Loan Prediction

Rohit Dixit

Problem Statement

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

Data Glimpse

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
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 guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status Loan	approved (Y/N)

R Code

Importing Library

```
library(caret)
library(mlbench)
library(ggplot2)
library(ggthemes)
library(plyr)
library(RANN)
library(gridExtra)
library(caTools)
```

Data Importing

Data available at https://datahack-prod.s3.ap-south-1.amazonaws.com/train_file/train_u6lujuX_

CVtuZ9i.csv

```
data <- read.csv(url("https://datahack-prod.s3.ap-south-1.amazonaws.com/train_file/train_u6lujuX
```

Data Exploration

```
summary(data)
```

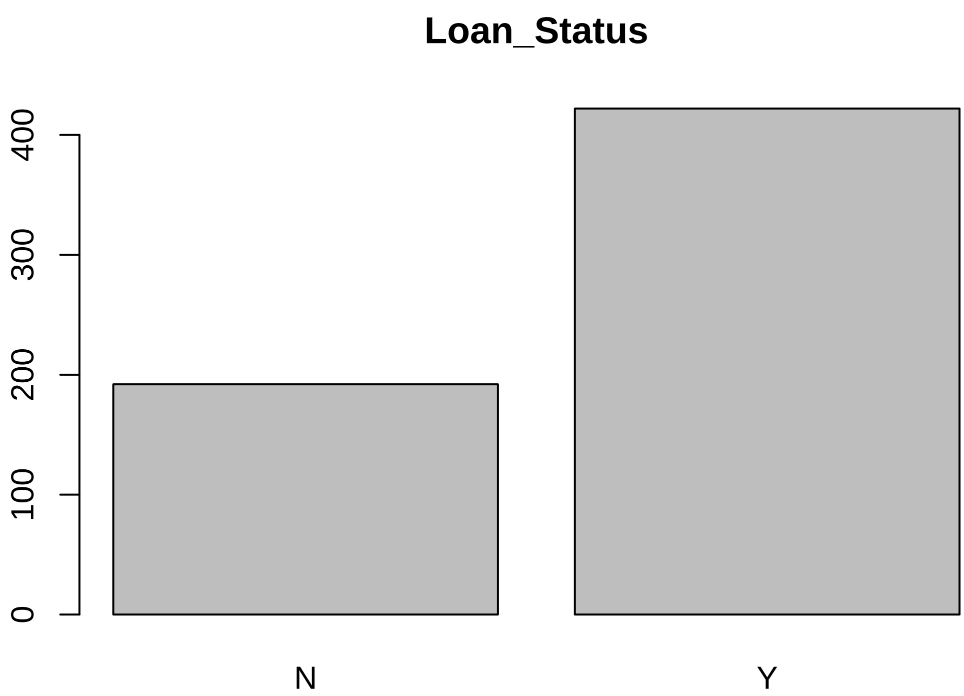
```
##       Loan_ID        Gender     Married    Dependents        Education
##  LP001002:  1          : 13        :  3      : 15     Graduate     :480
##  LP001003:  1    Female:112    No :213    0  :345     Not Graduate:134
##  LP001005:  1    Male  :489    Yes:398    1  :102
##  LP001006:  1                             2  :101
##  LP001008:  1                             3+: 51
##  LP001011:  1
##  (Other) :608
##  Self_Employed ApplicantIncome CoapplicantIncome   LoanAmount
##     : 32       Min.   :  150   Min.   :    0      Min.   :  9.0
##  No :500       1st Qu.: 2878   1st Qu.:    0      1st Qu.:100.0
##  Yes: 82       Median : 3812   Median : 1188      Median :128.0
##                Mean   : 5403   Mean   : 1621      Mean   :146.4
##                3rd Qu.: 5795   3rd Qu.: 2297      3rd Qu.:168.0
##                Max.   :81000   Max.   :41667      Max.   :700.0
##                                                   NA's   :22
##  Loan_Amount_Term Credit_History     Property_Area Loan_Status
##  Min.   : 12      Min.   :0.0000   Rural    :179   N:192
##  1st Qu.:360      1st Qu.:1.0000   Semiurban:233   Y:422
##  Median :360      Median :1.0000   Urban    :202
##  Mean   :342      Mean   :0.8422
##  3rd Qu.:360      3rd Qu.:1.0000
##  Max.   :480      Max.   :1.0000
##  NA's   :14       NA's   :50
```

The summary shows, we have NA values to handle let's explore our data more.
##Checking our target variable-**Loan_Status**

```
table(data$Loan_Status)
```

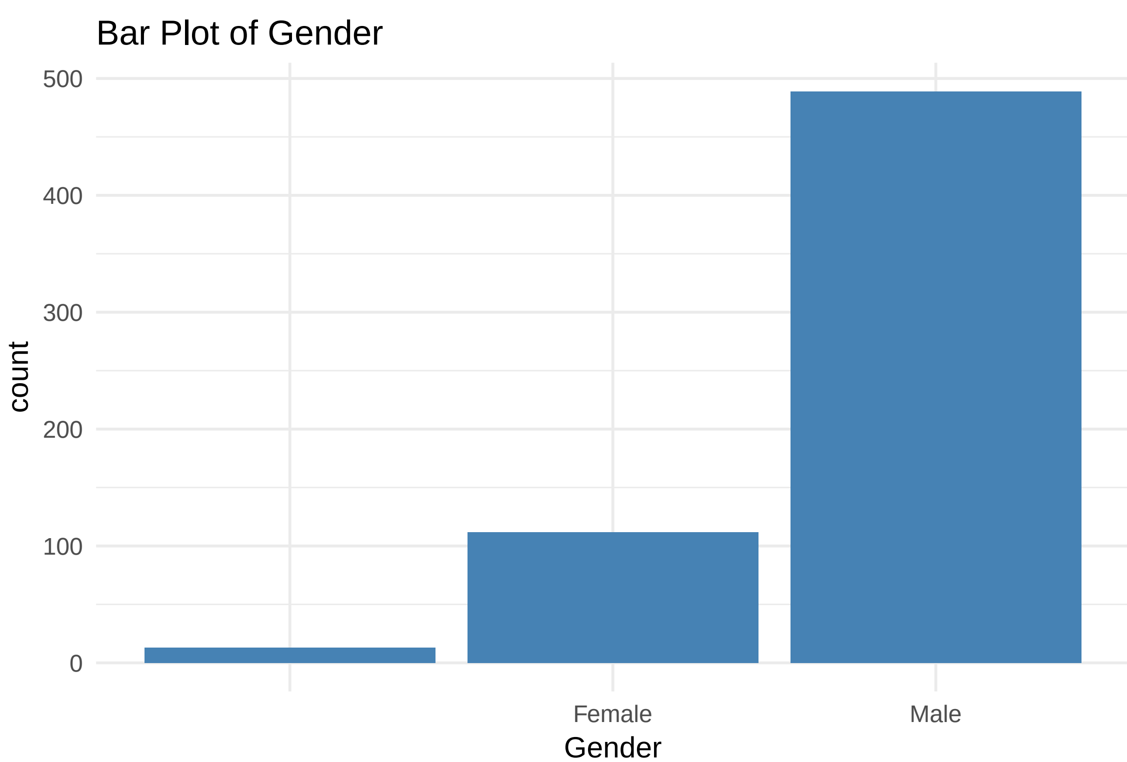
```
##
##   N   Y
## 192 422
```

```
barplot(table(data$Loan_Status),main= "Loan_Status")
```



