Loan Prediction Problem

## **Problem Statement**

#### **About Company**

Dream Housing Finance company deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan.

#### **Problem**

Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers. Here they have provided a partial data set.

## **Data**

|  |  |  |
| --- | --- | --- |
| **#** | **Variable** | **Description** |
| **1.** | **Loan\_ID** | Unique Loan ID |
| **2.** | **Gender** | Male/ Female |
| **3.** | **Married** | Applicant married (Y/N) |
| **4.** | **Dependents** | Number of dependents |
| **5.** | **Education** | Applicant Education (Graduate/ Under Graduate) |
| **6.** | **Self\_Employed** | Self employed (Y/N) |
| **7.** | **ApplicantIncome** | Applicant income |
| **8.** | **CoapplicantIncome** | Coapplicant income |
| **9.** | **LoanAmount** | Loan amount in thousands |
| **10.** | **Loan\_Amount\_Term** | Term of loan in months |
| **11.** | **Credit\_History** | credit history meets guidelines |
| **12.** | **Property\_Area** | Urban/ Semi Urban/ Rural |
| **13.** | **Loan\_Status** | Loan approved (Y/N) |

### Structured Analysis Planning

### The first thing we need to do and before jumping to analyze the data is to understand the problem statement and create a S.M.A.R.T objective. The next step is to identify our independent variables and our dependent variable. The below mind map illustrates the process I have conducted to structure plan the project.

### C:\Users\USER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\SPAP2.png

### Checking the data

### The next step is to look at the data we’re working with. Realistically, most of the data we will get, even from the government, can have errors, and it’s important to identify these errors before spending time analyzing the data.

### Normally, we must answer the following questions:

### Do we find something wrong in the data?

### Are there ambiguous variables in the dataset?

### Are there variables that should be fixed or removed?

### Let’s start by reading the data using the function read.csv() and show the first part of the dataset:

### tr <- read.csv('train.csv', header = TRUE)

### head(tr)

## The first row in the dataset defines the column header. Each of these headers is described in the above table. Now, we will run the summary function to have a quick look on the stats:

### summary(tr)

### Below are few conclusions we can infer by looking at the output:

### There are (+) sign on 51 record

### The mean of Credit\_history variable is 0.8422. That’s weird knowing that this variable has value of 1 for customers who have credit history and 0 otherwise.

### There are blank fields in Gender, Married, Dependents and Self\_Employed.

### There are NAs in LoanAmount, Loan\_Amount\_term and Credit\_History.

### Let’s do now a quick fix on some of the variables:

### tr <- read.csv(file="train.csv", na.strings=c("", "NA"), header=TRUE) # Convert blank fields to NA

### library(plyr)

### tr$Dependents <- revalue(tr$Dependents, c("3+"="3")) # Fix the special character in Dependents

### Now, let’s have a closer look at the missing data:

### sapply(tr, function(x) sum(is.na(x)))

### library(mice)

### library(VIM)

### mice\_plot <- aggr(tr, col=c('navyblue','red'),

### numbers=TRUE, sortVars=TRUE,

### labels=names(tr), cex.axis=.7,

### gap=3, ylab=c("Missing data","Pattern"))

### C:\Users\USER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Missing Values.png

From the chart and the table, there are seven variables that have missing data.

Now, it’s the time to give a look at the distribution of the data. We will start with the numerical variables: Loan Amount and ApplicantIncome:

Below are the histograms and the boxplots of the loan amount and the applicant income variables:

