Predicting Default Status of Bank Loan

Harshita Prasad

29 February 2020

Table of Contents

1. Introduction

1.1. Problem Statement

The loan default dataset has 8 variables and 850 records, each record being loan default status for each customer. Each Applicant was rated as “Defaulted” or “Not-Defaulted”. New applicants for loan application can also be evaluated on these 8 predictor variables and classified as a default or non-default based on predictor variables.

1.2. Data

We need to build a statistical model that will predict the default status of bank loan cases based on various factors. Given below is the details of data attributes in the dataset. Table 1.1 contains a sample of the dataset and table 1.2 contains all the independent variables whereas table 1.3 contains the dependent variable -

- Age: age of each customer
- Education: education categories
- Employment: employment status - corresponds to job status and being converted to numeric format
- Address: geographic area - converted to numeric values
- Income: gross income of each customer
- debtinc: individual’s debt payment to his or her gross income
- creddebt: debt-to-credit ratio is a measurement of how much you owe your creditors as a percentage of your available credit (credit limits)
- othdebt: any other debts
- default: default status of customers as defaulted or not defaulted

Table 1.1: default loan case Sample Data (Columns: 1-9)

Age	Education	Employment	Address	Income	debtinc	creddebt	othedebt	default
41	3	17	12	176	9.3	11.35939	5.008608	1
27	1	10	6	31	17.3	1.362202	4.000798	0
40	1	15	14	55	5.5	0.856075	2.168925	0
41	1	15	14	120	2.9	2.65872	0.82128	0
24	2	2	0	28	17.3	1.787436	3.056564	1
41	2	5	5	25	10.2	0.3927	2.1573	0

Table 1.2: Independent Variables

S.No.	Predictor
1	Age
2	Education
3	Employment
4	Address
5	Income
6	debtinc
7	creddebt
8	othdebt

Table 1.3: Dependent Variables

S.No.	Response
1	Default

2. Methodology

2.1. Pre-processing

The first step in building a statistical model starts with exploratory data analysis. It includes data mining which transforms the raw messy data into clean understandable data. Pre-processing of data mills the data through a series of steps like data cleaning, data transformations and data reduction and includes data visualization thorough graphs and plots. Going further, we will discuss every step in detail and build up the methodology.

2.1.1. Data Exploration

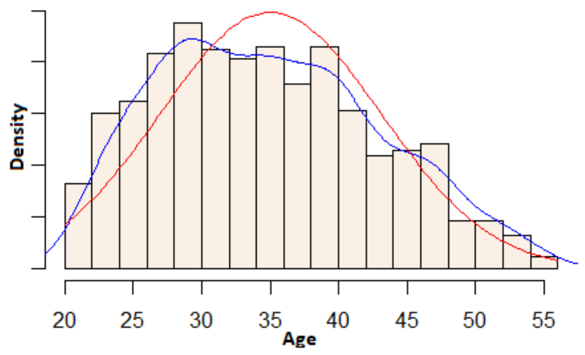
The first step in pre-processing the dataset is to explore it in detail. In doing so, we get a good understanding of the dataset and the observations gathered from this exploration helps us further to analyse it. In order to do that, we need to look at various dimensions of the data. This starts with identifying the distribution of the variables.

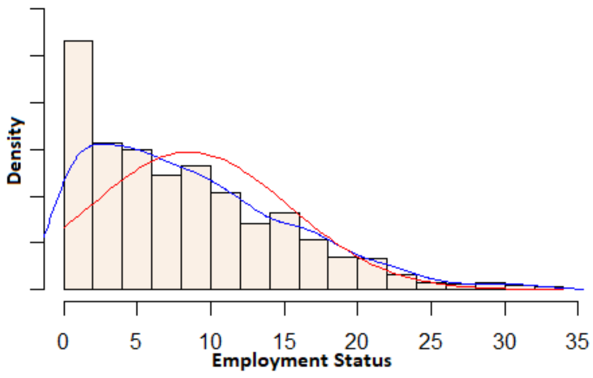
But before we do that, we need to make some changes in the dataset in order to understand it better:

- Changing variable names to make it easier to understand. ed to education, employ to employstatus and othdebt to otherdebt
- Converting education and default variables into categorical variables

Let's now talk about the distribution of the continuous variables. In order to predict a variable accurately, we need to know their behaviour. Once we determine the outcomes of the variables and assign a probability to them, we can then plot them and get a curve which is known as probability distribution curve. And, the likelihood of the target variable getting a value is the probability distribution or probability density function of the variable.

Normal Distribution is the most widely used probability distribution which comprises of the bell-shaped curve and the equality of mean, median and mode. The best way to showcase this is in the form of histograms combined with normal curve and Kernel Density Estimations. The variables that exhibit normal distribution is feasible to be forecasted with higher accuracy.





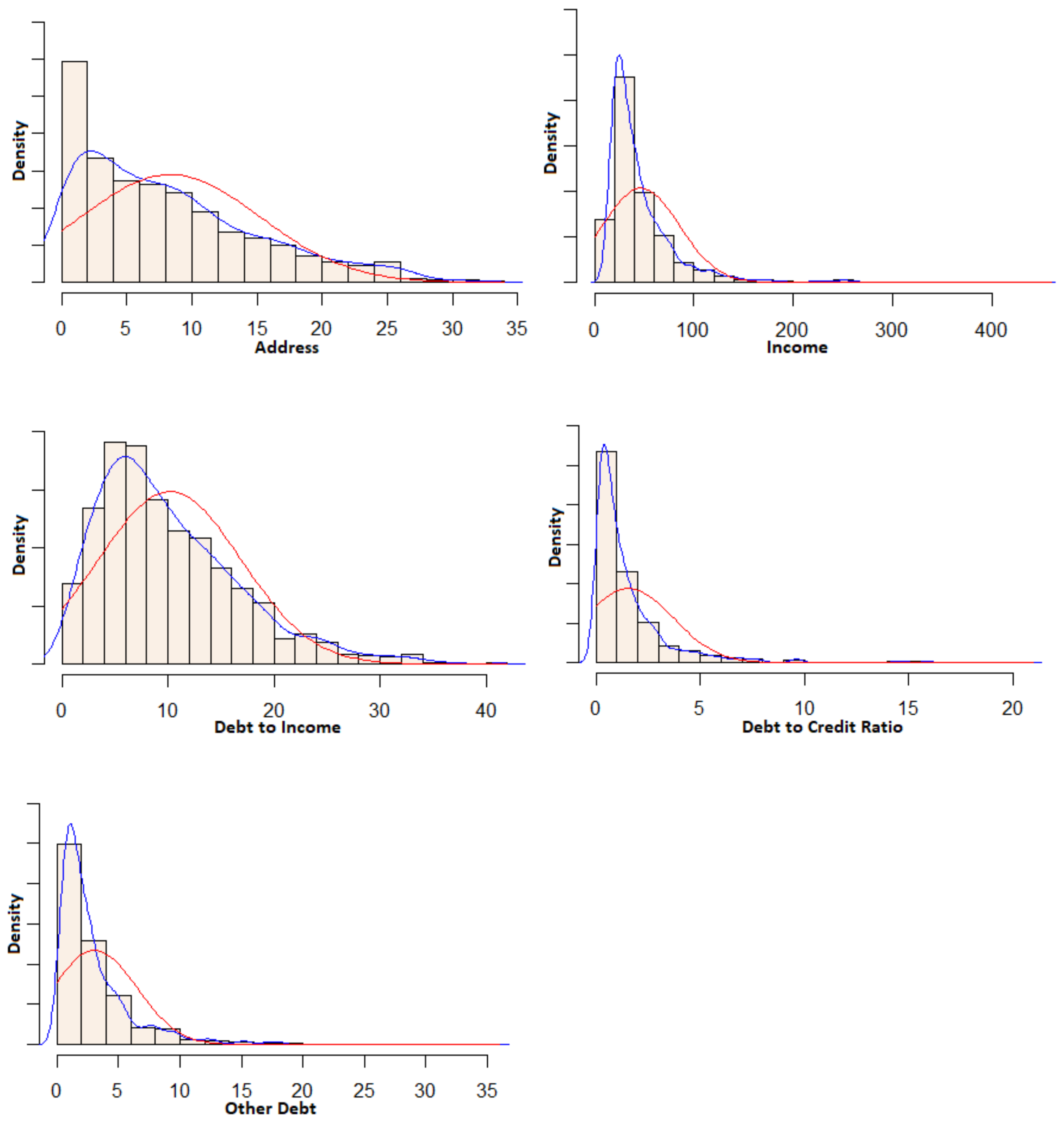
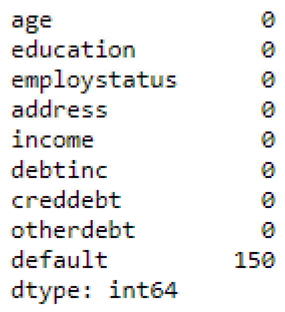


Figure 2.1: Probability Density Functions of continuous variables of default loan case data

2.1.2. Missing Values

Missing value analysis is the second step to data pre-processing. The values that are missing from the dataset needs to be taken care of. It may happen due to human error, optional questions and refusal to answer survey questions. In this case, if the missing data is above 30%, it either needs to be removed or calculated using one of the methods like central statistics method, distance-based methods using KNN, Euclidean distance, etc or prediction method.

