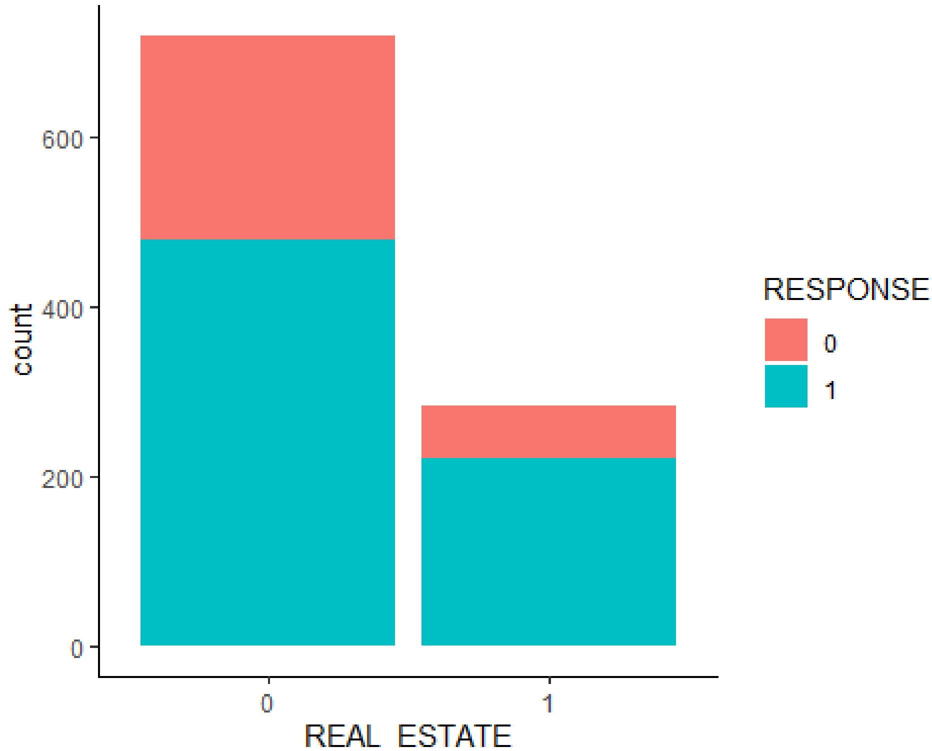
Response Rate when they own a resident (type = 1)

```
ggplot(data)+
  geom_bar(aes(x=PROP_UNKN_NONE,fill=RESPONSE)) + theme_classic()
```



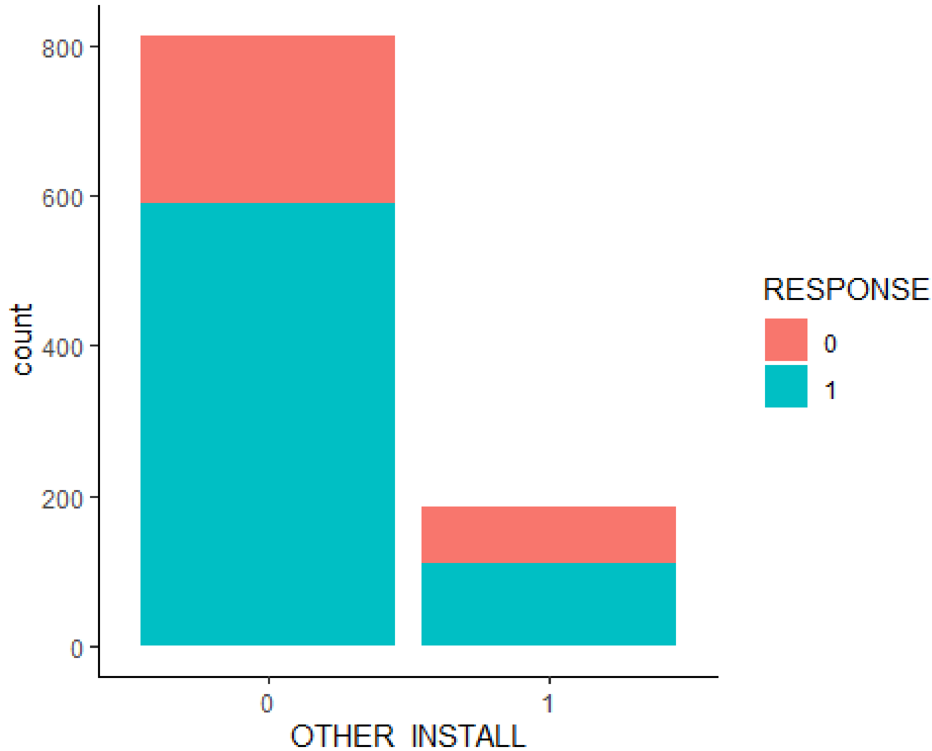
There is a higher Response Rate when they have a property.

```
ggplot(data)+
  geom_bar(aes(x=REAL_ESTATE,fill=RESPONSE)) + theme_classic()
```



There is a higher Response Rate when they have real estate

```
ggplot(data)+
  geom_bar(aes(x=OTHER_INSTALL,fill=RESPONSE)) + theme_classic()
```



There is a higher Response Rate when they don't have another installment plan credit.

The variables that are important to predict "good" and "bad" cases are:

Numerical variables:

AGE NUM_CREDITS NUM_DEPENDENTS

Categorical variables:

CHK_ACCT HISTORY SAV_ACCT EMPLOYMENT OWN_RES REAL_ESTATE OTHER_INSTALL RADIO/TV USED_CAR NEW_CAR RENT

Problem b)

We are considering 'HIGH' for 'GOOD applicants' and 'LOW' for 'BAD applicants'

```
names(data)[names(data) == 'OBS#'] <- 'OBS'
data <- subset(data, select = -c(OBS))
data$RESPONSE <- as.factor(ifelse(data$RESPONSE == 1,"HIGH","LOW"))
set.seed(96)
```

Model 1: Distributing the response variable values based on a 50% probability split.

Using the index function to assign 1 & 2 to the observations in the dataset named data.

```
set.seed(96)
index <- sample(2, nrow(data), replace = T, prob = c(0.5,0.5))
```

selecting index 1 for training data

```
train <- data[index == 1,]
```

selecting index 2 for training data

```
test <- data[index == 2,]
```

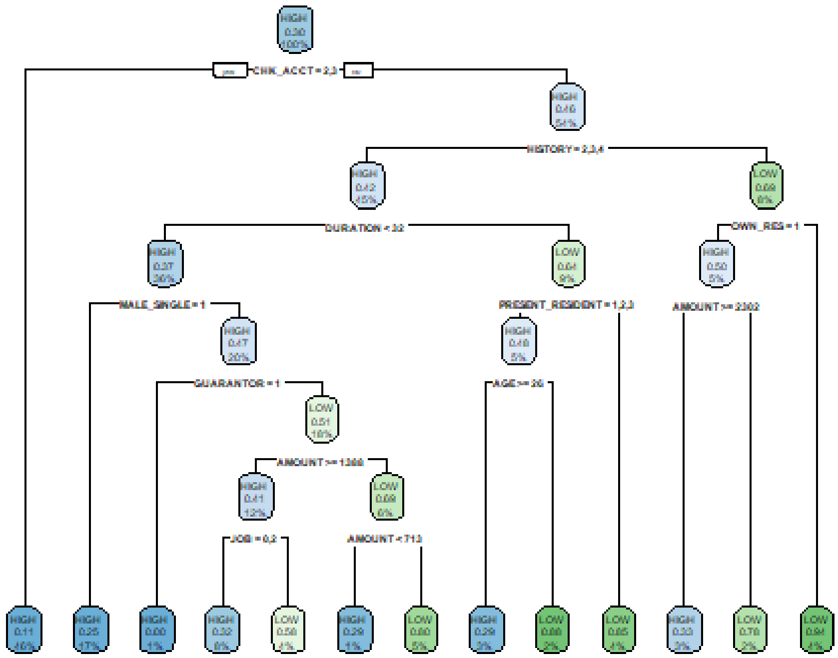
Creating formula with all the variables using ., to serve as an input parameter to rpart.

```
MyFormula = RESPONSE ~.

mytree_50_50_basic <- rpart(MyFormula, data=train)
print(mytree_50_50_basic)
```

```
## n= 501
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 501 151 HIGH (0.69860279 0.30139721)
##     2) CHK_ACCT=2,3 232  26 HIGH (0.88793103 0.11206897) *
##     3) CHK_ACCT=0,1 269 125 HIGH (0.53531599 0.46468401)
##       6) HISTORY=2,3,4 227  96 HIGH (0.57709251 0.42290749)
##        12) DURATION< 31.5 182  67 HIGH (0.63186813 0.36813187)
##          24) MALE_SINGLE=1 84  21 HIGH (0.75000000 0.25000000) *
##          25) MALE_SINGLE=0 98  46 HIGH (0.53061224 0.46938776)
##            50) GUARANTOR=1 7   0 HIGH (1.00000000 0.00000000) *
##            51) GUARANTOR=0 91  45 LOW (0.49450549 0.50549451)
##             102) AMOUNT>=1387.5 59  24 HIGH (0.59322034 0.40677966)
##               204) JOB=0,2 40  13 HIGH (0.67500000 0.32500000) *
##               205) JOB=1,3 19   8 LOW (0.42105263 0.57894737) *
##             103) AMOUNT< 1387.5 32  10 LOW (0.31250000 0.68750000)
##               206) AMOUNT< 713 7   2 HIGH (0.71428571 0.28571429) *
##               207) AMOUNT>=713 25   5 LOW (0.20000000 0.80000000) *
##        13) DURATION>=31.5 45  16 LOW (0.35555556 0.64444444)
##          26) PRESENT_RESIDENT=1,2,3 25  12 HIGH (0.52000000 0.48000000)
##            52) AGE>=25.5 17   5 HIGH (0.70588235 0.29411765) *
##            53) AGE< 25.5 8   1 LOW (0.12500000 0.87500000) *
##          27) PRESENT_RESIDENT=4 20   3 LOW (0.15000000 0.85000000) *
##       7) HISTORY=0,1 42  13 LOW (0.30952381 0.69047619)
##        14) OWN_RES=1 24  12 HIGH (0.50000000 0.50000000)
##          28) AMOUNT>=2301.5 15   5 HIGH (0.66666667 0.33333333) *
##          29) AMOUNT< 2301.5 9   2 LOW (0.22222222 0.77777778) *
##        15) OWN_RES=0 18   1 LOW (0.05555556 0.94444444) *
```

Pre-Pruning : Decision tree based on 50:50 split

```
rpart.plot(mytree_50_50_basic)
```



Predict function to predict the classes for the decision tree mytree_50_50_basic for training data.

```
mytree_train_predict_50_50 <- predict(mytree_50_50_basic, data = train , type
= "class")
```

