Data 622 - Homework #3

Mengqin Cai, Zhi Ying Chen, Donny Lofland, Grace Han, Zach Alexander

4/5/2021

Part 1: KNN on the Penguins dataset

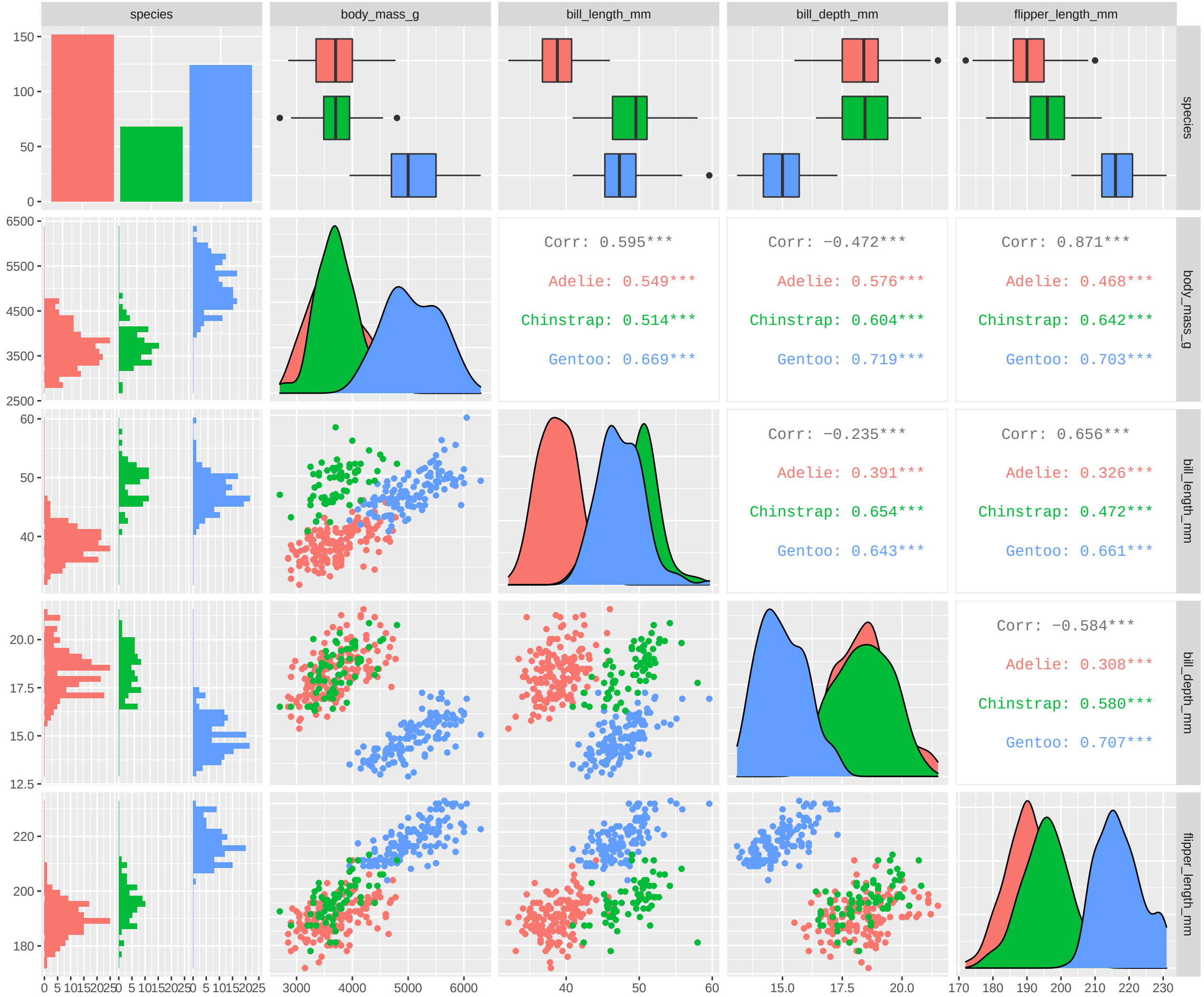
Please use K-nearest neighbor (KNN) algorithm to predict the species variable. Please be sure to walk through the steps you took. (40 points)

Similar to past assignments when using the Palmer Penguins dataset, we'll first do some quick exploratory analysis to examine the different features available.

