Data Science Assignment

## Introduction To Data

Kate, a manager at a financial institution has contacted you. She is asking for assistance in assessing the credit worthiness of future potential customers. She has a data set of 780 past loan customer cases, with 14 attributes for each case, including attributes such as financial standing, reason for the loan, employment, demographic information, foreign national, years residence in the district and the outcome/label variable Credit Standing - classifying each case as either a good loan or bad loan.  
The manager has 13 new customers, which she would like to know if she should consider them good or bad prospective loans.

## Data Details

Most of the attributes are self-explanatory; the name of some of the attributes are somewhat cumbersome but this is what you have been given; here are the further details of some of them:

Checking Acct - What level of regular checking account does the customer have –No acct, 0balance, low (balance), high (balance)

Credit History – All paid – no credit taken or all credit paid back duly, Bank Paid – All credit at this bank paid back Current – Existing loan/credit paid back duly till now, Critical – Risky account or other credits at other banks, Delay – Delay in paying back credit/loan in the past

Months Acct – The number of months the customer has an account with the bank.

## Reading the files

sheet1 <- read\_xlsx("C:\\College\\01\_Subjects\\DSA\\assignments\\assignment 1\\2.Credit\_Risk6\_final.xlsx", sheet = 1)  
sheet2 <- read\_xlsx("C:\\College\\01\_Subjects\\DSA\\assignments\\assignment 1\\2.Credit\_Risk6\_final.xlsx", sheet = 2)

### sheet1

## # A tibble: 6 x 13  
## ID `Checking Acct` `Credit History` `Loan Reason` `Savings Acct`  
## <dbl> <chr> <chr> <chr> <chr>   
## 1 781 No Acct All Paid Car New MedHigh   
## 2 782 Low Current Small Applia~ Low   
## 3 783 No Acct Current Small Applia~ Low   
## 4 784 High Current Business Low   
## 5 785 Low Current Small Applia~ Low   
## 6 786 Low Current Other No Acct   
## # ... with 8 more variables: Employment <chr>, `Personal Status` <chr>,  
## # Housing <chr>, `Job Type` <chr>, `Foreign National` <chr>, `Months  
## # since Checking Acct opened` <dbl>, `Residence Time` <dbl>, Age <dbl>

In sheet 1 (scoring data) we have 13 observation with 13 features.

### sheet2

## # A tibble: 6 x 14  
## ID `Checking Acct` `Credit History` `Loan Reason` `Savings Acct`  
## <dbl> <chr> <chr> <chr> <chr>   
## 1 1 No Acct All Paid Car New Low   
## 2 2 0Balance Current Car New Low   
## 3 3 0Balance Current Car New No Acct   
## 4 4 0Balance Current Furniture No Acct   
## 5 5 No Acct All Paid Small Applia~ No Acct   
## 6 6 Low Current Car New MedLow   
## # ... with 9 more variables: Employment <chr>, `Personal Status` <chr>,  
## # Housing <chr>, `Job Type` <chr>, `Foreign National` <chr>, `Months  
## # since Checking Acct opened` <dbl>, `Residence Time (In current  
## # district)` <dbl>, Age <dbl>, `Credit Standing` <chr>

Whereas In sheet 2 which is also our traning data, we have 780 observation and 14 column. Our column of interst is “credit\_standing” on which we are going to put our analysis.

assign data into different variable so we can manipulate the values without disturbing original data

data1 <- sheet1  
data2 <- sheet2

## Cleaning of Data

Some models are having problem in reading column name if they have special character in it, so we have to change them

names(data2) =str\_replace\_all(names(data2) , c(" " = "\_"))  
names(data2)

## [1] "ID"   
## [2] "Checking\_Acct"   
## [3] "Credit\_History"   
## [4] "Loan\_Reason"   
## [5] "Savings\_Acct"   
## [6] "Employment"   
## [7] "Personal\_Status"   
## [8] "Housing"   
## [9] "Job\_Type"   
## [10] "Foreign\_National"   
## [11] "Months\_since\_Checking\_Acct\_opened"   
## [12] "Residence\_Time\_(In\_current\_district)"  
## [13] "Age"   
## [14] "Credit\_Standing"

also we are changing the name of ‘Residence\_Time\_(In\_current\_district)’ column to ‘Residence\_Time’ because it has some special character like brackets

## [1] "Residence\_Time"

### Converting whole data into factors

Sometimes our model accepts the values as factors so, to remove the warning from models, its better to convert data to factor

## Classes 'tbl\_df', 'tbl' and 'data.frame': 780 obs. of 14 variables:  
## $ ID : num 1 2 3 4 5 6 7 8 9 10 ...  
## $ Checking\_Acct : Factor w/ 4 levels "0Balance","High",..: 4 1 1 1 4 3 1 3 1 4 ...  
## $ Credit\_History : Factor w/ 5 levels "All Paid","Bank Paid",..: 1 4 4 4 1 4 1 2 1 5 ...  
## $ Loan\_Reason : Factor w/ 10 levels "Business","Car New",..: 2 2 2 5 10 2 2 4 3 3 ...  
## $ Savings\_Acct : Factor w/ 5 levels "High","Low","MedHigh",..: 2 2 5 5 5 4 2 2 2 2 ...  
## $ Employment : Factor w/ 6 levels "Long","Medium",..: 2 4 1 1 1 6 1 2 1 4 ...  
## $ Personal\_Status : Factor w/ 3 levels "Divorced","Married",..: 3 1 1 NA 3 1 2 1 3 2 ...  
## $ Housing : Factor w/ 3 levels "Other","Own",..: 2 2 2 2 1 2 2 1 NA 3 ...  
## $ Job\_Type : Factor w/ 4 levels "Management","Skilled",..: 1 2 2 2 2 4 2 4 2 2 ...  
## $ Foreign\_National : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 1 2 1 2 2 ...  
## $ Months\_since\_Checking\_Acct\_opened: num 7 16 25 31 7 13 22 25 25 13 ...  
## $ Residence\_Time : num 3 2 2 4 4 2 3 4 4 4 ...  
## $ Age : num 44 28 28 30 35 22 29 33 62 40 ...  
## $ Credit\_Standing : Factor w/ 2 levels "Bad","Good": 2 1 1 2 2 2 2 2 2 2 ...