Calculating the training error by comparing predicted classes with response variable of original dataset.

```
mytree_train_error_50_50 <- mean(mytree_train_predict_50_50 !=
train$RESPONSE)
mytree_train_error_50_50

## [1] 0.1836327
```

Predict function to predict the classes for the decision tree mytree_50_50 for testing data.

```
mytree_test_predict_50_50 <- predict(mytree_50_50_basic, newdata = test, type
= "class")
```

Calculating the testing error by comparing predicted classes with response variable of original dataset.

```
mytree_test_error_50_50 <- mean(mytree_test_predict_50_50 != test$RESPONSE)
mytree_test_error_50_50

## [1] 0.2805611
```

Calculating the performance of the model by finding the difference between the test error & train data.

```
diff_50_50 = mytree_test_error_50_50 - mytree_train_error_50_50
print(diff_50_50)

## [1] 0.09692839
```

For gini split: Based on the summary command of the 50:50 split, below CP values are derived.

```
knitr::include_graphics("CP1.png")
```

	CP	nsplit	rel error	xerror	xstd
1	0.05298013	0	1.0000000	1.0000000	0.06801844
2	0.02649007	3	0.8079470	0.9602649	0.06722227
3	0.02317881	6	0.7284768	1.0132450	0.06827151
4	0.01986755	8	0.6821192	1.0264901	0.06851854
5	0.01655629	10	0.6423841	1.0264901	0.06851854
6	0.01000000	12	0.6092715	1.0132450	0.06827151

APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR 50-50 MODEL

Creating vectors for minsplit and minbucket values to be used for different combinations to test performance CP: 0.02649007 with least xerror of 0.9602649.

```
msplt <- c(12,48,102)
mbckt <- c(4,16,34)

for (i in msplt)
  {
  for (j in mbckt)
{
    #Using rpart function to construct the decision tree based on training
data,split on gini.
mytree_50_50 <- rpart(MyFormula, data = train, parms = list(split="gini")
,control = rpart.control (minsplit = i,minbucket = j,cp=0.02649007))
```

```
#Print the decision tree.
#print(mytree_50_50)

#Predict function to predict the classes for the decision tree mytree_50_50
for training data.
mytree_train_predict_50_50 <- predict(mytree_50_50, data = train , type =
"class")

#Display the values of the predicted classes of the decision mytree_50_50.
mytree_train_predict_50_50

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_50_50 <- mean(mytree_train_predict_50_50 !=
train$RESPONSE)
mytree_train_error_50_50

#Predict function to predict the classes for the decision tree mytree_50_50
for testing data.
mytree_test_predict_50_50 <- predict(mytree_50_50, newdata = test, type =
"class")
mytree_test_predict_50_50

#Calculating the testing error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_50_50 <- mean(mytree_test_predict_50_50 != test$RESPONSE)
mytree_test_error_50_50

#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_50_50 = mytree_test_error_50_50 - mytree_train_error_50_50

print(diff_50_50)
############################### Confusion Matrix for 50:50 Split
##################################################

cfmt <- table(train$RESPONSE,mytree_train_predict_50_50)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_train =  (tp)/(tp+fp)
accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
recall_train = (tp)/(tp+fn)
```

```
fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)



cfmt <- table(test$RESPONSE,mytree_test_predict_50_50)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test + precision_test)

# Printing the values for train data error, test data error, performance and
other parameters.
print(paste("Train data error: ", mytree_train_error_50_50))
print(paste("Test data error: ", mytree_test_error_50_50))
print(paste("Difference/performance", diff_50_50))
print(paste("precision of training data: ", precision_train))
print(paste("accuracy of training data: ", accuracymodel_train))
print(paste("recall of training data: ", recall_train))
print(paste("F-score of training data: ", fscore_train))
print(paste("precision of test data: ", precision_test))
print(paste("accuracy of test data: ", accuracymodel_test))
print(paste("recall of test data: ", recall_test))
print(paste("F-score of test data: ", fscore_test))
}
}

## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
```

```
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
```

```
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
```

```
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
```

```
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##         HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
```

50-50 SPLIT DECISION TREE BASED ON INFORMATION GAIN

```
mytree_50_50_basic <- rpart(MyFormula, data=train, parms =
list(split="information"))
```

Based on the summary command of the 50:50 split, below CP values are derived.

```
knitr::include_graphics("CP2.png")
```

	CP	nsplit	rel error	xerror	xstd
1	0.05298013	0	1.0000000	1.0000000	0.06801844
2	0.02649007	3	0.8079470	1.0529801	0.06899471
3	0.02317881	6	0.7284768	1.0463576	0.06887788
4	0.01655629	8	0.6821192	1.0066225	0.06814574
5	0.01324503	10	0.6490066	0.9403974	0.06680294
6	0.01000000	12	0.6225166	0.9470199	0.06694432

APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR information gain 50-50 MODEL

Creating vectors for minsplit and minbucket values to be used for different combinations to test performance CP: 0.01000000 with least xerror of 0.6225166.

```
for (i in msplt)
{
  for (j in mbckt)
  {

#Constructing the decision tree based on information gain for 50-50 split
mytree_50_50_info <- rpart(MyFormula, data = train, parms =
list(split="information"),control = rpart.control (minsplit = i,minbucket =
j, cp= 0.01000000))

#print(mytree_50_50_info)

mytree_train_predict_50_50 <- predict(mytree_50_50_info, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_50_50 <- mean(mytree_train_predict_50_50 !=
train$RESPONSE)
mytree_train_error_50_50

#Predict function to predict the classes for the decision tree mytree_50_50
for testing data.
mytree_test_predict_50_50 <- predict(mytree_50_50_info, newdata = test, type
= "class")
mytree_test_predict_50_50

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_50_50 <- mean(mytree_test_predict_50_50 != test$RESPONSE)
mytree_test_error_50_50

#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_50_50 = mytree_test_error_50_50 - mytree_train_error_50_50
print(diff_50_50)
cfmt <- table(train$RESPONSE,mytree_train_predict_50_50)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_train =  (tp)/(tp+fp)
accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
recall_train = (tp)/(tp+fn)
```

```
fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)



cfmt <- table(test$RESPONSE,mytree_test_predict_50_50)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_test)

# Printing the values for train data error, test data error, performance and
other parameters.
print(paste("Train data error: ", mytree_train_error_50_50))
print(paste("Test data error: ", mytree_test_error_50_50))
print(paste("Difference/performance", diff_50_50))
print(paste("precision of training data: ", precision_train))
print(paste("accuracy of training data: ", accuracymodel_train))
print(paste("recall of training data: ", recall_train))
print(paste("F-score of training data: ", fscore_train))
print(paste("precision of test data: ", precision_test))
print(paste("accuracy of test data: ", accuracymodel_test))
print(paste("recall of test data: ", recall_test))
print(paste("F-score of test data: ", fscore_test))
}
}

## [1] 0.1108684
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  322  28
##   LOW    53  98
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  299  51
##   LOW    85  64
## [1] "Train data error:  0.161676646706587"
## [1] "Test data error:  0.272545090180361"
## [1] "Difference/performance 0.110868443473774"
## [1] "precision of training data:  0.858666666666667"
```

```
## [1] "accuracy of training data:  0.838323353293413"
## [1] "recall of training data:  0.92"
## [1] "F-score of training data:  0.888275862068966"
## [1] "precision of test data:  0.778645833333333"
## [1] "accuracy of test data:  0.727454909819639"
## [1] "recall of test data:  0.854285714285714"
## [1] "F-score of test data:  0.814713896457766"
## [1] 0.06300425
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  309  41
##   LOW    69  82
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  299  51
##   LOW    90  59
## [1] "Train data error:  0.219560878243513"
## [1] "Test data error:  0.282565130260521"
## [1] "Difference/performance 0.0630042520170081"
## [1] "precision of training data:  0.817460317460317"
## [1] "accuracy of training data:  0.780439121756487"
## [1] "recall of training data:  0.882857142857143"
## [1] "F-score of training data:  0.848901098901099"
## [1] "precision of test data:  0.768637532133676"
## [1] "accuracy of test data:  0.717434869739479"
## [1] "recall of test data:  0.854285714285714"
## [1] "F-score of test data:  0.809201623815968"
## [1] 0.0510122
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  307  43
##   LOW    71  80
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  297  53
##   LOW    86  63
## [1] "Train data error:  0.227544910179641"
## [1] "Test data error:  0.278557114228457"
## [1] "Difference/performance 0.0510122040488162"
## [1] "precision of training data:  0.812169312169312"
## [1] "accuracy of training data:  0.772455089820359"
## [1] "recall of training data:  0.877142857142857"
## [1] "F-score of training data:  0.843406593406593"
## [1] "precision of test data:  0.775456919060052"
## [1] "accuracy of test data:  0.721442885771543"
## [1] "recall of test data:  0.848571428571429"
## [1] "F-score of test data:  0.810368349249659"
## [1] 0.07695231
##       mytree_train_predict_50_50
##        HIGH LOW
```

```
##   HIGH  310  40
##   LOW    60  91
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  297  53
##   LOW    85  64
## [1] "Train data error:  0.199600798403194"
## [1] "Test data error:  0.276553106212425"
## [1] "Difference/performance 0.0769523078092312"
## [1] "precision of training data:  0.837837837837838"
## [1] "accuracy of training data:  0.800399201596806"
## [1] "recall of training data:  0.885714285714286"
## [1] "F-score of training data:  0.861111111111111"
## [1] "precision of test data:  0.777486910994764"
## [1] "accuracy of test data:  0.723446893787575"
## [1] "recall of test data:  0.848571428571429"
## [1] "F-score of test data:  0.811475409836066"
## [1] 0.06702827
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  298  52
##   LOW    60  91
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  287  63
##   LOW    82  67
## [1] "Train data error:  0.223552894211577"
## [1] "Test data error:  0.290581162324649"
## [1] "Difference/performance 0.0670282681130725"
## [1] "precision of training data:  0.832402234636871"
## [1] "accuracy of training data:  0.776447105788423"
## [1] "recall of training data:  0.851428571428571"
## [1] "F-score of training data:  0.84180790960452"
## [1] "precision of test data:  0.777777777777778"
## [1] "accuracy of test data:  0.709418837675351"
## [1] "recall of test data:  0.82"
## [1] "F-score of test data:  0.798331015299026"
## [1] 0.0510122
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  307  43
##   LOW    71  80
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  297  53
##   LOW    86  63
## [1] "Train data error:  0.227544910179641"
## [1] "Test data error:  0.278557114228457"
## [1] "Difference/performance 0.0510122040488162"
## [1] "precision of training data:  0.812169312169312"
```

```
## [1] "accuracy of training data:  0.772455089820359"
## [1] "recall of training data:  0.877142857142857"
## [1] "F-score of training data:  0.843406593406593"
## [1] "precision of test data:  0.775456919060052"
## [1] "accuracy of test data:  0.721442885771543"
## [1] "recall of test data:  0.848571428571429"
## [1] "F-score of test data:  0.810368349249659"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
## [1] 0.01300005
##       mytree_train_predict_50_50
##        HIGH LOW
```

```
##   HIGH  321  29
##   LOW    93  58
##       mytree_test_predict_50_50
##        HIGH LOW
##   HIGH  325  25
##   LOW   103  46
## [1] "Train data error:  0.243512974051896"
## [1] "Test data error:  0.256513026052104"
## [1] "Difference/performance 0.013000052000208"
## [1] "precision of training data:  0.77536231884058"
## [1] "accuracy of training data:  0.756487025948104"
## [1] "recall of training data:  0.917142857142857"
## [1] "F-score of training data:  0.840314136125655"
## [1] "precision of test data:  0.759345794392523"
## [1] "accuracy of test data:  0.743486973947896"
## [1] "recall of test data:  0.928571428571429"
## [1] "F-score of test data:  0.83547557840617"
```

