PREDICTIVE MODELING FOR LOAN DEFAULT

PROJECT REPORT

Sudhanshu Kulkarni

Table of Contents

Introduction

The Irish Dummy Bank is a peer to peer lending bank based in Ireland, in which the bank provides funds for potential borrowers and the bank earns a profit depending on the risk they take (the borrowers credit score). The complete data set is borrowed from the Lending Club which is a US peer to peer lending bank company that enables borrowers to create unsecured personal loans between $1,000 and $40,000. The standard loan period is three years.

Objective and Business Questions

There are many aspects that can affect how capable a borrower can pay off a loan within a given term. It is important to provide loans to those who have a likelihood of repaying the debt, rather to anyone that requests for one. So, understanding the borrower's reason for the loan, financial status, and income play a role in predicting whether that borrower will pay off the loan without causing any major issues. For this reason, we want to identify the main features that contribute to a good loan. A good loan is defined as a loan where the repayments are made on time and as agreed upon between the borrower and the lender. Our goal is to differentiate if the borrower will end up as a good loan or a bad one. Our solution to figuring out if a future borrower will repay a loan on time is by identifying the different predictors that result in a good loan. When we can understand the features that increase the chances for a good loan, we can help banks concentrate on those specific areas to look at while deciding to give a loan to a borrower.

Dataset Description

The Kaggle dataset we are working on is a copied and cleaned version from a complete dataset from the Lending Club. The dimensions consist of 887,000 rows and 30 columns with no missing values. Hypothetically, if there were any missing values, there are a few options in dealing with them. We could either replace those values with the median or another specific value, or remove the respective objectives all together, or ignore them based on the model. Without any changes to the data, the following values are initially numeric: Emp_length_int, annual _income, Loan_amount, Interest_rate, Dti, Total_pymnt, Installments, all the category columns (in numbers), and the final_d. While the nominal values are all the only issue_d, the category names associated with the numeric category columns, and the region column. All the variables are dependent as well, except the target variable which is the loan condition. The major variables we will use are:

Variable	**Description**	**Category Description**
emp_length_int	borrower's length of employment	

home_ownership_cat	homeownership of borrower category	1 = RENT 2 = OWN 3 = MORTGAGE 4, 5, 6 = OTHER, NONE, ANY (combine and rename UNKNOWN)
income_category	borrower's income	1 = low 2 = medium 3 = high
annual_income	annual income value of borrower	
loan_amount	amount of loan	
term_cat	how long borrower can pay off loan	1 = 36 months 2 = 60 months
application_type_cat	type of loan application	1 = INDIVIDUAL 2 = JOINT
purpose_cat	purpose of the loan	1 = credit_card 2 = car 3 = small_business 4 = other 5 = wedding 6 = debt_consolidation 7 = home_improvement 8 = major_purchase 9 = medical 10 = moving 11 = vacation 12 = house 13 = renewable_energy 14 = educational
interest_payment_cat	interest borrower pays	1 = High 2 = Low
loan_condition_cat	condition of loan	0 = Good Loan 1 = Bad Loan
interest_rate	loan interest rate	

grade_cat	assigning a quality score to a loan based on a borrower's credit history, quality of collateral and likelihood of repayment of the principal and interest	1 = A 2 = B 3 = C 4 = D 5 = E 6 = F 7 = G
dti	ratio calculated using the borrower's total monthly debt payments on the total debt obligations, - - - excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income	
total_pymnt	total payment occurred	
installment	fixed amount paid in installments	
region	region of borrower	
loanPeriod	Loan period in years	

Data Pre-Processing and Preparation

We have decided to remove the following columns from the original excel sheet, since we felt it was not necessary for our analysis: id, year, total_rec_prncp, recoveries, and all the names values of the category columns (home_ownership_cat, income_category, term_cat, application_type_cat, purpose_cat, interest_payment_cat, loan_condition_cat, grade_cat). We used the category column numbers instead, because we felt it was easier to use for further analysis and usually the algorithms needed factors instead of strings. Instead of using the issue and final date column, we also decided to transform it into a single column as loanPeriod which shows how many years the loan took.

```
library(stringr)
str_pad(loanDetails$final_d, 8, pad = "0")

loanDetails$finalDate <- as.Date(str_pad(loanDetails$final_d, 8, pad = "0"),"%m%d%Y")
View(loanDetails)
loanDetails$issueDate <- as.Date(loanDetails$issue_d,"%m/%d/%Y")

loanDetails$loanPeriod <- loanDetails$finalDate - loanDetails$issueDate
```

After reading in the new cleaned data set, we wanted to check whether data is clean or not, whether the data has missing values or not, we use the following code:

```r
#read in csv
data <- read.csv('data.csv')

#no missing values
sum(is.na(data))
```

Output:

```
> #no missing values
> sum(is.na(data))
[1] 0
>
```

The result of the above function was very good as we found that the data is 100% clean and contains no missing values and hence does not require any process to handle missing values. Then we can see how many observations and variables we are left with. We can also see the data types as well, which we will change so that our analysis algorithms can work.

```
data                  887379 obs. of 17 variables
  emp_length_int : num 10 0.5 10 10 1 3 8 9 4 0.5 ...
  home_ownership_cat : int 1 1 1 1 1 1 1 1 2 1 ...
  annual_inc : int 24000 30000 12252 49200 80000 36000 47004 48000…
  income_cat : int 1 1 1 1 1 1 1 1 1 1 ...
  loan_amount : int 5000 2500 2400 10000 3000 5000 7000 3000 5600 …
  term_cat : int 1 2 1 1 2 1 2 1 2 2 ...
  application_type_cat: int 1 1 1 1 1 1 1 1 1 1 ...
  purpose_cat : int 1 2 3 4 4 5 6 2 3 4 ...
  interest_payment_cat: int 1 2 2 2 1 1 2 2 2 1 ...
  loan_condition_cat : int 0 1 0 0 0 0 0 0 1 1 ...
  interest_rate : num 10.7 15.3 16 13.5 12.7 ...
  grade_cat : int 2 3 3 3 2 1 3 5 6 2 ...
  dti : num 27.65 1 8.72 20 17.94 ...
  total_pymnt : num 5861 1009 3004 12226 3242 ...
  installment : num 162.9 59.8 84.3 339.3 67.8 ...
  region : Factor w/ 5 levels "cannught","leinster",..: 3 2 1 5 5 …
  loanPeriod : num 3.97 1.98 2.99 3.97 4.97 3.97 4.97 3.97 0.98 1 …
```

We first made sure that the categorical columns are turned into factors rather than integers.

```r
#make all category columns into factors
data$home_ownership_cat <- as.factor(data$home_ownership_cat)
data$income_cat <- as.factor(data$income_cat)
data$term_cat <- as.factor(data$term_cat)
data$application_type_cat <- as.factor(data$application_type_cat)
data$purpose_cat <- as.factor(data$purpose_cat)
data$interest_payment_cat <- as.factor(data$interest_payment_cat)
data$loan_condition_cat <- as.factor(data$loan_condition_cat)
data$grade_cat <- as.factor(data$grade_cat)
```