### To check null values in column

as.vector(lapply(data2, function(x)sum(is.na(x))))

## $ID  
## [1] 0  
##   
## $Checking\_Acct  
## [1] 0  
##   
## $Credit\_History  
## [1] 0  
##   
## $Loan\_Reason  
## [1] 0  
##   
## $Savings\_Acct  
## [1] 0  
##   
## $Employment  
## [1] 33  
##   
## $Personal\_Status  
## [1] 6  
##   
## $Housing  
## [1] 5  
##   
## $Job\_Type  
## [1] 0  
##   
## $Foreign\_National  
## [1] 0  
##   
## $Months\_since\_Checking\_Acct\_opened  
## [1] 0  
##   
## $Residence\_Time  
## [1] 0  
##   
## $Age  
## [1] 0  
##   
## $Credit\_Standing  
## [1] 0

We have 33 null values in employment column , 6 in personal status, 5 in Housing. Either we can remove these values or we can Impute these values in the data. 5 or 6 null values can be replaced easily on the basis of remaining most occured values but its not a good choice to replace 33 values on remaining values so we can remove from data. For now we are removing all the rows which are having null values

### Droping the null values

data2 <- data2 %>% drop\_na()  
data1 <- data1 %>% drop\_na()

We have removed the null values from the data, now we can again check the data if there are null values or not.

lapply(data2, function(x)sum(is.na(x)))

## $ID  
## [1] 0  
##   
## $Checking\_Acct  
## [1] 0  
##   
## $Credit\_History  
## [1] 0  
##   
## $Loan\_Reason  
## [1] 0  
##   
## $Savings\_Acct  
## [1] 0  
##   
## $Employment  
## [1] 0  
##   
## $Personal\_Status  
## [1] 0  
##   
## $Housing  
## [1] 0  
##   
## $Job\_Type  
## [1] 0  
##   
## $Foreign\_National  
## [1] 0  
##   
## $Months\_since\_Checking\_Acct\_opened  
## [1] 0  
##   
## $Residence\_Time  
## [1] 0  
##   
## $Age  
## [1] 0  
##   
## $Credit\_Standing  
## [1] 0

# (A) Exploratory Data Analysis (EDA)

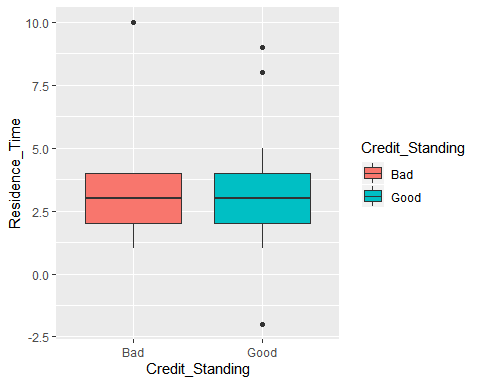
Gally library provide us the facility to draw all curve together w.r.t. each other.After that we can fatch the each curve using its index

curve <- ggpairs(data2[2:14],   
 mapping = ggplot2::aes(colour=Credit\_Standing),title = "gpairs curve",  
 lower=list(continuous=wrap("smooth", colour="blue")))

curve is the object which will save the each graph in it by its index. after giving column name we can fetch the indivisual graph

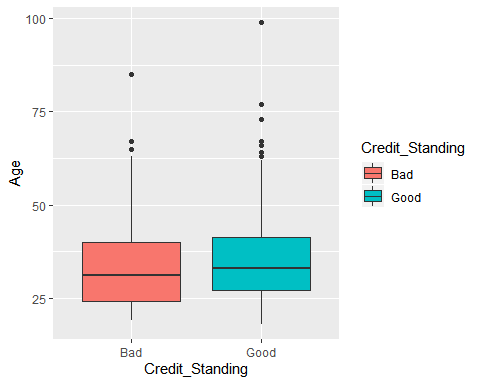
As we can see from the graph there are few outliers in both good and bad credit\_standing in checking account columns. We can easily remove them from the data, it might be outliers for this column but can play important role in the whole data as a combined predictors.

curve[11,13]



residence time column has also has 4 outliers 1 in bad and 3 in good category.

curve[12,13]

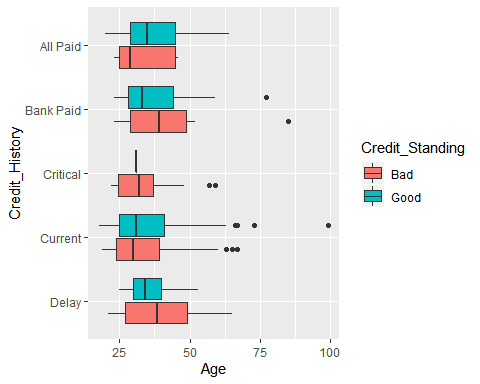


In comparison to other features Age is more diversified, it means age feature has more range of data, also g=has more no of outliers. Like residence time feature, age also have more outliers in good category.

### Trivariate curve

As we have defined our label as color in ggpair so we can visualize every feature In 2-d graph wrt its label. like we can see the relationship between ‘credit\_history’ and ‘age’ on the basis of good and bad

curve[2,12]



From this trivariate graph among (1)age along x-axis, (2)credit\_history along y-axis, and credit standing as color (orange color for Bad and green color for Good) Here one thing is to notice is those client who have credit\_history = critical are only belongs to Bad credit\_standing. There is only 1 client with id (400) who has credit\_history = critical and belongs to Good category.