MODEL 2: 70:30 SPLIT

```
# Assigning 1 & 2 as index to split test and train data
set.seed(96)
index <- sample(2, nrow(data), replace = T, prob = c(0.7,0.3))

#selecting index 1 for training
train <- data[index == 1,]

#selecting index 1 for testing
test <- data[index == 2,]

#Creating formula with all the response variables using ., to serve as an
input parameter to rpart.
MyFormula = RESPONSE ~.

mytree_70_30_basic = rpart(MyFormula, data=train)
print(mytree_70_30_basic)
```

```
## n= 679
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 679 208 HIGH (0.6936672 0.3063328)
##     2) CHK_ACCT=2,3 297  39 HIGH (0.8686869 0.1313131) *
##     3) CHK_ACCT=0,1 382 169 HIGH (0.5575916 0.4424084)
##       6) DURATION< 22.5 211  70 HIGH (0.6682464 0.3317536)
##        12) HISTORY=2,3,4 194  56 HIGH (0.7113402 0.2886598)
##          24) CHK_ACCT=1 101  20 HIGH (0.8019802 0.1980198) *
##          25) CHK_ACCT=0 93  36 HIGH (0.6129032 0.3870968)
##            50) DURATION< 11.5 30   5 HIGH (0.8333333 0.1666667) *
```

```
##             51) DURATION>=11.5 63  31 HIGH (0.5079365 0.4920635)
##              102) AMOUNT< 2224.5 42  17 HIGH (0.5952381 0.4047619)
##                204) AMOUNT>=1374 15   2 HIGH (0.8666667 0.1333333) *
##                205) AMOUNT< 1374 27  12 LOW (0.4444444 0.5555556)
##                  410) EMPLOYMENT=2,4 17   7 HIGH (0.5882353 0.4117647) *
##                  411) EMPLOYMENT=0,1,3 10   2 LOW (0.2000000 0.8000000) *
##              103) AMOUNT>=2224.5 21   7 LOW (0.3333333 0.6666667) *
##         13) HISTORY=0,1 17   3 LOW (0.1764706 0.8235294) *
##        7) DURATION>=22.5 171  72 LOW (0.4210526 0.5789474)
##         14) SAV_ACCT=1,2,3,4 59  25 HIGH (0.5762712 0.4237288)
##           28) HISTORY=1,3,4 25   4 HIGH (0.8400000 0.1600000) *
##           29) HISTORY=0,2 34  13 LOW (0.3823529 0.6176471)
##             58) SAV_ACCT=2,3,4 21  10 HIGH (0.5238095 0.4761905)
##              116) CHK_ACCT=1 10   2 HIGH (0.8000000 0.2000000) *
##              117) CHK_ACCT=0 11   3 LOW (0.2727273 0.7272727) *
##             59) SAV_ACCT=1 13   2 LOW (0.1538462 0.8461538) *
##         15) SAV_ACCT=0 112  38 LOW (0.3392857 0.6607143)
##           30) DURATION< 43.5 87  36 LOW (0.4137931 0.5862069)
##             60) USED_CAR=1 13   3 HIGH (0.7692308 0.2307692) *
##             61) USED_CAR=0 74  26 LOW (0.3513514 0.6486486) *
##           31) DURATION>=43.5 25   2 LOW (0.0800000 0.9200000) *
```

```
#Predict function to predict the classes for the decision tree
mytree_70_30_basic for training data.
mytree_train_predict_70_30 <- predict(mytree_70_30_basic, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 !=
train$RESPONSE)
mytree_train_error_70_30
```

```
## [1] 0.1870398
```

```
#Predict function to predict the classes for the decision tree mytree_70_30
for testing data.
mytree_test_predict_70_30 <- predict(mytree_70_30_basic, newdata = test, type
= "class")

#Calculating the testing error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$RESPONSE)
mytree_test_error_70_30
```

```
## [1] 0.2492212
```

```
#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30
```

```r
print(diff_70_30)

## [1] 0.06218142
```

Based on the summary command of the 70:30 split, below CP values are derived.

```r
knitr::include_graphics("CP3.png")
```

	CP	nsplit	rel error	xerror	xstd
1	0.06490385	0	1.0000000	1.0000000	0.05774892
2	0.05288462	2	0.8701923	0.9807692	0.05743329
3	0.04326923	3	0.8173077	0.9134615	0.05623841
4	0.03846154	4	0.7740385	0.9615385	0.05519672
5	0.01682692	5	0.7355769	0.8653846	0.05529531
6	0.01442308	7	0.7019231	0.8846154	0.05568180
7	0.01121795	9	0.6730769	0.9038462	0.05605592
8	0.01000000	14	0.6105769	0.8605769	0.05710637

APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR 70-30 MODEL

Creating vectors for minsplit and minbucket values to be used for different combinations to test performance. Based on summary ,using with CP = 0.01000000

```r
msplt <- c(12,48,102)
mbckt <- c(4,16,34)

for (i in msplt)
{
  for (j in mbckt)
  {
mytree_70_30 <- rpart(MyFormula, data = train,control = rpart.control
(minsplit = i,minbucket = j, cp = 0.01000000))
#print(mytree_70_30)

mytree_train_predict_70_30 <- predict(mytree_70_30, data = train , type =
"class")
#?predict
mytree_train_predict_70_30
mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 !=
train$RESPONSE)
```

```
mytree_train_error_70_30

mytree_test_predict_70_30 <- predict(mytree_70_30, newdata = test, type =
"class")
mytree_test_predict_70_30

mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$RESPONSE)
mytree_test_error_70_30

diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30
diff_70_30
print(diff_70_30)

############################## Confusion Matrix for 70:30 Split
##################################################

cfmt <- table(train$RESPONSE,mytree_train_predict_70_30)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_train =  (tp)/(tp+fp)
accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
recall_train = (tp)/(tp+fn)
fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)



cfmt <- table(test$RESPONSE,mytree_test_predict_70_30)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_test)
```

```
# Printing the values for train data error, test data error, performance and other parameters.
print(paste("Train data error: ", mytree_train_error_70_30))
print(paste("Test data error: ", mytree_test_error_70_30))
print(paste("Difference/performance", diff_70_30))
print(paste("precision of training data: ", precision_train))
print(paste("accuracy of training data: ", accuracymodel_train))
print(paste("recall of training data: ", recall_train))
print(paste("F-score of training data: ", fscore_train))
print(paste("precision of test data: ", precision_test))
print(paste("accuracy of test data: ", accuracymodel_test))
print(paste("recall of test data: ", recall_test))
print(paste("F-score of test data: ", fscore_test))
  }
}

## [1] 0.07543621
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  432  39
##   LOW    79 129
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  200  29
##   LOW    51  41
## [1] "Train data error:  0.173784977908689"
## [1] "Test data error:  0.249221183800623"
## [1] "Difference/performance 0.0754362058919338"
## [1] "precision of training data:  0.845401174168297"
## [1] "accuracy of training data:  0.826215022091311"
## [1] "recall of training data:  0.917197452229299"
## [1] "F-score of training data:  0.879837067209776"
## [1] "precision of test data:  0.796812749003984"
## [1] "accuracy of test data:  0.750778816199377"
## [1] "recall of test data:  0.873362445414847"
## [1] "F-score of test data:  0.833333333333333"
## [1] 0.06909098
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  422  49
##   LOW    86 122
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  198  31
##   LOW    55  37
## [1] "Train data error:  0.198821796759941"
## [1] "Test data error:  0.26791277258567"
## [1] "Difference/performance 0.0690909758257287"
## [1] "precision of training data:  0.830708661417323"
## [1] "accuracy of training data:  0.801178203240059"
```

```
## [1] "recall of training data:  0.895966029723992"
## [1] "F-score of training data:  0.862104187946884"
## [1] "precision of test data:  0.782608695652174"
## [1] "accuracy of test data:  0.73208722741433"
## [1] "recall of test data:  0.864628820960699"
## [1] "F-score of test data:  0.821576763485477"
## [1] -0.00477154
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  417  54
##   LOW   110  98
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  202  27
##   LOW    49  43
## [1] "Train data error:  0.241531664212077"
## [1] "Test data error:  0.236760124610592"
## [1] "Difference/performance -0.00477153960148469"
## [1] "precision of training data:  0.791271347248577"
## [1] "accuracy of training data:  0.758468335787923"
## [1] "recall of training data:  0.885350318471338"
## [1] "F-score of training data:  0.835671342685371"
## [1] "precision of test data:  0.804780876494024"
## [1] "accuracy of test data:  0.763239875389408"
## [1] "recall of test data:  0.882096069868996"
## [1] "F-score of test data:  0.841666666666667"
## [1] 0.08744305
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  415  56
##   LOW    75 133
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  187  42
##   LOW    48  44
## [1] "Train data error:  0.192930780559647"
## [1] "Test data error:  0.280373831775701"
## [1] "Difference/performance 0.0874430512160544"
## [1] "precision of training data:  0.846938775510204"
## [1] "accuracy of training data:  0.807069219440353"
## [1] "recall of training data:  0.881104033970276"
## [1] "F-score of training data:  0.863683662851197"
## [1] "precision of test data:  0.795744680851064"
## [1] "accuracy of test data:  0.719626168224299"
## [1] "recall of test data:  0.816593886462882"
## [1] "F-score of test data:  0.806034482758621"
## [1] 0.07401851
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  433  38
```

```
##   LOW   100 108
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  201  28
##   LOW    61  31
## [1] "Train data error:  0.203240058910162"
## [1] "Test data error:  0.277258566978193"
## [1] "Difference/performance 0.0740185080680311"
## [1] "precision of training data:  0.812382739212008"
## [1] "accuracy of training data:  0.796759941089838"
## [1] "recall of training data:  0.91932059447983"
## [1] "F-score of training data:  0.862549800796813"
## [1] "precision of test data:  0.767175572519084"
## [1] "accuracy of test data:  0.722741433021807"
## [1] "recall of test data:  0.877729257641921"
## [1] "F-score of test data:  0.818737270875764"
## [1] -0.00477154
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  417  54
##   LOW   110  98
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  202  27
##   LOW    49  43
## [1] "Train data error:  0.241531664212077"
## [1] "Test data error:  0.236760124610592"
## [1] "Difference/performance -0.00477153960148469"
## [1] "precision of training data:  0.791271347248577"
## [1] "accuracy of training data:  0.758468335787923"
## [1] "recall of training data:  0.885350318471338"
## [1] "F-score of training data:  0.835671342685371"
## [1] "precision of test data:  0.804780876494024"
## [1] "accuracy of test data:  0.763239875389408"
## [1] "recall of test data:  0.882096069868996"
## [1] "F-score of test data:  0.841666666666667"
## [1] 0.01176827
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  430  41
##   LOW   116  92
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  208  21
##   LOW    57  35
## [1] "Train data error:  0.231222385861561"
## [1] "Test data error:  0.242990654205607"
## [1] "Difference/performance 0.0117682683440463"
## [1] "precision of training data:  0.787545787545788"
## [1] "accuracy of training data:  0.768777614138439"
```

```
## [1] "recall of training data:  0.912951167728238"
## [1] "F-score of training data:  0.845624385447394"
## [1] "precision of test data:  0.784905660377359"
## [1] "accuracy of test data:  0.757009345794392"
## [1] "recall of test data:  0.908296943231441"
## [1] "F-score of test data:  0.842105263157895"
## [1] 0.005877252
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  430  41
##   LOW   120  88
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  209  20
##   LOW    58  34
## [1] "Train data error:  0.237113402061856"
## [1] "Test data error:  0.242990654205607"
## [1] "Difference/performance 0.00587725214375182"
## [1] "precision of training data:  0.781818181818182"
## [1] "accuracy of training data:  0.762886597938144"
## [1] "recall of training data:  0.912951167728238"
## [1] "F-score of training data:  0.842311459353575"
## [1] "precision of test data:  0.782771535580524"
## [1] "accuracy of test data:  0.757009345794392"
## [1] "recall of test data:  0.912663755458515"
## [1] "F-score of test data:  0.842741935483871"
## [1] -0.00477154
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  417  54
##   LOW   110  98
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  202  27
##   LOW    49  43
## [1] "Train data error:  0.241531664212077"
## [1] "Test data error:  0.236760124610592"
## [1] "Difference/performance -0.00477153960148469"
## [1] "precision of training data:  0.791271347248577"
## [1] "accuracy of training data:  0.758468335787923"
## [1] "recall of training data:  0.885350318471338"
## [1] "F-score of training data:  0.835671342685371"
## [1] "precision of test data:  0.804780876494024"
## [1] "accuracy of test data:  0.763239875389408"
## [1] "recall of test data:  0.882096069868996"
## [1] "F-score of test data:  0.841666666666667"
```