By taking a look at the summary of the dataset we can get a better understanding of the data composition. Through this, we figured out what values seem to be wrong like loanPeriod (being continuous values):

```
> summary(data)
 emp_length_int   home_ownership_cat   annual_inc      income_cat  loan_amount    term_cat   application_type_cat
 Min.   : 0.500   1:356117           Min.   :      0   1:729616   Min.   :  500   1:621125   1:886868
 1st Qu.: 3.000   2: 87470           1st Qu.:  45000   2:140977   1st Qu.: 8000   2:266254   2:   511
 Median : 6.050   3:443557           Median :  65000   3: 16786   Median :13000
 Mean   : 6.051   4:   182           Mean   :  75028              Mean   :14755
 3rd Qu.:10.000   5:    50           3rd Qu.:  90000              3rd Qu.:20000
 Max.   :10.000   6:     3           Max.   :9500000              Max.   :35000

  purpose_cat     interest_payment_cat loan_condition_cat interest_rate   grade_cat       dti           total_pymnt
 6      :524215   1:465316             0:819950           Min.   : 5.32   1:148202   Min.   :   0.00   Min.   :    0
 1      :206182   2:422063             1: 67429           1st Qu.: 9.99   2:254535   1st Qu.:  11.91   1st Qu.: 1915
 7      : 51829                                           Median :12.99   3:245860   Median :  17.65   Median : 4895
 4      : 42894                                           Mean   :13.25   4:139542   Mean   :  18.16   Mean   : 7559
 8      : 17277                                           3rd Qu.:16.20   5: 70705   3rd Qu.:  23.95   3rd Qu.:10617
 3      : 10377                                           Max.   :28.99   6: 23046   Max.   :9999.00   Max.   :57778
 (Other): 34605                                                           7:  5489
  installment              region         loanPeriod
 Min.   :  15.67   cannught    :155029   Min.   :0.000
 1st Qu.: 260.70   leinster    :214646   1st Qu.:0.980
 Median : 382.55   munster     :104574   Median :0.990
 Mean   : 436.72   Northern-Irl:204399   Mean   :1.297
 3rd Qu.: 572.60   ulster      :208731   3rd Qu.:1.980
 Max.   :1445.46                         Max.   :8.000
```

The home_ownership_cat column originally had 6 different factors. However, since 4, 5, and 6 values were “Other”, “None”, and “Any”, we agreed to combine these 3 categories into one single one and name it “unknown”. We cannot conclude what other home ownership, none, or any homeownership really means. This is why we decided to combine the 3.

```
levels(data$home_ownership_cat) <- c("1", "2", "3", "4", "4", "4")
```

By looking at a small portion of the loanPeriod column, the values are continuous which would not make sense when it comes to years. We cannot explain clearly how something can take 3.97 years.

```
> data$loanPeriod
  [1] 3.97 1.98 2.99 3.97 4.97 3.97 4.97 3.97 0.98 1.00 1.99 1.99
```

So, to fix this problem, we rounded the numbers to the nearest whole number. The below is the transformed data:

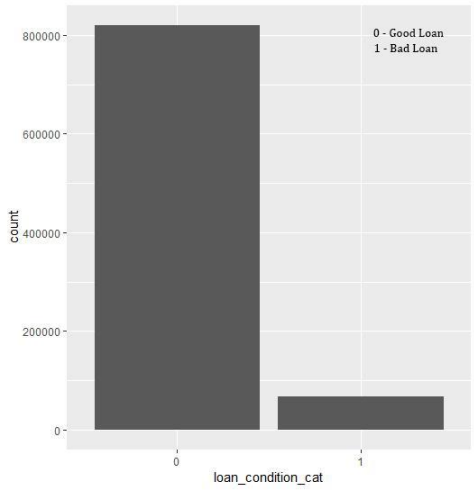
```
data$loanPeriod <- round(data$loanPeriod, digits = 0)
```

```
> data$loanPeriod
  [1] 4 2 3 4 5 4 5 4 1 1 2 2 1 4 2 4 4 2 4 1 4 2 4 2 1 1 5 1 2 3 4
```

Now our dataset is prepared and ready for our models.

```
data                887379 obs. of 17 variables
  emp_length_int : num 10 0.5 10 10 1 3 8 9 4 0.5 ...
  home_ownership_cat : Factor w/ 6 levels "1","2","3","4",..:…
  annual_inc : int 24000 30000 12252 49200 80000 36000 47004 …
  income_cat : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 …
  loan_amount : int 5000 2500 2400 10000 3000 5000 7000 3000 …
  term_cat : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 2 1 2 2 …
  application_type_cat: Factor w/ 2 levels "1","2": 1 1 1 1 1…
  purpose_cat : Factor w/ 14 levels "1","2","3","4",..: 1 2 3…
  interest_payment_cat: Factor w/ 2 levels "1","2": 1 2 2 2 1…
  loan_condition_cat : Factor w/ 2 levels "0","1": 1 2 1 1 1 …
  interest_rate : num 10.7 15.3 16 13.5 12.7 ...
  grade_cat : Factor w/ 7 levels "1","2","3","4",..: 2 3 3 3 …
  dti : num 27.65 1 8.72 20 17.94 ...
  total_pymnt : num 5861 1009 3004 12226 3242 ...
  installment : num 162.9 59.8 84.3 339.3 67.8 ...
  region : Factor w/ 5 levels "cannught","leinster",..: 3 2 1…
  loanPeriod : num 3.97 1.98 2.99 3.97 4.97 3.97 4.97 3.97 0.…
```

Next, we prepare data by creating a dataset which is not biased. Our current dataset was biased on the target variable (loan_condition_cat), as we can see in the figure below.



To eliminate this bias, we selected all the data which had loan_condition_cat = 1 (67,429 obs) and randomly sampled the same number of observations loan_condition_cat = 0 (67,429 obs).

```
#subsetting data as per the loan category into goodLoanData and badLoanData
goodLoanData <- subset(data, data$loan_condition_cat == 0)
badLoanData <- subset(data, data$loan_condition_cat == 1)
```

```
#to create train data set which contains 50% data of both good loan and bad loan we have
#sampled goodLoanData which has 67429 observation which is count of bad loan obervations
sampleGoodLoanData <- sample_n(goodLoanData,67429)

#binding 50% of good loan and bad loan data to create trainData
trainData <- rbind(sampleGoodLoanData,badLoanData)

# to create test data set I have selected 33% of the total train data to include 50% of good loan
#observations and 50% of bad loan observations
sampleGoodLoanTestData <- sample_n(goodLoanData,22251)
sampleBadLoanTestData <- sample_n(badLoanData,22251)
testData <- rbind(sampleGoodLoanTestData,sampleBadLoanTestData)
```

We then achieved a balanced dataset to run our algorithms upon.

