Homework #3 DATA622

Group 1

9/23/2021

Group 1 Members:

- David Moste
- Vinayak Kamath
- Kenan Sooklall
- Christian Thieme
- Lidiia Tronina

Introduction

We have been provided a dataset containing the loan approval status of 614 loan applicants. In addition to the status, we have also been given 12 additional columns of data describing different characteristics of each loan applicant. Our task is to explore the data, and then use the knowledge gained to to create models using Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Decision Trees, and Random Forests. Finally, we'll compare and contrast each method and its associated accuracy metrics.

The data has the following variables:

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/Undergraduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit History	credit history meets guidlines
Property_Area	Urban/Semi Urban/Rural
Loan_Status (Target)	Loan approved (Y/N)

As can be seen from the table above, the dataset has a wide variety of numeric and categorical data. We'll dive deeper into this data now.

Exploratory Data Analysis

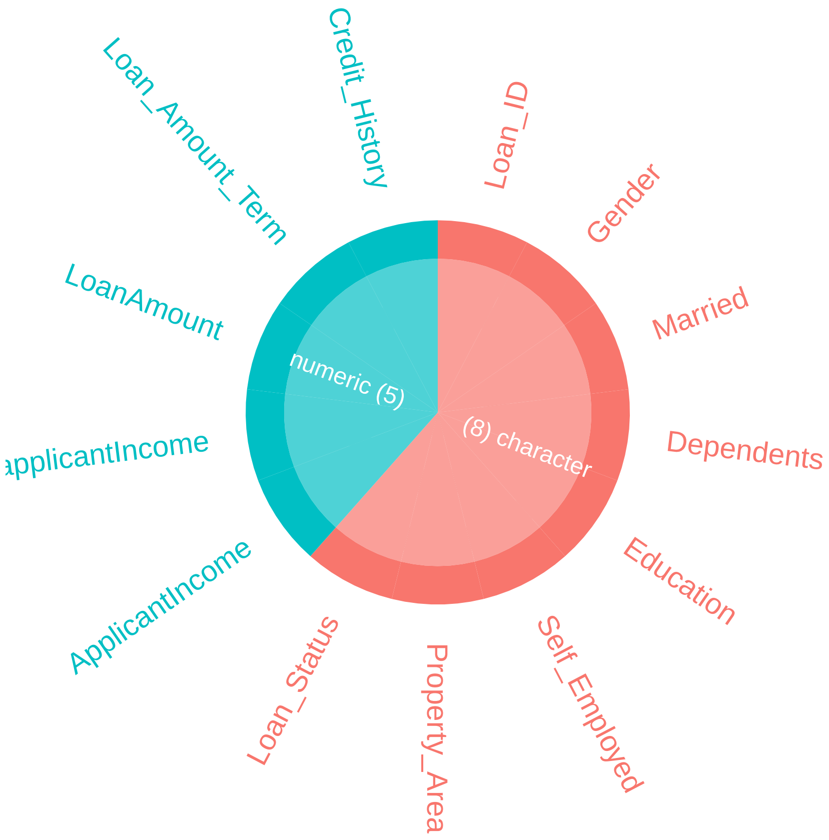
Let's begin exploring by taking a high level look at our dataset:

```
glimpse(loans)
```

```
## Rows: 614
## Columns: 13
## $ Loan_ID           <chr> "LP001002", "LP001003", "LP001005", "LP001006", "LP0~
## $ Gender            <chr> "Male", "Male", "Male", "Male", "Male", "Male", "Mal~
## $ Married           <chr> "No", "Yes", "Yes", "Yes", "No", "Yes", "Yes", "Yes"~
## $ Dependents        <chr> "0", "1", "0", "0", "0", "2", "0", "3+", "2", "1", "~
## $ Education         <chr> "Graduate", "Graduate", "Graduate", "Not Graduate", ~
## $ Self_Employed     <chr> "No", "No", "Yes", "No", "No", "Yes", "No", "No", "N~
## $ ApplicantIncome   <dbl> 5849, 4583, 3000, 2583, 6000, 5417, 2333, 3036, 4006~
## $ CoapplicantIncome <dbl> 0, 1508, 0, 2358, 0, 4196, 1516, 2504, 1526, 10968, ~
## $ LoanAmount        <dbl> NA, 128, 66, 120, 141, 267, 95, 158, 168, 349, 70, 1~
## $ Loan_Amount_Term  <dbl> 360, 360, 360, 360, 360, 360, 360, 360, 360, 360, 36~
## $ Credit_History    <dbl> 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, NA, ~
## $ Property_Area     <chr> "Urban", "Rural", "Urban", "Urban", "Urban", "Urban"~
## $ Loan_Status       <chr> "Y", "N", "Y", "Y", "Y", "Y", "Y", "N", "Y", "N", "Y~
```

From this view we can see that this dataset has 614 rows and 13 columns. As mentioned previously, there is a mix of both numeric and categorical data, although it appears that at least one numeric column should actually be categorical. As it stands, the split between numeric and categorical data looks like this:

```
inspectdf::inspect_types(loans) %>%
  show_plot
```



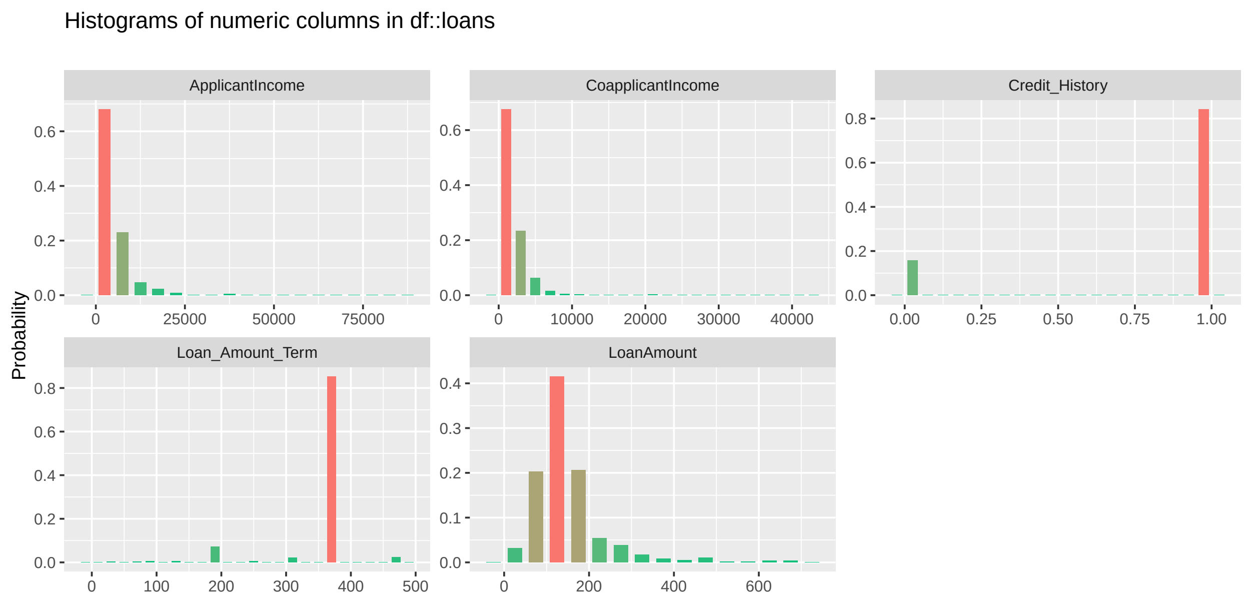
Continuous and categorical variables will be analyzed separately for the sake of clarity. We'll begin our analysis by looking at our numeric features.

