IDS 572_Homework 2

Arnav Mediratta(673223798), Nikhil Kataria(667978072), Shreya Barahate(650802060)

21/02/2022

Question a:

Our aim here is to use the German credit data set to analyze good and bad credit risk associated with each individual. We will create some plots and perform analysis to determine good predictors that could possibly help us to predict credit risk. Initially, we imported the data set provided in the xls file 'German credit.xls' which consists of 1000 observations of 32 variables. We read the data set using the read_xls().

```
df <- import("German Credit.xls")%>%
  as_tibble()
```

We used the str function to understand the data in more details. We observed that there are no character variables in the data set provided.

```
str(df)
```

```
## 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 ...
```

After str(), we observed that:

1. The data is in numerical format. So, we thought of converting it into categories which would give us a more clean data for analysis.
2. The RESPONSE variable in the data set corresponds to the risk label and would be our target variable. 0 means 'bad' and 1 means 'good' credit risk. This observation was also made after referring the GermanCreditVariablesDefinition.pdf
3. Few of the binary variables are described under a specific category like 'Purpose of credit'. So we thought to merge these variables into one later to have good analysis on the predictor variables.

Also, we have used the summary() to check the mean and median values of all the numeric variables in the data set.

```
summary(df)
```

```
##       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
```

To further check whether the data set has any missing values or not, we used the is.na() and observed that there are no missing values in the entire data set.

```
na <- is.na(df)%>%
  head(5)
na
```

```
##       OBS# CHK_ACCT DURATION HISTORY NEW_CAR USED_CAR FURNITURE RADIO/TV
## [1,] FALSE    FALSE    FALSE   FALSE   FALSE    FALSE     FALSE    FALSE
## [2,] FALSE    FALSE    FALSE   FALSE   FALSE    FALSE     FALSE    FALSE
## [3,] FALSE    FALSE    FALSE   FALSE   FALSE    FALSE     FALSE    FALSE
## [4,] FALSE    FALSE    FALSE   FALSE   FALSE    FALSE     FALSE    FALSE
## [5,] FALSE    FALSE    FALSE   FALSE   FALSE    FALSE     FALSE    FALSE
##      EDUCATION RETRAINING AMOUNT SAV_ACCT EMPLOYMENT INSTALL_RATE MALE_DIV
## [1,]     FALSE      FALSE  FALSE    FALSE      FALSE        FALSE    FALSE
## [2,]     FALSE      FALSE  FALSE    FALSE      FALSE        FALSE    FALSE
## [3,]     FALSE      FALSE  FALSE    FALSE      FALSE        FALSE    FALSE
## [4,]     FALSE      FALSE  FALSE    FALSE      FALSE        FALSE    FALSE
## [5,]     FALSE      FALSE  FALSE    FALSE      FALSE        FALSE    FALSE
##      MALE_SINGLE MALE_MAR_or_WID CO-APPLICANT GUARANTOR PRESENT_RESIDENT
## [1,]       FALSE           FALSE        FALSE     FALSE            FALSE
## [2,]       FALSE           FALSE        FALSE     FALSE            FALSE
## [3,]       FALSE           FALSE        FALSE     FALSE            FALSE
## [4,]       FALSE           FALSE        FALSE     FALSE            FALSE
## [5,]       FALSE           FALSE        FALSE     FALSE            FALSE
##      REAL_ESTATE PROP_UNKN_NONE   AGE OTHER_INSTALL  RENT OWN_RES NUM_CREDITS
```

```
## [1,]       FALSE          FALSE FALSE         FALSE FALSE   FALSE       FALSE
## [2,]       FALSE          FALSE FALSE         FALSE FALSE   FALSE       FALSE
## [3,]       FALSE          FALSE FALSE         FALSE FALSE   FALSE       FALSE
## [4,]       FALSE          FALSE FALSE         FALSE FALSE   FALSE       FALSE
## [5,]       FALSE          FALSE FALSE         FALSE FALSE   FALSE       FALSE
##        JOB NUM_DEPENDENTS TELEPHONE FOREIGN RESPONSE
## [1,] FALSE           FALSE     FALSE   FALSE    FALSE
## [2,] FALSE           FALSE     FALSE   FALSE    FALSE
## [3,] FALSE           FALSE     FALSE   FALSE    FALSE
## [4,] FALSE           FALSE     FALSE   FALSE    FALSE
## [5,] FALSE           FALSE     FALSE   FALSE    FALSE
```

To know the proportion of "Good" to "Bad" cases in the dataset, we executed the following code to get the data. It can seen that, there are 700 "Good" cases and 300 "Bad" cases in the data set.

```
summary(as.factor(df$RESPONSE))
```

```
##   0   1
## 300 700
```

Tidying the data set

After referring to the GermanCreditVariablesDefinition.pdf we observed that there are 7 categorical variables : OBS#, CHK_ACCT, HISTORY, SAV_ACCT, EMPLOYMENT, PRESENT_RESIDENT, JOB These variables have values given in ranges which are indicated by 0,1,2,3,4 in the data set. To have a clear understanding of these values while performing EDA and plotting graphs, we replaced reach value in the respective variables with its corresponding text using the mutate().

```
#For HISTORY#
#For instance, '1: all credits at this bank paid back duly' has been renamed as 'ALL DUES PAID' for sho

data_new<-df %>%
  mutate(HISTORY=replace(HISTORY, HISTORY==0,"NO CREDITS TAKEN"))%>%
  mutate(HISTORY=replace(HISTORY, HISTORY==1,"ALL DUES PAID")) %>%
  mutate(HISTORY=replace(HISTORY, HISTORY==2,"EXISTING DUES PAID")) %>%
  mutate(HISTORY=replace(HISTORY, HISTORY==3,"DUES DELAYED")) %>%
  mutate(HISTORY=replace(HISTORY, HISTORY==4,"CRITICAL ACCOUNT"))
```

```
#For CHK_ACCT#

data_new<-data_new %>%
  mutate(CHK_ACCT=replace(CHK_ACCT, CHK_ACCT==0,"LESS THAN 0"), CHK_ACCT=replace(CHK_ACCT, CHK_ACCT==1,
```

```
#For SAV_ACCT#

data_new<-data_new %>%
  mutate(SAV_ACCT=replace(SAV_ACCT, SAV_ACCT==0,"LESS THAN 100"),SAV_ACCT=replace(SAV_ACCT, SAV_ACCT==1
```

```
#For EMPLOYMENT#

data_new<-data_new %>%
  mutate(EMPLOYMENT=replace(EMPLOYMENT, EMPLOYMENT==0,"UNEMPLOYED"),EMPLOYMENT=replace(EMPLOYMENT, EMPL
```

```r
#For PRESENT_RESIDENT#

data_new<-data_new %>%
  mutate(PRESENT_RESIDENT=replace(PRESENT_RESIDENT, PRESENT_RESIDENT==1,"LESS THAN 1 YEAR"),PRESENT_RESI
```

```r
#For JOB#

data_new<-data_new %>%
  mutate(JOB=replace(JOB, JOB==0,"UNEMP/UNSKILLED/NON-RES"), JOB=replace(JOB, JOB==1,"UNSKILLED RES"),J
```

We also replaced the values for binary variables (TELEPHONE, FOREIGN, RESPONSE, OTHER_INSTALL) with its corresponding text to have a more detail view while performing exploratory data analysis. With all these steps we changed the variables into characters.

```r
#For TELEPHONE,FOREIGN, RESPONSE, OTHER_INSTALL#

data_new<-data_new%>%
  mutate(TELEPHONE = ifelse(TELEPHONE == 0, "NO","YES"), FOREIGN = ifelse(FOREIGN == 0,"NO","YES"), RESP
```

To see the result of the above mutated character variables in a vector format i.e. for each variable, how many observations are present under each range, we used the sapply(). sapply() takes a data frame as input and returns a vector or a matrix as output.

Output: For instance, the HISTORY variable has 49 values in 'All Dues Paid', 293 in 'CRITICAL ACCOUNT', 88 in 'DUES DELAYED', 530 in 'DUES PAID' and 40 in 'NO CREDITS TAKEN'.

```r
sapply( data_new[ sapply(data_new, is.character)], table)
```

```
## $CHK_ACCT
##
##         BETWEEN 0 AND 200 GREATER THAN EQUAL TO 200               LESS THAN 0
##                       269                        63                       274
##       NO CHECKING ACCOUNT
##                       394
##
## $HISTORY
##
##      ALL DUES PAID   CRITICAL ACCOUNT       DUES DELAYED EXISTING DUES PAID
##                 49                293                 88                530
##   NO CREDITS TAKEN
##                 40
##
## $SAV_ACCT
##
##        BETWEEN 100 and 500       BETWEEN 500 AND 1000
##                        103                         63
## GREATER THAN EQUAL TO 1000              LESS THAN 100
##                         48                        603
##                 NO SAVINGS
##                        183
##
## $EMPLOYMENT
##
```

```
##          BETWEEN 1 AND 4 YRS EMPLOYED BETWEEN 4 AND 7 YRS
##                          339                          174
##     EMPLOYED MORE THAN 7 YRS               LESS THAN 1 YR
##                          253                          172
##                   UNEMPLOYED
##                           62
##
## $PRESENT_RESIDENT
##
##         BETWEEN 2 AND 3 YEARS GREATER THAN EQUAL TO 2 YEARS
##                           149                           308
##              LESS THAN 1 YEAR             MORE THAN 4 YEARS
##                           130                           413
##
## $OTHER_INSTALL
##
##  NO YES
## 814 186
##
## $JOB
##
## SELF/HIGHLY QUALIFIED EMP     SKILLED EMP/ OFFICIAL   UNEMP/UNSKILLED/NON-RES
##                       148                       630                        22
##             UNSKILLED RES
##                       200
##
## $TELEPHONE
##
##  NO YES
## 596 404
##
## $FOREIGN
##
##  NO YES
## 963  37
##
## $RESPONSE
##
##  NO YES
## 300 700
```

After performing the summary() on the data_new, we can see that the above variables are displayed as character with its class and mode, while the rest of the variables are numeric with its mean, median, 1st and 3rd quartile values.

```
summary(data_new)
```

```
##       OBS#          CHK_ACCT           DURATION      HISTORY
##  Min.   :   1.0   Length:1000        Min.   : 4.0   Length:1000
##  1st Qu.: 250.8   Class :character   1st Qu.:12.0   Class :character
##  Median : 500.5   Mode  :character   Median :18.0   Mode  :character
##  Mean   : 500.5                      Mean   :20.9
##  3rd Qu.: 750.2                      3rd Qu.:24.0
```

```
##  Max.   :1000.0                      Max.   :72.0
##     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   Length:1000        Length:1000
##  1st Qu.:0.000   1st Qu.: 1366   Class :character   Class :character
##  Median :0.000   Median : 2320   Mode  :character   Mode  :character
##  Mean   :0.097   Mean   : 3271
##  3rd Qu.:0.000   3rd Qu.: 3972
##  Max.   :1.000   Max.   :18424
##   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   Length:1000        Min.   :0.000   Min.   :0.000
##  1st Qu.:0.000   Class :character   1st Qu.:0.000   1st Qu.:0.000
##  Median :0.000   Mode  :character   Median :0.000   Median :0.000
##  Mean   :0.052                      Mean   :0.282   Mean   :0.154
##  3rd Qu.:0.000                      3rd Qu.:1.000   3rd Qu.:0.000
##  Max.   :1.000                      Max.   :1.000   Max.   :1.000
##       AGE        OTHER_INSTALL           RENT          OWN_RES
##  Min.   :19.00   Length:1000        Min.   :0.000   Min.   :0.000
##  1st Qu.:27.00   Class :character   1st Qu.:0.000   1st Qu.:0.000
##  Median :33.00   Mode  :character   Median :0.000   Median :1.000
##  Mean   :35.55                      Mean   :0.179   Mean   :0.713
##  3rd Qu.:42.00                      3rd Qu.:0.000   3rd Qu.:1.000
##  Max.   :75.00                      Max.   :1.000   Max.   :1.000
##   NUM_CREDITS        JOB            NUM_DEPENDENTS   TELEPHONE
##  Min.   :1.000   Length:1000        Min.   :1.000   Length:1000
##  1st Qu.:1.000   Class :character   1st Qu.:1.000   Class :character
##  Median :1.000   Mode  :character   Median :1.000   Mode  :character
##  Mean   :1.407                      Mean   :1.155
##  3rd Qu.:2.000                      3rd Qu.:1.000
##  Max.   :4.000                      Max.   :2.000
##    FOREIGN            RESPONSE
##  Length:1000        Length:1000
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

One more observation was noted while going through the GermanCreditVariablesDefinition.pdf. The binary variables 'NEW_CAR', 'USED_CAR', 'FURNITURE', 'RADIO/TV', 'EDUCATION', 'RETRAINING' are all described under 'Purpose of credit'. That means, for these assets the loan can be granted. Therefore, we merged ‘NEW_CAR’, ‘USED_CAR’, ‘FURNITURE’, ‘RADIO/TV’, ‘EDUCATION’, ‘RETRAINING’ into into one header ‘PURPOSE’. For the values that does not belong to any of the above, we specified it as OTHERS. This was done using the unite(). unite() is a convenience function to paste together multiple columns into one. The parameters used within it are defined as: The ‘NEW_CAR:RETRAINING, remove = T,sep = ""’ parameter within the unite() will merge all the columns between NEW_CAR and RETRAINING to PURPOSE and then delete them from the data set. Further, ifelse statements are used with mutate() for data conversion.

```
#PURPOSE

data_new<-unite(data_new, "PURPOSE",NEW_CAR:RETRAINING, remove = T,sep = "")
data_new<-data_new%>%
  mutate(PURPOSE = ifelse(PURPOSE == "000100","RADIO/TV",PURPOSE), PURPOSE = ifelse(PURPOSE == "000010"
```

Similarly, the data for MALE_DIV, MALE_SINGLE, MALE_MAR_WID was merged into STATUS

```
#STATUS

data_new<-unite(data_new, "STATUS", MALE_DIV:MALE_MAR_or_WID, remove = T, sep = "")
data_new<-data_new%>%
  mutate(STATUS = ifelse(STATUS == "000", "OTHER(FEMALE)", STATUS), STATUS = ifelse(STATUS == "100", "M
```

Similarly, the data for CO-APPLICANT and GUARANTOR was merged into OTHER_PARTIES

```
#OTHER PARTIES

data_new<-unite(data_new, "OTHER_PARTIES", 'CO-APPLICANT':GUARANTOR, remove = T, sep = "")
data_new<-data_new%>%
  mutate(`OTHER_PARTIES` = ifelse(`OTHER_PARTIES` == "00", "NONE", `OTHER_PARTIES`), `OTHER_PARTIES` = i
```

The data for REAL_ESTATE and PROP_UNKN_NONE was merged into ASSETS

```
#ASSETS

data_new<-unite(data_new, "ASSETS", REAL_ESTATE:PROP_UNKN_NONE, remove = T, sep = "")
data_new<-data_new%>%
  mutate(ASSETS = ifelse(ASSETS == "00", "NONE", ASSETS), ASSETS = ifelse(ASSETS == "01", "NO PROPERTY"
```

The data for RENT and OWN_RES was merged into HOUSING

```
#HOUSING

data_new <- unite(data_new, "HOUSING", RENT:OWN_RES, remove = T, sep = "")
data_new<-data_new%>%
  mutate(HOUSING = ifelse(HOUSING == "00","NONE",HOUSING), HOUSING = ifelse(HOUSING == "01","OWN RESIDE
```

To see how the modified data looks after as compared to the original data set, we executed the summary() on the old as well as the modified dataset.

```
summary(df)
```

```
##       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
```

```
summary(data_new)
```

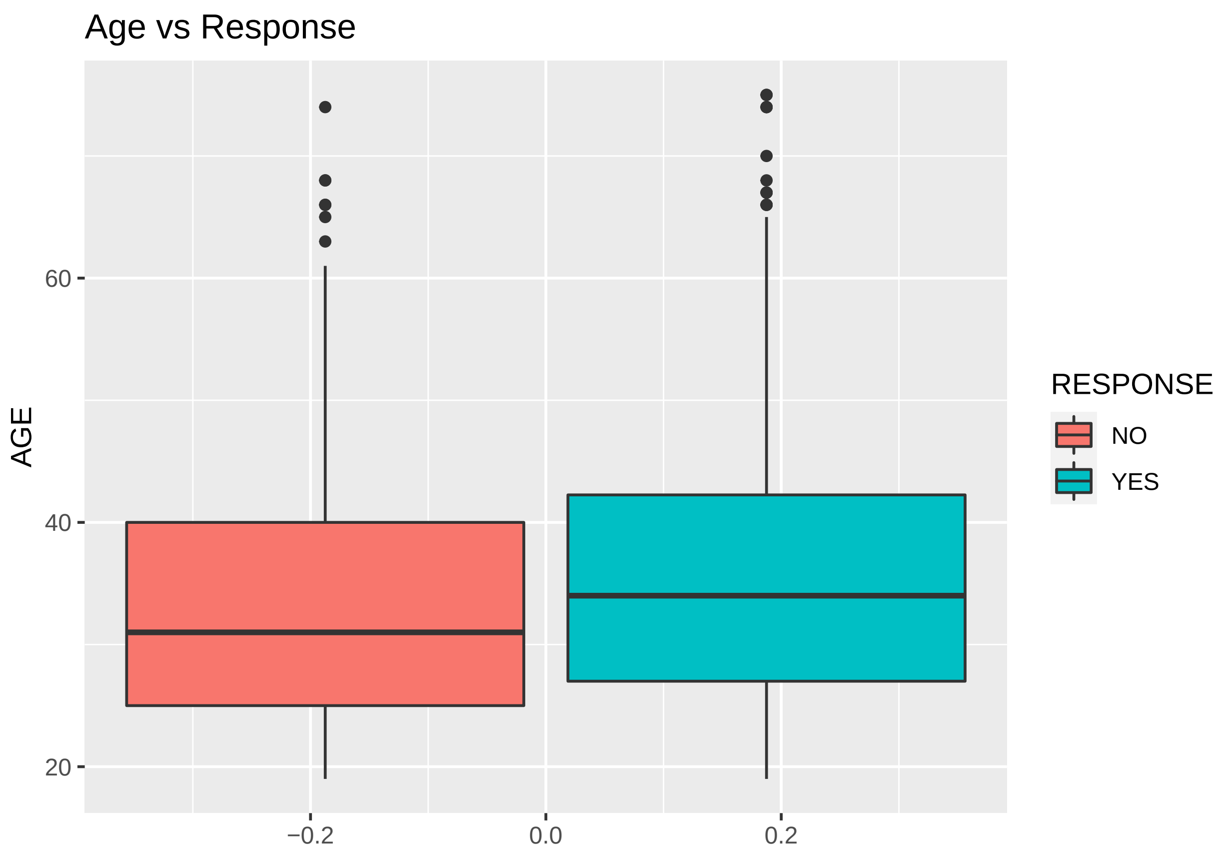
```
##       OBS#          CHK_ACCT            DURATION      HISTORY
## Min.   :   1.0   Length:1000        Min.   : 4.0   Length:1000
## 1st Qu.: 250.8   Class :character   1st Qu.:12.0   Class :character
## Median : 500.5   Mode  :character   Median :18.0   Mode  :character
## Mean   : 500.5                      Mean   :20.9
## 3rd Qu.: 750.2                      3rd Qu.:24.0
## Max.   :1000.0                      Max.   :72.0
##   PURPOSE              AMOUNT       SAV_ACCT          EMPLOYMENT
## Length:1000        Min.   :  250   Length:1000        Length:1000
## Class :character   1st Qu.: 1366   Class :character   Class :character
## Mode  :character   Median : 2320   Mode  :character   Mode  :character
##                    Mean   : 3271
##                    3rd Qu.: 3972
##                    Max.   :18424
##  INSTALL_RATE      STATUS           OTHER_PARTIES      PRESENT_RESIDENT
## Min.   :1.000   Length:1000        Length:1000        Length:1000
## 1st Qu.:2.000   Class :character   Class :character   Class :character
## Median :3.000   Mode  :character   Mode  :character   Mode  :character
## Mean   :2.973
## 3rd Qu.:4.000
## Max.   :4.000
##    ASSETS              AGE        OTHER_INSTALL        HOUSING
## Length:1000        Min.   :19.00   Length:1000        Length:1000
## Class :character   1st Qu.:27.00   Class :character   Class :character
## Mode  :character   Median :33.00   Mode  :character   Mode  :character
##                    Mean   :35.55
##                    3rd Qu.:42.00
##                    Max.   :75.00
##  NUM_CREDITS        JOB            NUM_DEPENDENTS   TELEPHONE
## Min.   :1.000   Length:1000        Min.   :1.000   Length:1000
## 1st Qu.:1.000   Class :character   1st Qu.:1.000   Class :character
## Median :1.000   Mode  :character   Median :1.000   Mode  :character
## Mean   :1.407                      Mean   :1.155
## 3rd Qu.:2.000                      3rd Qu.:1.000
## Max.   :4.000                      Max.   :2.000
##   FOREIGN            RESPONSE
## Length:1000        Length:1000
## Class :character   Class :character
## Mode  :character   Mode  :character
##
##
##
```

Data Exploration

After the data is cleaned and modified, we moved on to explore the data. Initially, we started exploring individual variables against the target variable and then moved on to multivariate relationships.

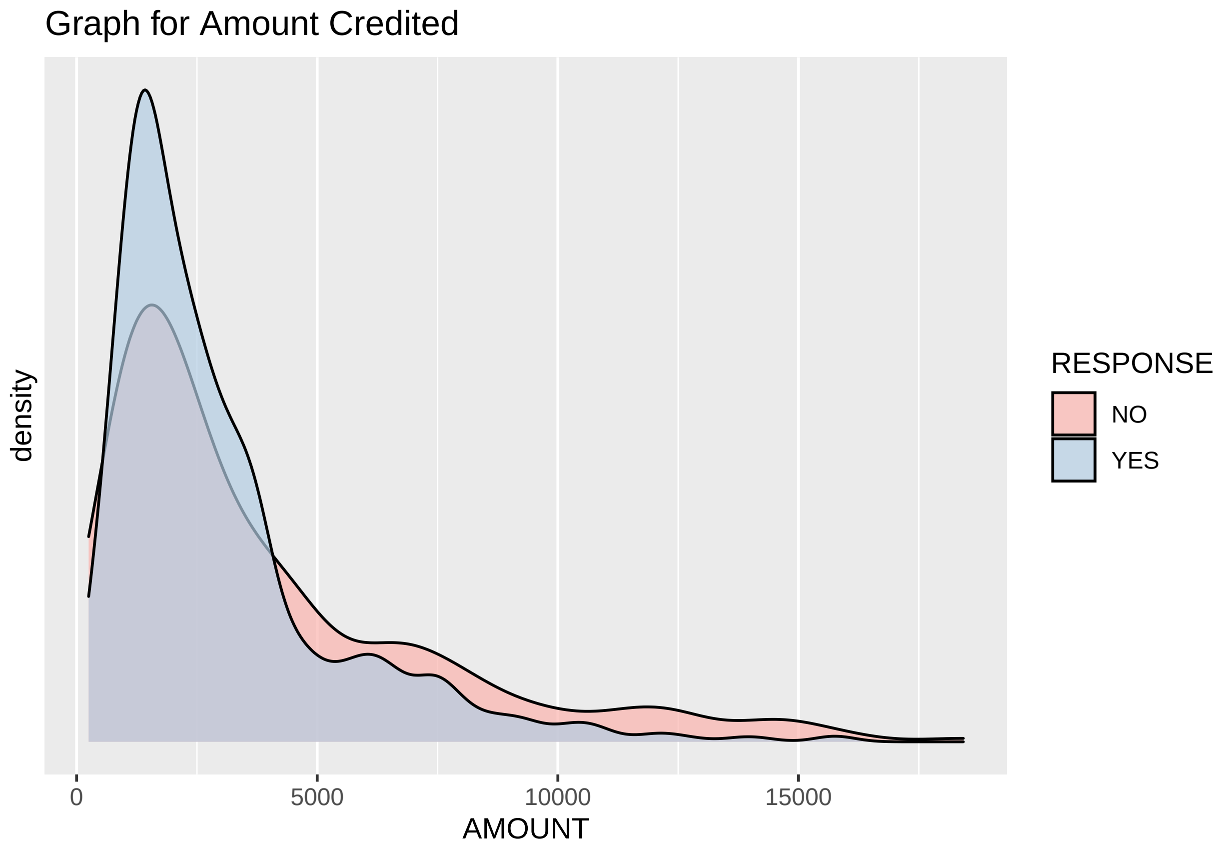
We started by exploring the data for Age vs Response. From the below boxplot, we can see that from the age variable, the median value for good records is better than that of bad records. Most of individuals who would want to avail a credit facility are aged between 20 and 40.From this we can conclude that people in this age are willing to take calculated risk and accounts for good credit.People above the age of 60 can be considered as outliers.