**par(mfrow=c(2,2))**

**hist(tr$LoanAmount,**

**main="Histogram for LoanAmount",**

**xlab="Loan Amount",**

**border="blue",**

**col="maroon",**

**las=1,**

**breaks=20, prob = TRUE)**

**#lines(density(tr$LoanAmount), col='black', lwd=3)**

**boxplot(tr$LoanAmount, col='maroon',xlab = 'LoanAmount', main = 'Box Plot for Loan Amount')**

**hist(tr$ApplicantIncome,**

**main="Histogram for Applicant Income",**

**xlab="Income",**

**border="blue",**

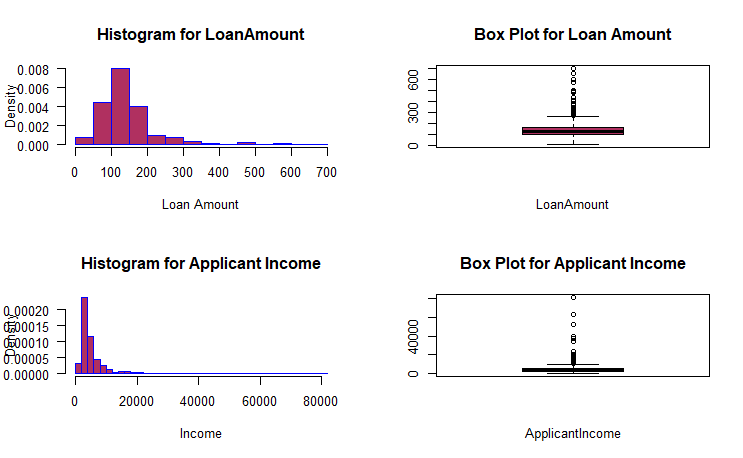
**col="maroon",**

**las=1,**

**breaks=50, prob = TRUE)**

**#lines(density(tr$ApplicantIncome), col='black', lwd=3)**

**boxplot(tr$ApplicantIncome, col='maroon',xlab = 'ApplicantIncome', main = 'Box Plot for Applicant Income')**

**dev.off()**

Here we notice that there are few extreme values in both variables. Let’s also examine if the applicant’ loan amounts distribution is affected by their educational level:

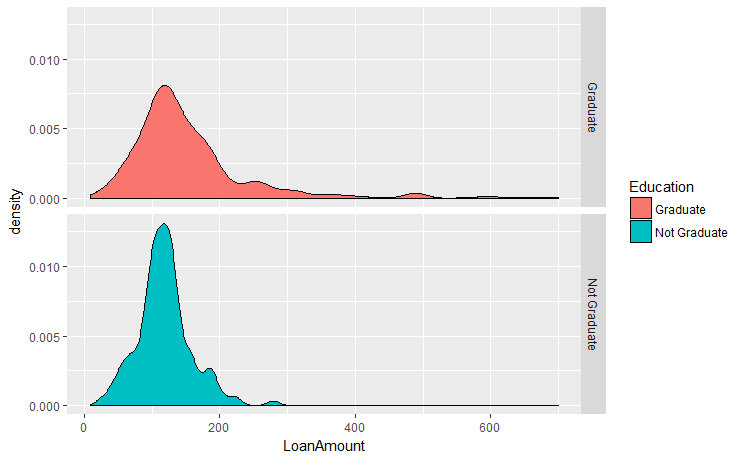
**library(ggplot2)**

**data(tr, package="lattice")**

**ggplot(data=tr, aes(x=LoanAmount, fill=Education)) +**

**geom\_density() +**

**facet\_grid(Education~.)**



We note that graduates have more outliers and their loan amount distribution is wider.

Now let’s give a look at the categorical variables in the dataset:

**## Visualizing Categorical Variables**

**par(mfrow=c(2,3))**

**counts <- table(tr$Loan\_Status, tr$Gender)**

**barplot(counts, main="Loan Status by Gender",**

**xlab="Gender", col=c("darkgrey","maroon"),**

**legend = rownames(counts))**

**counts2 <- table(tr$Loan\_Status, tr$Education)**

**barplot(counts2, main="Loan Status by Education",**

**xlab="Education", col=c("darkgrey","maroon"),**

**legend = rownames(counts2))**

**counts3 <- table(tr$Loan\_Status, tr$Married)**

**barplot(counts3, main="Loan Status by Married",**

**xlab="Married", col=c("darkgrey","maroon"),**

**legend = rownames(counts3))**

**counts4 <- table(tr$Loan\_Status, tr$Self\_Employed)**

**barplot(counts4, main="Loan Status by Self Employed",**

**xlab="Self\_Employed", col=c("darkgrey","maroon"),**

**legend = rownames(counts4))**

**counts5 <- table(tr$Loan\_Status, tr$Property\_Area)**

**barplot(counts5, main="Loan Status by Property\_Area",**

**xlab="Property\_Area", col=c("darkgrey","maroon"),**

**legend = rownames(counts5))**

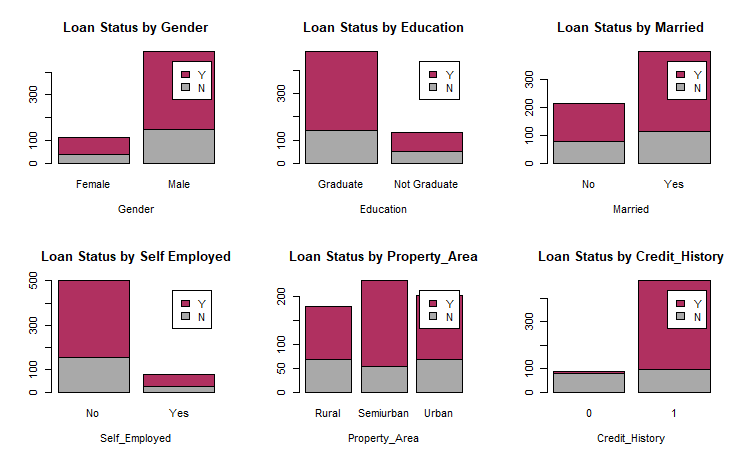
**counts6 <- table(tr$Loan\_Status, tr$Credit\_History)**

**barplot(counts6, main="Loan Status by Credit\_History",**

**xlab="Credit\_History", col=c("darkgrey","maroon"),**

**legend = rownames(counts5))**

**dev.off()**



If we look at the Gender graph, we note that males have more records and more than half of the applicants’ applications have been approved. There are less female applicants but still more than half of their applications have been approved. We look at the other charts with the same eye to evaluate how each category performed in regards to the approval of the loan applications.

### Tidying the data

Now that we’ve identified several errors in the data set, we need to fix them before we continue with our analysis. Let’s review the issues:

* There are missing values in some variables. Based on the importance of the variables, we will decide on the method to use.
* Looking at the distributions of the data, we noticed that ApplicantIncome and LoanAmount have outliers.

Fixing outliers can be tricky. It’s hard to tell if they were caused by measurement error, errors while recording, or if the outliers are real anomaly. If we decide to remove records, we have to document the reason behind this decision.

In this data set, we will assume that missing values are systematic because the missing data are coming in certain variables in a random manner. Also, we note that missing values are on both numerical and categorical data, therefore, we will be using the mice package in R. This package helps in imputing missing values with plausible data values. These values are inferred from a distribution that is designed for each missing data point. In the missing data plot above, we note that 0.78 of the data are not missing any information, 0.07 are missing the Credit\_History value, and the remaining ones show other missing patterns.

The mice() function takes care of the imputing process:

**imputed\_Data <- mice(tr, m=2, maxit = 2, method = 'cart', seed = 500)**

It’s important to mention that mice stands for multiple imputation by chained equations. The ‘m’ argument in the function indicates how many rounds of imputation we want to do. For simplicity, I will choose 2. The ‘method’ argument indicates which of the many methods for imputations we want to use. I chose CART which stands for classification and regression trees. This method work with all variables types, and that’s why I chose it. Now let’s merge the imputed data into our original dataset. We can do this by using the complete() function:

**tr <- complete(imputed\_Data,2) #here I chose the second round of data imputation**

Check missing data again, we note that there is no missing data after the imputation:

**sapply(tr, function(x) sum(is.na(x)))**

It’s time to treat the extreme values. Looking at the LoanAmount variable, we guess that extreme values are possible as some customers, for some reason, may want to apply for higher loan amounts. We will perform the log transformation to normalize the data:

**tr$LogLoanAmount <- log(tr$LoanAmount)**

**par(mfrow=c(1,2))**

**hist(tr$LogLoanAmount,**

**main="Histogram for Loan Amount",**

**xlab="Loan Amount",**

**border="blue",**

**col="maroon",**

**las=1,**

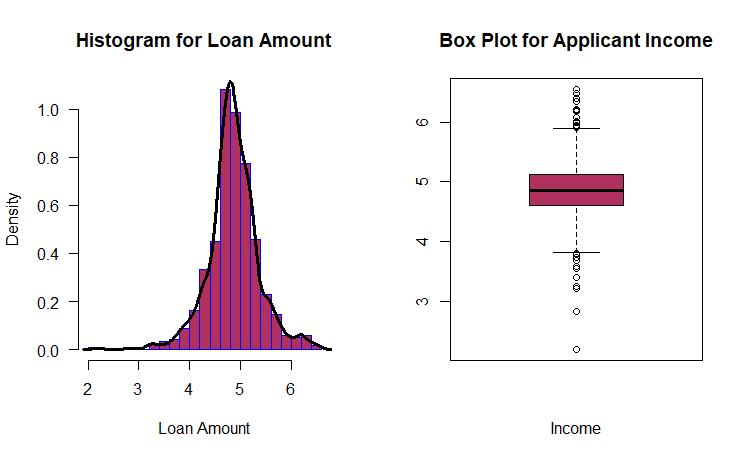
**breaks=20, prob = TRUE)**

**lines(density(tr$LogLoanAmount), col='black', lwd=3)**

**boxplot(tr$LogLoanAmount, col='maroon',xlab = 'Income', main = 'Box Plot for Applicant Income')**

**dev.off()**

Now the distribution looks closer to normal and effect of extreme values has significantly subsided.



Coming to ApplicantIncome, it will be a good idea to combine both ApplicantIncome and Co-applicants as total income and then perform log transformation of the combined variable.

we will use the CART imputation method. If we know that the values for a measurement fall in a certain range, we can fill in empty values with the average of that measurement.

**tr$Income <- tr$ApplicantIncome + tr$CoapplicantIncome**

**tr$ApplicantIncome <- NULL**

**tr$CoapplicantIncome <- NULL**

**tr$LogIncome <- log(tr$Income)**

**par(mfrow=c(1,2))**

**hist(tr$LogIncome,**

**main="Histogram for Applicant Income",**

**xlab="Income",**

**border="blue",**

**col="maroon",**

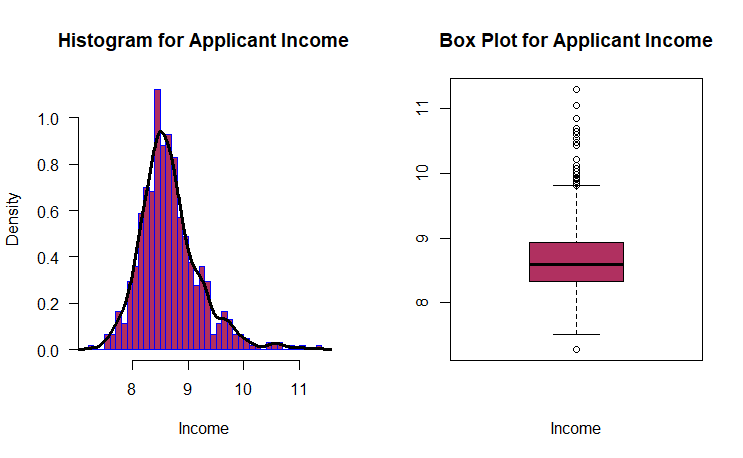
**las=1,**

**breaks=50, prob = TRUE)**

**lines(density(tr$LogIncome), col='black', lwd=3)**

**boxplot(tr$LogIncome, col='maroon',xlab = 'Income', main = 'Box Plot for Applicant Income')**

**dev.off()**



We see that the distribution is better and closer to a normal distribution.