Lets's Explore our Independent variables-

1. Gender

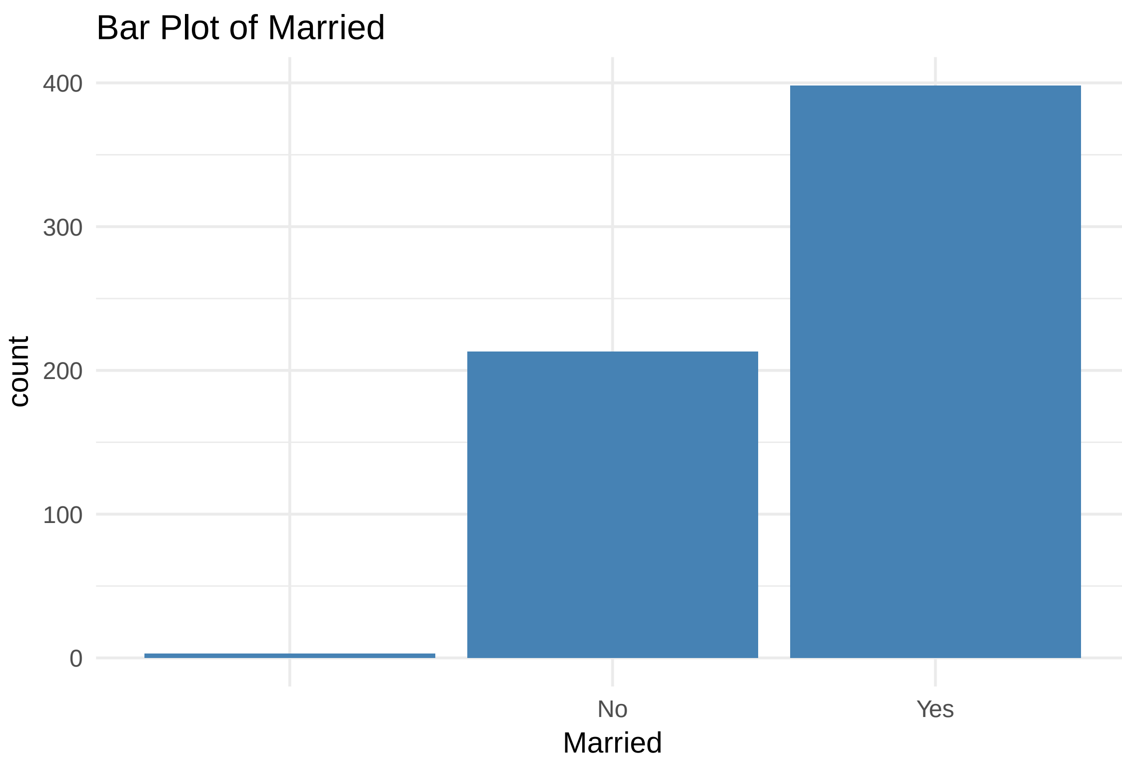


- Take Away points:
- Majority of are Male applicants

- We have Unknwon Gender

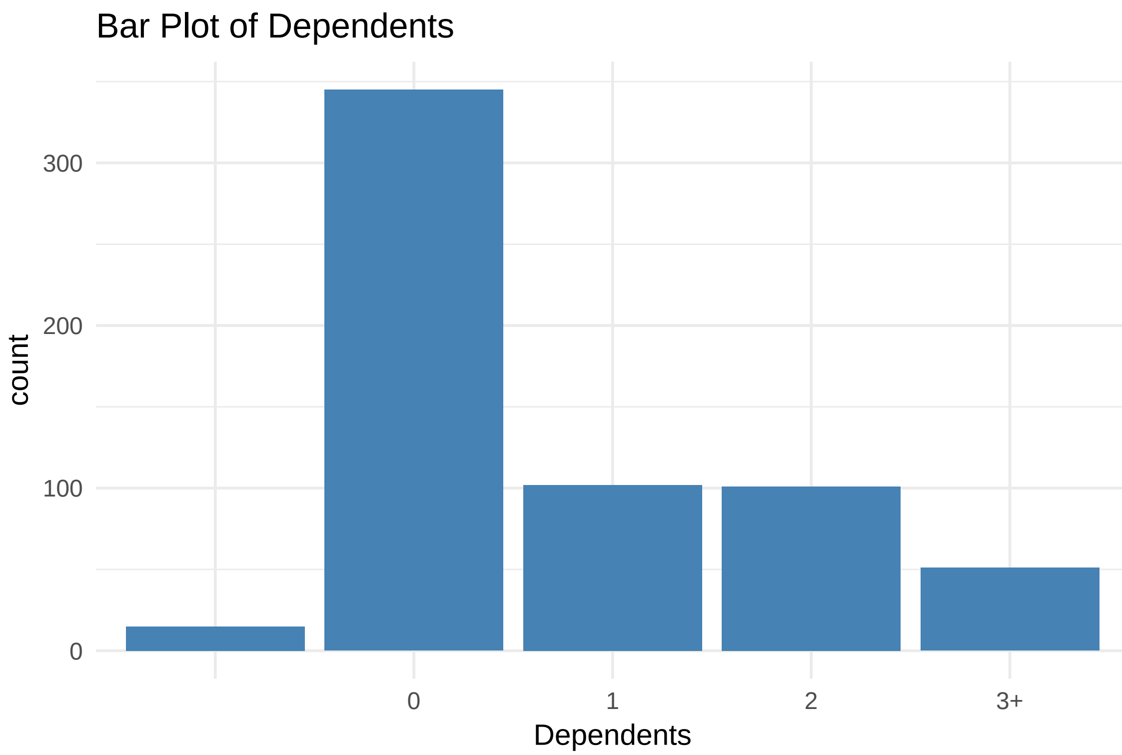
---

2. Married



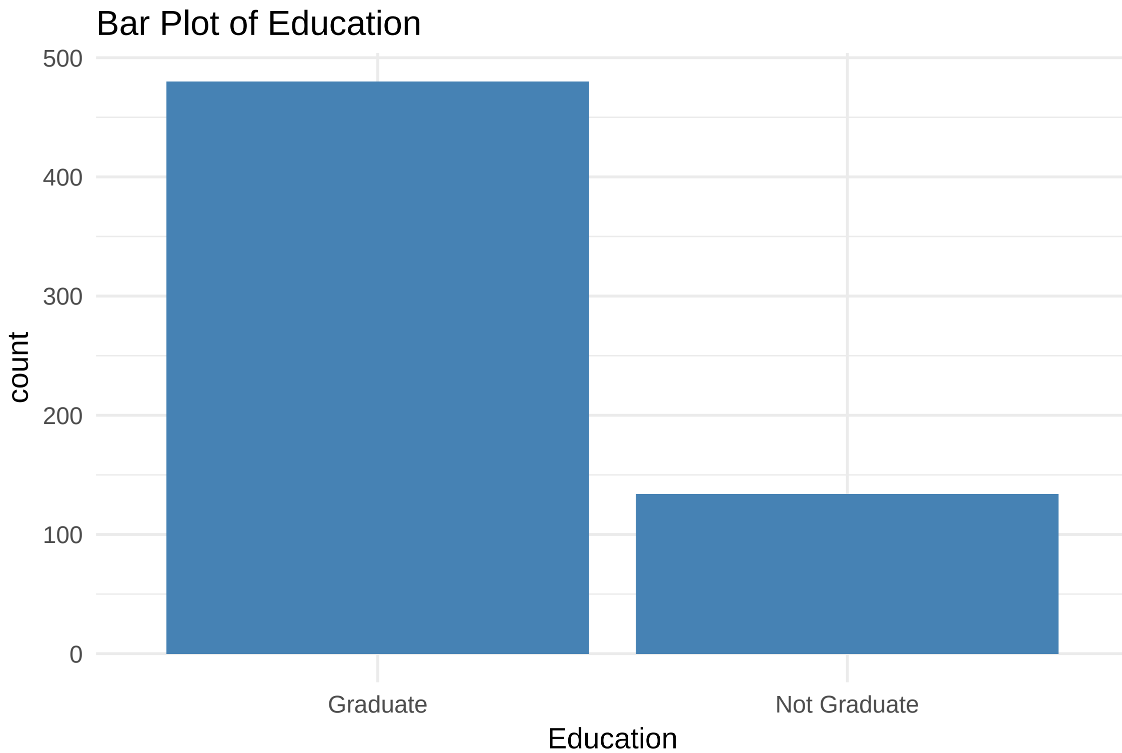
- Take Away points:
- Majority of applicants are Married
- We have Unknwon, So preprocessing will be needed

3. Dependents



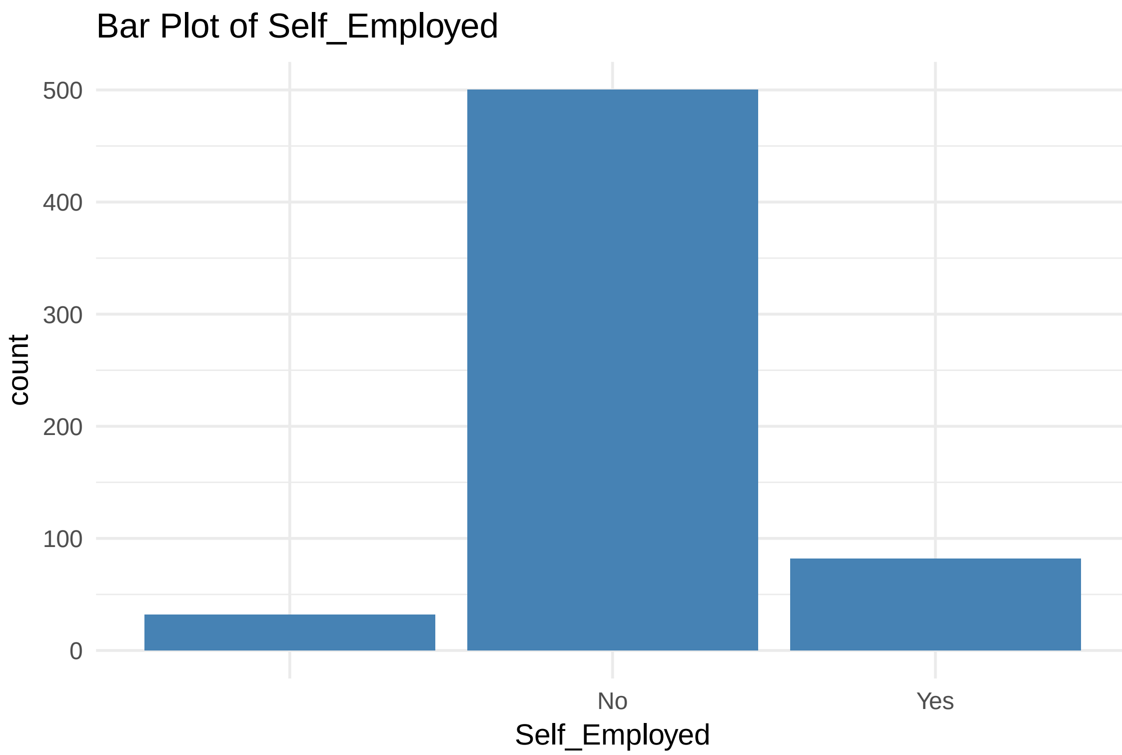
- Take Away points:
- Majority of applicants have no dependents
- We have Unknwon

4. Education



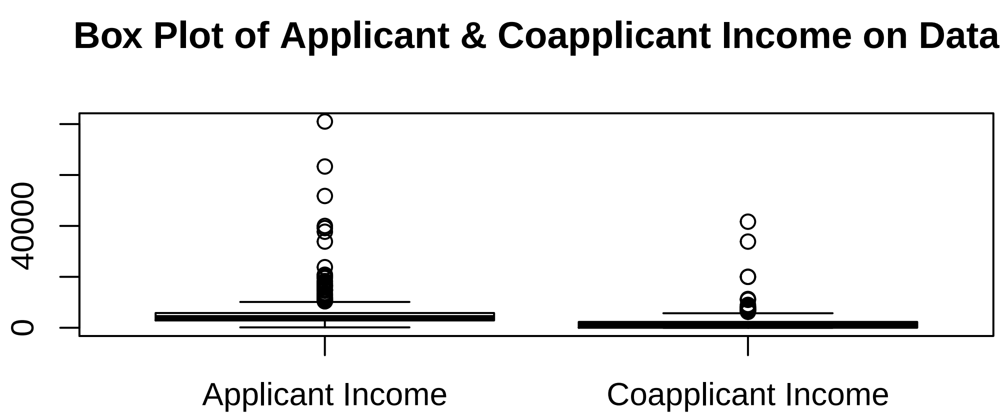
- Take Away points:
- Majority of are Graduates