```
#AGE
data_new %>%
  ggplot()+
  geom_boxplot(aes(y = AGE, fill = RESPONSE))+
  labs(title = "Age vs Response")
```



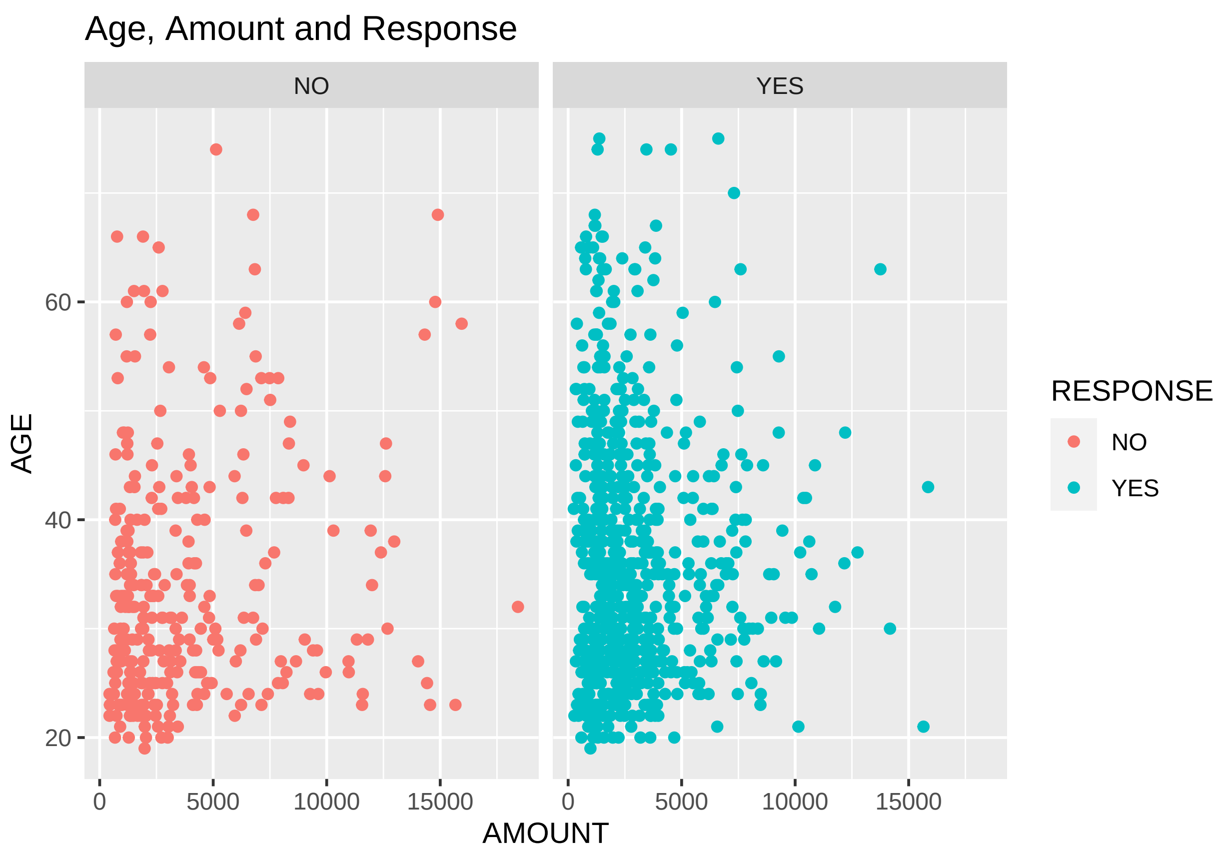
The density graph is used to check the continuous variable AMOUNT with the target variable RESPONSE. From the graph we observed that people take most loans between 1000 to 3000. Also, as the loan amount increases after 5000 the chances of considering a good risk reduces.

```
#AMOUNT
ggplot(data = data_new)+
  geom_density(aes(x = AMOUNT, fill = RESPONSE),alpha = 0.7)+
  scale_fill_brewer(palette = "Pastel1")+
  scale_y_continuous(breaks = 1)+
  labs(title = "Graph for Amount Credited")
```



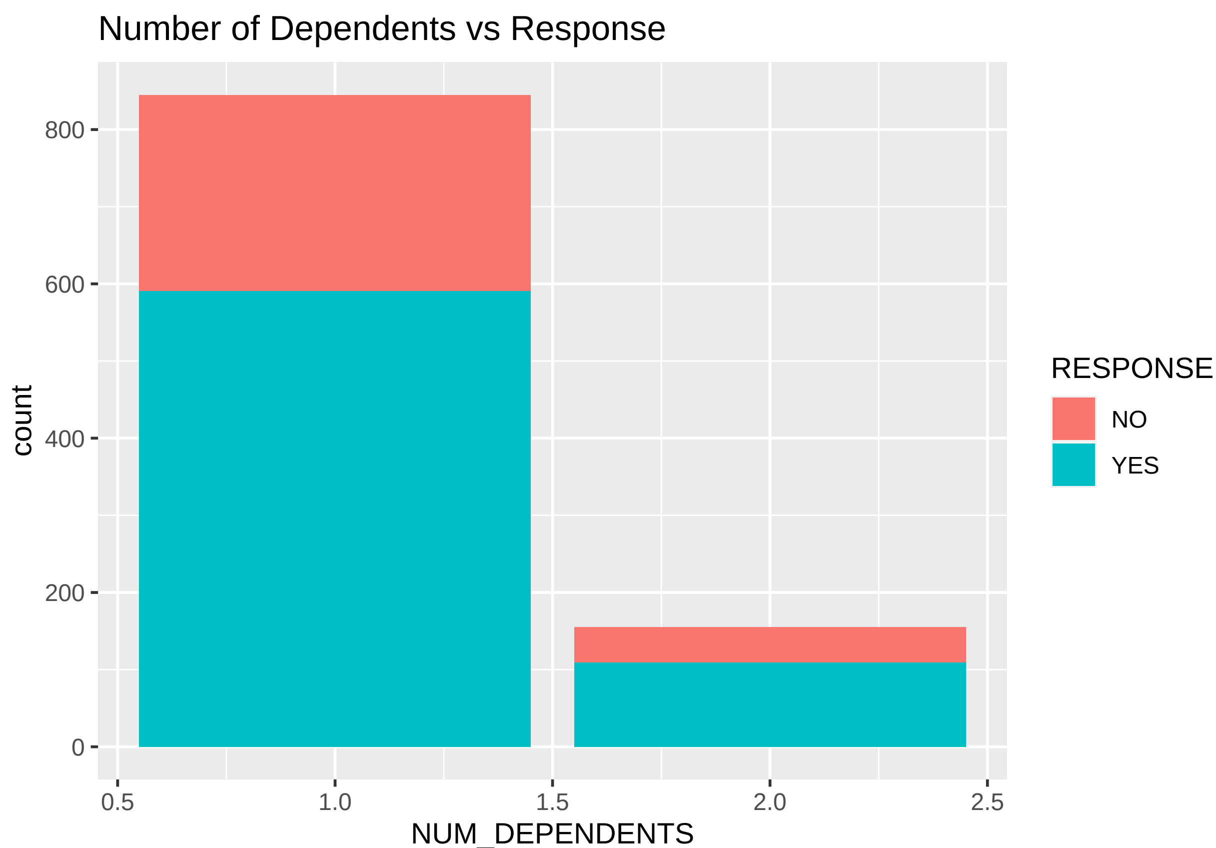
By comparing the data between AGE and AMOUNT, it can be observed that most loan with good credits are taken between the amount of 0 and 5000, regardless of the age, while youngsters with large credit amount are more likely to be treated as bad credit risk. The division of the graphs has been done using facet_grid() on the target variable. This allows to split the graph based on the target variable RESPONSE. (This has been used in multiple graphs below)

```
#AGE vs AMOUNT
ggplot(data = data_new)+
  geom_point(aes(x = AMOUNT, y = AGE, col = RESPONSE))+
  facet_grid(~RESPONSE)+
  labs(title = "Age, Amount and Response")
```



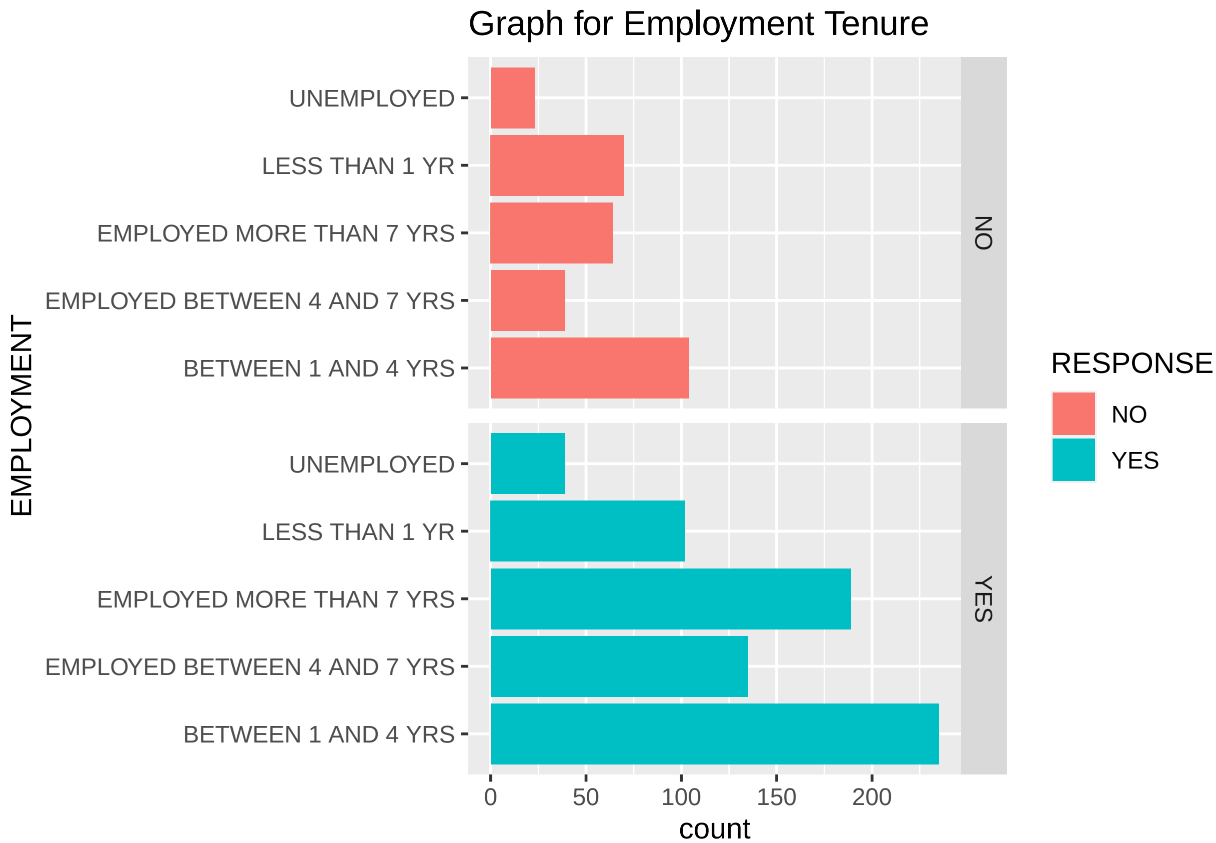
From the below graph it can be analyzed that people who have one dependent are more likely to borrow credit.

```
#NUM_DEPENDENTS
ggplot(data_new, aes(x = NUM_DEPENDENTS, fill = RESPONSE)) +
  geom_bar(width = 0.9) +
  labs(title = "Number of Dependents vs Response")
```



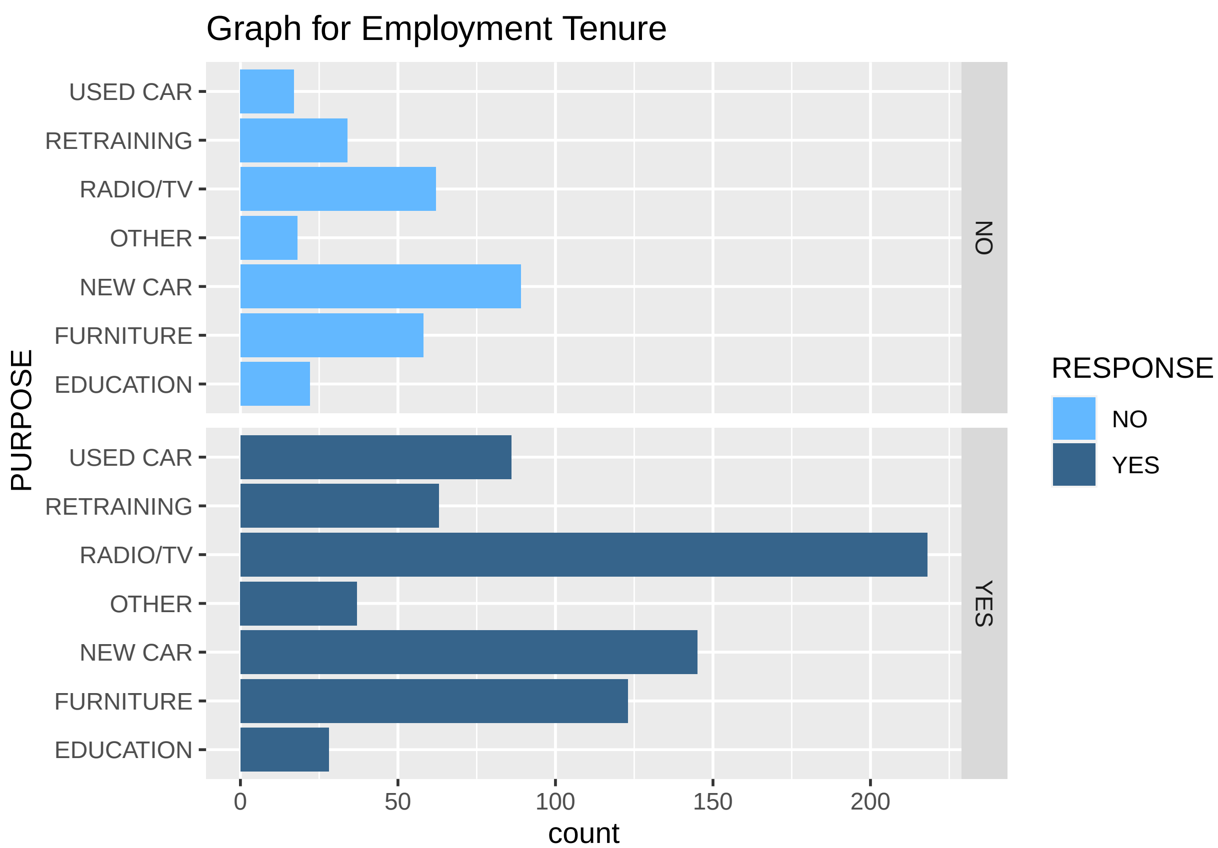
The below graph has been plotted to check the credit risk for each Employment category against RESPONSE.It can be seen that good risk individuals are the ones having high employment tenure between 1 and 4 years as compare to the bad risk individuals who are having low employment tenure. That is, as the employment tenure increases the chances of being a good credit risk also increases.

```
#EMPLOYMENT
ggplot(data_new, aes(y = EMPLOYMENT, fill = RESPONSE)) +
  geom_bar() + facet_grid(RESPONSE~.)+
  labs(title = "Graph for Employment Tenure")
```



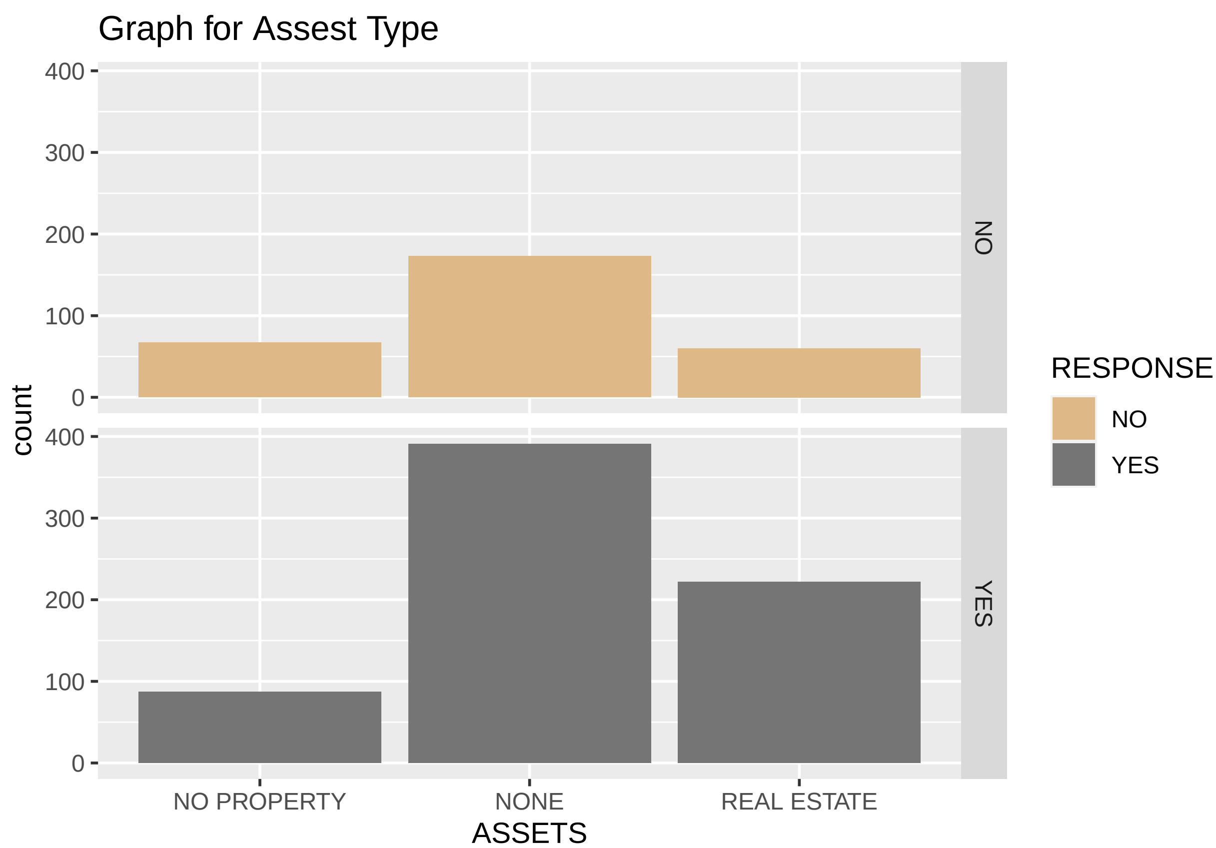
PURPOSE has been divided into 7 categories. From the bar chart we can conclude that people usually take loan for Radio/TV, New Car, Furniture as they have the max count.

```
#PURPOSE
ggplot(data_new) +
  geom_bar(aes(y = PURPOSE, fill = RESPONSE)) + facet_grid(RESPONSE~.)+
  labs(title = "Graph for Employment Tenure")+
  scale_fill_manual(values = c("steelblue1", "steelblue4"))
```



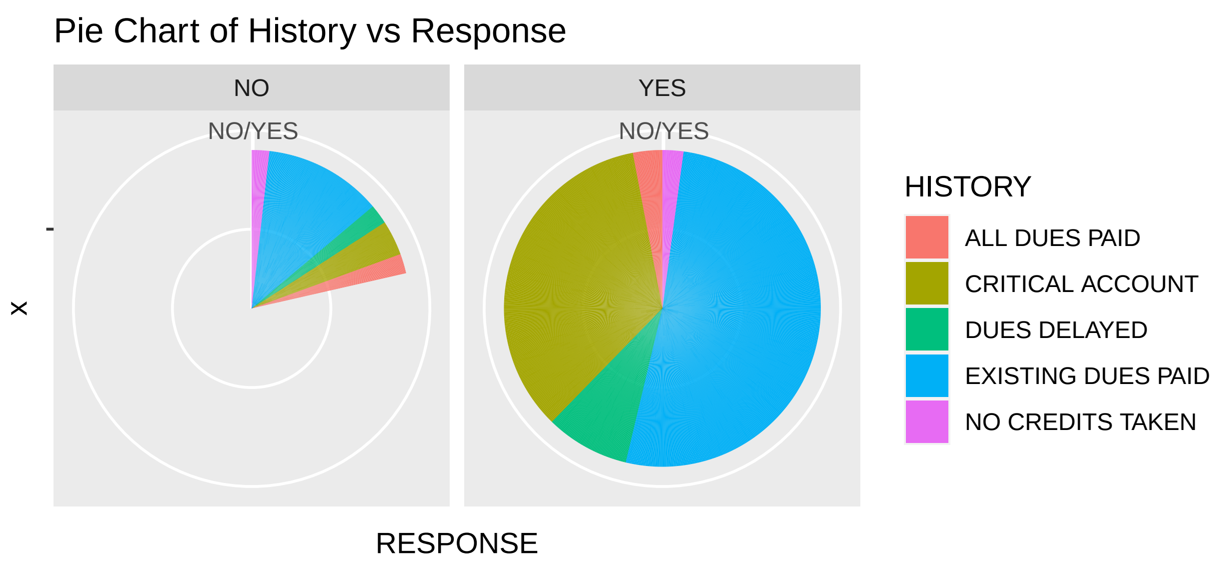
From the bar graph, people who have no property are less likely to be considered as a good credit risk.