Data Analysis

Association Rules Mining

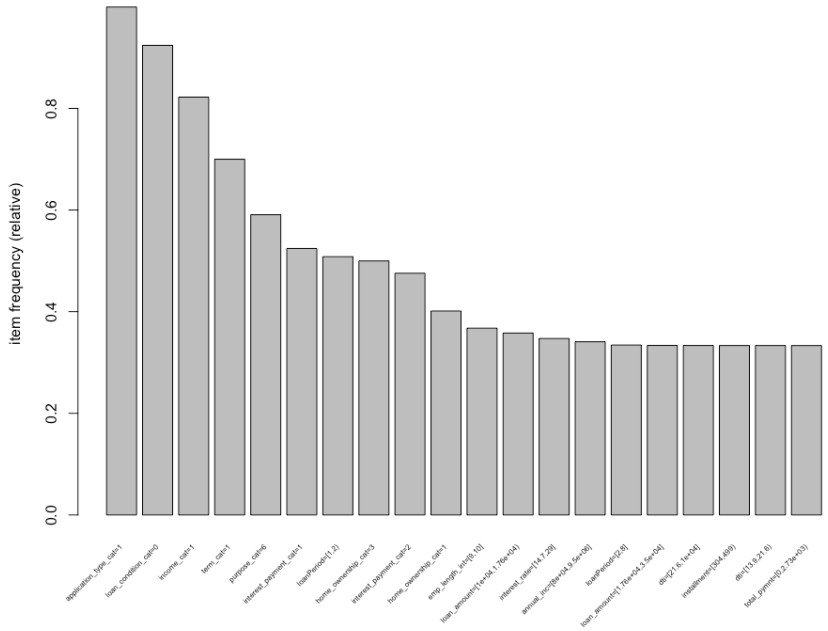
To perform the Association Rules Mining, our first step was to install the necessary packages. We also discretized the numeric columns because it is required to have the dataset change into transactions.

```
install.packages("arules")
install.packages("arulesViz")
library(arules)
library(arulesViz)
```

```
data_loan <- data
data_loan$emp_length_int <- discretize(data_loan$emp_length_int, methods = interval, default = NULL)
data_loan$annual_inc <- discretize(data_loan$annual_inc, methods = interval, default = NULL)
data_loan$loan_amount <- discretize(data_loan$loan_amount, methods = interval, default = NULL)
data_loan$interest_rate <- discretize(data_loan$interest_rate, methods = interval, default = NULL)
data_loan$dti <- discretize(data_loan$dti, methods = NULL, default = NULL)
data_loan$total_pymnt <- discretize(data_loan$total_pymnt, methods = NULL, default = NULL)
data_loan$installment <- discretize(data_loan$installment, methods = NULL, default = NULL)
data_loan$loanPeriod <- discretize(data_loan$loanPeriod, methods = NULL, default = NULL)

#make dataset into transactions
data_loan <- as(data_loan, "transactions")
```

We looked at an item frequency plot which shows which item or specific feature was most frequent.

freqplot <- itemFrequencyPlot(data_loan, topN = 20, support=0.1, cex.names=0.5)



As you can see in the above plot, the most frequent item in the transactions data was the application_type_cat=1 which means the most types of application loans was individual. The next frequent was loan_condition_cat=0, which represents most of the loans resulted in Good Loans. The next was income_cat=1 which means most loans given were to borrowers who had low income. Keeping this in mind, we can move towards creating the rules. Using the default parameters in the apriori function, there were 74,860 rules, which is too many to deal with.

rules <- apriori(data_loan, parameter = list(support = 0.001, conf=0.8))

Since there are too many rules created, we played with the support and confidence parameters to lower the number of rules. In the end, we were able to lower it to 22 rules. With these 22 rules, it is best to look at the rules with high support and confidence. So we can sort the rules by making sure it shows the values with support above 0.5 and then sort the rules by confidence.

```
ruleset <- apriori(data=data_loan, parameter = list(support = .4, confidence = 0.4),
appearance = list(default="lhs", rhs="loan_condition_cat=0"))

goodrules <- ruleset[quality(ruleset)$support > 0.50]

rules_conf <- sort(goodrules, by="confidence", decreasing = TRUE)

inspect(head(rules_conf))
```

```
    lhs                            rhs                       support confidence     lift  count
[1] {interest_payment_cat=1}    => {loan_condition_cat=0} 0.5031210  0.9594749 1.038378 446459
[2] {application_type_cat=1,
     interest_payment_cat=1}    => {loan_condition_cat=0} 0.5029001  0.9594598 1.038361 446263
[3] {term_cat=1}                => {loan_condition_cat=0} 0.6515750  0.9308819 1.007433 578194
[4] {term_cat=1,
     application_type_cat=1}    => {loan_condition_cat=0} 0.6513068  0.9308569 1.007406 577956
[5] {income_cat=1,
     term_cat=1}                => {loan_condition_cat=0} 0.5410180  0.9264263 1.002611 480088
[6] {income_cat=1,
     term_cat=1,
     application_type_cat=1}    => {loan_condition_cat=0} 0.5407599  0.9263956 1.002578 479859
```

These rules show which features lead to a Good Loan, which is the target variable we want to focus on. With these rules, we can infer that these features are the best predictors for a good loan. This means that bank lenders should focus on these factors, if they want to ensure that their borrower will be able to repay the loan on time. So, loans with a high interest payment, an individual application type, the term only being 36 months, and a borrower with a low income can result in a good loan. To take a closer look at the good predictors, we decided to look at them visually.

Clustering

To perform clustering, the necessary packages that need to be installed is Rweka.

```
install.packages("RWeka")
library(RWeka)
```

Using the RWeka package, we can use the kmeans algorithm to take a look at the clustering. But for kmeans to work, the columns must be numeric because kmeans computes the means of values. So, must first make factor columns into numeric.

```
clus_data <- data
clus_data$home_ownership_cat <- as.numeric(clus_data$home_ownership_cat)
clus_data$income_cat <- as.numeric(clus_data$income_cat)
clus_data$term_cat <- as.numeric(clus_data$term_cat)
clus_data$application_type_cat <- as.numeric(clus_data$application_type_cat)
clus_data$purpose_cat <- as.numeric(clus_data$purpose_cat)
clus_data$interest_payment_cat <- as.numeric(clus_data$interest_payment_cat)
clus_data$loan_condition_cat <- as.numeric(clus_data$loan_condition_cat)
clus_data$grade_cat <- as.numeric(clus_data$grade_cat)
```

Since the region column cannot become numeric, we can just remove it, since it will not help the clustering.

```
clus_data <- clus_data[,-16]
```

