HUMBOLDT-UNIVERSITÄT ZU BERLIN

Predictive Analysis of Customer Credit Data

Statistical Programming Languages (SS 16)

Yousuf Hasan Siddiqui
Orhan Ipek
Shikhar Srivastava

Credit approval process is challenging and absolutely essential in risk management. This project aims to learn and implement advanced methods used in the industry.

Contents

Introduction

Credit represents the agreement between two parties, when one immediate value or service receiving party agrees to repay the agreed value on the future with a specified repayment structure and schedule. Consumption is substantially increasing on every aspect of life after constant increase in innovation and production capabilities. More stable expected incomes and future expectations have also undeniable effect. These factors trigger more financing needs for individuals and businesses. A poorly managed credit lending process and lack of regulations have caused a huge increase in outstanding loans with a 5.9% increase between 2008-2012.

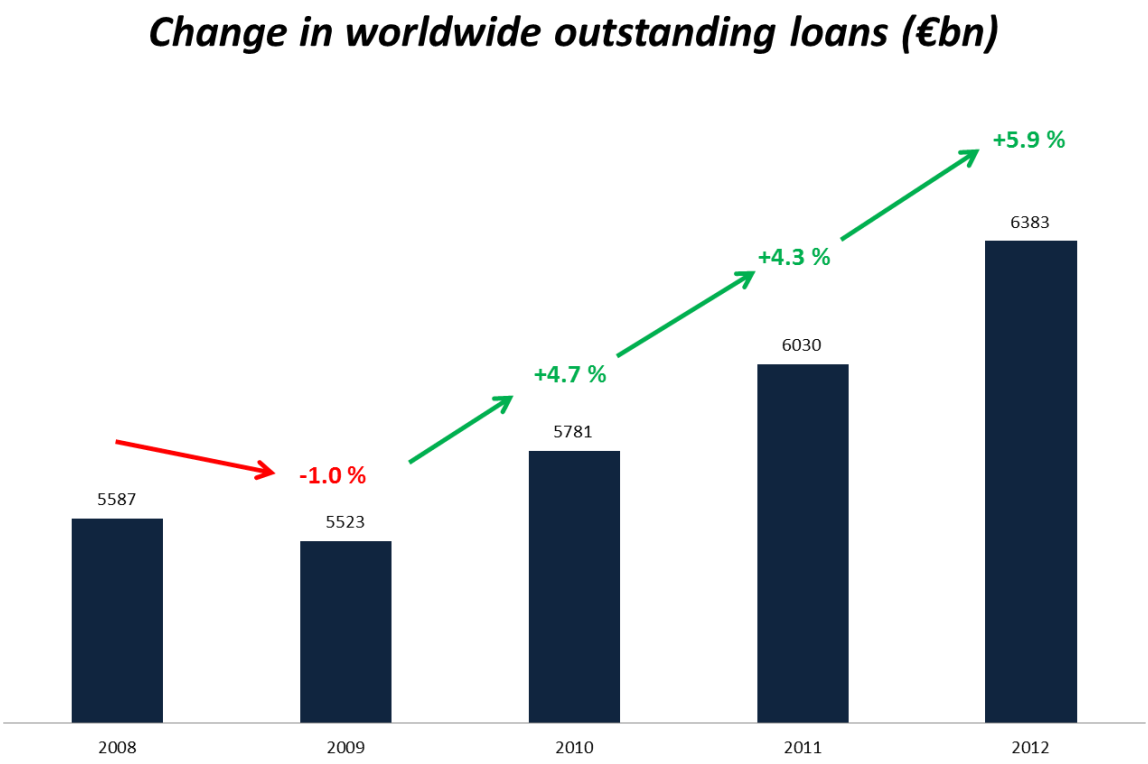


Figure 1: Outstanding Loans Trend

Therefore, credit evaluation and approval process is the one of the most important and researched topics in the financial sector and risk management fields. This process aims to predict if an individual or a business will be able to capable to repay the agreed value on the contract. Since, knowing the intentions and future plans of the borrower is not completely possible and lenders only have personal information like name, address, expected income, purpose of the credit and credit history of the customer. Credit approving process is using those information and repayment patterns from historical observation in order to do the prediction. This process was conducted manually in the old days of finance. However, increase in the number of applications forced lender financial institutions to create a more stable, agile and fast method. Computing machines, credit algorithms and machine learning and their occurrence have adapted in an extreme fast pace in the sector and how they manage risk.

In this project we aimed to use predictive analytics and machine learning algorithms in order to provide predictions for future customers by using a historical German credit data. Predictive analytics creates the opportunity to analyze vast amounts of data and find trends and patterns to use in predictions. Machine learning is used for the self-update of the algorithms and brings agility to the process.

Motivation

We have chosen credit sector because of the popularity and dynamism of the topic. Not especially in the financial sector but most of the companies especially subscription based ones are building their own algorithms and evaluation processes to evaluate their customers and if it is valuable to serve those customers or not. Big data creates the opportunity to manage the risk rapidly and efficiently in this case. We wanted to use advanced methodologies used in industry.

Data

The source German Credit Data we used in the project is by Professor Dr. Hans Hofmann from Institut für Statistik and Ökonometrie Universität Hamburg.

- Contains both quantitative and qualitative attributes
- Variables: 20
- Number of observations: 1000
- Data contained no missing values but outliers.

Complete information regarding the data set is attached in appendix A.3.

Descriptive Statistics

Descriptive statistics are performed to describe features of a data set and find existing patterns in the collection of information. To be slightly more specific, we performed exploratory data analysis which included summarizing the main characteristics of the data set through visual techniques which including graphs and plots. As our data set contained both qualitative and quantitative variables we had to be selective in our approach regarding which descriptive statistic measure to use while trying to explore the characteristics of each variable. In total we used five different summary statistics which included the following;

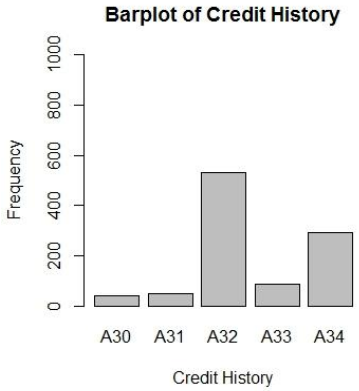
- Qualitative Variables (Categorical Variables)
    - Bar Plots
    - Mosaic Plots

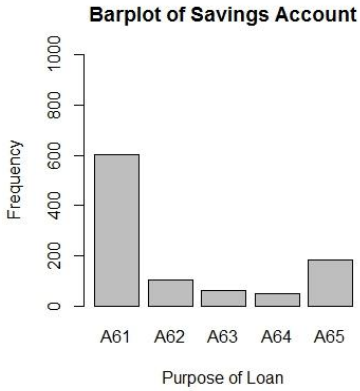
- Quantitative Variables (Continuous Variables)
    - Histograms
    - Kernel Plots
    - Box Plots

Qualitative Variables

Bar Plots

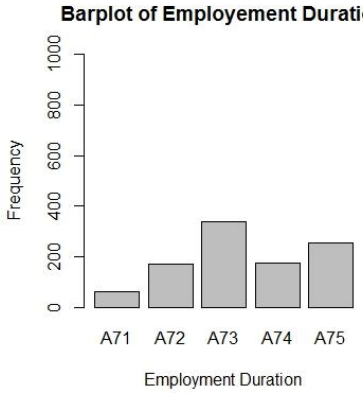
Bar plots provide the knowledge about the population's distribution around the categorical variables.

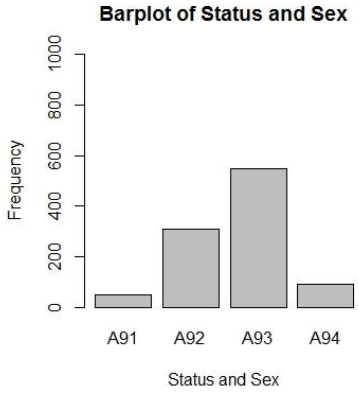




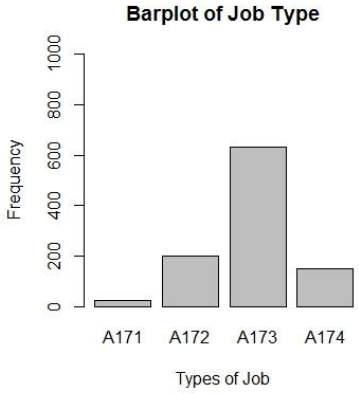
Legend
Credit history
A30 : no credits taken/
all credits paid back duly
A31 : all credits at this bank paid back duly
A32 : existing credits paid back duly till now
A33 : delay in paying off in the past
A34 : critical account/
other credits existing (not at this bank)
Savings account/bonds
A61 : ... < 100 DM
A62 : 100 <= ... < 500 DM
A63 : 500 <= ... < 1000 DM
A64 : .. >= 1000 DM
A65 : unknown/ no savings account

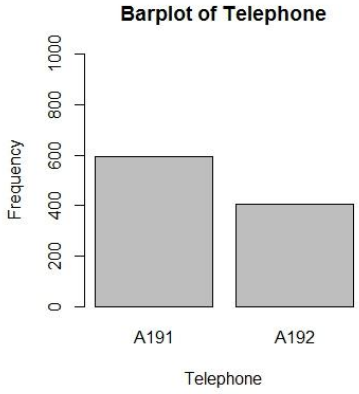
Most of the population paid their previous debts on time, however have relatively small savings.





Legend
Present employment since
A71 : unemployed
A72 : ... < 1 year
A73 : 1 <= ... < 4 years
A74 : 4 <= ... < 7 years
A75 : .. >= 7 years
Personal status and sex
A91 : male : divorced/separated
A92 : female : divorced/separated/married
A93 : male : single
A94 : male : married/widowed
A95 : female : single



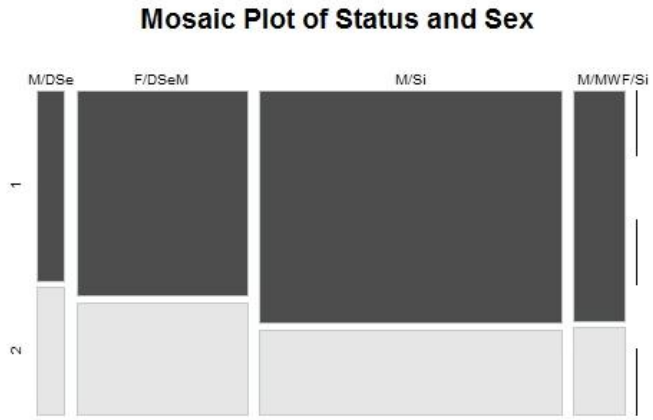


Legend
Job
A171 : unemployed/ unskilled - non-resident
A172 : unskilled - resident
A173 : skilled employee / official
A174 : management/ self-employed/
highly qualified employee/ officer
Telephone
A191 : none
A192 : yes, registered under the customer's name

Employment duration is fairly evenly distributed among categories with considerably low demand from unemployed customers. Surprisingly, population doesn't contain any female and single individuals and it is dominated by single male individuals. Skilled employees and officials are the biggest credit demanders in the population according to historical data.

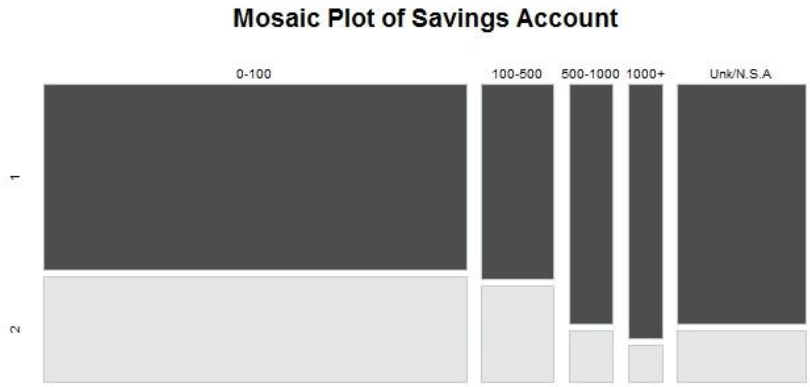
Mosaic Plots

Although bar plots was representing some information about the population and the distribution of population in categorical variables, they were not providing information of their relation with credit approval decision process. To get a simple picture about this relation mosaic plots are used.



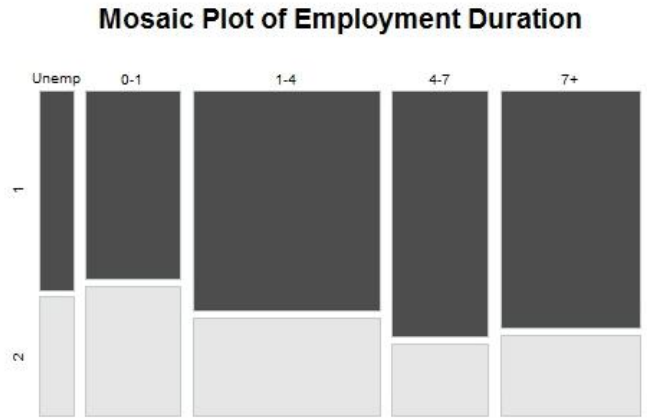
Legend
Status and sex
M/DSe : male : divorced/separated
F/DSeM: female : divorced/separated/married
M/Si : male : single
M/MW : male : married/widowed
F/Si : female : single

The population size of each type is different, observing the significant decrease in the approval rate for male: divorced/separate population. Single males have the higher chance to be classified as 'Good' than any other type in this category. Females only have one type but it can be said that females average have a lower approval date than males.



Legend
Savings account/bonds
0-100 : ... < 100 DM
100-500 : 100 <= ... < 500 DM
500-1000 : 500 <= ... < 1000 DM
1000+ : .. >= 1000 DM
Unk/N.S.A : unknown/ no savings account

The chance to be classified as 'Good' is increasing with the amount in the customers savings accounts/bonds. It is an intuitional result however it also creates a dimension in the data consistency check.



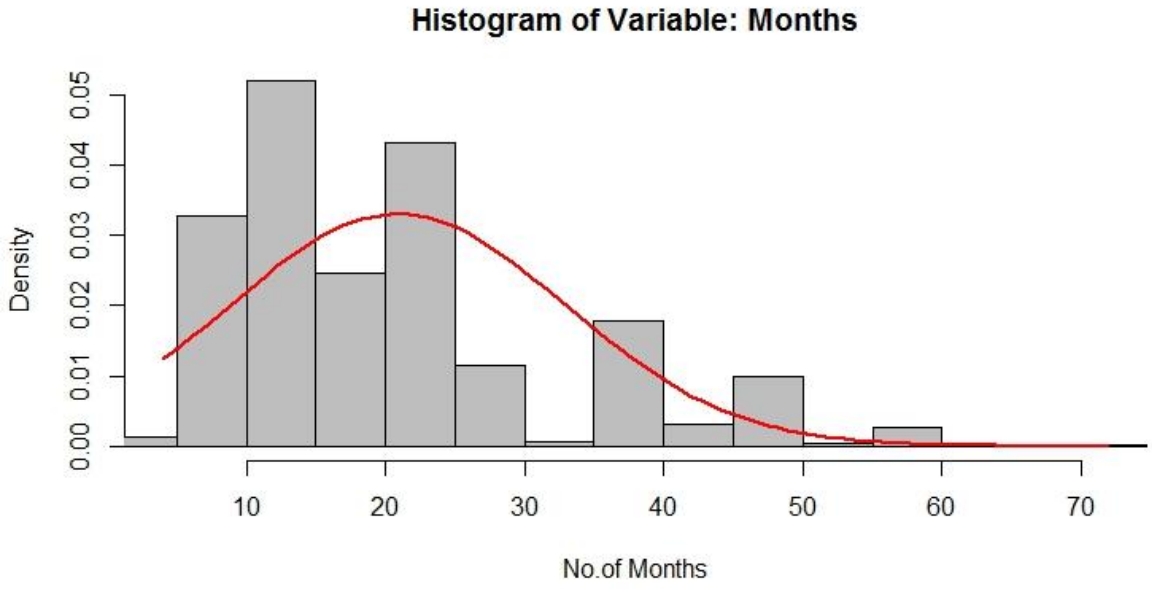
Legend
Savings account/bonds
Present employment since
Unemp : unemployed
0-1 : ... < 1 year
1-4 : 1 <= ... < 4 years
4-7 : 4 <= ... < 7 years
7+ : .. >= 7 years

Increase in the Employment Duration also affects the decision process positively. Slightly higher chance for the unemployed population can be explained by higher savings and not working population.

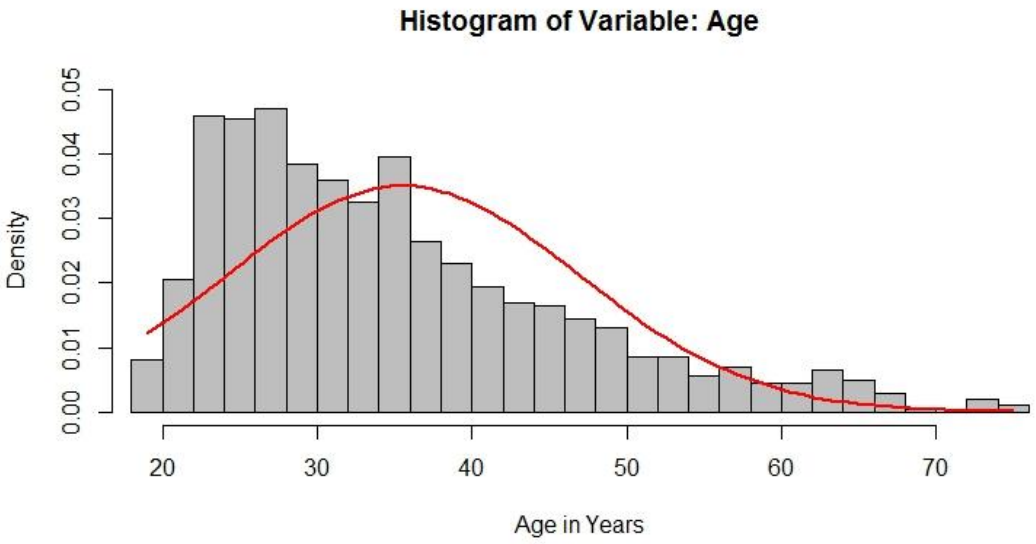
Quantitative Variables

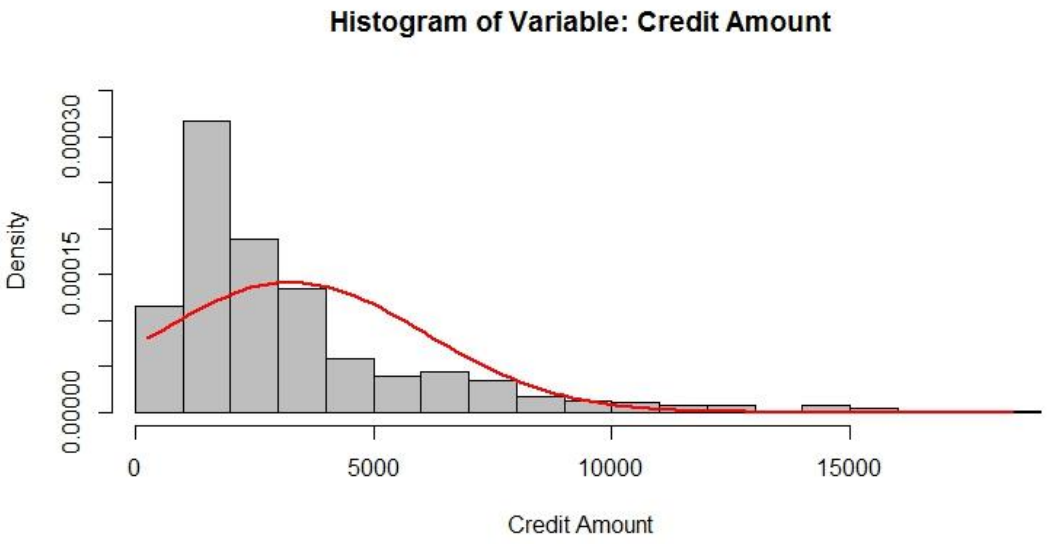
Histograms

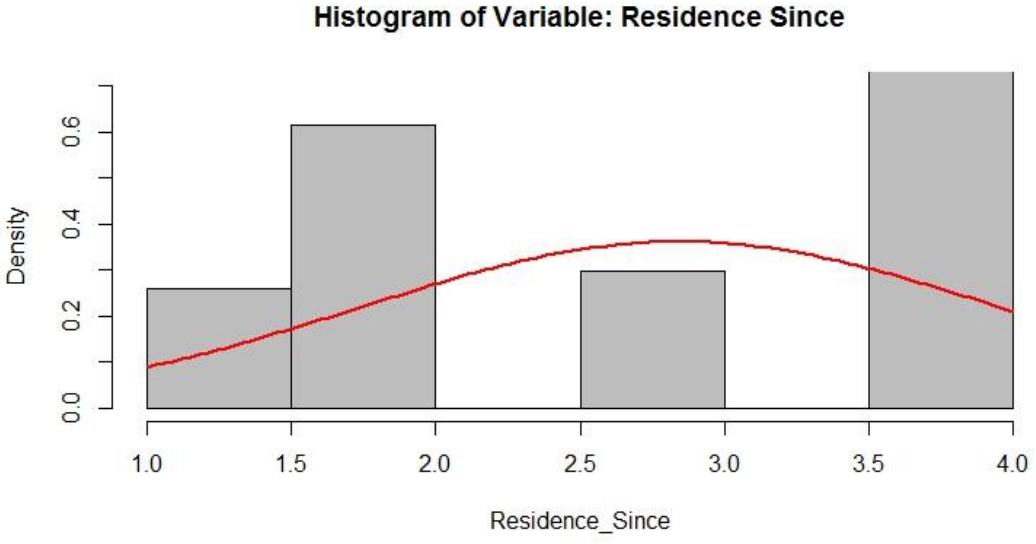
For quantitative variables, histograms were chosen as the first method to preview the data. For histograms selected bin size creates different variations and end ranges in data. Bin size for the variables selected automatically by R software.