```
#ASSETS
ggplot(data_new) +
  geom_bar(aes(x = ASSETS, fill = RESPONSE)) + facet_grid(RESPONSE~.)+
  labs(title = "Graph for Assest Type")+
  scale_fill_manual(values = c("burlywood", "gray46"))
```



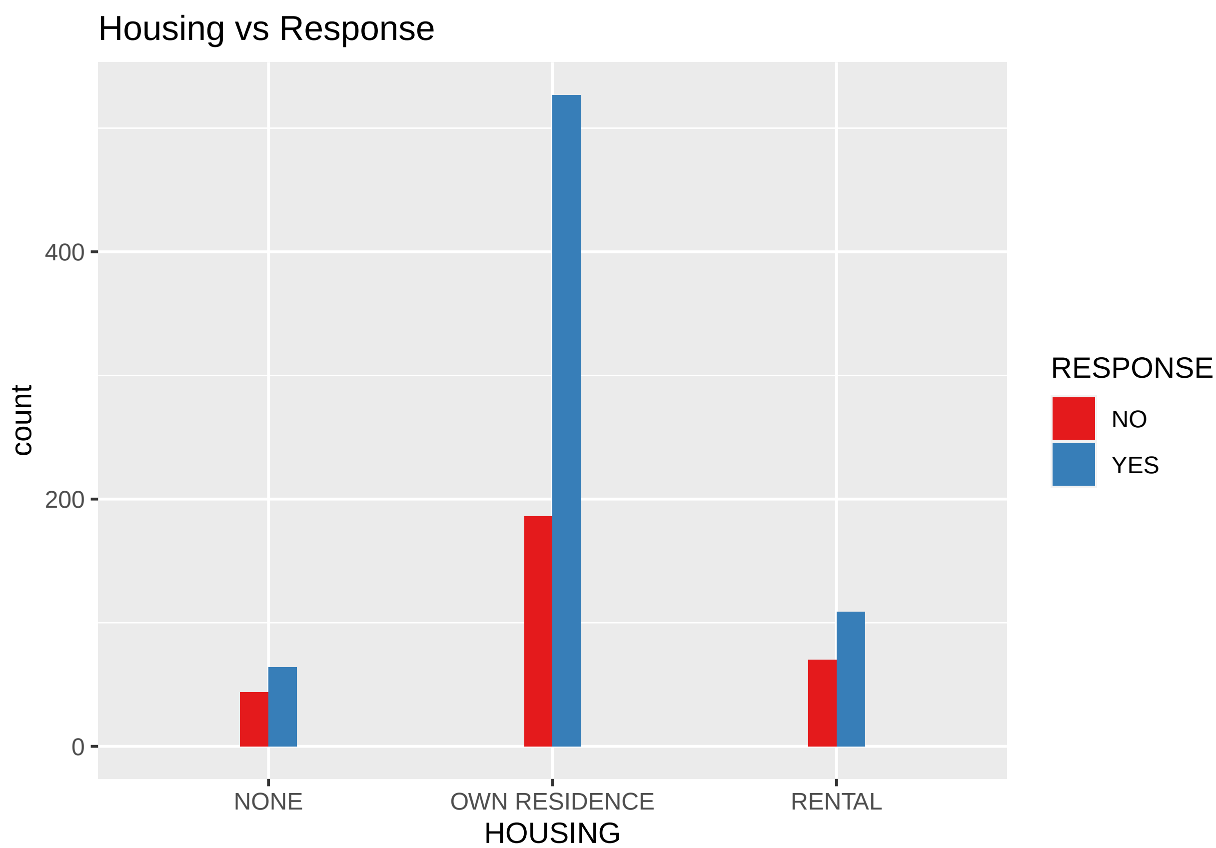
Below pie chart indicates that, individuals who have paid their existing dues or have a critical account are more likely to borrow credit and be considered as good credit risk. We can also see that people who have all dues paid or no credits taken are more likely to be considered as bad credit and less likely to be getting a loan. The pie chart has been obtained using the coord_polar()

```
#HISTORY
ggplot(data = data_new)+
  geom_bar(aes(x="",y = RESPONSE, fill = HISTORY),width = 1, stat = "identity", position = "stack")+
  coord_polar("y")+
  facet_grid(~RESPONSE)+
  labs(title = "Pie Chart of History vs Response")
```



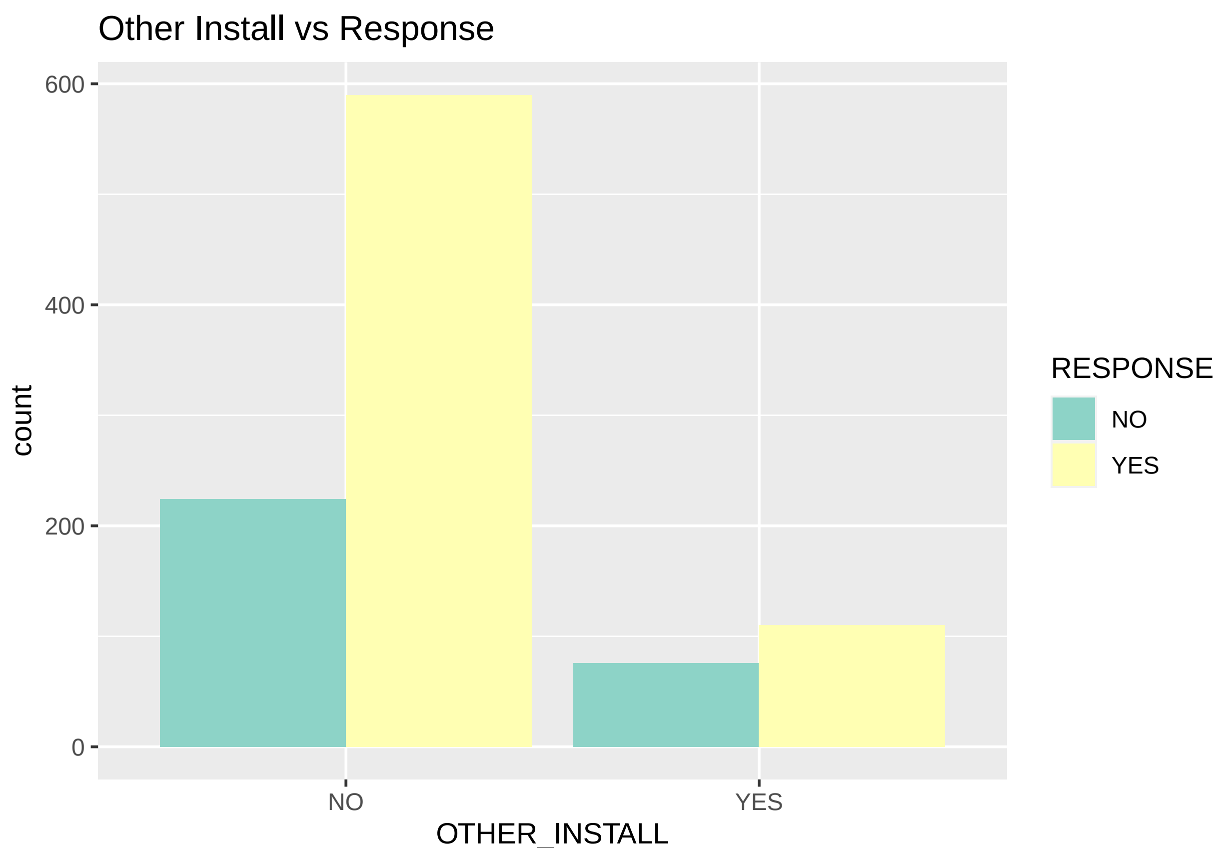
From the below graph, majority of the individuals have their own house and hence can be considered to get a loan. Further, people who lives on rent or do not have their own house are less likely to be considered for the loan. Also, individuals who have their own house are considered having a good credit risk.

```
#HOUSING
ggplot(data_new) +
  geom_bar(aes(x= HOUSING, fill=RESPONSE), width = 0.20, position = "dodge") +
  scale_fill_brewer(palette = "Set1")+
  labs(title = "Housing vs Response")
```



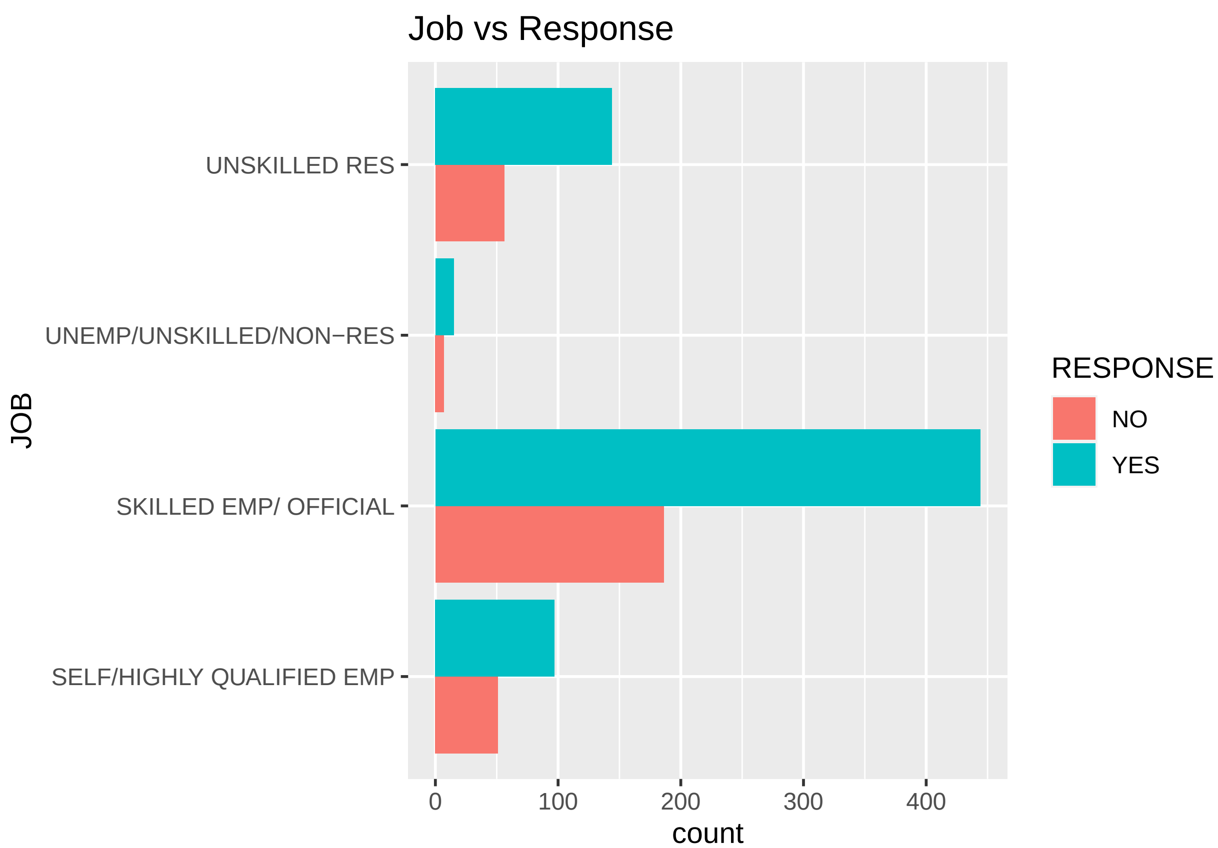
From the below graph, we noted that individuals who do not have any other loans that they are currently paying are more likely to be considered as good credit risk.

```
#OTHER_INSTALL
ggplot(data = data_new)+
  geom_bar(aes(x = OTHER_INSTALL, fill = RESPONSE), position = "dodge")+
  scale_fill_brewer(palette = "Set3")+
  labs(title = "Other Install vs Response")
```



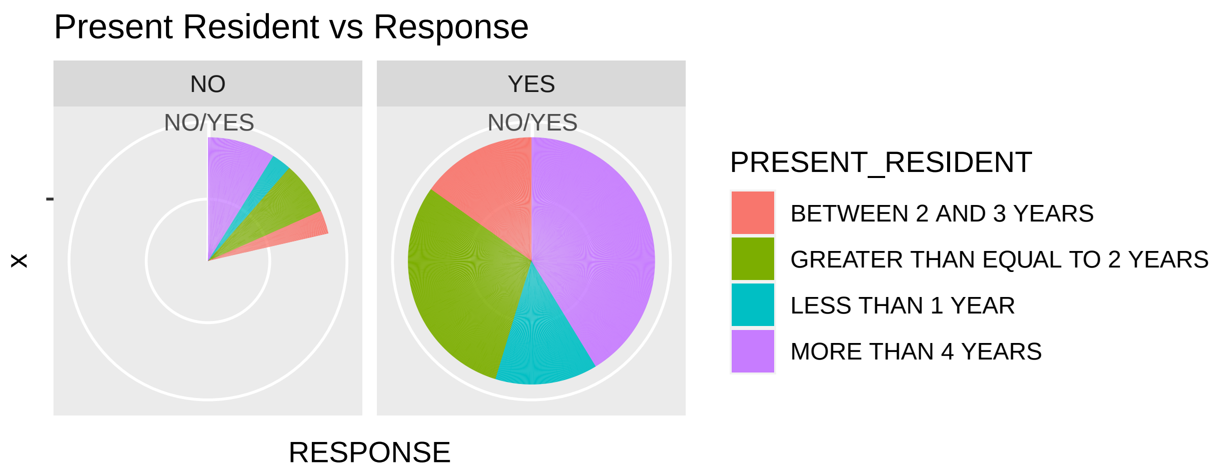
From the below graph we can observe that most loans are taken by people who belong to the skilled employee/official category. Also, if unemployed, possibility of getting credit facility remains minimal.

```
#JOB
ggplot(data = data_new)+
  geom_bar(aes(y = JOB, fill = RESPONSE), position = "dodge")+
  labs(title = "Job vs Response")
```



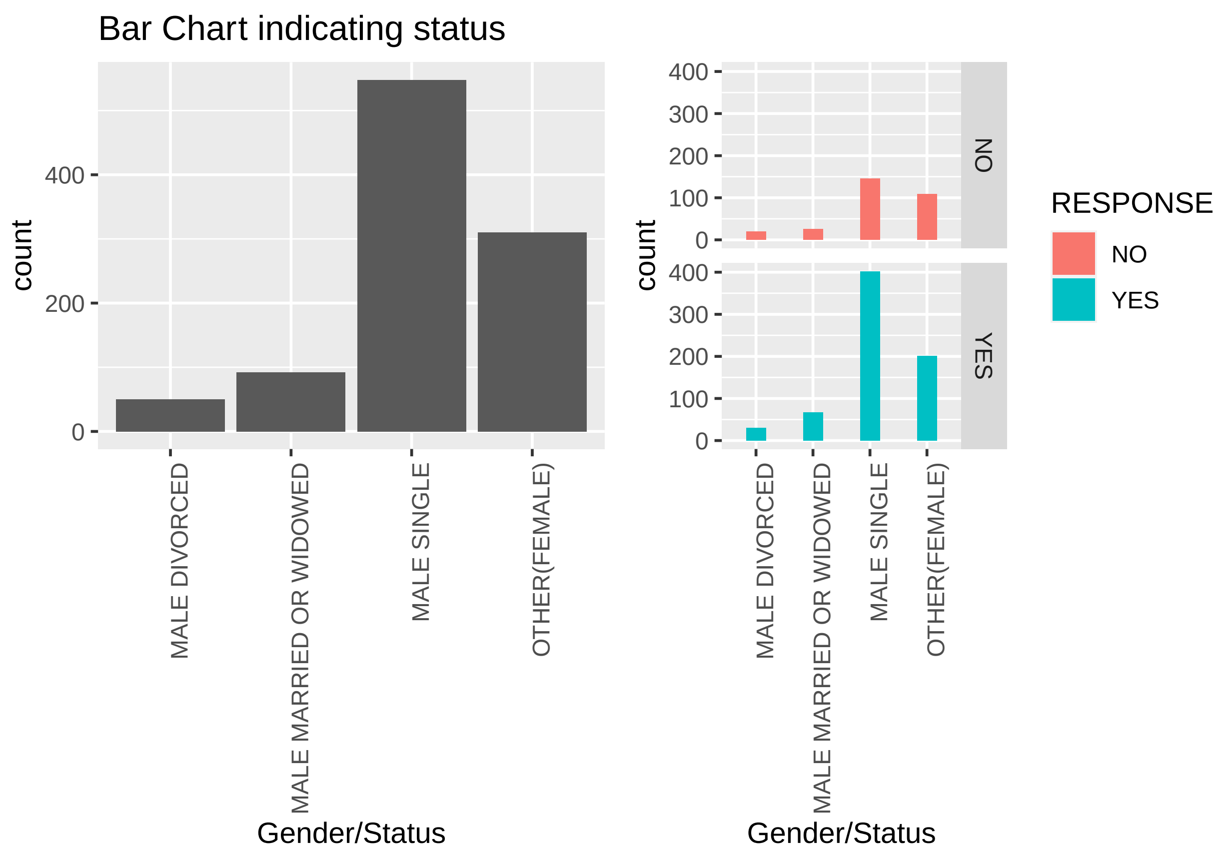
Individuals who are residing in the same house since few years are considered to be more reliable and can be granted more credit facility. Also, it is evident that individuals who are residing in the same location for a long period of time can be considered as good credit risk.

```
#PRESENT_RESIDENT
ggplot(data = data_new)+
  geom_bar(aes(x="",y = RESPONSE, fill = PRESENT_RESIDENT),width = 1, stat = "identity", position = "sta
  coord_polar("y")+
  facet_grid(~RESPONSE)+
  labs(title = "Present Resident vs Response")
```



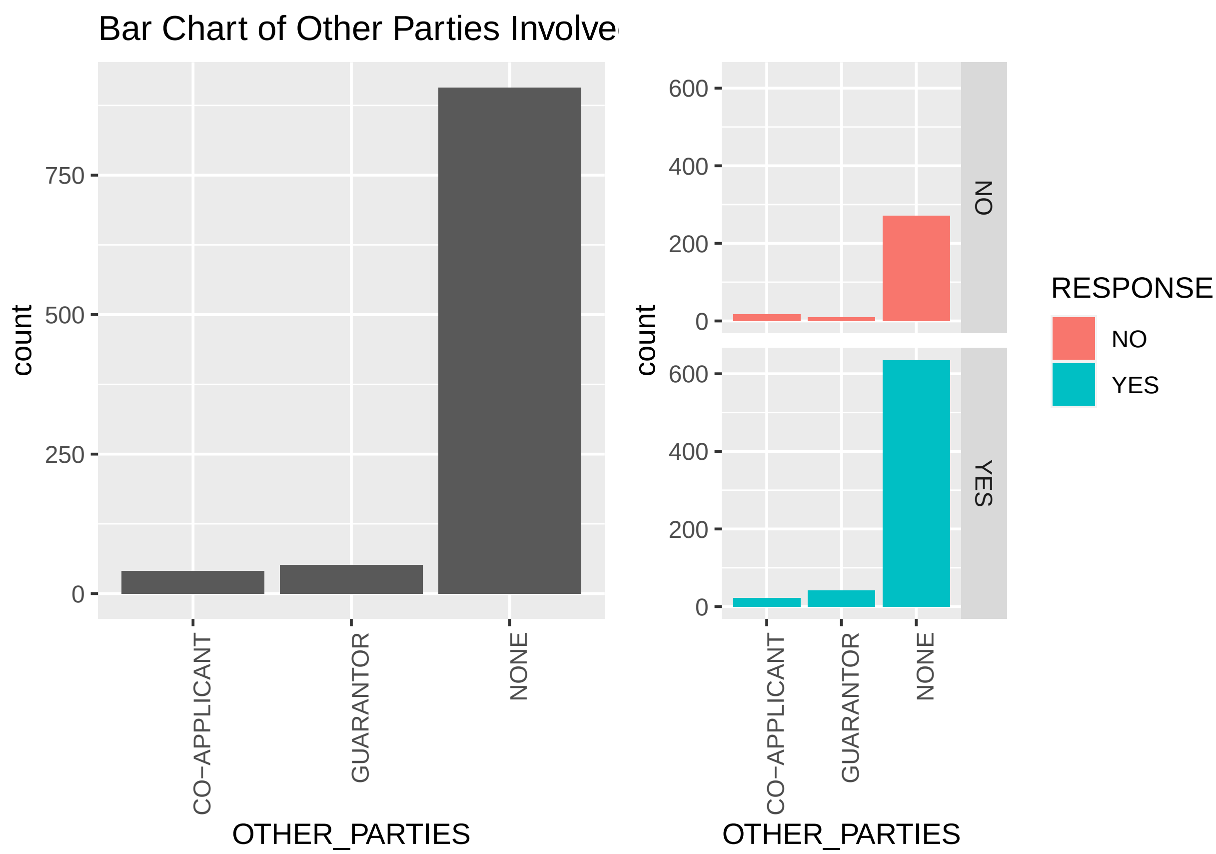
It can be seen that majority of the credit facility is availed by the male singles. Also it can be observed that male singles are a good credit risk. Also it can be noted that the bad credit risk is proportionately more for females.

```
#STATUS
p1 <- ggplot(data_new, aes(x = STATUS)) + geom_bar() +
  labs(title = 'Bar Chart indicating status', x = "Gender/Status") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

p2 <- ggplot(data_new, aes(x = STATUS, fill = RESPONSE)) +
  geom_bar(width=0.35) + facet_grid(RESPONSE~.) +
  labs(title = '', x = "Gender/Status") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

plot_grid(p1, p2, ncol = 2)
```



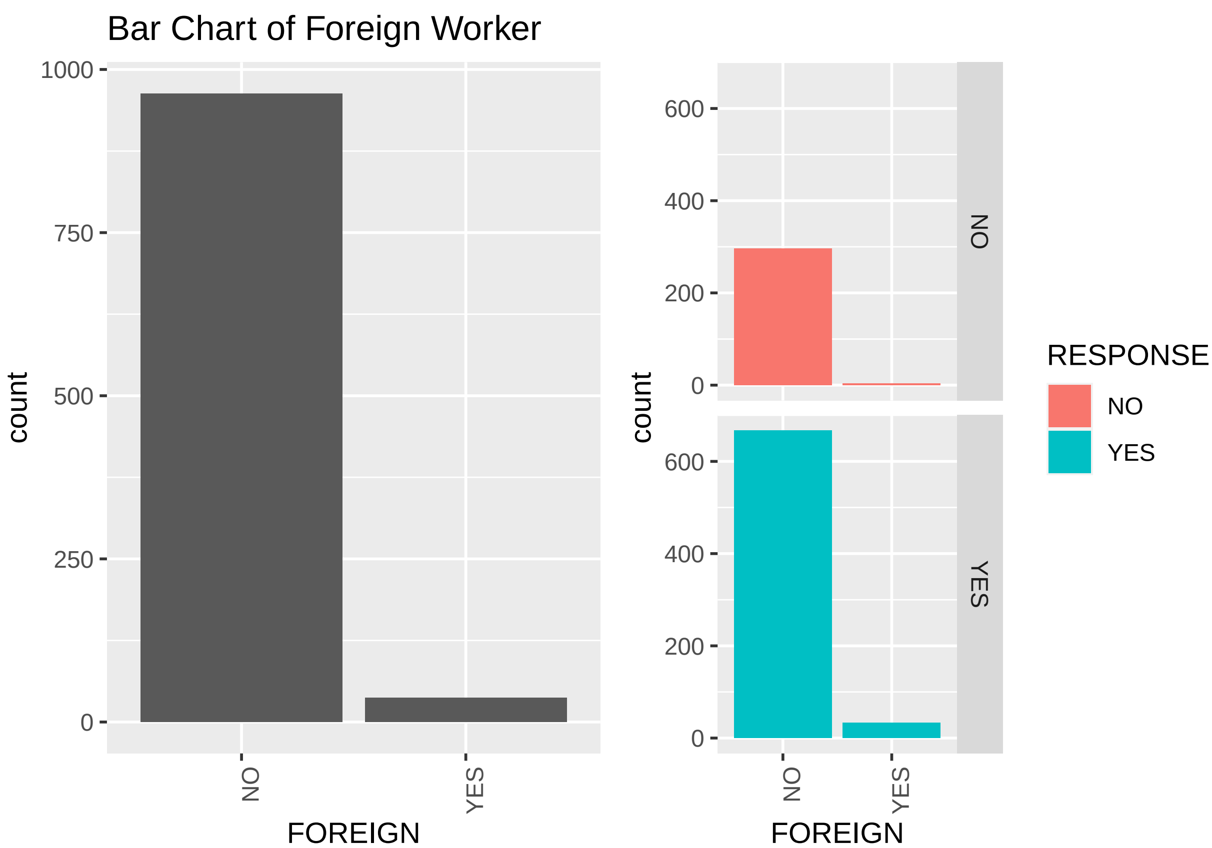
Here it can be observed that None stood out the most here as most of the inidividuals applied for credit alone.

```
#OTHER_PARTIES
p3 <- ggplot(data_new, aes(x = OTHER_PARTIES)) + geom_bar() +
  labs(title = 'Bar Chart of Other Parties Involved') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

p4 <- ggplot(data_new, aes(x = OTHER_PARTIES, fill = RESPONSE)) +
  geom_bar() + facet_grid(RESPONSE~.) +
  labs(title = '') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

plot_grid(p3, p4, ncol = 2)
```



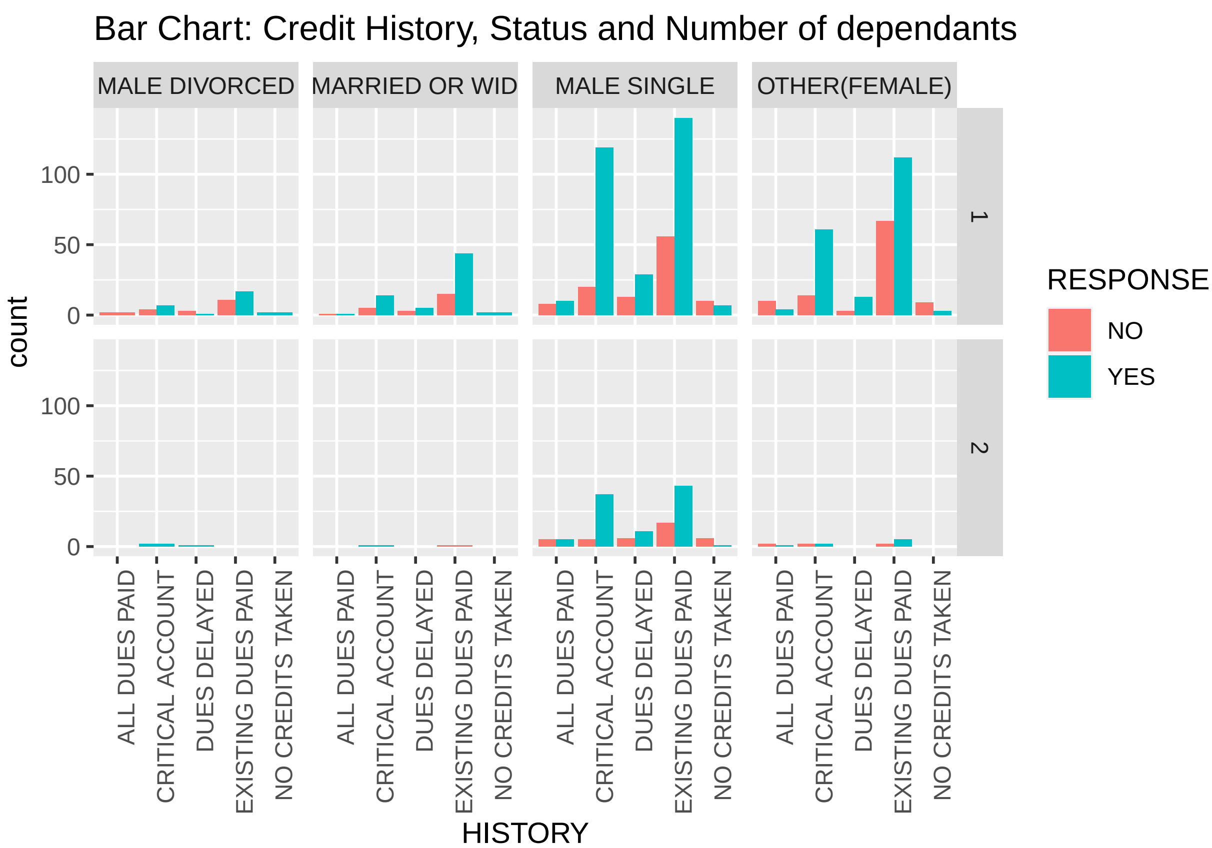
From the below bar graph, we can see that individuals who are not foreigners are more likely to have a good credit risk.