We can see below, that there are four continuous variables: `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm` and `body_mass_g`. Additionally, there are a few categorical variables: `island`, `sex`, and `year`.

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

For the continuous variables, we can examine the distributions, broken out by the target variable, `species`:



By separating our distributions by our target variable, `species`, we can see that many of the feature interactions show clustering between Adelie and Chinstrap penguins (red and green), while Gentoo penguins tend to contrast the other two species for most interactions. This is also confirmed by most of the single-variable distributions split out by species in the plots below. With the exception of the distribution of `bill_length_mm`, distributions for `body_mass_g`, `bill_depth_mm`, and `flipper_length_mm` all show there to be overlapping distributions between Adelie and Chinstrap penguin species.

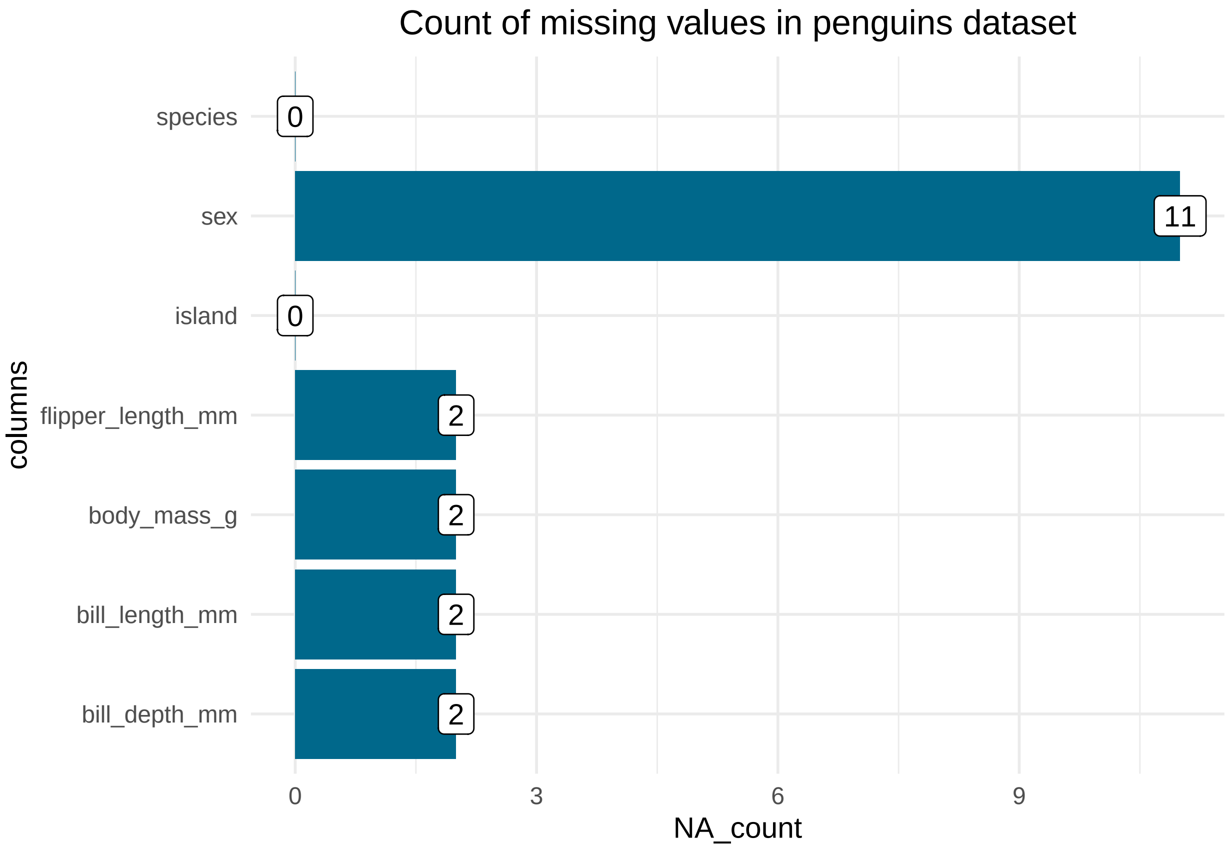
Next, we'll do some data tidying to get our dataset ready for our KNN model. Since `year` reflects the date/time of recording, and is not beneficial for our machine learning algorithm, we'll remove it from our dataset:

```r
penguins <- penguins %>% dplyr::select(-year)
```