5. Self_Employed



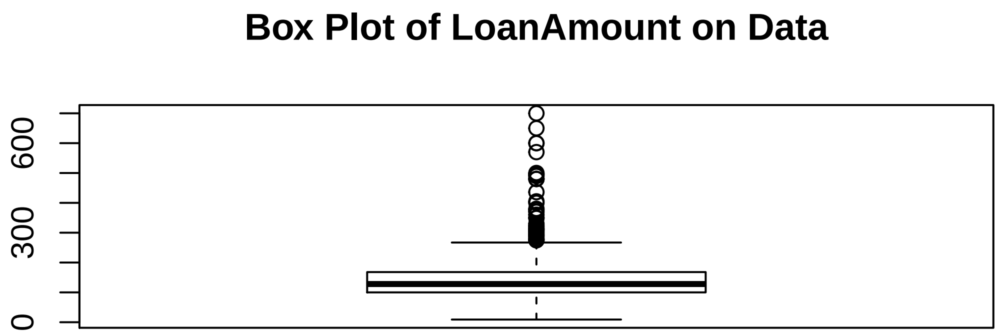
- Take Away points:
- Majority of applicants are not self employed
- We have Unknwon values, so preprocessing needed

6. ApplicantIncome (Numeric) & CoapplicantIncome (Numeric)



- Take Away points:
- Plots show right skewness
- We have outliers so scaling and centering willl be needed

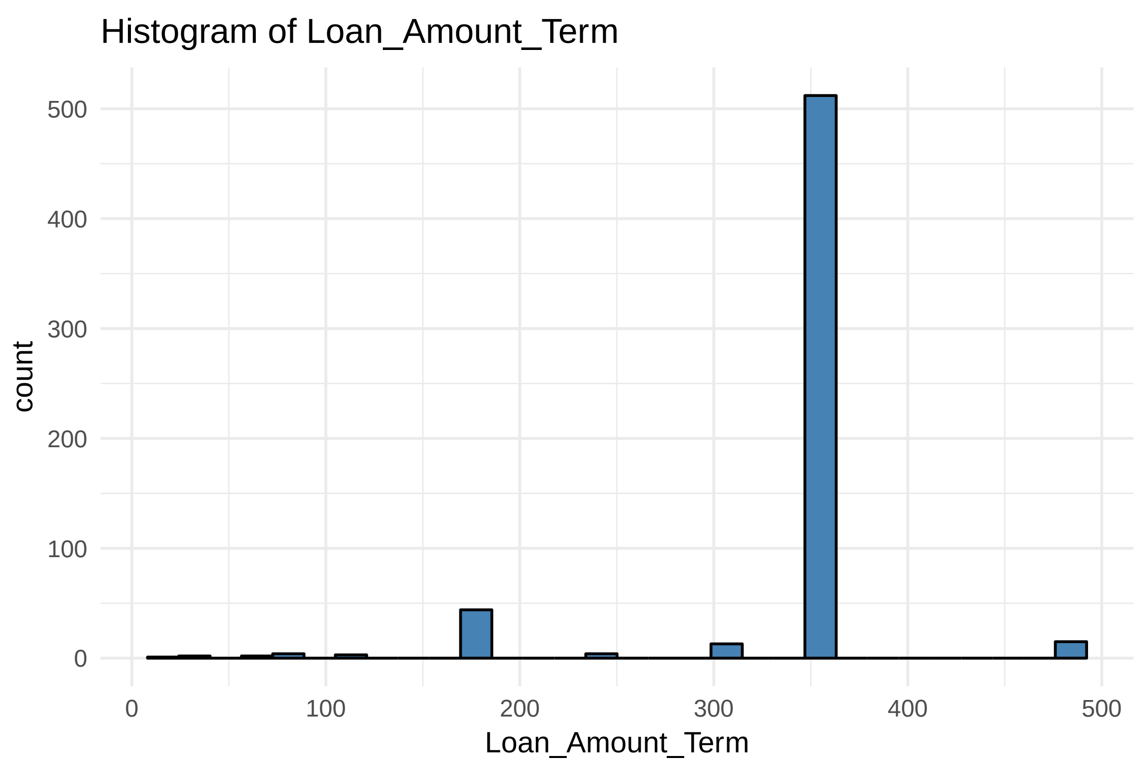
7. LoanAmount (Numeric)



- Take Away points:
- Plots show right skewness
- We have outliers so scaling and centering willl be needed

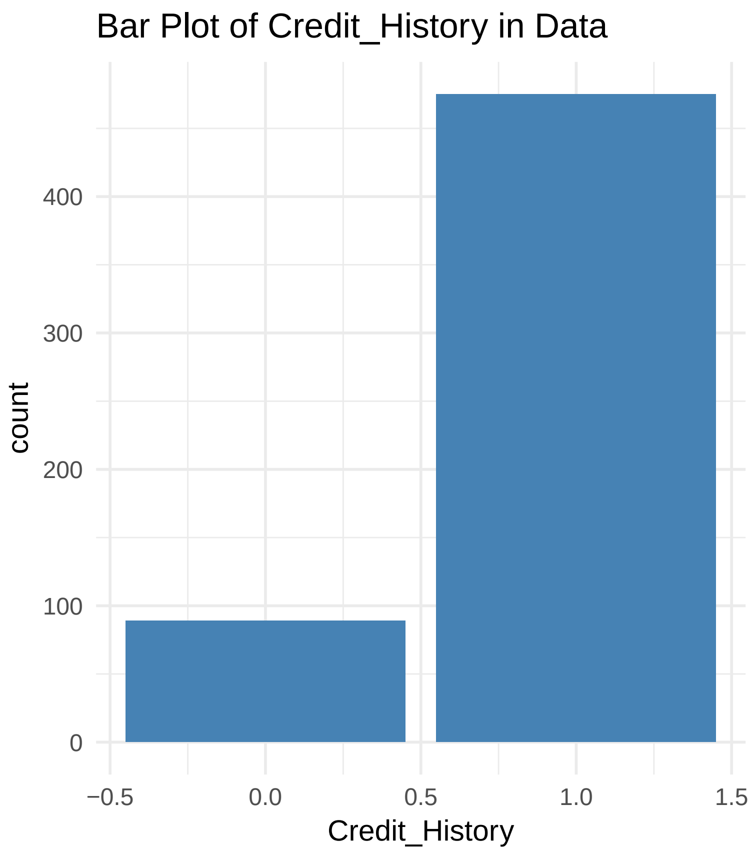
8. Loan_Amount_Term (Numeric)

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



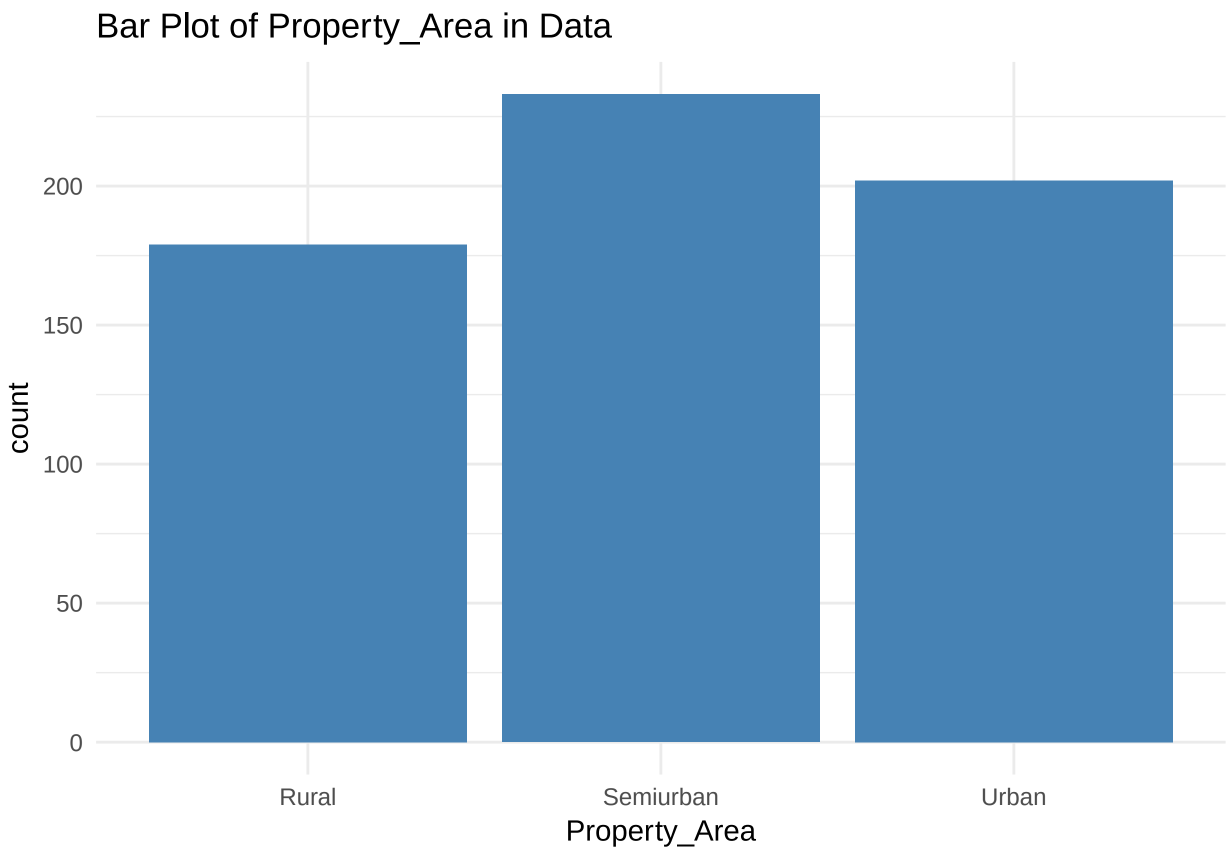
- Take Away points:
- Majority have 360 months as loan amount term
- We might have a few typo as 350 monts and 6 months tuple are present