Lower durations are more demanded than the higher durations it can be said that most of the population prefers 0-2 year durations.

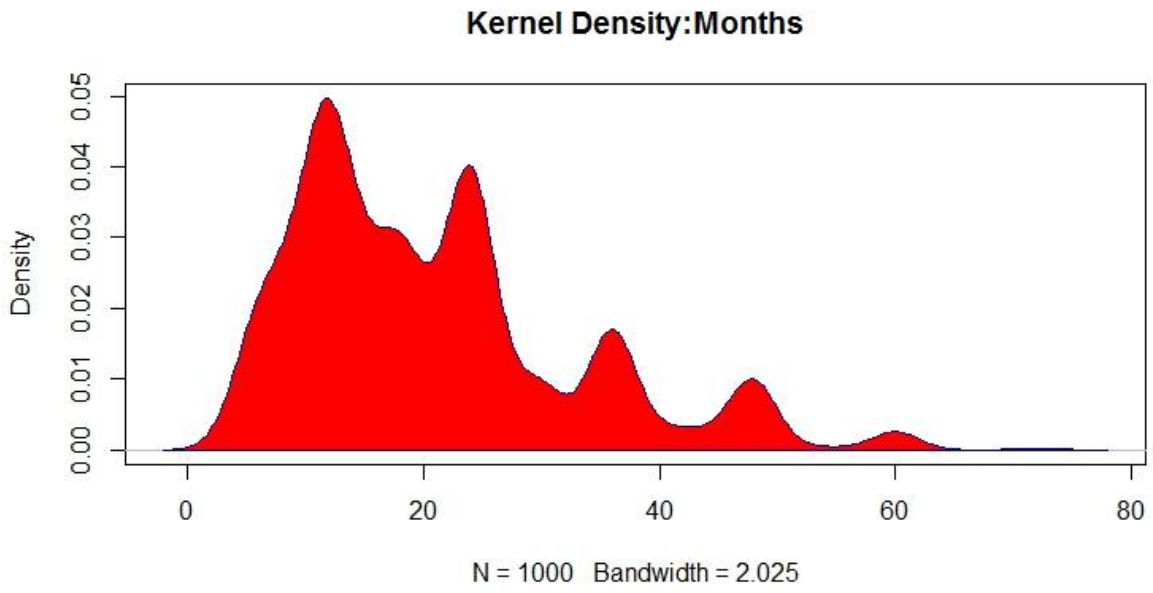

Histogram of Variable: Age
Density
0.00
0.01
0.02
0.03
0.04
0.05
20
30
40
50
60
70
Age in Years

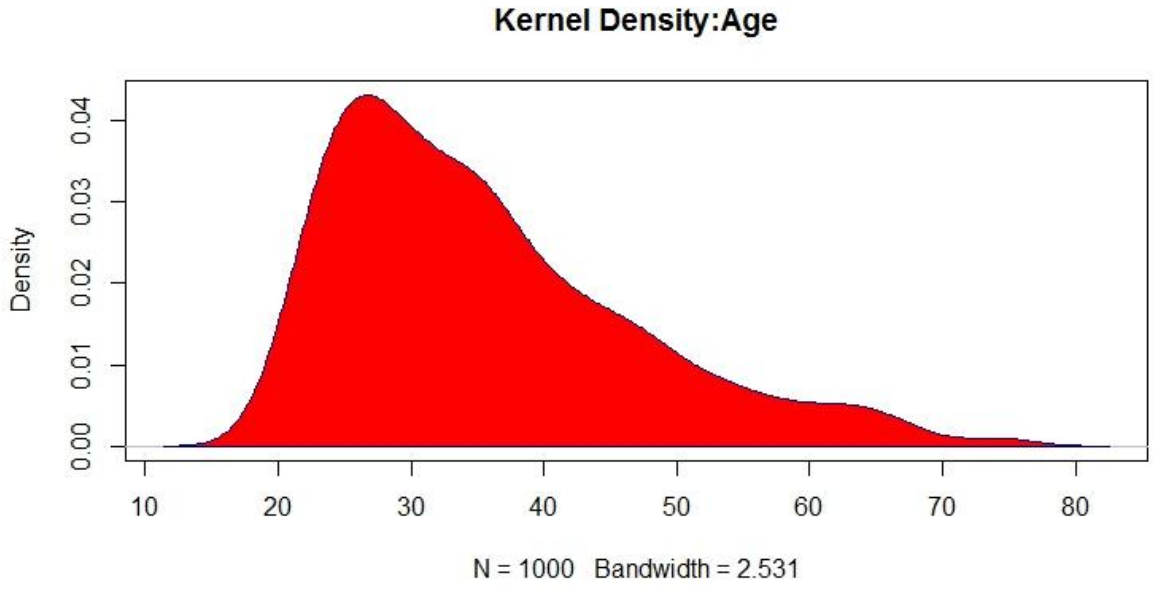

Histogram of Variable: Credit Amount
Density
0.00000
0.00015
0.00030
0
5000
10000
15000
Credit Amount

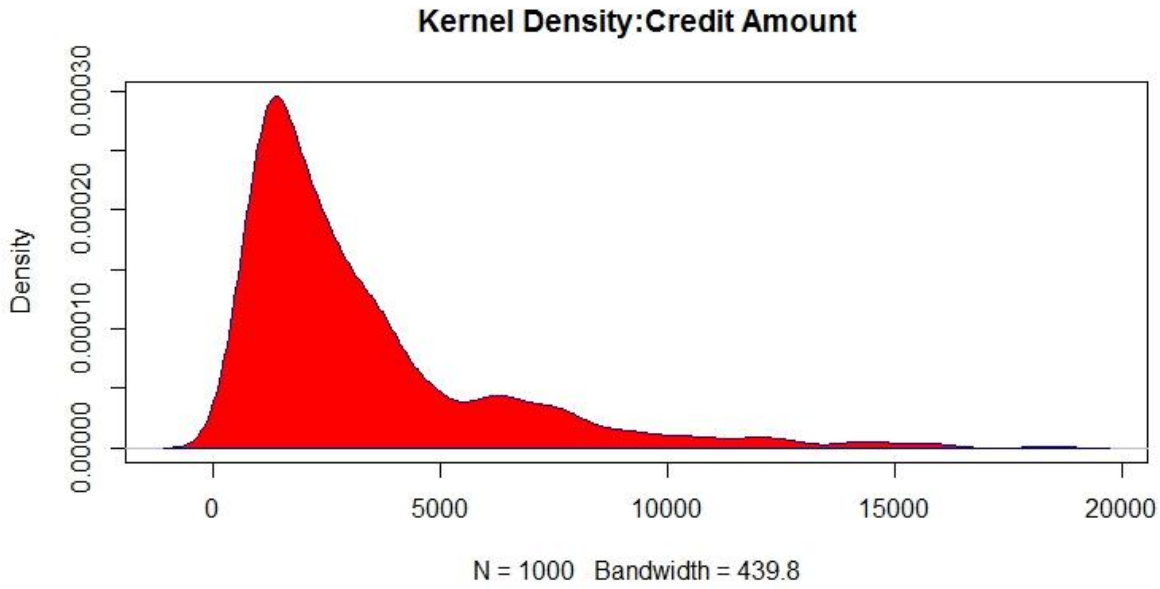

Histogram of Variable: Residence Since
Density
0.0
0.2
0.4
0.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Residence_Since

Kernel Density Plots

To acquire smoother the distributions represented in the histograms kernel density estimation is used and kernel density plots are expected to give a better representation of the data.

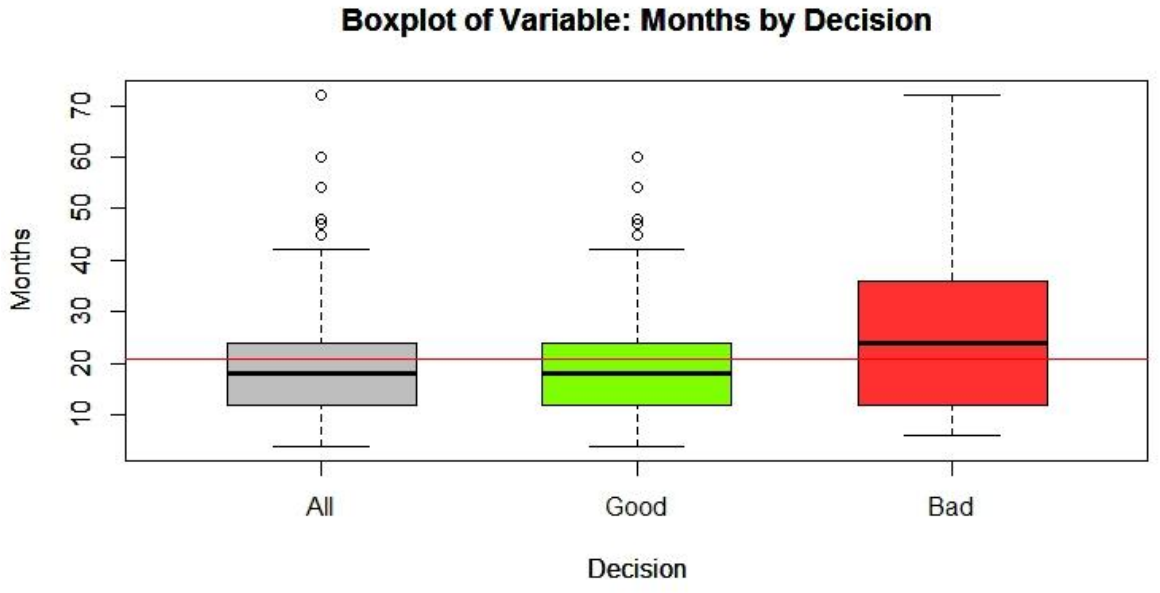




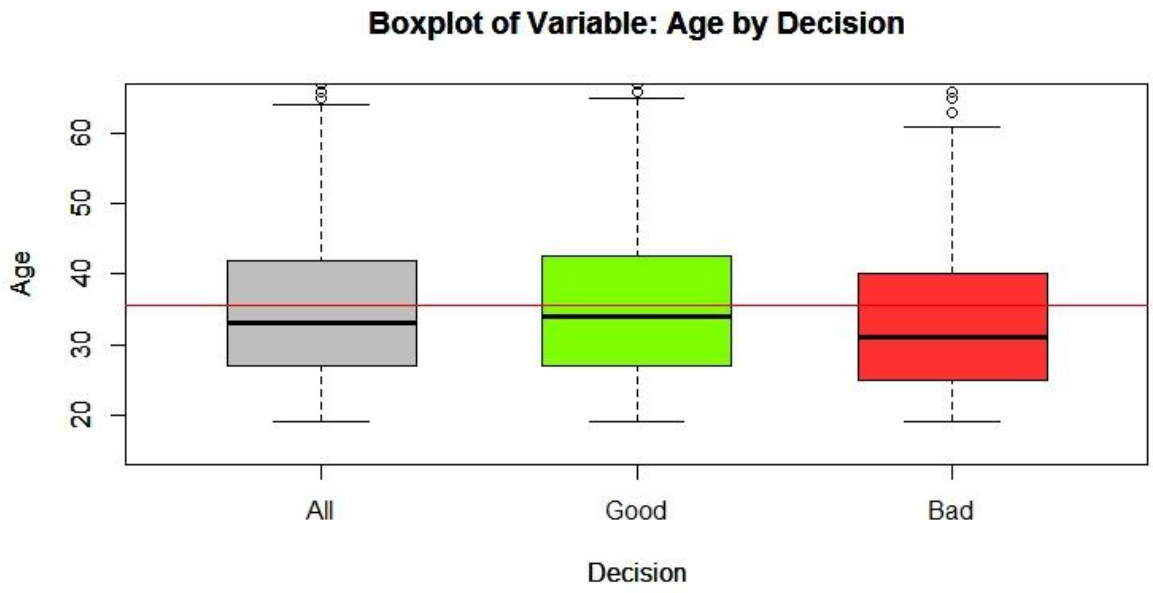


Box Plots

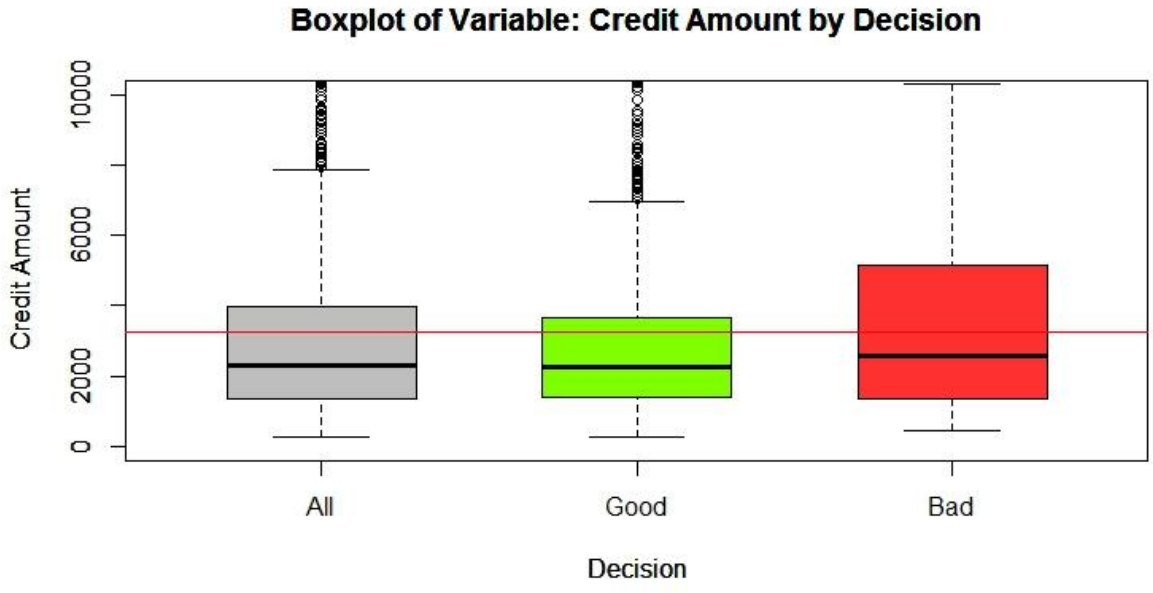
Lastly to observe if the date contains outliers or not box plots are used. Box plots are one of the best methods to detect outliers and consistency of the data.



"Good" classified customers demanded credits with lower median duration than "Bad" ones. As it is discussed before data contains outliers. The difference between the mean of the population and median is representing the skewedness that has been observed in histogram analysis.



As it is expected median age is higher on “Good” decision. Skewedness creates a mean median difference as well. Outliers towards the upper quartiles are observed. Since the lower values have a lower limit outliers are not observed on that side.



Median credit amounts for both decisions are considerable closer and the effect of the credit amount can’t be on the decision process.

Model Selection

In data analysis there are largely two broad categories of models which can be used to predict data trends. These include 'classification' and 'prediction' models with their use depending on the nature of the data these models are subject to. Classification models predict categorical class labels while prediction models predict a continuous valued function. In our case, we have credit scoring data with the target variable being a discrete categorical variable with a binary (1,0) value. Therefore, it was implied that we use a classification model for our predictions. For our predictive analysis we used three different algorithms which have been thoroughly researched and documented and used for forecasting trends in various fields ranging from the banking and finance sector to environmental sciences. These include:

1. Logistic Regression
2. Artificial Neural Networks
3. Random Forest

Each of these algorithms can be fine-tuned to fit a specific application by adjusting their meta-parameters. We have defined meta parameters for each algorithm in their respective section and the range of values used for each are shown in the model overview (table 2).

Logistic Regression:

Logistic regression is the cornerstone of all regressive predictive analysis and used when the dependent variable is dichotomous in nature. Logit Regression assumes that the target variable is a stochastic event i.e. it can have either one or the other output and as per our dataset it proceeds in predicting whether a particular customer has a good or bad credit score. If the likelihood of getting a good score is greater than 0.5 it is assumed good and if lesser than 0.5 then bad. Therefore, it is imperative that the outcome variable must be coded as 0 and 1. The maximum likelihood function used is:

$$\boldsymbol{p(x) = \frac{1}{1 + e^{-(\beta o + \beta 1 x)}}}$$

The value of coefficients, βo and β1 are chosen such that the likelihood $p(x)$ is as close to either of the discrete values 1 or 0 of the output.

Meta-Parameters (LogR):

lambda: regularization factor which penalizes the large weights value of coefficients during the iterative fitting process of finding the maximum likelihood function. This helps in reducing overfitting of the model since large weights means model is trying to fit the noise which is not desirable

Artificial Neural Networks (ANN):

ANN is a close replication of the brain's nervous system used during the decision making process. The mathematical modeling of how neurons interact with each other was initially formulated in the early 1940s but gained recognition after the introduction of a back-propagation algorithm in the 1980s which helped in developing a feedback loop to train the model. In the neural brain network, each neuron receives input from a preceding neuron or a multiple of neurons. This neuron then aggregates and processes the information and if the activation level of the neuron exceeds a specific threshold it sends an output to neurons further in the chain. The algorithm developed for this works exactly the same way as shown in the simplified schematic view of one such network.

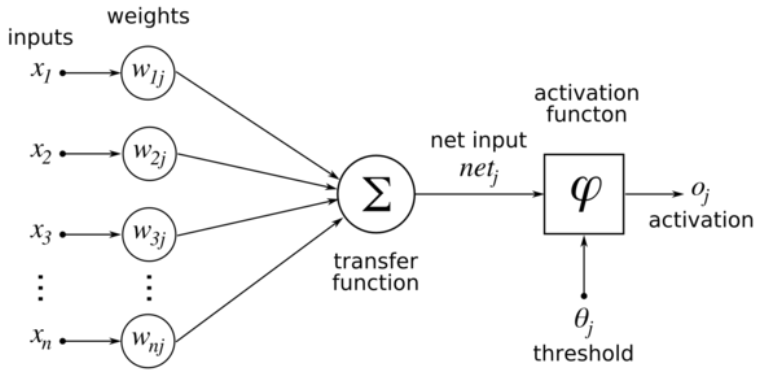


Figure 2: Artificial Neural Network Schema

(Source: https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions)

As illustrated in figure 2, 'x' represents the inputs from preceding neurons in the network with 'w' being the weights randomly assigned to each input signal. The transfer functions job is to aggregate all the inputs according to their respective weightages and produce a net input for the activation function in the neuron. If this aggregated input exceeds the threshold for a neuron an output is produced for neurons further down in the chain. The introduction of backpropagation meant that the output of each such system could be compared to a preset output value to give us the network error represented by $\|y - \bar{y}\|$. This network error was back propagated through the entire network and help in updating the previously assigned random weights for each input. Back-propagation is an iterative methodology with small incremental weight adjustments leading to the optimum learning curve of the algorithm. To summarize, this system can approximate any continuous function i.e. learn highly non-linear, complex relationships in the data.

<u>Meta-Parameters (ANN):</u>

Size: Number of neurons in hidden layer

Decay: parameter of weight decay, for removing noise from inputs. Also, large weights of inputs indicate model complexity. Complex models are prone to overfitting. This parameter also controls overfitting.

Maxit: Maximum number of iteration for back propagation.

Random Forest:

The random forest, a descendent of the decision tree methodology, is a homogenous ensemble learning algorithm. Decision trees give significantly better results compared to classical methods when the relationship between independent and dependent variables are nonlinear and complex. The process at the heart of this methodology is stratification of the sample space and then drawing a random sample from those strata.

The set of independent variables, in our case variables V1 to V20, are divided into 'n' unique and non-overlapping region and ordered by the most commonly occurring classes. A classification error parameter is used for selecting the values in which the regions are divided. The algorithm selects values which minimizes the error rate at each instance of division. The reoccurring binary splitting of trees is usually continued till prediction error is minimized. A limitation of such a methodology is that due to the increasing number of splits it is possible for the algorithm to reach a 0% error rate which is an indicator of over-fitting. Therefore, it will always be possible to obtain a perfect fit of the training data resulting in high variance.

One way to counter this is through the process of pruning. This technique reduces the size of the decision trees i.e. the no. of splits, by removing sections of the tree that provide limited information to the classification process. Pruning can either be performed in a top down, which trims sub-section of the tree starting from the root, or bottom up methodology which starts trimming at the bottom most leaf nodes and makes its way up the tree. This technique increases the overall predictive accuracy of the model and reduces variance through the reduction of overfitting. However, Breiman (1996), argued that a better technique to reduce overfitting without affecting bias of the base model is to perform bootstrapping which injects diversity at a data level. We decided to pay heed to Breiman's advise and a detailed description of the steps performed are in a later section of the report.

Lastly, the reason why Random Forest is often thought of as the best off-the-shelf methods and used in numerous prediction analysis is because the method de correlates predictions of different sub samples. It randomly selects m out of a total n predictor for building decision trees for every split thereby ensuring every subsection has different variables to work with. If this was not done, the resulting predictions will be highly correlated as the algorithm would be choosing the same variables ranked according to their correlation with the target variable.