Additionally, when looking at the number of missing values, we can see the following:

```r
penguins %>%
  summarise_all(funs(sum(is.na(.)))) %>%
  pivot_longer(cols = 1:7, names_to = 'columns', values_to = 'NA_count') %>%
  arrange(desc(NA_count)) %>%
  ggplot(aes(y = columns, x = NA_count)) + geom_col(fill = 'deepskyblue4') +
  geom_label(aes(label = NA_count)) +
  theme_minimal() +
  labs(title = 'Count of missing values in penguins dataset') +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```



We can see that 11 individuals have at least one missing data point. Therefore, we'll drop these from our dataset as well.

```
penguins <- na.omit(penguins)
```

Now, we can take a look at a summary of data before splitting to get a sense of what still needs to be tidy'd in order to get it ready for our KNN model:

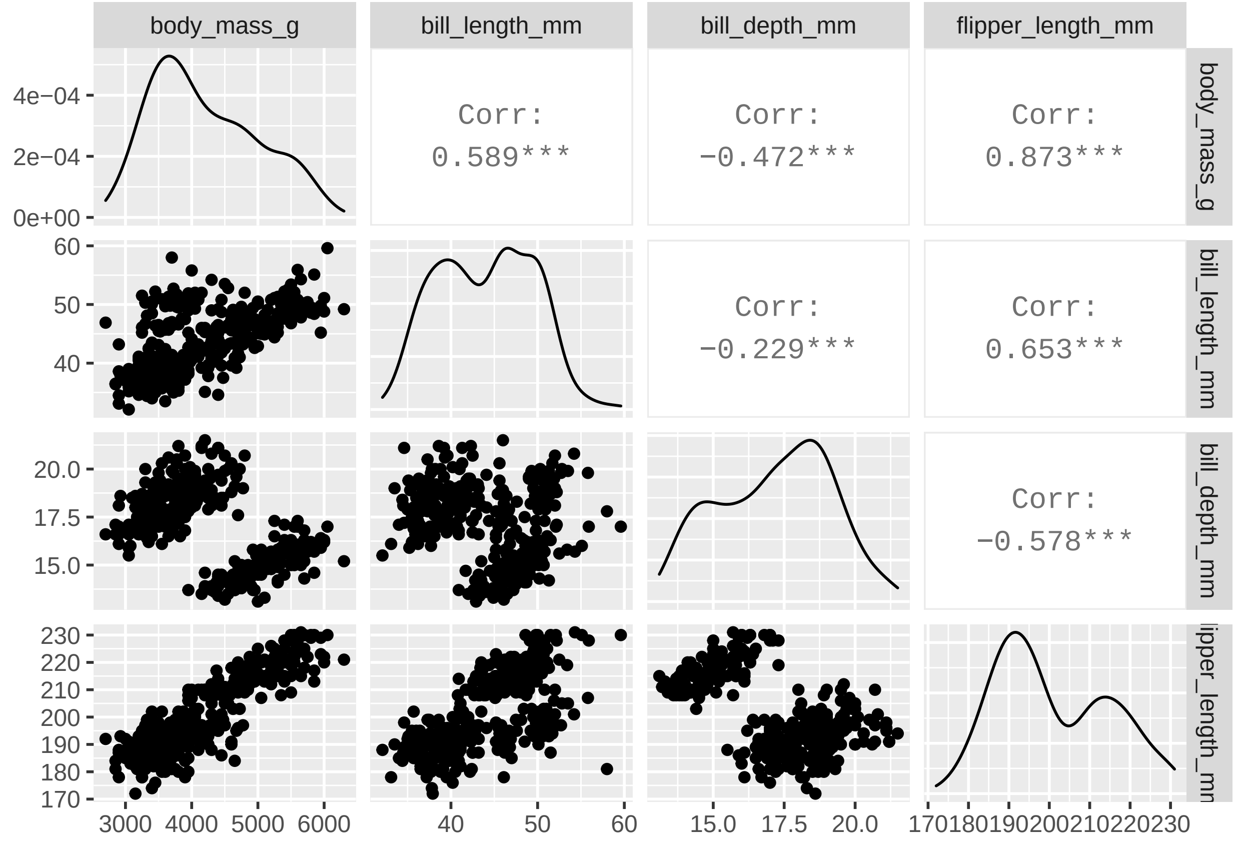
```
summary(penguins)
```