# (B) Decision Tree Model

We can change the label value from Good to 1 and Bad to 0, so that it will become more logical to our model to understand the numeric values.

data2$Credit\_Standing <- ifelse(data2$Credit\_Standing == 'Good',1,0)  
head(data2$Credit\_Standing,10)

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

Setting the seed to 252

seed = 252  
set.seed(seed)

We are dividing data into train\_data and test\_data so that we can train the data and parallely we will be able to check the accuracy of the model using test\_data We are using caret library to divide the data into train and test with the ratio of 75%.

index <- createDataPartition(data2$Credit\_Standing, p=0.8, list=FALSE)  
train\_data <- data2[index,]  
dim(train\_data)

## [1] 590 14

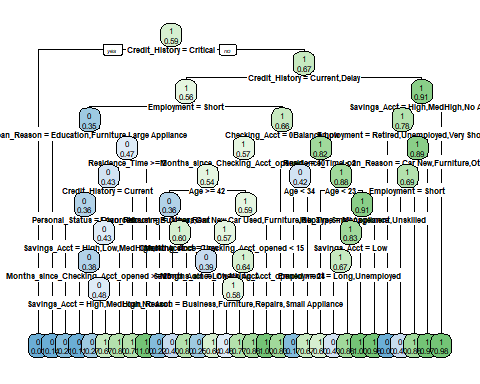
test\_data <- data2[-index,]  
dim(test\_data)

## [1] 147 14

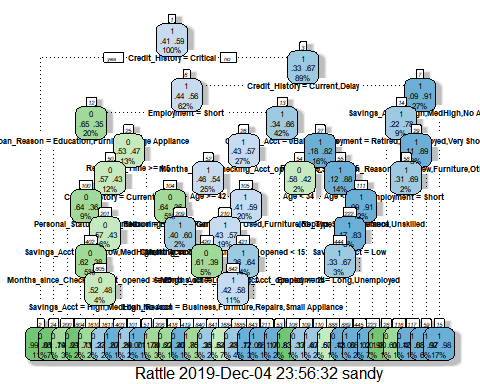
We are instructed to use decision tree to do our prediction therefore we are going to use rpart library to use import decision tree

fit <- rpart(Credit\_Standing~ .-Credit\_Standing-ID, data = train\_data,   
 method = "class", parms= list(split= 'information'),  
 control = rpart.control(cp = .001,minsplit = 5,minbucket = 5,maxdepth = 10, xval = 10))

rpart.plot(fit,cex= .5, extra=6)



fancyRpartPlot(fit, cex=.5)



### parameters which are used here are as follows:

method = class because it is my classification tree

parms= list(split= ‘information’)) because we want information gain or we can give gini as well, we can give annova if problem is regression

In control we have

cp = 0.2 complexity parameter, used to give constaint over overfitting, it tells you, what is the quality of split. it is the no. by which splting the node will decrease the relative error, value below .001 will not give the split. Low value of cp can overfit our model.

minsplit = 5 any node which has 5 no of observation will not go for further splitting

minbucket = 5 if terminal node has 5 observation it will not go for further split

maxdepth = 10 maximum depth of tree will not be higher than 10

xval = 10 no of cross validation

Note - here we are giving very low value of cp and other parameter to get full grown tree

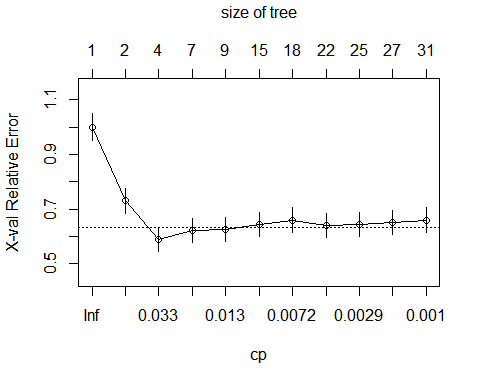
### Tunning of model

Now we will try to to prune the model rpart.plot gives you two functions

printcp(fit)

##   
## Classification tree:  
## rpart(formula = Credit\_Standing ~ . - Credit\_Standing - ID, data = train\_data,   
## method = "class", parms = list(split = "information"), control = rpart.control(cp = 0.001,   
## minsplit = 5, minbucket = 5, maxdepth = 10, xval = 10))  
##   
## Variables actually used in tree construction:  
## [1] Age Checking\_Acct   
## [3] Credit\_History Employment   
## [5] Housing Job\_Type   
## [7] Loan\_Reason Months\_since\_Checking\_Acct\_opened  
## [9] Personal\_Status Residence\_Time   
## [11] Savings\_Acct   
##   
## Root node error: 241/590 = 0.40847  
##   
## n= 590   
##   
## CP nsplit rel error xerror xstd  
## 1 0.2697095 0 1.00000 1.00000 0.049543  
## 2 0.0726141 1 0.73029 0.73029 0.046112  
## 3 0.0152144 3 0.58506 0.58921 0.043086  
## 4 0.0145228 6 0.53942 0.62241 0.043886  
## 5 0.0110650 8 0.51037 0.62656 0.043982  
## 6 0.0082988 14 0.43568 0.64315 0.044358  
## 7 0.0062241 17 0.41079 0.65975 0.044719  
## 8 0.0041494 21 0.38174 0.63900 0.044265  
## 9 0.0020747 24 0.36929 0.64315 0.044358  
## 10 0.0010373 26 0.36515 0.65145 0.044540  
## 11 0.0010000 30 0.36100 0.65975 0.044719

plotcp(fit)

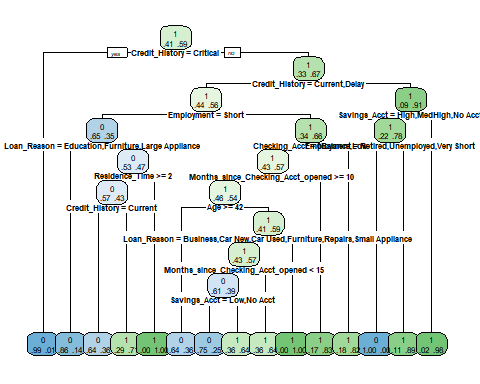


By looking at output of both parameter we can have the best value of cp print cp gives us tabular value and plotcp gives us plot after looking at the table of printcp we can say we have the lowest value of xerror = .58921 at cp = 0.0152144. But the standart way of selecting cp is get the minimim value of xerror and add its xstd value in it. Now select the maximum value of xerror which is less than this value. So in our case .58921 +0.0152144. = 0.632296 and now the maximum value of xerror which is less than 0.632296 is 0.62656 . for this respective value of cp is 0.0110650.