############Constructing the decision tree based on information gain for 70-30 split

```r
mytree_70_30_basic = rpart(MyFormula, data=train,parms =
list(split="information"))

knitr::include_graphics("CP4.png")
```

```
CP             nsplit rel error    xerror      xstd
1 0.06490385        0 1.0000000 1.0000000 0.05774892
2 0.05288462        2 0.8701923 0.9519231 0.05693862
3 0.03125000        3 0.8173077 0.9423077 0.05676797
4 0.01682692        5 0.7548077 0.9134615 0.05623841
5 0.01000000       10 0.6586538 0.8750000 0.05549012
```

We will choose the cp = 0.01000000 as that provides least xerror.

```r
for (i in msplt)
{
  for (j in mbckt)
  {

mytree_70_30_info <- rpart(MyFormula, data = train, parms =
list(split="information"),control = rpart.control (minsplit = i,minbucket =
j, cp=0.01000000))

#print(mytree_70_30_info)

mytree_train_predict_70_30 <- predict(mytree_70_30_info, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 !=
train$RESPONSE)
mytree_train_error_70_30

#Predict function to predict the classes for the decision tree mytree_70_30
for testing data.
mytree_test_predict_70_30 <- predict(mytree_70_30_info, newdata = test, type
= "class")
```

```
mytree_test_predict_70_30

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$RESPONSE)
mytree_test_error_70_30

#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30

print(diff_70_30)

cfmt <- table(train$RESPONSE,mytree_train_predict_70_30)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_train =  (tp)/(tp+fp)
accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
recall_train = (tp)/(tp+fn)
fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)



cfmt <- table(test$RESPONSE,mytree_test_predict_70_30)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_test)

# Printing the values for train data error, test data error, performance and
other parameters.
print(paste("Train data error: ", mytree_train_error_70_30))
```

```
print(paste("Test data error: ", mytree_test_error_70_30))
print(paste("Difference/performance", diff_70_30))
print(paste("precision of training data: ", precision_train))
print(paste("accuracy of training data: ", accuracymodel_train))
print(paste("recall of training data: ", recall_train))
print(paste("F-score of training data: ", fscore_train))
print(paste("precision of test data: ", precision_test))
print(paste("accuracy of test data: ", accuracymodel_test))
print(paste("recall of test data: ", recall_test))
print(paste("F-score of test data: ", fscore_test))
}
}
```

```
## [1] 0.06710895
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  448  23
##   LOW   107 101
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  208  21
##   LOW    62  30
## [1] "Train data error:  0.191458026509573"
## [1] "Test data error:  0.258566978193146"
## [1] "Difference/performance 0.0671089516835735"
## [1] "precision of training data:  0.807207207207207"
## [1] "accuracy of training data:  0.808541973490427"
## [1] "recall of training data:  0.951167728237792"
## [1] "F-score of training data:  0.873294346978558"
## [1] "precision of test data:  0.77037037037037"
## [1] "accuracy of test data:  0.741433021806854"
## [1] "recall of test data:  0.908296943231441"
## [1] "F-score of test data:  0.833667334669339"
## [1] 0.08251552
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  435  36
##   LOW    92 116
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  197  32
##   LOW    55  37
## [1] "Train data error:  0.188512518409426"
## [1] "Test data error:  0.271028037383178"
## [1] "Difference/performance 0.0825155189737519"
## [1] "precision of training data:  0.825426944971537"
## [1] "accuracy of training data:  0.811487481590574"
## [1] "recall of training data:  0.923566878980892"
## [1] "F-score of training data:  0.871743486973948"
## [1] "precision of test data:  0.781746031746032"
```

```
## [1] "accuracy of test data:  0.728971962616822"
## [1] "recall of test data:  0.860262008733624"
## [1] "F-score of test data:  0.819126819126819"
## [1] -0.00477154
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  417  54
##   LOW   110  98
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  202  27
##   LOW    49  43
## [1] "Train data error:  0.241531664212077"
## [1] "Test data error:  0.236760124610592"
## [1] "Difference/performance -0.00477153960148469"
## [1] "precision of training data:  0.791271347248577"
## [1] "accuracy of training data:  0.758468335787923"
## [1] "recall of training data:  0.885350318471338"
## [1] "F-score of training data:  0.835671342685371"
## [1] "precision of test data:  0.804780876494024"
## [1] "accuracy of test data:  0.763239875389408"
## [1] "recall of test data:  0.882096069868996"
## [1] "F-score of test data:  0.841666666666667"
## [1] 0.07515175
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  439  32
##   LOW   101 107
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  203  26
##   LOW    61  31
## [1] "Train data error:  0.195876288659794"
## [1] "Test data error:  0.271028037383178"
## [1] "Difference/performance 0.0751517487233837"
## [1] "precision of training data:  0.812962962962963"
## [1] "accuracy of training data:  0.804123711340206"
## [1] "recall of training data:  0.932059447983015"
## [1] "F-score of training data:  0.868447082096934"
## [1] "precision of test data:  0.768939393939394"
## [1] "accuracy of test data:  0.728971962616822"
## [1] "recall of test data:  0.88646288209607"
## [1] "F-score of test data:  0.823529411764706"
## [1] 0.0630302
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  429  42
##   LOW    95 113
##       mytree_test_predict_70_30
##        HIGH LOW
```

```
##   HIGH  198  31
##   LOW    54  38
## [1] "Train data error:  0.201767304860088"
## [1] "Test data error:  0.264797507788162"
## [1] "Difference/performance 0.0630302029280736"
## [1] "precision of training data:  0.818702290076336"
## [1] "accuracy of training data:  0.798232695139912"
## [1] "recall of training data:  0.910828025477707"
## [1] "F-score of training data:  0.862311557788945"
## [1] "precision of test data:  0.785714285714286"
## [1] "accuracy of test data:  0.735202492211838"
## [1] "recall of test data:  0.864628820960699"
## [1] "F-score of test data:  0.823284823284823"
## [1] -0.00477154
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  417  54
##   LOW   110  98
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  202  27
##   LOW    49  43
## [1] "Train data error:  0.241531664212077"
## [1] "Test data error:  0.236760124610592"
## [1] "Difference/performance -0.00477153960148469"
## [1] "precision of training data:  0.791271347248577"
## [1] "accuracy of training data:  0.758468335787923"
## [1] "recall of training data:  0.885350318471338"
## [1] "F-score of training data:  0.835671342685371"
## [1] "precision of test data:  0.804780876494024"
## [1] "accuracy of test data:  0.763239875389408"
## [1] "recall of test data:  0.882096069868996"
## [1] "F-score of test data:  0.841666666666667"
## [1] 0.02388981
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  415  56
##   LOW    97 111
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  199  30
##   LOW    50  42
## [1] "Train data error:  0.225331369661267"
## [1] "Test data error:  0.249221183800623"
## [1] "Difference/performance 0.0238898141393565"
## [1] "precision of training data:  0.810546875"
## [1] "accuracy of training data:  0.774668630338733"
## [1] "recall of training data:  0.881104033970276"
## [1] "F-score of training data:  0.844354018311292"
## [1] "precision of test data:  0.799196787148594"
```

```
## [1] "accuracy of test data:  0.750778816199377"
## [1] "recall of test data:  0.868995633187773"
## [1] "F-score of test data:  0.832635983263598"
## [1] 0.0179988
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  415  56
##   LOW   101 107
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  200  29
##   LOW    51  41
## [1] "Train data error:  0.231222385861561"
## [1] "Test data error:  0.249221183800623"
## [1] "Difference/performance 0.0179987979390619"
## [1] "precision of training data:  0.804263565891473"
## [1] "accuracy of training data:  0.768777614138439"
## [1] "recall of training data:  0.881104033970276"
## [1] "F-score of training data:  0.840932117527862"
## [1] "precision of test data:  0.796812749003984"
## [1] "accuracy of test data:  0.750778816199377"
## [1] "recall of test data:  0.873362445414847"
## [1] "F-score of test data:  0.833333333333333"
## [1] -0.00477154
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  417  54
##   LOW   110  98
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  202  27
##   LOW    49  43
## [1] "Train data error:  0.241531664212077"
## [1] "Test data error:  0.236760124610592"
## [1] "Difference/performance -0.00477153960148469"
## [1] "precision of training data:  0.791271347248577"
## [1] "accuracy of training data:  0.758468335787923"
## [1] "recall of training data:  0.885350318471338"
## [1] "F-score of training data:  0.835671342685371"
## [1] "precision of test data:  0.804780876494024"
## [1] "accuracy of test data:  0.763239875389408"
## [1] "recall of test data:  0.882096069868996"
## [1] "F-score of test data:  0.841666666666667"
```