```
##       species          island    bill_length_mm  bill_depth_mm  
##  Adelie   :146   Biscoe   :163   Min.   :32.10   Min.   :13.10  
##  Chinstrap: 68   Dream    :123   1st Qu.:39.50   1st Qu.:15.60  
##  Gentoo   :119   Torgersen: 47   Median :44.50   Median :17.30  
##                                  Mean   :43.99   Mean   :17.16  
##                                  3rd Qu.:48.60   3rd Qu.:18.70  
##                                  Max.   :59.60   Max.   :21.50  
##  flipper_length_mm  body_mass_g       sex     
##  Min.   :172       Min.   :2700   female:165  
##  1st Qu.:190       1st Qu.:3550   male  :168  
##  Median :197       Median :4050               
##  Mean   :201       Mean   :4207               
##  3rd Qu.:213       3rd Qu.:4775               
##  Max.   :231       Max.   :6300               
```

From above, we can see that we'll need to remove our species variable, and save it to a separate outcome variable in order to evaluate the performance of our KNN model later on. We can do this by running the following syntax:

```
species_actual <- penguins %>% dplyr::select(species)
penguins <- penguins %>% dplyr::select(-species)
```

Additionally, we can see below in the distributions of each of our continuous variables that the scales are inconsistent across features. For better model performance, we'll want to standardize each of our features.

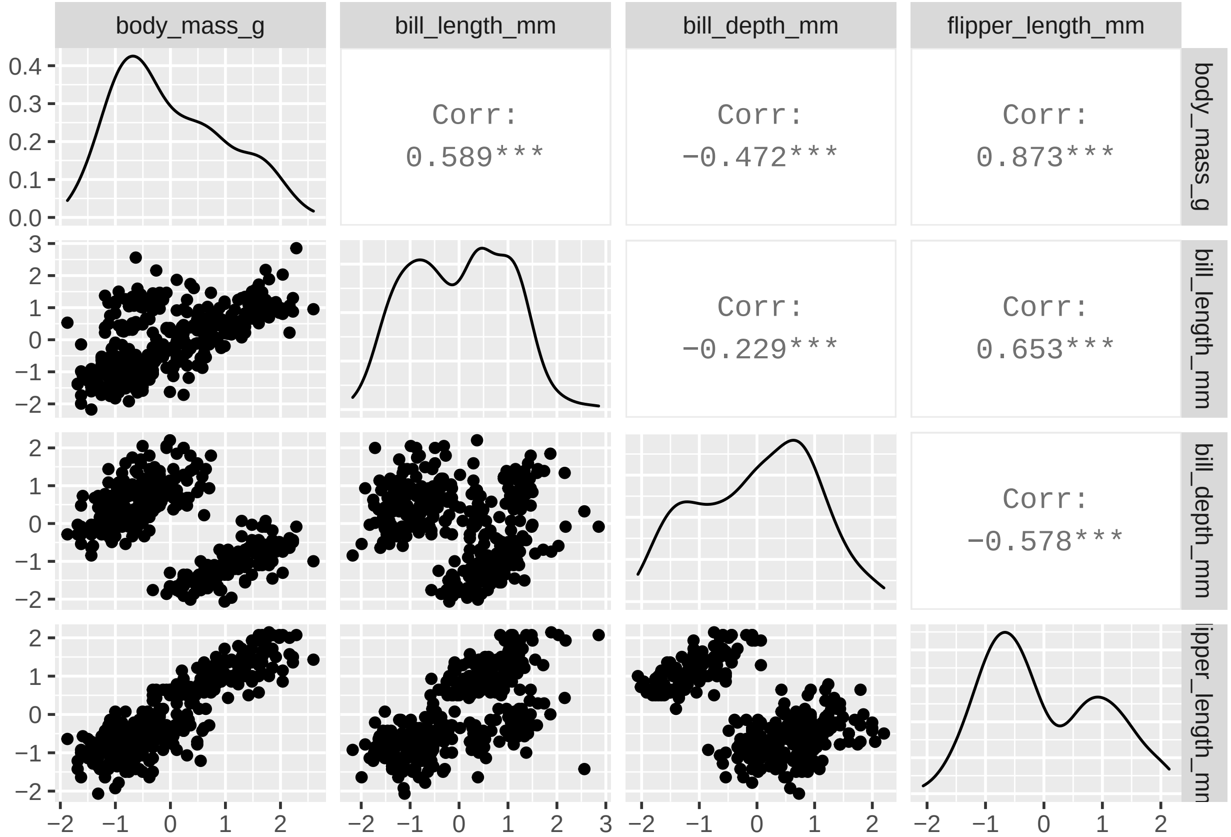


In order to do this, we'll administer a z-score standardization to fix our scaling, which will ultimately help with our clustering.