Numeric Features

	n	min	mean	median	max	sd
ApplicantIncome	614	150	5403.4593	3812.5	81000	6109.04167
CoapplicantIncome	614	0	1621.2458	1188.5	41667	2926.24837
LoanAmount	592	9	146.4122	128.0	700	85.58733
Loan__Amount__Term	600	12	342.0000	360.0	480	65.12041

Looking at the above summary table, we can see that there are outliers present in all numerical variables, with ApplicantIncome being the worst offender. Even at three times the standard deviation, its maximum value lies far outside of the 68-95-99.7 rule. There is also a significant difference between its mean and median, indicating there is skew present in this variable as well before any charts are actively looked at. Futher, we'll look at each of these features distribution:

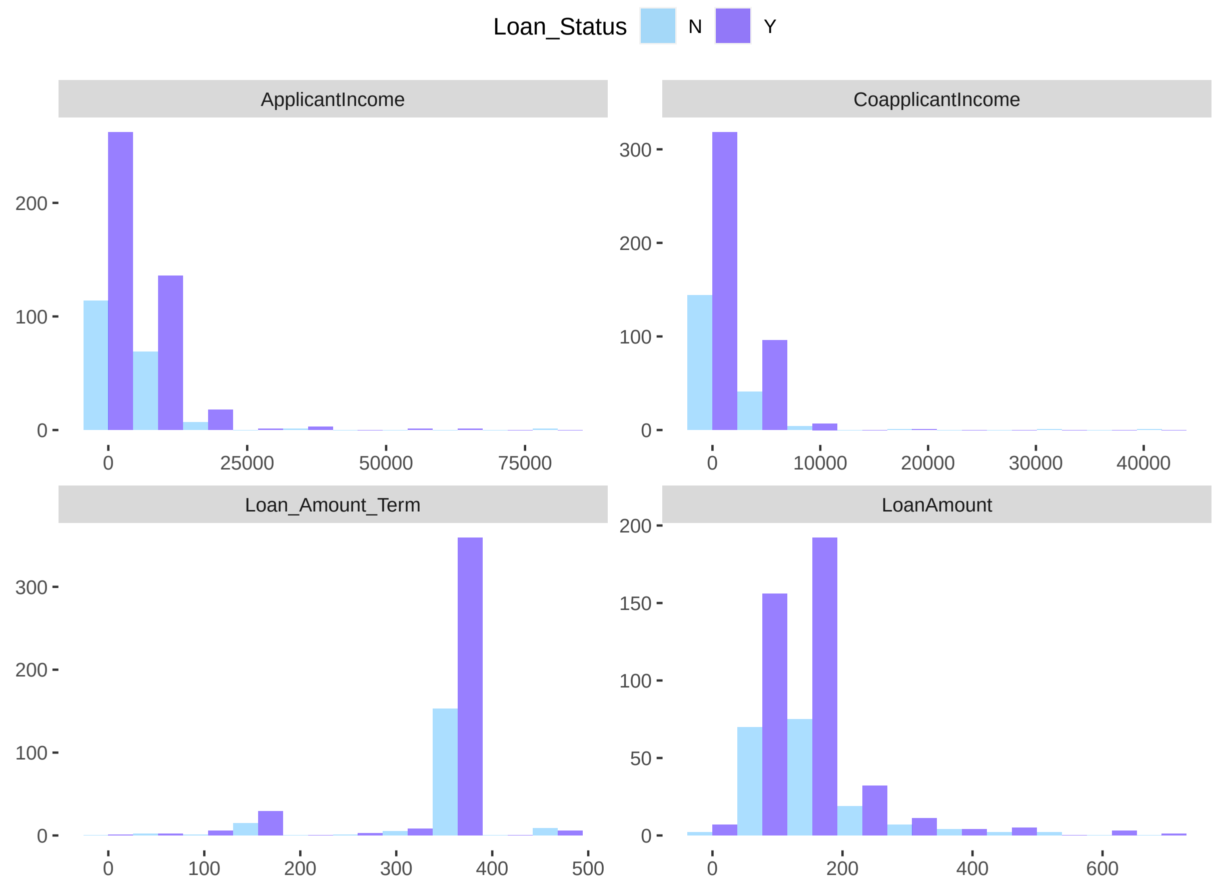
```
inspectdf::inspect_num(loans) %>%
 show_plot()
```



Looking at the above plots, we can see that:

- `ApplicantIncome` must be on a weekly or monthly basis as most values are fairly low
- `CoapplicantIncome` follows the same trend as applicant income
- `Credit_History` appears to be a categorical variable either 0 or 1
- `Loan_Amount_Term` appears to be 365 days (1 year) more than 80% of the time
- `LoanAmount` is fairly small, ranging from \$0-700. It appears that most loans are for somewhere between \$50 and \$200.

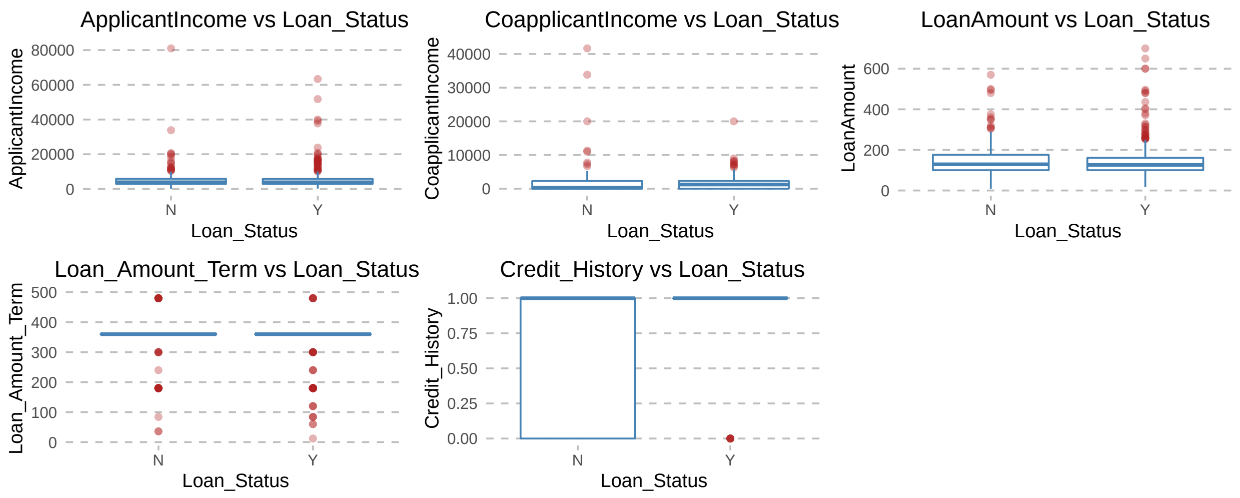
It will also be helpful to understand the shape of the distributions of these variables for each application approval status, both Yes and No.