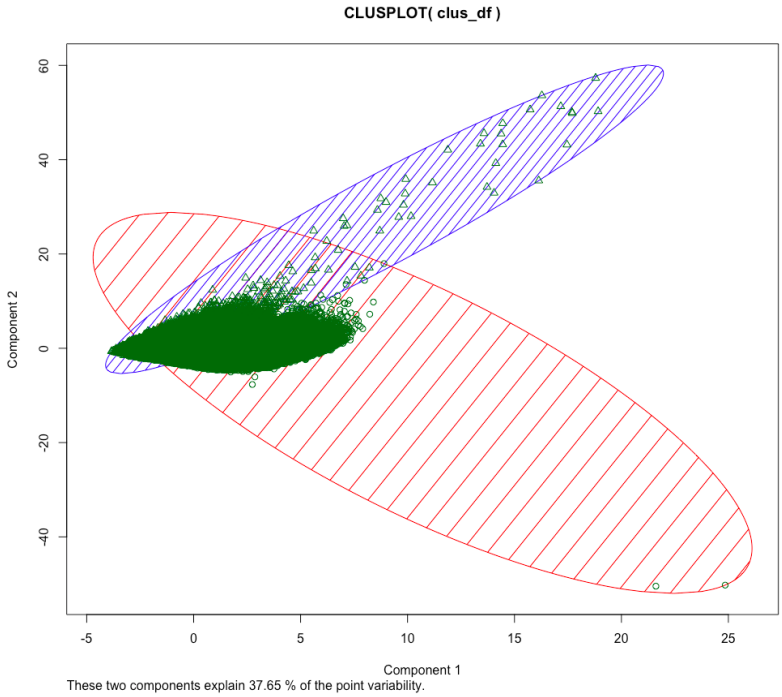
We then standardized the columns. Once the columns are standardized, we can use the kmeans algorithm.

```
clus_df <- scale(clus_data)
model_r <- kmeans(clus_df, centers = 2, nstart=25)
```

```
> model_r
K-means clustering with 2 clusters of sizes 465547, 421832

Cluster means:
  emp_length_int home_ownership_cat  annual_inc  income_cat loan_amount   term_cat
1   -0.007619558         0.04794703  0.06213875  0.06422598  -0.1115518 -0.3377282
2    0.008409183        -0.05291584 -0.06857827 -0.07088181   0.1231120  0.3727274
  application_type_cat purpose_cat interest_payment_cat loan_condition_cat interest_rate
1         -0.006544008  -0.1315117           -0.9485095         -0.1360772    -0.7632805
2          0.007222172   0.1451405            1.0468047          0.1501791     0.8423803
   grade_cat         dti total_pymnt installment  loanPeriod
1 -0.7340062 -0.06874231  -0.1258926 -0.09651358 -0.03065219
2  0.8100722  0.07586616   0.1389390  0.10651541  0.03382872
```

In the above picture, we see that there are two clusters which makes sense because we only want to predict if a loan will default or not. To visualize the clusters, it is best to plot it:

```
install.packages("cluster")
library(cluster)
clusplot(clus_df, model_r$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



Taking a look at the cluster, at first, we did not think it was interpretable or there was an error. But after further research and understanding, we can conclude that the clusters actually just show that there is overlap between loans that default and not for a reason. Since most of the loans did not default, explains the huge clump of data (the green area). And because a loan defaulting does not necessarily mean it is black or white. There is no definitive reason for a loan to default because even a person with high income can still default if they are not making smart choices in repaying their loans. So, in conclusion, banks can still and should take a risk because in most cases, the loans do not default. There are some cases where there are defaults, but there is no definitive reason for it, which is why the cluster has a few data points that are very far apart from the main data points.

Naive Bayes

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes Theorem with strong naive assumptions between the features.

Install the following required packages for confusion matrix and data manipulation respectively.

```
install.packages('e1071')
install.packages('caret')
library(e1071)
library(caret)
```

Train the model in the following manner.

```
nbTrain <- naiveBayes(loan_condition_cat ~ ., data = trainData)
```

Predict the outcome as follows on train data.

nbTrainPred <- predict(nbTrain, trainData, type = 'class')

Check the accuracy of train data using the confusion matrix.

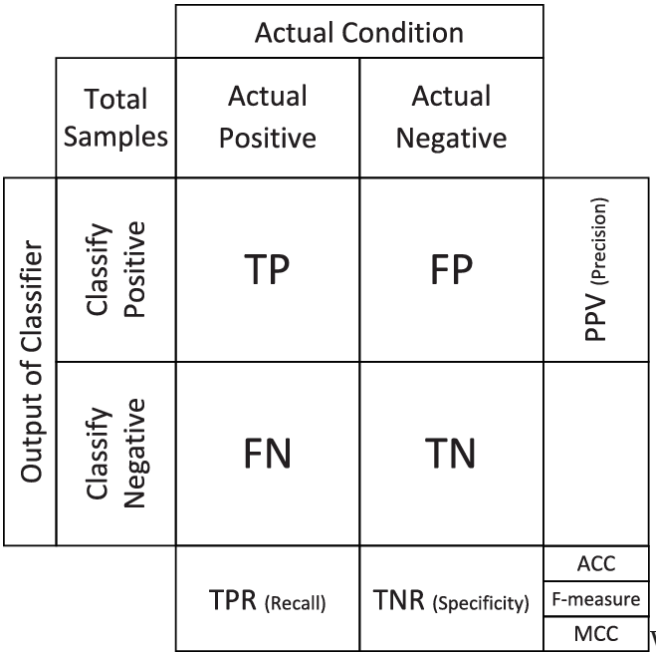
confusionMatrix(nbTrainPred, as.factor(trainData$loan_condition_cat))

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 42642 22048
         1 24787 45381

               Accuracy : 0.6527
                 95% CI : (0.6502, 0.6553)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3054
```

Predict the above on test data and check the accuracy.

nbTrainPred <- predict(nbTrain, testData, type = 'class')
confusionMatrix(nbTrainPred, as.factor(testData$loan_condition_cat))

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 14089  7308
         1  8162 14943

               Accuracy : 0.6524
                 95% CI : (0.6479, 0.6568)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3048
```



We can see the syntax of a confusion matrix here. So, it it is evident that in our model, 14089 values are True positive and 14943 are True Negative. In situations like this, we can say that actual negative when given as positive can be allowed to consider as a good predictor as it is not life threatening but just risk taking. Here that value is 7308.

Hence, we can say that the accuracy of the model is around 65.24 %.
It is also observed that however randomly we sample the data, the accuracy of the model is around 65%.

Decision Tree

Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems. For this, we begin with installing the package which is necessary to build trees i.e. tree.

```
install.packages("tree")
library(tree)
```

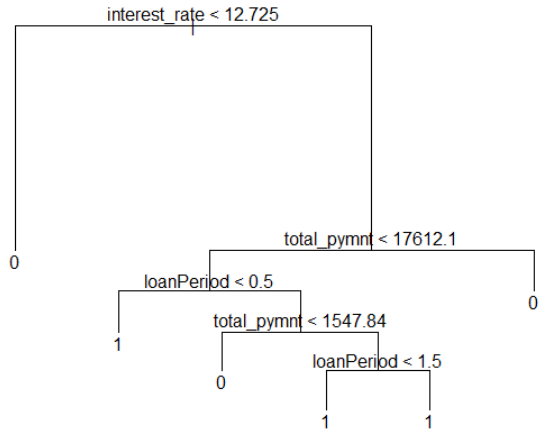
Then we build the decision tree on the target variable, loan_condition_cat and check its summary.