### Now prune the model with best value of cp which is 0.0110650

tree.fit <- prune(fit, cp =0.0110650)  
#is same as   
fit <- rpart(Credit\_Standing~ .-Credit\_Standing, data = train\_data,   
 method = "class", parms= list(split= 'information'),  
 control = rpart.control(cp = 0.0044053,minsplit = 5,minbucket = 5,maxdepth = 10, xval = 10))

rpart.plot(tree.fit,cex= .5, extra=4)



### Explanation of tree graph

value given above in box is majority class value below value in box are percentage of good and bad credit standing, majority class is given above the percentage so first of all is our root node 1, where 38% are good and 62% are in bad category. the first split is on the basis of (critical\_history = critical) is true, they all are bad credit standing, other will be further splitted Second node contains good credit-standing of 68% and bad is 32 %, it is splitted on the basis of (credit\_history == Current.delay) is true or not, and goes on.

### Check for the accuracy of model

we can test accuracy of model on test data

pred\_rpart <- predict(fit,test\_data, type="class")

type = class will give us output in class format either 0 or 1

caret::confusionMatrix(pred\_rpart, as.factor(test\_data$Credit\_Standing))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 45 21  
## 1 19 62  
##   
## Accuracy : 0.7279   
## 95% CI : (0.6484, 0.7979)  
## No Information Rate : 0.5646   
## P-Value [Acc > NIR] : 3.185e-05   
##   
## Kappa : 0.4485   
##   
## Mcnemar's Test P-Value : 0.8744   
##   
## Sensitivity : 0.7031   
## Specificity : 0.7470   
## Pos Pred Value : 0.6818   
## Neg Pred Value : 0.7654   
## Prevalence : 0.4354   
## Detection Rate : 0.3061   
## Detection Prevalence : 0.4490   
## Balanced Accuracy : 0.7251   
##   
## 'Positive' Class : 0   
##

accuracy of model = 72.79

Here, I found one best code to check which are the values going to the each node on stack overflow by MrFlick

tree = trained model

df = data frame used in modelling

nodes= nodes in which you want to check data

subset.rpart <- function (tree, df, nodes) {  
 if (!inherits(tree, "rpart"))   
 stop("Not a legitimate \"rpart\" object")  
 stopifnot(nrow(df)==length(tree$where))  
 frame <- tree$frame  
 n <- row.names(frame)  
 node <- as.numeric(n)  
   
 if (missing(nodes)) {  
 xy <- rpart:::rpartco(tree)  
 i <- identify(xy, n = 1L, plot = FALSE)  
 if(i> 0L) {  
 return( df[tree$where==i, ] )  
 } else {  
 return(df[0,])  
 }  
 }  
 else {  
 if (length(nodes <- rpart:::node.match(nodes, node)) == 0L)   
 return(df[0,])  
 return ( df[tree$where %in% as.numeric(nodes), ] )  
 }  
}

So if we want to get the data which is going to node 2 is as follows

subset.rpart(fit, train\_data, 2)

## # A tibble: 67 x 14  
## ID Checking\_Acct Credit\_History Loan\_Reason Savings\_Acct Employment  
## <dbl> <fct> <fct> <fct> <fct> <fct>   
## 1 11 Low Critical Furniture Low Very Short  
## 2 26 Low Critical Business MedLow Short   
## 3 27 0Balance Critical Education Low Short   
## 4 40 High Critical Car New MedHigh Medium   
## 5 45 0Balance Critical Furniture Low Short   
## 6 51 0Balance Critical Car Used Low Long   
## 7 61 0Balance Critical Small Appl~ Low Short   
## 8 62 High Critical Small Appl~ No Acct Short   
## 9 87 No Acct Critical Car New Low Medium   
## 10 121 0Balance Critical Car New Low Short   
## # ... with 57 more rows, and 8 more variables: Personal\_Status <fct>,  
## # Housing <fct>, Job\_Type <fct>, Foreign\_National <fct>,  
## # Months\_since\_Checking\_Acct\_opened <dbl>, Residence\_Time <dbl>,  
## # Age <dbl>, Credit\_Standing <dbl>

At the end we can also get the summary of our model summary(fit)

# (C) Prediction for scoring data

Using this model we can predict the credit Standig for the Scoring data set Before that we need to perform all the operation that we have performed on training data

names(data1) =str\_replace\_all(names(data1) , c(" " = "\_"))

Also we are changing the name of ‘Residence\_Time\_(In\_current\_district)’ column to ‘Residence\_Time’ because it has some special character like brackets

converting whole data into factors

data1[sapply(data1, is.character)] <- lapply(data1[sapply(data1, is.character)],   
 as.factor)  
str(data1)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 13 obs. of 13 variables:  
## $ ID : num 781 782 783 784 785 786 787 788 789 790 ...  
## $ Checking\_Acct : Factor w/ 4 levels "0Balance","High",..: 4 3 4 2 3 3 2 3 4 4 ...  
## $ Credit\_History : Factor w/ 3 levels "All Paid","Critical",..: 1 3 3 3 3 3 3 1 3 3 ...  
## $ Loan\_Reason : Factor w/ 6 levels "Business","Car New",..: 2 6 6 1 6 5 6 4 6 4 ...  
## $ Savings\_Acct : Factor w/ 4 levels "High","Low","MedHigh",..: 3 2 2 2 2 4 2 1 2 4 ...  
## $ Employment : Factor w/ 5 levels "Long","Medium",..: 3 2 5 2 2 4 1 5 5 2 ...  
## $ Personal\_Status : Factor w/ 3 levels "Divorced","Married",..: 3 3 1 3 1 3 3 1 2 1 ...  
## $ Housing : Factor w/ 3 levels "Other","Own",..: 3 3 2 3 2 1 2 2 2 3 ...  
## $ Job\_Type : Factor w/ 4 levels "Management","Skilled",..: 4 2 2 2 4 3 2 2 2 4 ...  
## $ Foreign\_National : Factor w/ 2 levels "No","Yes": 1 1 1 2 2 2 2 2 2 2 ...  
## $ Months\_since\_Checking\_Acct\_opened: num 11 37 13 16 9 49 37 12 19 16 ...  
## $ Residence\_Time : num 2 4 2 4 2 4 2 1 4 4 ...  
## $ Age : num 39 23 28 25 43 39 30 19 38 32 ...