There does not seem to be a *significant* difference between those who get loan approval and those who do not for any of the predictors.

As part of our analysis of numeric features, let's look at the relationship between these features and `Loan_Status`:

```r
loan_names <- loans %>% select_if(is.numeric)# %>% select(-Credit_History)
int_names <- names(loan_names)
myGlist <- vector('list', length(loan_names))
names(myGlist) <- int_names

for (i in int_names) {

 myGlist[[i]] <-
     ggplot(loans) +
     aes_string(x = as.factor(loans$Loan_Status), y = i) +
     geom_boxplot(color = 'steelblue',
                  outlier.color = 'firebrick',
                  outlier.alpha = 0.35) +
      labs(title = paste0(i,' vs Loan_Status'), y = i, x= 'Loan_Status') +
      theme_minimal() +
      theme(
        plot.title = element_text(hjust = 0.45),
        panel.grid.major.y =  element_line(color = "grey",
                                           linetype = "dashed"),
        panel.grid.major.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        axis.ticks.x = element_line(color = "grey")
      )

    }
  myGlist <- within(myGlist, rm(target_name))
  gridExtra::grid.arrange(grobs = myGlist, ncol = 3)
```



These charts are pretty hard to read, and again, we don't see any significant relationships between these variables and `Loan_Status`. We note that we need to change `Credit_History` to a categorical variable.

Categorical Features

Next, let's turn our attention to our categorical variables. We have both binary categorical variables, and variables with 3 or more classes.

Table 3: Summary statistics for Categorical Variables

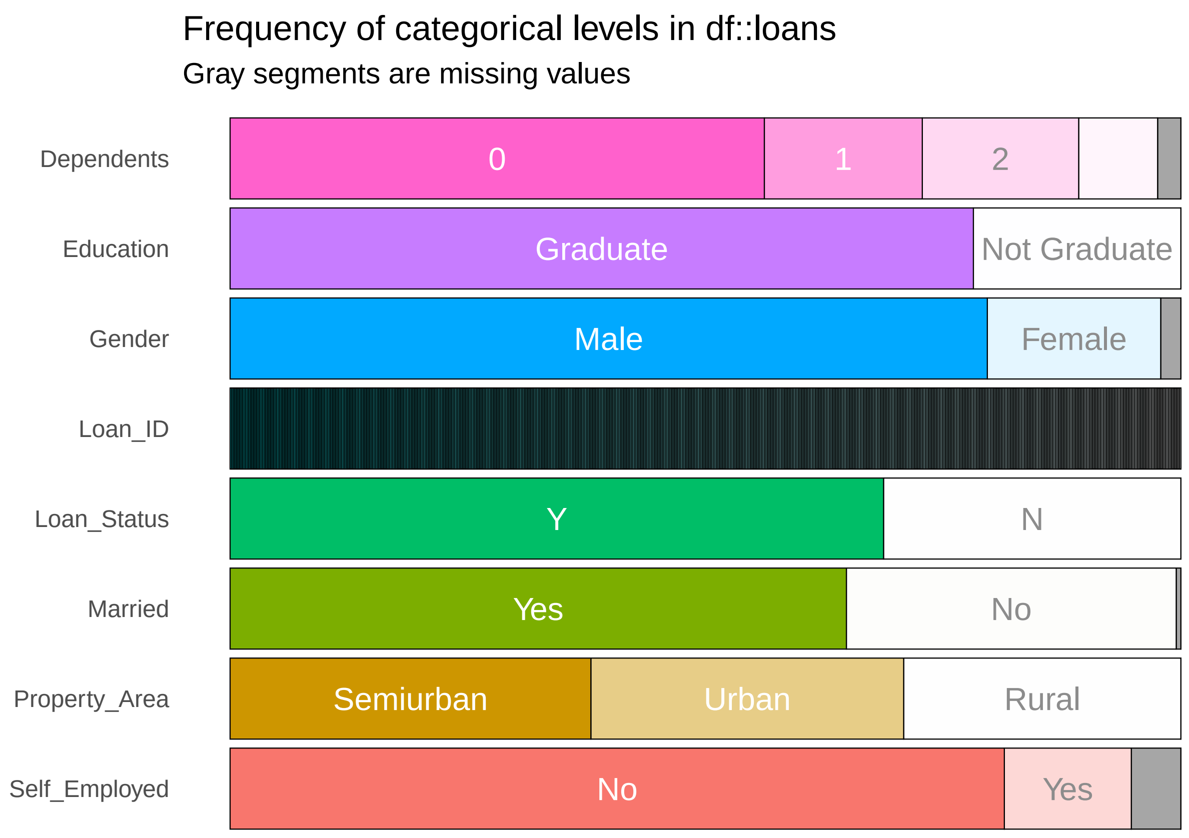
Dependents	Property_Area
0 :345	Rural :179
1 :102	Semiurban:233
2 :101	Urban :202
3+ : 51	NA
NA's: 15	NA

Table 4: Summary statistics for Binary Categorical Variables

Gender	Married	Education	Self_Employed	Credit_History	Loan_Status
Female:112	No :213	Graduate :480	No :500	0 : 89	N:192
Male :489	Yes :398	Not Graduate:134	Yes : 82	1 :475	Y:422
NA's : 13	NA's: 3	NA	NA's: 32	NA's: 50	NA

`Dependents` and `Property_Area` each comprise multiple categories. On the other hand, `Gender`, `Married`, `Education`, `Self_Employed`, `Credit_History`, `Loan_Status` are all binaries. These summary tables are helpful, although sometimes its easier to view this data in a visual format:

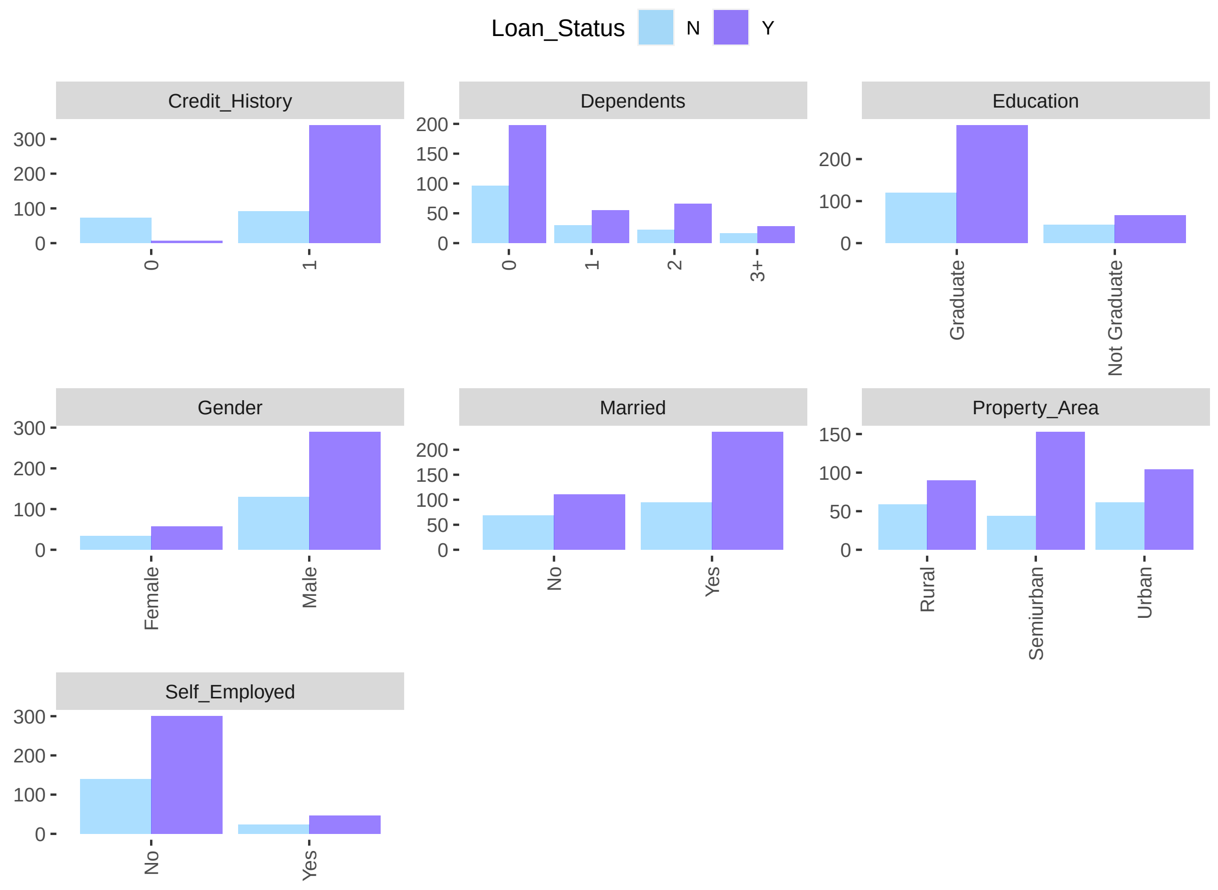
```r
inspect_cat(loans) %>%
  show_plot()
```



In looking at the chart and the summary tables, we see some very interesting things about the demographic of this dataset:

- Half of the population do not have any dependents
- Most people in this dataset are graduates
- Over 75% of the population is male
- 65% of the population is married
- Most of the individuals are not self-employed

We can also examine the dispersion of approval status between these variables.

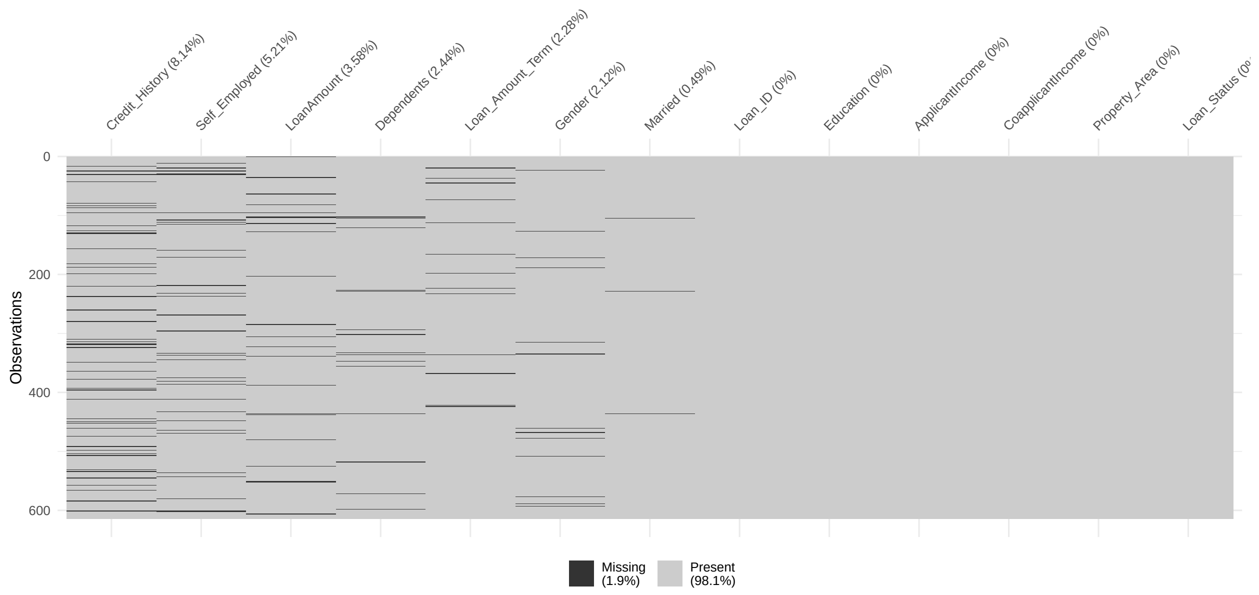


It looks like likelihoods are higher for applicants who have credit history and who are married and live in Semi-Urban area; as well as for those with a graduate degree.

Missing Data

The data appears to have some missing values that we'll have to deal with as we move along. Since we know there are missing values, let's see how pervasive the issue is:

```r
visdat::vis_miss(loans, sort_miss = TRUE)
```



7 columns have missing data, ranging from 0.49% to 8.14%. What's interesting is there are 22 instances where `LoanAmount` is missing. In total, ~2% of the data has missing values.

Having completed our EDA, we'll now turn our attention to modeling.

Modeling

LDA

First, we need to split the data into training(80%) and test dataset(20%).

```
set.seed(123)
training.samples <- loans2$Loan_Status %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data <- loans2[training.samples, ]
test.data <- loans2[-training.samples, ]
```

The purpose of the linear discriminant analysis is to find combination of the variables that give best possible separation between groups in our data set.