```
penguins[, c("bill_length_mm","bill_depth_mm", "flipper_length_mm", "body_mass_g")] <- scale(penguins[,
```

As you can see below, after the z-score standardization, the scaling on the x and y axis is a lot more consistent across our features.

```
penguins %>%
  dplyr::select(body_mass_g, bill_length_mm, bill_depth_mm, flipper_length_mm) %>%
  ggpairs()
```



Although this may help with our KNN model, we'll have to be careful about interpretability later on! Finally, we'll want to dummy code our `island` and `sex` variables, since they are categorical, we'll need to convert them to 1s and 0s:

```
island_dcode <- as.data.frame(dummy.code(penguins$island))
penguins <- cbind(penguins, island_dcode)
```

We'll also want to remove our original `island` variable from this dataset, since we created our dummy variables:

```
penguins <- penguins %>% dplyr::select(-island)
```

The same process will then be applied to the `sex` variable:

```
penguins$sex <- dummy.code(penguins$sex)
```

Finally, after a fair amount of data tidying and investigation, our dataset is ready for our KNN model. We can take a quick look at the updated version of our dataset:

```
kable(head(penguins)) %>% kable_styling(bootstrap_options = "basic")
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	Biscoe	Dream	Torgersen
1	-0.8946955	0.7795590	-1.4246077	-0.5676206	0	1	0	0
2	-0.8215515	0.1194043	-1.0678666	-0.5055254	1	0	0	0
3	-0.6752636	0.4240910	-0.4257325	-1.1885721	1	0	0	0
5	-1.3335592	1.0842457	-0.5684290	-0.9401915	1	0	0	0
6	-0.8581235	1.7444004	-0.7824736	-0.6918109	0	1	0	0
7	-0.9312674	0.3225288	-1.4246077	-0.7228585	1	0	0	0

Splitting into training and testing datasets

With our dataset tidy'd, we were then able to split it into a training and test set.

```
sample_size<-floor(0.8*nrow(penguins))

set.seed(123)
train_ind<-sample(seq_len(nrow(penguins)),size=sample_size)
train_penguins<-penguins[train_ind,]
test_penguins<-penguins[-train_ind,]
```

We'll also do the same for our `target` variable, using the same split:

```
species_actual_train <- species_actual$species[train_ind]
species_actual_test <- species_actual$species[-train_ind]
```

Fitting the KNN model

Now, with our data split accordingly, we'll need to identify the appropriate number for k. To do this, we will first take a standard approach of calculating the square root of the number of rows in our training dataset:

```
sqrt(nrow(train_penguins))
```

```
## [1] 16.30951
```

We can see above, that it is roughly between 16 and 17. Therefore, we'll perform two KNN models with k=16 and k=17:

```
set.seed(123)
k16<-knn(train_penguins,test_penguins,cl=species_actual_train,k=16)
k17<-knn(train_penguins,test_penguins,cl=species_actual_train,k=17)
```

```
misClassError <- mean(k16 != species_actual_test)
paste0('The number of misclassified penguins with 16 neighbors is: ', misClassError)
```

```
## [1] "The number of misclassified penguins with 16 neighbors is: 0"
```

```
table(k16, species_actual_test)
```

```
##              species_actual_test
## k16           Adelie Chinstrap Gentoo
##   Adelie        28         0      0
##   Chinstrap      0        15      0
##   Gentoo         0         0     24
```

```
misClassError <- mean(k17 != species_actual_test)
paste0('The number of misclassified penguins with 17 neighbors is: ', misClassError)
```

```
## [1] "The number of misclassified penguins with 17 neighbors is: 0"
```

```
table(k17, species_actual_test)
```

```
##              species_actual_test
## k17           Adelie Chinstrap Gentoo
##   Adelie        28         0      0
##   Chinstrap      0        15      0
##   Gentoo         0         0     24
```