To check null values in column

lapply(data1, function(x)sum(is.na(x)))

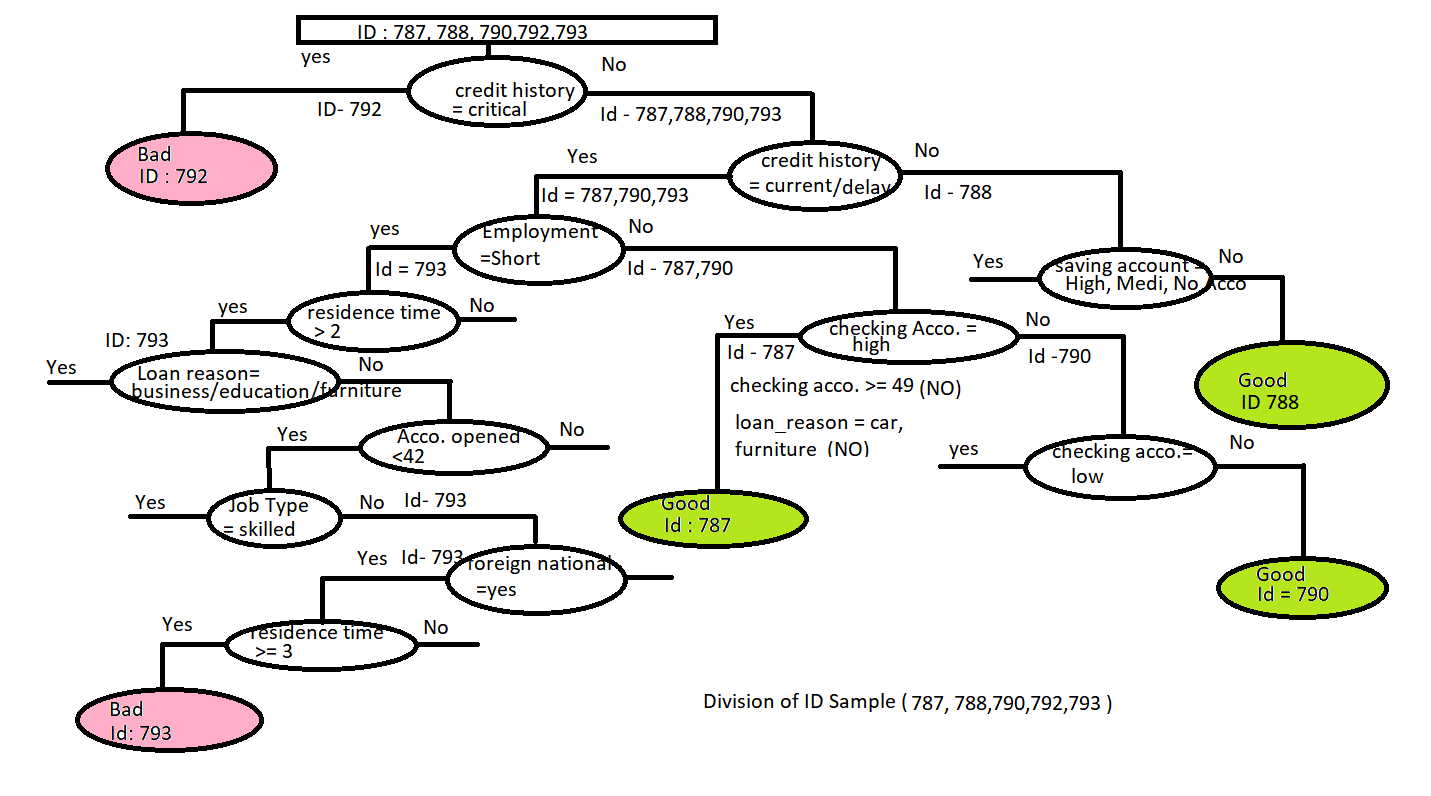
## $ID  
## [1] 0  
##   
## $Checking\_Acct  
## [1] 0  
##   
## $Credit\_History  
## [1] 0  
##   
## $Loan\_Reason  
## [1] 0  
##   
## $Savings\_Acct  
## [1] 0  
##   
## $Employment  
## [1] 0  
##   
## $Personal\_Status  
## [1] 0  
##   
## $Housing  
## [1] 0  
##   
## $Job\_Type  
## [1] 0  
##   
## $Foreign\_National  
## [1] 0  
##   
## $Months\_since\_Checking\_Acct\_opened  
## [1] 0  
##   
## $Residence\_Time  
## [1] 0  
##   
## $Age  
## [1] 0

Now we can use the old model we built for training our model

pred\_rpart <- predict(fit,data1, type="class")  
pred\_scoreing\_data <- ifelse(pred\_rpart == 1,'Good','Bad')  
data1$my\_pred <- pred\_scoreing\_data

### Selecting 5 potential customers rows from data

To show how our model is working, we are taking 5 clients with the id 787, 788, 790, 792, 793.



an image caption Source: decision tree for 5 selected clients

First of all, model checks whether our model has any client who has credit value is critical or not, if it is ‘yes’ then it will be classified directly “credit standing” as “Bad”. So, here our client with Id -792 has credit history = critical so it will be “Bad”.

Remaining Id - 787, 788, 790, 793 Here all remaining ids will go to no. Again our model will check for if credit history = current/delay. Ids 787, 790, 793 has current or delay as credit\_history. So, it will again check for the next condition. And the id 788 doesn’t have this value. It will again check for the next condition which is saving account = high/ medium or no account. Id 788 has nothing in it so it will again go to no. and it will be classified “credit standing” as a “Good”.

Remaining Id - 787, 790, 793 remaining ids will check for the next condition which is employment = short. Id 793 has employment = short but ids 787 and 793 will go for another condition. so, for now, we are taking id 793 which has employment = short. id 793 again will check for residence time > 2, In our case it is 3 so it will again go for next condition like loan reason = business or education then account open, job type, what is the foreign national and finally it will check for if my residence time is greater than or equal to 3 . Our Id- 793 has residence time = 3 so it will be classified “credit standing” as “bad”.

Remaining Ids - 787, 790 remaining Id will check for the next condition checking account is high or not. Id 787 has checking account= high. Id 787 will check for some more new condition according to our tree-like loan reasons = no or not. So, Id 787 will be classified “credit standing” as “Good”, after checking all the conditions.

Remaining Id 790 Id 790 will check for another condition that is checking account = low. our Id has checking acco = no account so it will be classified “credit standing” as “Good”.

So, using our model we have reached to final result which is as follows- Id -792 = “Bad” Id -788 = “Good” Id -793 = “Bad” Id -787 = “Good” Id -790 = “Good”

In this way our model is predicting all the values correctly for the following ids.