Meta-Parameters (RF):

ntree: Maximum number of trees allowed to be grown.

Ensemble Modelling

All the different base models described above capture different aspects of relationships between the variables under consideration. Each algorithm is accompanied by its various strengths and weaknesses which are highly reflective on their respective application e.g. linear versus non-linear models. In addition, a major hindrance to the use of one model is the bias-variance tradeoff. The objective of most machine learning models is to reduce the impact of both these statistical quantities, yet doing so with one specific classifier is next to impossible. A complex classifier will tend to over fit the data leading to low bias and high variance while a simple classifier will result in high bias and low variance. Natural conclusions lead to the fact that as various mathematical algorithms perceive data differently, there is no such thing as the one algorithm fits all strategy, hence an intuitive solution is to combine their model predictions in as an ensemble.

Ensemble learning takes each models prediction and combines them, according to their weightage, to generate one ensemble. Within the realm of classification model ensembles, there are further two types which may be used; homogenous and heterogeneous types. It is imperative to identify the differences between the two types and their use is subject to the application scenario and ease of user operation. Homogenous ensemble, as the name suggests, is a combination of identical base models with diversity injected either by using different parametric values for each case or injecting diversity on the data used by them through subsampling through bagging or K-fold validation. On the other hand, heterogeneous ensembles combine different base classifiers but are often run on similar data. We use the ensemble learning method stacking for our analysis.

Stacked Generalization

"Stacked generalization, also known as Stacking, is a method that combines models by learning a meta-level (or level-1) model that predicts the correct class based on the decisions of the base level (or level-0) models" (Oleg et.al 2009). Stacking can also be seen as a more sophisticated version of cross-validation with strategy of winner-takes-all. In this experiment, we created the base level candidates using traditional machine learning algorithms. The predictions created on validation test data were then used for training the ensemble. The second layer of algorithm was applied on these predictions to create final predictions. This experiment used Random Forest and Logistic Regression as second level generalizers over the base level classifier's results. The optimal parametric values' combination at the second layer stage was determined by trial and error method.

Data Mining:

Data Mining is an essential focus of most modelling and simulation processes. The aim is to discover, investigate and summarize patterns in a particular data set which will enable algorithms to run as desired and produce output which is least susceptible to errors. Although there is no right or wrong while beginning this process, a mandatory set of steps are always helpful for ease of operation as and to ensure a desired quality threshold (Jakeman et al. 2006). Hence, we tried to implement a similar industry wide approach to enable minimization of errors.

Data Pre-Processing

Due to the nature of real world data, pre-processing is the first and arguably one of the most critical steps in predictive modeling. A data set containing missing values and outliers, usually a result of human or machine error, must be cleaned and normalized to guarantee data precision. However, counter arguments have been made to this widely accepted notion that such missing values and outliers contain information in them and are actually representative of the data and should be refrained from imputation Wainer (1976). In this project we shall stick to the more accepted ideology of imputation. Our data set did not contain any missing values but had two different types of variables; continuous and categorical. We look into the pre-processing of each type separately as there methodologies differ quite a bit.

In the case of continuous variables, we performed outlier detection by using the z-score methodology. This method uses the mean and standard deviation of a particular dataset to generate a z-score using the formula:

$$z = \frac{x - \mu}{\sigma}$$

Typically, a variable exhibiting a z –score that exceeds a numeric value of 3 is labelled and treated as an outlier. The z-score methodology might be unreliable for small data sets, especially those containing less than or equal to 10 observations (Iglewicz B. et al.1993), but we were in no danger as our dataset is well above that number. The following user-defined outlier function was developed and applied on the list of continuous variables:

```
remove_outliers = function(x, na.rm = TRUE, ...)

{
VarMean = mean(x, na.rm = na.rm, ...)
VarSD = sd(x,na.rm = na.rm, ...)
y = x
y[abs((x - VarMean)/VarSD) >3] = NA
y
}
```

Line (3) calculates the mean and line (4) the standard deviation for the observations. Line (6) applies the z-score formula and checks if this score is greater than 3. Note the function 'abs' is used to ensure

only the absolute values are calculated i.e. anything above +3 and below -3 will be treated the same, irrelevant of the sign. All such values are then imputed with an NA value.

Using 'sapply' we applied the outlier function on the entire dataset thereby replacing all observations in each continuous variable, with a z-score of greater than 3, with NA values. There are a number of techniques which are used for treatment of NA values such as replacement with 0, mean, median, KNN imputation and case deletion (Acuna E., 2004). We chose against imputing these values with 0 or case deletion because it would have led to loss of information from the dataset. The KNN imputation was left out to the complexity of the task and our inexperience with handling such a function. Eventually, by the process of elimination we decided to impute all the NA values with the mean of each respective variable using the function:

```
means = colMeans(dat_cont[,colnames(dat_cont)],na.rm=T)

for(idv in colnames(dat_cont))dat_cont[is.na(dat_cont[,idv]),idv]
=means[idv]
```

The final step of pre-processing for continuous variables included normalization of the predictors. As the types of continuous variables in our data set are very different in nature e.g. 'Credit_Amount' and 'Age' it is difficult for algorithms to assimilate them under one range. The observations (predictors) were therefore scaled between -1 to 1 so that the machine learning algorithms could function properly. The following scaling function was developed:

```
min_max_scaling=function(col)((col-min(col))/(max(col)min(col))*2-1)
##check
min_max_scaling(c(1,2,3,4))
# [1] -1.0000000 -0.3333333  0.3333333  1.0000000
```

Line (1) defined the function used for scaling while line (3) checks if our function is performing the intended task. Since, by the nature of the methodology that we used, applying multiple algorithms on the entire data, scaling was justified as it rarely hurts the algorithms which do not require it.

In the case of categorical variables, dummy encoding was performed for increased comprehensibility of the data to algorithms. Categorical variables are usually fixed discrete values with each value often repeated multiple times in a data set. Dummy encoding transform each categorical variable into multiple variables with each transformed variable being binary in nature i.e. having either a value of 0 or 1. After looking at the data dictionary, we could easily identify the categorical variable and flags were created for each such column having two or more unique values. Taking attribute 15 i.e. 'Housing' as an example. It has three unique values which include: A151-rent, A152-own and A152-for free. Initially housing was represented by V11 with these three values in the observations. Post the dummy encoding process we created variables V11A151, V11A152 and V11A153 each containing a binary element. Below is the snippet of code which was run to perform this operation

```
dummies = dummyVars( ~ ., data = dat_cat[,1:(ncol(dat_cat)-2)])
category_data_dummies=as.data.frame(predict(dummies, newdata =
dat_cat[,1:(ncol(dat_cat)-2)]))
```

Data Partitioning

A major task in predictive modelling is to ensure that the results are accurate and not subject to bias which may come as an endogenous input when using the same data repeatedly. It is essential to make sure that the models are built and tested on a prudently segregated and unbiased data set. We split the data three ways into training, validation and test sets. The training set, as the name suggests, trains the base model the results of which are then compared to the validation set to check the accuracy or as sometime it may refer to as the error rate of the model. If the accuracy results are below an unacceptable level, parameter tunings of the base models is performed until a desired threshold is met. The test sample is not introduced to the model to prevent the model from learning from the data contained in it.

To make sure that all sets are unbiased and not leaning towards certain attributes, we decided to implement 'stratified sampling' which ensures similar variable value distribution in all three samples. Theoretically, this means for each unique value of a variable there will be three duplicate combinations with every value of other variables. Unfortunately, due to a comparatively small data set of a 100 observations we were unable to perform stratification on each variable due to the amount of permutation involved, therefore, only the target variable was subjected to stratified sampling. This meant the sample percentage of people having good and bad credit was constant in all three data sets.

There is a debate in contemporary literature related to the exact ratios in which these data sets are split. It is desirable however to keep the training set to be the biggest in size as they are used for base learning. When it comes to the validation set, though it is much smaller in size, it should be representative of the diversity of the data to accurate measure the efficiency of the model and in effect the same logic applies to the test set. Therefore we chose a ratio of 60:20:20 for train, validation and test sets respectively.

For this operation, the 'createFolds' function of caret package was used. In addition, we decided to remove all variables having a variance of zero using the caret package's 'nearZeroVar' function. This helped in slimming down the sample size and decreasing the processing time involved in running the algorithm. Performing the deletion operation does not result in loss of information as zero variance implies that the variable does not add additional information to the sample set.

```
library(caret)
set.seed(3456)

##dividing rows in 5 folds with control over target variable
devIndex = as.data.frame(createFolds(data_final$dv, k=5,  list = TRUE))

data_div_1= data_final[ devIndex$Fold1, c("id","dv")]
data_div_2= data_final[ devIndex$Fold2, c("id","dv")]
data_div_3= data_final[ devIndex$Fold3, c("id","dv")]
data_div_4= data_final[ devIndex$Fold4, c("id","dv")]
data_div_5= data_final[ devIndex$Fold5, c("id","dv")]


md_rows = rbind(data_div_1,data_div_2,data_div_3) ## Training dataset 60%
val_rows = data_div_4                             ## Validation dataset 20%
test_rows = data_div_5                            ## Test dataset 20%
```

Minimization: Bias and Variance

As mentioned before, after dividing the data, the validation and test sets were kept untouched and operations to minimize the bias and variance are performed on the training set as this is the sample on which the classifier algorithms are run. Since the data was being split, the training set contains a lesser number of observations therefore resampling techniques are used for obtaining additional information about the fitted model and ensuring a robust training process. The two most common and established industrywide statistical methodologies used are K-Fold cross validation and Bootstrap Aggregation.

Cross validation is performed to minimize uncertainty in test error rates value. In K-fold cross-validation, the sample is split into K equal sized subsamples. These divisions are random in nature and the numeric value of K depends on the application. Out of these K subsamples, one is identified for the purpose of validation testing of the model while the remaining K-1 subsamples are used as training data. This process is repeated a K number of times with each subsample getting an opportunity to be a validation test set and the remaining four as training set. This process reduces the variance and over-fitting of the model. We choose a value of 5 for K, signifying that on each run we had 4 training sets i.e. 80% of the original training sample. In addition, similar to the original split between training, validation and test data, K-Fold cross-validation is also performed using the stratifying technique with control over the target variable.

Bootstrap Aggregation, frequently known as bagging, is a frequently used statistical method used for reducing variance and over-fitting. The process involves a simple procedure of random sampling with replacement. Each fold which was created in the previous step is subjected to bagging by randomly selecting 70% from original data and insertion of rest of the 30% by randomly selecting them from the chosen 70% population. This process was repeated 5 times to make sure that randomness in each sample. We tried to increase the frequency of the process above 5 but due to limited our processing power, the available RAM was exceeded and resulted in memory error. In total, 30 training sets, 25 bags and 5 folds, were used for building candidate library

Performance Metrics

There are numerous methods to judge the performance of models be it single algorithms or ensembles. We used one of the popular metric used in industry i.e. Area under ROC curve, or AUC for short. To understand this metric we first need to look at a confusion matrix.

	Predicted True	**Predicted False**
Actual True	True Positive	False Negative
Actual False	False Positive	True Negative

Table 1: Confusion Matrix

Through the confusion matrix we identify the specificity, the ratio of true positive to actual true, and 1-specificty, the ratio of false positive to actual false. The ROC (Receiver operating characteristic) curve is a plot between these statistical quantities and the area under these curves gives an estimate of the performance of the model in question.

Methodology

For each kind of classification algorithm (LogR, random Forest, Neural Network), a data frame of combinations of different parametric values was created. Using different parametric values of the three classification algorithms a total of 4560 model scores were created using the aforementioned methodology.

Classifier	Parametric Values	No.of Models	No.of Candidates
Logistic Regression	Lambda: 2^-19 to 2^6 with increments in power of 1 Cp: “aic”, “bic”	26	780
ANN	Size: 1 to 15 with 2 unit increments Decay: 10^-4 to 10^0 with increments in power of 1 Maxit: 100, 200, 1000	72	2160
Random Forest	ntree: 100 to 1800 with increments of 100 mtry: sqrt(m)/2, sqrt(m), sqrt(m)*2 node size: ceiling (n/200) nxm -size of dataset	54	1620
Total		152	4560

Table 2: Model Overview

For each combination of parameters every training set was used to create a model. Validation and Test set were then used for creating prediction scores of each model. Since there were 30 training sets, we used "doSNOW" and "foreach" packages for optimal system usage by running for loops in parallel computation over all the cores of system. This helped us reducing the processing time by 1/3rd as we could use 3 cores instead of 1.

The validation and test set scores thus created were compiled respectively in separate CSVs and combined using "llply" function of "plyr" library. Learning from the validation scores, CSV were created and testing it on the test set i.e. using validation scores created from aforementioned method as input to classification algorithms and building models on them. The model is then tested on test set to check for accuracy.

Since we were ensembling using the classification algorithm, we wanted to remove effects of multicollinearity. Hence, we checked for correlations between the scores and removed those which showed a score above 0.85 with others. The left out validation scores are now used as the training

input for classification algorithms. Finally, we used AUC performance parameter to judge the prediction performances.

Results

After running individual the base classifiers and then ensembles we saw the following results.

Algorithm	AUC value
Stacking with RF	0.812
Stacking with LogR	0.7
ANN	0.770
RF	0.791
LogR	0.761

Table 3: Performance Measures

Stacking with RF performed the best with an AUC value of 0.812**.**

Conclusion

Because of its nature, financial industry (loan segment in particular) strives for maximal accuracy in future predictions. The decisions based on these predictions directly affects respective firms' portfolio. In our experiment, we see a 2.5% improvement in performance of predictions using a heterogeneous ensemble learning technique (Stacked Generalization) over top performing base classifier. Commenting on the satisfaction reached with this improvement would subject to require inputs from data-owners. Generally speaking, a 2.5% improvement in prediction rates of bad customers in loan industry translates to significant gains. However, everything comes with its sets of pros and cons. Resource and time consumption are the two major drawbacks of using heterogeneous ensemble techniques. These algorithms are data and diversity hungry. In principle, the more data and more number of classifiers we feed in the algorithms, the better is the prediction accuracy. The optimal parameter configuration of the generalizers used in Stacking varies for each problem. It requires manual setting and numerous trial and error based iterations to zero down to the optimal values, which means additional time consumption. On the other hand, for building all the candidates we had to use parallel computing to shorten the time required which required more resource consumption. We used an 8GB Ram Windows machine to do this. On an average, with different supervisors, it took 70% of RAM memory but not more than 300 seconds of learning for one candidate.

However, these drawbacks can be ignored due to the fact that our computational processing abilities are increasing exponentially. The experiment proved the superiority of ensemble method over other traditional learners. With respect to the specific problem set being dealt in this experiment, we feel it would be injustice to generalize the results. The dataset used had only around a thousand observations, and the type and number of independent variables used were limited. In general, there is no single ideal data set for generalizing performance of prediction algorithms. It is common knowledge that statistics work better with more data. We argue that the experiment has to be performed on more varied and bigger real-world datasets to get a clearer picture of comparison between Stacking methodology over base learners. This could be taken as future work for this project. Our expectation is that heterogeneous ensemble algorithms will, in most of the cases, out-perform traditional machine learning methods by a large margin.

References

[1] Jakeman A.J. ,Letcher R.A. ,Norton J.P. (2006) Ten iterative steps in development and evaluation of environmental models

[2] Wainer, H. (1976). Estimating coefficients in linear models: It don't make no nevermind. Psychological Bulletin, 83, 312-317.

[3] Iglewicz B. , Hoaglin D.C (1993) How to detect and handle outliers, ASQC Quality Press

[4] Acuna E., Rodriguez C.(2004) The Treatment of Missing Values and its Effect on Classifier Accuracy, 639-647

[5] Oleg O. and Giorgio V. (2009) Applications of Supervised and Unsupervised Ensemble Methods

[6] Kraus, Anne (2014) Recent Methods from Statistics and Machine Learning for Credit Scoring