```
#FOREIGN
a1 <- ggplot(data_new, aes(x = FOREIGN)) + geom_bar() +
  labs(title = 'Bar Chart of Foreign Worker') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

a2 <- ggplot(data_new, aes(x = FOREIGN, fill = RESPONSE)) +
  geom_bar() + facet_grid(RESPONSE~.) +
  labs(title = '') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

plot_grid(a1, a2, ncol = 2)
```



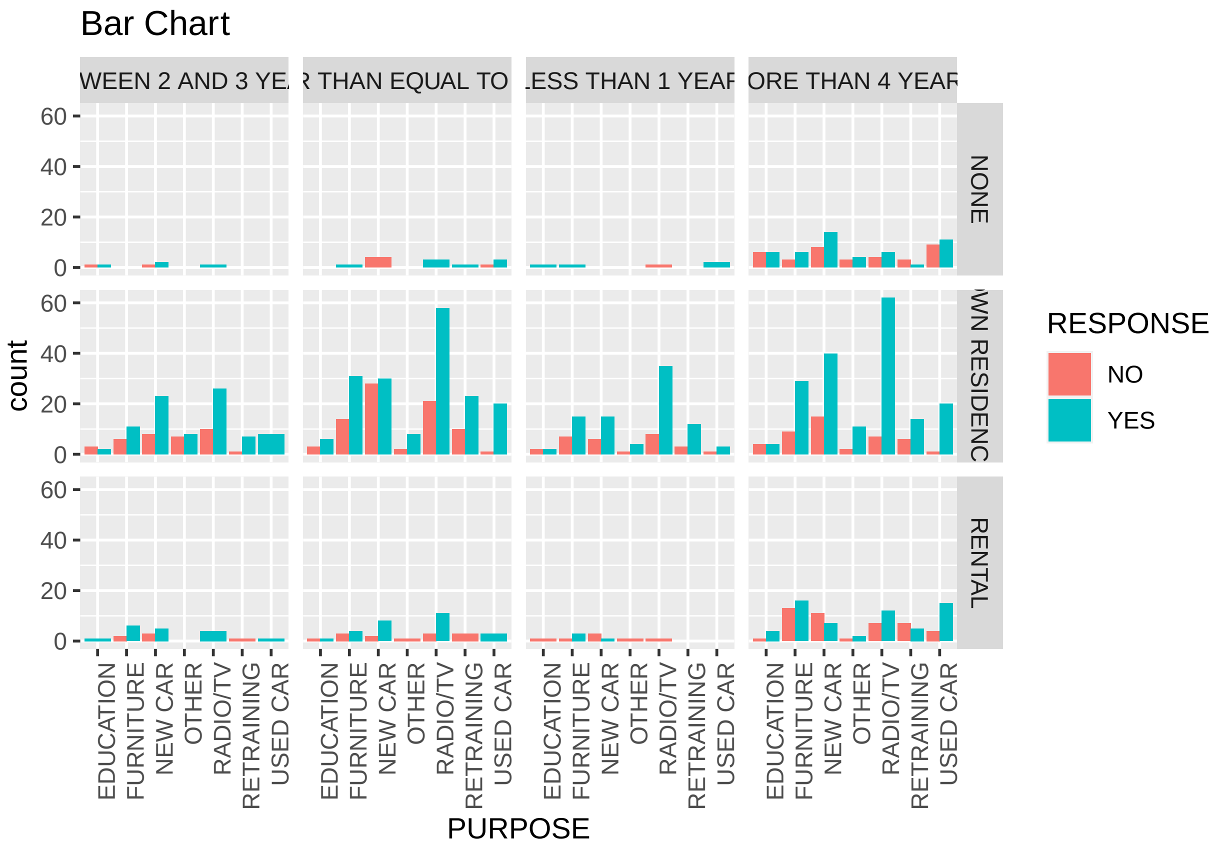
It can be observed that most of the females had only 1 dependants. Whereas, proportion of males having more dependants are slightly more. Also, it can be observed males with more than 1 dependants have either critical account or have existing dues paid.

```
#Multivariate (STATUS, HISTORY, NUM_DEPENDENTS)

ggplot(data_new) +
  geom_bar(aes(x = HISTORY, fill = RESPONSE), position = 'dodge') + facet_grid(factor(NUM_DEPENDENTS)~S
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = 'Bar Chart: Credit History, Status and Number of dependants', x = "HISTORY", fill = "RES
```



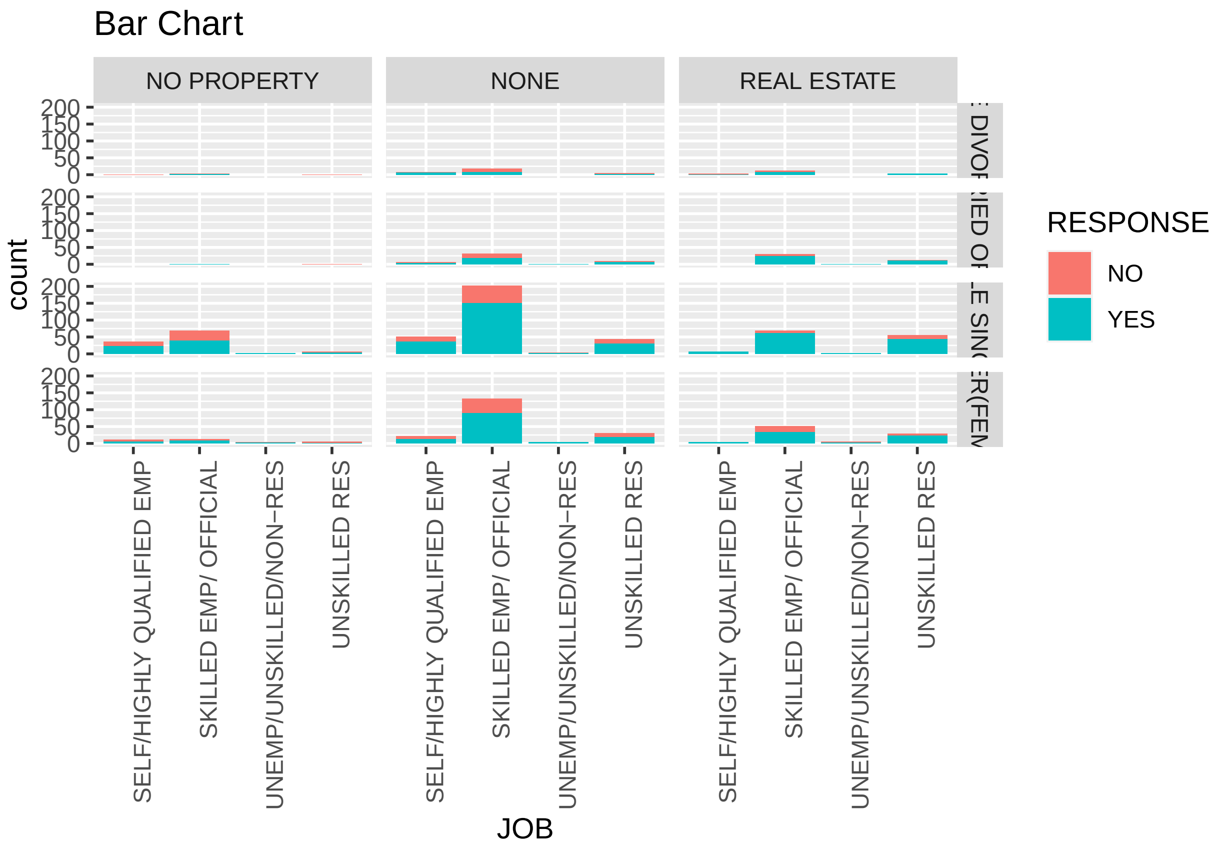
It can be clearly seen that individuals who have their own residence and have stayed for more than 4 years are more likely to be good credit risk candidates. It can also be seen that applicants who do not have their own residence and also do not stay on rent are not likely to avail the credit facility. It can be seen that applicants who like in a rental space are more likely to be considerd a bad credit risk proportionately.

```
#Multivariate (PURPOSE, HOUSING, PRESENT_RESIDENT)

ggplot(data_new) +
  geom_bar(aes(x = PURPOSE, fill = RESPONSE), position = 'dodge') + facet_grid(HOUSING~PRESENT_RESIDENT)
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = 'Bar Chart', x = "PURPOSE", fill = "RESPONSE")
```



It can be observed that Male singles and females who are skilled employed/officials are more likely to avail the credit facility. Self Employed, Un-skilled individuals do not wish to avail the credit facility. Moreover, females who own no property are more likely to be considered as bad credit risk. Candidates who are skilled and have their own real estate are considered to be a good credit risk.

```
#Multivariate (JOB, STATUS, ASSET)

ggplot(data_new) +
  geom_bar(aes(x = JOB, fill = RESPONSE), position = 'stack') + facet_grid(STATUS~ASSETS) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = 'Bar Chart', x = "JOB", fill = "RESPONSE")
```



Since we have converted the numeric variables into categorical ones, we did not plot a correlation matrix on them.

To check which of the input variables does not have relationship with the target variable, we performed the hypothesis testing using Chi-Squared analysis and found that PRESENT_RESIDENT, TELEPHONE, NUM_DEPENDENTS, NUM_CREDITS and INSTALL_RATE does not have relationship with the target variable RESPONSE. For hypothesis testing, we took alpha value to be 0.05. If the p-value of any relation is less than 0.05 we can say that there exists a relation between the input and target variable. For the below listed variables, the p-value is greater than 0.05 and hence cannot be considered as "good" predictors.

```
table(data_new$PRESENT_RESIDENT, data_new$RESPONSE)
```

```
##
##                                   NO YES
##    BETWEEN 2 AND 3 YEARS          43 106
##    GREATER THAN EQUAL TO 2 YEARS  97 211
##    LESS THAN 1 YEAR               36  94
##    MORE THAN 4 YEARS             124 289
```

```
chisq.test(data_new$PRESENT_RESIDENT, data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$TELEPHONE, data_new$RESPONSE)
```

```
##
##         NO YES
##   NO  187 409
##   YES 113 291
```

```r
chisq.test(data_new$TELEPHONE, data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$NUM_DEPENDENTS, data_new$RESPONSE)
```

```
##
##       NO YES
##   1 254 591
##   2  46 109
```

```r
chisq.test(data_new$NUM_DEPENDENTS, data_new$RESPONSE, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$NUM_DEPENDENTS and data_new$RESPONSE
## X-squared = 0.0090893, df = 1, p-value = 0.924
```

```r
table(data_new$NUM_CREDITS, data_new$RESPONSE)
```

```
##
##       NO YES
##   1 200 433
##   2  92 241
##   3   6  22
##   4   2   4
```

```r
chisq.test(data_new$NUM_CREDITS, data_new$RESPONSE, correct = F)
```

```
## Warning in chisq.test(data_new$NUM_CREDITS, data_new$RESPONSE, correct = F):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$NUM_CREDITS and data_new$RESPONSE
## X-squared = 2.6712, df = 3, p-value = 0.4451
```

```r
table(data_new$INSTALL_RATE, data_new$RESPONSE)
```

```
##
##      NO YES
##   1  34 102
##   2  62 169
##   3  45 112
##   4 159 317
```

```r
chisq.test(data_new$INSTALL_RATE, data_new$RESPONSE, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$INSTALL_RATE and data_new$RESPONSE
## X-squared = 5.4768, df = 3, p-value = 0.14
```

Also, we performed the Chi-Squared testing on the remaining variables to check their relation with the target variable. All the below listed variables have a p-value less than 0.05 and hence can be considered as "good" predictors.

```r
# Useful
table(data_new$HISTORY, data_new$RESPONSE)
```

```
##
##                         NO YES
##   ALL DUES PAID         28  21
##   CRITICAL ACCOUNT      50 243
##   DUES DELAYED          28  60
##   EXISTING DUES PAID 169 361
##   NO CREDITS TAKEN      25  15
```

```r
chisq.test(data_new$HISTORY, data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$EMPLOYMENT, data_new$RESPONSE)
```

```
##
##                                   NO YES
##   BETWEEN 1 AND 4 YRS            104 235
##   EMPLOYED BETWEEN 4 AND 7 YRS    39 135
##   EMPLOYED MORE THAN 7 YRS        64 189
##   LESS THAN 1 YR                  70 102
##   UNEMPLOYED                      23  39
```

```r
chisq.test(data_new$EMPLOYMENT, data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$STATUS, data_new$RESPONSE)
```

```
##
##                              NO YES
##   MALE DIVORCED              20  30
##   MALE MARRIED OR WIDOWED    25  67
##   MALE SINGLE               146 402
##   OTHER(FEMALE)             109 201
```

```r
chisq.test(data_new$STATUS, data_new$RESPONSE, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$STATUS and data_new$RESPONSE
## X-squared = 9.6052, df = 3, p-value = 0.02224
```

```r
table(data_new$OTHER_INSTALL, data_new$RESPONSE)
```

```
##
##          NO YES
##   NO    224 590
##   YES    76 110
```

```r
chisq.test(data_new$OTHER_INSTALL, data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$FOREIGN, data_new$RESPONSE)
```

```
##
##          NO YES
##   NO    296 667
##   YES     4  33
```

```r
chisq.test(data_new$FOREIGN, data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$HOUSING, data_new$RESPONSE)
```

```
##
##                  NO YES
##   NONE           44  64
##   OWN RESIDENCE 186 527
##   RENTAL         70 109
```

```r
chisq.test(data_new$HOUSING, data_new$RESPONSE, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$HOUSING and data_new$RESPONSE
## X-squared = 18.2, df = 2, p-value = 0.0001117
```

```r
table(data_new$PURPOSE, data_new$RESPONSE)
```

```
##
##               NO YES
##   EDUCATION   22  28
##   FURNITURE   58 123
##   NEW CAR     89 145
##   OTHER       18  37
##   RADIO/TV    62 218
##   RETRAINING  34  63
##   USED CAR    17  86
```

```r
chisq.test(data_new$PURPOSE, data_new$RESPONSE, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$PURPOSE and data_new$RESPONSE
## X-squared = 30.757, df = 6, p-value = 2.821e-05
```

```r
table(data_new$CHK_ACCT,data_new$RESPONSE)
```

```
##
##                               NO YES
##   BETWEEN 0 AND 200          105 164
##   GREATER THAN EQUAL TO 200   14  49
##   LESS THAN 0                135 139
##   NO CHECKING ACCOUNT         46 348
```

```r
chisq.test(data_new$CHK_ACCT,data_new$RESPONSE, correct = F)
```

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

```r
table(data_new$ASSETS, data_new$RESPONSE)
```

```
##
##                NO YES
##   NO PROPERTY  67  87
##   NONE        173 391
##   REAL ESTATE  60 222
```

```r
chisq.test(data_new$ASSETS, data_new$RESPONSE, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  data_new$ASSETS and data_new$RESPONSE
## X-squared = 23.719, df = 2, p-value = 7.072e-06
```

```r
table(data_new$SAV_ACCT, data_new$RESPONSE)
```

```
##
##                               NO YES
##   BETWEEN 100 and 500         34  69
##   BETWEEN 500 AND 1000        11  52
##   GREATER THAN EQUAL TO 1000   6  42
##   LESS THAN 100              217 386
##   NO SAVINGS                  32 151
```

```r
chisq.test(data_new$SAV_ACCT, data_new$RESPONSE, correct = F)
```

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

From the data exploration phase, we found that the numerical features were quite tidy, and hence no cleaning needed to be undertaken. There were no missing values in the data set. But, in case of the categorical variables,we had to modify data to bind few categorical variables into one. We did not remove any of the original variables in the dataset, as we wanted to explore the data in detail and play with the fine granularity at the initial phase of model building. From the graphs plotted and hypothesis testing done in the EDA phase, we can say that History, Purpose, Employment, Foreign, Age, Housing, Status, Assets, SAV_ACCT, CHK_ACCT, OTHER_INSTALL can possibly be useful variables in determining "good" or "bad" cases.

===

Question b:

Splitting the data into training and test sets:

After the data set was cleaned, we have split our data into two data sets: Training and Test. We trained our models on the training data and using different combinations of model parameters, we have evaluated the model on our testing data. We have used the following split criteria on our models:

Training - 50%, Test - 50% Training - 70%, Test - 30% Training - 80%, Test - 20% Training - 75% Test - 25% We've built our model using rpart and C5.0. We have removed the `OBS#` variable while developing our models. This was done because OBS shows just the row number of each observation present and does not have any relation with the target variable RESPONSE. Out of the different parameters that rpart takes as arguments, we used some of them like minsplit, cp, minbucket. We have first developed our tree without using any parameters. After that, we have performed pre-pruning on the model. C5.0 uses information entropy computation to determine the best rule that splits the data, while rpart uses 'gini' co-efficient.

Model 1: Splitting the data as 50% Training and 50% Test

Initially, we our building our model without pre-pruning the data to find out the model performance.

```
#Gini for 0.5 and 0.5 (Default Tree without pruning)

data_new$RESPONSE<- as.factor(data_new$RESPONSE)
#The random seed is set to a fixed value below to make the results reproducible.
set.seed(125)
#Splitting criteria
ind<-sample(2, nrow(data_new), replace =T, prob = c(0.5, 0.5))
train<-data_new[ind==1,]
test<-data_new[ind==2,]

#myFormula specifies that the target RESPONSE is dependent variable while all others (used as ~.) are i
myFormula = RESPONSE ~. -`OBS#`
# Default decision tree without using pruning parameters
mytree <- rpart(myFormula, data = train)

#Check Train error
pred_train<-predict(mytree,data=train,type="class")
mean(train$RESPONSE!=pred_train)
```

```
## [1] 0.1484536
```

```
#Check Test error
pred_test<-predict(mytree, newdata = test,type="class")
mean(test$RESPONSE!=pred_test)
```

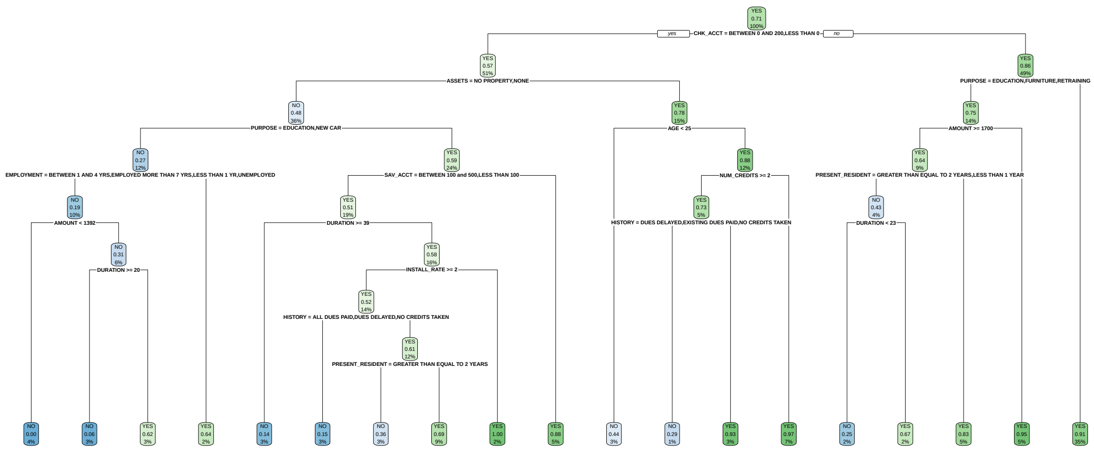
```
## [1] 0.3145631
```

```
#See Decision Tree
mytree
```

```
## n= 485
```

```
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 485 139 YES (0.28659794 0.71340206)
##     2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 248 107 YES (0.43145161 0.56854839)
##       4) ASSETS=NO PROPERTY,NONE 175  84 NO (0.52000000 0.48000000)
##         8) PURPOSE=EDUCATION,NEW CAR 59  16 NO (0.72881356 0.27118644)
##          16) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS,LESS THAN 1 YR,UNEMPLOYED 48   9
##            32) AMOUNT< 1392 19   0 NO (1.00000000 0.00000000) *
##            33) AMOUNT>=1392 29   9 NO (0.68965517 0.31034483)
##              66) DURATION>=19.5 16   1 NO (0.93750000 0.06250000) *
##              67) DURATION< 19.5 13   5 YES (0.38461538 0.61538462) *
##          17) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS 11   4 YES (0.36363636 0.63636364) *
##         9) PURPOSE=FURNITURE,OTHER,RADIO/TV,RETRAINING,USED CAR 116  48 YES (0.41379310 0.58620690)
##          18) SAV_ACCT=BETWEEN 100 and 500,LESS THAN 100 92  45 YES (0.48913043 0.51086957)
##            36) DURATION>=39 14   2 NO (0.85714286 0.14285714) *
##            37) DURATION< 39 78  33 YES (0.42307692 0.57692308)
##              74) INSTALL_RATE>=1.5 69  33 YES (0.47826087 0.52173913)
##               148) HISTORY=ALL DUES PAID,DUES DELAYED,NO CREDITS TAKEN 13   2 NO (0.84615385 0.15384
##               149) HISTORY=CRITICAL ACCOUNT,EXISTING DUES PAID 56  22 YES (0.39285714 0.60714286)
##                 298) PRESENT_RESIDENT=GREATER THAN EQUAL TO 2 YEARS 14   5 NO (0.64285714 0.35714286
##                 299) PRESENT_RESIDENT=BETWEEN 2 AND 3 YEARS,LESS THAN 1 YEAR,MORE THAN 4 YEARS 42  1
##              75) INSTALL_RATE< 1.5 9   0 YES (0.00000000 1.00000000) *
##          19) SAV_ACCT=BETWEEN 500 AND 1000,GREATER THAN EQUAL TO 1000,NO SAVINGS 24   3 YES (0.12500
##       5) ASSETS=REAL ESTATE 73  16 YES (0.21917808 0.78082192)
##        10) AGE< 24.5 16   7 NO (0.56250000 0.43750000) *
##        11) AGE>=24.5 57   7 YES (0.12280702 0.87719298)
##          22) NUM_CREDITS>=1.5 22   6 YES (0.27272727 0.72727273)
##            44) HISTORY=DUES DELAYED,EXISTING DUES PAID,NO CREDITS TAKEN 7   2 NO (0.71428571 0.28571
##            45) HISTORY=CRITICAL ACCOUNT 15   1 YES (0.06666667 0.93333333) *
##          23) NUM_CREDITS< 1.5 35   1 YES (0.02857143 0.97142857) *
##     3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 237  32 YES (0.13502110 0.86497890)
##       6) PURPOSE=EDUCATION,FURNITURE,RETRAINING 67  17 YES (0.25373134 0.74626866)
##        12) AMOUNT>=1700 45  16 YES (0.35555556 0.64444444)
##          24) PRESENT_RESIDENT=GREATER THAN EQUAL TO 2 YEARS,LESS THAN 1 YEAR 21   9 NO (0.57142857 0
##            48) DURATION< 22.5 12   3 NO (0.75000000 0.25000000) *
##            49) DURATION>=22.5 9   3 YES (0.33333333 0.66666667) *
##          25) PRESENT_RESIDENT=BETWEEN 2 AND 3 YEARS,MORE THAN 4 YEARS 24   4 YES (0.16666667 0.83333
##        13) AMOUNT< 1700 22   1 YES (0.04545455 0.95454545) *
##       7) PURPOSE=NEW CAR,OTHER,RADIO/TV,USED CAR 170  15 YES (0.08823529 0.91176471) *
```

```
#Plot Decision Tree
rpart.plot(mytree)
```




```r
#Parameters to check reliability of the model
cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##       pred
## actual  NO YES
##    NO   62  99
##    YES  63 291
```