```
tree.loanData <- tree(loan_condition_cat~., data = trainData)
summary(tree.loanData)
```

```
Classification tree:
tree(formula = loan_condition_cat ~ ., data = trainData)
Variables actually used in tree construction:
[1] "interest_rate" "total_pymnt"   "loanPeriod"
Number of terminal nodes:  6
Residual mean deviance:  1.247 = 168100 / 134900
Misclassification error rate: 0.3328 = 44875 / 134858
```

Here, we see the number of nodes, residual mean deviance and the misclassification error rate. To make it perceptible, we plot it and annotate it using the text function.

```
plot(tree.loanData)
text(tree.loanData, splits = TRUE, all = FALSE ,pretty = 0)
```



For detailed summary, we will print the tree.

tree.loanData

```
1) root 134858 187000 0 ( 0.5000 0.5000 )
  2) interest_rate < 12.725 49728  61940 0 ( 0.6853 0.3147 ) *
  3) interest_rate > 12.725 85130 114000 1 ( 0.3918 0.6082 )
    6) total_pymnt < 17612.1 77259 101600 1 ( 0.3676 0.6324 )
     12) loanPeriod < 0.5 17490  18720 1 ( 0.2266 0.7734 ) *
     13) loanPeriod > 0.5 59769  80860 1 ( 0.4089 0.5911 )
       26) total_pymnt < 1547.84 7097   9210 0 ( 0.6477 0.3523 ) *
       27) total_pymnt > 1547.84 52672  69780 1 ( 0.3767 0.6233 )
         54) loanPeriod < 1.5 32614  40040 1 ( 0.3036 0.6964 ) *
         55) loanPeriod > 1.5 20058  27800 1 ( 0.4955 0.5045 ) *
    7) total_pymnt > 17612.1 7871  10380 0 ( 0.6290 0.3710 ) *
```

After building the tree, we will predict the tree using predict function and class labels. Then we will evaluate the error rate using the misclassification table.

```
tree.loanDataPred <- predict(tree.loanData,testData,type = "class")
with(testData,table(tree.loanDataPred,loan_condition_cat))
```

```
                  loan_condition_cat
tree.loanDataPred     0     1
                0 14359  6908
                1  7892 15343
```

We can calculate the error rate by taking the sum of two diagonals and divide it by the total.

$$\frac{29702}{44502} \times 100 = 66.74\%$$

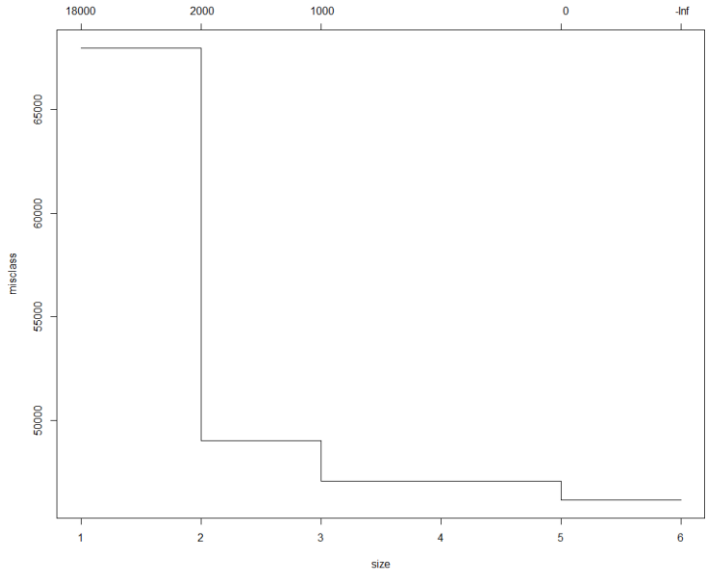
Then, we used the cross-validation technique and misclassification error to prune the tree.

```
cv.loanData = cv.tree(tree.loanData, FUN = prune.misclass)
cv.loanData
```

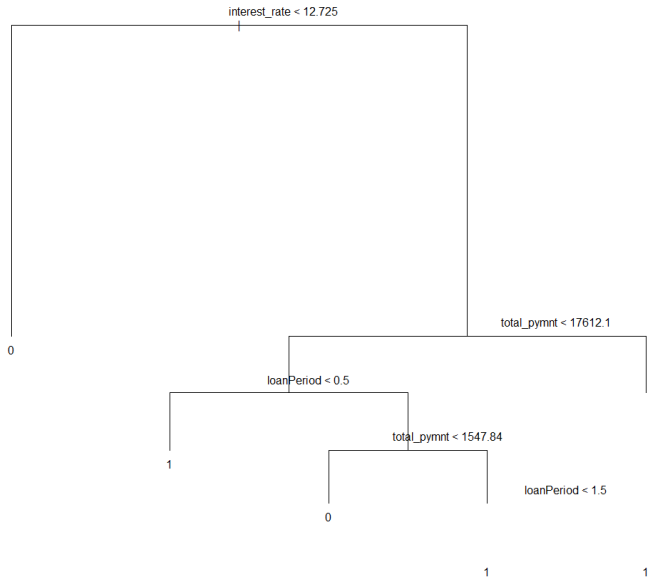
```
$size
[1] 6 5 3 2 1

$dev
[1] 46159 46159 47054 49003 67965

$k
[1]    -Inf     0.0  1048.5  2031.0 18426.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

Here we observed the size, deviance and cost complexity parameter as the tree was pruned. For more clarity we plot it out.

```
plot(cv.loanData)
```



Looking at the plot, you see a downward spiral part because of the misclassification error on cross-validated points. So, we pick the value 5 as we have less error rate and prune the tree. Finally, let's plot and annotate that tree to see the outcome.



Then we use the pruned tree to evaluate the test datatset.

```
tree.loanDataPred <- predict(prune.loanData,testData,type = "class")
with(testData,table(tree.loanDataPred,loan_condition_cat))
```

```
                  loan_condition_cat
tree.loanDataPred     0     1
                0 14359  6908
                1  7892 15343
```

We can calculate the error rate by taking the sum of two diagonals and divide it by the total.

$$\frac{29702}{44502} \times 100 = 66.74\%$$

So, we can conclude that the accuracy of the decision tree is 66.74% with pruning.

Random Forest

Trying to improve the accuracy of the prediction, we go one step further to make use of random forest technique. Random Forest is used to improve the predictive performance of Decision Trees by reducing the variance in the trees by averaging them. Decision Trees are considered very simple and easily interpretable as well as understandable modelling techniques, but a major drawback in them is that they have a poor predictive performance and poor generalization on test dataset. Random forest is a combination of multiple decision trees and is more robust and powerful compared to decision tree.
To implement this, we first load the necessary library.

```
install.packages("randomForest")
library(randomForest)
```

Now, we will build a Random Forest model with default parameters and then we will fine tune the model by changing 'mtry'. We can tune the random forest model by changing the number of trees (ntree) and the number of variables randomly sampled at each stage (mtry).