We can see from fig 2.2 that there are 150 missing values in the default variable which constitutes to about 17% of the dataset. We can drop them altogether but since the number of values is not a lot, we can calculate it using one of the mentioned methods. here, KNN imputation method seems like an appropriate method to impute the missing values in the dataset since default is a categorical variable.



```
age               0
education         0
employstatus      0
address           0
income            0
debtinc           0
creddebt          0
otherdebt         0
default         150
dtype: int64
```

Figure 2.2: Missing Value Analysis

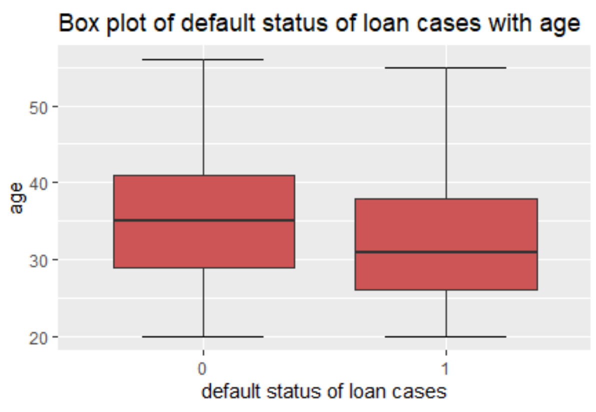
2.1.3. Outlier Analysis

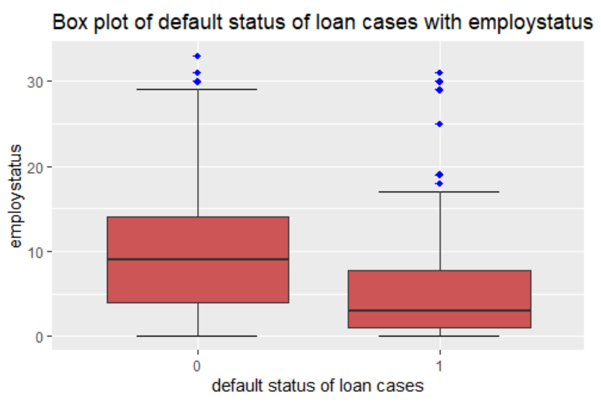
After dealing with the missing values, we proceed with the outlier analysis. This analysis is done to handle the observations which are inconsistent with the rest of the dataset. And, it can only be done on continuous variables.

Outliers can happen due to poor data quality, malfunctioning, manual error, exceptional data, etc. In order to detect the outliers and deal with them, we can utilise various methods. We can follow these steps:

- Detect the outliers using the Box Plot graphs.
- Take any one route:
  - Remove the outliers using box plot method
  - Impute it using central statistics like mean, median or KNN method

We can see from the box plots (fig 2.3) that all variables except age has outliers. In order to deal with it, we have used KNN imputation method. The method includes substituting the outliers first with NA and then treating them as missing values. It creates a predictive model to estimate values that will substitute the missing data. Since attributes have correlation among them, they can be used to create a predictive model where the variable with missing data is used as class-attribute and the remaining variables are used as input for the predictive model.





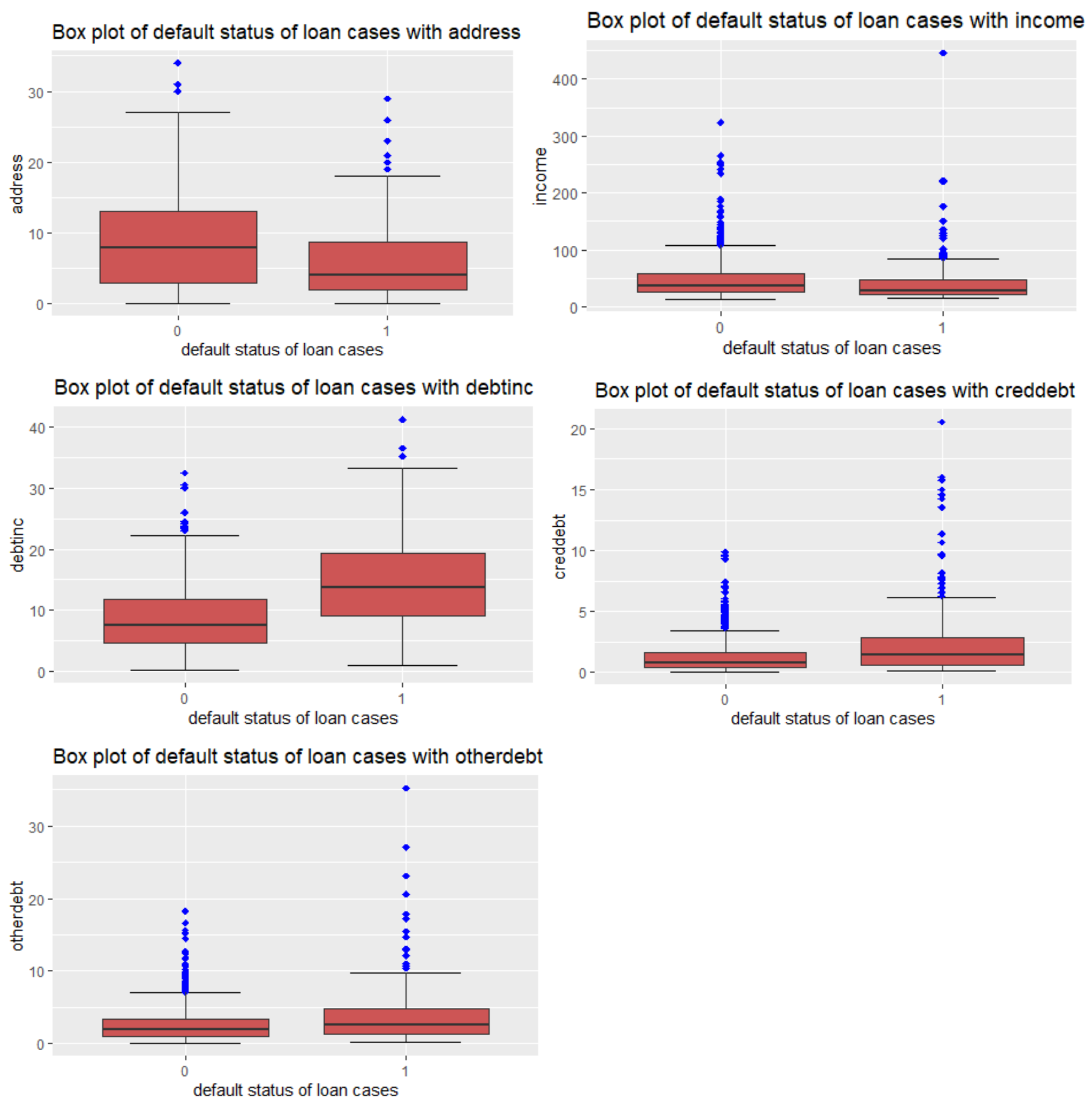


Figure 2.3: Box Plots of continuous variables

2.1.4. Feature Selection

The next step to pre-processing is feature selection analysis. It is done to extract relevant or meaningful features by selecting subsets of relevant features (variables and predictors) out of the data to be considered for model construction. It's also called dimensionality reduction. We look for redundant variables which will not be helpful in predicting the target variable and remove them from our dataset.

There are many methods that helps in feature selection analysis. Let's see some of them in detail:

1. **Correlation Analysis** – It tells you the association between two continuous variables. For two independent variables to be considered for the model, there should be no correlation between them. And for one independent and one dependent variable to be considered, they

should be highly correlated. In order to do this analysis, we filter out the numerical variables and do a correlation analysis on them. We do it by plotting a correlation plot and studying it. Dark blue colour indicates a high correlation and light pink colour shows less or no correlation.