MODEL 3: 80:20 SPLIT

```r
# Assigning 1 & 2 as index to split test and train data
set.seed(96)
index <- sample(2, nrow(data), replace = T, prob = c(0.8,0.2))
```

```r
#selecting index 1 for training
train <- data[index == 1,]

#selecting index 1 for testing
test <- data[index == 2,]

#Creating formula with all the response variables using ., to serve as an
input parameter to rpart.
MyFormula = RESPONSE ~.

mytree_80_20_basic = rpart(MyFormula, data=train)
print(mytree_80_20_basic)

## n= 786
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 786 240 HIGH (0.69465649 0.30534351)
##     2) CHK_ACCT=2,3 346  43 HIGH (0.87572254 0.12427746) *
##     3) CHK_ACCT=0,1 440 197 HIGH (0.55227273 0.44772727)
##       6) DURATION< 22.5 244  83 HIGH (0.65983607 0.34016393)
##        12) HISTORY=2,3,4 223  65 HIGH (0.70852018 0.29147982) *
##        13) HISTORY=0,1 21   3 LOW (0.14285714 0.85714286) *
##       7) DURATION>=22.5 196  82 LOW (0.41836735 0.58163265)
##        14) SAV_ACCT=3,4 34  11 HIGH (0.67647059 0.32352941)
##          28) CHK_ACCT=1 17   1 HIGH (0.94117647 0.05882353) *
##          29) CHK_ACCT=0 17   7 LOW (0.41176471 0.58823529) *
##        15) SAV_ACCT=0,1,2 162  59 LOW (0.36419753 0.63580247)
##          30) DURATION< 43.5 128  54 LOW (0.42187500 0.57812500)
##            60) USED_CAR=1 18   5 HIGH (0.72222222 0.27777778) *
##            61) USED_CAR=0 110  41 LOW (0.37272727 0.62727273)
##             122) NEW_CAR=0 87  37 LOW (0.42528736 0.57471264)
##               244) EMPLOYMENT=3,4 32  14 HIGH (0.56250000 0.43750000)
##                 488) SAV_ACCT=1 7   0 HIGH (1.00000000 0.00000000) *
##                 489) SAV_ACCT=0 25  11 LOW (0.44000000 0.56000000)
##                   978) HISTORY=2,4 17   6 HIGH (0.64705882 0.35294118) *
##                   979) HISTORY=0,1,3 8   0 LOW (0.00000000 1.00000000) *
##               245) EMPLOYMENT=0,1,2 55  19 LOW (0.34545455 0.65454545)
##                 490) AGE< 28.5 23  10 HIGH (0.56521739 0.43478261)
##                   980) HISTORY=1,2 15   4 HIGH (0.73333333 0.26666667) *
##                   981) HISTORY=0,4 8   2 LOW (0.25000000 0.75000000) *
##                 491) AGE>=28.5 32   6 LOW (0.18750000 0.81250000) *
##             123) NEW_CAR=1 23   4 LOW (0.17391304 0.82608696) *
##          31) DURATION>=43.5 34   5 LOW (0.14705882 0.85294118) *

#Predict function to predict the classes for the decision tree
mytree_70_30_basic for training data.
mytree_train_predict_80_20 <- predict(mytree_80_20_basic, data = train , type
= "class")
```

```r
#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_80_20 <- mean(mytree_train_predict_80_20 !=
train$RESPONSE)
mytree_train_error_80_20

## [1] 0.192112

#Predict function to predict the classes for the decision tree mytree_80_20
for testing data.
mytree_test_predict_80_20 <- predict(mytree_80_20_basic, newdata = test, type
= "class")

#Calculating the testing error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_80_20 <- mean(mytree_test_predict_80_20 != test$RESPONSE)
mytree_test_error_80_20

## [1] 0.2663551

#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_80_20 = mytree_test_error_80_20 - mytree_train_error_80_20

print(diff_80_20)

## [1] 0.07424318

knitr::include_graphics("CP5.png")
```

```
CP          nsplit rel error    xerror       xstd
1 0.06666667      0 1.0000000 1.0000000 0.05379965
2 0.06250000      2 0.8666667 0.9458333 0.05294151
3 0.05000000      3 0.8041667 0.8791667 0.05176696
4 0.01666667      4 0.7541667 0.8166667 0.05053957
5 0.01250000      6 0.7208333 0.8666667 0.05153153
6 0.01000000     13 0.6291667 0.8666667 0.05153153
```

APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR 80-20 MODEL

Creating vectors for minsplit and minbucket values to be used for different combinations to test performance. Based on above mentioned CP values,based on leaset xerror we will choose CP = 0.01000000

```
msplt <- c(12,48,102)
mbckt <- c(4,16,34)
for (i in msplt)
{
  for (j in mbckt)
  {

mytree_80_20 <- rpart(MyFormula, data = train, parms =
list(split="gini"),control = rpart.control (minsplit = i,minbucket =
j,cp=0.01000000))
#print(mytree_80_20)

mytree_train_predict_80_20 <- predict(mytree_80_20, data = train , type =
"class")
?predict
mytree_train_predict_80_20
mytree_train_error_80_20 <- mean(mytree_train_predict_80_20 !=
train$RESPONSE)
mytree_train_error_80_20

mytree_test_predict_80_20 <- predict(mytree_80_20, newdata = test, type =
"class")
mytree_test_predict_80_20

mytree_test_error_80_20 <- mean(mytree_test_predict_80_20 != test$RESPONSE)
mytree_test_error_80_20

diff_80_20 = mytree_test_error_80_20 - mytree_train_error_80_20
diff_80_20
print(diff_80_20)

############################## Confusion Matrix for 80:20 Split
#################################################

cfmt <- table(train$RESPONSE,mytree_train_predict_80_20)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]
```

```
#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_train =  (tp)/(tp+fp)
accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
recall_train = (tp)/(tp+fn)
fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)

cfmt <- table(test$RESPONSE,mytree_test_predict_80_20)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_test)

# Printing the values for train data error, test data error, performance and
other parameters.
print(paste("Train data error: ", mytree_train_error_80_20))
print(paste("Test data error: ", mytree_test_error_80_20))
print(paste("Difference/performance", diff_80_20))
print(paste("precision of training data: ", precision_train))
print(paste("accuracy of training data: ", accuracymodel_train))
print(paste("recall of training data: ", recall_train))
print(paste("F-score of training data: ", fscore_train))
print(paste("precision of test data: ", precision_test))
print(paste("accuracy of test data: ", accuracymodel_test))
print(paste("recall of test data: ", recall_test))
print(paste("F-score of test data: ", fscore_test))
  }}

## [1] 0.08395758
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  513  33
##   LOW    92 148
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  139  15
##   LOW    37  23
## [1] "Train data error:  0.159033078880407"
## [1] "Test data error:  0.242990654205607"
## [1] "Difference/performance 0.0839575753252003"
```

```
## [1] "precision of training data:  0.847933884297521"
## [1] "accuracy of training data:  0.840966921119593"
## [1] "recall of training data:  0.93956043956044"
## [1] "F-score of training data:  0.891398783666377"
## [1] "precision of test data:  0.789772727272727"
## [1] "accuracy of test data:  0.757009345794392"
## [1] "recall of test data:  0.902597402597403"
## [1] "F-score of test data:  0.842424242424242"
## [1] 0.04072436
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  490  56
##   LOW   114 126
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  138  16
##   LOW    39  21
## [1] "Train data error:  0.216284987277354"
## [1] "Test data error:  0.257009345794392"
## [1] "Difference/performance 0.0407243585170388"
## [1] "precision of training data:  0.811258278145695"
## [1] "accuracy of training data:  0.783715012722646"
## [1] "recall of training data:  0.897435897435897"
## [1] "F-score of training data:  0.852173913043478"
## [1] "precision of test data:  0.779661016949153"
## [1] "accuracy of test data:  0.742990654205608"
## [1] "recall of test data:  0.896103896103896"
## [1] "F-score of test data:  0.833836858006042"
## [1] 0.04329267
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  492  54
##   LOW   125 115
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  137  17
##   LOW    41  19
## [1] "Train data error:  0.227735368956743"
## [1] "Test data error:  0.271028037383178"
## [1] "Difference/performance 0.0432926684264345"
## [1] "precision of training data:  0.79740680713128"
## [1] "accuracy of training data:  0.772264631043257"
## [1] "recall of training data:  0.901098901098901"
## [1] "F-score of training data:  0.846087704213242"
## [1] "precision of test data:  0.769662921348315"
## [1] "accuracy of test data:  0.728971962616822"
## [1] "recall of test data:  0.88961038961039"
## [1] "F-score of test data:  0.825301204819277"
## [1] 0.04324511
##       mytree_train_predict_80_20
```

```
##        HIGH LOW
##   HIGH  489  57
##   LOW   100 140
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  137  17
##   LOW    35  25
## [1] "Train data error:  0.199745547073791"
## [1] "Test data error:  0.242990654205607"
## [1] "Difference/performance 0.0432451071318161"
## [1] "precision of training data:  0.830220713073005"
## [1] "accuracy of training data:  0.800254452926209"
## [1] "recall of training data:  0.895604395604396"
## [1] "F-score of training data:  0.861674008810573"
## [1] "precision of test data:  0.796511627906977"
## [1] "accuracy of test data:  0.757009345794392"
## [1] "recall of test data:  0.88961038961039"
## [1] "F-score of test data:  0.840490797546012"
## [1] 0.03690756
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  497  49
##   LOW   124 116
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  138  16
##   LOW    39  21
## [1] "Train data error:  0.220101781170483"
## [1] "Test data error:  0.257009345794392"
## [1] "Difference/performance 0.036907564623909"
## [1] "precision of training data:  0.800322061191626"
## [1] "accuracy of training data:  0.779898218829516"
## [1] "recall of training data:  0.91025641025641"
## [1] "F-score of training data:  0.851756640959726"
## [1] "precision of test data:  0.779661016949153"
## [1] "accuracy of test data:  0.742990654205608"
## [1] "recall of test data:  0.896103896103896"
## [1] "F-score of test data:  0.833836858006042"
## [1] 0.04329267
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  492  54
##   LOW   125 115
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  137  17
##   LOW    41  19
## [1] "Train data error:  0.227735368956743"
## [1] "Test data error:  0.271028037383178"
## [1] "Difference/performance 0.0432926684264345"
```

```
## [1] "precision of training data:  0.79740680713128"
## [1] "accuracy of training data:  0.772264631043257"
## [1] "recall of training data:  0.901098901098901"
## [1] "F-score of training data:  0.846087704213242"
## [1] "precision of test data:  0.769662921348315"
## [1] "accuracy of test data:  0.728971962616822"
## [1] "recall of test data:  0.88961038961039"
## [1] "F-score of test data:  0.825301204819277"
## [1] 0.04241278
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  494  52
##   LOW   113 127
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  137  17
##   LOW    37  23
## [1] "Train data error:  0.209923664122137"
## [1] "Test data error:  0.252336448598131"
## [1] "Difference/performance 0.0424127844759934"
## [1] "precision of training data:  0.813838550247117"
## [1] "accuracy of training data:  0.790076335877863"
## [1] "recall of training data:  0.904761904761905"
## [1] "F-score of training data:  0.856895056374675"
## [1] "precision of test data:  0.78735632183908"
## [1] "accuracy of test data:  0.747663551401869"
## [1] "recall of test data:  0.88961038961039"
## [1] "F-score of test data:  0.835365853658537"
## [1] 0.03690756
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  497  49
##   LOW   124 116
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  138  16
##   LOW    39  21
## [1] "Train data error:  0.220101781170483"
## [1] "Test data error:  0.257009345794392"
## [1] "Difference/performance 0.036907564623909"
## [1] "precision of training data:  0.800322061191626"
## [1] "accuracy of training data:  0.779898218829516"
## [1] "recall of training data:  0.91025641025641"
## [1] "F-score of training data:  0.851756640959726"
## [1] "precision of test data:  0.779661016949153"
## [1] "accuracy of test data:  0.742990654205608"
## [1] "recall of test data:  0.896103896103896"
## [1] "F-score of test data:  0.833836858006042"
## [1] 0.01909586
##       mytree_train_predict_80_20
```

```
##         HIGH LOW
##    HIGH  465  81
##    LOW   106 134
##        mytree_test_predict_80_20
##         HIGH LOW
##    HIGH  133  21
##    LOW    34  26
## [1] "Train data error:  0.237913486005089"
## [1] "Test data error:  0.257009345794392"
## [1] "Difference/performance 0.0190958597893034"
## [1] "precision of training data:  0.814360770577933"
## [1] "accuracy of training data:  0.762086513994911"
## [1] "recall of training data:  0.851648351648352"
## [1] "F-score of training data:  0.832587287376902"
## [1] "precision of test data:  0.796407185628742"
## [1] "accuracy of test data:  0.742990654205608"
## [1] "recall of test data:  0.863636363636364"
## [1] "F-score of test data:  0.828660436137072"
```

```
mytree_80_20_basic = rpart(MyFormula, data=train,parms =
list(split="information"))
```

```
knitr::include_graphics("CP6.png")
```