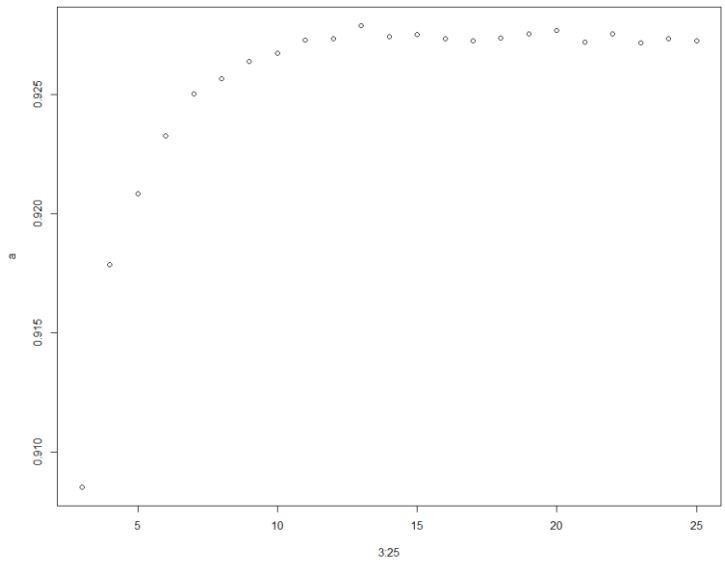
```
# random forests with default parameters
rf.loanData = randomForest(loan_condition_cat~., data = trainData, importance = TRUE)
rf.loanData
```

```
Call:
 randomForest(formula = loan_condition_cat ~ ., data = trainData,       importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 17.39%
Confusion matrix:
      0     1 class.error
0 55093 12336   0.1829480
1 11112 56317   0.1647956
```

We then used a for loop to find the mtry value for which the accuracy is maximum. Then, we plot the accuracy with the mtry value.

```
a=c()
i=5
for (i in 3:25) {
 model3 <- randomForest(loan_condition_cat~., data = trainData, ntree = 500, mtry = i, importance = TRUE)
 predValid <- predict(model3, testData, type = "class")
 a[i-2] = mean(predValid == testData$loan_condition_cat)
}

a

plot(3:25,a)
```

```
> a
 [1] 0.9085434 0.9178689 0.9208350 0.9232619 0.9250371 0.9256663 0.9263853 0.9267224 0.9272842 0.9273516 0.9278909
[12] 0.9274190 0.9275089 0.9273291 0.9272617 0.9273741 0.9275313 0.9276886 0.9271943 0.9275313 0.9271718 0.9273291
[23] 0.9272617
```



We then tried to fine tune the model. We observe from the graph that we get highest accuracy at mtry 14 and 24. We choose 14 instead of 24 to reduce complexity of the model.

```
rf.loanDataTuned = randomForest(loan_condition_cat~., data = trainData, ntree = 500, mtry = 14, importance = TRUE)
rf.loanDataTuned
```

```
Call:
 randomForest(formula = loan_condition_cat ~ ., data = trainData,      ntree = 500, mtry = 14, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 14

        OOB estimate of  error rate: 15.63%
Confusion matrix:
      0     1 class.error
0 56592 10837   0.1607172
1 10245 57184   0.1519376
```

We see that error rate has decreased compared to the previous output.
We then predicted the output on the training dataset to measure the accuracy.

```
rf.trainDataPred <- predict(rf.loanDataTuned, trainData, type = "class")
table(rf.trainDataPred, trainData$loan_condition_cat)
```

```
rf.trainDataPred     0     1
               0 67429     0
               1     0 67429
```

After the training dataset, we predicted the model on test dataset. To check the accuracy, we used the confusion matrix.

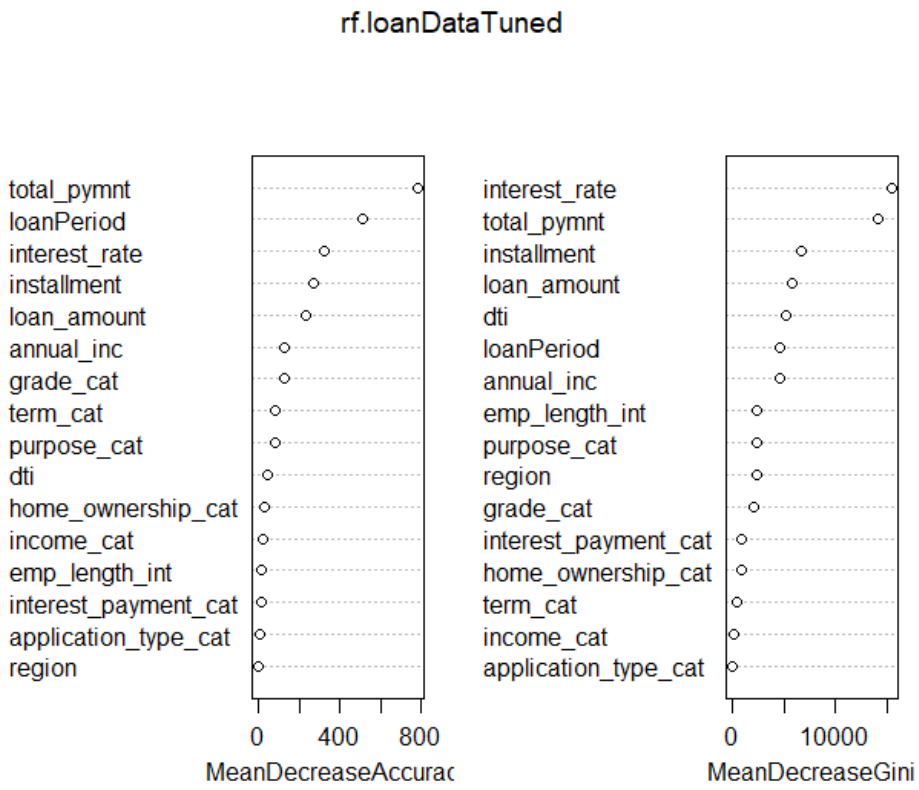
```
rf.testDataPred <- predict(rf.loanDataTuned, testData, type = "class", keep.forest=FALSE, importance = TRUE)
mean(rf.testDataPred == testData$loan_condition_cat)
table(rf.testDataPred,testData$loan_condition_cat)
```

```
> mean(rf.testDataPred == testData$loan_condition_cat)
[1] 0.9275089
> table(rf.testDataPred,testData$loan_condition_cat)

rf.testDataPred     0     1
              0 19025     0
              1  3226 22251
```

Then we observed the important parameters to give business recommendations.

importance(rf.loanDataTuned)
varImpPlot(rf.loanDataTuned)

	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
emp_length_int	10.008985	7.355737	12.004563	2281.402992
home_ownership_cat	19.152849	19.525396	29.018601	844.038090
annual_inc	83.700361	85.110184	130.863802	4540.311651
income_cat	11.233233	16.526165	23.039188	142.853427
loan_amount	131.926729	155.410988	232.526370	5744.190409
term_cat	80.733725	17.846215	84.425606	423.036271
application_type_cat	3.813765	6.352165	7.011285	5.247985
purpose_cat	94.495193	13.888848	80.630188	2280.470740
interest_payment_cat	10.541657	-4.181391	10.886488	881.860406
interest_rate	232.424763	41.886056	325.184042	15478.157165
grade_cat	135.387085	-83.968915	125.366530	1999.950574
dti	41.515764	19.111951	42.321748	5229.304542
total_pymnt	382.776384	402.930960	784.854252	14171.538050
installment	159.662257	164.466166	268.797820	6612.648508
region	1.003308	3.031509	2.791562	2252.763119
loanPeriod	206.679941	403.679486	510.637767	4540.646772



Lastly, checking the accuracy of the classification by taking the sum of diagonals and divind by the total.

$$\frac{41276}{44502} \times 100 = 92.75\%$$

With the random forest we can conclude that the accuracy comes out to be 92.75% which is very good.