```r
accuracy<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy
```

```
## [1] 0.6854369
```

```r
precision
```

```
## [1] 0.7461538
```

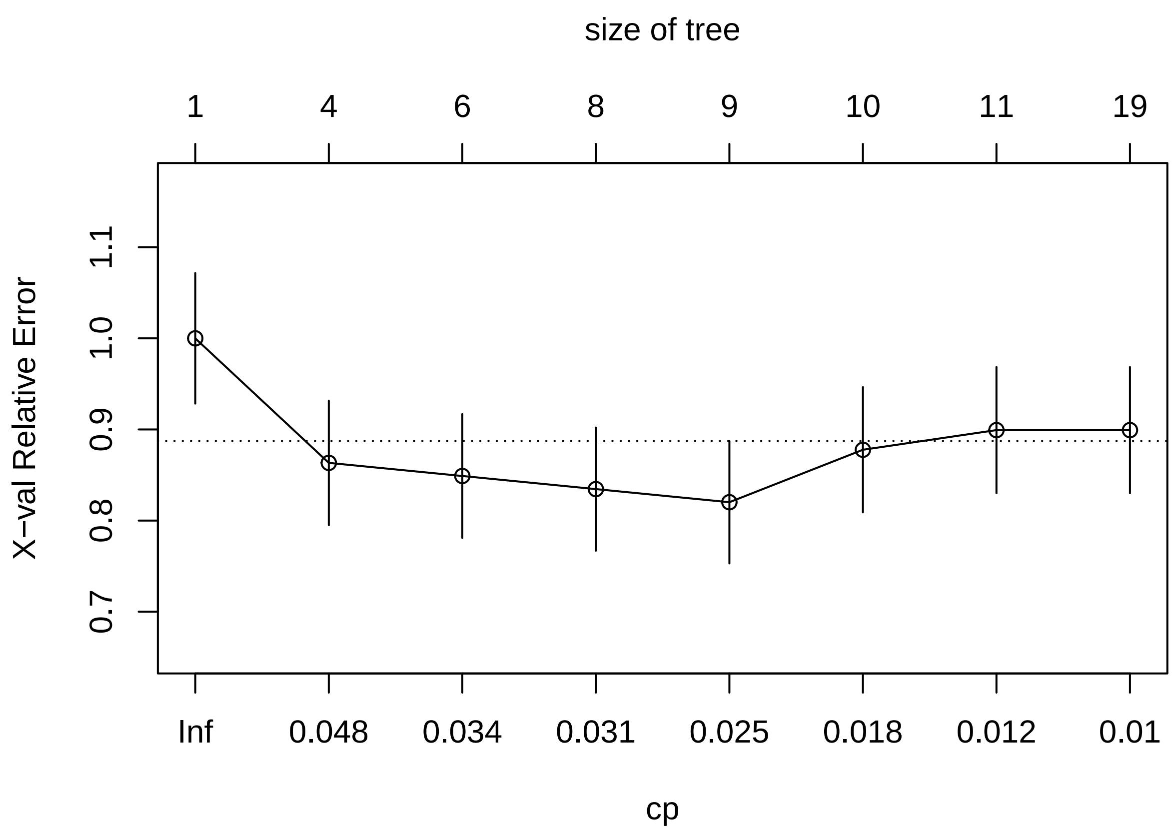
```r
recall
```

```
## [1] 0.8220339
```

Output of Model 1: For the model above, we checked how it performed on test data as well as on training data and it was found that the error rate on test was 31% while on train it was 14%. To check the model performance, we considered the parameters like recall, precision and accuracy. The accuracy of the overall model was low as 68%. Since it is a fully grown tree as there are no prunning parameters set, the model is overfitting and hence the accuracy on the test data is less. The precision value was also observed to be low (74%) and the model is giving high false positive rate. Considering all these factors, we can conclude that the model is is not reliable.

We then pruned the model to check whether its performance with different parameters like minsplit, minbucket and CP. To set the CP value, plotcp() is used. This check the minimum xerror rate so that the corresponding minimum CP value can be taken. Further, to test the model performance on different minsplit and minbucket values, we have implemented a 'for loop'. This loop inputs different values for minsplit and minbucket and checks the error rate on train and test data. We then select the values with min error rate and perform pre-pruning on our model to check its reliability. Here, plotcp() gives the min CP of 0.01 at the xerror 0.89 value.

```
plotcp(mytree)
```




```
#Gini for 0.5 and 0.5 (Default Tree with pruning)
#for loop
x<-c(1:10)
msplit <- 3
mbucket <- 1

for (val in x) {
```

```
myFormula = RESPONSE ~. -`OBS#`
myTree2 <- rpart(myFormula, data = train, control = rpart.control(minsplit = msplit, minbucket = mbucke
cat("Min Split :",msplit, "Min Bucket:", mbucket,"\n")

pred_train<-predict(myTree2,data=train,type="class")
print(mean(train$RESPONSE != pred_train))

pred_test<-predict(myTree2, newdata = test,type="class")
print(mean(test$RESPONSE != pred_test))
print(table(actual = test$RESPONSE, pred = pred_test))

msplit<-msplit + 10
mbucket<-mbucket + 10
}
```

```
## Min Split : 3 Min Bucket: 1
## [1] 0.08865979
## [1] 0.3242718
##       pred
## actual  NO YES
##    NO   69  92
##    YES  75 279
## Min Split : 13 Min Bucket: 11
## [1] 0.171134
## [1] 0.3106796
##       pred
## actual  NO YES
##    NO   64  97
##    YES  63 291
## Min Split : 23 Min Bucket: 21
## [1] 0.2
## [1] 0.2873786
##       pred
## actual  NO YES
##    NO   75  86
##    YES  62 292
## Min Split : 33 Min Bucket: 31
## [1] 0.214433
## [1] 0.2796117
##       pred
## actual  NO YES
##    NO   62  99
##    YES  45 309
## Min Split : 43 Min Bucket: 41
## [1] 0.2247423
## [1] 0.2932039
##       pred
## actual  NO YES
##    NO   69  92
##    YES  59 295
## Min Split : 53 Min Bucket: 51
## [1] 0.2309278
## [1] 0.3067961
```

```
##       pred
## actual  NO YES
##    NO   33 128
##    YES  30 324
## Min Split : 63 Min Bucket: 61
## [1] 0.2371134
## [1] 0.3087379
##       pred
## actual  NO YES
##    NO   42 119
##    YES  40 314
## Min Split : 73 Min Bucket: 71
## [1] 0.2371134
## [1] 0.3087379
##       pred
## actual  NO YES
##    NO   42 119
##    YES  40 314
## Min Split : 83 Min Bucket: 81
## [1] 0.2618557
## [1] 0.3339806
##       pred
## actual  NO YES
##    NO   70  91
##    YES  81 273
## Min Split : 93 Min Bucket: 91
## [1] 0.2618557
## [1] 0.3339806
##       pred
## actual  NO YES
##    NO   70  91
##    YES  81 273
```

We executed the 'for loop' and initialized minsplit to 3 and minbucket to 1 and at every iteration we incremented the value of minsplit and minbucket by 10. Finally, we noted the min value of test error for the corresponding minsplit and minbucket values. Here, we selected the Min Split : 33 and Min Bucket: 31 as it has the minimum test error. Pruning on the model is done below considering these values of minsplit, minbucket and CP.

```
#Choosing the best prune tree with lowest test error

myTreeprun <- rpart(myFormula, data = train, control = rpart.control(minsplit = 33, minbucket = 31, cp 

#Train error
pred_train<-predict(myTreeprun,data=train,type="class")
mean(train$RESPONSE != pred_train)
```

```
## [1] 0.214433
```

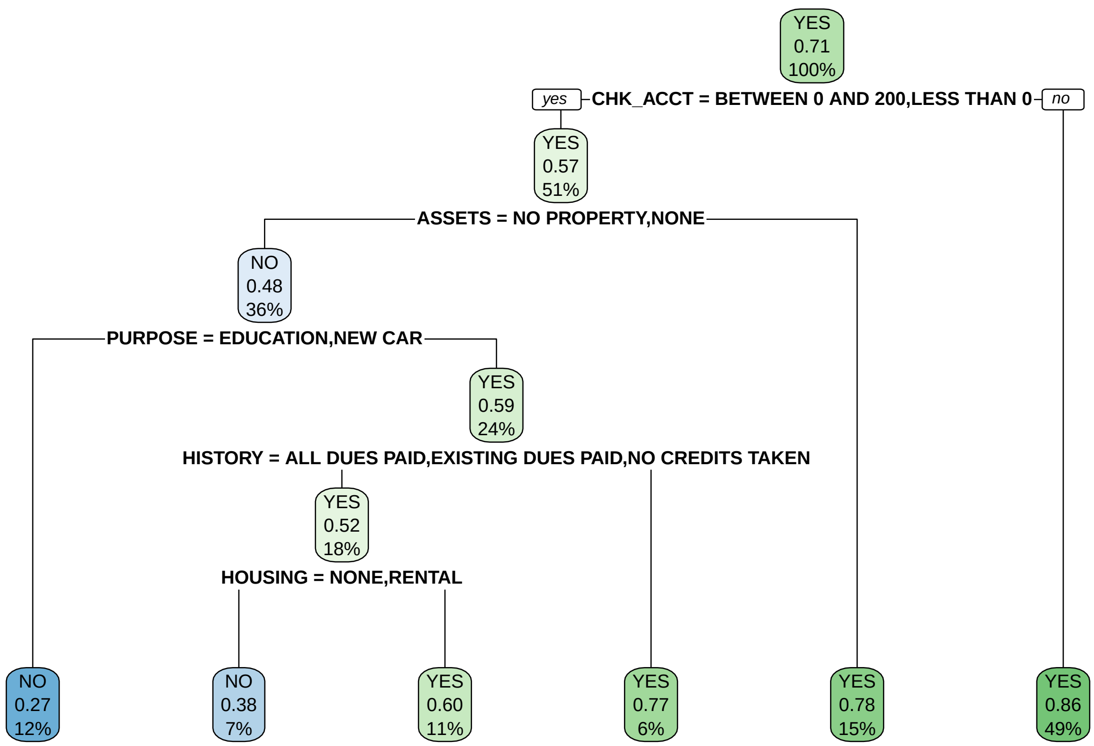
```
#Test error
pred_test<-predict(myTreeprun, newdata = test,type="class")
mean(test$RESPONSE != pred_test)
```

```
## [1] 0.2796117
```

```r
myTreeprun
```

```
## n= 485
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 485 139 YES (0.2865979 0.7134021)
##    2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 248 107 YES (0.4314516 0.5685484)
##      4) ASSETS=NO PROPERTY,NONE 175  84 NO (0.5200000 0.4800000)
##        8) PURPOSE=EDUCATION,NEW CAR 59  16 NO (0.7288136 0.2711864) *
##        9) PURPOSE=FURNITURE,OTHER,RADIO/TV,RETRAINING,USED CAR 116  48 YES (0.4137931 0.5862069)
##         18) HISTORY=ALL DUES PAID,EXISTING DUES PAID,NO CREDITS TAKEN 85  41 YES (0.4823529 0.517647
##           36) HOUSING=NONE,RENTAL 32  12 NO (0.6250000 0.3750000) *
##           37) HOUSING=OWN RESIDENCE 53  21 YES (0.3962264 0.6037736) *
##         19) HISTORY=CRITICAL ACCOUNT,DUES DELAYED 31   7 YES (0.2258065 0.7741935) *
##      5) ASSETS=REAL ESTATE 73  16 YES (0.2191781 0.7808219) *
##    3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 237  32 YES (0.1350211 0.8649789) *
```

```r
rpart.plot(myTreeprun)
```




```r
#Parameters to check reliability of the model
cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##       pred
## actual  NO YES
##    NO   62  99
##    YES  45 309
```

```
accuracy55<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision55<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall55<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy55
```

```
## [1] 0.7203883
```

```
precision55
```

```
## [1] 0.7573529
```

```
recall55
```

```
## [1] 0.8728814
```

Output: After pruning the data, there was no changed observed in the accuracy, recall or precision values. From this we can conclude that since there is no improvement even after pruning, the default model was already using the best decision tree parameters.It can be seen that even after prunning the performance parameters have no improvement. This could be because the model is under fit and gives no improvement on the test data. Thus the 50-50 cannot be considered a reliable model.

Model 2: Splitting the data as 80% Training and 20% Test

Initially, we are building our model without pre-pruning the data to find out the model performance.

```
# Gini for 0.8 and 0.2 (Default Tree without prunning)

set.seed(999)
ind<-sample(2, nrow(data_new), replace =T, prob = c(0.8, 0.2))
train<-data_new[ind==1,]
test<-data_new[ind==2,]

myFormula = RESPONSE ~. -`OBS#`

mytree2 <- rpart(myFormula, data = train,parms = list(split = "gini"))

# Train Error
pred_train<-predict(mytree2,data=train,type="class")
mean(train$RESPONSE!=pred_train)
```

```
## [1] 0.1835206
```

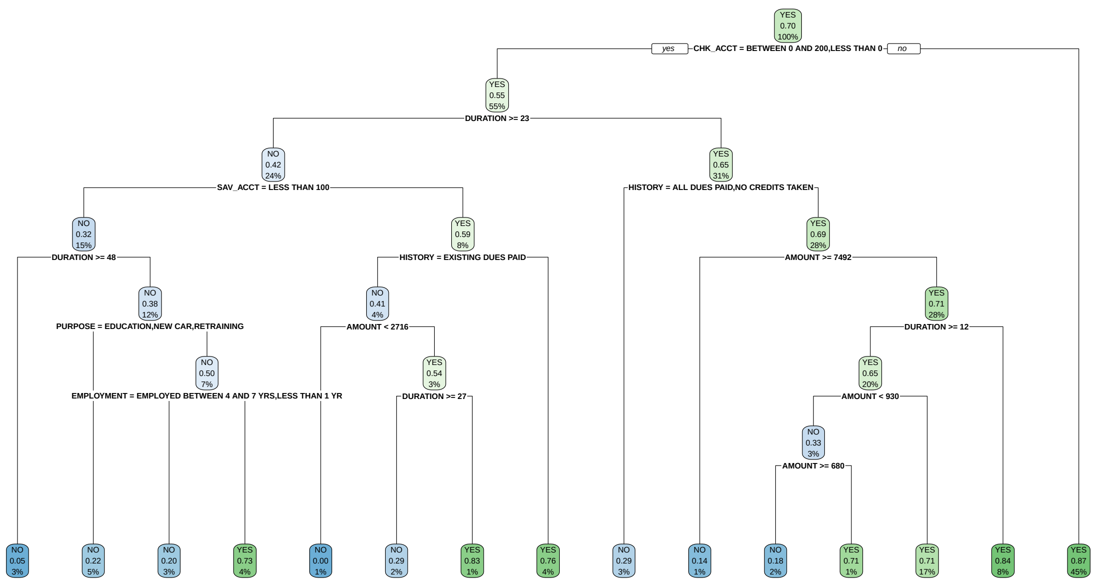
```r
#Test Error
pred_test<-predict(mytree2, newdata = test,type="class")
mean(test$RESPONSE!=pred_test)
```

```
## [1] 0.2462312
```

```r
print(mytree2)
```

```
## n= 801
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 801 244 YES (0.30461923 0.69538077)
##     2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 438 196 YES (0.44748858 0.55251142)
##       4) DURATION>=22.5 189  79 NO (0.58201058 0.41798942)
##         8) SAV_ACCT=LESS THAN 100 121  39 NO (0.67768595 0.32231405)
##          16) DURATION>=47.5 22   1 NO (0.95454545 0.04545455) *
##          17) DURATION< 47.5 99  38 NO (0.61616162 0.38383838)
##            34) PURPOSE=EDUCATION,NEW CAR,RETRAINING 41   9 NO (0.78048780 0.21951220) *
##            35) PURPOSE=FURNITURE,OTHER,RADIO/TV,USED CAR 58  29 NO (0.50000000 0.50000000)
##              70) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,LESS THAN 1 YR 25   5 NO (0.80000000 0.2000
##              71) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS,UNEMPLOYED 33   9 YES (0.27
##         9) SAV_ACCT=BETWEEN 100 and 500,BETWEEN 500 AND 1000,GREATER THAN EQUAL TO 1000,NO SAVINGS 6
##          18) HISTORY=EXISTING DUES PAID 34  14 NO (0.58823529 0.41176471)
##            36) AMOUNT< 2715.5 8   0 NO (1.00000000 0.00000000) *
##            37) AMOUNT>=2715.5 26  12 YES (0.46153846 0.53846154)
##              74) DURATION>=27 14   4 NO (0.71428571 0.28571429) *
##              75) DURATION< 27 12   2 YES (0.16666667 0.83333333) *
##          19) HISTORY=ALL DUES PAID,CRITICAL ACCOUNT,DUES DELAYED,NO CREDITS TAKEN 34   8 YES (0.2352
##       5) DURATION< 22.5 249  86 YES (0.34538153 0.65461847)
##        10) HISTORY=ALL DUES PAID,NO CREDITS TAKEN 21   6 NO (0.71428571 0.28571429) *
##        11) HISTORY=CRITICAL ACCOUNT,DUES DELAYED,EXISTING DUES PAID 228  71 YES (0.31140351 0.688596
##          22) AMOUNT>=7491.5 7   1 NO (0.85714286 0.14285714) *
##          23) AMOUNT< 7491.5 221  65 YES (0.29411765 0.70588235)
##            46) DURATION>=11.5 157  55 YES (0.35031847 0.64968153)
##              92) AMOUNT< 930 24   8 NO (0.66666667 0.33333333)
##               184) AMOUNT>=679.5 17   3 NO (0.82352941 0.17647059) *
##               185) AMOUNT< 679.5 7   2 YES (0.28571429 0.71428571) *
##              93) AMOUNT>=930 133  39 YES (0.29323308 0.70676692) *
##            47) DURATION< 11.5 64  10 YES (0.15625000 0.84375000) *
##     3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 363  48 YES (0.13223140 0.86776860) *
```

```r
rpart.plot(mytree2)
```




```r
# How reliable is the model

cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##       pred
## actual  NO YES
##    NO   22  34
##    YES  15 128
```

```r
accuracy<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy
```

```
## [1] 0.7537688
```

```r
precision
```

```
## [1] 0.7901235
```

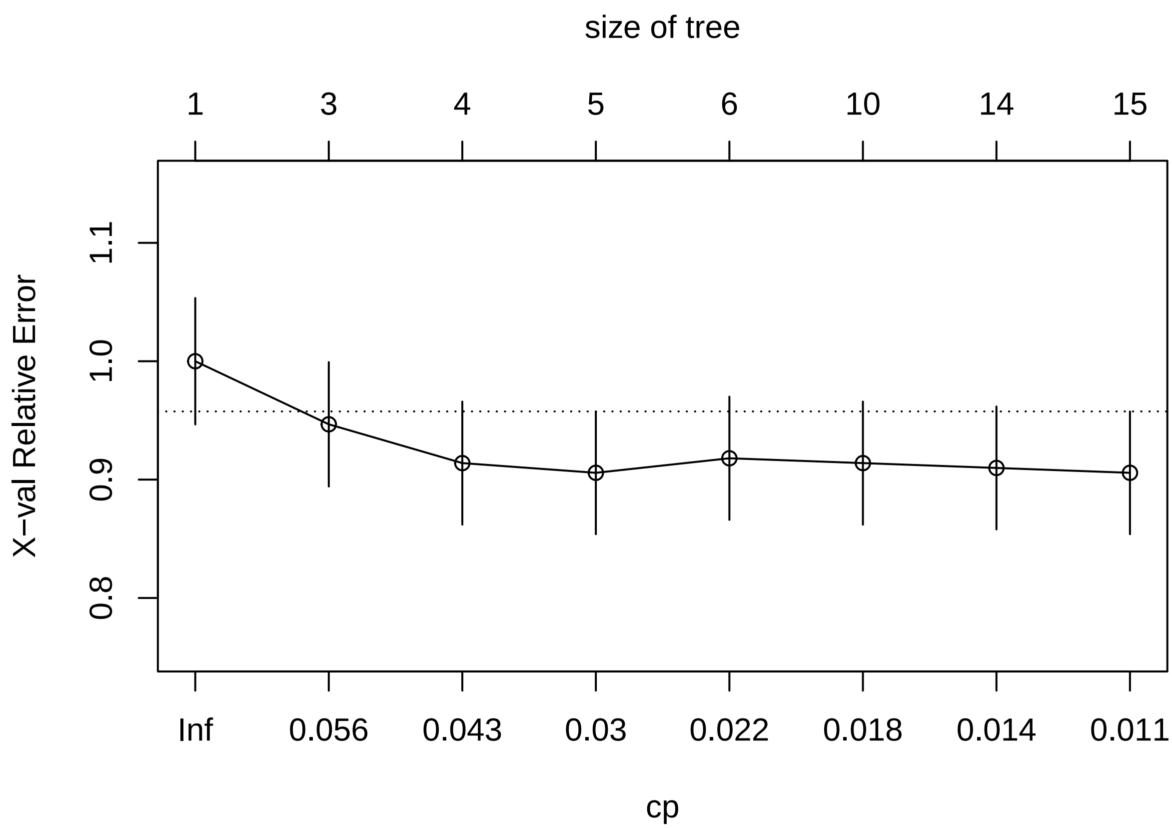
```r
recall
```

```
## [1] 0.8951049
```

Output of Model 2: For the model above, we checked how it performed on test data as well as on training data and it was found that the error rate on test was 24% while on train it was 18%. The model performance was observed on accuracy, precision and recall. The accuracy of the overall model was low as 75%. The precision value was also observed to be low (79%), while recall is 89%. We have 34 predictions misclassified as good even though they are bad.

We then pruned the model to check whether its performance with different parameters like minsplit, minbucket and CP. Here, plotcp() gives the min CP of 0.01 at the xerror 0.94 value.

```
plotcp(mytree2)
```




```
#Gini for 0.8 and 0.2 (Default Tree with prunning)
x<-c(1:10)
msplit <- 3
mbucket <- 1

for (val in x) {

myFormula = RESPONSE~. -`OBS#`
myTree <- rpart(myFormula, data = train, control = rpart.control(minsplit = msplit, minbucket = mbucket

cat("Min Split :",msplit, "Min Bucket:", mbucket,"\n")
pred_train<-predict(myTree, data = train, type = "class")
print(mean(train$RESPONSE != pred_train))
pred_test<-predict(myTree, newdata = test, type = "class")
print(mean(test$RESPONSE != pred_test))
```

```r
print(table(actual = test$RESPONSE, pred = pred_test))

msplit<-msplit + 10
mbucket<-mbucket + 10
}
```

```
## Min Split : 3 Min Bucket: 1
## [1] 0.1722846
## [1] 0.2462312
##        pred
## actual  NO YES
##     NO  22  34
##     YES 15 128
## Min Split : 13 Min Bucket: 11
## [1] 0.1935081
## [1] 0.2361809
##        pred
## actual  NO YES
##     NO  20  36
##     YES 11 132
## Min Split : 23 Min Bucket: 21
## [1] 0.2047441
## [1] 0.2261307
##        pred
## actual  NO YES
##     NO  24  32
##     YES 13 130
## Min Split : 33 Min Bucket: 31
## [1] 0.2322097
## [1] 0.2562814
##        pred
## actual  NO YES
##     NO  28  28
##     YES 23 120
## Min Split : 43 Min Bucket: 41
## [1] 0.2397004
## [1] 0.2562814
##        pred
## actual  NO YES
##     NO  25  31
##     YES 20 123
## Min Split : 53 Min Bucket: 51
## [1] 0.2397004
## [1] 0.2562814
##        pred
## actual  NO YES
##     NO  25  31
##     YES 20 123
## Min Split : 63 Min Bucket: 61
## [1] 0.2409488
## [1] 0.2562814
##        pred
## actual  NO YES
```

```
##    NO   26  30
##    YES  21 122
## Min Split : 73 Min Bucket: 71
## [1] 0.2521848
## [1] 0.2663317
##       pred
## actual  NO YES
##    NO   24  32
##    YES  21 122
## Min Split : 83 Min Bucket: 81
## [1] 0.2659176
## [1] 0.281407
##       pred
## actual  NO YES
##    NO   24  32
##    YES  24 119
## Min Split : 93 Min Bucket: 91
## [1] 0.2659176
## [1] 0.281407
##       pred
## actual  NO YES
##    NO   24  32
##    YES  24 119
```

We executed the for loop and initialized minsplit to 3 and minbucket to 1 and at every iteration we incremented the value of minsplit and minbucket by 10. Finally, we noted the min value of test error for the corresponding minsplit and minbucket values. Here, we selected the Min Split : 23 and Min Bucket: 21 as it had the minimum error rate for the test data. Pruning on the model is done below considering these values of minsplit, minbucket and CP.