Appendix

A.1 Source Code

```
### Libraries to Install

install.packages("caret", repos='http://cran.us.r-project.org')
install.packages("nnet", repos='http://cran.us.r-project.org')
install.packages("pROC", repos='http://cran.us.r-project.org')
install.packages("randomForest", repos='http://cran.us.r-project.org')
install.packages("doSNOW", repos='http://cran.us.r-project.org')
install.packages("foreach", repos='http://cran.us.r-project.org')
install.packages("stepPlr", repos='http://cran.us.r-project.org')
install.packages("plyr", repos='http://cran.us.r-project.org')
install.packages("ROCR", repos='http://cran.us.r-project.org')
install.packages("compiler", repos='http://cran.us.r-project.org')
install.packages("R.utils", repos='http://cran.us.r-project.org')
install.packages("verification", repos='http://cran.us.r-project.org')

#####Descriptive Statistics - Quantitative Variables

### BoxPlots

library(formatR)
tidy_source(file = "boxplot.R")

getwd()


C_D = read.table('german.data')

ncol(C_D)
nrow(C_D)
C_D$All = c("All")
#Setting Variable Names
colnames(C_D) =
c("Account_Status","Months","Credit_History","Credit_Purpose","Credit_Amoun
t","Savings_Account"

,"Employed_Since","Installment_Rate","Status_and_Sex","Other_Debtors","Resi
dence_Since",

"Property","Age","Other_Installement_Plans","Housing_Type","Existing_Credit
","Job_Type",
                    "Dependents","Telephone","Foreign_Worker","Cost_Matrix")

Good = subset(C_D, Cost_Matrix == '1')
Bad = subset(C_D, Cost_Matrix == '2')


#####Boxplots######
graphics.off()

par(mfrow=c(3,2))


#Variable: Months
mmonths = mean(C_D$Months)

boxplot(C_D$Months, at = 1, xlim = c(0.5, 3.5), main="Boxplot of Variable:
Months by Decision",
        xlab="Decision", ylab="Months", col="gray", boxwex = 1.2)
```

```
axis(1, at=1, labels=c('All'))
boxplot(Good$Months, at = 2, add = TRUE, main="Boxplot of Variable: Months
by Decision",
        xlab="Decision", ylab="Months", col="chartreuse", boxwex = 1.2)
axis(1, at=2, labels=c('Good'))
boxplot(Bad$Months, at = 3, add = TRUE, main="Boxplot of Variable: Months
by Decision",
        xlab="Decision", ylab="Months", col="firebrick1", boxwex = 1.2)
axis(1, at=3, labels=c('Bad'))
abline(h=mean(mmonths), col="red")



#Variable: Age

mAge = mean(C_D$Age)

boxplot(C_D$Age, at = 1, xlim = c(0.5, 3.5), ylim = c(15,65), main="Boxplot
of Variable: Age by Decision",
        xlab="Decision", ylab="Age", col="gray", boxwex = 1.2)
axis(1, at=1, labels=c('All'))
boxplot(Good$Age, at = 2, ylim = c(15,65), add = TRUE, main="Boxplot of
Variable: Age by Decision",
        xlab="Decision", ylab="Age", col="chartreuse", boxwex = 1.2)
axis(1, at=2, labels=c('Good'))
boxplot(Bad$Age, at = 3, ylim = c(15,65), add = TRUE, main="Boxplot of
Variable: Age by Decision",
        xlab="Decision", ylab="Age", col="firebrick1", boxwex = 1.2)
axis(1, at=3, labels=c('Bad'))
abline(h=mean(mAge), col="red")



#Variable: Credit Amount

mCredit_Amount = mean(C_D$Credit_Amount)

boxplot(C_D$Credit_Amount, at = 1, xlim = c(0.5, 3.5), ylim = c(0,10000),
main="Boxplot of Variable: Credit Amount by Decision",
        xlab="Decision", ylab="Credit Amount", col="gray", boxwex = 1.2,
breaks = 2000)
axis(1, at=1, labels=c('All'))
boxplot(Good$Credit_Amount, at = 2, ylim = c(0,10000), add = TRUE,
main="Boxplot of Variable: Credit Amount by Decision",
        xlab="Decision", ylab="Credit Amount", col="chartreuse", boxwex =
1.2, breaks = 2000)
axis(1, at=2, labels=c('Good'))
boxplot(Bad$Credit_Amount, at = 3, ylim = c(0,10000), add = TRUE,
main="Boxplot of Variable: Credit Amount by Decision",
        xlab="Decision", ylab="Credit Amount", col="firebrick1", boxwex =
1.2, breaks = 2000)
axis(1, at=3, labels=c('Bad'))
abline(h=mean(mCredit_Amount), col="red")

#Variable: Installment Rate

mInstallment_Rate = mean(C_D$Installment_Rate)

boxplot(C_D$Installment_Rate, at = 1, xlim = c(0.5, 3.5), ylim =
c(0.5,4.5), main="Boxplot of Variable: Installment Rate by Decision",
        xlab="Decision", ylab="Installment Rate (%)", col="gray", boxwex =
1.2)
axis(1, at=1, labels=c('All'))
```

```
boxplot(Good$Installment_Rate, at = 2, ylim = c(0.5,4.5), add = TRUE,
main="Boxplot of Variable: Installment Rate by Decision",
        xlab="Decision", ylab="Installment Rate (%)", col="chartreuse",
boxwex = 1.2)
axis(1, at=2, labels=c('Good'))
boxplot(Bad$Installment_Rate, at = 3, ylim = c(0.5,4.5), add = TRUE,
main="Boxplot of Variable: Installment Rate by Decision",
        xlab="Decision", ylab="Installment Rate (%)", col="firebrick1",
boxwex = 1.2)
axis(1, at=3, labels=c('Bad'))
abline(h=mean(mInstallment_Rate), col="red")

#Variable: Residence Since

mResidence_Since = mean(C_D$Residence_Since)

boxplot(C_D$Residence_Since, at = 1, xlim = c(0.5, 3.5), ylim = c(0.5,4.5),
main="Boxplot of Variable: Residence Since by Decision",
        xlab="Decision", ylab="Residence Since", col="gray", boxwex = 1.2,
breaks = 0.5)
axis(1, at=1, labels=c('All'))
boxplot(Good$Residence_Since, at = 2, ylim = c(0.5,4.5), add = TRUE,
main="Boxplot of Variable: Residence Since by Decision",
        xlab="Decision", ylab="Residence Since", col="chartreuse", boxwex =
1.2, breaks = 0.5)
axis(1, at=2, labels=c('Good'))
boxplot(Bad$Residence_Since, at = 3, ylim = c(0.5,4.5), add = TRUE,
main="Boxplot of Variable: Residence Since by Decision",
        xlab="Decision", ylab="Residence Since", col="firebrick1", boxwex =
1.2, breaks = 0.5)
axis(1, at=3, labels=c('Bad'))
abline(h=mean(mResidence_Since), col="red")


#Variable: Existing Credit

mExisting_Credit = mean(C_D$Existing_Credit)

boxplot(C_D$Existing_Credit, at = 1, xlim = c(0.5, 3.5), ylim = c(0.5,3.5),
main="Boxplot of Variable: Existing Credit by Decision",
        xlab="Decision", ylab="Existing Credit", col="gray", boxwex = 1.2)
axis(1, at=1, labels=c('All'))
boxplot(Good$Existing_Credit, at = 2, ylim = c(0.5,3.5), add = TRUE,
main="Boxplot of Variable: Existing Credit by Decision",
        xlab="Decision", ylab="Existing Credit", col="chartreuse", boxwex =
1.2)
axis(1, at=2, labels=c('Good'))
boxplot(Bad$Existing_Credit, at = 3, ylim = c(0.5,3.5), add = TRUE,
main="Boxplot of Variable: Existing Credit by Decision",
        xlab="Decision", ylab="Existing Credit", col="firebrick1", boxwex =
1.2)
axis(1, at=3, labels=c('Bad'))
abline(h=mean(mExisting_Credit), col="red")

### Kernel Plots


par(mfrow=c(3,3))

d1 = density(C_D$Months)
plot(d1,main = "Kernel Density:Months")
polygon(d1, col="red", border="darkblue")
```

```
d2 = density(C_D$Age)
plot(d2,main = "Kernel Density:Age")
polygon(d2, col="red", border="darkblue")

d3 = density(C_D$Credit_Amount)
plot(d3,main = "Kernel Density:Credit Amount")
polygon(d3, col="red", border="darkblue")

d4 = density(C_D$Residence_Since)
plot(d4,main = "Kernel Density:Residence Since")
polygon(d4, col="red", border="darkblue")

d5 = density(C_D$Installment_Rate)
plot(d5,main = "Kernel Density:Installment Rate")
polygon(d5, col="red", border="darkblue")

d6 = density(C_D$Existing_Credit)
plot(d6,main = "Kernel Density:Existing Credit")
polygon(d6, col="red", border="darkblue")

d7 = density(C_D$Dependents)
plot(d7,main = "Kernel Density:Dependents")
polygon(d7, col="red", border="darkblue")

### Histograms

graphics.off()


par(mfrow=c(3,3))

#Variable: Months

g1 = C_D$Months
m1 = mean(g1)
std1 = sqrt(var(g1))
h1 = hist(g1,col ="gray",breaks = 10, prob=TRUE,
          ylim = c(0,0.05),xlim = c(min(C_D$Months),max(C_D$Months)),xlab =
"No.of Months",
          main = "Histogram of Variable: Months")
c1 = curve(dnorm(x, mean=m1, sd=std1),
      col="red", lwd=2, add=TRUE, yaxt="n")


#Variable:Age


g2 = C_D$Age
m2 = mean(g2)
std2 = sqrt(var(g2))
h2 = hist(g2,col = "gray",breaks = 20, prob= TRUE,
          ylim = c(0,0.05),xlim = c(min(C_D$Age),max(C_D$Age)),xlab = "Age
in Years",
          main = "Histogram of Variable: Age")
c2 = curve(dnorm(x, mean=m2, sd=std2),
           col="red", lwd=2, add=TRUE, yaxt="n")
#Variable:Credit Amount

g3 = C_D$Credit_Amount
m3 = mean(g3)
std3 = sqrt(var(g3))
h3 = hist(g3,col = "gray",breaks = 20, prob= TRUE,
```

```
          ylim = c(0,0.00035),xlim =
c(min(C_D$Credit_Amount),max(C_D$Credit_Amount)),xlab = "Credit Amount",
          main = "Histogram of Variable: Credit Amount")
c3 = curve(dnorm(x, mean=m3, sd=std3),
           col="red", lwd=2, add=TRUE, yaxt="n")

#Variable: Residence Since

g4 = C_D$Residence_Since
m4 = mean(g4)
std4 = sqrt(var(g4))
h4 = hist(g4,col = "gray",breaks = 5,prob= TRUE,
          ylim = c(0,0.7),xlim =
c(min(C_D$Residence_Since),max(C_D$Residence_Since)),xlab =
"Residence_Since",
          main = "Histogram of Variable: Residence Since")
c4 = curve(dnorm(x, mean=m4, sd=std4),
           col="red", lwd=2, add=TRUE, yaxt="n")



#Variable: Installment Rate


g5 = C_D$Installment_Rate
m5 = mean(g5)
std5 = sqrt(var(g5))
h5 = hist(g5,col = "gray",breaks = 5,prob= TRUE,ylim = c(0,0.5),
          xlim =
c(min(C_D$Installment_Rate),max(C_D$Installment_Rate)),xlab = "Installment
Rate",
          main = "Histogram of Variable: Installment Rate")
c5 = curve(dnorm(x, mean=m5, sd=std5),
           col="red", lwd=2, add=TRUE, yaxt="n")

#Variable: Existing Credit


g6 = C_D$Existing_Credit
m6 = mean(g6)
std6 = sqrt(var(g6))
h6 = hist(g6,col = "gray",breaks = 5,prob= TRUE,ylim = c(0,1.5),
          xlim = c(min(C_D$Existing_Credit),max(C_D$Existing_Credit)),xlab
= "Existing Credit",
          main = "Histogram of Variable: Existing Credit")
c6 = curve(dnorm(x, mean=m6, sd=std6),
           col="red", lwd=2, add=TRUE, yaxt="n")


#Variable: Dependents


g7 = C_D$Dependents
m7 = mean(g7)
std7 = sqrt(var(g7))
h7 = hist(g7,col = "gray",breaks = 4,prob= TRUE,ylim = c(0,4.5),
          xlim = c(min(C_D$Dependents),max(C_D$Dependents)),xlab =
"Dependents",
          main = "Histogram of Variable: Dependents")
c7 = curve(dnorm(x, mean=m7, sd=std7),
           col="red", lwd=2, add=TRUE, yaxt="n")
```

```
#####Descriptive Statistics - Qualitative Variables

###Bar Plots

par(mfrow=c(2,3))

count.purpose = table(C_D$Credit_Purpose)
b1 = barplot(count.purpose, main="Barplot of Credit Purpose",
        xlab="Purpose of Loan",ylab = "Frequency",
        ylim = c(0,1000))

count.status = table(C_D$Account_Status)
b2 = barplot(count.status, main="Barplot of Account Status",
             xlab="Account Status",ylab = "Frequency",
             ylim = c(0,1000))

count.history = table(C_D$Credit_History)
b3 = barplot(count.history, main="Barplot of Credit History",
             xlab="Credit History",ylab = "Frequency",
             ylim = c(0,1000))


count.savings = table(C_D$Savings_Account)
b4 = barplot(count.savings, main="Barplot of Savings Account",
             xlab="Purpose of Loan",ylab = "Frequency",
             ylim = c(0,1000))



count.employ.since = table(C_D$Employed_Since)
b5 = barplot(count.employ.since, main="Barplot of Employement Duration",
             xlab="Employment Duration",ylab = "Frequency",
             ylim = c(0,1000))



count.status.sex = table(C_D$Status_and_Sex)
b6 = barplot(count.status.sex, main="Barplot of Status and Sex",
             xlab="Status and Sex",ylab = "Frequency",
             ylim = c(0,1000))

par(mfrow=c(2,3))

count.debtors = table(C_D$Other_Debtors)
b7 = barplot(count.history, main="Barplot of Other_Debtors",
             xlab="Other Debtors",ylab = "Frequency",
             ylim = c(0,1000))


count.other.installment.plans = table(C_D$Other_Installement_Plans)
b8 = barplot(count.other.installment.plans, main="Barplot of Other
Installment ",
             xlab="Other Installment Plans",ylab = "Frequency",
             ylim = c(0,1000))

count.housing = table(C_D$Housing_Type)
b9 = barplot(count.housing, main="Barplot of Housing Type",
             xlab="Employment Duration",ylab = "Frequency",
             ylim = c(0,1000))

count.property = table(C_D$Property)
b10 = barplot(count.property, main="Barplot of Property",
             xlab="Property",ylab = "Frequency",
```

```
              ylim = c(0,1000))

count.job.type = table(C_D$Job_Type)
b11 = barplot(count.job.type, main="Barplot of Job Type",
              xlab="Types of Job",ylab = "Frequency",
              ylim = c(0,1000))

count.telephone = table(C_D$Telephone)
b12 = barplot(count.telephone, main="Barplot of Telephone",
              xlab="Telephone", ylab = "Frequency",
              ylim = c(0,1000))

### Mosaic Plots

par(mfrow=c(3,2))

#Variable: Status_and_Sex

## A91 : male   : divorced/separated
## A92 : female : divorced/separated/married
## A93 : male   : single
## A94 : male   : married/widowed
## A95 : female : single

levels(C_D$Status_and_Sex) = c("M/DSe", "F/DSeM", "M/Si", "M/MW", "F/Si")
levels(C_D$Cost_Matrix) = c("Good", "Bad")

Status_Sex = table(C_D$Status_and_Sex, C_D$Cost_Matrix)

mosaicplot(Status_Sex, cex = 0.60, color = TRUE, border = "azure3")

#Variable: Job_Type

## A171 : unemployed/ unskilled  - non-resident
## A172 : unskilled - resident
## A173 : skilled employee / official
## A174 : management/ self-employed/highly qualified employee/ officer

levels(C_D$Job_Type) = c("Unemp/Uns/NR", "Uns/Res", "Skilled/Off",
"M/SE/HQE/O")
levels(C_D$Cost_Matrix) = c("Good", "Bad")

Job = table(C_D$Job_Type, C_D$Cost_Matrix)

mosaicplot(Job, cex = 0.60, color = TRUE, border = "azure3")


#Variable: Credit history

## A30 : no credits taken/all credits paid back duly
## A31 : all credits at this bank paid back duly
## A32 : existing credits paid back duly till now
## A33 : delay in paying off in the past
## A34 : critical account/other credits existing (not at this bank)

levels(C_D$Credit_History) = c("NCR/D", "Th.Bank.D","Ex.Cr.D.", "Delay",
"Cr.Acc.")
levels(C_D$Cost_Matrix) = c("Good", "Bad")

Credit_Hist= table(C_D$Credit_History, C_D$Cost_Matrix)

mosaicplot(Credit_Hist, cex = 0.60, color = TRUE, border = "azure3")
```

```
#Variable: Savings_Account

## A61 :          ... <  100 DM
## A62 :   100 <= ... <  500 DM
## A63 :   500 <= ... < 1000 DM
## A64 :          .. >= 1000 DM
## A65 :   unknown/ no savings account

levels(C_D$Savings_Account) = c("0-100", "100-500", "500-1000", "1000+",
"Unk/N.S.A")
levels(C_D$Cost_Matrix) = c("Good", "Bad")

Savings_Acc= table(C_D$Savings_Account, C_D$Cost_Matrix)

mosaicplot(Savings_Acc, cex = 0.60, color = TRUE, border = "azure3")

#Variable: Employed_Since

## A71 : unemployed
## A72 :       ... < 1 year
## A73 : 1  <= ... < 4 years
## A74 : 4  <= ... < 7 years
## A75 :       .. >= 7 years

help(mosaic)

levels(C_D$Employed_Since) = c("Unemp", "0-1", "1-4", "4-7", "7+")
levels(C_D$Cost_Matrix) = c("Good", "Bad")

Employed_Sin= table(C_D$Employed_Since, C_D$Cost_Matrix)

mosaicplot(Employed_Sin, cex = 0.60, color = TRUE, border = "azure3")

##### Pre-Processing

### Categorical Dummy Encoding

#Loading the data
dat=read.csv("http://archive.ics.uci.edu/ml/machine-learning-
databases/statlog/german/german.data", as.is=T,header=F,sep=" ")

dir.create(paste(home,'Model',sep=''))

setwd(paste(home,'Model',sep=''))

dir.create("preproc_data")


#Categorical columns
grep("A",dat)


dat_cat=dat[,grep("A",dat)]

###Dummy Encoding

## Flags creation for categorical columns with 2 unique values
# After looking into data dictionary, we know which ones are such columns.
Encoding them as 0 and 1

unique(dat_cat[,ncol(dat_cat)-1])
```

```
dat_cat[,ncol(dat_cat)-1]=ifelse((dat_cat[,ncol(dat_cat)-
1])==unique(dat_cat[,ncol(dat_cat)-1])[[1]],1,0)

unique(dat_cat[,ncol(dat_cat)])

dat_cat[,ncol(dat_cat)]=ifelse((dat_cat[,ncol(dat_cat)])==unique(dat_cat[,n
col(dat_cat)])[[1]],1,0)


## Flags creation for categorical dataset with more than 2 values



library(caret)
dummies = dummyVars( ~ ., data = dat_cat[,1:(ncol(dat_cat)-2)])
category_data_dummies=as.data.frame(predict(dummies, newdata =
dat_cat[,1:(ncol(dat_cat)-2)]))
dim(category_data_dummies)
head(category_data_dummies)

##Joining both kind of variables in one dataset
data_cat=cbind(dat_cat[,c(ncol(dat_cat)-
1,ncol(dat_cat))],category_data_dummies)

#Saving
write.csv(data_cat,"preproc_data/cat_data.csv",row.names=F)

### Continuous Variable Imputing and Scaling


dat_cont=dat[, setdiff(1:20,grep("A",dat))]



##Handling Outliers
## Function for replacing outlier on Z value>3 with NA
remove_outliers = function(x, na.rm = TRUE, ...)
{
  VarMean = mean(x, na.rm = na.rm, ...)
  VarSD = sd(x,na.rm = na.rm, ...)
  y = x
  y[abs((x - VarMean)/VarSD) >3] = NA
  y
}


##applying above function

dat_cont= sapply(dat_cont,function(x)remove_outliers(x))


##Checking number of NAs
table(is.na(dat_cont))


##Imputing NA with mean of the Variables


means = colMeans(dat_cont[,colnames(dat_cont)],na.rm=T)
for(idv in colnames(dat_cont)) dat_cont[is.na(dat_cont[,idv]),idv]
=means[idv]
```

```
##Checking if any NAs left
table(is.na(dat_cont))



###Scaling of Continuous variables

##MinMaxScaling (-1 to 1)

min_max_scaling=function(col)((col-min(col))/(max(col)-min(col))*2-1)

##check
min_max_scaling(c(1,2,3,4))
# [1] -1.0000000 -0.3333333  0.3333333  1.0000000

dat_cont=as.data.frame(dat_cont)
dat_cont2 = sapply(dat_cont,function(col)min_max_scaling(col))

##saving
write.csv(dat_cont2,"preproc_data/cont_data.csv",row.names=F)


### Merging and Sampling

colnames(dat)[21]=paste('dv')

dat$id=rownames(dat)

dat_cat = read.csv("preproc_data/cat_data.csv",as.is=T)

dat_cont2 = read.csv("preproc_data/cont_data.csv",as.is=T)


##Combining all datasets
data_final=cbind(dat[,c("id","dv")],dat_cont2,dat_cat)

##Changing Target variable's values from 1 and 2  to 0 and 1 resp. as it is
more comprehensible form for logR algorithm
data_final$dv=data_final$dv-1



### Sampling into training, validation and test datasets.

## We would want to have similar ratio of goods vs bads in all the samples
to maintain for unbiased training and testing. CARET package provides this
functionality with control over target variable.
library(caret)

set.seed(3456)

##dividing rows in 5 folds with control over target variable
devIndex = as.data.frame(createFolds(data_final$dv, k=5,  list = TRUE))

data_div_1= data_final[ devIndex$Fold1, c("id","dv")]
data_div_2= data_final[ devIndex$Fold2, c("id","dv")]
data_div_3= data_final[ devIndex$Fold3, c("id","dv")]
data_div_4= data_final[ devIndex$Fold4, c("id","dv")]
data_div_5= data_final[ devIndex$Fold5, c("id","dv")]


md_rows = rbind(data_div_1,data_div_2,data_div_3) ## Training dataset 60%
```

```
val_rows = data_div_4                                   ## Validation dataset 20%
test_rows = data_div_5                                  ## Test dataset 20%

data_md = merge( data_final , md_rows,by=c('id','dv'))
data_val = merge( data_final , val_rows,by=c('id','dv'))
data_test = merge( data_final , test_rows,by=c('id','dv'))

data_md$sample="train"
data_val$sample="val"
data_test$sample='test'


full_dv_data= rbind(data_md[,(colnames(data_md))],
data_val[,(colnames(data_val))], data_test[,(colnames(data_test))])
class(full_dv_data$id)


full_dv_data$id=as.numeric(full_dv_data$id)
class(full_dv_data$id)
head(full_dv_data)


full_dv_data=full_dv_data[order(full_dv_data[,'id']),]

dim(full_dv_data)
length(unique(full_dv_data$id))

write.csv(full_dv_data,"preproc_data/dv.csv",row.names=F)
data_final=full_dv_data




###removing 0 variance predictors
data_final = data_final[sapply(data_final, function(x)
length(levels(factor(x,exclude=NULL)))>1)]

nzv=nearZeroVar(data_final)
nzv
#11 16 17 23 26 27 29 34 46 53 58


if(length(nzv)>0) data_final=data_final[,-nzv]

###Separating data sets and removing Sample and dv_binary column

##Checking for the target variable distribution in over all dataset
nrow(data_final[data_final$dv==0,])/nrow(data_final)*100
#70

data_final_train=data_final[data_final$sample=='train',c(1:ncol(data_final)
-1)]
nrow(data_final_train[data_final_train$dv==0,])/nrow(data_final_train)*100
#70.333

data_final_val=data_final[data_final$sample=='val',c(1:ncol(data_final)-1)]
nrow(data_final_val[data_final_val$dv==0,])/nrow(data_final_val)*100
#71.5

data_final_test=data_final[data_final$sample=='test',c(1:ncol(data_final)-
1)]
nrow(data_final_test[data_final_test$dv==0,])/nrow(data_final_test)*100
#67.5
```

```
### taking output in CSV format
write.csv(data_final_train,"preproc_data/train.csv",row.names=F)
write.csv(data_final_val,"preproc_data/val.csv",row.names=F)
write.csv(data_final_test,"preproc_data/test.csv",row.names=F)


##### Cross Validation and Bagging

### Creating samples using methodology of cross validation

##creating 5 folds manually (instead of using caret) out of training data
set

dat_dv=training_data[,c("dv","id")]



###Creating folds manually with control on dv

set.seed(243)
rand_num=c(sample(c(1:length(which(dat_dv$dv==0)))),sample(c(1:length(which
(dat_dv$dv==1))))))
dat_dv2=cbind(dat_dv[order(dat_dv[,'dv']),],rand_num)

##separating data on value of target variable
dat_dv2_0=dat_dv2[dat_dv2$dv==0,'rand_num']
dat_dv2_0=as.data.frame(dat_dv2_0)
dat_dv2_1=dat_dv2[dat_dv2$dv==1,'rand_num']
dat_dv2_1=as.data.frame(dat_dv2_1)


##Creating bins or folds with division or rows in 5 equal parts
#For dataset with 0 value of target variable
breaks=unique(quantile(dat_dv2_0[,1], probs = seq(0, 1, by= 0.2)))
dat_dv2_0[,paste(colnames(dat_dv2_0[1]),"bin",sep="_")] =
cut(dat_dv2_0[,1], breaks,include.lowest=TRUE
,labels=c(1:ifelse(length(breaks)>1,(length(breaks) - 1),length(breaks))))
colnames(dat_dv2_0)[1]=paste("rand_num")
colnames(dat_dv2_0)[2]=paste("bin")

#For dataset with 1 value of target variable
breaks=unique(quantile(dat_dv2_1[,1], probs = seq(0, 1, by= 0.2)))
dat_dv2_1[,paste(colnames(dat_dv2_1[1]),"bin",sep="_")] =
cut(dat_dv2_1[,1], breaks,include.lowest=TRUE
,labels=c(1:ifelse(length(breaks)>1,(length(breaks) - 1),length(breaks))))
colnames(dat_dv2_1)[1]=paste("rand_num")
colnames(dat_dv2_1)[2]=paste("bin")

##Recombining
dat_dv2_bin=rbind(dat_dv2_0,dat_dv2_1)


dat_dv2=cbind(dat_dv2,dat_dv2_bin)
dat_dv2=dat_dv2[,c(1,2,5)]
dat_dv2=dat_dv2[order(dat_dv2[,'id']),]

##creating 5 different datasets using the combination of the folds with one
left out
    for(j in 1:5)
    {
```

```
    assign(paste("data_fold_",j,sep=''), dat_dv2[ dat_dv2$bin!=j,
c("id","dv")])
    }


##creating a list of all the datasets
dflist = list(data_fold_1,data_fold_2,data_fold_3,data_fold_4,data_fold_5)

max_row_num=as.integer(max(as.character(lapply(dflist,function(x)nrow(x))))
)


###Creating 5 bootstrap samples


    for(k in 1:5)
    {
        for(l in 1:5)
        {
            set.seed(243)
            ##taking 67 to 70% of original data and rest of the samples
will be repeated in each bag
            a=dflist[[k]][sample(1:nrow(dflist[[k]]),floor(runif(1, 67,
70)*nrow(dflist[[k]])/100)),]
            set.seed(243)
            b=rbind(a,a[sample(1:nrow(a),max_row_num-nrow(a)),])
            b=merge(training_data,b,by=c('id','dv'))
            b=b[order(b[,'id']),]
            write.csv(b,
paste("preproc_data/dfold",k,"_bag",l,".csv",sep=''),row.names=F)

        }
        ##also saving each fold created from cross validation method

write.csv(merge(training_data,dflist[[k]],by=c('id','dv')),paste("preproc_d
ata/dfold",k,".csv",sep=''),row.names=F)
    }


##### Classifier Codes

### Logistic Regression

preproc_dir=paste(home,'Model',"/","preproc_data",sep='')
dir.create("candidates_val")
candidates_val=paste(home,'Model',"/","candidates_val",sep='')
dir.create("candidates_test")
candidates_test=paste(home,'Model',"/","candidates_test",sep='')

##loading the names of file in the directory
setwd(preproc_dir)
myFiles2 = list.files(pattern="dfold.*csv")
myFiles = myFiles2


##### Logistic Regrssion in parallel processing
library(stepPlr)
library(doSNOW)
library(foreach)

setwd(preproc_dir)
```

```
##creating a matrix of all the possible combinations of parameters we want
to run
lr_parameters=expand.grid(lambda = 2^seq(-19,6,2), cp = c("aic","bic"))

##For user's ease, a progress bar would show how many models have been
created
pb = winProgressBar(title = paste("LogR Progress Bar"), min = 0, max =
length(myFiles), width = 400)


for (j in 1:length(myFiles))
    {

        a=read.csv(paste(preproc_dir,'/',myFiles[j],sep=''),as.is=T)
        colnames(a)=paste(gsub(".","",colnames(a),fixed=TRUE))    ##fixing
column names
        val_data= read.csv(paste(preproc_dir,'/',"val.csv",sep=''),as.is=T)

colnames(val_data)=paste(gsub(".","",colnames(val_data),fixed=TRUE))
        val_data=val_data[,colnames(a)]
        test_data=
read.csv(paste(preproc_dir,'/',"test.csv",sep=''),as.is=T)

colnames(test_data)=paste(gsub(".","",colnames(test_data),fixed=TRUE))
        test_data=test_data[,colnames(a)]


        cl=makeCluster(3) #change the 3 to your number of CPU cores
        registerDoSNOW(cl)

        lr_model = foreach(k = 1:nrow(lr_parameters),.packages="stepPlr")
%dopar%
                {
                lr_model = plr(x = a[,-c(1:3)],y =
as.numeric(a[,2]),lambda=lr_parameters[k,1], cp=lr_parameters[k,2])
##building models
                }
        stopCluster(cl)

        cl=makeCluster(3) #change the 3 to your number of CPU cores
        registerDoSNOW(cl)

        lr_val = foreach(k = 1:nrow(lr_parameters), .combine='cbind',
.packages="stepPlr") %dopar%
                {
                predict(lr_model[[k]],val_data[,-c(1:3)],type="response")
##using models to predict validation sample values
                }
        stopCluster(cl)

        val_out_local=as.data.frame(lr_val)
        colnames(val_out_local) =
paste(gsub(".","_",colnames(lr_val),fixed=TRUE)) ##fixing column names
        colnames(val_out_local) =
paste(gsub("result",paste("lr_val",j,sep="_"),colnames(lr_val),fixed=TRUE))

        if(j==1)
            {lr_val_full=val_out_local
        }else
        {lr_val_full=cbind(lr_val_full,val_out_local)}

        cl=makeCluster(3) #change the 3 to your number of CPU cores
        registerDoSNOW(cl)
```

```
        lr_test = foreach(k = 1:nrow(lr_parameters), .combine='cbind',
.packages="stepPlr") %dopar%
                {
                predict(lr_model[[k]],test_data[,-c(1:3)],type="response")
##using models to predict test sample values
                }
        stopCluster(cl)

        test_out_local=as.data.frame(lr_test)
        colnames(test_out_local) =
paste(gsub(".","_",colnames(lr_test),fixed=TRUE))    ##fixing column names
        colnames(test_out_local) =
paste(gsub("result",paste("lr_test",j,sep="_"),colnames(lr_test),fixed=TRUE
))
        if(j==1)
            {lr_test_full=test_out_local
        }else
        {lr_test_full=cbind(lr_test_full,test_out_local)}
    setWinProgressBar(pb,j, title=paste(" LogR :",
round(j/length(myFiles)*100, 0),"% done"))

    }

    setwd(candidates_val)
    write.csv(lr_val_full,"lr_val.csv",row.names=F)

    setwd(candidates_test)
    write.csv(lr_test_full,"lr_test.csv",row.names=F)

    setwd(home)
close(pb)

### Random Forest

preproc_dir=paste(home,'Model',"/","preproc_data",sep='')
dir.create("candidates_val")
candidates_val=paste(home,'Model',"/","candidates_val",sep='')
dir.create("candidates_test")
candidates_test=paste(home,'Model',"/","candidates_test",sep='')



setwd(preproc_dir)
myFiles2 = list.files(pattern="dfold.*csv")
myFiles = myFiles2 #[grep("g1|g2|g3|g4",myFiles2)]

a=read.csv(paste(getwd(),'/',myFiles[1],sep=''),as.is=T)

##creating a matrix of all the possible combinations of parameters we want
to run
rf_parameters=expand.grid(ntree = seq(100,1800,100), mtry =
c(floor((ncol(a)-3)/6),floor((ncol(a)-3)/3),floor((ncol(a)-3)*1.5)),
nodesize=5)



library(randomForest)
library(doSNOW)
library(foreach)

##For user's ease, a progress bar would show how many models have been
created
```

```
pb = winProgressBar(title = paste("RF Progress Bar"), min = 0, max =
length(myFiles), width = 400)

for(j in 1:length(myFiles))
    {
        a=read.csv(paste(preproc_dir,'/',myFiles[j],sep=''),as.is=T)
        colnames(a)=paste(gsub(".","",colnames(a),fixed=TRUE))
        a$dv=gsub('0','r',a$dv)
        a$dv=gsub('1','s',a$dv)
        val_data= read.csv(paste(preproc_dir,'/',"val.csv",sep=''),as.is=T)

colnames(val_data)=paste(gsub(".","",colnames(val_data),fixed=TRUE))
        val_data=val_data[,colnames(a)]
        test_data=
read.csv(paste(preproc_dir,'/',"test.csv",sep=''),as.is=T)

colnames(test_data)=paste(gsub(".","",colnames(test_data),fixed=TRUE))
        test_data=test_data[,colnames(a)]



        cl=makeCluster(3) #change the 2 to your number of CPU cores
        registerDoSNOW(cl)

        rf_model = foreach(k =
1:nrow(rf_parameters),.packages="randomForest") %dopar% {
                            randomForest(x = a[,-c(1:3)],y =
as.factor(a[,2]),
ntree=rf_parameters[k,1],mtry=rf_parameters[k,2],nodesize=rf_parameters[k,3
])
                        }
        stopCluster(cl)

        cl=makeCluster(2) #change the 2 to your number of CPU cores
        registerDoSNOW(cl)

        rf_val = foreach(k = 1:nrow(rf_parameters), .combine='cbind',
.packages="randomForest") %dopar%
                {
                predict(rf_model[[k]], val_data[,-c(1:3)], type="prob")[,2]
##using models to predict validation sample values
                }
        stopCluster(cl)

        val_out_local=as.data.frame(rf_val)
        colnames(val_out_local) =
paste(gsub(".","_",colnames(rf_val),fixed=TRUE))
        colnames(val_out_local) =
paste(gsub("result",paste("rf_val",j,sep="_"),colnames(rf_val),fixed=TRUE))
        if(j==1)
            {rf_val_full=val_out_local
        }else
        {rf_val_full=cbind(rf_val_full,val_out_local)}

        cl=makeCluster(2) #change the 2 to your number of CPU cores
        registerDoSNOW(cl)

        rf_test = foreach(k = 1:nrow(rf_parameters), .combine='cbind',
.packages="randomForest") %dopar%
                {
                predict(rf_model[[k]], test_data[,-c(1:3)],
type="prob")[,2] ##using models to predict test sample values
                }
```

```
        stopCluster(cl)

        test_out_local=as.data.frame(rf_test)
        colnames(test_out_local) =
paste(gsub(".","_",colnames(rf_test),fixed=TRUE))
        colnames(test_out_local) =
paste(gsub("result",paste("rf_test",j,sep="_"),colnames(rf_test),fixed=TRUE
))
        if(j==1)
            {rf_test_full=test_out_local
        }else
        {rf_test_full=cbind(rf_test_full,test_out_local)}
    setWinProgressBar(pb,j, title=paste(" RF :",
round(j/length(myFiles)*100, 0),"% done"))

    }

    setwd(candidates_val)
    write.csv(rf_val_full,"rf_val.csv",row.names=F)

    setwd(candidates_test)
    write.csv(rf_test_full,"rf_test.csv",row.names=F)

close(pb)

### Artificial Neural Network

preproc_dir=paste(home,'Model',"/","preproc_data",sep='')
dir.create("candidates_val")
candidates_val=paste(home,'Model',"/","candidates_val",sep='')
dir.create("candidates_test")
candidates_test=paste(home,'Model',"/","candidates_test",sep='')



setwd(preproc_dir)
myFiles2 = list.files(pattern="dfold.*csv")
myFiles = myFiles2 # [grep("g1|g2|g3|g4",myFiles2)]


#### ANN Model


library(nnet)
library(doSNOW)
library(foreach)


##creating a matrix of all the possible combinations of parameters we want
to run
ann_parameters=expand.grid(size =seq(1,15,2), decay=10^seq(-
4,0,2),maxit=c(100,200,1000))

##For user's ease, a progress bar would show how many models have been
created

pb = winProgressBar(title = paste("ANN Progress Bar"), min = 0, max =
length(myFiles), width = 400)



for (j in 1:length(myFiles))
```

```
    {

        a=read.csv(paste(preproc_dir,'/',myFiles[j],sep=''),as.is=T)
        colnames(a)=paste(gsub(".","",colnames(a),fixed=TRUE)) ##fixing
column names
        val_data= read.csv(paste(preproc_dir,'/',"val.csv",sep=''),as.is=T)

colnames(val_data)=paste(gsub(".","",colnames(val_data),fixed=TRUE))
        val_data=val_data[,colnames(a)]
        test_data=
read.csv(paste(preproc_dir,'/',"test.csv",sep=''),as.is=T)

colnames(test_data)=paste(gsub(".","",colnames(test_data),fixed=TRUE))
        test_data=test_data[,colnames(a)]


        cl=makeCluster(3) #change the 2 to your number of CPU cores
        registerDoSNOW(cl)

        nn_model = foreach(k = 1:nrow(ann_parameters),.packages="nnet")
%dopar%
                    {
                    nnet(x = a[,-c(1:3)], y = (a[,2]),
size=ann_parameters[k,1], decay=ann_parameters[k,2],
maxit=ann_parameters[k,3]) ##building models
                    }
        stopCluster(cl)

        cl=makeCluster(2) #change the 2 to your number of CPU cores
        registerDoSNOW(cl)

        ann_val = foreach(k = 1:nrow(ann_parameters), .combine='cbind',
.packages="nnet") %dopar%
                    {
                    predict(nn_model[[k]],val_data[,-c(1:3)])  ##using
models to predict validation sample values
                    }
        stopCluster(cl)

        val_out_local=as.data.frame(ann_val)

z1=paste(expand.grid(l=paste("ann_val_",j,sep=""),m=c(1:nrow(ann_parameters
)))[,1],expand.grid(l=paste("ann_val_",j,sep=""),m=c(1:nrow(ann_parameters)
))[,2],sep="_")
        colnames(val_out_local) = paste(z1) ##fixing column names

        if(j==1)
            {ann_val_full=val_out_local
        }else
        {ann_val_full=cbind(ann_val_full,val_out_local)}

        cl=makeCluster(2) #change the 2 to your number of CPU cores
        registerDoSNOW(cl)

        ann_test = foreach(k = 1:nrow(ann_parameters), .combine='cbind',
.packages="nnet") %dopar%
                    {
                    predict(nn_model[[k]],test_data[,-c(1:3)]) ##using
models to predict test sample values
                    }
        stopCluster(cl)

        test_out_local=as.data.frame(ann_test)
```

```
z2=paste(expand.grid(l=paste("ann_test_",j,sep=""),m=c(1:nrow(ann_parameter
s)))[,1],expand.grid(l=paste("ann_test_",j,sep=""),m=c(1:nrow(ann_parameter
s)))[,2],sep="_")
        colnames(test_out_local) = paste(z2) ##fixing column names
        if(j==1)
            {ann_test_full=test_out_local
        }else
        {ann_test_full=cbind(ann_test_full,test_out_local)}
    setWinProgressBar(pb,j, title=paste(" ANN :",
round(j/length(myFiles)*100, 0),"% done"))

    }

    setwd(candidates_val)
    write.csv(ann_val_full,"ann_val.csv",row.names=F)

    setwd(candidates_test)
    write.csv(ann_test_full,"ann_test.csv",row.names=F)

    setwd(home)
    close(pb)

##### Compilation

preproc_dir=paste(home,'Model',"/","preproc_data",sep='')

candidates_val=paste(home,'Model',"/","candidates_val",sep='')

candidates_test=paste(home,'Model',"/","candidates_test",sep='')

setwd(candidates_val)
myFiles = list.files(pattern="*csv")

library(plyr)
##Importing all the model csv files
import = llply(myFiles, read.csv)
## Mass cbind
b=do.call("cbind", import)


setwd(preproc_dir)
c=read.csv("val.csv",as.is=T)
b=cbind(c[,c("id","dv")],b)
setwd(candidates_val)

#Saving all
write.csv(b,"all_val.csv",row.names=F)

#in test samples column names have "test" string as against "val" in
validation samples
col=gsub('val','test',colnames(b))


setwd(candidates_test)
myFiles = list.files(pattern="*csv")


import = llply(myFiles, read.csv)
b=do.call("cbind", import)

setwd(preproc_dir)
c=read.csv("test.csv",as.is=T)
```