9. Credit_History (Factor)



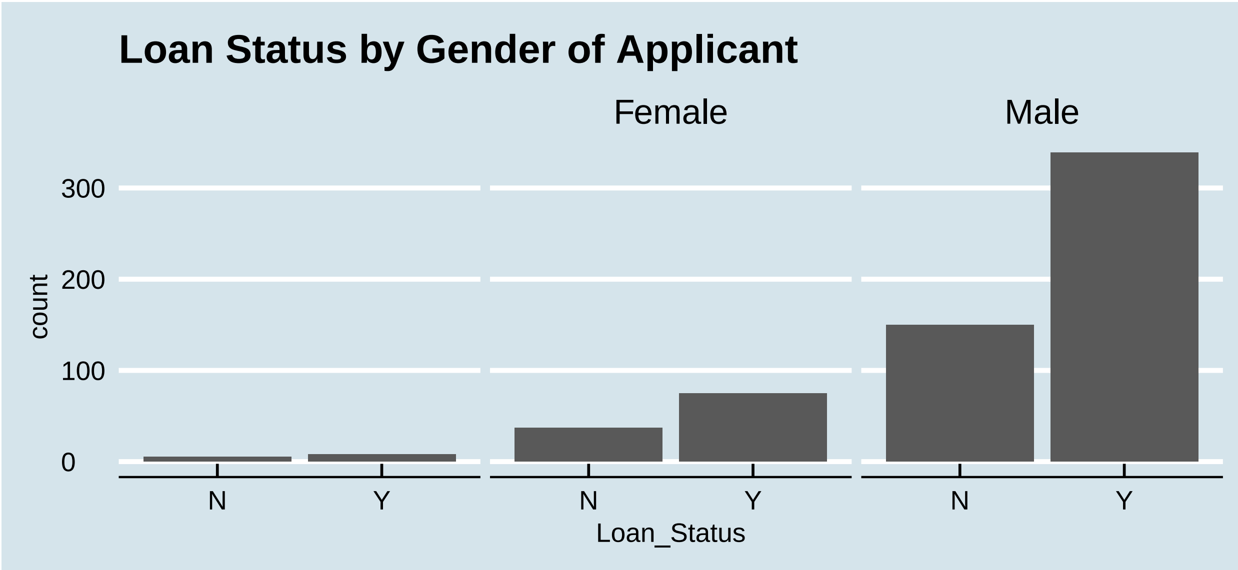
- Take Away points:-
- It should be a factor variable like yes or no etc.

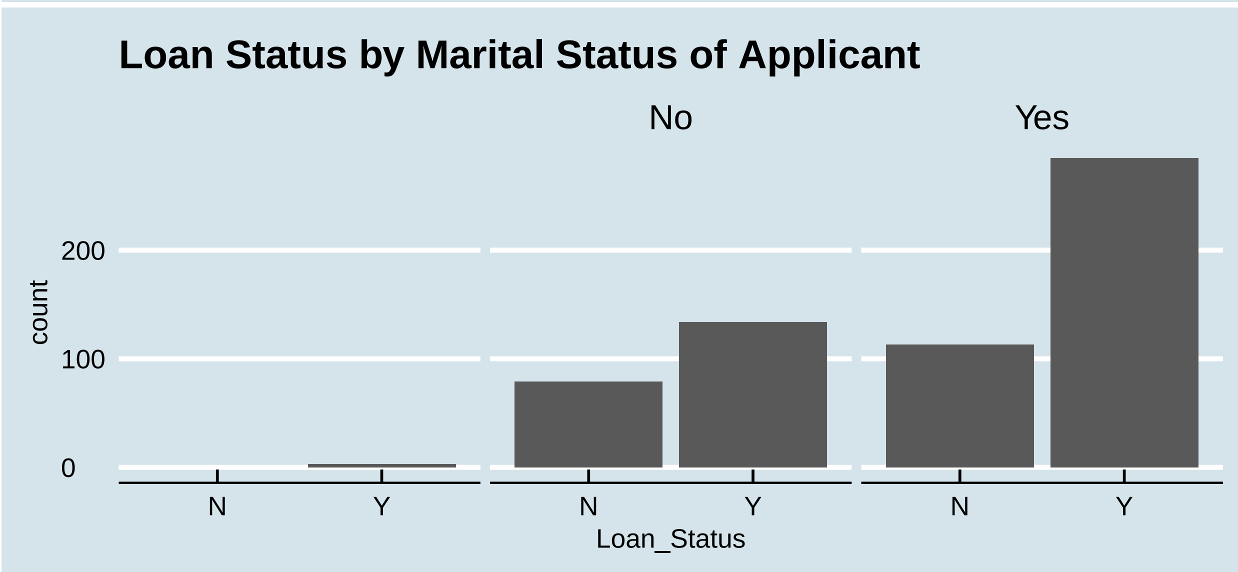
10. Property_Area

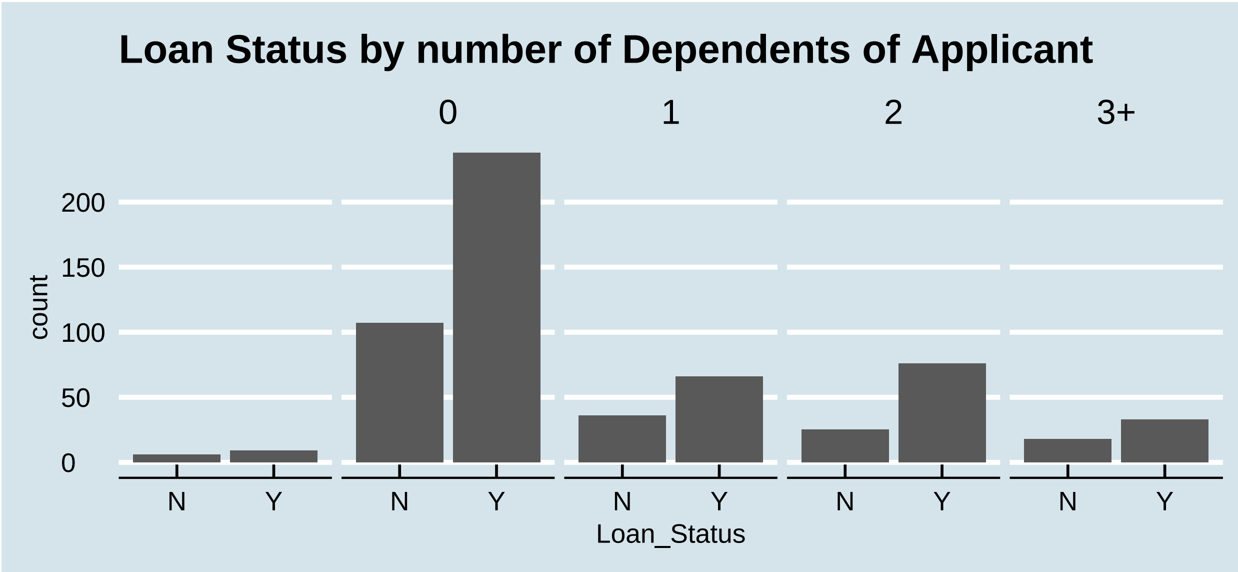


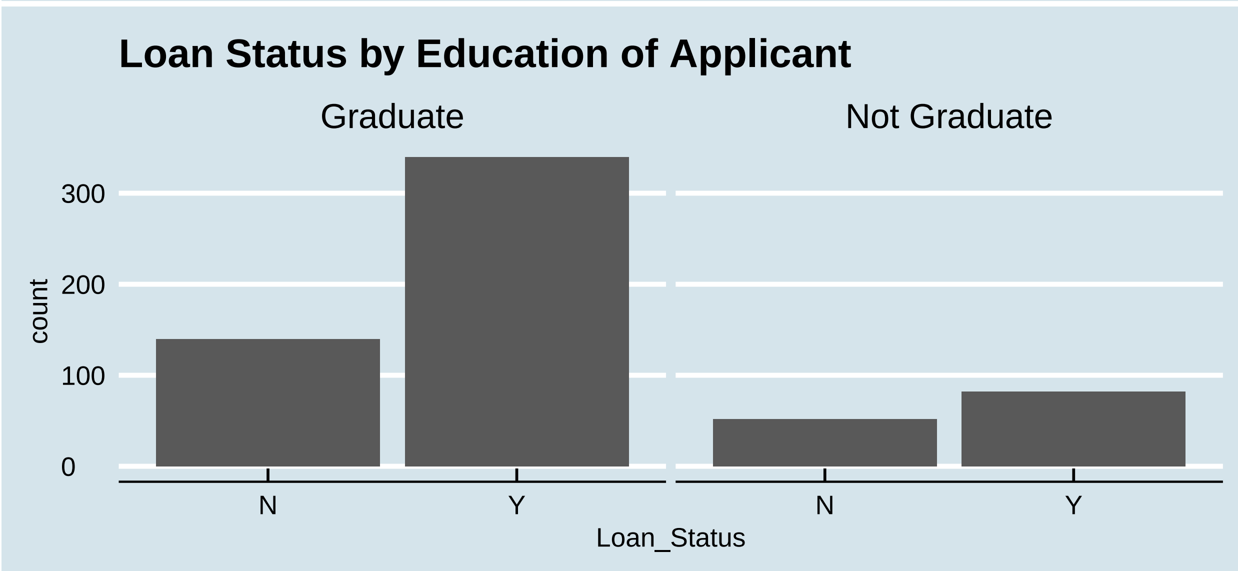
- Take Away points:
- Majority of property holdings are in semiurban area