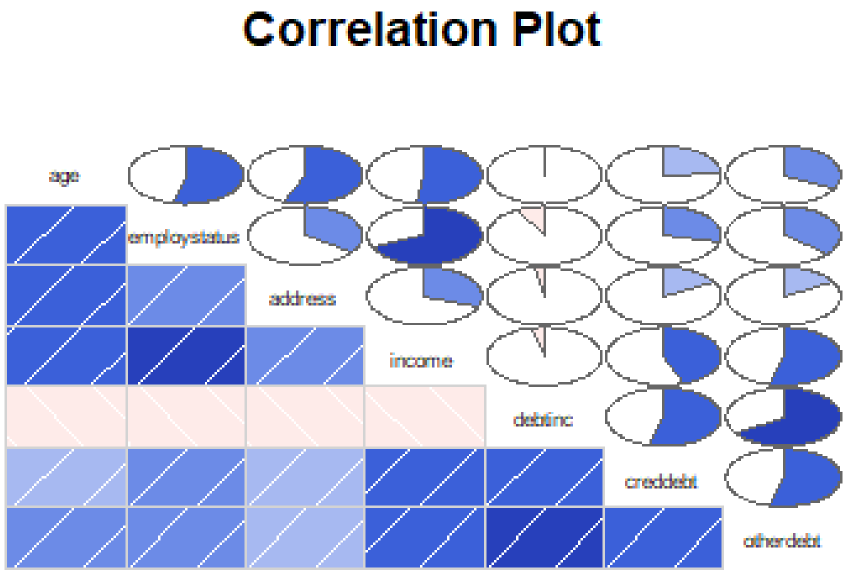


Figure 2.4: Correlation Plot

We can see that no two variables have high correlation. Since all the variables seem important for the analysis, we can choose to skip this part.

2. **Chi-Square Test of Independence** – It compares two categorical variables in a contingency table to check their association. For two independent variables to be considered, there should be no dependency. And, for one independent and dependent variable to be considered, they should have high dependency. In order to do a chi squared test we take the following steps:
   a. Establish Hypothesis testing –
      Null hypothesis (H_0) – two variables are independent
      Alternate hypothesis (H_1) - two variables are not independent
   b. To understand it -
      i. Chi squared statistics > critical value, reject Null hypothesis (H_0)
         p-value < 0.05, reject H_0 and remove the redundant variable
      ii. Chi squared statistics < critical value, accept Null hypothesis (H_0)
         p-value > 0.05, accept H_0 and keep the variables

After doing the analysis, we can choose to skip this part as no variable's p-value is less than 0.05.

3. **ANOVA (Analysis of Variance) Test** – It is used test association between one categorical variable and one continuous variable. It compares the means of two or more categories in a variable and tries to find a significance.
   a. Hypothesis testing -
      i. Null hypothesis (H_0) – means of all categories in a variable are same

ii. Alternate hypothesis (H_1) - mean of at least one category in a variable is different

b. To understand it -

i. p-value < 0.05, reject H_0

ii. p-value > 0.05, accept H_0

4. **Multicollinearity –** It is the occurrence of several independent variables in a regression model which are closely related. It can be dealt with in couple of ways. Either we choose to ignore it, which happens when prediction of dependent variable is the objective. Or, get rid of the redundant variable by using variable selection technique. And for this we need to check the VIF (variance inflation factor):
   a. If VIF = 1, not correlated to any of the variables
   b. If VIF is between 1 to 5, moderately correlated
   c. If VIF > 5, highly correlated

   In the case of high correlation in multiple variables, remove the one with highest VIF.

2.1.5. Feature Scaling

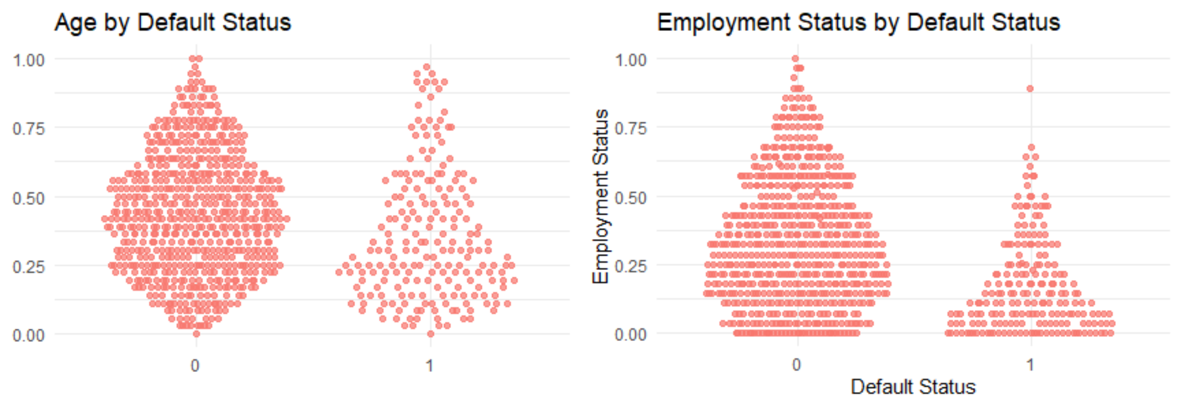
This next step to pre-processing is used to limit the range of variables so that they can be compared on common grounds. It is performed only on continuous variables.

- ❖ Normalization – It is the process of reducing unwanted variation either within or between variables to bring all the variables into proportion with one another. It gives a common scale to variables by converting the values in the range of 0 to 1.
- ❖ Standardization – this method is used when the data is normally/uniformly distributed. To convert the values, the formula subtracts mean from individual points and divides it by standard deviation.

Since most of our data is not normally distributed, we will choose normalization. We have also checked for variance before applying normalization.

2.2. Visualizations

This next step in building a statistical model includes visualizing the variables in different forms to understand it better. For the numerical variables we have chosen to go with a beeswarm plot in fig 2.5 to compare each of the independent variables with the default status variable. It clearly shows the variable values at default or not default statuses. For the categorical variable, we have chosen a bar plot in fig 2.6 to compare the variable with the target variable.