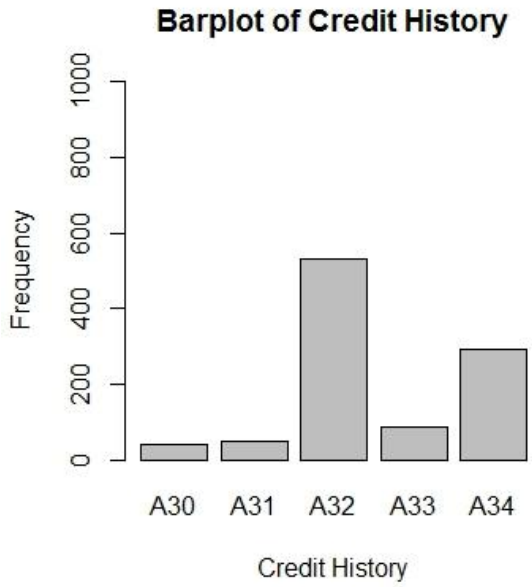
```
b=cbind(c[,c("id","dv")],b)
setwd(candidates_test)
b=b[,col]
write.csv(b,"all_test.csv",row.names=F)


##### Removing Correlation

df1  = read.csv("all_val.csv",as.is=T)
dim(df1)
col=colnames(df1)

library('caret')
#Removing Models with near zero variances

nzv=nearZeroVar(df1)
if (length(nzv)>0) df1=df1[,-nzv]

df2 = cor(df1)
df2[!lower.tri(df2)] = 0

#Taking only those models which are highly correlated with other models
a=apply(df2,2,function(x) any(x > 0.85))
a[1]=FALSE
a[2]=FALSE

#Removing them
reduced_Data = df1[,!a]

dim(reduced_Data)
write.csv(reduced_Data,"cor_removed_val.csv",row.names=F)


setwd(candidates_test)

df1  = read.csv("all_test.csv",as.is=T)
col=gsub('val','test',col)
names(a)=paste(gsub('val','test',names(a)))
df1=df1[,col]
dim(df1)
 if (length(nzv)>0) df1=df1[,-nzv]
reduced_Data = df1[,!a]
dim(reduced_Data)
write.csv(reduced_Data,"cor_removed_test.csv",row.names=F)




###### Ensemble Run and Results

### Stacking Ensemble

preproc_dir=paste(home,'Model',"/","preproc_data",sep='')

candidates_val=paste(home,'Model',"/","candidates_val",sep='')

candidates_test=paste(home,'Model',"/","candidates_test",sep='')


dir.create("ensemble_results")

ensemble_results=paste(home,'Model',"/","ensemble_results",sep='')
```

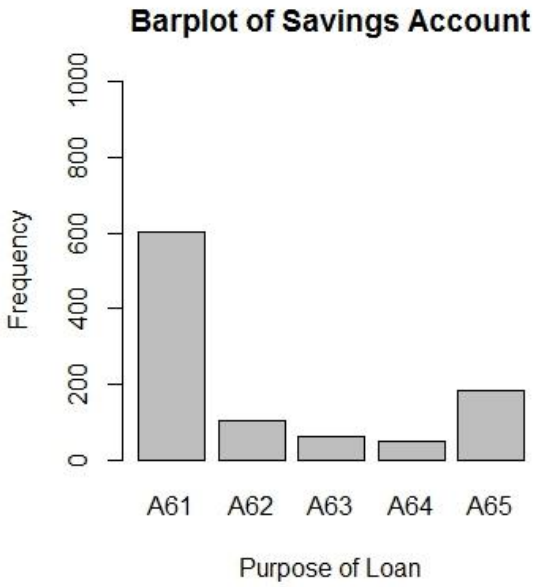
```
setwd(ensemble_results)

setwd(candidates_val)


ensembleSource_val = "cor_removed_val.csv"


### Final Result Compiler

preproc_dir=paste(home,'Model',"/","preproc_data",sep='')

candidates_val=paste(home,'Model',"/","candidates_val",sep='')

candidates_test=paste(home,'Model',"/","candidates_test",sep='')

dir.create("ensemble_results")

ensemble_results=paste(home,'Model',"/","ensemble_results",sep='')

library(verification)



## AUC
auc = function (obs, pred)
{
  out = array()
  for(i in 1:ncol(pred)){
    out[i] = (roc.area(as.numeric(obs), as.numeric(pred[,i]))$A)
    }
  out
}
```

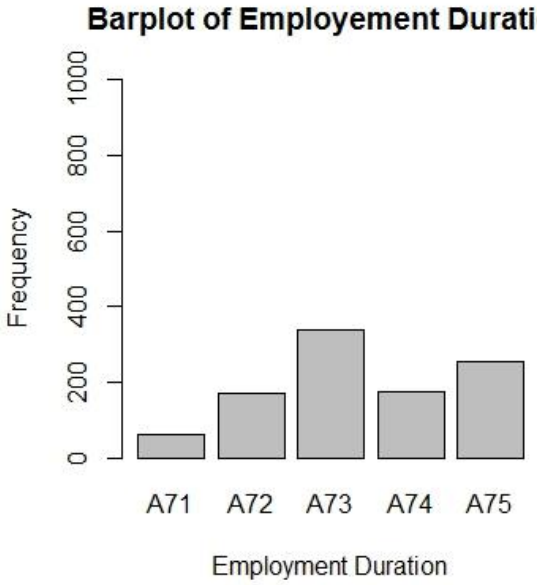
A.2 Complete list of plots (Descriptive Statistics)