Support Vector Machine

In our attempt to get even higher accuracy, we tried to use the Support Vector Machine. As for every other step, we begin with installing and importing libraries.

```
install.packages("kernlab")
library(kernlab)
```

While building this model on the entire training dataset, RStudio crashed several times and it even took days for the code to run. So, we executed the code on randomly sampled dataset.

```
svmSampleTrain <- sample_n(trainData,30000)
svm.model <- ksvm(loan_condition_cat~., data = svmSampleTrain, kernel = "rbfdot",
kpar="automatic", C=75, cross=50,
            prob.model=TRUE, type = "C-svc")
svm.model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 50

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0650154113023071

Number of Support Vectors : 8418

Objective Function Value : -277373
Training error : 0.125933
Cross validation error : 0.2614
Probability model included.
```

Then we used this model to predict the test dataset. Further, we used the confusion matrix to predict the accuracy.

```
svm.Pred <- predict(svm.model,testData)
table(svm.Pred,testData$loan_condition_cat)
```

```
svm.Pred     0     1
       0 16386  4193
       1  5865 18058
```

Model Comparison

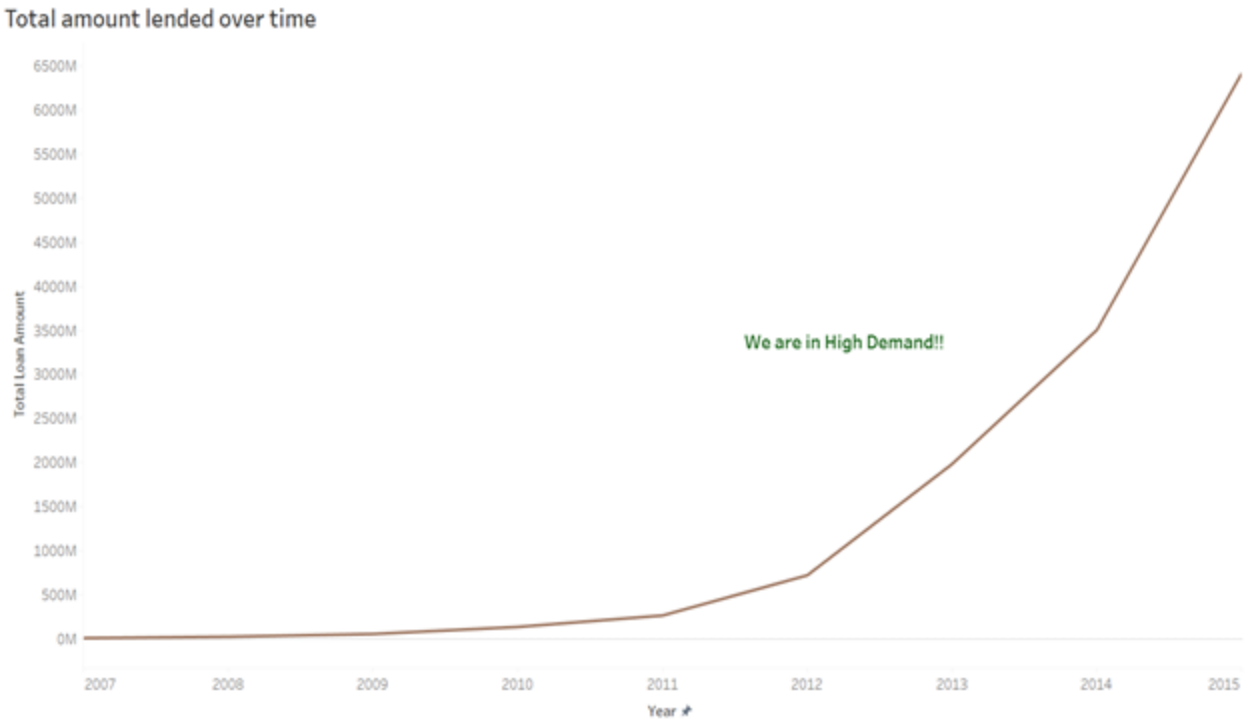
Model Name	Accuracy
Naïve Bayes	65.24 %
Decision Tree	66.74%
Random Forest	92.75%
Support Vector Machine	87.41%

Challenges

- Changing final_d column in an appropriate date format, so that it is easier to work with.
- Figuring out which columns are significant to our analysis, and which ones we should remove.
- Deciding whether to keep the category columns that have integers or the respective columns with the nominal values in them. We decided to use the category columns, because it already factors in the nominal values as numbers which is what we would need to use to make further analysis.
- Models like Random Forest and SVM took a lot of time to execute.

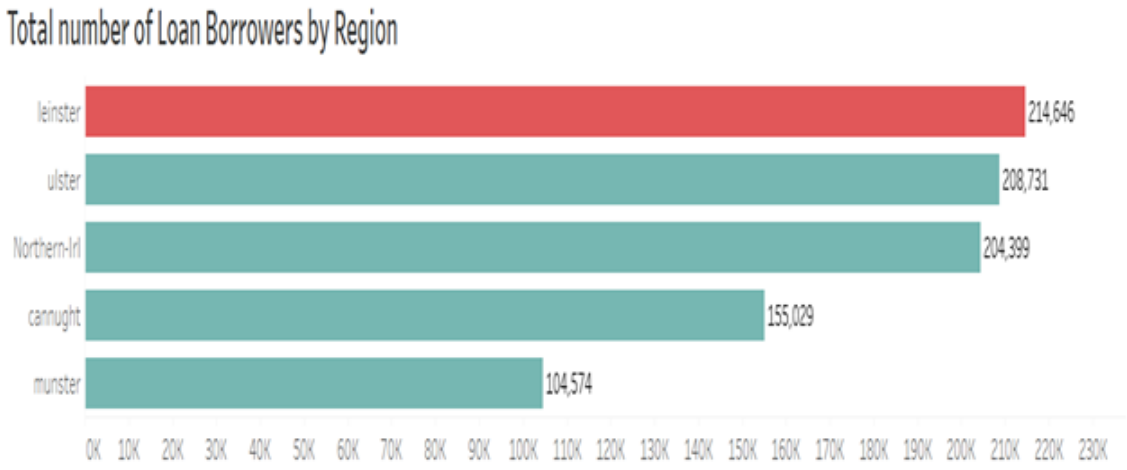
Conclusion

1. **What is the trend of total amount of loan lent over time?**
   Looking at the visualization, the borrowers can learn about the lending trend of the club and compare and contrast it with other lenders/ banks and can decide for themselves whether or not to borrow loan from here.
   It also works in the advantage of the lenders as they can quickly observe the amount of funds they have lent over time and see how they progress increasing the customer base ensuring their growth.