# (D) Trying Two ensamble models

### as an ensamble method we are using random forest here

here we are dividing the data into features and label

x <- train\_data[,1:12]  
y <- train\_data$Credit\_Standing

Direct from the help page for the randomForest() function in R: mtry: Number of variables randomly sampled as candidates at each split. ntree: Number of trees to grow. In caret only mtry parameter is available for tuning. because its effect on the final accuracy and that it must be found empirically for a dataset. We can choose the value of ntree whatever we want, upto some point it helps in incresing the accuracy.

We will create a base model for comperison using some defaults parameters in range where mtry=floor(sqrt(ncol(x))) or mtry=3 and ntree=500.

number = 10 which is no of folds repeats = 3 whcih tells us that it will be repeted by 3 times 10 folds and 3 repetations will take time but it will reduce overfitting of model

### 1 Create model with default paramters

control <- trainControl(method="repeatedcv", number=10, repeats=3)  
seed <- 252  
metric <- "Accuracy"  
set.seed(seed)  
mtry <- sqrt(ncol(x))  
tunegrid <- expand.grid(.mtry=mtry)  
rf\_default <- train(as.factor(Credit\_Standing)~.-Credit\_Standing, data=train\_data, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)  
print(rf\_default)

## Random Forest   
##   
## 590 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 531, 531, 531, 531, 531, 530, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.7469261 0.4596298  
##   
## Tuning parameter 'mtry' was held constant at a value of 3.464102

accuracy = 0.7469261 in initial

Predicting values for testing set

pred\_rf\_default <- predict(rf\_default, test\_data, mtry= 3)  
confusionMatrix(as.factor(pred\_rf\_default), as.factor(test\_data$Credit\_Standing))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 37 11  
## 1 27 72  
##   
## Accuracy : 0.7415   
## 95% CI : (0.6629, 0.8101)  
## No Information Rate : 0.5646   
## P-Value [Acc > NIR] : 6.716e-06   
##   
## Kappa : 0.4587   
##   
## Mcnemar's Test P-Value : 0.01496   
##   
## Sensitivity : 0.5781   
## Specificity : 0.8675   
## Pos Pred Value : 0.7708   
## Neg Pred Value : 0.7273   
## Prevalence : 0.4354   
## Detection Rate : 0.2517   
## Detection Prevalence : 0.3265   
## Balanced Accuracy : 0.7228   
##   
## 'Positive' Class : 0   
##

Accuracy : 0.7415

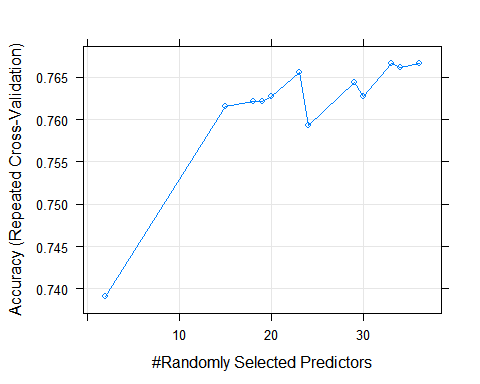
### 2 Random Search

In random search we will try to put some random values of mtry (search=“random”) in control param

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="random")  
set.seed(seed)  
metric <- "Accuracy"  
mtry <- sqrt(ncol(x))  
rf\_random <- train(as.factor(Credit\_Standing)~.-Credit\_Standing, data=train\_data, method="rf", metric=metric, tuneLength=15, trControl=control)  
print(rf\_random)

## Random Forest   
##   
## 590 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 531, 531, 531, 531, 531, 530, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.7390340 0.4258001  
## 15 0.7615767 0.4954894  
## 18 0.7621420 0.4962637  
## 19 0.7621423 0.4962703  
## 20 0.7627554 0.4981565  
## 23 0.7655705 0.5040901  
## 24 0.7593175 0.4910987  
## 29 0.7644405 0.5031396  
## 30 0.7627547 0.4999177  
## 33 0.7667008 0.5084590  
## 34 0.7661166 0.5065800  
## 36 0.7666725 0.5080151  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 33.

plot(rf\_random)



Best accuracy is 0.7667008 at mtry = 33

Accuracy on testing data

pred\_rf\_random <- predict(rf\_random, test\_data, mtry= 33)  
confusionMatrix(as.factor(pred\_rf\_random), as.factor(test\_data$Credit\_Standing))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 43 17  
## 1 21 66  
##   
## Accuracy : 0.7415   
## 95% CI : (0.6629, 0.8101)  
## No Information Rate : 0.5646   
## P-Value [Acc > NIR] : 6.716e-06   
##   
## Kappa : 0.4704   
##   
## Mcnemar's Test P-Value : 0.6265   
##   
## Sensitivity : 0.6719   
## Specificity : 0.7952   
## Pos Pred Value : 0.7167   
## Neg Pred Value : 0.7586   
## Prevalence : 0.4354   
## Detection Rate : 0.2925   
## Detection Prevalence : 0.4082   
## Balanced Accuracy : 0.7335   
##   
## 'Positive' Class : 0   
##

Accuracy : 0.7415

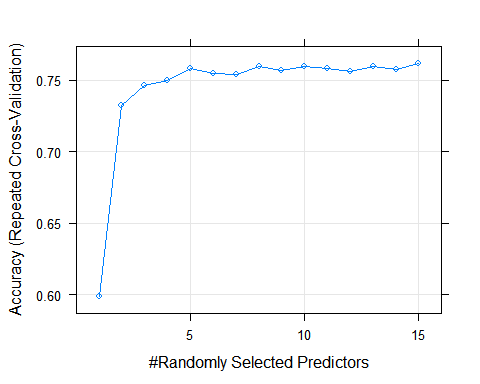
### 3 grid search

Another search that we have is grid. Grid is the combination of parameters and each axis defines a set of parameters to feed in algo. But here we are having only one value to tune so it will be a set of linear vector (mtry=c(1:15)).