```
#choosing the best prunned tree with lowest test error

myTreeprun <- rpart(myFormula, data = train, control = rpart.control(minsplit = 23, minbucket = 21, cp =

# Train Error
pred_train<-predict(myTreeprun,data=train,type="class")
mean(train$RESPONSE!=pred_train)
```

```
## [1] 0.2047441
```

```
#Test Error
pred_test<-predict(myTreeprun, newdata = test,type="class")
mean(test$RESPONSE!=pred_test)
```

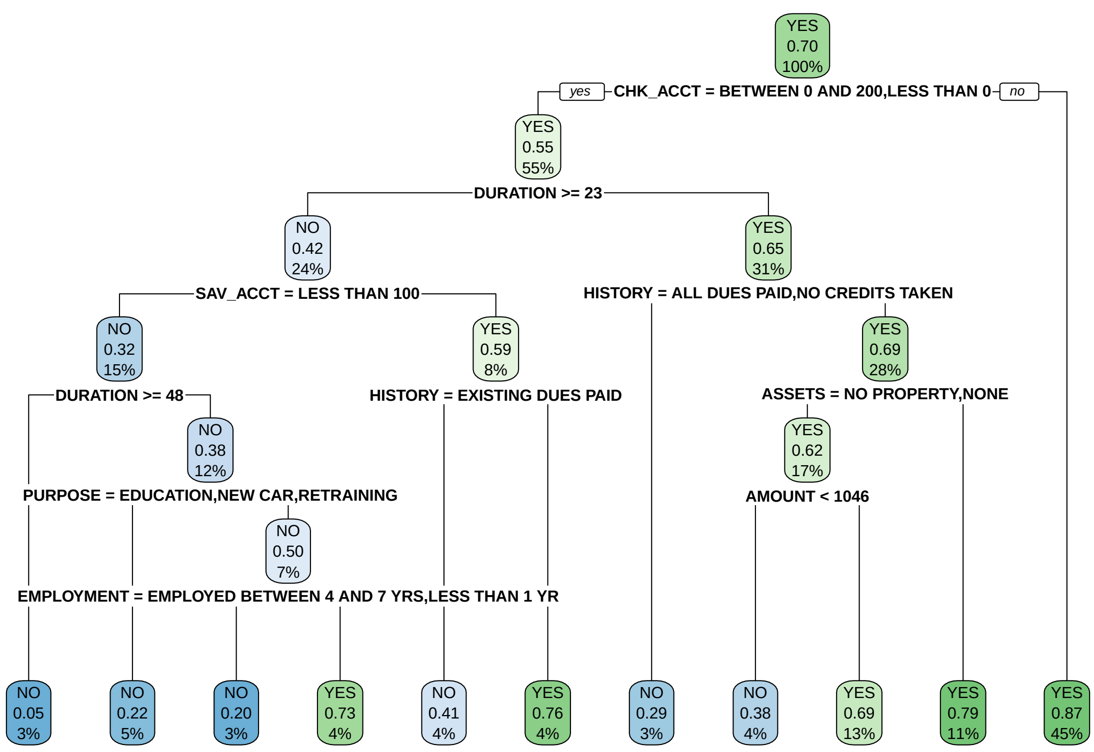
```
## [1] 0.2261307
```

```
print(myTreeprun)
```

```
## n= 801
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
```

```
##
##  1) root 801 244 YES (0.30461923 0.69538077)
##    2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 438 196 YES (0.44748858 0.55251142)
##      4) DURATION>=22.5 189  79 NO (0.58201058 0.41798942)
##        8) SAV_ACCT=LESS THAN 100 121  39 NO (0.67768595 0.32231405)
##         16) DURATION>=47.5 22   1 NO (0.95454545 0.04545455) *
##         17) DURATION< 47.5 99  38 NO (0.61616162 0.38383838)
##           34) PURPOSE=EDUCATION,NEW CAR,RETRAINING 41   9 NO (0.78048780 0.21951220) *
##           35) PURPOSE=FURNITURE,OTHER,RADIO/TV,USED CAR 58  29 NO (0.50000000 0.50000000)
##             70) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,LESS THAN 1 YR 25   5 NO (0.80000000 0.200000
##             71) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS,UNEMPLOYED 33   9 YES (0.272
##        9) SAV_ACCT=BETWEEN 100 and 500,BETWEEN 500 AND 1000,GREATER THAN EQUAL TO 1000,NO SAVINGS 68
##         18) HISTORY=EXISTING DUES PAID 34  14 NO (0.58823529 0.41176471) *
##         19) HISTORY=ALL DUES PAID,CRITICAL ACCOUNT,DUES DELAYED,NO CREDITS TAKEN 34   8 YES (0.23529
##      5) DURATION< 22.5 249  86 YES (0.34538153 0.65461847)
##       10) HISTORY=ALL DUES PAID,NO CREDITS TAKEN 21   6 NO (0.71428571 0.28571429) *
##       11) HISTORY=CRITICAL ACCOUNT,DUES DELAYED,EXISTING DUES PAID 228  71 YES (0.31140351 0.6885964
##         22) ASSETS=NO PROPERTY,NONE 137  52 YES (0.37956204 0.62043796)
##           44) AMOUNT< 1045.5 29  11 NO (0.62068966 0.37931034) *
##           45) AMOUNT>=1045.5 108  34 YES (0.31481481 0.68518519) *
##         23) ASSETS=REAL ESTATE 91  19 YES (0.20879121 0.79120879) *
##    3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 363  48 YES (0.13223140 0.86776860) *
```

```
rpart.plot(myTreeprun)
```



```r
# How reliable is the prunned model


cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##       pred
## actual  NO YES
##    NO   24  32
##    YES  13 130
```

```r
accuracy82<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision82<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall82<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy82
```

```
## [1] 0.7738693
```

```r
precision82
```

```
## [1] 0.8024691
```

```r
recall82
```

```
## [1] 0.9090909
```

Output of Model 2 after pruning: After pruning the data, we found that the performance of the model increases and it is more reliable as the accuracy increases from 75% in the non-pruned default tree to 77%. We can also see that the precision also increases by 1% from the previous model.It means that it will have an increased rate of observations that are actually positive and are predicted positive. Similarly since recall is increased the False negative count of the model is less. It can be clearly seen that the split criteria of 80-20 performs far better than the 50-50 since the performance parameters show improvements. This can be seen because the 50-50 model was under fit as the train data was less for good predictions. With increase in the precision rate it can be concluded that this model will have less false positives which can be seen in the confusion matrix above.

Model 3:Splitting the data as 75% Training and 25% Test

Initially, we our building our model without pre-pruning the data to find out the model performance.

```r
# Gini for 0.75 and 0.25 (Default Tree without prunning)

data_new$RESPONSE<- as.factor(data_new$RESPONSE)

set.seed(777)
ind<-sample(2, nrow(data_new), replace =T, prob = c(0.75, 0.25))
train<-data_new[ind==1,]
test<-data_new[ind==2,]
```

```r
myFormula = RESPONSE ~. -`OBS#`

mytree2 <- rpart(myFormula, data = train,parms = list(split = "gini"))

# Train Error
pred_train<-predict(mytree2,data=train,type="class")
mean(train$RESPONSE!=pred_train)
```

```
## [1] 0.1828794
```

```r
#Test Error
pred_test<-predict(mytree2, newdata = test,type="class")
mean(test$RESPONSE!=pred_test)
```

```
## [1] 0.2576419
```

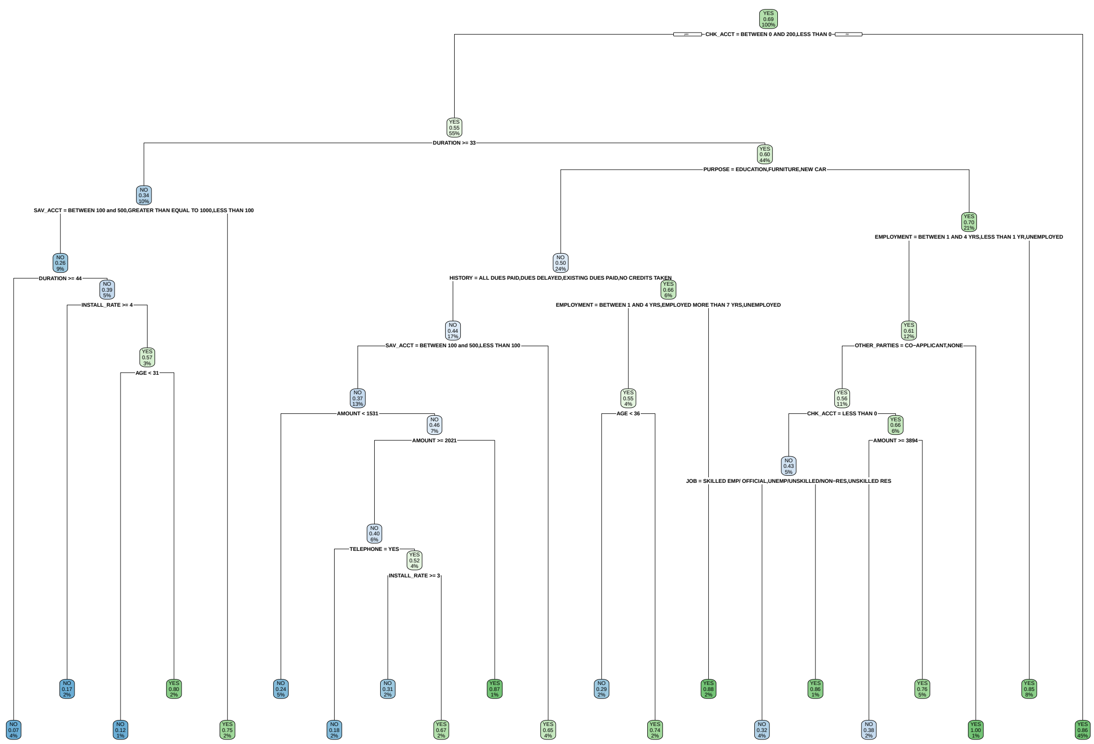
```r
print(mytree2)
```

```
## n= 771
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 771 239 YES (0.30998703 0.69001297)
##     2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 421 191 YES (0.45368171 0.54631829)
##       4) DURATION>=33 80  27 NO (0.66250000 0.33750000)
##         8) SAV_ACCT=BETWEEN 100 and 500,GREATER THAN EQUAL TO 1000,LESS THAN 100 68  18 NO (0.735294
##          16) DURATION>=43.5 27   2 NO (0.92592593 0.07407407) *
##          17) DURATION< 43.5 41  16 NO (0.60975610 0.39024390)
##            34) INSTALL_RATE>=3.5 18   3 NO (0.83333333 0.16666667) *
##            35) INSTALL_RATE< 3.5 23  10 YES (0.43478261 0.56521739)
##              70) AGE< 31 8   1 NO (0.87500000 0.12500000) *
##              71) AGE>=31 15   3 YES (0.20000000 0.80000000) *
##         9) SAV_ACCT=BETWEEN 500 AND 1000,NO SAVINGS 12   3 YES (0.25000000 0.75000000) *
##       5) DURATION< 33 341 138 YES (0.40469208 0.59530792)
##        10) PURPOSE=EDUCATION,FURNITURE,NEW CAR 182  91 NO (0.50000000 0.50000000)
##          20) HISTORY=ALL DUES PAID,DUES DELAYED,EXISTING DUES PAID,NO CREDITS TAKEN 132  58 NO (0.56
##            40) SAV_ACCT=BETWEEN 100 and 500,LESS THAN 100 98  36 NO (0.63265306 0.36734694)
##              80) AMOUNT< 1531 42  10 NO (0.76190476 0.23809524) *
##              81) AMOUNT>=1531 56  26 NO (0.53571429 0.46428571)
##               162) AMOUNT>=2020.5 48  19 NO (0.60416667 0.39583333)
##                 324) TELEPHONE=YES 17   3 NO (0.82352941 0.17647059) *
##                 325) TELEPHONE=NO 31  15 YES (0.48387097 0.51612903)
##                   650) INSTALL_RATE>=2.5 13   4 NO (0.69230769 0.30769231) *
##                   651) INSTALL_RATE< 2.5 18   6 YES (0.33333333 0.66666667) *
##               163) AMOUNT< 2020.5 8   1 YES (0.12500000 0.87500000) *
##            41) SAV_ACCT=BETWEEN 500 AND 1000,GREATER THAN EQUAL TO 1000,NO SAVINGS 34  12 YES (0.352
##          21) HISTORY=CRITICAL ACCOUNT 50  17 YES (0.34000000 0.66000000)
##            42) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS,UNEMPLOYED 33  15 YES (0.4545
##              84) AGE< 35.5 14   4 NO (0.71428571 0.28571429) *
##              85) AGE>=35.5 19   5 YES (0.26315789 0.73684211) *
```

```
##           43) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,LESS THAN 1 YR 17   2 YES (0.11764706 0.88235
##        11) PURPOSE=OTHER,RADIO/TV,RETRAINING,USED CAR 159  47 YES (0.29559748 0.70440252)
##          22) EMPLOYMENT=BETWEEN 1 AND 4 YRS,LESS THAN 1 YR,UNEMPLOYED 94  37 YES (0.39361702 0.60638
##            44) OTHER_PARTIES=CO-APPLICANT,NONE 85  37 YES (0.43529412 0.56470588)
##              88) CHK_ACCT=LESS THAN 0 35  15 NO (0.57142857 0.42857143)
##               176) JOB=SKILLED EMP/ OFFICIAL,UNEMP/UNSKILLED/NON-RES,UNSKILLED RES 28   9 NO (0.6785
##               177) JOB=SELF/HIGHLY QUALIFIED EMP 7   1 YES (0.14285714 0.85714286) *
##              89) CHK_ACCT=BETWEEN 0 AND 200 50  17 YES (0.34000000 0.66000000)
##               178) AMOUNT>=3893.5 13   5 NO (0.61538462 0.38461538) *
##               179) AMOUNT< 3893.5 37   9 YES (0.24324324 0.75675676) *
##            45) OTHER_PARTIES=GUARANTOR 9   0 YES (0.00000000 1.00000000) *
##          23) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,EMPLOYED MORE THAN 7 YRS 65  10 YES (0.15384615
##     3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 350  48 YES (0.13714286 0.86285714) *
```

```
rpart.plot(mytree2)
```

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



```
# How reliable is the model

cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##       pred
```

```
## actual  NO YES
##    NO   29  32
##    YES  27 141
```

```
accuracy<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy
```

```
## [1] 0.7423581
```

```
precision
```

```
## [1] 0.8150289
```

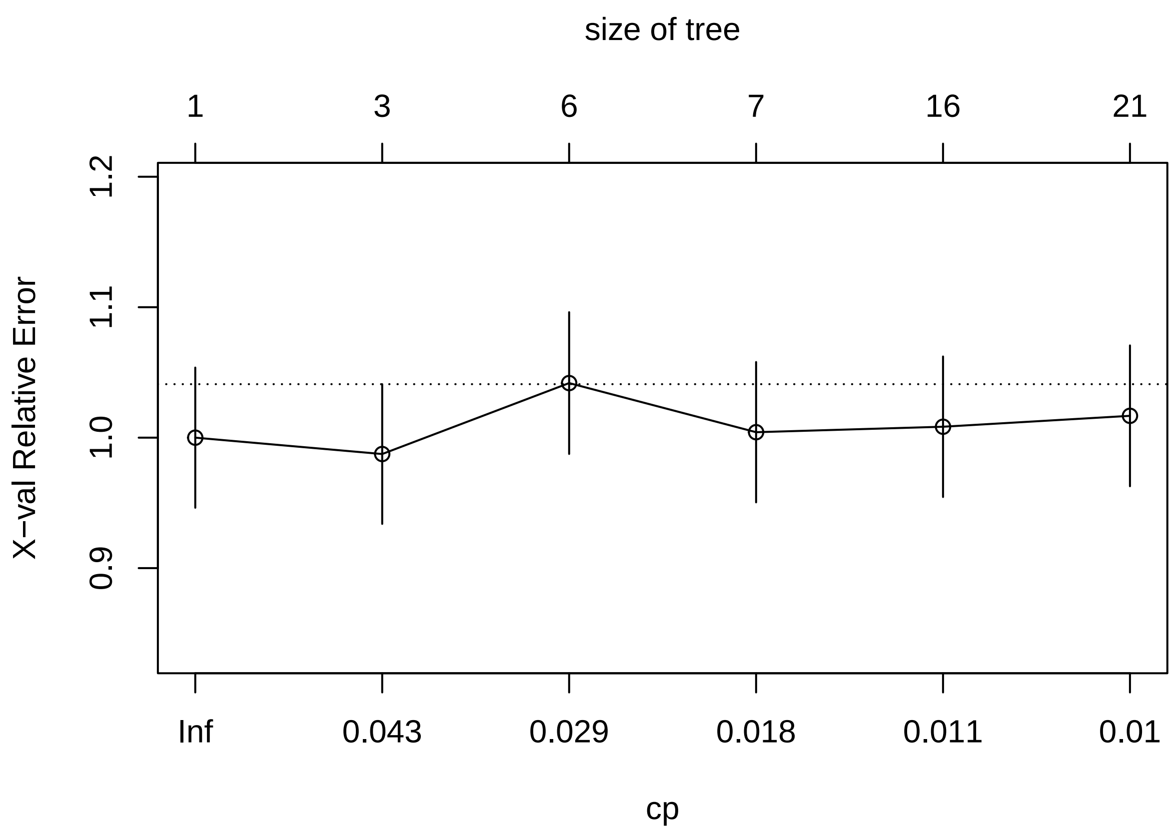
```
recall
```

```
## [1] 0.8392857
```

Output of Model 3: For the model above, we checked how it performed on test data as well as on training data and it was found that the error rate on test was 25% while on train it was 18%. To check the model performance, we considered the parameters like recall, precision and accuracy. The accuracy of the overall model was low as 74%. The precision value was also observed to be low (74%). Considering all these factors, we can conclude that the model is is not reliable.

We then pruned the model to check whether its performance with different parameters like minsplit, minbucket and CP. To set the CP value, plotcp() is used. This check the minimum xerror rate so that the corresponding minimum CP value can be taken. Further, to test the model performance on different minsplit and minbucket values, we have implemented a 'for loop'. This loop inputs different values for minsplit and minbucket and checks the error rate on train and test data. We then select the values with min error rate and perform pre-pruning on our model to check its reliability. Here, plotcp() gives the min CP of 0.01 at the xerror 0.90 value.

```
plotcp(mytree2)
```




```
#Gini for 0.75 and 0.25 (Default Tree with prunning)
x<-c(1:10)
msplit <- 3
mbucket <- 1

for (val in x) {

myFormula = RESPONSE~. -`OBS#`
myTree <- rpart(myFormula, data = train, control = rpart.control(minsplit = msplit, minbucket = mbucket

cat("Min Split :",msplit, "Min Bucket:", mbucket,"\n")
pred_train<-predict(myTree, data = train, type = "class")
print(mean(train$RESPONSE != pred_train))
pred_test<-predict(myTree, newdata = test, type = "class")
print(mean(test$RESPONSE != pred_test))
print(table(actual = test$RESPONSE, pred = pred_test))

msplit<-msplit + 10
mbucket<-mbucket + 10
}
```

```
## Min Split : 3 Min Bucket: 1
## [1] 0.1802853
## [1] 0.2489083
##       pred
## actual  NO YES
```

```
##    NO   28  33
##    YES  24 144
## Min Split : 13 Min Bucket: 11
## [1] 0.2127108
## [1] 0.2663755
##       pred
## actual  NO YES
##    NO   22  39
##    YES  22 146
## Min Split : 23 Min Bucket: 21
## [1] 0.228275
## [1] 0.2401747
##       pred
## actual  NO YES
##    NO   22  39
##    YES  16 152
## Min Split : 33 Min Bucket: 31
## [1] 0.2425422
## [1] 0.2489083
##       pred
## actual  NO YES
##    NO   31  30
##    YES  27 141
## Min Split : 43 Min Bucket: 41
## [1] 0.2477302
## [1] 0.2358079
##       pred
## actual  NO YES
##    NO   24  37
##    YES  17 151
## Min Split : 53 Min Bucket: 51
## [1] 0.2542153
## [1] 0.2620087
##       pred
## actual  NO YES
##    NO   30  31
##    YES  29 139
## Min Split : 63 Min Bucket: 61
## [1] 0.2542153
## [1] 0.2620087
##       pred
## actual  NO YES
##    NO   30  31
##    YES  29 139
## Min Split : 73 Min Bucket: 71
## [1] 0.2542153
## [1] 0.2620087
##       pred
## actual  NO YES
##    NO   30  31
##    YES  29 139
## Min Split : 83 Min Bucket: 81
## [1] 0.2801556
## [1] 0.231441
```

```
##       pred
## actual  NO YES
##    NO   30  31
##    YES  22 146
## Min Split : 93 Min Bucket: 91
## [1] 0.2801556
## [1] 0.231441
##       pred
## actual  NO YES
##    NO   30  31
##    YES  22 146
```

We executed the for loop and initialized minsplit to 3 and minbucket to 1 and at every iteration we incremented the value of minsplit and minbucket by 10. Finally, we noted the min value of test error for the corresponding minsplit and minbucket values. Here, we selected the Min Split : 43 and Min Bucket: 41 it has the minimum test error. Pruning on the model is done below considering these values of minsplit, minbucket and CP.

```r
#choosing the best prunned tree with lowest test error

myTreeprun <- rpart(myFormula, data = train, control = rpart.control(minsplit = 43, minbucket = 41, cp =

# Train Error
pred_train<-predict(myTreeprun,data=train,type="class")
mean(train$RESPONSE!=pred_train)
```

```
## [1] 0.2477302
```

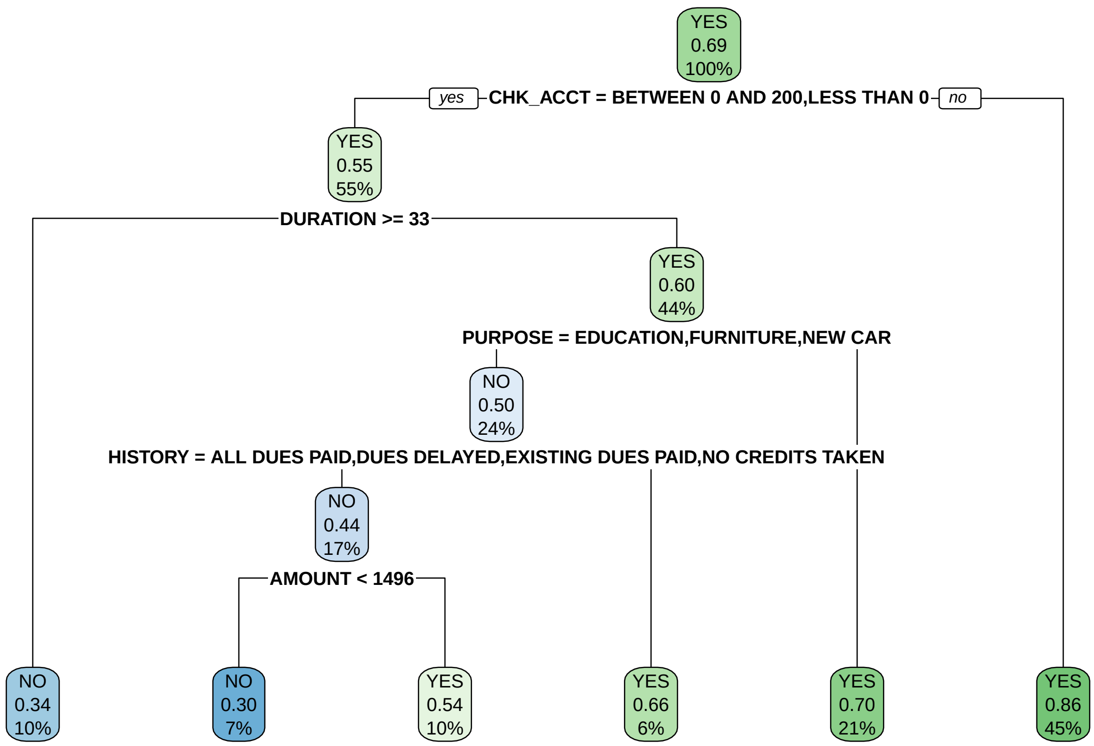
```r
#Test Error
pred_test<-predict(myTreeprun, newdata = test,type="class")
mean(test$RESPONSE!=pred_test)
```

```
## [1] 0.2358079
```

```r
print(myTreeprun)
```

```
## n= 771
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 771 239 YES (0.3099870 0.6900130)
##    2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 421 191 YES (0.4536817 0.5463183)
##      4) DURATION>=33 80  27 NO (0.6625000 0.3375000) *
##      5) DURATION< 33 341 138 YES (0.4046921 0.5953079)
##       10) PURPOSE=EDUCATION,FURNITURE,NEW CAR 182  91 NO (0.5000000 0.5000000)
##         20) HISTORY=ALL DUES PAID,DUES DELAYED,EXISTING DUES PAID,NO CREDITS TAKEN 132  58 NO (0.560
##           40) AMOUNT< 1495.5 54  16 NO (0.7037037 0.2962963) *
##           41) AMOUNT>=1495.5 78  36 YES (0.4615385 0.5384615) *
##         21) HISTORY=CRITICAL ACCOUNT 50  17 YES (0.3400000 0.6600000) *
##       11) PURPOSE=OTHER,RADIO/TV,RETRAINING,USED CAR 159  47 YES (0.2955975 0.7044025) *
##    3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 350  48 YES (0.1371429 0.8628571) *
```

```
rpart.plot(myTreeprun)
```




```
# How reliable is the prunned model

cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##        pred
## actual  NO YES
##    NO   24  37
##    YES  17 151
```

```
accuracy72<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision72<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall72<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy72
```

```
## [1] 0.7641921
```

```
precision72
```

```
## [1] 0.8031915
```

```
recall72
```

```
## [1] 0.8988095
```

Output of Model 3 after pruning: After pruning the data, we found that the performance of the model increases and it is more reliable as the accuracy increases from 74% in the non-pruned default tree to 74%. We can also see that the precision decreases even though the accuracy increases. It means that there is high chance that the model will not predict positive even if the observation is actually positive. But since the recall has increased by 6% from the non pruned default tree, the percentage of actual positive predicted correctly will be great. When compared to Model 2(80-20) this model has low performance parameters. One reason could be that the model could be overfitted and since all the performance parameters are based on the test data. The over fitted model shall not be that great with unseen data. Till now the best model is model 3(75-25). **

Model 4: Splitting the data as 70% Training and 30% Test

Initially, we our building our model without pre-pruning the data to find out the model performance.