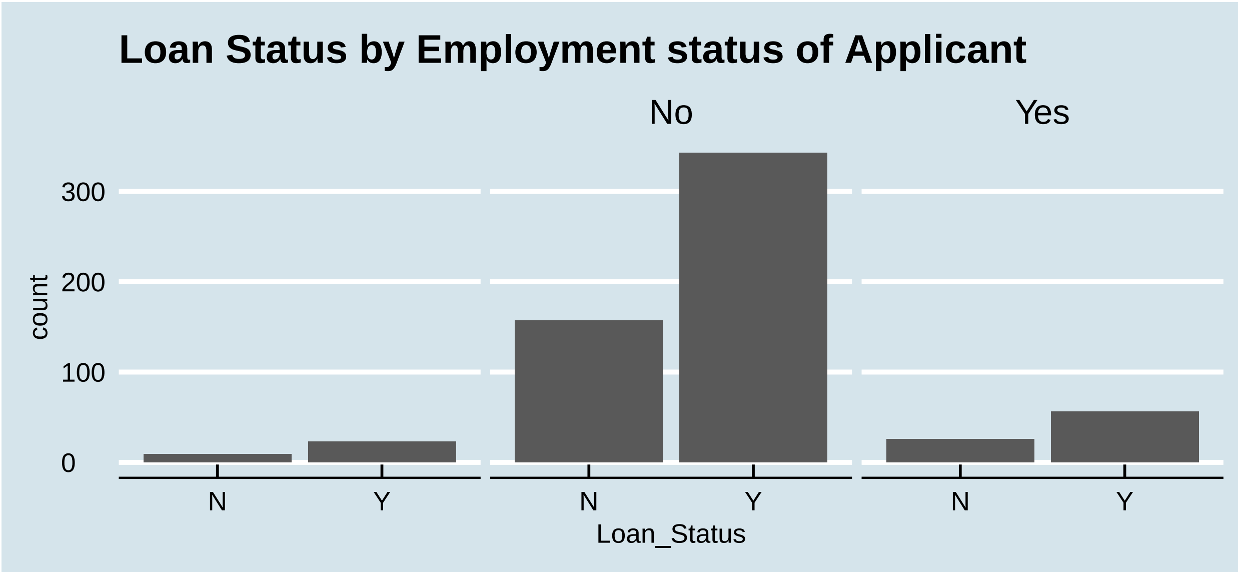
Various Multiple plots explaining relationship between different variables

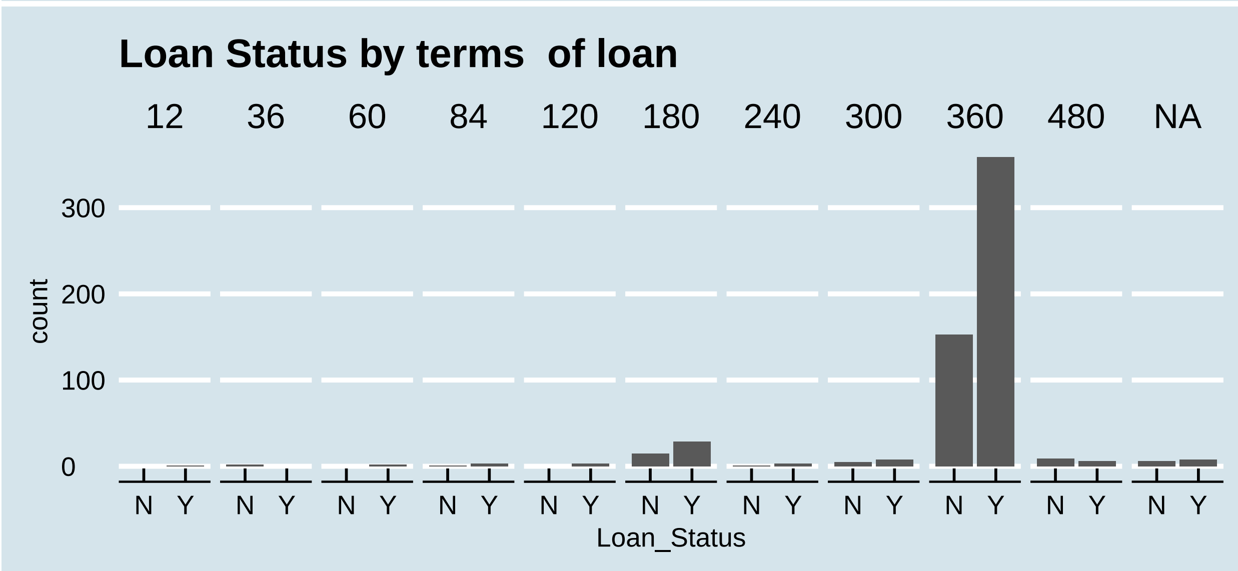


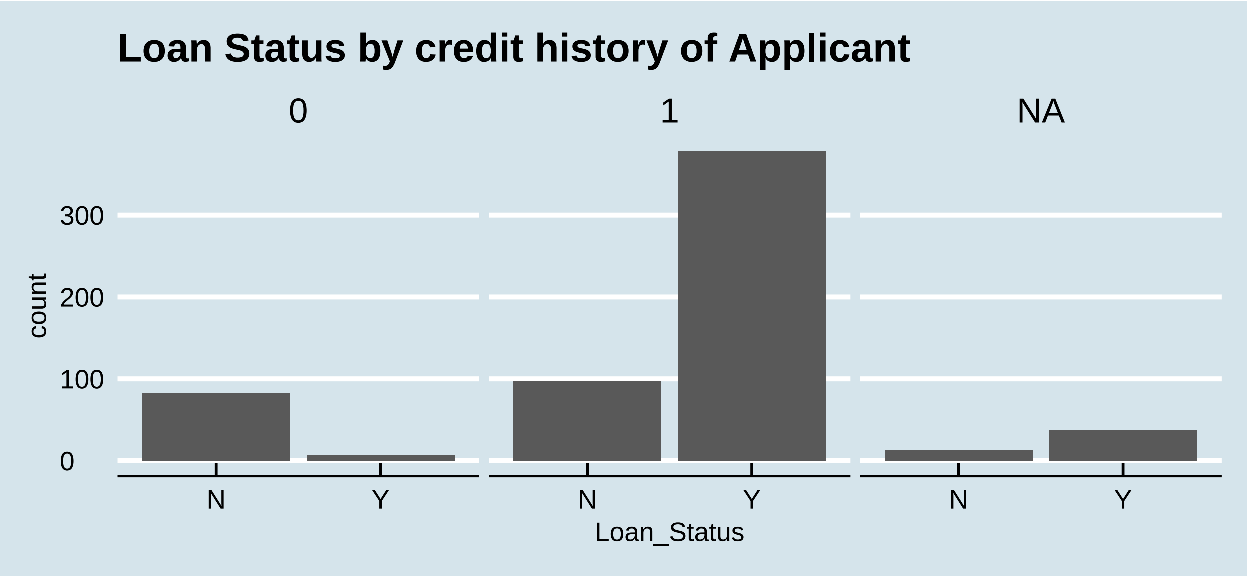


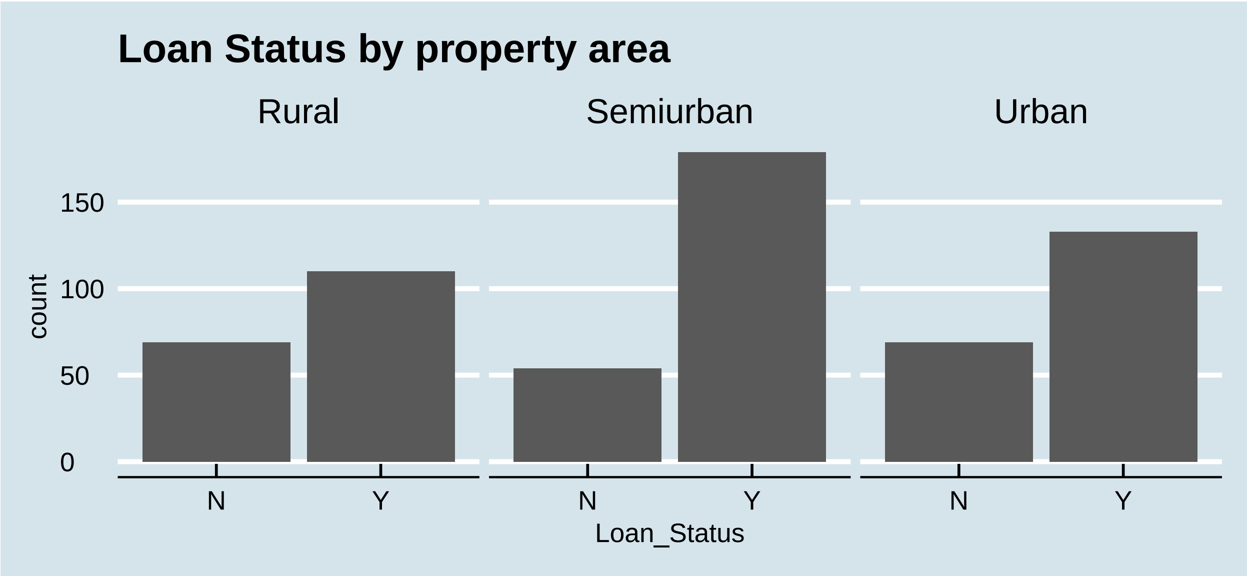

Loan Status by number of Dependents of Applicant
0
1
2
3+
count
200
150
100
50
0
N
Y
Loan_Status

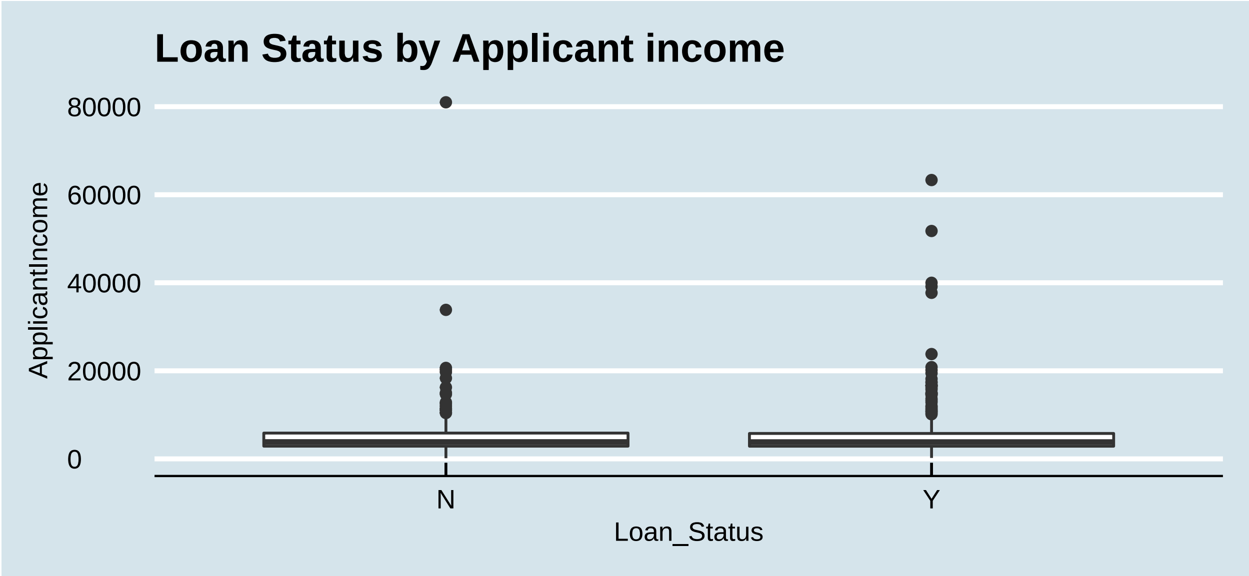

Loan Status by Education of Applicant
Graduate
Not Graduate
count
300
200
100
0
N
Y
Loan_Status