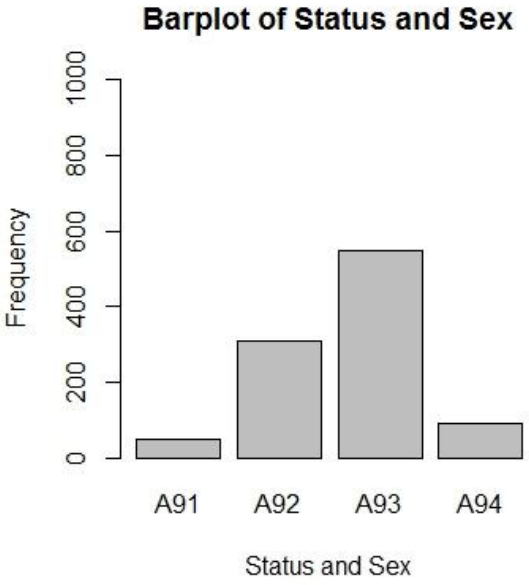
Categorical Variables

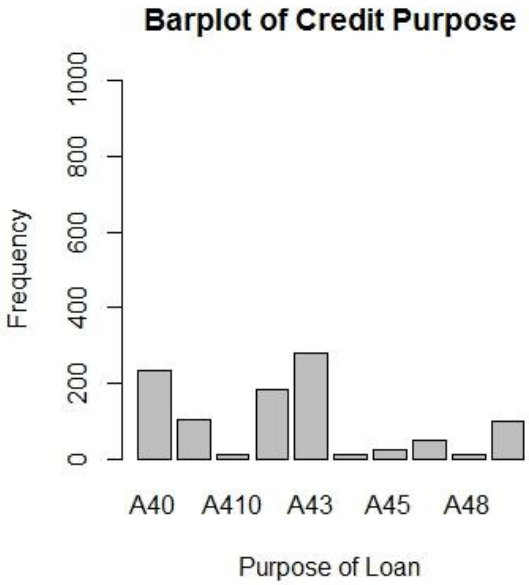
Bar Plots

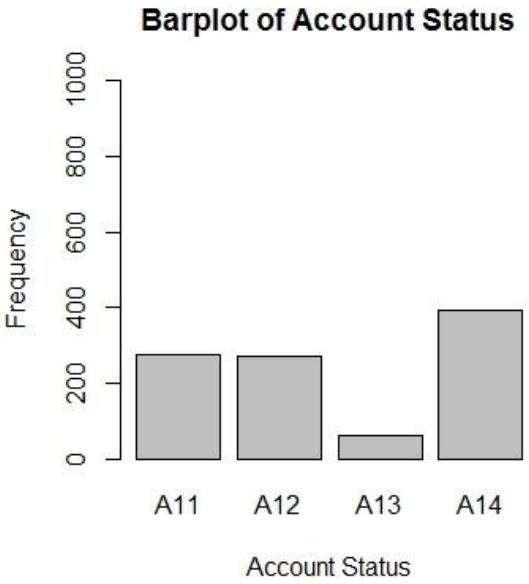


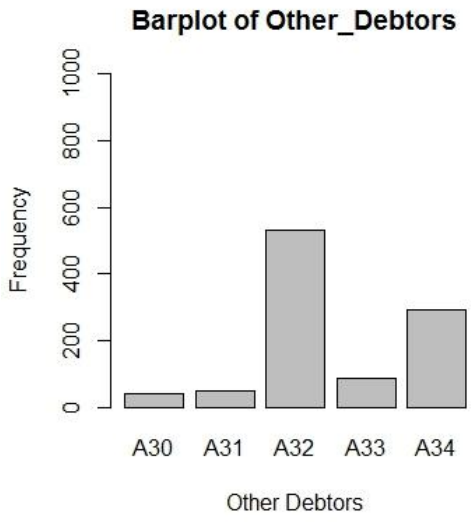


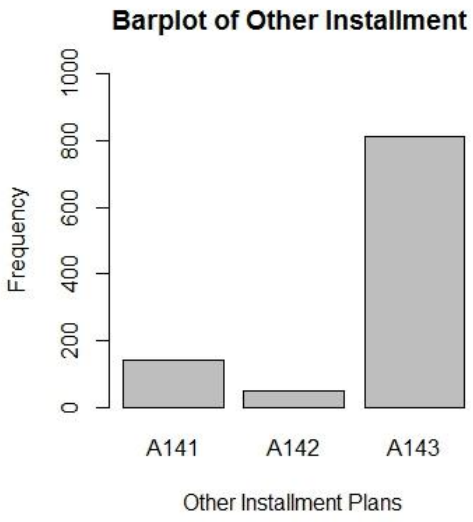

Barplot of Employement Duration
Frequency
0
200
400
600
800
1000
A71
A72
A73
A74
A75
Employment Duration

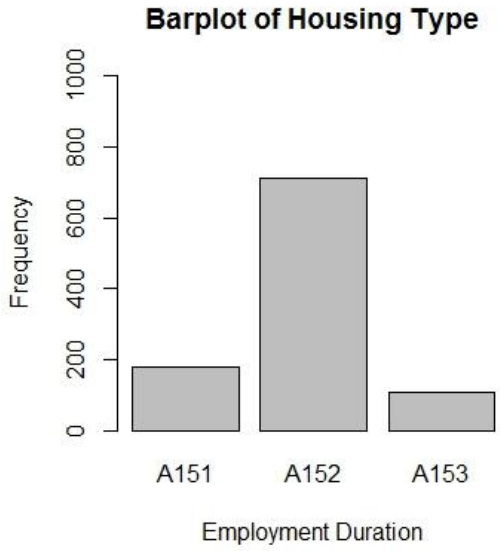

Barplot of Status and Sex
Frequency
0
200
400
600
800
1000
A91
A92
A93
A94
Status and Sex

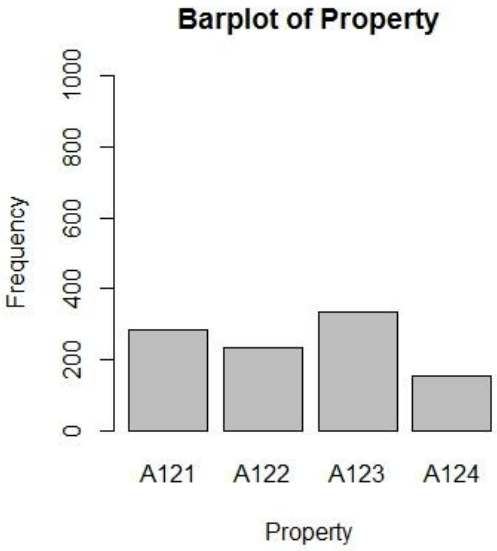

Barplot of Credit Purpose
Frequency
0
200
400
600
800
1000
A40
A410
A43
A45
A48
Purpose of Loan

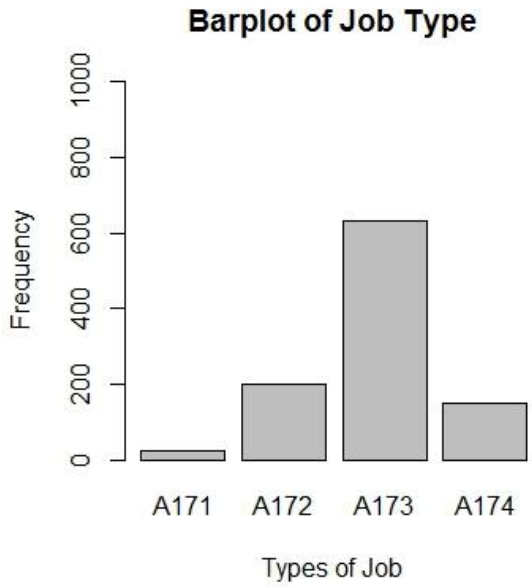

Barplot of Account Status
Frequency
0
200
400
600
800
1000
A11
A12
A13
A14
Account Status

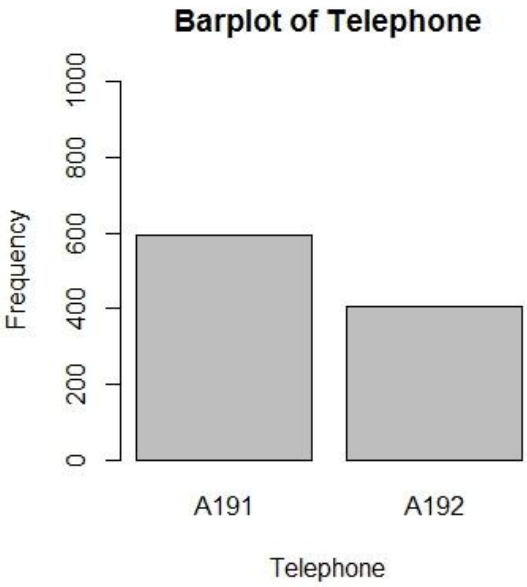

Barplot of Other_Debtors
Frequency
0
200
400
600
800
1000
A30
A31
A32
A33
A34
Other Debtors


Barplot of Other Installment
Frequency
0
200
400
600
800
1000
A141
A142
A143
Other Installment Plans

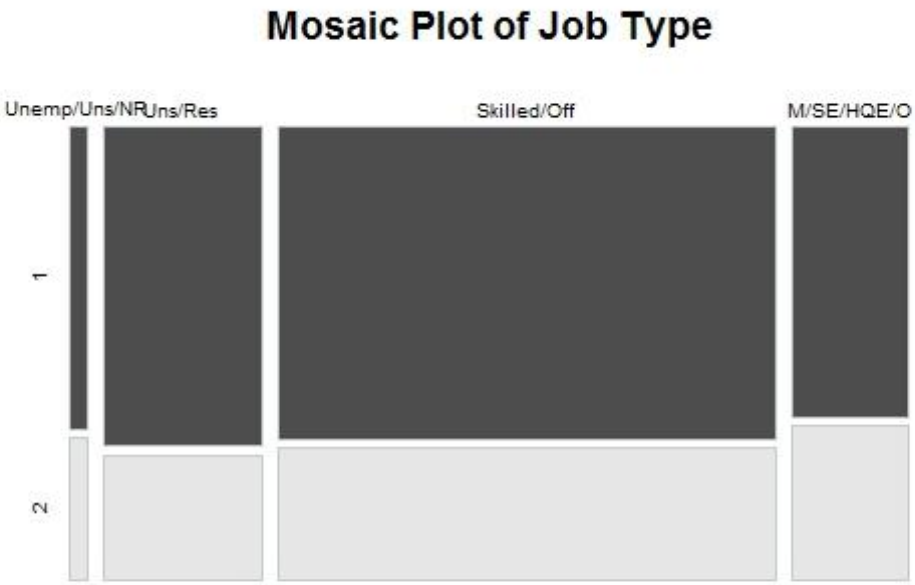

Barplot of Housing Type
Frequency
0
200
400
600
800
1000
A151
A152
A153
Employment Duration

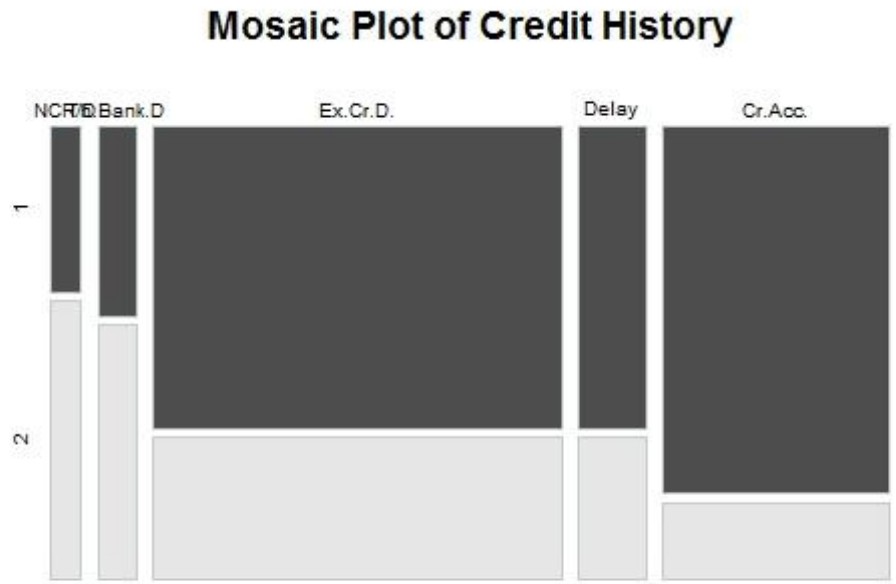

Barplot of Property
Frequency
0
200
400
600
800
1000
A121
A122
A123
A124
Property


Barplot of Job Type
Frequency
0
200
400
600
800
1000
A171
A172
A173
A174
Types of Job