The LDA output indicates that our prior probabilities are No = 0.309 and Yes = 0.690; in other words, 69% of the training observations are customers who got loan approval. It also provides the group means; these are the average of each predictor within each class, and are used by LDA as estimates. These suggest that customers that have approved applications, on average, have a lower loan amount and are more likely to be Male and be married with a credit history and from Semi-Urban Area. The coefficients of linear discriminants output provide the linear combination of variables that are used to form the LDA decision rule.

```
# Run LDA (from MASS library)
lda.loans <- MASS::lda(Loan_Status ~ Gender +Married +Dependents+Education +ApplicantIncome +Coapplican
              +LoanAmount + Loan_Amount_Term + Credit_History +Property_Area +Self_Employed , data = tr
lda.loans
```

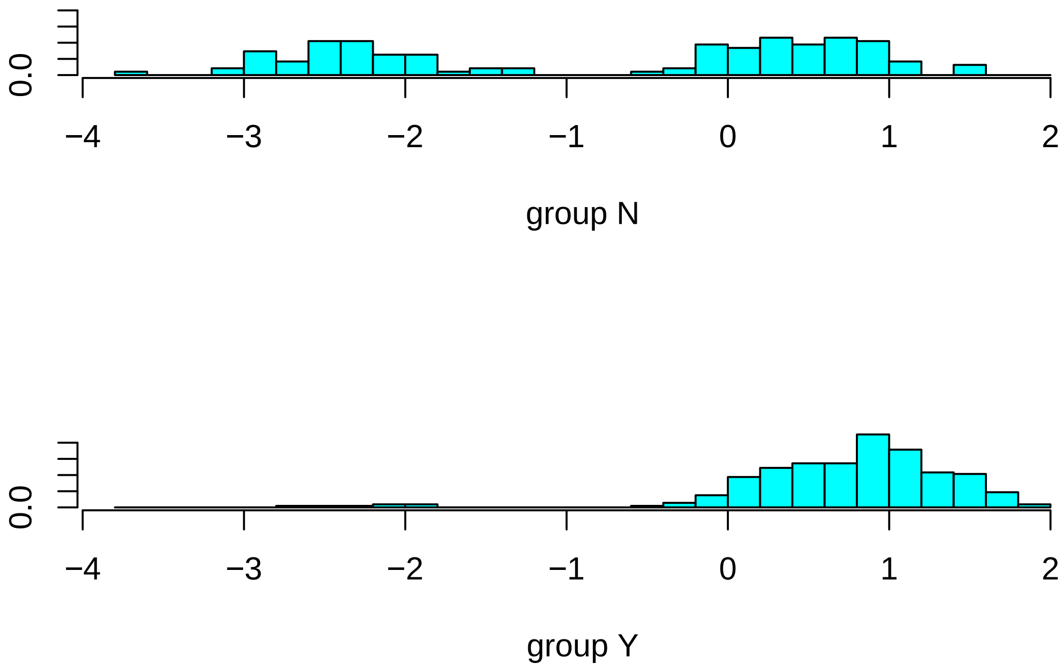
```
## Call:
## lda(Loan_Status ~ Gender + Married + Dependents + Education +
##     ApplicantIncome + CoapplicantIncome + LoanAmount + Loan_Amount_Term +
##     Credit_History + Property_Area + Self_Employed, data = train.data)
##
## Prior probabilities of groups:
##         N         Y
## 0.3090909 0.6909091
##
## Group means:
##   GenderMale MarriedYes Dependents1 Dependents2 Dependents3+
## N  0.7731092  0.5294118   0.1764706   0.1428571   0.09243697
## Y  0.8345865  0.6879699   0.1390977   0.2030075   0.08646617
##   EducationNot Graduate ApplicantIncome CoapplicantIncome LoanAmount
## N             0.2689076        5786.269          1768.336   150.0168
## Y             0.1842105        5018.222          1473.688   139.8083
##   Loan_Amount_Term Credit_History1 Property_AreaSemiurban Property_AreaUrban
## N         342.6555       0.5798319              0.2941176          0.3781513
## Y         340.3308       0.9736842              0.4586466          0.2894737
##   Self_EmployedYes
## N        0.1596639
## Y        0.1278195
##
## Coefficients of linear discriminants:
##                                  LD1
## GenderMale              2.433286e-01
## MarriedYes              6.018814e-01
## Dependents1            -4.065561e-01
## Dependents2             1.673478e-01
## Dependents3+            1.791003e-02
## EducationNot Graduate  -3.881465e-01
## ApplicantIncome        -5.456522e-06
## CoapplicantIncome      -5.389572e-05
## LoanAmount             -1.340982e-03
## Loan_Amount_Term       -8.934374e-04
## Credit_History1         2.965301e+00
## Property_AreaSemiurban  5.587254e-01
## Property_AreaUrban     -8.155911e-02
## Self_EmployedYes       -1.966809e-01
```

The linear discriminant function from the result in above is

$$0.24*GenderMale+0.602*MarriedYes-0.406*Dependents1+0.167*Dependents2+0.017*Dependents3-0.388*EducationN$$

We can use plot() to produce plots of the linear discriminants obtained by computing this formula for each training observation.

```
plot(lda.loans)
```



As you can see, when it's greater than 0, the probability increases that the customer will get approved.

Prediction accuracy of LDA compares prediction results from the model output with the actual data using the confusion matrix.

```
lda.predictions <- lda.loans %>% predict(test.data)
#Confusion Matrix
table(lda.predictions$class, test.data$Loan_Status)
```

```
##
##      N  Y
##   N 13  0
##   Y 16 66
```

Correct classification rate is 83%

```
#Prediction Accuracy
mean(lda.predictions$class== test.data$Loan_Status)
```

```
## [1] 0.8315789
```

KNN

To start with, knn is reliant on there being no missing data. To fill in the gaps, we can either impute or omit our missing values. Since we have already omitted these observations for LDA, we'll continue with that method here and directly pull from our loans2 dataset.

Following the required omission/imputation of missing data, we need to do a little bit of pre-processing. KNN is highly susceptible to data that is on different scales (large values will be much further from each other than small values). With this in mind, we have chosen to center and scale all of our predictors. The final step of pre-processing is to remove any predictors that have near-zero variance so that there are no overlapping predictors.

```
library(caret)
library(e1071)
knn_features <- subset(loans2, select=-c(Loan_Status))
knn_trans <- preProcess(knn_features,
                        method = c("center", "scale"))
knn_transformed_feat <- predict(knn_trans, knn_features)
nzv <- nearZeroVar(knn_transformed_feat, saveMetrics = TRUE)
nzv[nzv[,"nzv"] == TRUE,]
```

```
## [1] freqRatio     percentUnique zeroVar       nzv
## <0 rows> (or 0-length row.names)
```

It turns out that none of our predictors have near-zero variance, so we're good to proceed!

At this point, we are ready to build our model. We'll start by splitting our data into training and testing sets. We also need to remove Loan_ID since it will throw off our KNN model (a random categorical variable that has nothing to do with the response).

```
knn_processed <- cbind(loans2[,13], knn_transformed_feat)
knn_processed <- subset(knn_processed, select=-c(Loan_ID))
names(knn_processed)[1] <- ("Loan_Status")
knn_processed <- knn_processed[complete.cases(knn_processed),]
set.seed(54321)
train_ind <- sample(seq_len(nrow(knn_processed)),
                    size = floor(0.75*nrow(knn_processed)))
knn_train <- knn_processed[train_ind,]
knn_test <- knn_processed[-train_ind,]
```

Our KNN model uses the kknn library. With this library we are able to test different distances (Manhattan, Euclidean, etc.) as well as different weights (kernels). Our model found that a k value of around 9 with a distance of 5 and a weighting function of rectangular produced the best model with an accuracy of 82.5%.