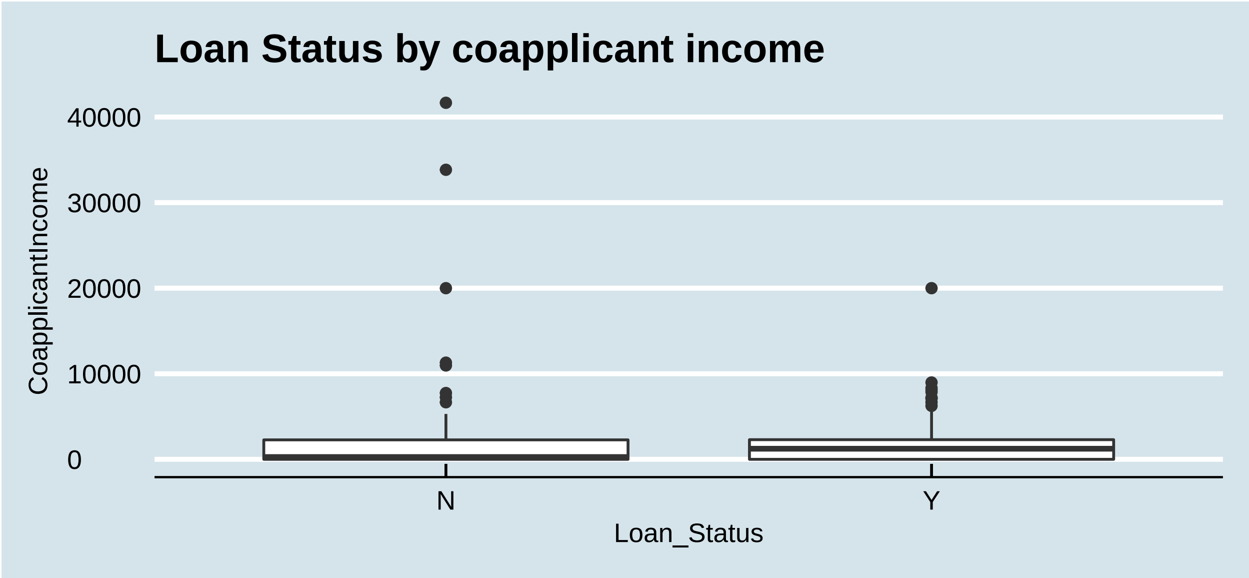

Loan Status by Employment status of Applicant
No
Yes
300
200
100
0
count
N
Y
N
Y
N
Y
Loan_Status

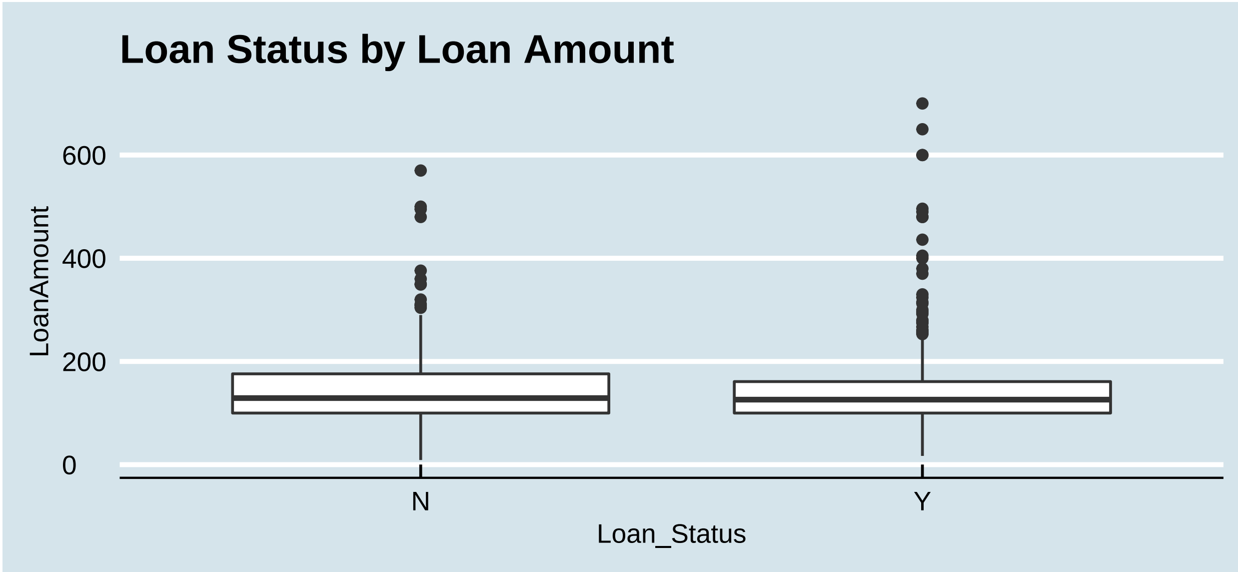

Loan Status by terms of loan
12
36
60
84
120
180
240
300
360
480
NA
300
200
100
0
count
N Y N Y N Y N Y N Y N Y N Y N Y N Y N Y N Y
Loan_Status


Loan Status by credit history of Applicant
0
1
NA
300
200
100
0
count
N
Y
Loan_Status


Loan Status by property area
Rural
Semiurban
Urban
150
100
50
0
count
N
Y
Loan_Status


Loan Status by Applicant income
80000
60000
40000
20000
0
ApplicantIncome
N
Y
Loan_Status


Loan Status by coapplicant income
40000
30000
20000
10000
0
CoapplicantIncome
N
Y
Loan_Status

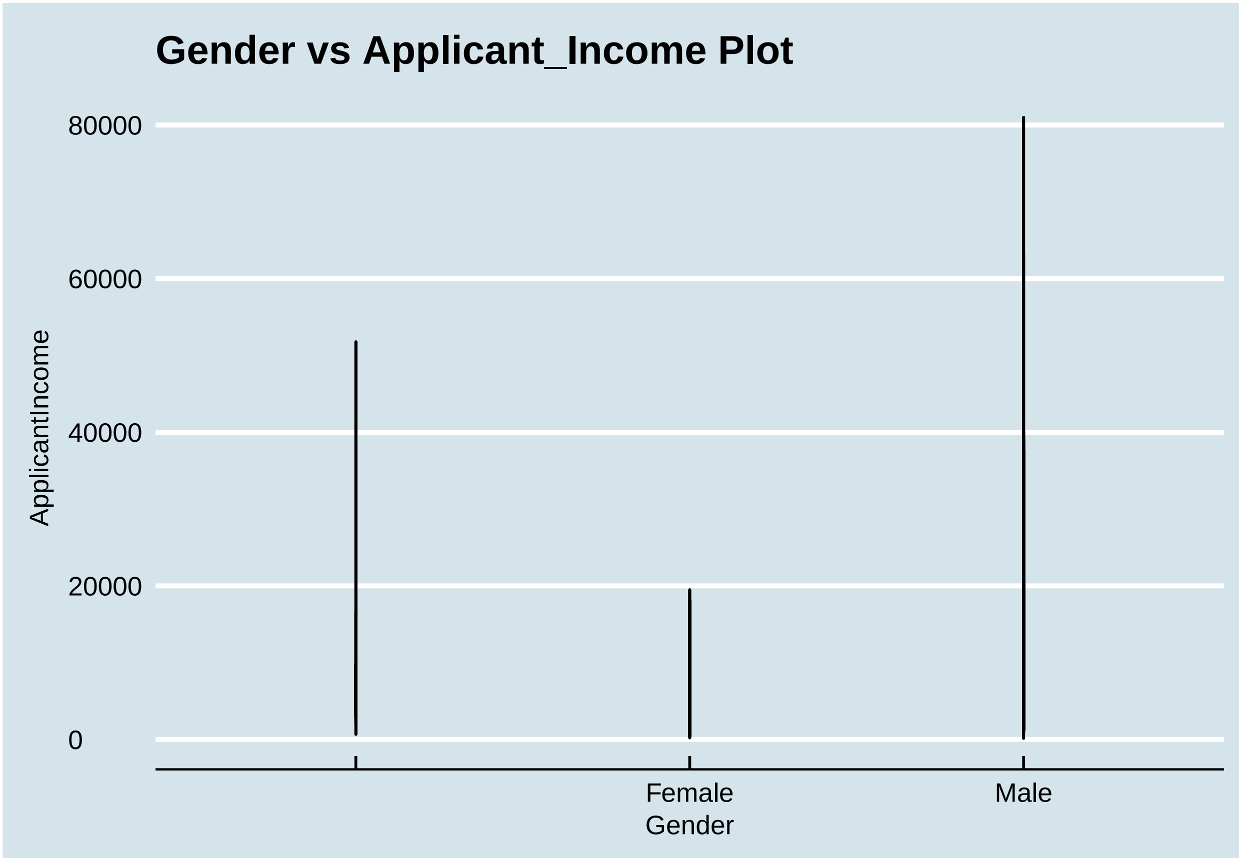


** Filling NA values**
####When there is "No" co-applicant income assuming as Unmarried and Married otherwise

```
#converting factor varibales to charcter later will convert back
data$Gender <- as.character(data$Gender)
data$Married <- as.character(data$Married)
data$Self_Employed <- as.character(data$Self_Employed)
data$Married[data$Married=="" & data$CoapplicantIncome==0]<-"No"
data$Married[data$Married==""]<- "Yes"
```