### Building Predictive Models

Now it’s the time to make the next big step in our analysis which is splitting the data into training and test sets.

A training set is the subset of the data that we use to train our models but the test set is a random subset of the data which are derived from the training set. We will use the test set to validate our models as un-foreseen data.

In a sparse data like ours, it’s easy to overfit the data. Overfit in simple terms means that the model will learn the training set that it won’t be able to handle most of the cases it has never seen before. Therefore, we are going to score the data using our test set. Once we split the data, we will treat the testing set like it no longer exists. Let’s split the data:

**set.seed(42)**

**sample <- sample.int(n = nrow(tr), size = floor(.70\*nrow(tr)), replace = F)**

**trainnew <- tr[sample, ]**

**testnew <- tr[-sample, ]**

### Logistic Regression

### We will now start with our first logistic regression model. We will not take all the variables in the model because this might cause an overfitting of the data. To choose our variables, let’s examine the importance of the variables logically. The chances that an applicant’s application would be approved is higher if:

### Applicants took a loan before. Credit history is the variable which answers that.

### Applicants with higher incomes. So, we might look at the income variable which we created.

### Applicants with higher education.

### Applicants who have stable jobs.

### We will use Credit\_History variable in our first logistic regression model.

### Logistic1 <- glm (Loan\_Status ~ Credit\_History,data = trainnew, family = binomial)

### summary(logistic1)

### my\_prediction\_tr1 <- predict(logistic1, newdata = trainnew, type = "response")

### table(trainnew$Loan\_Status, my\_prediction\_tr1 > 0.5)

### logistic\_test1 <- glm (Loan\_Status ~ Credit\_History,data = testnew, family = binomial)

### summary(logistic\_test1)

### my\_prediction\_te1 <- predict(logistic\_test1, newdata = testnew, type = "response")

### table(testnew$Loan\_Status, my\_prediction\_te1 > 0.5)

### Logistic Regression, in simple terms, predicts the probability of occurrence of an event by fitting data to a logit function. Regression coefficients represent the mean change in the response variable for one unit of change in the predictor variable while holding other predictors in the model constant. This type of models is part of a larger class of algorithms known as Generalized Linear Model or GLM. The equation of logistic regression is:

### g(E(y)) = α + βx1 + γx2

### g() is the function.

### E(y) is the expectation of the dependent variable.

### α + βx1 + γx2 is the linear predictor.

### The role of the link function is to link the expectation of ‘y’ to the linear predictor. Logistic regression has the following assumptions:

### GLM does not assume a linear relationship between dependent and independent variables.

### Dependent variable need not to be normally distributed.

### It uses maximum likelihood estimation (MLE).

### Errors need to be independent but not normally distributed.

### In the output, the first thing we see is the call, this is R reminding us about the model we have run. Next, we see the deviance residuals which are the measures of the model fit. This part shows the distribution of the deviance residuals for individual cases used in the model. The next part shows the coefficients, their standard errors, the z-statistic, and the associated p-value. The p-value for each term tests the null hypothesis that the coefficient is equal to zero (no effect). A low p-value (< 0.05) indicates that we can reject the null hypothesis and the predictor is meaningful to the model. Conversely, a larger p-value indicates that changes in the predictor are not associated with changes in the dependent variable and that it’s insignificant. The p-value for the Credit\_History is so small and therefore, it’s significant.