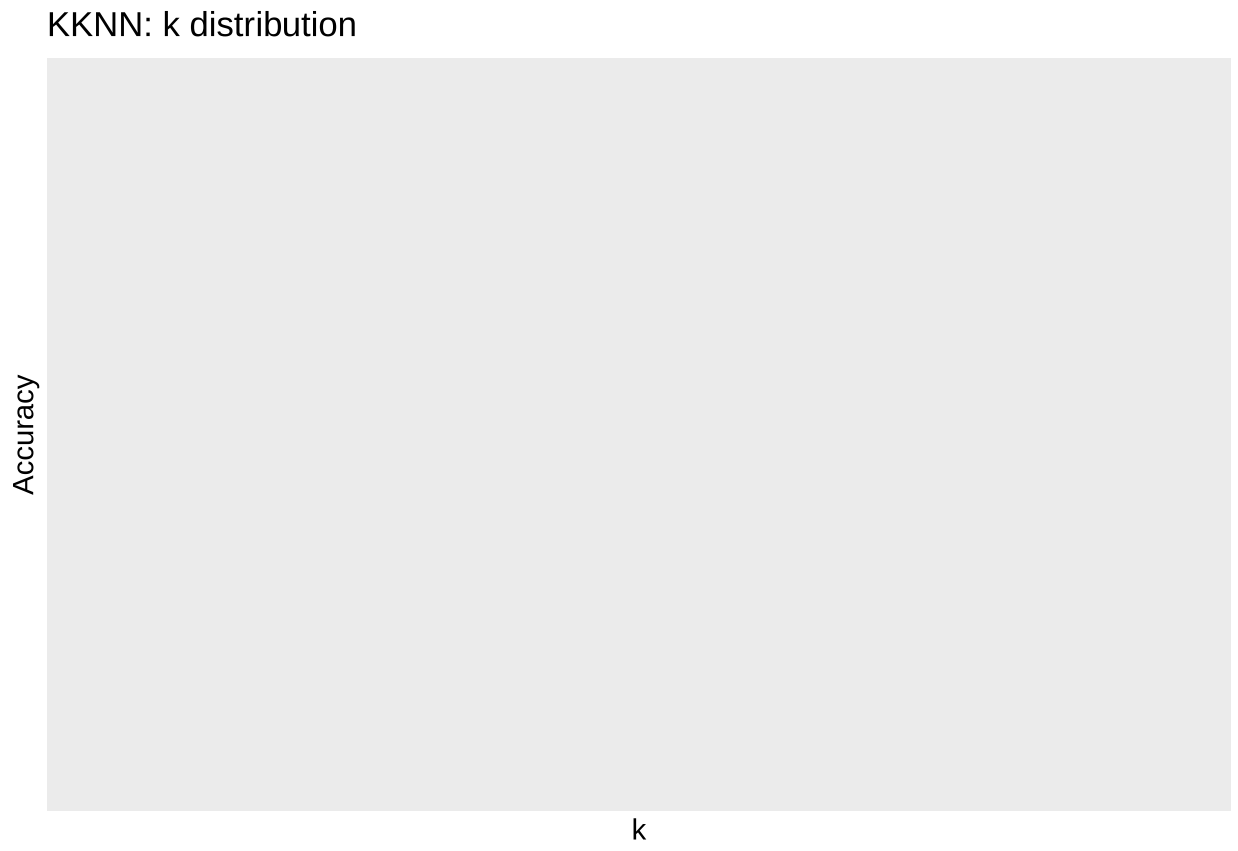
```
library(kknn)
```

```
##
## Attaching package: 'kknn'

## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
library(ggplot2)
kknn_func <- function(train_x, train_y, test_x, test_y){
  acc_df <- data.frame(matrix(nrow = 0, ncol = 4))

  weights <- c("rectangular","triangular",
               "biweight","triweight")

  for(d in 1:10){
    for(w in weights){
      for(i in 2:50){
        kknnModel <- kknn(train_y ~ .,
                          train_x,
                          test_x,
                          k = i,
                          distance = d,
                          kernel = w)

        cM <- table(test_y, fitted(kknnModel))
        accuracy <- (cM[1]+cM[4])/(cM[1]+cM[2]+cM[3]+cM[4])
        acc_df <- rbind(acc_df,c(i,accuracy,w,d))
      }
    }
  }
  colnames(acc_df) <- c("k", "Accuracy","Weight","Distance")
  acc_df[,1] <- as.integer(acc_df[,1])
  acc_df[,2] <- as.numeric(acc_df[,2])
  acc_df[,4] <- as.integer(acc_df[,4])
  return(acc_df)
}
kknn_acc <- kknn_func(knn_train[,-1],
                      knn_train[,1],
                      knn_test[,-1],
                      knn_test[,1])
head(kknn_acc[order(-kknn_acc$Accuracy),], n = 10)
```

```
##     k Accuracy      Weight Distance
## 1   1        1 rectangular        1
## 10 NA        1 rectangular        1
## 11 NA        1 rectangular        1
## 20 NA        1 rectangular        1
## 21 NA        1 rectangular        1
## 24 NA        1 rectangular        1
## 25 NA        1 rectangular        1
## 30 NA        1 rectangular        1
## 31 NA        1 rectangular        1
## 50  1        1        <NA>        1
```

```
acc_plot_data <- kknn_acc[which(kknn_acc$Distance == 5),]
ggplot(data = acc_plot_data, aes(x = k, y = Accuracy, color = Weight)) +
  geom_line() +
  geom_point() +
  labs(title = "KKNN: k distribution",
       x = "k",
       y = "Accuracy")
```