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")  
set.seed(seed)  
metric <- "Accuracy"  
tunegrid <- expand.grid(.mtry=c(1:15))  
rf\_gridsearch <- train(as.factor(Credit\_Standing)~.-Credit\_Standing, data=train\_data, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)  
print(rf\_gridsearch)

## Random Forest   
##   
## 590 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 531, 531, 531, 531, 531, 530, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 1 0.5983298 0.01945151  
## 2 0.7327719 0.41333609  
## 3 0.7463998 0.45880059  
## 4 0.7497701 0.46852255  
## 5 0.7581681 0.48850112  
## 6 0.7547782 0.48163645  
## 7 0.7542513 0.47985568  
## 8 0.7598919 0.49250687  
## 9 0.7570381 0.48509735  
## 10 0.7598730 0.49191930  
## 11 0.7581583 0.48744085  
## 12 0.7565014 0.48503004  
## 13 0.7598916 0.49035257  
## 14 0.7576024 0.48625367  
## 15 0.7621709 0.49680268  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 15.

plot(rf\_gridsearch)



Best accuracy is 0.7621709 at mtry = 15

Accuracy on testing data

pred\_rf\_gridsearch <- predict(rf\_gridsearch, test\_data, mtry= 15)  
confusionMatrix(as.factor(pred\_rf\_gridsearch), as.factor(test\_data$Credit\_Standing))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 40 15  
## 1 24 68  
##   
## Accuracy : 0.7347   
## 95% CI : (0.6556, 0.804)  
## No Information Rate : 0.5646   
## P-Value [Acc > NIR] : 1.487e-05   
##   
## Kappa : 0.4515   
##   
## Mcnemar's Test P-Value : 0.2002   
##   
## Sensitivity : 0.6250   
## Specificity : 0.8193   
## Pos Pred Value : 0.7273   
## Neg Pred Value : 0.7391   
## Prevalence : 0.4354   
## Detection Rate : 0.2721   
## Detection Prevalence : 0.3741   
## Balanced Accuracy : 0.7221   
##   
## 'Positive' Class : 0   
##

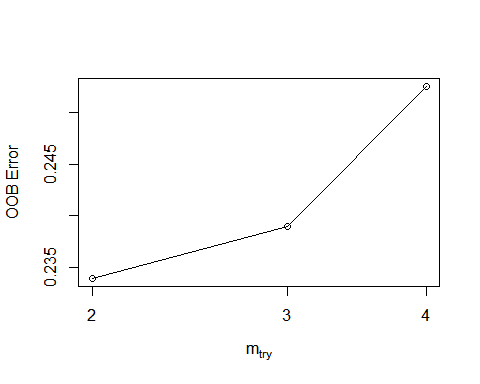
Accuracy : 0.7347

### 4 Algorithm Tune (tuneRF)

Some of algorithm provide method to tune its parameters like random forest we have tuneRF

set.seed(seed)  
bestmtry <- tuneRF(x, as.factor(train\_data$Credit\_Standing), stepFactor=1.5, improve=1e-5, ntree=500)

## mtry = 3 OOB error = 23.9%   
## Searching left ...  
## mtry = 2 OOB error = 23.39%   
## 0.0212766 1e-05   
## Searching right ...  
## mtry = 4 OOB error = 25.25%   
## -0.07971014 1e-05



print(bestmtry)

## mtry OOBError  
## 2.OOB 2 0.2338983  
## 3.OOB 3 0.2389831  
## 4.OOB 4 0.2525424

The most accurate value for mtry is 2 with an OOBError of 0.2338983. But above in grid search result we have seen that for mtry = 2 we were getting only 0.7327719. But yes it is another method to tune the model

### Boosting model (gredient boosting)

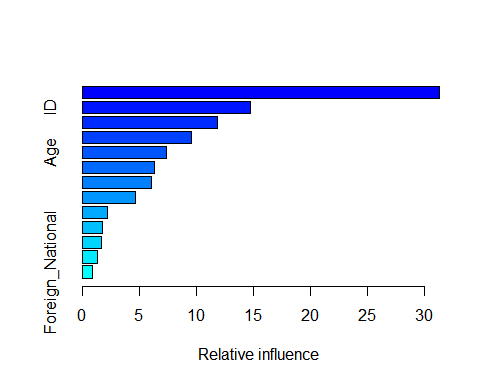
library(gbm)

## Loaded gbm 2.1.5

set.seed(seed)  
model\_boost = gbm(Credit\_Standing~.-Credit\_Standing,data = train\_data,   
 distribution = "bernoulli",n.trees = 1000,  
 shrinkage = 0.01, interaction.depth = 4)  
  
print(model\_boost)

## gbm(formula = Credit\_Standing ~ . - Credit\_Standing, distribution = "bernoulli",   
## data = train\_data, n.trees = 1000, interaction.depth = 4,   
## shrinkage = 0.01)  
## A gradient boosted model with bernoulli loss function.  
## 1000 iterations were performed.  
## There were 13 predictors of which 13 had non-zero influence.

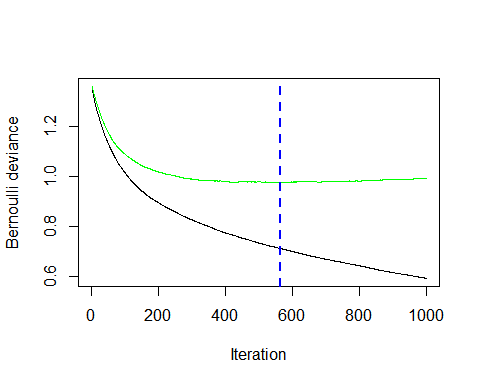
summary(model\_boost)



## var  
## Credit\_History Credit\_History  
## ID ID  
## Employment Employment  
## Loan\_Reason Loan\_Reason  
## Age Age  
## Checking\_Acct Checking\_Acct  
## Savings\_Acct Savings\_Acct  
## Months\_since\_Checking\_Acct\_opened Months\_since\_Checking\_Acct\_opened  
## Residence\_Time Residence\_Time  
## Personal\_Status Personal\_Status  
## Job\_Type Job\_Type  
## Housing Housing  
## Foreign\_National Foreign\_National  
## rel.inf  
## Credit\_History 31.3664660  
## ID 14.7976012  
## Employment 11.8648041  
## Loan\_Reason 9.6020434  
## Age 7.4322704  
## Checking\_Acct 6.3547730  
## Savings\_Acct 6.1126092  
## Months\_since\_Checking\_Acct\_opened 4.6344700  
## Residence\_Time 2.2015757  
## Personal\_Status 1.7708200  
## Job\_Type 1.6554309  
## Housing 1.3183098  
## Foreign\_National 0.8888264