Plot shows that if gender is male its income is more than female so

```
print(ggplot(data, aes(x=Gender, y=ApplicantIncome))+geom_line()+ggtitle("Gender vs Applicant_In
```




```
#Plot shows that if gender is male its income is more than female so
data$Dependents <- as.character(data$Dependents)
data$Gender[data$Gender=="" & data$Dependents==""] <- "Male"
```

When Dependents is unknown but not married then assuming no dependents

```
data$Dependents[data$Dependents=="" & data$Married=="No"]<- "0"
```

Most of the loan term is 360, so filling NA as 360 in loan amount and renaming 350 as 360 and 6 as 60 since their frequency is less and might be due to typing error while entering data

```
data$Loan_Amount_Term[is.na(data$Loan_Amount_Term)]<-"360"
library(car)
data$Loan_Amount_Term <- recode(data$Loan_Amount_Term,"'350'='360';'6'='60'")
```

Assuming “” empty factor in self employed. As most are not self employed

```
data$Self_Employed[data$Self_Employed==""] <- "No"
```

Assuming person with no credit history as another catrgory

```r
data$Credit_History<-recode(data$Credit_History,"NA=2")
#converting all character to factor back
data$Credit_History <- as.factor(data$Credit_History)
data$Gender <- as.factor(data$Gender)
data$Married <- as.factor(data$Married)
data$Dependents <- as.factor(data$Dependents)
data$Self_Employed <- as.factor(data$Self_Employed)
data$Loan_Amount_Term <- as.factor(data$Loan_Amount_Term)
```

To predict Remaining Gender by (Mode Imputation) & Dependents

```r
levels(data$Gender)[levels(data$Gender)==""] <- "Male"
levels(data$Dependents)[levels(data$Dependents)==""] <- "0"
```

We will predict Loan Amount using K-Nearrest neighbours

```r
preProcValues <- preProcess(data, method = c("knnImpute","center","scale"))
Complete_Data_processed <- predict(preProcValues, data)
```

Checking for High Coorelation among Numeric variables

```r
cor(Complete_Data_processed[,c(7,8,9)])
```

```
##                   ApplicantIncome CoapplicantIncome LoanAmount
## ApplicantIncome         1.0000000        -0.1166046  0.5788562
## CoapplicantIncome      -0.1166046         1.0000000  0.1835371
## LoanAmount              0.5788562         0.1835371  1.0000000
```

No strong coorelation, So moving ahead.

Splitting Training and Test Data set

```r
#Spliting training set into two parts based on outcome: 70% and 30%
set.seed(1)
index <- createDataPartition(Complete_Data_processed$Loan_Status, p=0.70, list=FALSE)
trainSet <- Complete_Data_processed[ index,-1]
testSet <- Complete_Data_processed[-index,-1]
```

Creating Train control (3-fold cross validation)

```r
# Create model with default paramters
fitControl <- trainControl(method = "cv", number = 3, savePredictions = 'final', classProbs = T)
```

Creating Random Forest model#79.23%

```r
set.seed(2)
#Training the random forest model
model_rf<-train(Loan_Status~.,data=trainSet,method='rf',trControl=fitControl,tuneLength=5)
#Predicting using random forest model
testSet$pred_rf<-predict(object = model_rf,testSet[,-13])
#Checking the accuracy of the random forest model
confusionMatrix(testSet$Loan_Status,testSet$pred_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N  32  25
##          Y  13 113
##
##                Accuracy : 0.7923
##                  95% CI : (0.7263, 0.8487)
##     No Information Rate : 0.7541
##     P-Value [Acc > NIR] : 0.13135
##
##                   Kappa : 0.4863
##  Mcnemar's Test P-Value : 0.07435
##
##             Sensitivity : 0.7111
##             Specificity : 0.8188
##          Pos Pred Value : 0.5614
##          Neg Pred Value : 0.8968
##              Prevalence : 0.2459
##          Detection Rate : 0.1749
##    Detection Prevalence : 0.3115
##       Balanced Accuracy : 0.7650
##
##        'Positive' Class : N
##
```

Train KNN #74.86% Accuracy

```r
set.seed(3)
#Training the knn model
model_knn<-train(Loan_Status~.,data=trainSet,method='knn',trControl=fitControl,tuneLength=3)
#Predicting using knn model
testSet$pred_knn<-predict(object = model_knn,testSet[,-13])
#Checking the accuracy of the knn model
confusionMatrix(testSet$Loan_Status,testSet$pred_knn)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   N   Y
##          N  25  32
##          Y  14 112
##
##                Accuracy : 0.7486
##                  95% CI : (0.6793, 0.8097)
##     No Information Rate : 0.7869
##     P-Value [Acc > NIR] : 0.91008
##
##                   Kappa : 0.3585
##  Mcnemar's Test P-Value : 0.01219
##
##             Sensitivity : 0.6410
##             Specificity : 0.7778
##          Pos Pred Value : 0.4386
##          Neg Pred Value : 0.8889
##              Prevalence : 0.2131
##          Detection Rate : 0.1366
##    Detection Prevalence : 0.3115
##       Balanced Accuracy : 0.7094
##
##        'Positive' Class : N
##
```

Train Logistic Regression # 86.34% Accuracy

```
set.seed(4)
#Training the Logistic regression model
model_lr<-train(Loan_Status~.,data=trainSet,method='glm',trControl=fitControl,tuneLength=3)
#Predicting using Logistic regression model
testSet$pred_lr<-predict(object = model_lr,testSet[,-13])
#Checking the accuracy of the Logistic regression model
confusionMatrix(testSet$Loan_Status,testSet$pred_lr)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N  35  22
##          Y   3 123
##
##                Accuracy : 0.8634
##                  95% CI : (0.805, 0.9096)
##     No Information Rate : 0.7923
##     P-Value [Acc > NIR] : 0.0089345
##
##                   Kappa : 0.6495
```

```
##  Mcnemar's Test P-Value : 0.0003182
##
##             Sensitivity : 0.9211
##             Specificity : 0.8483
##          Pos Pred Value : 0.6140
##          Neg Pred Value : 0.9762
##              Prevalence : 0.2077
##          Detection Rate : 0.1913
##    Detection Prevalence : 0.3115
##       Balanced Accuracy : 0.8847
##
##        'Positive' Class : N
##
```

Train SVM using Linear Kernel #86.34% Accuracy

```r
set.seed(5)
#Training the SVM using Linear Kernel
model_svm<-train(Loan_Status~.,data=trainSet,method='svmLinear',trControl=fitControl,tuneLength=
#Predicting using SVM using Linear Kernel
testSet$pred_svm<-predict(object = model_svm,testSet[,-13])
#Checking the accuracy of the svm model
confusionMatrix(testSet$Loan_Status,testSet$pred_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N  34  23
##          Y   2 124
##
##                Accuracy : 0.8634
##                  95% CI : (0.805, 0.9096)
##     No Information Rate : 0.8033
##     P-Value [Acc > NIR] : 0.02193
##
##                   Kappa : 0.6458
##  Mcnemar's Test P-Value : 6.334e-05
##
##             Sensitivity : 0.9444
##             Specificity : 0.8435
##          Pos Pred Value : 0.5965
##          Neg Pred Value : 0.9841
##              Prevalence : 0.1967
##          Detection Rate : 0.1858
##    Detection Prevalence : 0.3115
##       Balanced Accuracy : 0.8940
##
```

```
##        'Positive' Class : N
##
```