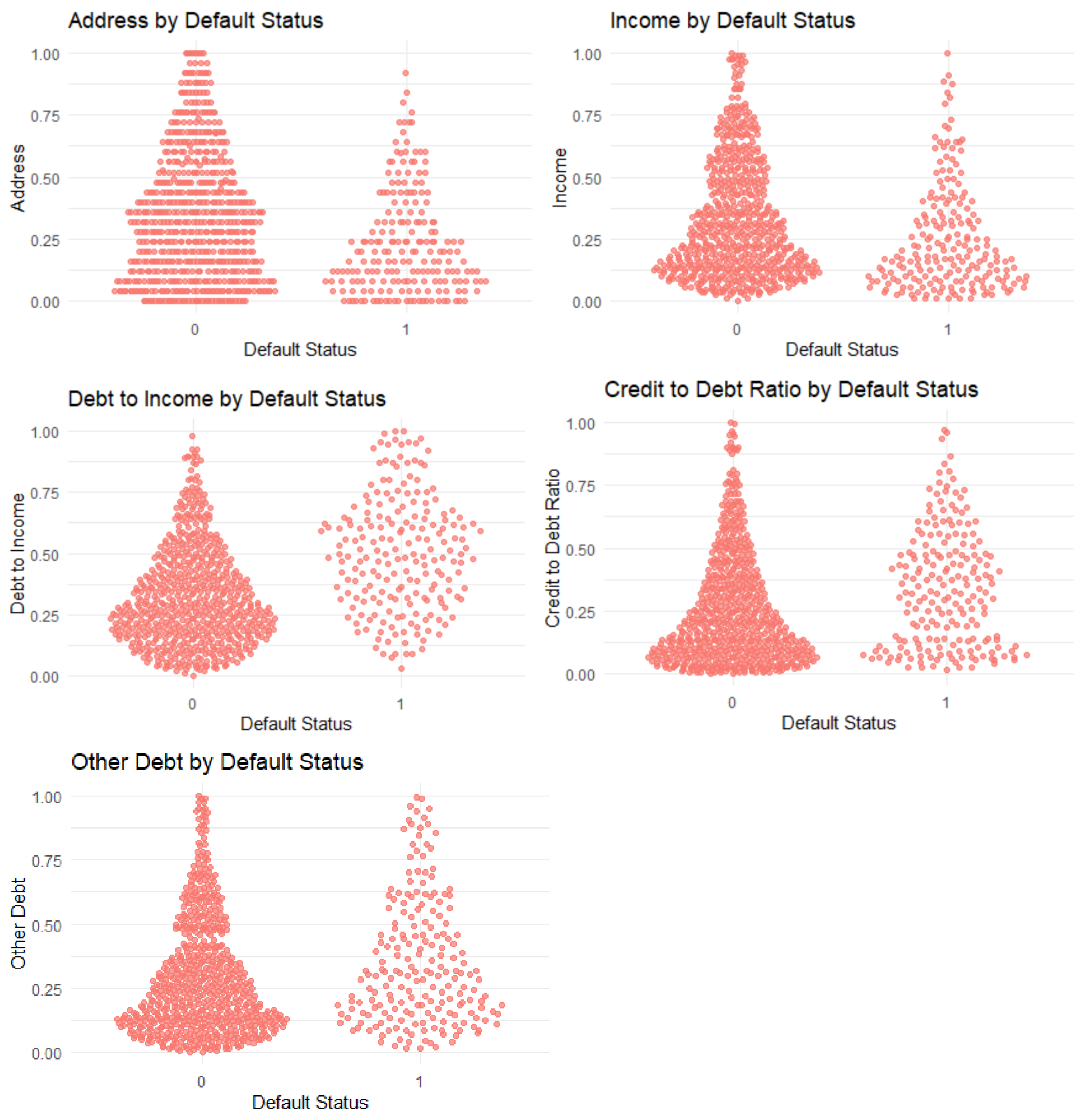


Figure 2.5: Beeswarm Plot

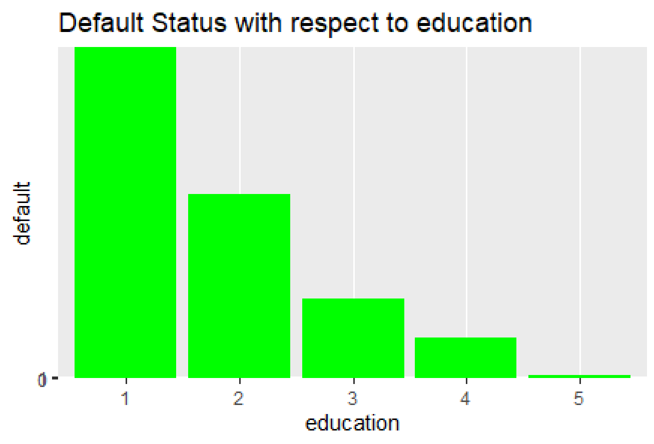


Figure 2.6: Bar Graph

3. Modelling

3.1. Model Selection

The next step in analysing the data is selecting the appropriate model and developing it for further process. For this, we need to split the data into train and test data. Then build a model using the train data to predict the output using the test data. Model having less false negative rate and more accuracy will be chosen as our final model.

For our data, we have divided the data into 80% as train data and 20% as test data. Since our data has a categorical target variable, we will use appropriate models. Following are the few popular models used for this purpose:

3.1.1. Logistic Regression

It is the simplest and most common model for predictive analysis when the dependent variable is categorical. It is used to describe data and to explain the relationship between one dependent categorical variable and one or more independent variable. The idea is to predict the probability of possible outcomes using logistic function to estimate the prediction. Also, which variables are significant predictors of the target variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the target variable. We can use binary logistic regression in case of two levels in the dependent variable or we can use multinomial logistic regression model for more than two levels in the dependent variable.

3.1.2. Decision Tree

It builds both regression and classification models based on a branching series of Boolean tests i.e. tree like graph. The motive is to create a training model used to predict the class/value of target variable by learning the decision roots inferred from historical data. Each node/ branch represents an attribute and each leaf represents a class label. It can handle both categorical and numerical data. There are different types of algorithms for decision trees like C5.0 in tandem with Entropy, C4, CART in tandem with GINI index, etc.

3.1.3. Random Forest

It is an ensemble that consists of 'n' number of decision trees which combines Breiman "bragging" idea and the random selection of features. The idea is to combine multiple decision trees in determining the final output rather than using individual decision tree which will increase accuracy and decrease misclassification error rate. It can handle large number of independent variables without deletion and gives an estimate about importance of variables. It can build both regression and classification models. For our model, we have considered n to be first 100 and then 500 to get more varied results.

3.1.4. KNN Implementation

KNN stands for K-nearest neighbour and is called a lazy learning method. It is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. It is a supervised learning method used for both classification and regression models. The steps include picking 'k' number of neighbours to be used for classification or regression. Then choosing a method to measure distance like Euclidean, Manhattan, weighted distance, etc. Also, keeping a data set with records. For every new point, identifying the chosen number of nearest neighbours using the chosen method. Then let it be a vote for classification or mean/median for regression.

3.1.5. Naïve Bayes

Naïve Bayes is one of the most practical learning methods used for probabilistic classification method. It works on Bayes theorem of probability to predict the class of unknown dataset. Bayes' theorem with strong (naïve) independence assumptions between the features. In this method the main assumption is that the attribute values are conditionally independent given the target value.

4. Conclusion

4.1. Model Evaluation

In order to conclude our analysis, we need to evaluate the models and choose one. For this, we can look at the confusion matrix, calculate various rates in order to choose the most appropriate algorithm for our data.

4.1.1. Confusion matrix

Confusion matrix, also known as Error Matrix, describes the performance of classification model. The row of the matrix represents actual values/class whereas the column represents predicted class/values. It can be used for both binary and multinomial classifier problem. Since our data deals with a binary target variable, we will look at it in detail.

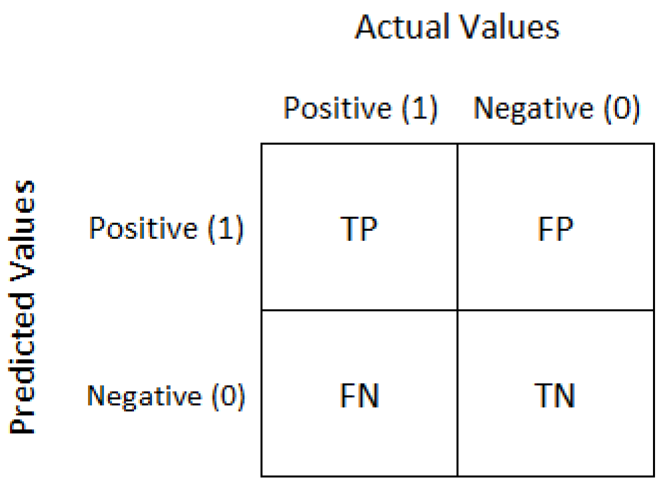


Figure 4.1: Confusion Matrix

This tables showcases what confusion matrix is all about. For every algorithm there are 4 possibilities. Let's look at them in detail with respect to our dataset:

1. TP (True Positive) – these are cases in which we predicted yes, the loan was default, and they are default.
2. TN (True Negative) – these are the cases in which we predicted not default, the loan was not default.
3. FP (False Positive) – these are the cases in which we predicted yes but the cases were not default. Also known as Type I error.
4. FN (False Negative) – these are the cases in which we predicted no but the cases were default. Also known as Type II error.

For every algorithm we have come up with the confusion matrices showcasing all the TP, TN, FP and FN.