	CP	nsplit	rel error	xerror	xstd
1	0.06666667	0	1.0000000	1.000000	0.05379965
2	0.06250000	2	0.8666667	1.033333	0.05428683
3	0.03333333	3	0.8041667	0.825000	0.05071060
4	0.01458333	5	0.7375000	0.825000	0.05071060
5	0.01000000	11	0.6375000	0.912500	0.05237102

Based on the above summary for information gain, the CP to be considered is 0.01000000 for least x error

```
for (i in msplt)
{
  for (j in mbckt)
  {
#Constructing the decision tree based on information gain for 80-20 split
```

```
mytree_80_20_info <- rpart(MyFormula, data = train, parms =
list(split="information"),control = rpart.control (minsplit = i,minbucket =
j,cp=0.01000000))

#print(mytree_80_20_info)

mytree_train_predict_80_20 <- predict(mytree_80_20_info, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_80_20 <- mean(mytree_train_predict_80_20 !=
train$RESPONSE)
mytree_train_error_80_20

#Predict function to predict the classes for the decision tree mytree_80_20
for testing data.
mytree_test_predict_80_20 <- predict(mytree_80_20_info, newdata = test, type
= "class")
mytree_test_predict_80_20

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_80_20 <- mean(mytree_test_predict_80_20 != test$RESPONSE)
mytree_test_error_80_20

#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_80_20 = mytree_test_error_80_20 - mytree_train_error_80_20
print(diff_80_20)

############################## Confusion Matrix for 80:20 Split
#################################################

cfmt <- table(train$RESPONSE,mytree_train_predict_80_20)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_train =  (tp)/(tp+fp)
accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
recall_train = (tp)/(tp+fn)
fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)
```

```
cfmt <- table(test$RESPONSE,mytree_test_predict_80_20)
print(cfmt)

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_test)


print(paste("Train data error: ", mytree_train_error_80_20))
print(paste("Test data error: ", mytree_test_error_80_20))
print(paste("Difference/performance", diff_80_20))
print(paste("precision of training data: ", precision_train))
print(paste("accuracy of training data: ", accuracymodel_train))
print(paste("recall of training data: ", recall_train))
print(paste("F-score of training data: ", fscore_train))
print(paste("precision of test data: ", precision_test))
print(paste("accuracy of test data: ", accuracymodel_test))
print(paste("recall of test data: ", recall_test))
print(paste("F-score of test data: ", fscore_test))
  }
}

## [1] 0.08657345
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  508  38
##   LOW   118 122
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  132  22
##   LOW    39  21
## [1] "Train data error:  0.198473282442748"
## [1] "Test data error:  0.285046728971963"
## [1] "Difference/performance 0.0865734465292145"
## [1] "precision of training data:  0.811501597444089"
## [1] "accuracy of training data:  0.801526717557252"
## [1] "recall of training data:  0.93040293040293"
## [1] "F-score of training data:  0.866894197952218"
```

```
## [1] "precision of test data:  0.771929824561403"
## [1] "accuracy of test data:  0.714953271028037"
## [1] "recall of test data:  0.857142857142857"
## [1] "F-score of test data:  0.812307692307692"
## [1] 0.1252051
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  498  48
##   LOW    96 144
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  126  28
##   LOW    38  22
## [1] "Train data error:  0.183206106870229"
## [1] "Test data error:  0.308411214953271"
## [1] "Difference/performance 0.125205108083042"
## [1] "precision of training data:  0.838383838383838"
## [1] "accuracy of training data:  0.816793893129771"
## [1] "recall of training data:  0.912087912087912"
## [1] "F-score of training data:  0.873684210526316"
## [1] "precision of test data:  0.768292682926829"
## [1] "accuracy of test data:  0.691588785046729"
## [1] "recall of test data:  0.818181818181818"
## [1] "F-score of test data:  0.792452830188679"
## [1] 0.04329267
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  492  54
##   LOW   125 115
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  137  17
##   LOW    41  19
## [1] "Train data error:  0.227735368956743"
## [1] "Test data error:  0.271028037383178"
## [1] "Difference/performance 0.0432926684264345"
## [1] "precision of training data:  0.79740680713128"
## [1] "accuracy of training data:  0.772264631043257"
## [1] "recall of training data:  0.901098901098901"
## [1] "F-score of training data:  0.846087704213242"
## [1] "precision of test data:  0.769662921348315"
## [1] "accuracy of test data:  0.728971962616822"
## [1] "recall of test data:  0.88961038961039"
## [1] "F-score of test data:  0.825301204819277"
## [1] 0.08446886
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  512  34
##   LOW   131 109
##       mytree_test_predict_80_20
```

```
##        HIGH LOW
##   HIGH  132  22
##   LOW    41  19
## [1] "Train data error:  0.209923664122137"
## [1] "Test data error:  0.294392523364486"
## [1] "Difference/performance 0.0844688592423486"
## [1] "precision of training data:  0.796267496111975"
## [1] "accuracy of training data:  0.790076335877863"
## [1] "recall of training data:  0.937728937728938"
## [1] "F-score of training data:  0.861227922624054"
## [1] "precision of test data:  0.763005780346821"
## [1] "accuracy of test data:  0.705607476635514"
## [1] "recall of test data:  0.857142857142857"
## [1] "F-score of test data:  0.807339449541284"
## [1] 0.08488502
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  519  27
##   LOW   134 106
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  135  19
##   LOW    43  17
## [1] "Train data error:  0.204834605597964"
## [1] "Test data error:  0.289719626168224"
## [1] "Difference/performance 0.0848850205702599"
## [1] "precision of training data:  0.7947932618683"
## [1] "accuracy of training data:  0.795165394402036"
## [1] "recall of training data:  0.950549450549451"
## [1] "F-score of training data:  0.865721434528774"
## [1] "precision of test data:  0.758426966292135"
## [1] "accuracy of test data:  0.710280373831776"
## [1] "recall of test data:  0.876623376623377"
## [1] "F-score of test data:  0.813253012048193"
## [1] 0.04329267
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  492  54
##   LOW   125 115
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  137  17
##   LOW    41  19
## [1] "Train data error:  0.227735368956743"
## [1] "Test data error:  0.271028037383178"
## [1] "Difference/performance 0.0432926684264345"
## [1] "precision of training data:  0.79740680713128"
## [1] "accuracy of training data:  0.772264631043257"
## [1] "recall of training data:  0.901098901098901"
## [1] "F-score of training data:  0.846087704213242"
```

```
## [1] "precision of test data:  0.769662921348315"
## [1] "accuracy of test data:  0.728971962616822"
## [1] "recall of test data:  0.88961038961039"
## [1] "F-score of test data:  0.825301204819277"
## [1] 0.06154432
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  489  57
##   LOW   115 125
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  131  23
##   LOW    37  23
## [1] "Train data error:  0.21882951653944"
## [1] "Test data error:  0.280373831775701"
## [1] "Difference/performance 0.0615443152362607"
## [1] "precision of training data:  0.809602649006622"
## [1] "accuracy of training data:  0.78117048346056"
## [1] "recall of training data:  0.895604395604396"
## [1] "F-score of training data:  0.850434782608696"
## [1] "precision of test data:  0.779761904761905"
## [1] "accuracy of test data:  0.719626168224299"
## [1] "recall of test data:  0.850649350649351"
## [1] "F-score of test data:  0.813664596273292"
## [1] 0.06876174
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  527  19
##   LOW   151  89
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  139  15
##   LOW    46  14
## [1] "Train data error:  0.216284987277354"
## [1] "Test data error:  0.285046728971963"
## [1] "Difference/performance 0.0687617416946089"
## [1] "precision of training data:  0.777286135693215"
## [1] "accuracy of training data:  0.783715012722646"
## [1] "recall of training data:  0.965201465201465"
## [1] "F-score of training data:  0.861111111111111"
## [1] "precision of test data:  0.751351351351351"
## [1] "accuracy of test data:  0.714953271028037"
## [1] "recall of test data:  0.902597402597403"
## [1] "F-score of test data:  0.820058997050148"
## [1] 0.01909586
##       mytree_train_predict_80_20
##        HIGH LOW
##   HIGH  465  81
##   LOW   106 134
##       mytree_test_predict_80_20
```

```
##        HIGH LOW
##   HIGH  133  21
##   LOW    34  26
## [1] "Train data error:  0.237913486005089"
## [1] "Test data error:  0.257009345794392"
## [1] "Difference/performance 0.0190958597893034"
## [1] "precision of training data:  0.814360770577933"
## [1] "accuracy of training data:  0.762086513994911"
## [1] "recall of training data:  0.851648351648352"
## [1] "F-score of training data:  0.832587287376902"
## [1] "precision of test data:  0.796407185628742"
## [1] "accuracy of test data:  0.742990654205608"
## [1] "recall of test data:  0.863636363636364"
## [1] "F-score of test data:  0.828660436137072"
```

```
knitr::include_graphics("ModelValues.png")
```