### We have also generated a confusion table to check the accuracy of the model on both the train and the test data:

### Train data: 81.12%

### Test data: 83.24%

### Let’s add other variables and check the accuracy:

### logistic2 <- glm (Loan\_Status ~ Credit\_History+Education+Self\_Employed+Property\_Area+LogLoanAmount+

### LogIncome,data = trainnew, family = binomial)

### summary(logistic2)

### my\_prediction\_tr2 <- predict(logistic2, newdata = trainnew, type = "response")

### table(trainnew$Loan\_Status, my\_prediction\_tr2 > 0.5)

### logistic\_test2 <- glm (Loan\_Status ~ Credit\_History+Education+Self\_Employed+Property\_Area+LogLoanAmount+

### LogIncome,data = testnew, family = binomial)

### summary(logistic\_test2)

### my\_prediction\_te2 <- predict(logistic\_test2, newdata = testnew, type = "response")

### table(testnew$Loan\_Status, my\_prediction\_te2 > 0.5)

### Train data: 81.11%

### Test data: 83.78%

### We note that adding variables improved the accuracy of the test set.

### Decision Tree

### Decision trees create a set of binary splits on the predictor variables in order to create a tree that can be used to classify new observations into one of two groups. Here, we will be using classical trees. The algorithm of this model is the following:

### Choose the predictor variable that best splits the data into two groups;

### Separate the data into these two groups;

### Repeat these steps until a subgroup contains fewer than a minimum number of observations;

### To classify a case, run it down the tree to a terminal node, and assign it the model outcome value assigned in the previous step.

### library(rpart)

### # grow tree

### dtree <- rpart(Loan\_Status ~ Credit\_History+Education+Self\_Employed+Property\_Area+LogLoanAmount+

### LogIncome,method="class", data=trainnew,parms=list(split="information"))

### dtree$cptable

### plotcp(dtree)

### dtree.pruned <- prune(dtree, cp=.02290076)

### library(rpart.plot)

### prp(dtree.pruned, type = 2, extra = 104,

### fallen.leaves = TRUE, main="Decision Tree")

### dtree.pred <- predict(dtree.pruned, trainnew, type="class")

### dtree.perf <- table(trainnew$Loan\_Status, dtree.pred,

### dnn=c("Actual", "Predicted"))

### dtree.perf

### C:\Users\USER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cp.png

### C:\Users\USER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Decision tree.png

### In R, decision trees can be grown and pruned using the rpart() function and prune() function in the rpart package. First, the tree is grown using the rpart() function. We printed the tree and the summary to examine the fitted model. The tree may be too large and need to be pruned. To choose a final tree size, examine the cptable of the list returned by rpart(). It contains data about the prediction error for different tree sizes. The complexity parameter (cp) is used to penalize larger trees. Tree size is defined by the number of branch splits (nsplit). A tree with n splits has n + 1 terminal nodes. The (rel error) contains the error rate for a tree of a given size in the training sample. The cross-validated error (xerror) is based on 10-fold cross validation, using the training sample. The (xstd) contains the standard error of the cross-validation error.

### The plotcp() function plots the cross-validated error against the complexity parameter. To choose the final tree size, we need to choose the smallest tree whose cross-validated error is within one standard error of the minimum cross-validated error value. In our case, the minimum cross-validated error is 0.618 with a standard error of 0.0618. So, the smallest tree with a cross-validated error is within 0.618 ± 0.0618 that is between 0.56 and 0.68 is selected. From the table, a tree with one splits (cross-validated error = 0.618) fits the requirement.

### From the cptable, a tree with one splits has a complexity parameter of 0.02290076, so the statement prune(dtree, cp=0.2290076) returns a tree with the desired size.

### We have then plotted the tree: pruned tree for predicting the loan status. We look at the tree at the top moving left if a condition is true or right otherwise. When an observation hits a terminal node, it’s classified. Each node contains the probability of the classes in that node, along with percentage of the sample.