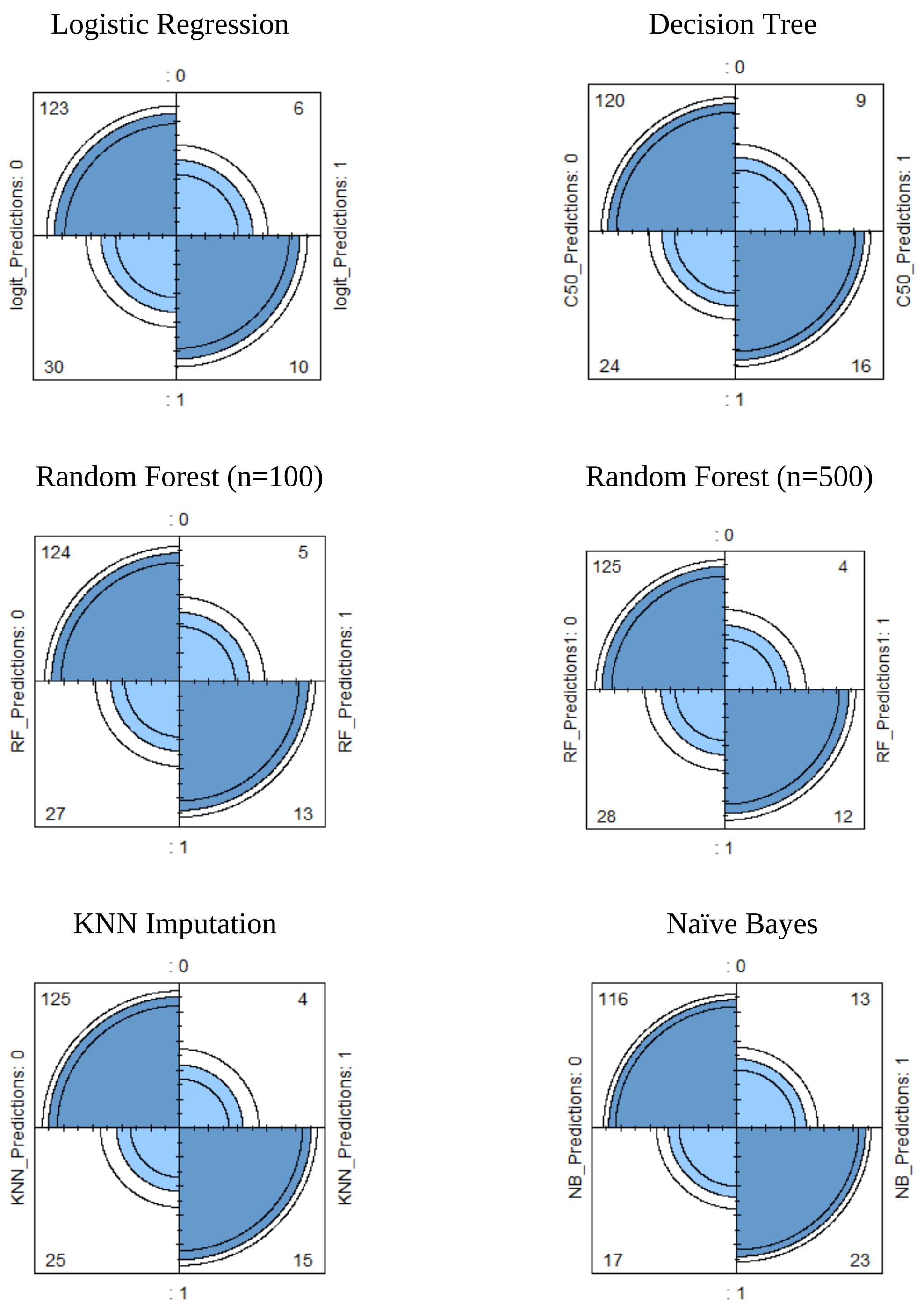


Figure 4.2: Confusion Matrices for all algorithms

Next, we look at list of rates that can be calculated from confusion matrix to make actual comparisons:

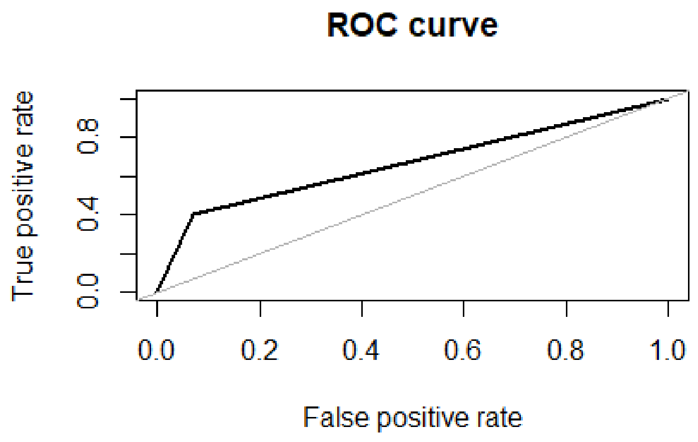
1. Accuracy – how often is the classifier correct
   – (TP + TN)/Total
2. Misclassification/Error Rate – how often is it wrong
   – (FP + FN)/Total OR (1 – Accuracy)

3. True Positive Rate/Sensitivity/Recall – when it's yes, how often does it predict yes – TP/ (FN + TP)
4. True Negative Rate/Specificity – when it's no, how often does it predict no – TN/ (TN + FP)
5. False Positive Rate/Type I Error – when it's no, how often does it predict yes – FP/ (TN + FP)
6. False Negative Rate/Type II Error – when it's yes, how often does it predict no – FN/ (FN + TP)

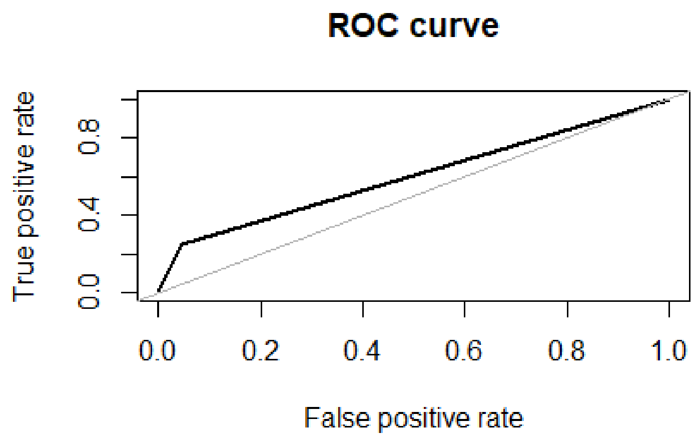
To make it easier for us, we are going to look at Accuracy and False Negative Rate to select our algorithm for this model.

4.1.2. ROC – AUC Curve

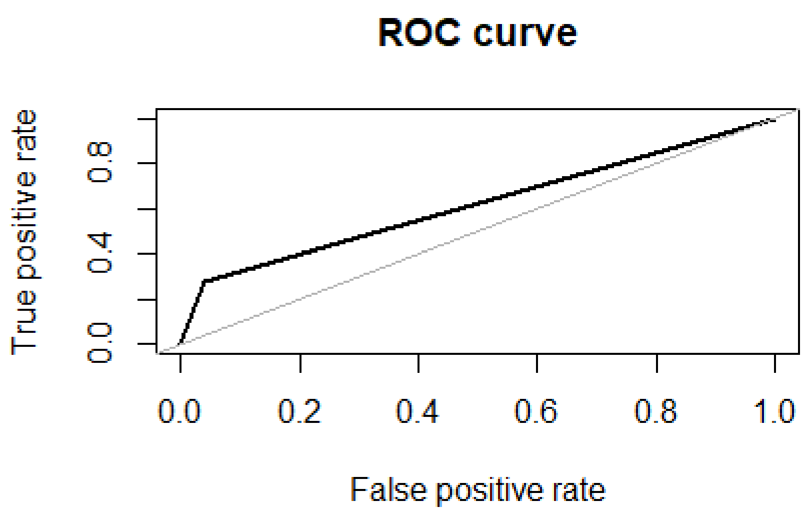
Based on these calculations above, we can also draw the ROC – AUC Curve. It is a performance measurement for classification problem at various settings. ROC is a probability curve and AUC show the degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between default loan cases and non-default loan cases.