Train a neural network #86.34% Accuracy

```
## # weights:  26
## initial  value 241.607779
## iter  10 value 149.517273
## iter  20 value 127.710120
## iter  30 value 126.588846
## iter  40 value 126.029697
## iter  50 value 121.700681
## iter  60 value 120.221279
## iter  70 value 119.807079
## iter  80 value 119.056870
## iter  90 value 117.901239
## iter 100 value 115.177102
## final  value 115.177102
## stopped after 100 iterations
## # weights:  76
## initial  value 343.758046
## iter  10 value 128.724449
## iter  20 value 111.441716
## iter  30 value 97.195704
## iter  40 value 86.324962
## iter  50 value 80.075296
## iter  60 value 77.980833
## iter  70 value 77.158751
## iter  80 value 77.024450
## iter  90 value 76.785771
## iter 100 value 76.576172
## final  value 76.576172
## stopped after 100 iterations
## # weights:  126
## initial  value 211.943423
## iter  10 value 126.248348
## iter  20 value 99.557038
## iter  30 value 92.188964
## iter  40 value 77.244078
## iter  50 value 59.278158
## iter  60 value 49.101449
## iter  70 value 46.299893
## iter  80 value 44.463508
## iter  90 value 42.766697
## iter 100 value 42.449665
## final  value 42.449665
## stopped after 100 iterations
```

```
## # weights:  26
## initial  value 233.905555
## iter  10 value 143.894661
## iter  20 value 135.422000
## iter  30 value 135.115441
## iter  40 value 135.112543
## final  value 135.112539
## converged
## # weights:  76
## initial  value 204.988926
## iter  10 value 135.383418
## iter  20 value 125.723067
## iter  30 value 115.806228
## iter  40 value 114.462903
## iter  50 value 113.688992
## iter  60 value 113.482458
## iter  70 value 113.475873
## final  value 113.475871
## converged
## # weights:  126
## initial  value 180.901531
## iter  10 value 145.221877
## iter  20 value 123.149934
## iter  30 value 113.897814
## iter  40 value 111.337327
## iter  50 value 107.419057
## iter  60 value 105.857682
## iter  70 value 105.262190
## iter  80 value 104.987466
## iter  90 value 104.859437
## iter 100 value 104.838679
## final  value 104.838679
## stopped after 100 iterations
## # weights:  26
## initial  value 183.442361
## iter  10 value 132.661696
## iter  20 value 124.224864
## iter  30 value 121.159304
## iter  40 value 120.528077
## iter  50 value 119.932671
## iter  60 value 119.864119
## iter  70 value 119.842750
## iter  80 value 119.825287
## iter  90 value 119.756092
## iter 100 value 119.742486
## final  value 119.742486
## stopped after 100 iterations
## # weights:  76
```

```
## initial  value 222.842087
## iter  10 value 146.969689
## iter  20 value 119.675355
## iter  30 value 112.441079
## iter  40 value 107.181980
## iter  50 value 105.330389
## iter  60 value 104.538924
## iter  70 value 103.386829
## iter  80 value 101.597302
## iter  90 value 101.412839
## iter 100 value 101.310385
## final  value 101.310385
## stopped after 100 iterations
## # weights:  126
## initial  value 194.576393
## iter  10 value 128.427226
## iter  20 value 104.137054
## iter  30 value 84.448245
## iter  40 value 73.541010
## iter  50 value 70.971411
## iter  60 value 69.944384
## iter  70 value 69.557328
## iter  80 value 69.393395
## iter  90 value 69.186708
## iter 100 value 68.725392
## final  value 68.725392
## stopped after 100 iterations
## # weights:  26
## initial  value 203.522942
## iter  10 value 154.390537
## iter  20 value 140.217436
## iter  30 value 135.274336
## iter  40 value 134.102692
## iter  50 value 129.564085
## iter  60 value 124.150980
## iter  70 value 123.895857
## iter  80 value 122.948446
## iter  90 value 122.521300
## iter 100 value 122.467005
## final  value 122.467005
## stopped after 100 iterations
## # weights:  76
## initial  value 232.783710
## iter  10 value 137.773048
## iter  20 value 112.139806
## iter  30 value 101.374050
## iter  40 value 89.777531
## iter  50 value 84.250346
```

```
## iter  60 value 81.750131
## iter  70 value 79.704078
## iter  80 value 77.730368
## iter  90 value 77.296005
## iter 100 value 77.175227
## final  value 77.175227
## stopped after 100 iterations
## # weights:  126
## initial  value 239.585355
## iter  10 value 135.753538
## iter  20 value 111.058529
## iter  30 value 86.211313
## iter  40 value 75.552273
## iter  50 value 71.215526
## iter  60 value 69.484372
## iter  70 value 69.404452
## iter  80 value 69.403473
## final  value 69.403471
## converged
## # weights:  26
## initial  value 196.502965
## iter  10 value 155.143213
## iter  20 value 144.597206
## iter  30 value 142.919593
## iter  40 value 142.576770
## final  value 142.525381
## converged
## # weights:  76
## initial  value 221.467368
## iter  10 value 141.428066
## iter  20 value 129.171970
## iter  30 value 125.015292
## iter  40 value 124.155055
## iter  50 value 122.800475
## iter  60 value 122.702830
## iter  70 value 122.694350
## final  value 122.694258
## converged
## # weights:  126
## initial  value 205.595689
## iter  10 value 144.118527
## iter  20 value 127.031281
## iter  30 value 121.861286
## iter  40 value 118.573841
## iter  50 value 115.349470
## iter  60 value 113.618866
## iter  70 value 112.797354
## iter  80 value 112.445836
```

```
## iter  90 value 111.863146
## iter 100 value 111.542537
## final  value 111.542537
## stopped after 100 iterations
## # weights:  26
## initial  value 189.973132
## iter  10 value 149.636832
## iter  20 value 141.469513
## iter  30 value 135.802793
## iter  40 value 135.322248
## iter  50 value 135.079995
## iter  60 value 134.949564
## iter  70 value 134.888891
## iter  80 value 134.708657
## iter  90 value 134.654222
## iter 100 value 134.644278
## final  value 134.644278
## stopped after 100 iterations
## # weights:  76
## initial  value 237.432650
## iter  10 value 145.153010
## iter  20 value 117.129246
## iter  30 value 91.877121
## iter  40 value 78.628049
## iter  50 value 74.560477
## iter  60 value 73.845960
## iter  70 value 73.614944
## iter  80 value 73.404854
## iter  90 value 73.201881
## iter 100 value 73.050780
## final  value 73.050780
## stopped after 100 iterations
## # weights:  126
## initial  value 197.375289
## iter  10 value 124.509208
## iter  20 value 85.671448
## iter  30 value 62.659671
## iter  40 value 57.039918
## iter  50 value 56.346162
## iter  60 value 56.030728
## iter  70 value 55.799581
## iter  80 value 55.325322
## iter  90 value 54.308802
## iter 100 value 51.046473
## final  value 51.046473
## stopped after 100 iterations
## # weights:  26
## initial  value 178.344510
```

```
## iter  10 value 137.279412
## iter  20 value 130.223773
## iter  30 value 127.396300
## iter  40 value 122.521616
## iter  50 value 122.449512
## final  value 122.449389
## converged
## # weights:  76
## initial  value 224.478056
## iter  10 value 140.774266
## iter  20 value 117.287977
## iter  30 value 95.686248
## iter  40 value 87.605428
## iter  50 value 84.504295
## iter  60 value 84.070471
## iter  70 value 83.782648
## iter  80 value 83.450874
## iter  90 value 83.434236
## iter 100 value 83.426779
## final  value 83.426779
## stopped after 100 iterations
## # weights:  126
## initial  value 260.576974
## iter  10 value 134.237441
## iter  20 value 95.848230
## iter  30 value 60.038842
## iter  40 value 50.877094
## iter  50 value 47.169872
## iter  60 value 44.356049
## iter  70 value 43.579297
## iter  80 value 43.566980
## iter  90 value 43.563111
## iter 100 value 43.553607
## final  value 43.553607
## stopped after 100 iterations
## # weights:  26
## initial  value 184.095827
## iter  10 value 156.785914
## iter  20 value 137.805121
## iter  30 value 134.119143
## iter  40 value 133.961889
## iter  50 value 133.938631
## final  value 133.938622
## converged
## # weights:  76
## initial  value 194.121829
## iter  10 value 140.740336
## iter  20 value 123.060727
```

```
## iter  30 value 121.052288
## iter  40 value 118.396174
## iter  50 value 117.147254
## iter  60 value 116.488361
## iter  70 value 116.336405
## iter  80 value 116.330738
## final  value 116.330632
## converged
## # weights:  126
## initial  value 189.624543
## iter  10 value 135.752669
## iter  20 value 118.906312
## iter  30 value 113.234166
## iter  40 value 108.815703
## iter  50 value 105.111727
## iter  60 value 103.885646
## iter  70 value 103.536037
## iter  80 value 103.343375
## iter  90 value 103.309495
## iter 100 value 103.299845
## final  value 103.299845
## stopped after 100 iterations
## # weights:  26
## initial  value 234.340156
## iter  10 value 155.479956
## iter  20 value 133.470635
## iter  30 value 129.241780
## iter  40 value 127.803529
## iter  50 value 126.480931
## iter  60 value 125.742875
## iter  70 value 125.333249
## iter  80 value 124.988729
## iter  90 value 124.915467
## iter 100 value 124.875373
## final  value 124.875373
## stopped after 100 iterations
## # weights:  76
## initial  value 200.347281
## iter  10 value 130.476518
## iter  20 value 100.219838
## iter  30 value 87.053718
## iter  40 value 83.142937
## iter  50 value 80.141979
## iter  60 value 77.959515
## iter  70 value 75.942986
## iter  80 value 74.159585
## iter  90 value 73.184735
## iter 100 value 73.022364
```

```
## final  value 73.022364
## stopped after 100 iterations
## # weights:  126
## initial  value 192.489465
## iter  10 value 122.801766
## iter  20 value 85.346140
## iter  30 value 66.415372
## iter  40 value 61.192629
## iter  50 value 60.649193
## iter  60 value 59.574297
## iter  70 value 59.253946
## iter  80 value 59.100452
## iter  90 value 58.824535
## iter 100 value 58.271534
## final  value 58.271534
## stopped after 100 iterations
## # weights:  26
## initial  value 268.483065
## iter  10 value 227.556056
## iter  20 value 217.117459
## iter  30 value 209.553623
## iter  40 value 208.061588
## iter  50 value 207.859873
## final  value 207.857193
## converged
```

Train a neural network #86.34% Accuracy

```r
# Checking the accuracy of the Neural network model
confusionMatrix(testSet$Loan_Status, testSet$pred_nn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N  35  22
##          Y   3 123
##
##                Accuracy : 0.8634
##                  95% CI : (0.805, 0.9096)
##     No Information Rate : 0.7923
##     P-Value [Acc > NIR] : 0.0089345
##
##                   Kappa : 0.6495
##  Mcnemar's Test P-Value : 0.0003182
##
##             Sensitivity : 0.9211
##             Specificity : 0.8483
```

```
##          Pos Pred Value : 0.6140
##          Neg Pred Value : 0.9762
##              Prevalence : 0.2077
##          Detection Rate : 0.1913
##    Detection Prevalence : 0.3115
##       Balanced Accuracy : 0.8847
##
##        'Positive' Class : N
##
```

```
# tune nueral neturok #86.34% Accuracy
my.grid <- expand.grid(.decay = c(0.5, 0.1), .size = c(1:3))
model_nn_tune <- train(Loan_Status ~ ., data = trainSet, method = "nnet", trControl = fitControl
    tuneLength = 3, tuneGrid = my.grid, trace = F)
testSet$pred_nn_tune <- predict(model_nn_tune, testSet[, -13])
# Checking the accuracy of the tuned Neural network model
confusionMatrix(testSet$Loan_Status, testSet$pred_nn_tune)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N  35  22
##          Y   3 123
##
##                Accuracy : 0.8634
##                  95% CI : (0.805, 0.9096)
##     No Information Rate : 0.7923
##     P-Value [Acc > NIR] : 0.0089345
##
##                   Kappa : 0.6495
##  Mcnemar's Test P-Value : 0.0003182
##
##             Sensitivity : 0.9211
##             Specificity : 0.8483
##          Pos Pred Value : 0.6140
##          Neg Pred Value : 0.9762
##              Prevalence : 0.2077
##          Detection Rate : 0.1913
##    Detection Prevalence : 0.3115
##       Balanced Accuracy : 0.8847
##
##        'Positive' Class : N
##
```

Conclusion

Will use Logistic Regression for this dataset as the accuracy is same for Logistic Regression and SVM and Neural network, as time required to train a logistic regression model is less than svm or neural network model.