50:50 Split - Basic - Pre Pruning				
		Train data error	Test data error	error
		0.2805611	0.1836327	0.09692839
50:50 Split - Post Pruning				
Gini (C&R), CP = 0.026490007				
minsplit	minbucket	Train data error	Test data error	error
12	4	0.243512974	0.256513026	0.013000052
	16	0.243512974	0.256513026	0.01300005
	34	0.24351297	0.256513026	0.0510122
48	4	0.213572854	0.278557114	0.013000052
	16	0.24351297	0.256513026	0.013000052
	34	0.243512974	0.256513026	0.013000052
102	4	0.243512974	0.256513026	0.013000052
	16	0.24351297	0.256513026	0.013000052
	34	0.243512974	0.256513026	0.013000052
Info Gain 50:50				
12	4	0.161676646	0.27254509	0.110868443
	16	0.219560878	0.28256513	0.063004252
	34	0.22754491	0.278557114	0.051012204
48	4	0.199600798	0.276553106	0.076952308
	16	0.223552894	0.290581162	0.067028268
	34	0.22754491	0.278557114	0.051012204
102	4	0.243512974	0.256513026	0.013000052
	16	0.243512974	0.256513026	0.013000052
	34	0.243512974	0.256513026	0.013000052
	min gini			0.01300005
	min info gain			0.013000052

70:30 Split - Basic - Pre Pruning				
		Train data error	Test data error	error
		0.2492212	0.1870398	0.06218142
70:30 Split - Post Pruning				
Gini 70:30				
minsplit	minbucket	Train data error	Test data error	error
12	4	0.173784977	0.249221184	0.07543621
	16	0.198821797	0.267912773	0.06909098
	34	0.241531664	0.236760125	-0.0047715
48	4	0.192930781	0.280373832	0.08744305
	16	0.203240059	0.277258567	0.07401851
	34	0.241531664	0.236760125	-0.0047715
102	4	0.231222386	0.242990654	0.01176827
	16	0.237113402	0.242990654	0.00587725
	34	0.241531664	0.236760125	-0.0047715
Info Gain 70:30				
12	4	0.191458027	0.258566978	0.06710895
	16	0.188512518	0.271028037	0.08251552
	34	0.241531664	0.236760125	-0.0047715
48	4	0.195876289	0.271028037	0.07515175
	16	0.201767305	0.264797508	0.0630302
	34	0.241531664	0.236760125	-0.0047715
102	4	0.22533137	0.249221184	0.02388981
	16	0.231222386	0.249221184	0.0179988
	34	0.241531664	0.236760125	-0.0047715
	min gini			-0.0047715
	min info gain			-0.0047715

80:20 Split - Basic - Pre Pruning				
		Train data error	Test data e	error
		0.2663551	0.192112	0.074243
80:20 Split - Post Pruning				
Gini 80:20				
minsplit	minbucket	Train data error	Test data e	error
12	4	0.159033079	0.2429907	0.083958
	16	0.216284987	0.2570093	0.040724
	34	0.227735369	0.271028	0.043293
48	4	0.199745547	0.2429907	0.043245
	16	0.220101781	0.2570093	0.036908
	34	0.227735369	0.271028	0.043293
102	4	0.209923664	0.2523364	0.042413
	16	0.220101781	0.2570093	0.036908
	34	0.237913486	0.2570093	0.019096
Info Gain 80:20				
12	4	0.198473282	0.2850467	0.086573
	16	0.183206107	0.3084112	0.125205
	34	0.227735369	0.271028	0.043293
48	4	0.209923664	0.2943925	0.084469
	16	0.204834606	0.2897196	0.084885
	34	0.227735369	0.271028	0.043293
102	4	0.218829517	0.2803738	0.061544
	16	0.216284987	0.2850467	0.068762
	34	0.237913486	0.2570093	0.019096
	min gini			0.019096
	min info gain			0.019096

Que: Is there any specific model you would prefer to implement?

Prior to the pruning techniques, the 70:30 model provides the least error rate & high performance therefore, this model can be preferred.

Que: Also, does pruning give a better model – please explain why or why not?

The 70:30 split model(post pruning, min split = 102, minbucket 4/16/34) can be implemented as this gives the least error and therefore best performance.

Que: Moreover this model performs similar on train and test data so that indicates the stability of the model.Do you see any performance differences across different types of decision tree learners?

Based on the above mentioned performance indicators based on minsplit, minbucket across information gain(C5.0) & gini(C&R) split, the performance values are consistent across both the methods.

Problem c)

We are considering 'HIGH' for 'GOOD applicants' and 'LOW' for 'BAD applicants'. From question b) 70-30 split is giving the best accuracy with least error. Hence, for question c) we are considering 70-30 split decision tree for missclassification cost analysis.

Assigning 1 & 2 as index to split test and train data.

Selected Variables = CHK_ACCT, HISTORY, SAV_ACCT, EMPLOYMENT,OWN_RES,REAL_ESTATE, OTHER_INSTALL,USED_CAR, NEW_CAR,RENT

```
set.seed(96)
index <- sample(2, nrow(data), replace = T, prob = c(0.7,0.3))
```

selecting index 1 for training

```
train <- data[index == 1,]
```

selecting index 1 for testing

```
test <- data[index == 2,]
```

Creating formula for all and selected values.

```
MyFormula_allValues = RESPONSE ~ .
MyFormula_selectedValues = RESPONSE ~ CHK_ACCT + HISTORY  + SAV_ACCT +
EMPLOYMENT +
  OWN_RES + REAL_ESTATE + OTHER_INSTALL + USED_CAR + NEW_CAR + RENT
```

creating loss matrix with 1:5 ratio for TN:FP

```
lossMatrix <- matrix(c(0, 1, 5, 0), byrow=TRUE, ncol=2)
lossMatrix

##      [,1] [,2]
## [1,]    0    1
## [2,]    5    0
```

Creating three trees to analyze misscalssification cost 1) Without adding Loss and All Variables 2) With adding Loss and All Variables 3) With adding Loss and Selected Variables

```
mytree_wol <- rpart(MyFormula_allValues, data = train)
mytree_wl_av <- rpart(MyFormula_allValues, data = train, parms =
list(loss=lossMatrix))
mytree_wl_sv <- rpart(MyFormula_selectedValues, data = train, parms =
list(loss=lossMatrix))
```

Train data for Tree Without Loss and All values.

```
mytree_wol_pTr <- predict(mytree_wol, data = train , type = "class")
cf_wol_tr<-table(actual = train$RESPONSE, pred = mytree_wol_pTr)
cf_wol_tr
```

```
##       pred
## actual HIGH LOW
##   HIGH  426  45
##   LOW    82 126
```

```
cf_wol_tr_accuracy<-(cf_wol_tr[1,1]+cf_wol_tr[2,2])/
(cf_wol_tr[1,1]+cf_wol_tr[1,2]+cf_wol_tr[2,1]+cf_wol_tr[2,2])
cf_wol_tr_error<- 1 - cf_wol_tr_accuracy
cf_wol_tr_precision<-(cf_wol_tr[1,1]/(cf_wol_tr[1,1]+cf_wol_tr[2,1]))
```

Train data for Tree With Loss and All values.

```
mytree_wl_av_pTr <- predict(mytree_wl_av, data = train , type = "class")
cf_wl_av_tr<-table(actual = train$RESPONSE, pred = mytree_wl_av_pTr)
cf_wl_av_tr
```

```
##       pred
## actual HIGH LOW
##   HIGH  239 232
##   LOW     8 200
```

```
cf_wl_av_tr_accuracy<-(cf_wl_av_tr[1,1]+cf_wl_av_tr[2,2])/
(cf_wl_av_tr[1,1]+cf_wl_av_tr[1,2]+cf_wl_av_tr[2,1]+cf_wl_av_tr[2,2])
cf_wl_av_tr_error<- 1 - cf_wl_av_tr_accuracy
cf_wl_av_tr_precision<-(cf_wl_av_tr[1,1]/(cf_wl_av_tr[1,1]+cf_wl_av_tr[2,1]))
```

Train data for Tree With Loss and selected values.

```
mytree_wl_sv_pTr <- predict(mytree_wl_sv, data = train , type = "class")
cf_wl_sv_tr<-table(actual = train$RESPONSE, pred = mytree_wl_sv_pTr)
cf_wl_sv_tr
```

```
##       pred
## actual HIGH LOW
##   HIGH  183 288
##   LOW     6 202
```

```
cf_wl_sv_tr_accuracy<-(cf_wl_sv_tr[1,1]+cf_wl_sv_tr[2,2])/
(cf_wl_sv_tr[1,1]+cf_wl_sv_tr[1,2]+cf_wl_sv_tr[2,1]+cf_wl_sv_tr[2,2])
cf_wl_sv_tr_error<- 1 - cf_wl_sv_tr_accuracy
cf_wl_sv_tr_precision<-(cf_wl_sv_tr[1,1]/(cf_wl_sv_tr[1,1]+cf_wl_sv_tr[2,1]))



df_tr <- data.frame(TreeType = c("Tree Without Loss All Variables", "Tree
With Loss All Variables", "Tree With Loss Selected Variables"),
                 Accuracy = c(round(cf_wol_tr_accuracy,digits =2),
round(cf_wl_av_tr_accuracy,digits =2), round(cf_wl_sv_tr_accuracy,digits
=2)),
                 Error = c(round(cf_wol_tr_error,digits =2),
round(cf_wl_av_tr_error,digits =2), round(cf_wl_sv_tr_error,digits =2)),
                 Precision = c(round(cf_wol_tr_precision,digits =2),
round(cf_wl_av_tr_precision,digits =2), round(cf_wl_sv_tr_precision,digits
```

```
=2))
)
```