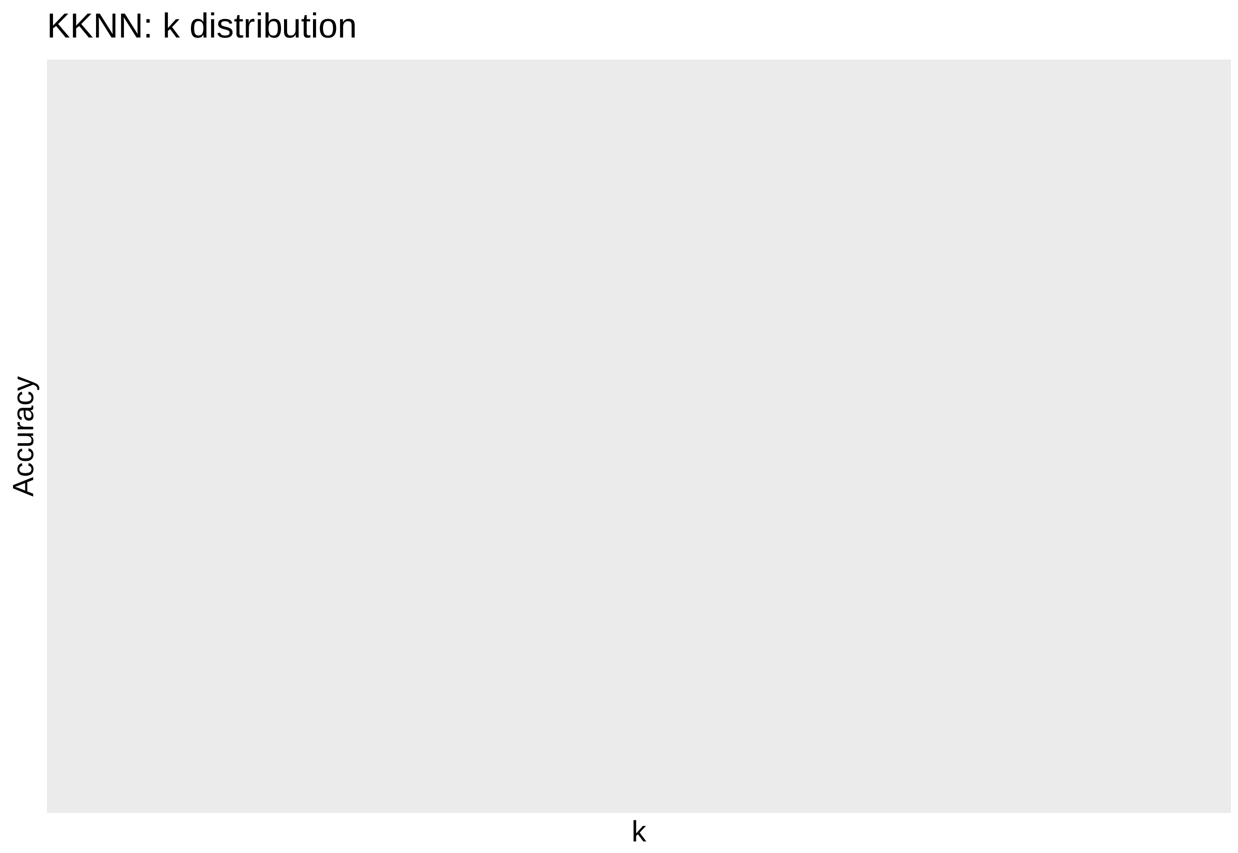


Just to validate the results obtained, we randomized our training and testing sets one more time and checked to see if the results were similar.

```
set.seed(12345)
train_ind <- sample(seq_len(nrow(knn_processed)),
                    size = floor(0.75*nrow(knn_processed)))
knn_train <- knn_processed[train_ind,]
knn_test <- knn_processed[-train_ind,]
library(kknn)
library(ggplot2)
kknn_func <- function(train_x, train_y, test_x, test_y){
  acc_df <- data.frame(matrix(nrow = 0, ncol = 4))

  weights <- c("rectangular","triangular",
               "biweight","triweight")

  for(d in 1:10){
    for(w in weights){
      for(i in 2:50){
        kknnModel <- kknn(train_y ~ .,
                          train_x,
                          test_x,
                          k = i,
                          distance = d,
                          kernel = w)

        cM <- table(test_y, fitted(kknnModel))
        accuracy <- (cM[1]+cM[4])/(cM[1]+cM[2]+cM[3]+cM[4])
        acc_df <- rbind(acc_df,c(i,accuracy,w,d))
      }
    }
  }
  colnames(acc_df) <- c("k", "Accuracy","Weight","Distance")
  acc_df[,1] <- as.integer(acc_df[,1])
  acc_df[,2] <- as.numeric(acc_df[,2])
  acc_df[,4] <- as.integer(acc_df[,4])
  return(acc_df)
}
kknn_acc <- kknn_func(knn_train[,-1],
                      knn_train[,1],
                      knn_test[,-1],
                      knn_test[,1])
head(kknn_acc[order(-kknn_acc$Accuracy),], n = 10)
```

```
##      k Accuracy      Weight Distance
## 1    1        1 rectangular        1
## 50   1        1        <NA>        1
## 99   1        1        <NA>        1
## 100 NA        1        <NA>        1
## 101 NA        1        <NA>        1
## 148  1        1        <NA>        1
## 149 NA        1        <NA>        1
## 150 NA        1        <NA>        1
## 296 NA        1        <NA>       NA
## 346 NA        1        <NA>       NA
```

```
acc_plot_data <- kknn_acc[which(kknn_acc$Distance == 5),]
ggplot(data = acc_plot_data, aes(x = k, y = Accuracy, color = Weight)) +
  geom_line() +
  geom_point() +
  labs(title = "KKNN: k distribution",
       x = "k",
       y = "Accuracy")
```



After tuning our hyper parameters again, we arrived at a similar result: a k value of around 9 and a rectangular weighting provided us with the best version of a KNN model.