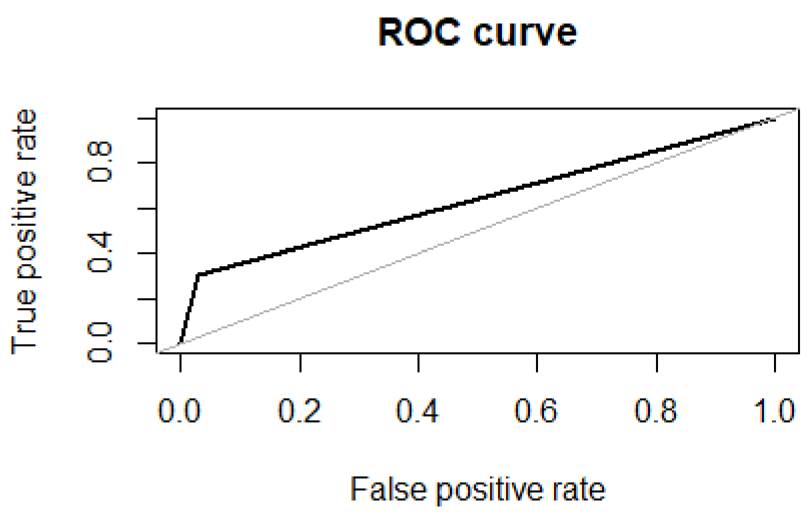
Logistic Regression



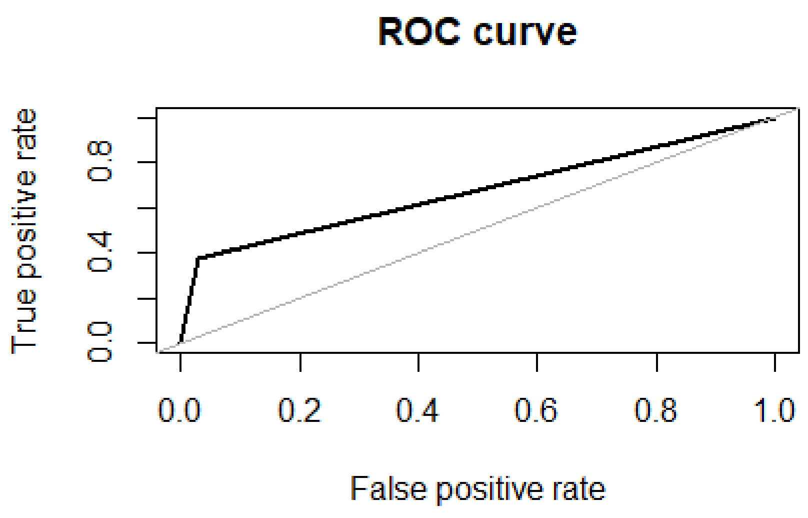
Decision Tree



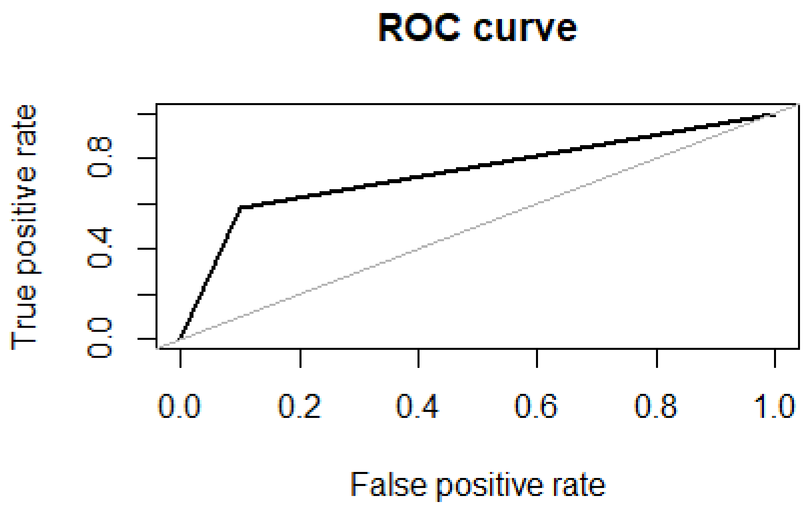
Random Forest (n = 100)



Random Forest (n = 500)



KNN Method



Naïve Bayes

Figure 4.3: ROC – AUC plots for all the algorithm

4.2. Model Selection

After calculation all the methods for each model, we can see from fig 4.4 that we can choose KNN Imputation model since it gives the most accuracy of 82.84% and least false negative rate of 21.05%.

(In Percent)

	Accuracy	False Negative Rate
Logistic Regression	78.70	75.00
Decision Tree	80.47	60.00
Random Forest (n = 100)	81.07	67.5
Random Forest (n = 500)	81.07	72.50
KNN Implementation	82.84	21.05
Naïve Bayes	82.25	42.50

Figure 4.4: Error Matrices

5. R Code

```
rm(list = ls())
setwd("C:/Users/Harshita Prasad/Desktop/loan default case")
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50",
"dummies", "e1071", "Information", "MASS", "rpart", "gbm", "ROSE", “sampling”,
“DataCombine”, “inTrees”, "psych", "usdm", "ggbeeswarm", "scales", "class", "yardstick")
lapply(x, require, character.only = TRUE)
rm(x)
original_data = read.csv('bank-loan.csv',header = T,na.strings = c(""," ","NA"))
df = original_data
str(df)
summary(df)
names(df)[names(df) == 'ed'] <- 'education'
names(df)[names(df) == 'employ'] <- 'employstatus'
names(df)[names(df) == 'othdebt'] <- 'otherdebt'
categorical_var = c('education', 'default')
numerical_var = c('age', 'employstatus', 'address', 'income','debtinc','creddebt','otherdebt')
typ_conv = function(df,var,type){
  df[var] = lapply(df[var], type)
  return(df)
}
df = typ_conv(df,categorical_var, factor)
str(df)
```

#Distribution Check

```
multi.hist(df$age, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab = "Age",ylab
= "Density"), dlty = c("solid", "solid"), bcol = "linen")
multi.hist(df$employstatus, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab =
"Employment Status",ylab = "Density"), dlty = c("solid", "solid"), bcol = "linen")
multi.hist(df$address, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab =
"Address",ylab = "Density"), dlty = c("solid", "solid"), bcol = "linen")
multi.hist(df$income, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab =
"Income",ylab = "Density"), dlty = c("solid", "solid"), bcol = "linen")
multi.hist(df$debtinc, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab = "Debt
Income Ratio",ylab = "Density"), dlty = c("solid", "solid"), bcol = "linen")
multi.hist(df$creddebt, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab = "Credit
Debit Ratio",ylab = "Density"), dlty = c("solid", "solid"), bcol = "linen")
multi.hist(df$otherdebt, main = NA, dcol = c("blue", "red"), title(main = NULL,xlab =
"Other Debt",ylab = "Density"), dlty = c("solid", "solid"), bcol = "linen")
```

# To check for number of missing data

```
apply(df, 2, function(x) {sum(is.na(x))})
missing_val = data.frame(apply(df,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
row.names(missing_val) = NULL
names(missing_val)[1] =  "missing_percentage"
missing_val$missing_percentage = (missing_val$missing_percentage/nrow(df)) * 100
missing_val = missing_val[,c(2,1)]
```

```
write.csv(missing_val, "Missing_values.csv", row.names = F)
```

#knn method

```
df = knnImputation(df)
sum(is.na(df))
```

#outlier analysis

```
for (i in 1:length(numerical_var))
{
  assign(paste0("bp",i), ggplot(aes_string(x="default",y = (numerical_var[i])), d=df)+
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.colour="blue", fill = "indianred3" ,outlier.shape=18,
                        outlier.size=2, notch=FALSE) +
           theme(legend.position="bottom")+
           labs(x="default status of loan cases", y=numerical_var[i])+
           ggtitle(paste("Box plot of default status of loan cases with",numerical_var[i])))
}
gridExtra::grid.arrange(bp1,ncol=1)
gridExtra::grid.arrange(bp2,ncol=1)
gridExtra::grid.arrange(bp3,ncol=1)
gridExtra::grid.arrange(bp4,ncol=1)
gridExtra::grid.arrange(bp5,ncol=1)
gridExtra::grid.arrange(bp6,ncol=1)
gridExtra::grid.arrange(bp7,ncol=1)
```

# Replace all outliers with NA and impute with KNN imputation

```
for(i in numerical_var){
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  df[,i][df[,i] %in% val] = NA
}
df = knnImputation(df)
```

#Correlation

```
corrgram(df[,numerical_var], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")
```

#Chi-squared Test of Independence - if p<0.05, reject null hypohesis, i.e keep the variable.

```
categorical_df = df[,categorical_var]
for (i in categorical_var){
  for (j in categorical_var){
    print(i)
    print(j)
    print(chisq.test(table(categorical_df[,i], categorical_df[,j]))$p.value)
  }
}
```