```
#Gini for 0.7 and 0.3 (Default Tree without pruning)

data_new$RESPONSE<- as.factor(data_new$RESPONSE)
#The random seed is set to a fixed value below to make the results reproducible.
set.seed(1123)
#Splitting criteria
ind<-sample(2, nrow(data_new), replace =T, prob = c(0.7, 0.3))
train<-data_new[ind==1,]
test<-data_new[ind==2,]

#myFormula specifies that the target RESPONSE is dependent variable while all others (used as ~.) are i
myFormula = RESPONSE ~. -`OBS#`
# Default decision tree without using pruning parameters
mytree <- rpart(myFormula, data = train)

#Check Train error
pred_train<-predict(mytree,data=train,type="class")
mean(train$RESPONSE!=pred_train)
```

```
## [1] 0.1528384
```

```
#Check Test error
pred_test<-predict(mytree, newdata = test,type="class")
mean(test$RESPONSE!=pred_test)
```

```
## [1] 0.2460064
```

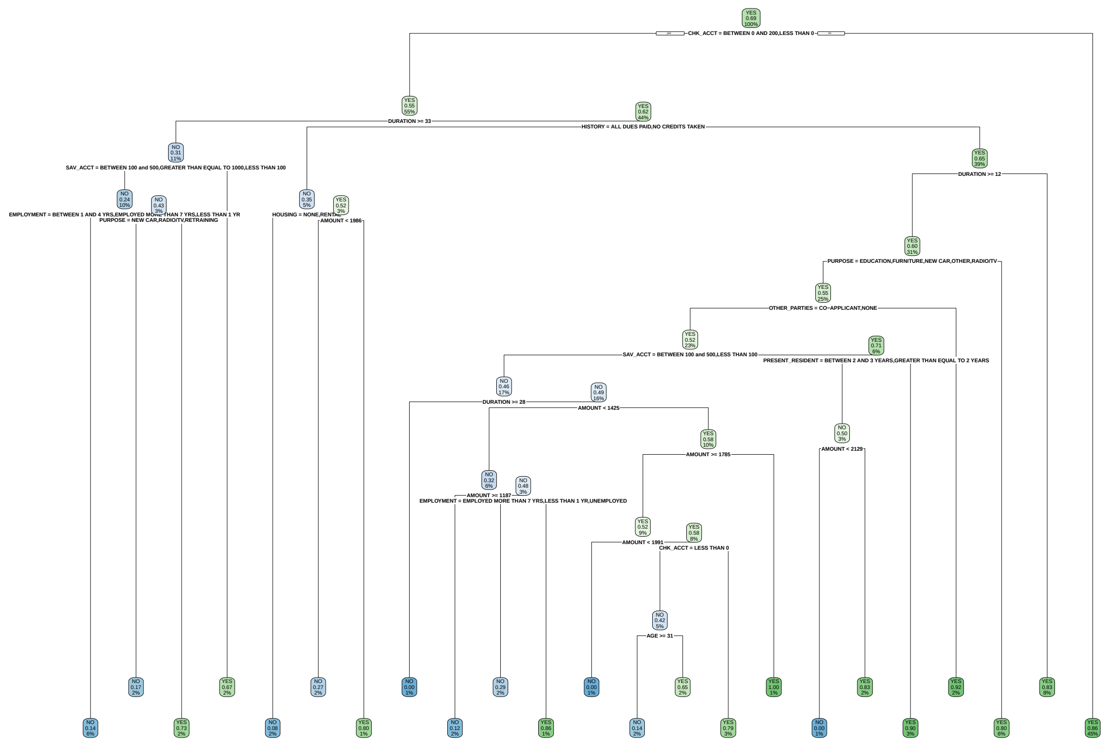
```
#See Decision Tree
mytree
```

```
## n= 687
##
```

```
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##     1) root 687 211 YES (0.30713246 0.69286754)
##       2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 377 169 YES (0.44827586 0.55172414)
##         4) DURATION>=33 78  24 NO (0.69230769 0.30769231)
##           8) SAV_ACCT=BETWEEN 100 and 500,GREATER THAN EQUAL TO 1000,LESS THAN 100 66  16 NO (0.7575
##            16) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS,LESS THAN 1 YR 43   6 NO (0.8
##            17) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,UNEMPLOYED 23  10 NO (0.56521739 0.43478261)
##              34) PURPOSE=NEW CAR,RADIO/TV,RETRAINING 12   2 NO (0.83333333 0.16666667) *
##              35) PURPOSE=EDUCATION,FURNITURE,OTHER,USED CAR 11   3 YES (0.27272727 0.72727273) *
##           9) SAV_ACCT=BETWEEN 500 AND 1000,NO SAVINGS 12   4 YES (0.33333333 0.66666667) *
##         5) DURATION< 33 299 115 YES (0.38461538 0.61538462)
##          10) HISTORY=ALL DUES PAID,NO CREDITS TAKEN 34  12 NO (0.64705882 0.35294118)
##            20) HOUSING=NONE,RENTAL 13   1 NO (0.92307692 0.07692308) *
##            21) HOUSING=OWN RESIDENCE 21  10 YES (0.47619048 0.52380952)
##              42) AMOUNT< 1986 11   3 NO (0.72727273 0.27272727) *
##              43) AMOUNT>=1986 10   2 YES (0.20000000 0.80000000) *
##          11) HISTORY=CRITICAL ACCOUNT,DUES DELAYED,EXISTING DUES PAID 265  93 YES (0.35094340 0.6490
##            22) DURATION>=11.5 211  84 YES (0.39810427 0.60189573)
##              44) PURPOSE=EDUCATION,FURNITURE,NEW CAR,OTHER,RADIO/TV 170  76 YES (0.44705882 0.552941
##                88) OTHER_PARTIES=CO-APPLICANT,NONE 157  75 YES (0.47770701 0.52229299)
##                 176) SAV_ACCT=BETWEEN 100 and 500,LESS THAN 100 116  53 NO (0.54310345 0.45689655)
##                   352) DURATION>=27.5 7   0 NO (1.00000000 0.00000000) *
##                   353) DURATION< 27.5 109  53 NO (0.51376147 0.48623853)
##                     706) AMOUNT< 1425 38  12 NO (0.68421053 0.31578947)
##                      1412) AMOUNT>=1187 17   2 NO (0.88235294 0.11764706) *
##                      1413) AMOUNT< 1187 21  10 NO (0.52380952 0.47619048)
##                        2826) EMPLOYMENT=EMPLOYED MORE THAN 7 YRS,LESS THAN 1 YR,UNEMPLOYED 14   4 NO
##                        2827) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED BETWEEN 4 AND 7 YRS 7   1 YES (
##                     707) AMOUNT>=1425 71  30 YES (0.42253521 0.57746479)
##                      1414) AMOUNT>=1785 62  30 YES (0.48387097 0.51612903)
##                        2828) AMOUNT< 1991 7   0 NO (1.00000000 0.00000000) *
##                        2829) AMOUNT>=1991 55  23 YES (0.41818182 0.58181818)
##                          5658) CHK_ACCT=LESS THAN 0 31  13 NO (0.58064516 0.41935484)
##                           11316) AGE>=30.5 14   2 NO (0.85714286 0.14285714) *
##                           11317) AGE< 30.5 17   6 YES (0.35294118 0.64705882) *
##                          5659) CHK_ACCT=BETWEEN 0 AND 200 24   5 YES (0.20833333 0.79166667) *
##                      1415) AMOUNT< 1785 9   0 YES (0.00000000 1.00000000) *
##                 177) SAV_ACCT=BETWEEN 500 AND 1000,GREATER THAN EQUAL TO 1000,NO SAVINGS 41  12 YES
##                   354) PRESENT_RESIDENT=BETWEEN 2 AND 3 YEARS,GREATER THAN EQUAL TO 2 YEARS 20  10 N
##                     708) AMOUNT< 2128.5 8   0 NO (1.00000000 0.00000000) *
##                     709) AMOUNT>=2128.5 12   2 YES (0.16666667 0.83333333) *
##                   355) PRESENT_RESIDENT=LESS THAN 1 YEAR,MORE THAN 4 YEARS 21   2 YES (0.09523810 0.
##                89) OTHER_PARTIES=GUARANTOR 13   1 YES (0.07692308 0.92307692) *
##              45) PURPOSE=RETRAINING,USED CAR 41   8 YES (0.19512195 0.80487805) *
##            23) DURATION< 11.5 54   9 YES (0.16666667 0.83333333) *
##       3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 310  42 YES (0.13548387 0.86451613)
```

```r
#Plot Decision Tree
rpart.plot(mytree)
```

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



```
#Parameters to check reliability of the model
cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##        pred
## actual  NO YES
##    NO   35  54
##    YES  23 201
```

```
accuracy<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy
```

```
## [1] 0.7539936
```

```
precision
```

```
## [1] 0.7882353
```

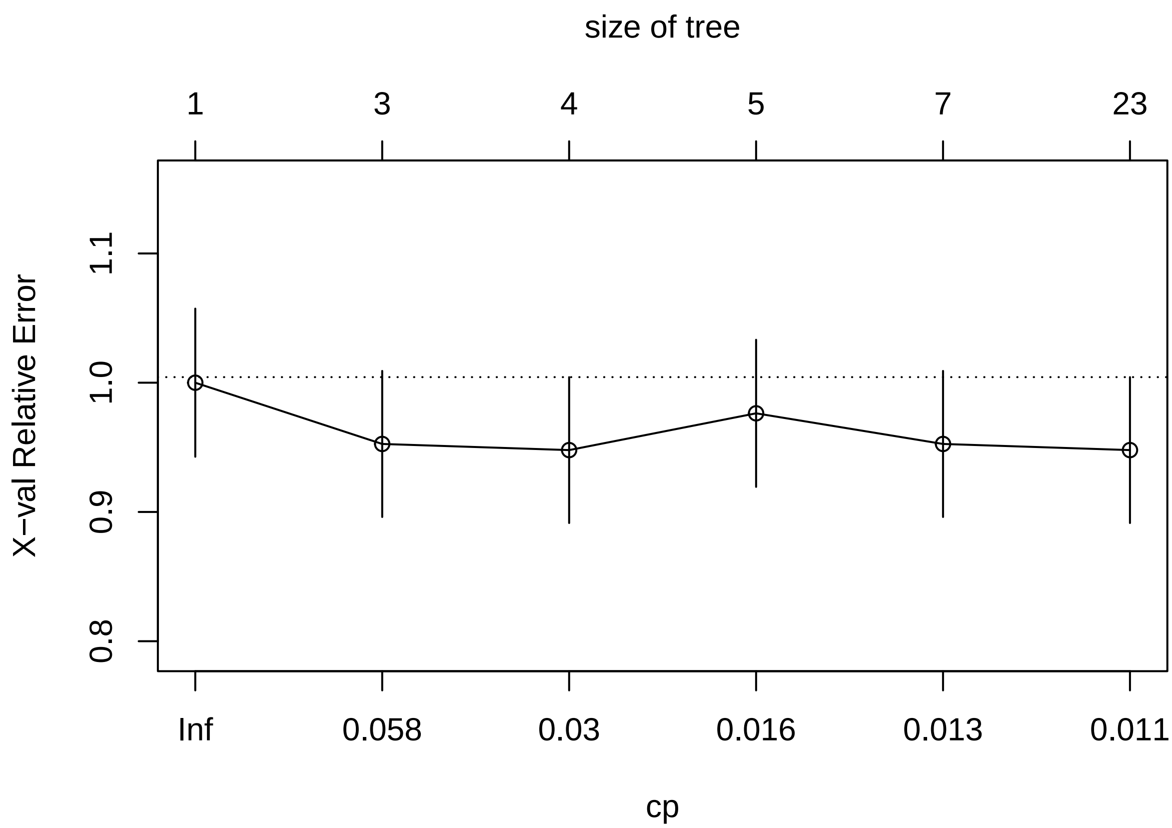
```
recall
```

```
## [1] 0.8973214
```

Output of Model 4: For the model above, we checked how it performed on test data as well as on training data and it was found that the error rate on test was 24% while on train it was 15%. The model performance was observed on accuracy, precision and recall. The accuracy of the overall model was low as 75%. The precision value was also observed to be low (78%), while recall is 89%. We have 54 predictions misclassified as good even though they are bad.

We then pruned the model to check whether its performance with different parameters like minsplit, minbucket and CP. Here, plotcp() gives the min CP of 0.01 at the xerror 0.94 value.

```
plotcp(mytree)
```




```
#Gini for 0.7 and 0.3 (Default Tree without pruning)
#for loop
x<-c(1:10)
msplit <- 0
mbucket <- 0

for (val in x) {

myFormula = RESPONSE ~. -`OBS#`
myTree2 <- rpart(myFormula, data = train, control = rpart.control(minsplit = msplit, minbucket = mbucke
cat("Min Split :",msplit, "Min Bucket:", mbucket,"\n")

pred_train<-predict(myTree2,data=train,type="class")
print(mean(train$RESPONSE != pred_train))
```

```r
pred_test<-predict(myTree2, newdata = test,type="class")
print(mean(test$RESPONSE != pred_test))
print(table(actual = test$RESPONSE, pred = pred_test))

msplit<-msplit + 10
mbucket<-mbucket + 8
}
```

```
## Min Split : 0 Min Bucket: 0
## [1] 0.1382824
## [1] 0.2492013
##       pred
## actual  NO YES
##    NO   40  49
##    YES  29 195
## Min Split : 10 Min Bucket: 8
## [1] 0.1659389
## [1] 0.2044728
##       pred
## actual  NO YES
##    NO   47  42
##    YES  22 202
## Min Split : 20 Min Bucket: 16
## [1] 0.2110626
## [1] 0.2428115
##       pred
## actual  NO YES
##    NO   53  36
##    YES  40 184
## Min Split : 30 Min Bucket: 24
## [1] 0.2256186
## [1] 0.2492013
##       pred
## actual  NO YES
##    NO   41  48
##    YES  30 194
## Min Split : 40 Min Bucket: 32
## [1] 0.2256186
## [1] 0.2492013
##       pred
## actual  NO YES
##    NO   41  48
##    YES  30 194
## Min Split : 50 Min Bucket: 40
## [1] 0.2401747
## [1] 0.2555911
##       pred
## actual  NO YES
##    NO   31  58
##    YES  22 202
## Min Split : 60 Min Bucket: 48
## [1] 0.2416303
## [1] 0.2619808
```

```
##       pred
## actual  NO YES
##    NO   32  57
##    YES  25 199
## Min Split : 70 Min Bucket: 56
## [1] 0.2518195
## [1] 0.2651757
##       pred
## actual  NO YES
##    NO   37  52
##    YES  31 193
## Min Split : 80 Min Bucket: 64
## [1] 0.2518195
## [1] 0.2651757
##       pred
## actual  NO YES
##    NO   37  52
##    YES  31 193
## Min Split : 90 Min Bucket: 72
## [1] 0.2518195
## [1] 0.2651757
##       pred
## actual  NO YES
##    NO   37  52
##    YES  31 193
```

We executed the 'for loop' and initialized minsplit to 3 and minbucket to 1 and at every iteration we incremented the value of minsplit and minbucket by 10. Finally, we noted the min value of test error for the corresponding minsplit and minbucket values. Here, we selected the Min Split : 10 and Min Bucket: 8 as it has the minimum test error. Pruning on the model is done below considering these values of minsplit, minbucket and CP.

```
#Choosing the best prune tree with lowest test error

myTreeprun <- rpart(myFormula, data = train, control = rpart.control(minsplit = 10, minbucket = 8, cp =

#Train error
pred_train<-predict(myTreeprun,data=train,type="class")
mean(train$RESPONSE != pred_train)
```

```
## [1] 0.1659389
```

```
#Test error
pred_test<-predict(myTreeprun, newdata = test,type="class")
mean(test$RESPONSE != pred_test)
```

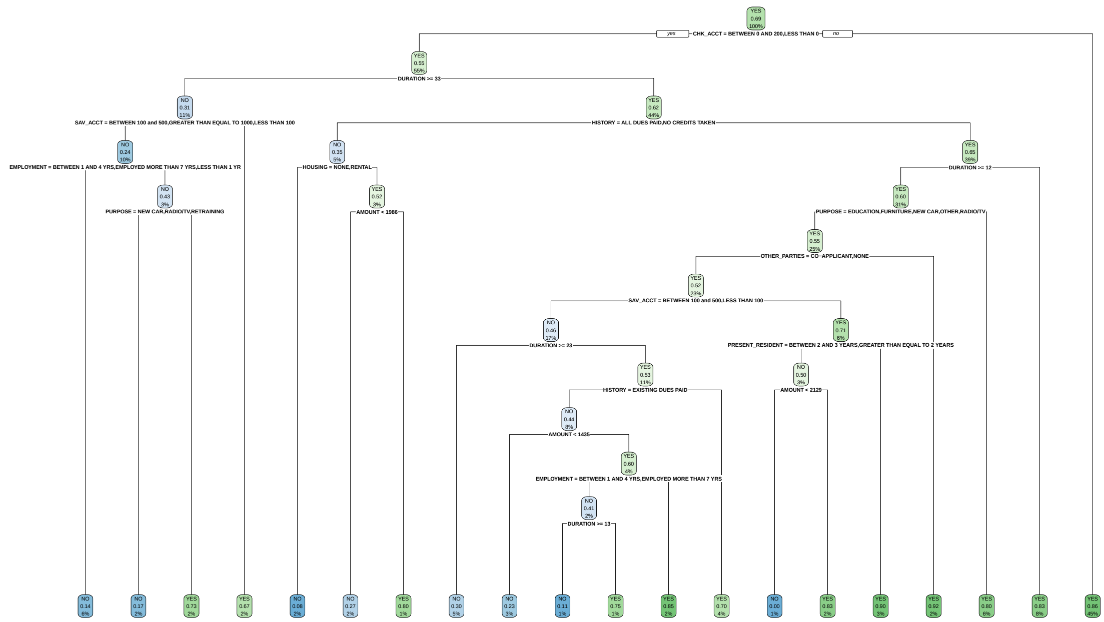
```
## [1] 0.2044728
```

```
myTreeprun
```

```
## n= 687
##
```

```
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 687 211 YES (0.30713246 0.69286754)
##      2) CHK_ACCT=BETWEEN 0 AND 200,LESS THAN 0 377 169 YES (0.44827586 0.55172414)
##        4) DURATION>=33 78  24 NO (0.69230769 0.30769231)
##          8) SAV_ACCT=BETWEEN 100 and 500,GREATER THAN EQUAL TO 1000,LESS THAN 100 66  16 NO (0.757575
##           16) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS,LESS THAN 1 YR 43   6 NO (0.860
##           17) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,UNEMPLOYED 23  10 NO (0.56521739 0.43478261)
##             34) PURPOSE=NEW CAR,RADIO/TV,RETRAINING 12   2 NO (0.83333333 0.16666667) *
##             35) PURPOSE=EDUCATION,FURNITURE,OTHER,USED CAR 11   3 YES (0.27272727 0.72727273) *
##          9) SAV_ACCT=BETWEEN 500 AND 1000,NO SAVINGS 12   4 YES (0.33333333 0.66666667) *
##        5) DURATION< 33 299 115 YES (0.38461538 0.61538462)
##         10) HISTORY=ALL DUES PAID,NO CREDITS TAKEN 34  12 NO (0.64705882 0.35294118)
##           20) HOUSING=NONE,RENTAL 13   1 NO (0.92307692 0.07692308) *
##           21) HOUSING=OWN RESIDENCE 21  10 YES (0.47619048 0.52380952)
##             42) AMOUNT< 1986 11   3 NO (0.72727273 0.27272727) *
##             43) AMOUNT>=1986 10   2 YES (0.20000000 0.80000000) *
##         11) HISTORY=CRITICAL ACCOUNT,DUES DELAYED,EXISTING DUES PAID 265  93 YES (0.35094340 0.64905
##           22) DURATION>=11.5 211  84 YES (0.39810427 0.60189573)
##             44) PURPOSE=EDUCATION,FURNITURE,NEW CAR,OTHER,RADIO/TV 170  76 YES (0.44705882 0.5529411
##               88) OTHER_PARTIES=CO-APPLICANT,NONE 157  75 YES (0.47770701 0.52229299)
##                176) SAV_ACCT=BETWEEN 100 and 500,LESS THAN 100 116  53 NO (0.54310345 0.45689655)
##                  352) DURATION>=22.5 37  11 NO (0.70270270 0.29729730) *
##                  353) DURATION< 22.5 79  37 YES (0.46835443 0.53164557)
##                    706) HISTORY=EXISTING DUES PAID 52  23 NO (0.55769231 0.44230769)
##                     1412) AMOUNT< 1435 22   5 NO (0.77272727 0.22727273) *
##                     1413) AMOUNT>=1435 30  12 YES (0.40000000 0.60000000)
##                       2826) EMPLOYMENT=BETWEEN 1 AND 4 YRS,EMPLOYED MORE THAN 7 YRS 17   7 NO (0.588
##                         5652) DURATION>=13 9   1 NO (0.88888889 0.11111111) *
##                         5653) DURATION< 13 8   2 YES (0.25000000 0.75000000) *
##                       2827) EMPLOYMENT=EMPLOYED BETWEEN 4 AND 7 YRS,LESS THAN 1 YR,UNEMPLOYED 13   2
##                    707) HISTORY=CRITICAL ACCOUNT,DUES DELAYED 27   8 YES (0.29629630 0.70370370) *
##                177) SAV_ACCT=BETWEEN 500 AND 1000,GREATER THAN EQUAL TO 1000,NO SAVINGS 41  12 YES (
##                  354) PRESENT_RESIDENT=BETWEEN 2 AND 3 YEARS,GREATER THAN EQUAL TO 2 YEARS 20  10 NO
##                    708) AMOUNT< 2128.5 8   0 NO (1.00000000 0.00000000) *
##                    709) AMOUNT>=2128.5 12   2 YES (0.16666667 0.83333333) *
##                  355) PRESENT_RESIDENT=LESS THAN 1 YEAR,MORE THAN 4 YEARS 21   2 YES (0.09523810 0.90
##               89) OTHER_PARTIES=GUARANTOR 13   1 YES (0.07692308 0.92307692) *
##             45) PURPOSE=RETRAINING,USED CAR 41   8 YES (0.19512195 0.80487805) *
##           23) DURATION< 11.5 54   9 YES (0.16666667 0.83333333) *
##      3) CHK_ACCT=GREATER THAN EQUAL TO 200,NO CHECKING ACCOUNT 310  42 YES (0.13548387 0.86451613) *
```

```
rpart.plot(myTreeprun)
```




```r
#Parameters to check reliability of the model
cf<-table(actual = test$RESPONSE, pred = pred_test)
cf
```

```
##       pred
## actual  NO YES
##    NO   47  42
##    YES  22 202
```

```r
accuracy73<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
precision73<-(cf[2,2]/(cf[2,2]+cf[1,2]))
recall73<-(cf[2,2]/(cf[2,2]+cf[2,1]))
accuracy73
```

```
## [1] 0.7955272
```

```r
precision73
```

```
## [1] 0.8278689
```

```r
recall73
```

```
## [1] 0.9017857
```

Output of Model 4 after pruning: The number of true positives and true negatives have increased after pruning. Also, we can see that the number of false positives have reduced in comparison to the previous model. From the nature of the data it is evident that the model should focus more on reducing the False positive as the risk associated with it is too high. This model 4(70-30) gives a good trade off of all the performance parameters with accuracy, precision, and recall as 79%, 82% and 90% respectively. This model performs better than the model 3. Hence, the model4 is the best with all the performance parameters showing best results of all the models.

#Implementing C5.0

We also tried to implement our models on different decision tree parameter like C5.0 A C5.0 model works by splitting the sample based on the field that provides the maximum information gain. Each sub-sample defined by the first split is then split again, usually based on a different field, and the process repeats until the subsamples cannot be split any further. Finally, the lowest-level splits are reexamined, and those that do not contribute significantly to the value of the model are removed or pruned. For its implementation, we have installed the library(C50). We implemented the C5.0 parameter on our models to see their performances.