Test data for Tree Without Loss and All values.

```
mytree_wol_pTs <- predict(mytree_wol, newdata = test , type = "class")
cf_wol_ts<-table(actual = test$RESPONSE, pred = mytree_wol_pTs)
cf_wol_ts

##       pred
## actual HIGH LOW
##   HIGH  197  32
##   LOW    48  44

cf_wol_ts_accuracy<-(cf_wol_ts[1,1]+cf_wol_ts[2,2])/
(cf_wol_ts[1,1]+cf_wol_ts[1,2]+cf_wol_ts[2,1]+cf_wol_ts[2,2])
cf_wol_ts_error<- 1 - cf_wol_ts_accuracy
cf_wol_ts_precision<-(cf_wol_ts[1,1]/(cf_wol_ts[1,1]+cf_wol_ts[2,1]))
```

Test data for Tree With Loss and All values.

```
mytree_wl_av_pTs <- predict(mytree_wl_av, newdata = test , type = "class")
cf_wl_av_ts<-table(actual = test$RESPONSE, pred = mytree_wl_av_pTs)
cf_wl_av_ts

##       pred
## actual HIGH LOW
##   HIGH  106 123
##   LOW    16  76

cf_wl_av_ts_accuracy<-(cf_wl_av_ts[1,1]+cf_wl_av_ts[2,2])/
(cf_wl_av_ts[1,1]+cf_wl_av_ts[1,2]+cf_wl_av_ts[2,1]+cf_wl_av_ts[2,2])
cf_wl_av_ts_error<- 1 - cf_wl_av_ts_accuracy
cf_wl_av_ts_precision<-(cf_wl_av_ts[1,1]/(cf_wl_av_ts[1,1]+cf_wl_av_ts[2,1]))
```

Test data for Tree With Loss and Selected values.

```
mytree_wl_sv_pTs <- predict(mytree_wl_sv, newdata = test , type = "class")
cf_wl_sv_ts<-table(actual = test$RESPONSE, pred = mytree_wl_sv_pTs)
cf_wl_sv_ts

##       pred
## actual HIGH LOW
##   HIGH   95 134
##   LOW    11  81

cf_wl_sv_ts_accuracy<-(cf_wl_sv_ts[1,1]+cf_wl_sv_ts[2,2])/
(cf_wl_sv_ts[1,1]+cf_wl_sv_ts[1,2]+cf_wl_sv_ts[2,1]+cf_wl_sv_ts[2,2])
cf_wl_sv_ts_error<- 1 - cf_wl_sv_ts_accuracy
cf_wl_sv_ts_precision<-(cf_wl_sv_ts[1,1]/(cf_wl_sv_ts[1,1]+cf_wl_sv_ts[2,1]))
```

```
df_ts <- data.frame(TreeType = c("Tree Without Loss All Variables", "Tree
With Loss All Variables", "Tree With Loss Selected Variables"),
                 Accuracy = c(round(cf_wol_ts_accuracy,digits =2),
round(cf_wl_av_ts_accuracy,digits =2), round(cf_wl_sv_ts_accuracy,digits
=2)),
                 Error = c(round(cf_wol_ts_error,digits =2),
round(cf_wl_av_ts_error,digits =2), round(cf_wl_sv_ts_error,digits =2)),
                 Precision = c(round(cf_wol_ts_precision,digits =2),
round(cf_wl_av_ts_precision,digits =2), round(cf_wl_sv_ts_precision,digits
=2))
)
```

On training data:

```
print(df_tr)

##                            TreeType Accuracy Error Precision
## 1   Tree Without Loss All Variables     0.81  0.19      0.84
## 2      Tree With Loss All Variables     0.65  0.35      0.97
## 3 Tree With Loss Selected Variables     0.57  0.43      0.97
```

On test data

```
print(df_ts)

##                            TreeType Accuracy Error Precision
## 1   Tree Without Loss All Variables     0.75  0.25      0.80
## 2      Tree With Loss All Variables     0.57  0.43      0.87
## 3 Tree With Loss Selected Variables     0.55  0.45      0.90
```

Yes, there are changes in the model/performance.

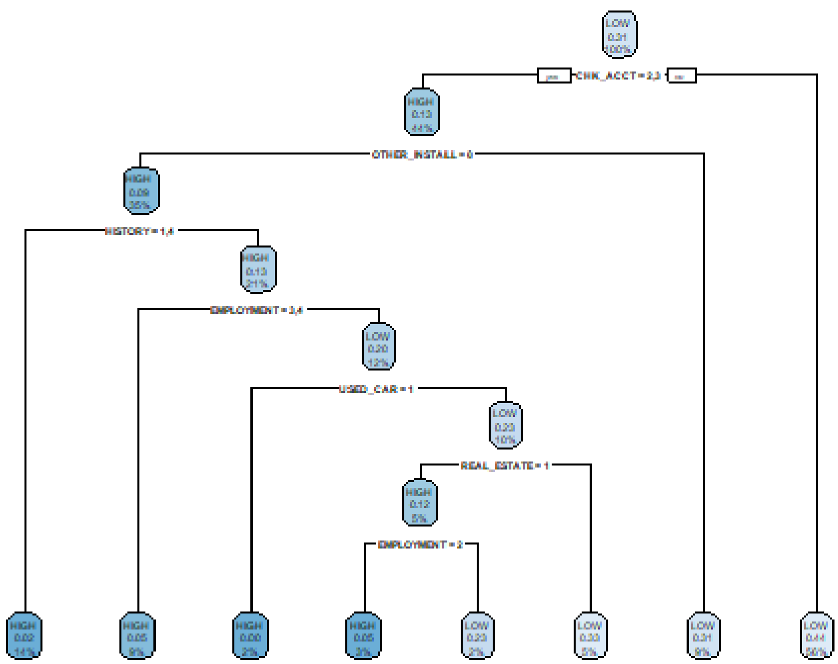
We can see a decrease in the Accuracy and increase in the Precision (Decrease in FALSE POSITIVE cases)

Benefits from specifying missclassification costs: Avoid FALSE POSITIVE cases (incorrectly saying an applicant is good credit risk)

Printing the decision tree with adding Loss and Selected Variables. As the precision is maximum in this case.

```
mytree_wl_sv

## n= 679
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 679 471 LOW (0.69366716 0.30633284)
##     2) CHK_ACCT=2,3 297 195 HIGH (0.86868687 0.13131313)
##       4) OTHER_INSTALL=0 238 105 HIGH (0.91176471 0.08823529)
##         8) HISTORY=1,4 96  10 HIGH (0.97916667 0.02083333) *
```

```
##          9) HISTORY=0,2,3 142  95 HIGH (0.86619718 0.13380282)
##           18) EMPLOYMENT=3,4 60  15 HIGH (0.95000000 0.05000000) *
##           19) EMPLOYMENT=0,1,2 82  66 LOW (0.80487805 0.19512195)
##             38) USED_CAR=1 12   0 HIGH (1.00000000 0.00000000) *
##             39) USED_CAR=0 70  54 LOW (0.77142857 0.22857143)
##               78) REAL_ESTATE=1 34  20 HIGH (0.88235294 0.11764706)
##                156) EMPLOYMENT=2 21   5 HIGH (0.95238095 0.04761905) *
##                157) EMPLOYMENT=0,1 13  10 LOW (0.76923077 0.23076923) *
##               79) REAL_ESTATE=0 36  24 LOW (0.66666667 0.33333333) *
##        5) OTHER_INSTALL=1 59  41 LOW (0.69491525 0.30508475) *
##      3) CHK_ACCT=0,1 382 213 LOW (0.55759162 0.44240838) *

rpart.plot(mytree_wl_sv)
```



Problem d)

From the tree in question c, we can derive the following decision rules for “Good” applicants.

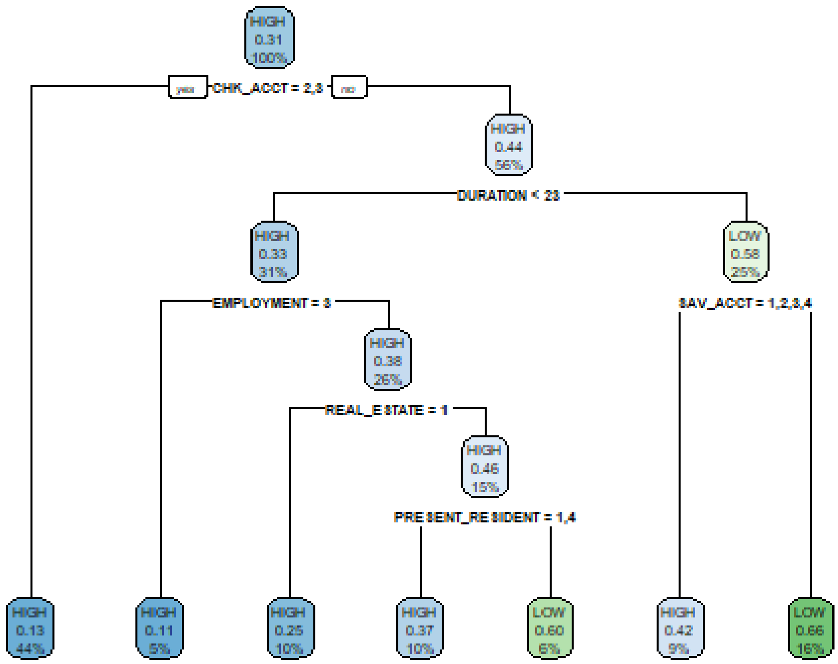
1) If CHK_ACCT equals 2 or 3 AND OTHER_INSTLL equals 0 AND HISTORY equals 1 or 4 THEN ‘GOOD Applicant’
2) If CHK_ACCT equals 2 or 3 AND OTHER_INSTLL equals 0 AND HISTORY is not equal to 1 or 4 AND EMPLOYEMENT equals 3 or 4 THEN ‘GOOD Applicant’

The above decision rules are chosen on the basis of having highest confidence value among all the other rules.

Problem e)

Findings for Problem b):

```
rpart.plot(mytree_70_30)
```



We are choosing the above decision tree as this provides the best performance which can be attributed to the difference between the training and testing error derived over various minsplit and minbucket values.

Findings for Problem c):

Our approach includes calculation of accuracy, error and precision of the below three trees 1) Without adding Loss and All Variables 2) With adding Loss and All Variables 3) With adding Loss and important Variables

We are considering precision because we want to reduce the False Positive (incorrectly saying an applicant is good credit risk) cases.

Precision = True Positive/(True Positive + False Positive)

On Training Data:

```
print(df_tr)

##                             TreeType Accuracy Error Precision
## 1   Tree Without Loss All Variables     0.81  0.19      0.84
```

```
## 2      Tree With Loss All Variables     0.65  0.35      0.97
## 3 Tree With Loss Selected Variables     0.57  0.43      0.97
```

In the above table, it is observed that, the accuracy without loss matrix is highest among all i.e 81% and its corresponding precision value is 84%.

In our case, it is important to INCREASE the precision i.e. lesser FALSE POSITIVE cases. So we consider the tree by adding loss.

After adding loss, it is observed that the Accuracy has decreased and Precision has increased. This is a trade-off that we have to consider.