#ANOVA

```
anova_age =(lm(age ~ default, data = df))
summary(anova_age)
```

```r
anova_employstatus =(lm(employstatus ~ default, data = df))
summary(anova_employstatus)
anova_address =(lm(address ~ default, data = df))
summary(anova_address)
anova_income =(lm(income ~ default, data = df))
summary(anova_income)
anova_debtinc =(lm(debtinc ~ default, data = df))
summary(anova_debtinc)
anova_creddebt =(lm(creddebt ~ default, data = df))
summary(anova_creddebt)
anova_otherdebt =(lm(otherdebt ~ default, data = df))
summary(anova_otherdebt)

#multicollinearity
vif(df)

#Normalisation
for(i in numerical_var){
  print(i)
  df[,i] = (df[,i] - min(df[,i]))/
    (max(df[,i] - min(df[,i])))
}

# Visualization by rank using beeswarm-syle plots
ggplot(df, aes(x = df$default, y = df$age, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Age by Default Status",
       x = "Default Status",
       y = "Age") +
  theme_minimal() +
  theme(legend.position = "none")

ggplot(df, aes(x = df$default, y = df$employstatus, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Employment Status by Default Status",
       x = "Default Status",
       y = "Employment Status") +
  theme_minimal() +
  theme(legend.position = "none")

ggplot(df, aes(x = df$default, y = df$address, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Address by Default Status",
       x = "Default Status",
       y = "Address") +
  theme_minimal() +
  theme(legend.position = "none")

ggplot(df, aes(x = df$default, y = df$income, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Income by Default Status",
       x = "Default Status",
       y = "Income") +
  theme_minimal() +
  theme(legend.position = "none")

ggplot(df, aes(x = df$default, y = df$debtinc, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Debt to Income by Default Status",
       x = "Default Status",
       y = "Debt to Income") +
  theme_minimal() +
  theme(legend.position = "none")

ggplot(df, aes(x = df$default, y = df$creddebt, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Credit to Debt Ratio by Default Status",
       x = "Default Status",
       y = "Credit to Debt Ratio") +
  theme_minimal() +
  theme(legend.position = "none")

ggplot(df, aes(x = df$default, y = df$otherdebt, color = "rank")) +
  geom_quasirandom(alpha = 0.7, size = 1.5) +
  labs(title = "Other Debt by Default Status",
       x = "Default Status",
       y = "Other Debt") +
  theme_minimal() +
  theme(legend.position = "none")
```

#lets check categorical variables and target variables

```
for(i in 1:length(categorical_var))
{
  assign(paste0("b",i),ggplot(aes_string(y='default',x = (categorical_var[i])),
                              data=subset(df))+
           geom_bar(stat = "identity",fill = "green") +
           ggtitle(paste("Default Status with respect to",categorical_var[i])))+
    theme(axis.text.x = element_text( color="red", size=8))+
    theme(plot.title = element_text(face = "old"))
}
gridExtra::grid.arrange(b1,ncol=1)
```

#Clean the environment

```
rmExcept(c("original_data",'df'))
```

#Divide data into train and test using stratified sampling method

```
set.seed(1234)
train.index = createDataPartition(df$default, p = .80, list = FALSE)
train = df[ train.index,]
```

```
test  = df[-train.index,]
```

##Logistic Regression

```
logit_model = glm(default ~ ., data = train, family = "binomial")

#summary of the model
summary(logit_model)

#predict using logistic regression
logit_Predictions = predict(logit_model, newdata = test, type = "response")

#convert prob
logit_Predictions = ifelse(logit_Predictions > 0.5, 1, 0)

##Evaluate the performance of classification model
CM_LR = table(test$default, logit_Predictions)
confusionMatrix(CM_LR)
fourfoldplot(CM_LR)

# Area under ROC curve
roc.curve(test$default, logit_Predictions)

##False Negative rate
#FNR = FN/FN+TP

#Accuracy: 78.70%
#FNR: 75.00%
```

##Decision tree for classification

```
#Develop Model on training data
C50_model = C5.0(default ~., train, trials = 100, rules = TRUE)

#Summary of DT model
summary(C50_model)

#write rules into disk
write(capture.output(summary(C50_model)), "c50Rules.txt")

#Lets predict for test cases
C50_Predictions = predict(C50_model, test[,-9], type = "class")

##Evaluate the performance of classification model
CM_C50 = table(test$default, C50_Predictions)
confusionMatrix(CM_C50)
fourfoldplot(CM_C50)

# Area under ROC curve
roc.curve(test$default, C50_Predictions)

#Accuracy: 80.47%
```

```
#FNR: 60.00%
```

##Random Forest

```
RF_model = randomForest(default ~ ., train, importance = TRUE, ntree = 100)

#Extract rules fromn random forest
#transform rf object to an inTrees' format
treeList = RF2List(RF_model)

# Extract rules
exec = extractRules(treeList, train[,-9])  # R-executable conditions

# Visualize some rules
exec[1:2,]

# Make rules more readable:
readableRules = presentRules(exec, colnames(train))
readableRules[1:2,]

# Get rule metrics
ruleMetric = getRuleMetric(exec, train[,-9], train$default)  # get rule metrics

# evaulate few rules
ruleMetric[1:2,]

#Predict test data using random forest model
RF_Predictions = predict(RF_model, test[,-9])

##Evaluate the performance of classification model
CM_RF = table(test$default, RF_Predictions)
confusionMatrix(CM_RF)
fourfoldplot(CM_RF)

# Area under ROC curve
roc.curve(test$default, RF_Predictions)

#Accuracy = 81.07%
#FNR = 67.5%
```

##Random Forest1

```
RF_model1 = randomForest(default ~ ., train, importance = TRUE, ntree = 500)

#Extract rules fromn random forest
#transform rf object to an inTrees' format
treeList1 = RF2List(RF_model1)

# Extract rules
exec1 = extractRules(treeList1, train[,-9])  # R-executable conditions

# Visualize some rules
```

```
exec1[1:2,]

# Make rules more readable:
readableRules1 = presentRules(exec1, colnames(train))
readableRules1[1:2,]

# Get rule metrics
ruleMetric1 = getRuleMetric(exec1, train[,-9], train$default)  # get rule metrics

# evaulate few rules
ruleMetric1[1:2,]

#Predict test data using random forest model
RF_Predictions1 = predict(RF_model1, test[,-9])

##Evaluate the performance of classification model
CM_RF1 = table(test$default, RF_Predictions1)
confusionMatrix(CM_RF1)
fourfoldplot(CM_RF1)

# Area under ROC curve
roc.curve(test$default, RF_Predictions1)

#Accuracy = 81.07%
#FNR = 72.5%
```

##KNN Implementation

```
#Predict test data
KNN_Predictions = knn(train[, 1:8], test[, 1:8], train$default, k = 7)

#Confusion matrix
CM_KNN = table(test$default, KNN_Predictions)
confusionMatrix(CM_KNN)
fourfoldplot(CM_KNN)

# Area under ROC curve
roc.curve(test$default, KNN_Predictions)

#Accuracy
sum(diag(Conf_matrix))/nrow(test)

#Accuracy = 82.84%
#FNR = 21.05%
```

##Naive Bayes

```
#Develop model
NB_model = naiveBayes(default ~ ., data = train)

#predict on test cases #raw
NB_Predictions = predict(NB_model, test[,1:8], type = 'class')
```

```
#Look at confusion matrix
CM_NB = table(test$default, NB_Predictions)
confusionMatrix(CM_NB)
fourfoldplot(CM_NB)

# Area under ROC curve
roc.curve(test$default, NB_Predictions)

#Accuracy: 82.25%
#FNR: 42.5%

write.csv(df, "df.csv", row.names = F)
```