### Finally, we ran the confusion table to know the accuracy of the model.

### PS: We followed the same steps in the test data.

### dtree\_test <- rpart(Loan\_Status ~ Credit\_History+Education+Self\_Employed+Property\_Area+LogLoanAmount+

### LogIncome,method="class", data=testnew,parms=list(split="information"))

### dtree\_test$cptable

### plotcp(dtree\_test)

### dtree\_test.pruned <- prune(dtree\_test, cp=.01639344)

### prp(dtree\_test.pruned, type = 2, extra = 104,

### fallen.leaves = TRUE, main="Decision Tree")

### dtree\_test.pred <- predict(dtree\_test.pruned, testnew, type="class")

### dtree\_test.perf <- table(testnew$Loan\_Status, dtree\_test.pred,

### dnn=c("Actual", "Predicted"))

### dtree\_test.perf

### Accuracy:

### Train data: 81.81%

### Test data: 85.4%

### Results show better performance than the logistic model.

### Random Forest

### A random forest is an ensemble learning approach to supervised learning. This approach develops multiple predictive models, and the results are aggregated to improve classification. The algorithm is as follows:

### Grow many decision trees by sampling;

### Sample m < M variables at each node;

### Grow each tree fully without pruning;

### Terminal nodes are assigned to a class based on the mode of cases in that node;

### Classify new cases by sending them down all the trees and taking a vote.

### Random forests are grown using randomForest() function in the randomForest Package in R. The default number of trees is 500, the default number of variables sampled at each node is sqrt(M), and the minimum node size is 1.

### library(randomForest)

### set.seed(42)

### fit.forest <- randomForest(Loan\_Status ~ Credit\_History+Education+Self\_Employed+Property\_Area+LogLoanAmount+

### LogIncome, data=trainnew,

### na.action=na.roughfix,

### importance=TRUE)

### fit.forest

### importance(fit.forest, type=2)

### forest.pred <- predict(fit.forest, testnew)

### forest.perf <- table(testnew$Loan\_Status, forest.pred,

### dnn=c("Actual", "Predicted"))

### forest.perf

### Here is the accuracy of the model:

### Train data: 79.95%

### Test data: 82.16%

### The random forest function grew 500 traditional decision trees by sampling 429 observations with replacement from the training sample. Random forests provides natural measure of variable importance. The relative importance measure specified by type=2 option is the total decrease in node impurities from splitting on that variable, averaged over all trees. In our trees,the most important variable is Credit\_History and the least is Self\_Employed. We have finally measured the accuracy for the training sample and applied the prediction to the test sample. We note that the accuracy for both are less than the decision tree’s accuracy.

### We will run the same model but this time we will select the highest three in importance:

### set.seed(42)

### fit.forest2 <- randomForest(Loan\_Status ~ Credit\_History+LogLoanAmount+

### LogIncome, data=trainnew,importance=TRUE)

### fit.forest2

### forest.pred2 <- predict(fit.forest2, testnew)

### forest.perf\_test <- table(testnew$Loan\_Status, forest.pred2,

### dnn=c("Actual", "Predicted"))

### forest.perf\_test

### Here, we notice slight improvements on both samples where accuracy for the training sample is 80.88% and the accuracy for the test sample is 83.24%. Accuracy for decision tree is still better.

### Random forests tend to be very accurate compared to other classification methods though. Also, they can handle large problems. Personally, I have more confidence from the results generated from forest trees compared to decision trees. One problem which might occur with single decision tree is that it can overfit. Random forest, on the other hand, prevents overfitting by creating random subsets of the variables and building smaller trees using the subsets and then it combines the subtrees.

### Chosen Model & Scoring

### Although the accuracy for the decision tree is better, I’m choosing the random forest tree model. The reason is that the difference in accuracy slightly differ between the two models. Also, I prefer the forest model for the reasons mentioned in the previous section.

### Let’s now create a data frame with two columns: Loan\_ID and Loan\_Status containing our preditions:

### my\_solution <- data.frame(Loan\_ID = testnew$Loan\_ID, Loan\_Status = forest.pred2)

### Write the solution to a csv file with the name my\_solution.csv

### write.csv(my\_solution, file = "my\_solution.csv", row.names = FALSE)

### So now, we have predictions for 185 customers who apply for loans with accuracy of 83.24%. We can apply this method for any new data set with same variables to have a prediction about their eligibility of getting a loan.

## 