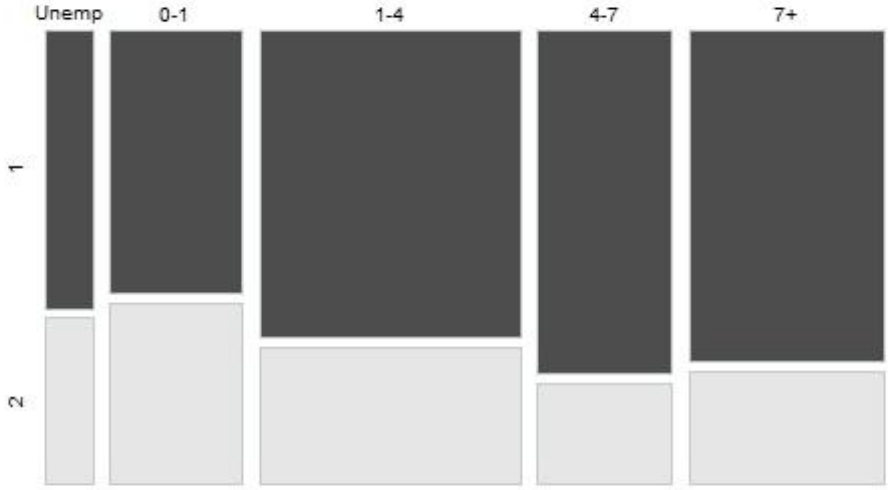

Barplot of Telephone
Frequency
0
200
400
600
800
1000
A191
A192
Telephone

Mosaic Plots

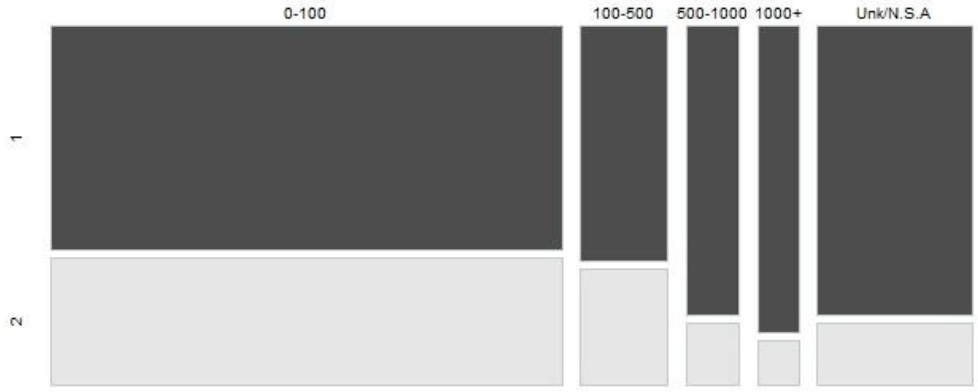




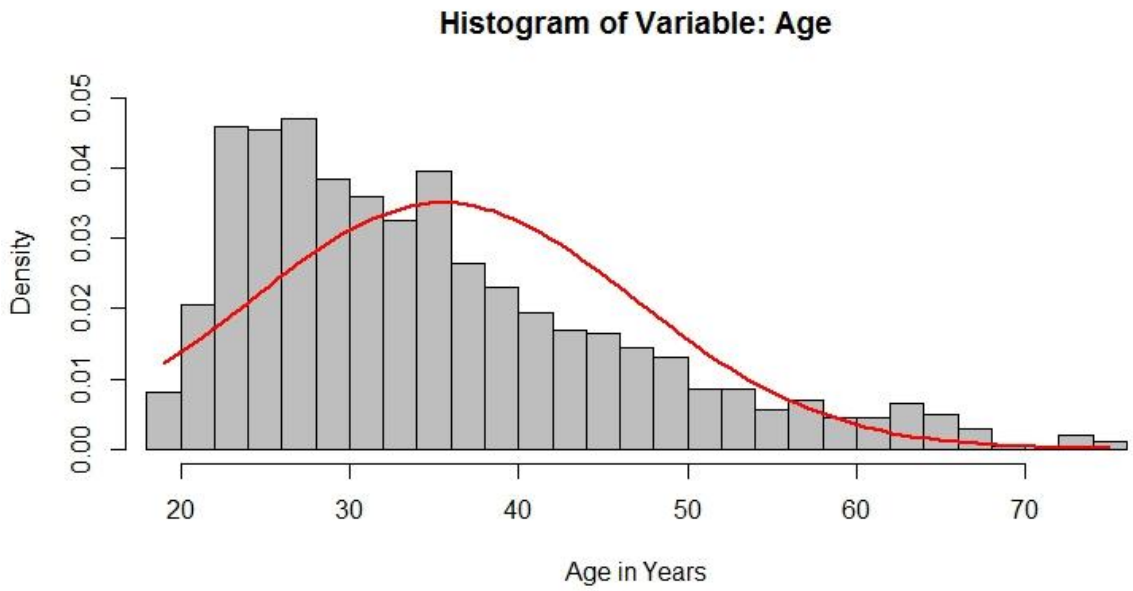
Mosaic Plot of Employment Duration

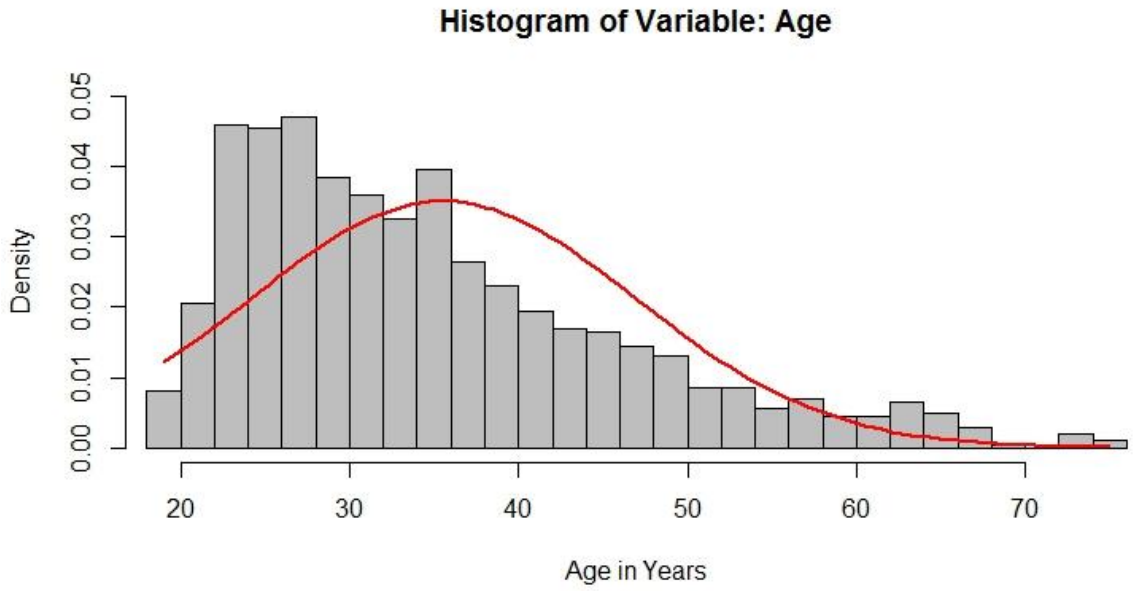


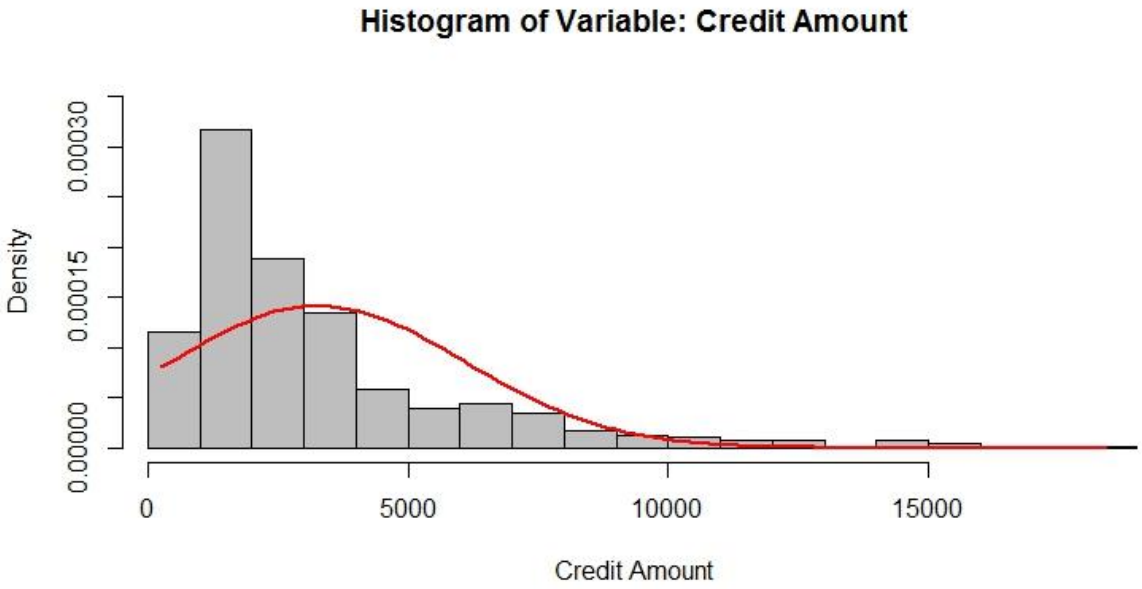
Mosaic Plot of Savings Account

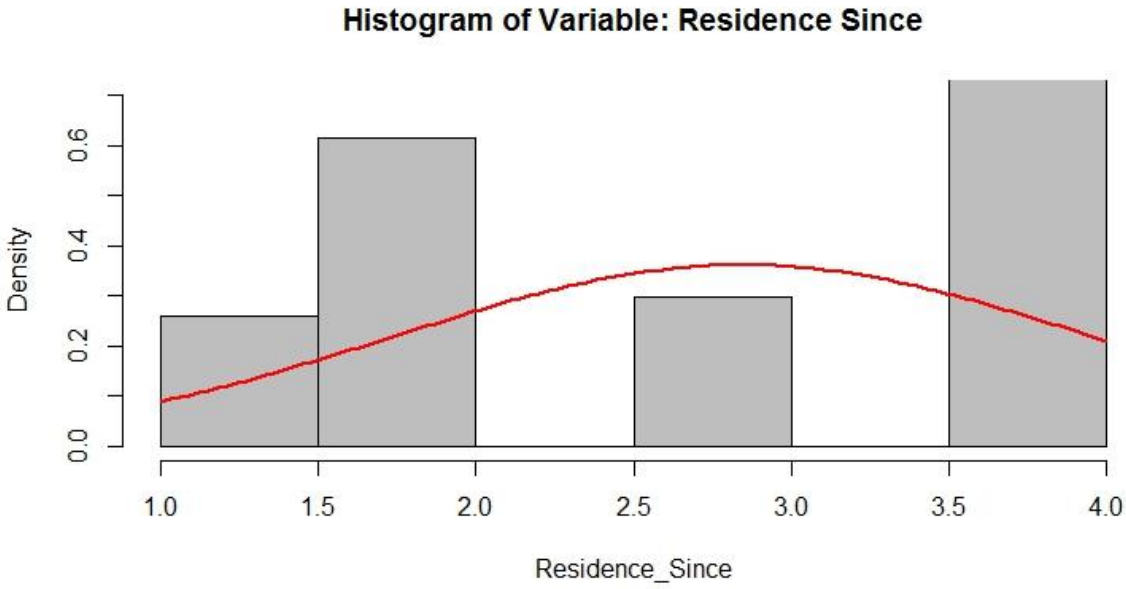


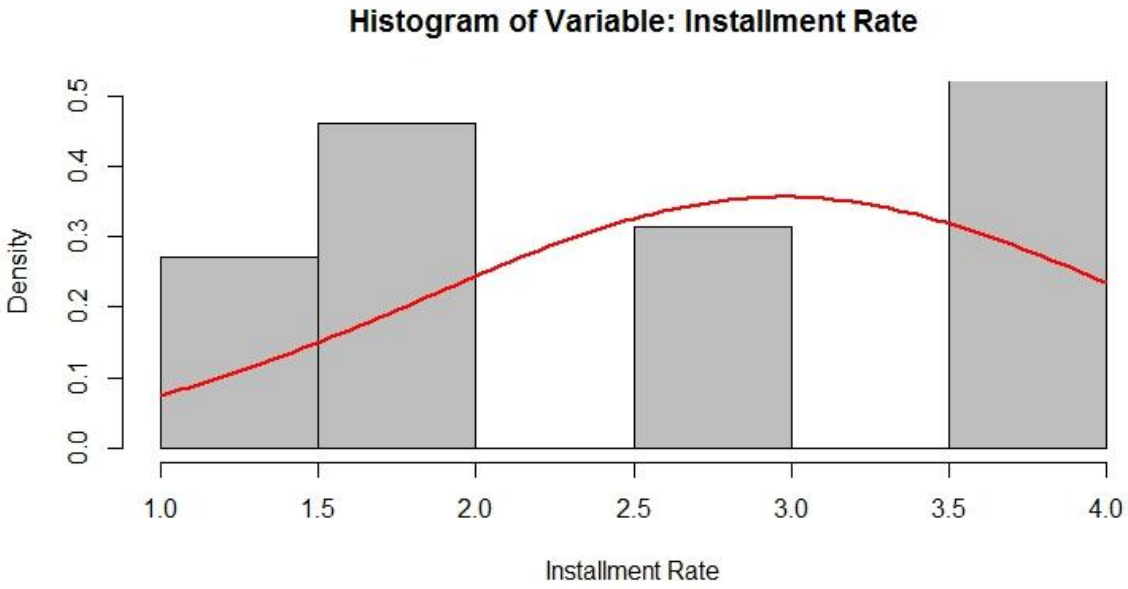
Histograms

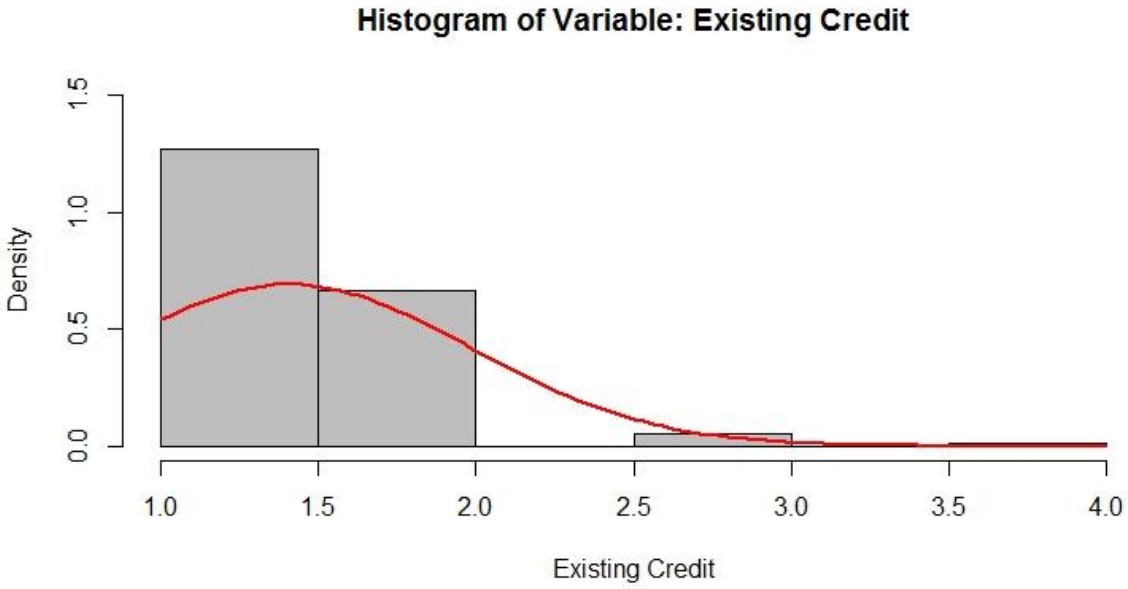

Histogram of Variable: Age
Density
0.00
0.01
0.02
0.03
0.04
0.05
20
30
40
50
60
70
Age in Years

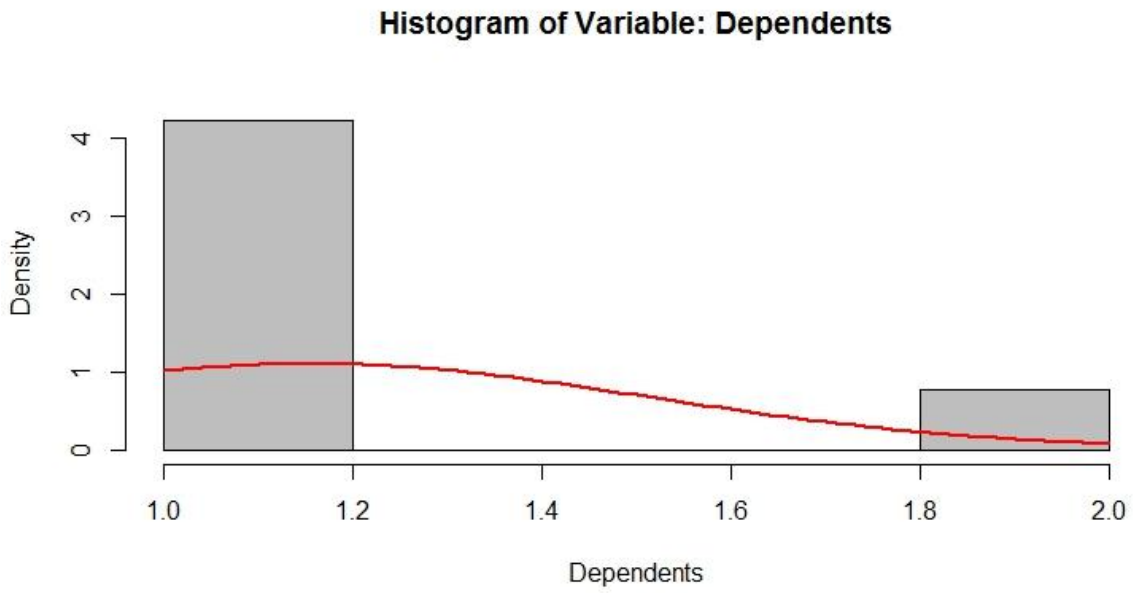

Histogram of Variable: Age
Density
0.00
0.01
0.02
0.03
0.04
0.05
20
30
40
50
60
70
Age in Years


Histogram of Variable: Credit Amount
Density
0.00000
0.00015
0.00030
0
5000
10000
15000
Credit Amount

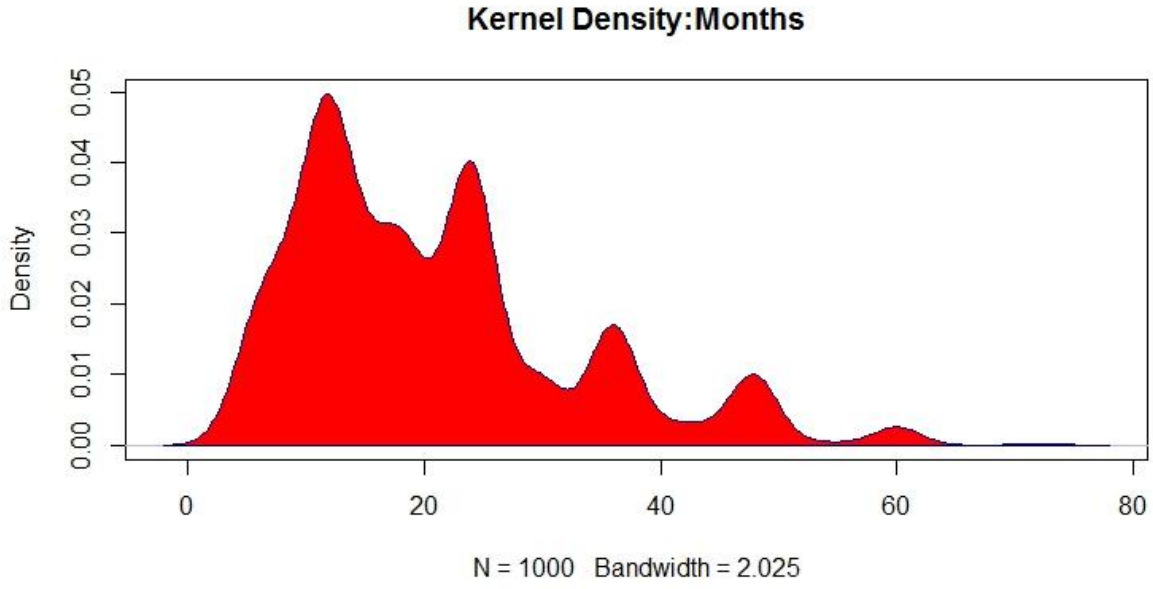

Histogram of Variable: Residence Since
Density
0.0
0.2
0.4
0.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Residence_Since

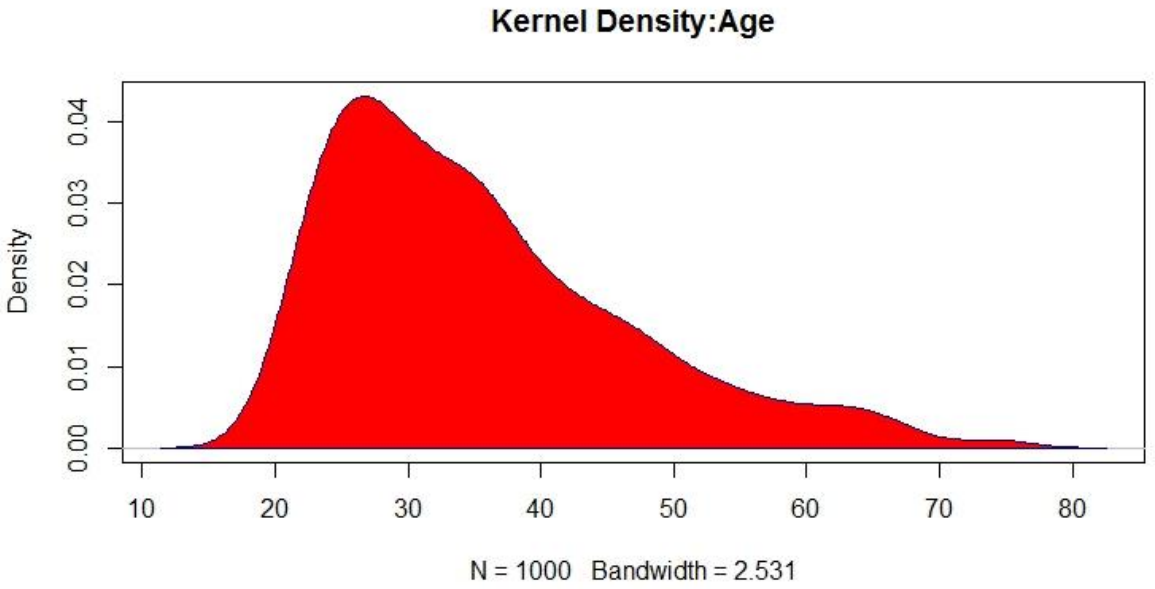

Histogram of Variable: Installment Rate
Density
0.0
0.1
0.2
0.3
0.4
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Installment Rate

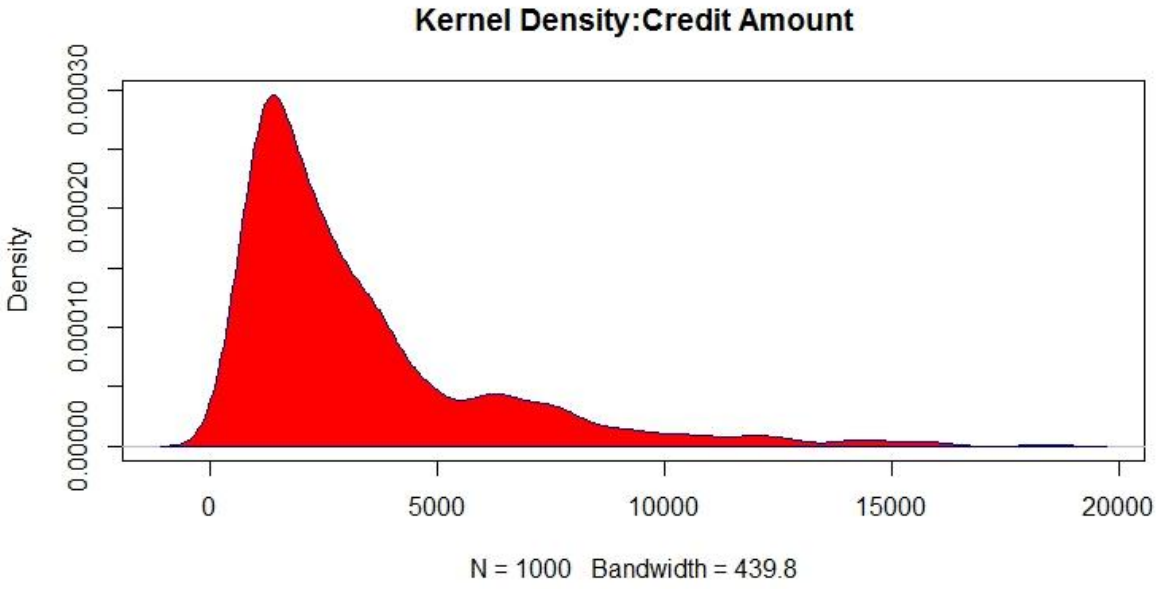

Histogram of Variable: Existing Credit
Density
0.0
0.5
1.0
1.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Existing Credit

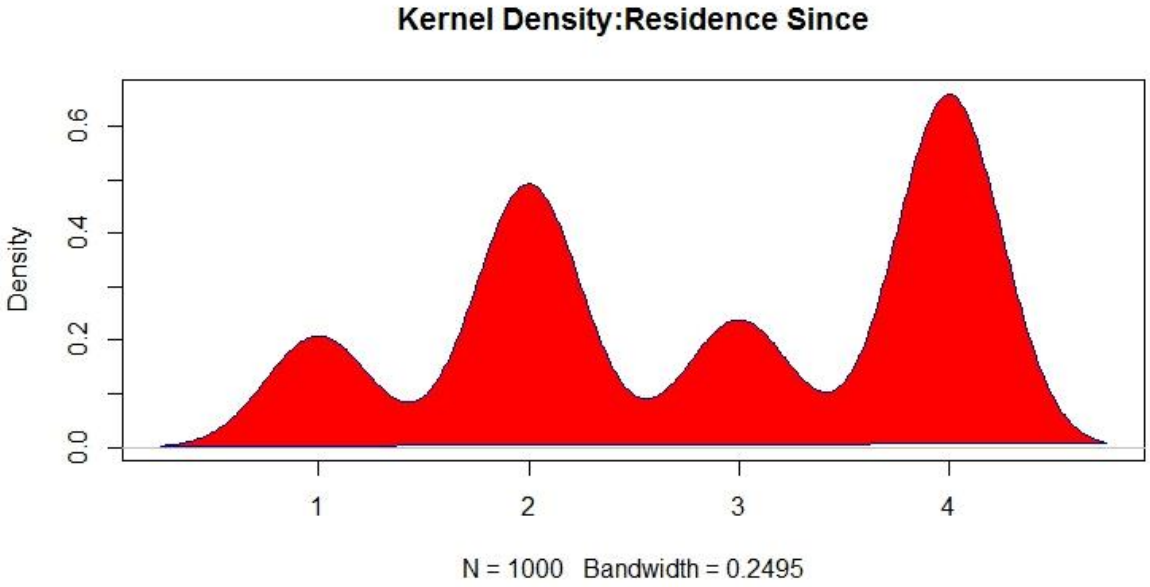


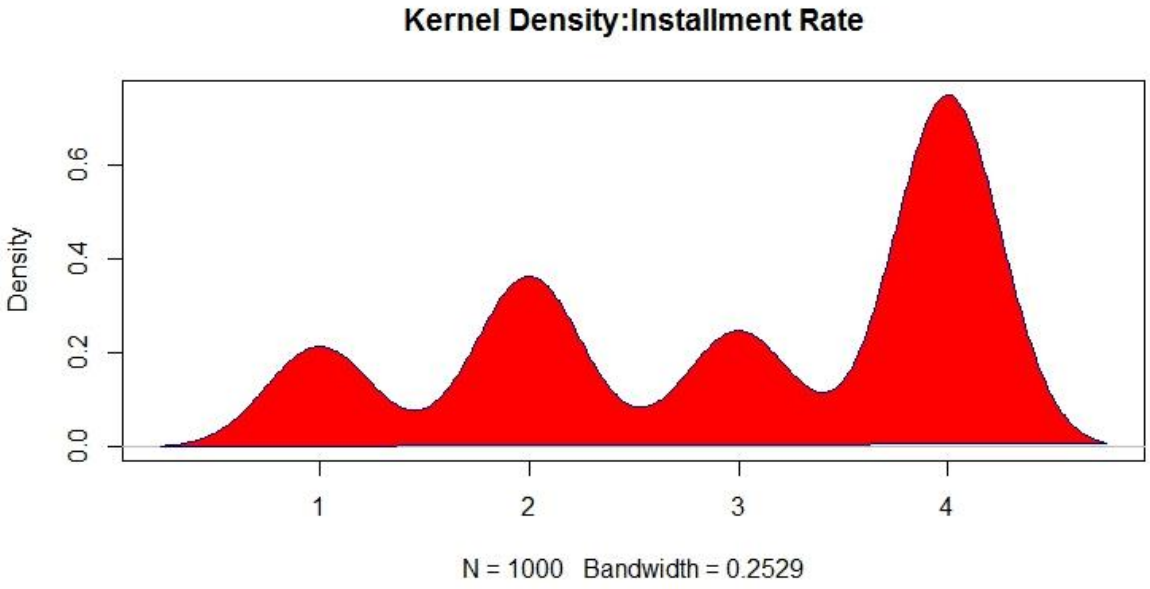
Kernel Density Plots

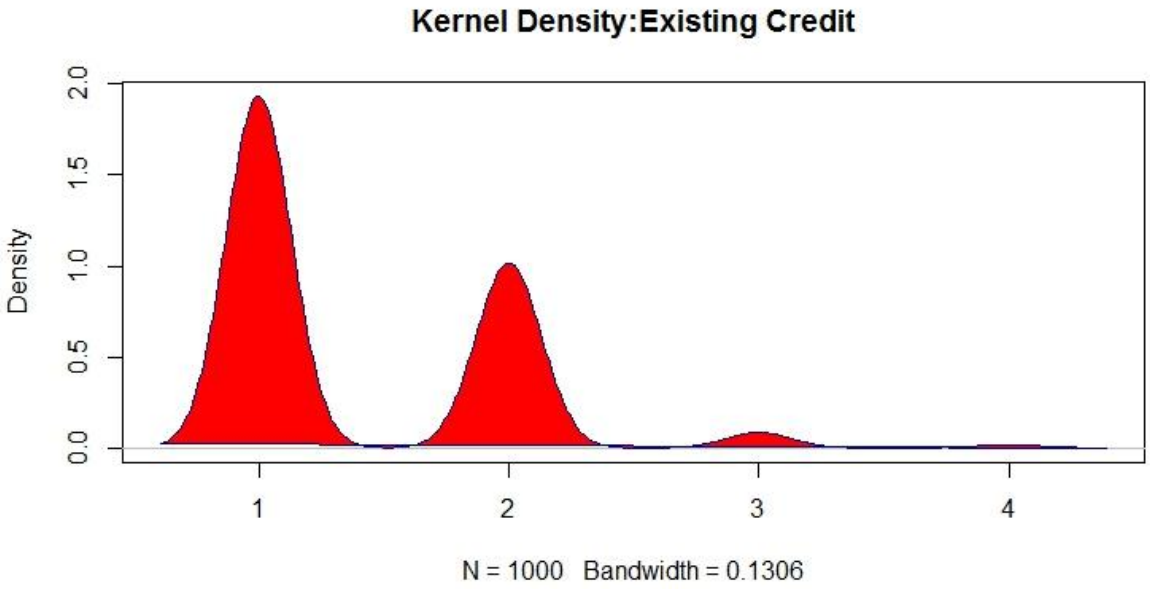


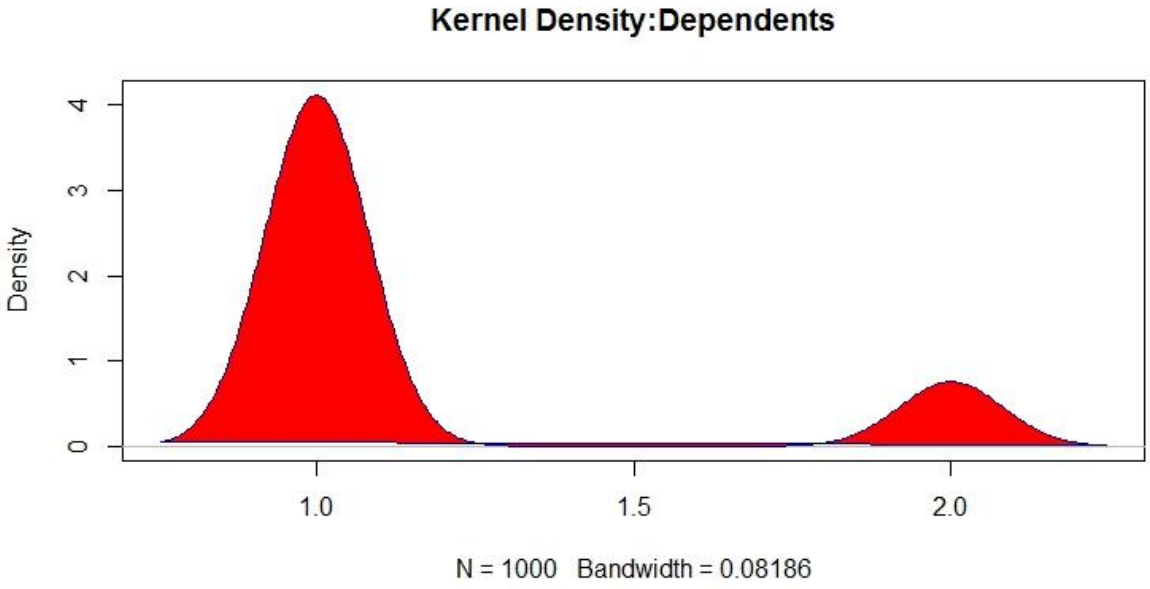

Kernel Density:Age
Density
0.00
0.01
0.02
0.03
0.04
10
20
30
40
50
60
70
80
N = 1000 Bandwidth = 2.531