```
#C5.0 on prob = c(0.5,0.5)
set.seed(7276)
ind<-sample(2, nrow(data_new), replace = T, prob = c(0.5,0.5))
ctrain<-data_new[ind==1,]
ctest<-data_new[ind==2,]

train.fit <- C5.0(ctrain[c(-1,-22)], factor(ctrain$RESPONSE))
train.fit
```

```
##
## Call:
## C5.0.default(x = ctrain[c(-1, -22)], y = factor(ctrain$RESPONSE))
##
## Classification Tree
## Number of samples: 515
## Number of predictors: 20
##
## Tree size: 54
##
## Non-standard options: attempt to group attributes
```

```
cpred.train <- predict(train.fit, ctrain)
mean(ctrain$RESPONSE!=cpred.train)
```

```
## [1] 0.08737864
```

```
cpred.test <- predict(train.fit, ctest)
print("Error Rate")
```

```
## [1] "Error Rate"
```

```r
mean(ctest$RESPONSE!=cpred.test)
```

```
## [1] 0.3010309
```

```r
cf<-table(actual = ctest$RESPONSE, pred = cpred.test)
cf
```

```
##        pred
## actual  NO YES
##    NO   66  87
##    YES  59 273
```

```r
c5accuracy55<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
c5precision55<-(cf[2,2]/(cf[2,2]+cf[1,2]))
c5recall55<-(cf[2,2]/(cf[2,2]+cf[2,1]))
print("Accuracy")
```

```
## [1] "Accuracy"
```

```r
c5accuracy55
```

```
## [1] 0.6989691
```

```r
print("Precision")
```

```
## [1] "Precision"
```

```r
c5precision55
```

```
## [1] 0.7583333
```

```r
print("Recall")
```

```
## [1] "Recall"
```

```r
c5recall55
```

```
## [1] 0.8222892
```

Output of the model: We can observe that although the error rate on the training data is very low (~9%), the error rate on the test data is almost ~30% which is very high. Also, on building the confusion matrix for the model, we can see that the model is providing us with false positive rate of ~24% (1-precision) which is not a desirable value.

```r
#c5.0 on prob = c(0.7,0.3)
set.seed(7576)
ind<-sample(2, nrow(data_new), replace = T, prob = c(0.7,0.3))
ctrain<-data_new[ind==1,]
ctest<-data_new[ind==2,]

train.fit <- C5.0(ctrain[c(-1,-22)], factor(ctrain$RESPONSE))
train.fit
```

```
##
## Call:
## C5.0.default(x = ctrain[c(-1, -22)], y = factor(ctrain$RESPONSE))
##
## Classification Tree
## Number of samples: 685
## Number of predictors: 20
##
## Tree size: 65
##
## Non-standard options: attempt to group attributes
```

```
cpred.train <- predict(train.fit, ctrain)
mean(ctrain$RESPONSE!=cpred.train)
```

```
## [1] 0.1021898
```

```
cpred.test <- predict(train.fit, ctest)
print("Error Rate")
```

```
## [1] "Error Rate"
```

```
mean(ctest$RESPONSE!=cpred.test)
```

```
## [1] 0.2984127
```

```
cf<-table(actual = ctest$RESPONSE, pred = cpred.test)
cf
```

```
##        pred
## actual  NO YES
##    NO   39  60
##    YES  34 182
```

```
c5accuracy73<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
c5precision73<-(cf[2,2]/(cf[2,2]+cf[1,2]))
c5recall73<-(cf[2,2]/(cf[2,2]+cf[2,1]))
print("Accuracy")
```

```
## [1] "Accuracy"
```

```
c5accuracy73
```

```
## [1] 0.7015873
```

```
print("Precision")
```

```
## [1] "Precision"
```

```
c5precision73
```

```
## [1] 0.7520661
```

```
print("Recall")
```

```
## [1] "Recall"
```

```
c5recall73
```

```
## [1] 0.8425926
```

Output of the model: We can observe that although the error rate on the training data is very low (~10%), the error rate on the test data is almost ~30% which is very high. Also, on building the confusion matrix for the model, we can see that the model is providing us with false positive rate of ~25% (1-precision) which is not a desirable value.

```
#c5.0 on prob = c(0.8,0.2)
set.seed(78876)
ind<-sample(2, nrow(data_new), replace = T, prob = c(0.8,0.2))
ctrain<-data_new[ind==1,]
ctest<-data_new[ind==2,]

train.fit <- C5.0(ctrain[c(-1,-22)], factor(ctrain$RESPONSE))
train.fit
```

```
##
## Call:
## C5.0.default(x = ctrain[c(-1, -22)], y = factor(ctrain$RESPONSE))
##
## Classification Tree
## Number of samples: 818
## Number of predictors: 20
##
## Tree size: 47
##
## Non-standard options: attempt to group attributes
```

```
cpred.train <- predict(train.fit, ctrain)
mean(ctrain$RESPONSE!=cpred.train)
```

```
## [1] 0.1308068
```

```
cpred.test <- predict(train.fit, ctest)
print("Error Rate")
```

```
## [1] "Error Rate"
```

```
mean(ctest$RESPONSE!=cpred.test)
```

```
## [1] 0.3296703
```

```
cf<-table(actual = ctest$RESPONSE, pred = cpred.test)
cf
```

```
##        pred
## actual  NO YES
##     NO  15  43
##     YES 17 107
```

```
c5accuracy82<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
c5precision82<-(cf[2,2]/(cf[2,2]+cf[1,2]))
c5recall82<-(cf[2,2]/(cf[2,2]+cf[2,1]))
print("Accuracy")
```

```
## [1] "Accuracy"
```

```
c5accuracy82
```

```
## [1] 0.6703297
```

```
print("Precision")
```

```
## [1] "Precision"
```

```
c5precision82
```

```
## [1] 0.7133333
```

```
print("Recall")
```

```
## [1] "Recall"
```

```
c5recall82
```

```
## [1] 0.8629032
```

Output of the model: We can observe that although the error rate on the training data is low (~13%), the error rate on the test data is almost ~33% which is very high. Also, on building the confusion matrix for the model, we can see that the model is providing us with false positive rate of ~29% (1-precision) which is not a desirable value.

```
#c5.0 on prob = c(0.75,0.25)
set.seed(7134)
ind<-sample(2, nrow(data_new), replace = T, prob = c(0.75,0.25))
ctrain<-data_new[ind==1,]
ctest<-data_new[ind==2,]

train.fit <- C5.0(ctrain[c(-1,-22)], factor(ctrain$RESPONSE))
train.fit
```

```
##
## Call:
## C5.0.default(x = ctrain[c(-1, -22)], y = factor(ctrain$RESPONSE))
##
## Classification Tree
## Number of samples: 759
## Number of predictors: 20
##
## Tree size: 65
##
## Non-standard options: attempt to group attributes
```

```
cpred.train <- predict(train.fit, ctrain)
print("Error Rate on Training")
```

```
## [1] "Error Rate on Training"
```

```
mean(ctrain$RESPONSE!=cpred.train)
```

```
## [1] 0.113307
```

```
cpred.test <- predict(train.fit, ctest)
print("Error Rate on test")
```

```
## [1] "Error Rate on test"
```

```
mean(ctest$RESPONSE!=cpred.test)
```

```
## [1] 0.2904564
```

```
cf<-table(actual = ctest$RESPONSE, pred = cpred.test)
cf
```

```
##       pred
## actual  NO YES
##    NO   29  52
##    YES  18 142
```

```
c5accuracy72<-(cf[2,2]+cf[1,1])/(cf[1,1]+cf[1,2]+cf[2,1]+cf[2,2])
c5precision72<-(cf[2,2]/(cf[2,2]+cf[1,2]))
c5recall72<-(cf[2,2]/(cf[2,2]+cf[2,1]))
print("Accuracy")
```

```
## [1] "Accuracy"
```

```
c5accuracy72
```

```
## [1] 0.7095436
```

```r
print("Precision")
```

```
## [1] "Precision"
```

```r
c5precision72
```

```
## [1] 0.7319588
```

```r
print("Recall")
```

```
## [1] "Recall"
```

```r
c5recall72
```

```
## [1] 0.8875
```

Output of the model: We can observe that although the error rate on the training data is low (~12%), the error rate on the test data is almost ~30% which is very high. Also, on building the confusion matrix for the model, we can see that the model is providing us with false positive rate of ~26% (1-precision) which is not a desirable value.

Summary:

We have calculated the Accuracy, Precision and Recall of all the trees (C&R and c5.0) that we have built and displayed them below

```r
cat("C&R Trees\n","50:50 Split\n","Accuracy","\t","Precision", "\t","Recall","\n",accuracy55,"\t",preci
```

```
## C&R Trees
##  50:50 Split
##  Accuracy     Precision   Recall
##  0.7203883    0.7573529   0.8728814
```

```r
print("70:30 Split")
```

```
## [1] "70:30 Split"
```

```r
cat("Accuracy","\t","Precision", "\t","Recall","\n",accuracy73,"\t",precision73,"\t",recall73,"\n")
```

```
## Accuracy      Precision   Recall
##  0.7955272    0.8278689   0.9017857
```

```r
print("80:20 Split")
```

```
## [1] "80:20 Split"
```

```
cat("Accuracy","\t","Precision", "\t","Recall","\n",accuracy82,"\t",precision82,"\t",recall82,"\n")
```

```
## Accuracy      Precision   Recall
##  0.7738693    0.8024691   0.9090909
```

```
print("75:25 Split")
```

```
## [1] "75:25 Split"
```

```
cat("Accuracy","\t","Precision", "\t","Recall","\n",accuracy72,"\t",precision72,"\t",recall72,"\n\n")
```

```
## Accuracy      Precision   Recall
##  0.7641921    0.8031915   0.8988095
```

```
cat("c5.0 Trees","\n","50:50 Split","\n","Accuracy","\t","Precision","\t","Recall","\n",c5accuracy55,"\t
```

```
## c5.0 Trees
##  50:50 Split
##  Accuracy     Precision   Recall
##  0.6989691    0.7583333   0.8222892
```

```
print("70:30 Split")
```

```
## [1] "70:30 Split"
```

```
cat("Accuracy","\t","Precision", "\t","Recall","\n",c5accuracy73,"\t",c5precision73,"\t",c5recall73,"\n
```

```
## Accuracy      Precision   Recall
##  0.7015873    0.7520661   0.8425926
```

```
print("80:20 Split")
```

```
## [1] "80:20 Split"
```

```
cat("Accuracy","\t","Precision", "\t","Recall","\n",c5accuracy82,"\t",c5precision82,"\t",c5recall82,"\n
```

```
## Accuracy      Precision   Recall
##  0.6703297    0.7133333   0.8629032
```

```
print("75:25 Split")
```

```
## [1] "75:25 Split"
```

```
cat("Accuracy","\t","Precision", "\t","Recall","\n",c5accuracy72,"\t",c5precision72,"\t",c5recall72,"\n
```

```
## Accuracy      Precision   Recall
##  0.7095436    0.7319588   0.8875
```

Based on the values of Accuracy, Precision and Recall of the CNR trees and the c5.0 trees, we can see that: 1. C&R trees are providing us with better accuracy, precision and recall in contrast to c5.0 trees. 2. From the C&R trees, we can also see that the best prediction values are being obtained by the tree with a 70:30 split.

Hence, taking into account all the values, we will be considering the tree with 70:30 split as our best decision model because we require high accuracy to obtain the best True Positive values and we also need high precision to ensure we have a low False Positive values.

==

Question c:

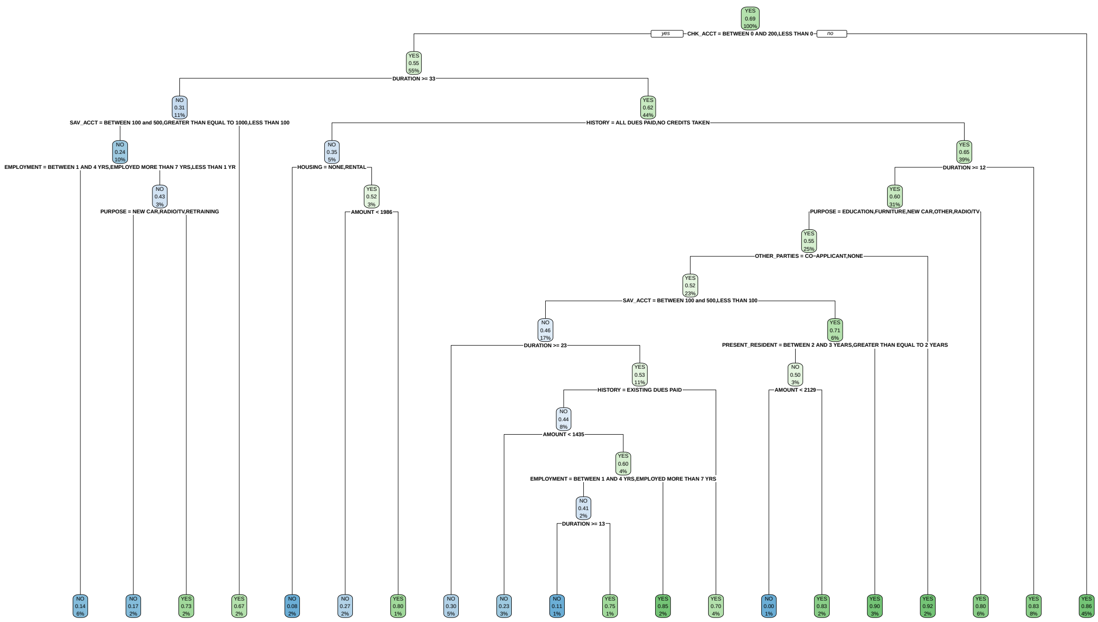
Based on the results from the above decision trees, the tree with the training and test split of 70:30 has shown us the best results. The 70:30 displays the lowest error rate when the minsplit value is set to 10, minbucket is set to 8 and cp is set to 0.01.

We will use this tree to check the consequences of mis-classification. First we will train our tree on the basis of the best minpslit and minbucket values obtained earlier. Then we will train another tree using the loss function to see the difference after adding the misclassification costs to our model. After this, we will make our predictions on the test data for both the trees.

```
# Training a model with best minsplit and minbucket values obtained from above analysis
myTree <- rpart(myFormula, data = train, control = rpart.control(minsplit = 10, minbucket = 8, cp = 0.0

# Performing prediction on the test data.
pred_test<-predict(myTree, newdata = test, type = "class")

# Bulding a confusion matrix for the actual response vs the predicted response.
pred_table <- table(actual = test$RESPONSE, pred = pred_test)
pred_table
```

```
##        pred
## actual  NO YES
##    NO   47  42
##    YES  22 202
```

```
# Plotting the tree which does not consider the misclassificaiton costs.
rpart.plot::rpart.plot(myTree)
```




```r
# Calculating the accuracy, precision and recall of the model.
accuracy<-(pred_table[2,2]+pred_table[1,1])/(pred_table[1,1]+pred_table[1,2]+pred_table[2,1]+pred_table
precision<-(pred_table[2,2]/(pred_table[2,2]+pred_table[1,2]))
recall<-(pred_table[2,2]/(pred_table[2,2]+pred_table[2,1]))

print("Without Loss function")
```

```
## [1] "Without Loss function"
```

```r
accuracy
```

```
## [1] 0.7955272
```

```r
precision
```

```
## [1] 0.8278689
```

```r
recall
```

```
## [1] 0.9017857
```

```
loss<-matrix(c(0,5,1,0), byrow = T, ncol = 2)
# Training a model with the same parameters as myTree but adding loss function to it.
misclassTree <- rpart(myFormula, data = train,method = "class", control = rpart.control(minsplit = 10, m

misclas_pred_train<-predict(misclassTree, train, type = "class")
mean(misclas_pred_train!=train$RESPONSE)
```

```
## [1] 0.3377001
```

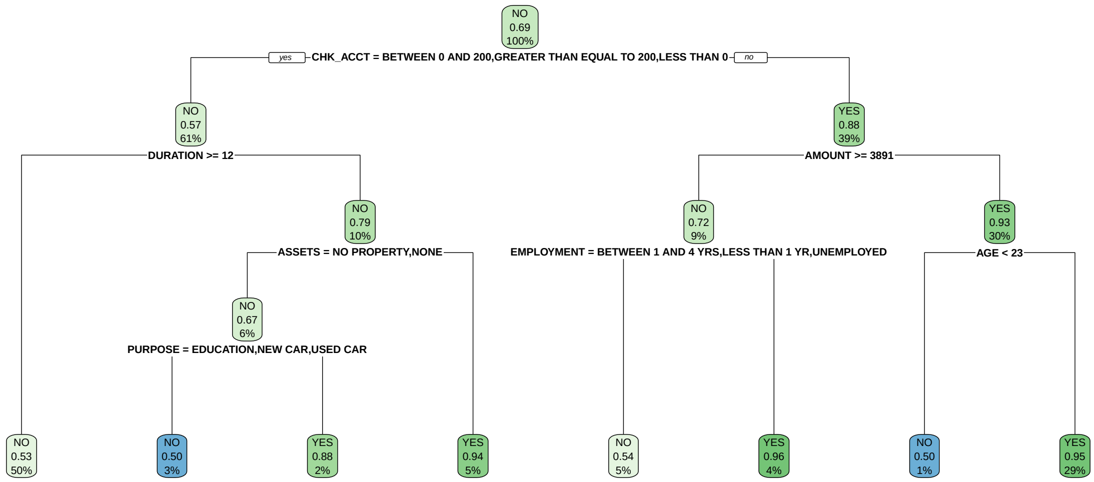
```
# Performing prediction using the tree using the loss function
misclas_pred_test <- predict(misclassTree, test, type ="class" )
mean(misclas_pred_test!=test$RESPONSE)
```

```
## [1] 0.3833866
```

```
# Building a confusion matrix for the actual response vs the predicted response.
misclas_table <- table(actual = test$RESPONSE, pred = misclas_pred_test)
misclas_table
```

```
##       pred
## actual  NO YES
##    NO   75  14
##    YES 106 118
```

```
# Plotting the tree which considers the misclassification costs.
rpart.plot::rpart.plot(misclassTree)
```




```
# Calculating the accuracy, precision and recall of the model
accuracy<-(misclas_table[2,2]+misclas_table[1,1])/(misclas_table[1,1]+misclas_table[1,2]+misclas_table[
precision<-(misclas_table[2,2]/(misclas_table[2,2]+misclas_table[1,2]))
recall<-(misclas_table[2,2]/(misclas_table[2,2]+misclas_table[2,1]))

print("With loss function")
```

```
## [1] "With loss function"
```

```
accuracy
```

```
## [1] 0.6166134
```

```
precision
```

```
## [1] 0.8939394
```

```
recall
```

```
## [1] 0.5267857
```

Output: After observing the predictions made by both models, we can see that the number of false positive values, i.e. predicted as yes even though actually bad has reduced significantly. However, another observation is that although the precision of the model has increased, the accuracy and recall has reduced significantly. We can also observe that after implementing the misclassification costs into the model, the tree size reduces significantly.

On calculating further, we found out that if we use the original mode, the misclassification costs incurred would be 23,200DM whereas, on the new model, the misclassification costs incurred would be 17600. So if the sole objective of the model is to ensure that the costs incurred is less, the new model would be a better choice. However if the requirement is to have higher accuracy, the original model would be a better option.

Accuracy of best models

TYPE	TRAINING	TEST
Model without misclassification cost (70:30)	84%	79.5%
Model with misclassification cost (70:30)	66.3%	61.6%
c5.0 (75:25)	88.7%	70.9%

==

Question d:

Decision rules from our best model are:

1. If Applicant has CHK_ACCT<0 OR CHK_ACCT>=200 OR 0<CHK_ACCT<200 and DURATION>=12 then the Applicant is not a good credit risk.
2. If Applicant has CHK_ACCT<0 OR CHK_ACCT>=200 OR 0<CHK_ACCT<200,DURATION<=12,ASSETS=NO PROPERTY/NONE and PURPOSE=EDUCATION/NEW CAR/USED CAR then the applicant is not a good credit risk.
3. If Applicant has CHK_ACCT<0 OR CHK_ACCT>=200 OR 0<CHK_ACCT<200,DURATION<=12,ASSETS=NO PROPERTY/NONE and PURPOSE=FURNITURE/OTHER/RADIO/TV then the applicant is a good credit risk.
4. If Applicant has CHK_ACCT<0 OR CHK_ACCT>=200 OR 0<CHK_ACCT<200,DURATION<=12,ASSETS=REA ESTATE then the applicant is a good applicant.
5. If Applicant has CHK_ACCT= NO CHECK ACCOUNT, AMOUNT>=3891, EMPLOYMENT<1 OR 1<EMPLOYMENT<4 OR UNEMPLOYED then the applicant is not a good credit risk.
6. If Applicant has CHK_ACCT= NO CHECK ACCOUNT, AMOUNT>=3891, 4<EMPLOYMENT<7 OR EMPLOYMENT>7 then the applicant is a good credit risk.
7. If Applicant has CHK_ACCT= NO CHECK ACCOUNT, AMOUNT<=3891, AGE<23 then the applicant is not a good credit risk.
8. If Applicant has CHK_ACCT= NO CHECK ACCOUNT, AMOUNT<=3891, AGE>23 then the applicant is a good credit risk.

Best decision rules for classifying GOOD APPLICANTS are:

1. If Applicant has CHK_ACCT<0 OR cHK_ACCT>=200 OR 0<cHK_ACCT<200,DURATION<=12,ASSETS=NO PROPERTY/NONE and PURPOSE=FURNITURE/OTHER/RADIO/TV then the applicant is a good credit risk.(CONFIDENCE=88%)(SUPPORT=(17/687)*100=2.47%)
2. If Applicant has CHK_ACCT<0 OR cHK_ACCT>=200 OR 0<cHK_ACCT<200,DURATION<=12,ASSETS=REAI ESTATE then the applicant is a good applicant.(CONFIDENCE=93%)(SUPPORT=(32/687)*100=4.65%)

3. If Applicant has CHK_ACCT= NO CHECK ACCOUNT, AMOUNT>=3891, 4<EMPLOYMENT<7 OR EMPLOYMENT>7 then the applicant is a good credit risk.(CONFIDENCE=96%)(SUPPORT=(26/287)*100)=3.7)
4. If Applicant has CHK_ACCT= NO CHECK ACCOUNT, AMOUNT<=3891, AGE>23 then the applicant is a good credit risk.(CONFIDENCE=94%)(SUPPORT=(201/687)*100=29.25%)

Although all the best decision rules to predict good applicants have low support parameter but the confidence of each is so high that we can consider all to be the best rules to predict with great precision.

==

Question e:

Exploratary Data Analysis helped us discover patterns and relations between target and other input variables. It helped us know which variables could be important variables to predict future data accurately.It also helped us find out outliers, summarize main characteristics within the data set. Creating models of different training and testing size helped us know the effect of training and testing size of data on the performance of the model. We found that just increasing training or just increaing testing size will not increase the performance parameters. It is important to find a correct tradeoff of training and testing data sizes to get the best performing model. We also learned that focusing on just one performance parameter will not always help. It is important to know what the model needs to predict efficiently and choose that model that satisfies the need. For example in our case we had to focus on limiting the False Positive predictions so it was more important to create a model having high precision.We also noted the effect of changing the control parameters(minsplit, minbucket, CP) on the decision tree options on the model performance. We noted how changing each control parameter affected the performance of the decision tree and chose the most optimal control parameters values to have the best model.