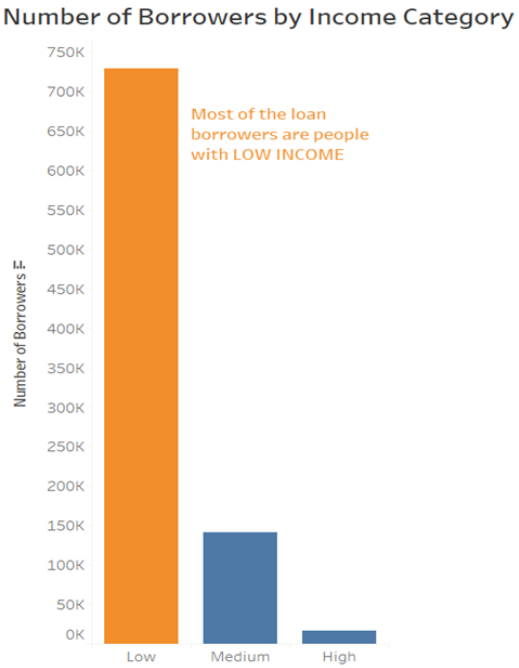
2. **To see which area has highest number of loan borrowers?**
   As economy varies for different regions, so does the employment opportunities and other factors like income, education level etc. The lender may want to strategize to maximize his returns by targeting the potential borrowers. So, we first begin with the following visualization to check which region has the highest number of loan borrowers.

   By using Tableau, we have visualized the top region where we have maximum number of loan borrowers which is Leinster. On hovering over the bars, we can even see the total number of loan borrowers in each region. By identifying top and bottommost region the lender may strategize marketing plans for better customer targeting.



3. **Which Income category people borrows maximum loan?**

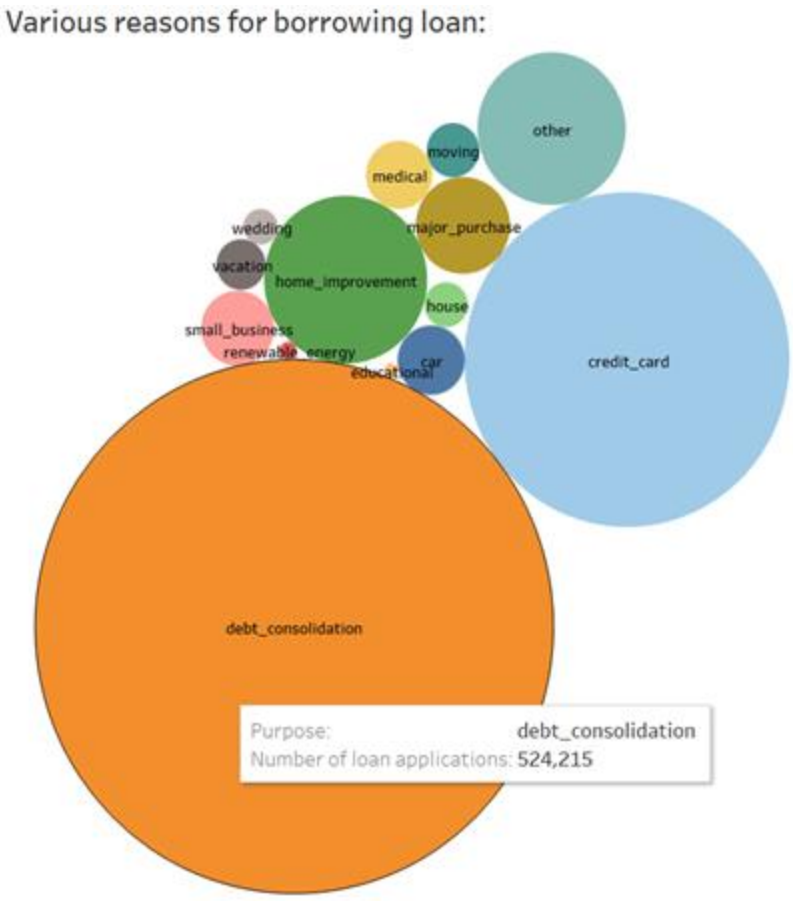
From the visualization below, we can see that most of the loan borrowers belong to the low category income. By identifying the top and bottom most regions, the lending club can plan marketing strategies for better customer targeting.



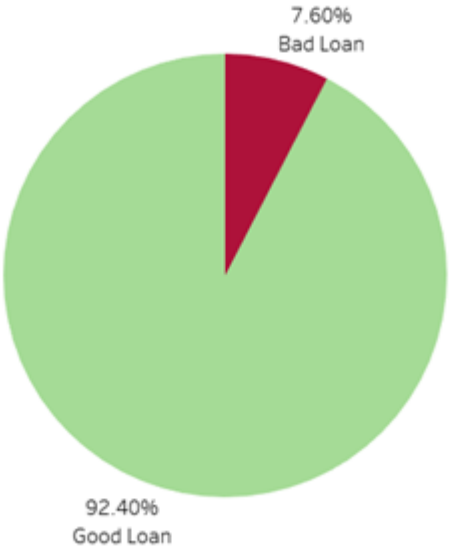
4. **How to design loan parameters depending on the purpose of the loan borrowed so as to maximize the returns?**

   Knowing the purpose for which the loan is borrowed can help the lenders devise the rate of interest and other investment factors smartly to come up with a loan plan to lure the borrowers and make more money from the extra savings they possess.

   Using Tableau, we created bubble chart where the size of the bubble is proportionate to the number of loan applications received for that particular reason. Here, we see various reasons for borrowing loan and from the visualization below, we can easily observe that debt consolidation, credit card and home improvement are the top three major purposes of borrowing loan.



5. **To predict whether the investment is worth taking risk or not? Whether the borrower would default on loan or not?**



From the given dataset, we found that there were approximately 8% of the instances where the borrower defaulted on loan and 92% of the instances where borrower did not default on loan.

To begin with, we use various strategies to create a model that would predict whether the borrower would default on loan or not.

From the data analysis, we learn that the amount of funds lent every year by Lending Company keeps rising. We observe that Leinster is top region where we have maximum number of loan borrowers. We also found out that most of the loan borrowers are people with low income. Out of all the reasons for borrowing loan, the top 3 reasons are debt consolidation, credit card, home improvement. We also observe that 92% of loan borrowers did not default on loan while 8% of loan borrowers defaulted on loan so far. The Random Forest model created predicts accurately 92.77% of the times whether a person would default on a loan or not.

References

1. Kaggle Dataset: https://www.kaggle.com/mrferozi/loan-data-for-dummy-bank
2. Decision Trees in R. (n.d.). Retrieved from https://www.datacamp.com/community/tutorials/decision-trees-R
3. How to implement Random Forests in R. (2018, January 09). Retrieved from https://www.r-bloggers.com/how-to-implement-random-forests-in-r/