Kernel Density:Credit Amount
Density
0.00000
0.00010
0.00020
0.00030
0
5000
10000
15000
20000
N = 1000 Bandwidth = 439.8

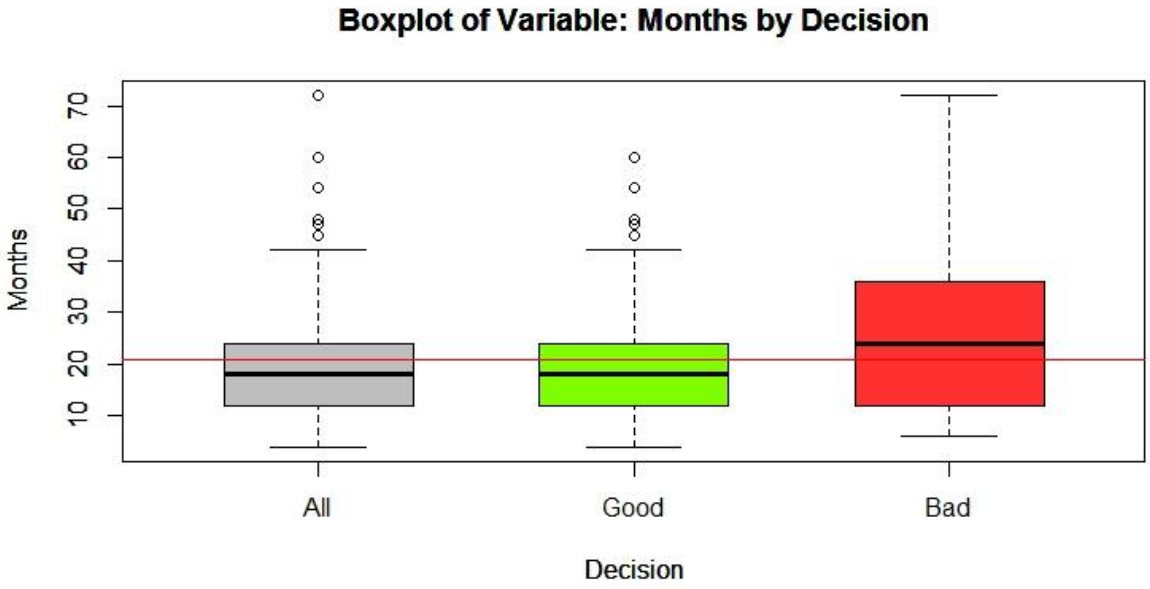

Kernel Density:Residence Since
Density
0.0
0.2
0.4
0.6
1
2
3
4
N = 1000 Bandwidth = 0.2495

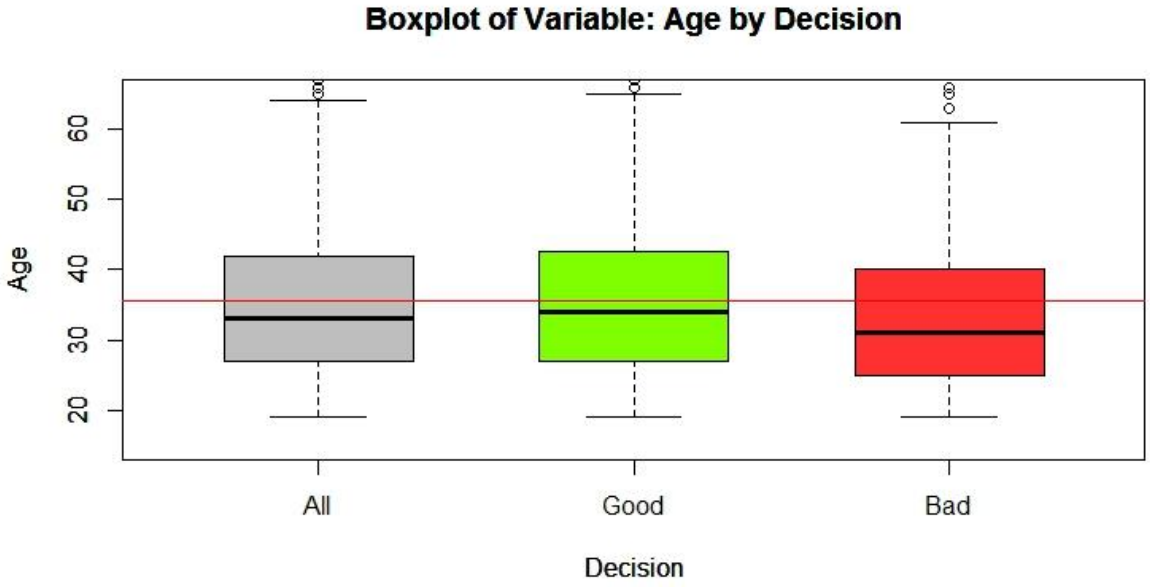

Kernel Density:Installment Rate
Density
0.0
0.2
0.4
0.6
1
2
3
4
N = 1000 Bandwidth = 0.2529

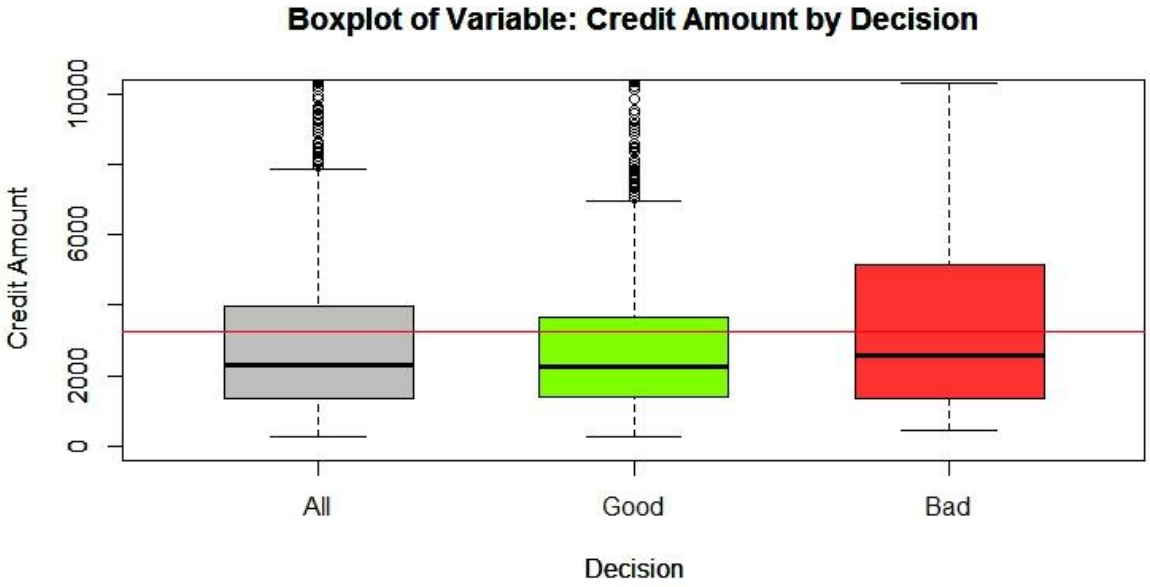

Kernel Density:Existing Credit
Density
0.0
0.5
1.0
1.5
2.0
1
2
3
4
N = 1000 Bandwidth = 0.1306

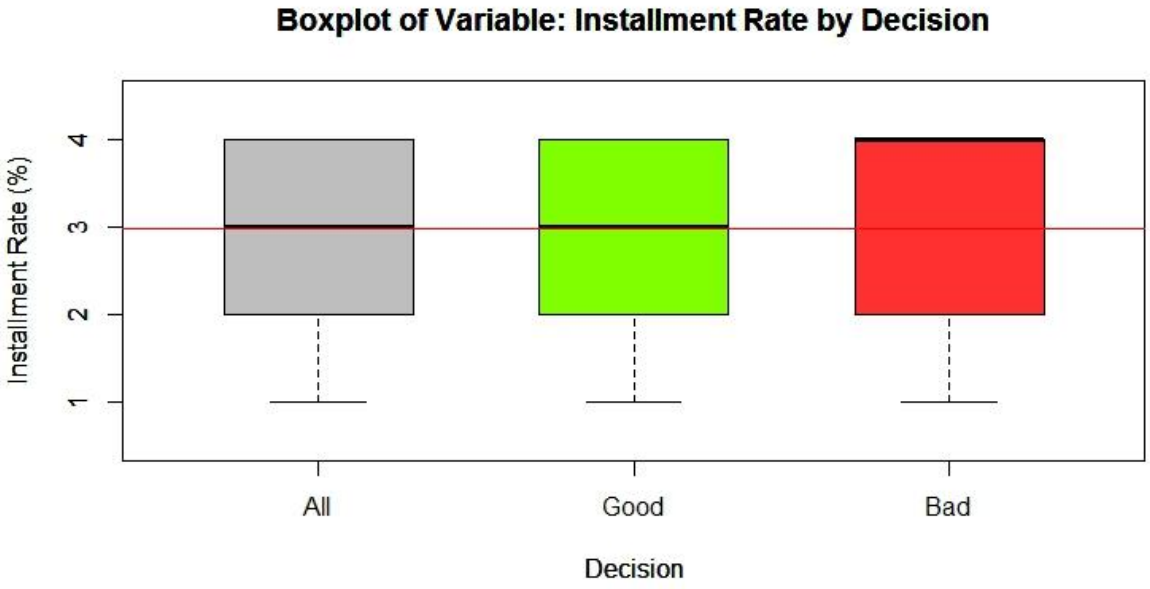

Kernel Density:Dependents
Density
0
1
2
3
4
1.0
1.5
2.0
N = 1000 Bandwidth = 0.08186

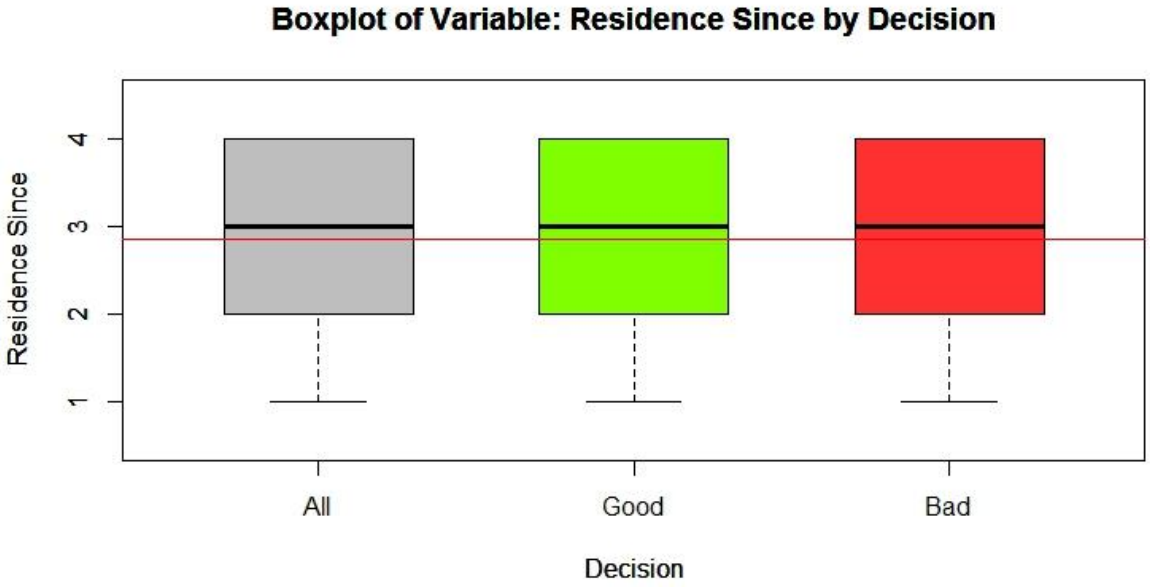
Box Plots

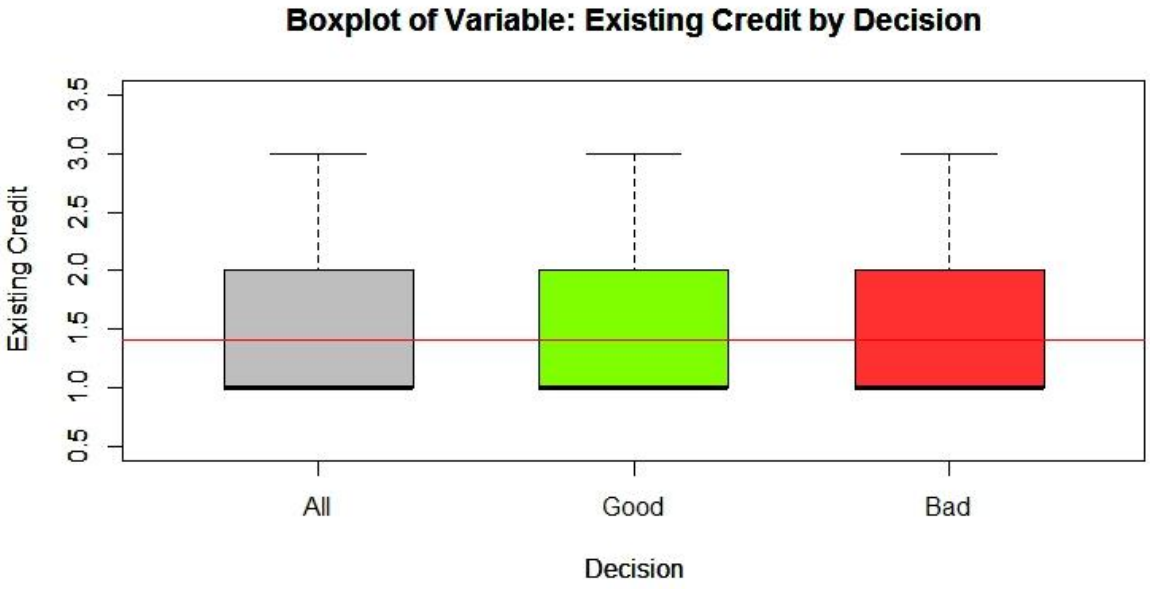

Boxplot of Variable: Months by Decision
Months
10
20
30
40
50
60
70
All
Good
Bad
Decision


Boxplot of Variable: Age by Decision
Age
20
30
40
50
60
All
Good
Bad
Decision


Boxplot of Variable: Credit Amount by Decision
Credit Amount
0
2000
6000
10000
All
Good
Bad
Decision


Boxplot of Variable: Installment Rate by Decision
Installment Rate (%)
1
2
3
4
All
Good
Bad
Decision


Boxplot of Variable: Residence Since by Decision
Residence Since
1
2
3
4
All
Good
Bad
Decision


Boxplot of Variable: Existing Credit by Decision
Existing Credit
0.5
1.0
1.5
2.0
2.5
3.0
3.5
All
Good
Bad
Decision

A.3 Data Dictionary
(Source: https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/)

Description of the German credit dataset.

1. Title: German Credit data

2. Source Information

Professor Dr. Hans Hofmann
Institut f"ur Statistik und "Okonometrie
Universit"at Hamburg
FB Wirtschaftswissenschaften
Von-Melle-Park 5
2000 Hamburg 13

3. Number of Instances: 1000

Two datasets are provided. the original dataset, in the form provided
by Prof. Hofmann, contains categorical/symbolic attributes and
is in the file "german.data".

For algorithms that need numerical attributes, Strathclyde University
produced the file "german.data-numeric". This file has been edited
and several indicator variables added to make it suitable for
algorithms which cannot cope with categorical variables. Several
attributes that are ordered categorical (such as attribute 17) have
been coded as integer. This was the form used by StatLog.

6. Number of Attributes german: 20 (7 numerical, 13 categorical)
Number of Attributes german.numer: 24 (24 numerical)

7. Attribute description for german

Attribute 1: (qualitative)
Status of existing checking account
A11 : ... < 0 DM
A12 : 0 <= ... < 200 DM
A13 : ... >= 200 DM /salary assignments for at least 1 year
A14 : no checking account

Attribute 2: (numerical)
Duration in month

Attribute 3: (qualitative)
Credit history
A30 : no credits taken/all credits paid back duly
A31 : all credits at this bank paid back duly
A32 : existing credits paid back duly till now
A33 : delay in paying off in the past
A34 : critical account/other credits existing (not at this bank)

Attribute 4: (qualitative)
Purpose
A40 : car (new)
A41 : car (used)
A42 : furniture/equipment
A43 : radio/television
A44 : domestic appliances
A45 : repairs
A46 : education
A47 : (vacation - does not exist?)
A48 : retraining
A49 : business
A410 : others

Attribute 5: (numerical)
Credit amount

Attribute 6: (qualitative)
Savings account/bonds
A61 : ... < 100 DM
A62 : 100 <= ... < 500 DM
A63 : 500 <= ... < 1000 DM
A64 : . >= 1000 DM
A65 : unknown/ no savings account

Attribute 7: (qualitative)
Present employment since
A71 : unemployed
A72 : ... < 1 year
A73 : 1 <= ... < 4 years
A74 : 4 <= ... < 7 years
A75 : .. >= 7 years

Attribute 8: (numerical)
Installment rate in percentage of disposable income

Attribute 9: (qualitative)
Personal status and sex
A91 : male : divorced/separated
A92 : female : divorced/separated/married
A93 : male : single
A94 : male : married/widowed
A95 : female : single

Attribute 10: (qualitative)
Other debtors / guarantors
A101 : none
A102 : co-applicant
A103 : guarantor

Attribute 11: (numerical)
Present residence since

Attribute 12: (qualitative)
Property
A121 : real estate
A122 : if not A121 : building society savings agreement/
life insurance
A123 : if not A121/A122 : car or other, not in attribute 6
A124 : unknown / no property

Attribute 13: (numerical)
Age in years

Attribute 14: (qualitative)
Other installment plans
A141 : bank
A142 : stores
A143 : none

Attribute 15: (qualitative)
Housing
A151 : rent
A152 : own
A153 : for free

Attribute 16: (numerical)
Number of existing credits at this bank

Attribute 17: (qualitative)
Job
A171 : unemployed/ unskilled - non-resident
A172 : unskilled - resident
A173 : skilled employee / official
A174 : management/ self-employed/
highly qualified employee/ officer

Attribute 18: (numerical)
Number of people being liable to provide maintenance for

Attribute 19: (qualitative)
Telephone
A191 : none
A192 : yes, registered under the customers name

Attribute 20: (qualitative)
foreign worker
A201 : yes
A202 : no

8. Cost Matrix

This dataset requires use of a cost matrix (see below)

	1	2
1	0	1
2	5	0

(1 = Good, 2 = Bad)

the rows represent the actual classification and the columns
the predicted classification.

It is worse to class a customer as good when they are bad (5),
than it is to class a customer as bad when they are good (1).