6. Python Code

```
import os
import pandas as pd
import numpy as np
import matplotlib as mlt
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from random import randrange, uniform
from scipy.stats import chi2_contingency
from fancyimpute import KNN

os.chdir ('C:/Users/Harshita Prasad/Desktop/loan default case')

original_data = pd.read_csv("bank-loan.csv", sep = ',')
df = original_data

df = df.rename(columns = {'ed':'education','employ':'employstatus','othdebt':'otherdebt'})

df.dtypes

df.describe()

numerical_var = ['age', 'employstatus', 'address', 'income', 'debtinc', 'creddebt', 'otherdebt']

categorical_var=['education', 'default']

df['education'] = df['education'].astype(object)
df['default'] = df['default'].astype(object)

#Pairplot for all numerical variables
pairwise_plot = sns.pairplot(data=df[numerical_var],kind='scatter')
pairwise_plot.fig.suptitle('Pairwise plot of all numerical variables')

# Univariate Analysis
sns.FacetGrid(df , height = 5).map(sns.distplot,'age').add_legend()
sns.FacetGrid(df , height = 5).map(sns.distplot,'employstatus').add_legend()
sns.FacetGrid(df , height = 5).map(sns.distplot,'address').add_legend()
sns.FacetGrid(df , height = 5).map(sns.distplot,'income').add_legend()
sns.FacetGrid(df , height = 5).map(sns.distplot,'debtinc').add_legend()
sns.FacetGrid(df , height = 5).map(sns.distplot,'creddebt').add_legend()
sns.FacetGrid(df , height = 5).map(sns.distplot,'otherdebt').add_legend()

#Missing value analysis
Missing_val = df.isnull().sum()
Missing_val
```

```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(df.isnull().sum())

#Reset index
missing_val = missing_val.reset_index()

#Rename variable
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})

#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(df))*100

#descending order
missing_val = missing_val.sort_values('Missing_percentage', ascending =
False).reset_index(drop = True)

#save output results
missing_val.to_csv("Missing_perc.csv", index = False)

#Apply KNN imputation algorithm
df = pd.DataFrame(KNN(k = 1).fit_transform(df), columns = df.columns)

#Box plot
for i in numerical_var:
    sns.boxplot(y=i,data=df)
    plt.title('Boxplot of '+i)
    plt.savefig('bp'+str(i)+'.png')
    plt.show()

##Outlier Analysis
def outlier_treatment(col):
    ''' calculating outlier indices and replacing them with NA  '''
    #Extract quartiles
    q75, q25 = np.percentile(df[col], [75 ,25])
    #Calculate IQR
    iqr = q75 - q25
    #Calculate inner and outer fence
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)
    #Replace with NA
    df.loc[df[col] < minimum,col] = np.nan
    df.loc[df[col] > maximum,col] = np.nan

outlier_treatment('employstatus')
outlier_treatment('address')
outlier_treatment('income')
outlier_treatment('debtinc')
outlier_treatment('creddebt')
outlier_treatment('otherdebt')
```

```
#Imputing with missing values using KNN
df = pd.DataFrame(KNN(k = 1).fit_transform(df), columns = df.columns, index=df.index)

##Correlation analysis
df_corr = df.loc[:,numerical_var]

#Set the width and height of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
corr = df_corr.corr()

#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)

#Chisquare test of independence
#loop for chi square values
for i in categorical_var:
    print(i)
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(df['default'], df[i]))
    print(p)

#if p<0.05, reject the null hypothesis i.e dont take the variable.

#Nomalisation
for i in numerical_var:
    print(i)
    df[i] = (df[i] - min(df[i]))/(max(df[i]) - min(df[i]))

#Divide data into train and test
X = df.values[:, 0:8]
Y = df.values[:,8]

X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.2)

#Create logistic data. Save target variable first
df_logit = pd.DataFrame(df['default'])

#Add continous variables
df_logit = df_logit.join(df[numerical_var])

##Create dummies for categorical variables
cat_var = ["education"]

for i in cat_var:
    temp = pd.get_dummies(df[i], prefix = i)
    df_logit = df_logit.join(temp)
Sample_Index = np.random.rand(len(df_logit)) < 0.8
```

```
train = df_logit[Sample_Index]
test = df_logit[~Sample_Index]

df_logit.shape

#select column indexes for independent variables
train_cols = train.columns[1:13]

np.asarray(df)

#Built Logistic Regression
import statsmodels.api as sm

logit = sm.Logit(train['default'], train[train_cols]).fit()

logit.summary()

#Predict test data
test['Actual_prob'] = logit.predict(test[train_cols])

test['ActualVal'] = 1
test.loc[test.Actual_prob < 0.5, 'ActualVal'] = 0

#Build confusion matrix
CM = pd.crosstab(test['default'], test['ActualVal'])

#check accuracy of model
#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

#accuracy_score(y_test, y_pred)*100
((TP+TN)*100)/(TP+TN+FP+FN)

#false negative rate
(FN*100)/(FN+TP)

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
import pylab as pl

# Compute Precision-Recall and plot curve
precision, recall, thresholds = precision_recall_curve(test['default'], test['ActualVal'])
area = auc(recall, precision)
print ("Area Under Curve: %0.2f" % area)

pl.clf()
pl.plot(recall, precision, label='Precision-Recall curve')
pl.xlabel('Recall')
pl.ylabel('Precision')
pl.ylim([0.0, 1.05])
pl.xlim([0.0, 1.0])
pl.title('Precision-Recall example: AUC=%0.2f' % area)
pl.legend(loc="lower left")
pl.show()

##Decision tree
#Import Libraries for decision tree
from sklearn import tree
from sklearn.metrics import accuracy_score

#replace target categories with Yes or No
df['default'] = df['default'].replace(0, 'No')
df['default'] = df['default'].replace(1, 'Yes')

#Decision Tree
C50_model = tree.DecisionTreeClassifier(criterion='entropy').fit(X_train, y_train)

#predict new test cases
C50_Predictions = C50_model.predict(X_test)

#build confusion matrix
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test, C50_Predictions)

CM = pd.crosstab(y_test, C50_Predictions)
CM

#check accuracy of model
accuracy_score(y_test, C50_Predictions)*100

#False Negative rate
(FN*100)/(FN+TP)

# Compute Precision-Recall and plot curve
precision, recall, thresholds = precision_recall_curve(y_test, C50_Predictions)
area = auc(recall, precision)
print ("Area Under Curve: %0.2f" % area)

pl.clf()
pl.plot(recall, precision, label='Precision-Recall curve')
pl.xlabel('Recall')
pl.ylabel('Precision')
pl.ylim([0.0, 1.05])
pl.xlim([0.0, 1.0])
pl.title('Precision-Recall example: AUC=%0.2f' % area)
pl.legend(loc="lower left")
pl.show()

#Random Forest
from sklearn.ensemble import RandomForestClassifier

RF_model = RandomForestClassifier(n_estimators = 100).fit(X_train, y_train)

RF_Predictions = RF_model.predict(X_test)

#build confusion matrix
CM = confusion_matrix(y_test, RF_Predictions)
CM = pd.crosstab(y_test, RF_Predictions)
CM

#check accuracy of model
accuracy_score(y_test, RF_Predictions)*100

#False Negative rate
(FN*100)/(FN+TP)

# Compute Precision-Recall and plot curve
precision, recall, thresholds = precision_recall_curve(y_test, RF_Predictions)
area = auc(recall, precision)
print ("Area Under Curve: %0.2f" % area)

pl.clf()
pl.plot(recall, precision, label='Precision-Recall curve')
pl.xlabel('Recall')
pl.ylabel('Precision')
pl.ylim([0.0, 1.05])
pl.xlim([0.0, 1.0])
pl.title('Precision-Recall example: AUC=%0.2f' % area)
pl.legend(loc="lower left")
pl.show()

#KNN implementation
from sklearn.neighbors import KNeighborsClassifier

KNN_model = KNeighborsClassifier(n_neighbors = 1).fit(X_train, y_train)

#predict test cases
KNN_Predictions = KNN_model.predict(X_test)

#build confusion matrix
CM = pd.crosstab(y_test, KNN_Predictions)

#check accuracy of model
#accuracy_score(y_test, y_pred)*100
((TP+TN)*100)/(TP+TN+FP+FN)
#False Negative rate
```

```
(FN*100)/(FN+TP)

# Compute Precision-Recall and plot curve
precision, recall, thresholds = precision_recall_curve(y_test, KNN_Predictions)
area = auc(recall, precision)
print ("Area Under Curve: %0.2f" % area)

pl.clf()
pl.plot(recall, precision, label='Precision-Recall curve')
pl.xlabel('Recall')
pl.ylabel('Precision')
pl.ylim([0.0, 1.05])
pl.xlim([0.0, 1.0])
pl.title('Precision-Recall example: AUC=%0.2f' % area)
pl.legend(loc="lower left")
pl.show()

#Naive Bayes
from sklearn.naive_bayes import GaussianNB

#Naive Bayes implementation
NB_model = GaussianNB().fit(X_train, y_train)

#predict test cases
NB_Predictions = NB_model.predict(X_test)

#Build confusion matrix
CM = pd.crosstab(y_test, NB_Predictions)

#check accuracy of model
#accuracy_score(y_test, y_pred)*100
((TP+TN)*100)/(TP+TN+FP+FN)

#False Negative rate
(FN*100)/(FN+TP)

# Compute Precision-Recall and plot curve
precision, recall, thresholds = precision_recall_curve(y_test, NB_Predictions)
area = auc(recall, precision)
print ("Area Under Curve: %0.2f" % area)

pl.clf()
pl.plot(recall, precision, label='Precision-Recall curve')
pl.xlabel('Recall')
pl.ylabel('Precision')
pl.ylim([0.0, 1.05])
pl.xlim([0.0, 1.0])
pl.title('Precision-Recall example: AUC=%0.2f' % area)
pl.legend(loc="lower left")
pl.show()
```