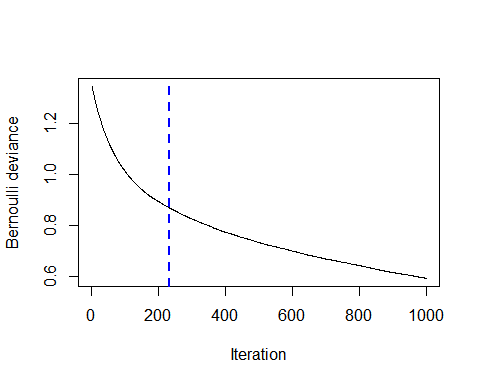
Tuning a gbm Model and Early Stopping

model\_boost = gbm(Credit\_Standing~.-Credit\_Standing,data = train\_data,   
 distribution = "bernoulli",n.trees = 1000,  
 shrinkage = 0.01, interaction.depth = 4,  
 cv.folds = 3)  
  
ntree\_opt\_cv <- gbm.perf(model\_boost, method = "cv")



ntree\_opt\_oob <- gbm.perf(model\_boost, method = "OOB")

## OOB generally underestimates the optimal number of iterations although predictive performance is reasonably competitive. Using cv\_folds>1 when calling gbm usually results in improved predictive performance.



print(ntree\_opt\_cv)

## [1] 563

print(ntree\_opt\_oob)

## [1] 231  
## attr(,"smoother")  
## Call:  
## loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,   
## length(x)/10), 50))  
##   
## Number of Observations: 1000   
## Equivalent Number of Parameters: 40   
## Residual Standard Error: 0.0001329

Optimum number of trees as per the “OOB” and “cv” method

ntree\_opt\_cv = 563

ntree\_opt\_oob = 231

Accuracy on testing data

prediction <- predict(model\_boost, test\_data,  
 n.trees = ntree\_opt\_cv,  
 type = "response")  
  
predictions <- as.factor(ifelse(prediction>.5,1,0))  
confusionMatrix(predictions,as.factor(test\_data$Credit\_Standing))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 39 16  
## 1 25 67  
##   
## Accuracy : 0.7211   
## 95% CI : (0.6412, 0.7918)  
## No Information Rate : 0.5646   
## P-Value [Acc > NIR] : 6.602e-05   
##   
## Kappa : 0.4234   
##   
## Mcnemar's Test P-Value : 0.2115   
##   
## Sensitivity : 0.6094   
## Specificity : 0.8072   
## Pos Pred Value : 0.7091   
## Neg Pred Value : 0.7283   
## Prevalence : 0.4354   
## Detection Rate : 0.2653   
## Detection Prevalence : 0.3741   
## Balanced Accuracy : 0.7083   
##   
## 'Positive' Class : 0   
##

Accuracy : 0.7211

# (E) Outliers

To find the pattern which may be recorded as wrong , our first approch is we can check the actual label with our predicted values if there are are wrong prediction , there may be a case that data is worng so we can say that there must be some problem with data that is why we are getting wrong prediction.

Here we are using our decision tree model for the prediction.

prediction = predict(fit, data2, type = "class")  
pred <- as.data.frame(prediction)  
index <- which(pred$prediction == data2$Credit\_Standing)  
data4 <- data2[-index,]  
data4$ID

## [1] 3 14 23 36 37 41 48 58 66 67 72 130 134 138 139 160 164  
## [18] 181 186 189 198 200 212 224 225 227 241 244 248 264 267 273 275 282  
## [35] 287 288 296 300 301 309 310 311 316 320 324 332 342 345 351 357 358  
## [52] 363 365 369 370 381 383 389 400 419 454 457 460 491 501 505 507 509  
## [69] 512 515 536 567 573 580 581 582 585 591 595 611 618 620 625 626 627  
## [86] 631 632 633 635 637 640 643 646 648 649 654 660 661 665 666 685 693  
## [103] 696 698 699 701 703 704 705 724 726 728 729 737 744 745 752 755 756  
## [120] 757 770 772

After getting the rows where we are getting wrong prediction,, we are recording the IDs of the observation. So that we can further analyse the data which will work as a subset for us.

After analysing the data, we came to know there are so many datapoints where all the features are same for an observation but the label which is written is different. It is clear that for the same features our prediction can not be different. Some observation we have observed are here-

207= 715,  
295= 700,  
296= 699,  
298= 309,  
299= 698,  
300= 577,  
302= 562,  
303= 502,  
304= 501,  
305= 498,  
306= 496,  
309= 298,  
310= 291,  
163= 729

ID which are mentioned here are having same features and different label. for example observation ID = 207 has same feature as ID= 715 but different label.

One thing which we can notice here is The id on left hand side are consucutive.If we observe them with keen eyes we will find the rows are between 295 to 310. It is suspicious that during a particular time that this process performed very poorly and produced inaccurate results.

we also observed here is there are multiple rows which are common here or repeated .

data3 <- sheet2[duplicated(sheet2[2:14]),]  
head(data3)

## # A tibble: 6 x 14  
## ID `Checking Acct` `Credit History` `Loan Reason` `Savings Acct`  
## <dbl> <chr> <chr> <chr> <chr>   
## 1 314 Low Current Business MedLow   
## 2 317 Low Current Car New Low   
## 3 321 No Acct Current Business High   
## 4 322 0Balance Current Car Used No Acct   
## 5 329 Low Current Furniture Low   
## 6 332 0Balance Current Car New Low   
## # ... with 9 more variables: Employment <chr>, `Personal Status` <chr>,  
## # Housing <chr>, `Job Type` <chr>, `Foreign National` <chr>, `Months  
## # since Checking Acct opened` <dbl>, `Residence Time (In current  
## # district)` <dbl>, Age <dbl>, `Credit Standing` <chr>

79= 431, 83= 430, 129= 359, 147= 427, 175= 350, 194= 426, 218= 423, 250= 528, 274= 703, 282= 333, 490= 697, 613= 663, 247= 315, 191= 387, 307= 489, 553= 719

So here ID 79 and ID 431 has repetative data and so on.