Decision Tree

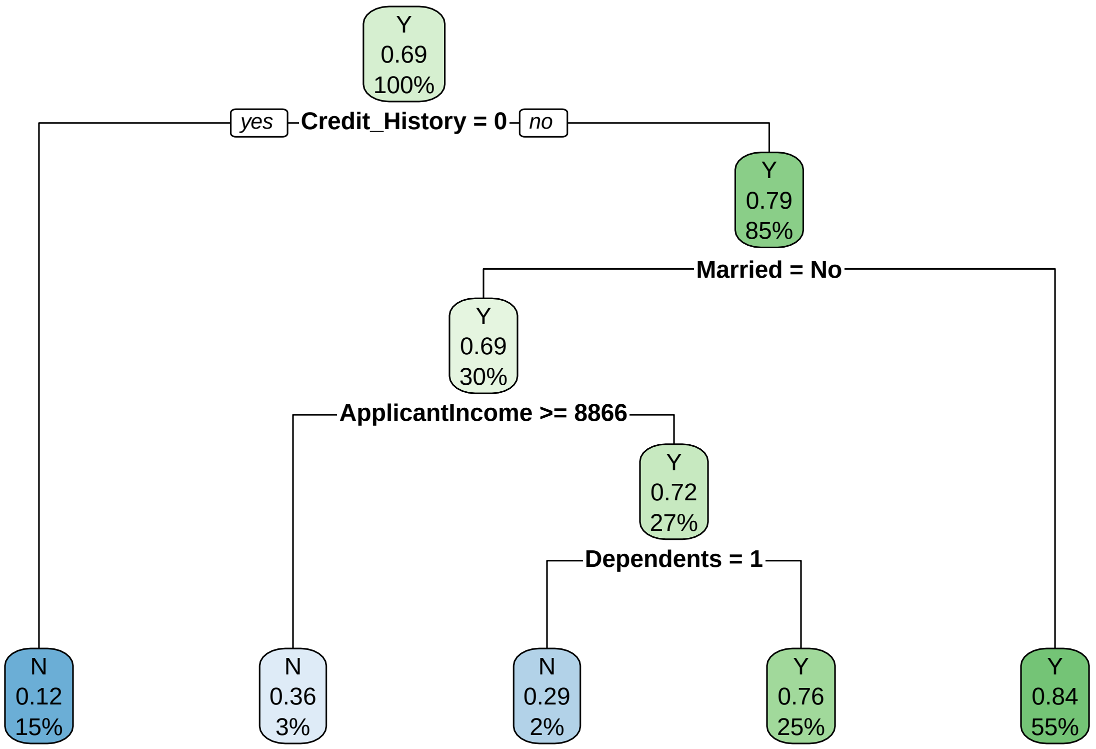
A decision tree is a supervised machine learning algorithm that can be used for both classification and regression problems. A decision tree is simply a series of sequential decisions made to reach a specific result.

```
library(rpart)
features <- c('Gender', 'Married', 'Dependents', 'Education', 'ApplicantIncome',
              'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History',
              'Property_Area', 'Self_Employed')
dt <- rpart(Loan_Status~., data=train.data %>% dplyr::select(c(all_of(features), Loan_Status)), method =
dt
```

```
## n= 385
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 385 119 Y (0.3090909 0.6909091)
##    2) Credit_History=0 57   7 N (0.8771930 0.1228070) *
##    3) Credit_History=1 328  69 Y (0.2103659 0.7896341)
##      6) Married=No 116  36 Y (0.3103448 0.6896552)
##       12) ApplicantIncome>=8866 11   4 N (0.6363636 0.3636364) *
##       13) ApplicantIncome< 8866 105  29 Y (0.2761905 0.7238095)
##         26) Dependents=1 7   2 N (0.7142857 0.2857143) *
##         27) Dependents=0,2,3+ 98  24 Y (0.2448980 0.7551020) *
##      7) Married=Yes 212  33 Y (0.1556604 0.8443396) *
```

The decision tree approves 70% of the loans and denies 30%.

```
library(rpart.plot)
rpart.plot(dt)
```



From the plot above we can see that having no credit history is a major factor in determining if the loan will be approved or not, followed by marriage, income and dependents. On closer examination having a good credit history and being married gives the highest chance to being approved.

```
dt.predictions <- predict(dt, test.data, type='class')
confusionMatrix(table(dt.predictions, test.data$Loan_Status), positive='Y')
```

```
## Confusion Matrix and Statistics
##
##
## dt.predictions  N  Y
##              N 14  7
##              Y 15 59
```

```
##
##                Accuracy : 0.7684
##                  95% CI : (0.6706, 0.8488)
##     No Information Rate : 0.6947
##     P-Value [Acc > NIR] : 0.07119
##
##                   Kappa : 0.4083
##
##  Mcnemar's Test P-Value : 0.13559
##
##             Sensitivity : 0.8939
##             Specificity : 0.4828
##          Pos Pred Value : 0.7973
##          Neg Pred Value : 0.6667
##              Prevalence : 0.6947
##          Detection Rate : 0.6211
##    Detection Prevalence : 0.7789
##       Balanced Accuracy : 0.6883
##
##        'Positive' Class : Y
##
```

The confusion matrix shows we correctly predicted Y 59 and N 14 times. Our model has higher precision (Pos Pred Value) than recall (Neg Perd Value). So if someone qualifies for a loan that is more accurate than someone being denied a loan.

Random Forest

The decision tree algorithm is quite easy to understand and interpret. But often, a single tree is not sufficient for producing effective results. This is where the Random Forest algorithm comes into the picture

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:psych':
##
##     outlier
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
rf <- randomForest(Loan_Status~., data=train.data %>% dplyr::select(c(all_of(features), Loan_Status)), 
rf
```

```
##
## Call:
##  randomForest(formula = Loan_Status ~ ., data = train.data %>%      dplyr::select(c(all_of(features)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 20.78%
## Confusion matrix:
##    N   Y class.error
## N 55  64  0.53781513
## Y 16 250  0.06015038
```

The random forest was made with 500 trees.

```
rf.predictions <- predict(rf, test.data, type='class')
confusionMatrix(table(rf.predictions, test.data$Loan_Status), positive='Y')
```

```
## Confusion Matrix and Statistics
##
##
## rf.predictions  N  Y
##              N 15  7
##              Y 14 59
##
##                Accuracy : 0.7789
##                  95% CI : (0.6822, 0.8577)
##     No Information Rate : 0.6947
##     P-Value [Acc > NIR] : 0.04436
##
##                   Kappa : 0.441
##
##  Mcnemar's Test P-Value : 0.19043
##
##             Sensitivity : 0.8939
##             Specificity : 0.5172
##          Pos Pred Value : 0.8082
##          Neg Pred Value : 0.6818
##              Prevalence : 0.6947
##          Detection Rate : 0.6211
##    Detection Prevalence : 0.7684
##       Balanced Accuracy : 0.7056
##
##        'Positive' Class : Y
##
```

The confusion matrix shows we correctly predicted Y 58 and N 15 times. Our model has higher recall (Neg Perd Value) than precision (Pos Pred Value). So if someone does not qualify for a loan that is more accurate than if someone qualifies.

Model Comparison

We will now compare our Decision Tree and Random Forest models to interpret their common classification metrics and determine which has the greatest predictive accuracy.

```r
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 3.6.3
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
DT_Model <- confusionMatrix(dt.predictions, test.data$Loan_Status)$byClass
DT_Model <- data.frame(DT_Model)

RF_Model <- confusionMatrix(rf.predictions, test.data$Loan_Status)$byClass
RF_Model <- data.frame(RF_Model)

compare <- data.frame(DT_Model, RF_Model)

compare %>%  kableExtra::kbl() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed",
```

We can see from the above that the random forest tree has the better performance then the decision tree model for many of the parameters like Sensitivity, Neg Pred Value and Balanced Accuracy. Thus, RF Model performs better predicting loan approvals and predicting loan rejections.

Summary

The random forest looks a better choice in our case here. While decision tree are easy to interpret, the random forest does a better handling with noise. Also with a slightly higher Accuracy of 0.7789 against the Accuracy of 0.7684 of the decision tree; our random forest is the better performing model of the two.

	DT_Model	RF_Model
Sensitivity	0.4827586	0.5172414
Specificity	0.8939394	0.8939394
Pos Pred Value	0.6666667	0.6818182
Neg Pred Value	0.7972973	0.8082192
Precision	0.6666667	0.6818182
Recall	0.4827586	0.5172414
F1	0.5600000	0.5882353
Prevalence	0.3052632	0.3052632
Detection Rate	0.1473684	0.1578947
Detection Prevalence	0.2210526	0.2315789
Balanced Accuracy	0.6883490	0.7055904