From our resulting confusion matrices, that show the classified penguins from our KNN model compared to the actual species designations, we can see that both a k of 16 and a k of 17 preformed well on our test set. Essentially, the misclassification rate showed no difference with k=16 or k=17. Although this approach seemed to work quite effectively, we do need to be careful about how we'd go about using either one of these models to predict new data. With an artificially high k value, we could be underfitting the data.

Therefore, as a final step, we can run one final KNN model using the `caret` package, where the built-in `train()` function will run KNN classification that automatically picks the optimal number of neighbors (k).

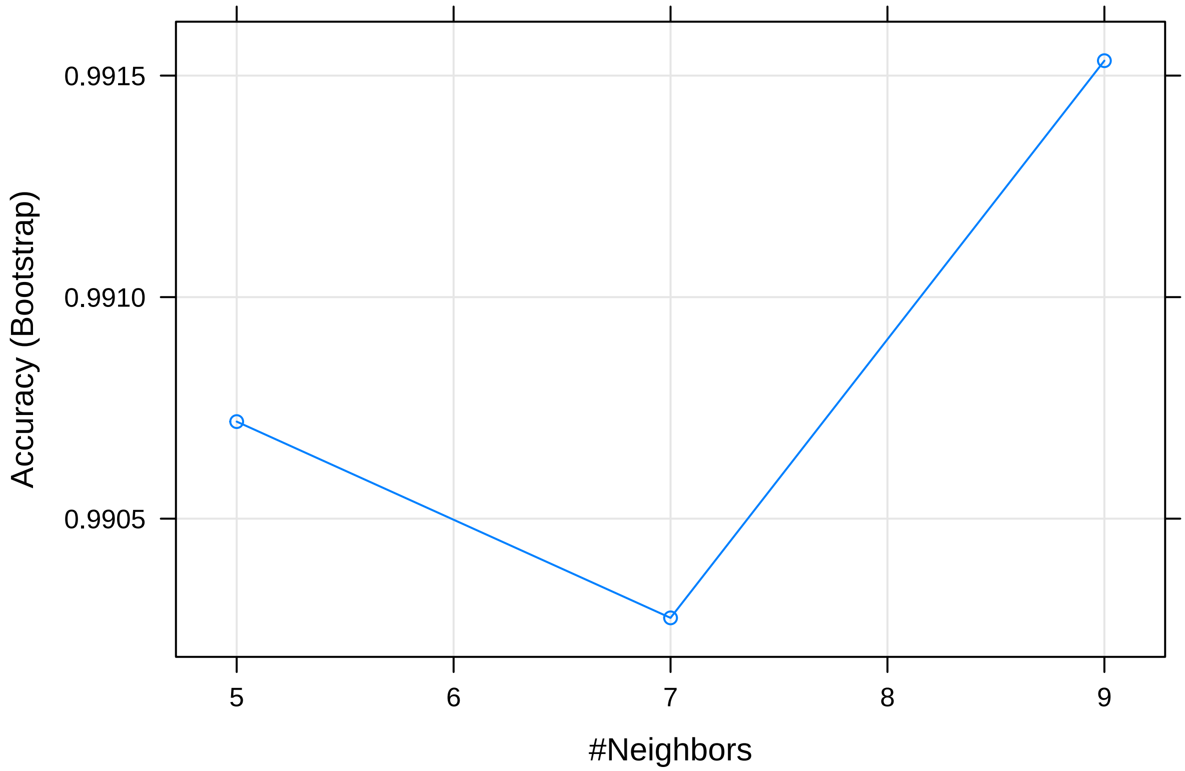
```
knn_caret <- train(train_penguins, species_actual_train, method = "knn", preProcess = c("center", "scal

knn_caret
```

```
## k-Nearest Neighbors
##
## 266 samples
##   8 predictor
##   3 classes: 'Adelie', 'Chinstrap', 'Gentoo'
##
## Pre-processing: centered (7), scaled (7), ignore (1)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 266, 266, 266, 266, 266, 266, ...
## Resampling results across tuning parameters:
##
```

```
##   k  Accuracy   Kappa    
##   5  0.9907191  0.9854142
##   7  0.9902761  0.9847094
##   9  0.9915336  0.9866120
## 
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

From our output, we can see that a k value of 9 was chose as the optimal value for our model based on the Accuracy and Kappa values calculated. Since this inter-rater reliability metric of Kappa is quite important when working with unbalanced datasets such as our penguins dataset, this k value of 9 may ultimately perform better than our initial KNN models where k=16 or k=17. We can see this relationship determined by the `train` function for finding the optimal neighbors plotted below:

```
plot(knn_caret)
```



We can see that a neighbors value of 9 clearly showed the highest accuracy value.

```
knn_caret_predictions <- predict(knn_caret, newdata = test_penguins)
confusionMatrix(knn_caret_predictions, as.factor(species_actual_test))
```

```
## Confusion Matrix and Statistics
## 
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        28         0      0
##   Chinstrap      0        15      0
##   Gentoo         0         0     24
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9464, 1)
##     No Information Rate : 0.4179
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Adelie Class: Chinstrap Class: Gentoo
## Sensitivity                 1.0000           1.0000        1.0000
## Specificity                 1.0000           1.0000        1.0000
## Pos Pred Value              1.0000           1.0000        1.0000
## Neg Pred Value              1.0000           1.0000        1.0000
## Prevalence                  0.4179           0.2239        0.3582
## Detection Rate              0.4179           0.2239        0.3582
## Detection Prevalence        0.4179           0.2239        0.3582
## Balanced Accuracy           1.0000           1.0000        1.0000
```

In the end, with the `caret` package identifying the "optimal" k-value, we can see that all three of our KNN models performed very well on our test dataset. Although it will likely depend on the utility of the KNN model, it does appear that any of our three models could be chose to provide accurate predictions of the three penguin species.

Part 2: Decision Trees on loan approval dataset

Please use the attached dataset on loan approval status to predict loan approval using Decision Trees. Please be sure to conduct a thorough exploratory analysis to start thetask and walk us through your reasoning behind all the steps you are taking.

First, we decided to read in the loan approval dataset and take a look at its features:

```
loan <- read.csv("https://raw.githubusercontent.com/DaisyCai2019/NewData/master/Loan_approval.csv")
kable(head(loan)) %>% kable_styling(bootstrap_options = "basic")
```

Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
LP001002	Male	No	0	Graduate	No	5849	0
LP001003	Male	Yes	1	Graduate	No	4583	1508
LP001005	Male	Yes	0	Graduate	Yes	3000	0
LP001006	Male	Yes	0	Not Graduate	No	2583	2358
LP001008	Male	No	0	Graduate	No	6000	0
LP001011	Male	Yes	2	Graduate	Yes	5417	4196

As we can see from a glimpse of the dataset above, the following features are available:

- **Loan_ID**: a unique identifier for each loan
- **Gender**: split into male/female
- **Married**: indicates whether the applicant is either married ("Yes") or not married ("No")
- **Dependents**: records the number of dependents to the applicant
- **Education**: indicates whether the applicant is a graduate or undergraduate student
- **Self_Employed**: indicates whether the applicant is either self employed ("Yes) or not ("No")
- **ApplicantIncome**: indicates the applicant's income
- **CoapplicantIncome**: indicates the coapplicant's income
- **LoanAmount**: indicates the loan amount (in thousands)
- **Loan_Amount_Term**: indicates the loan amount term in number of months
- **Credit_History**: indicates whether or not the applicant's credit history meets loan guidelines (1 or 0)
- **Property_Area**: indicates whether the applicant's property is "urban", "semi urban" or "rural"
- **Loan_Status**: the target variable, indicates whether or not the applicant received the loan

---

Exploratory data analysis of the loan approval dataset

Now, we can run some exploratory data analysis to get a better sense of how to tidy and interpret these features. Here's an initial summary of the dataset:

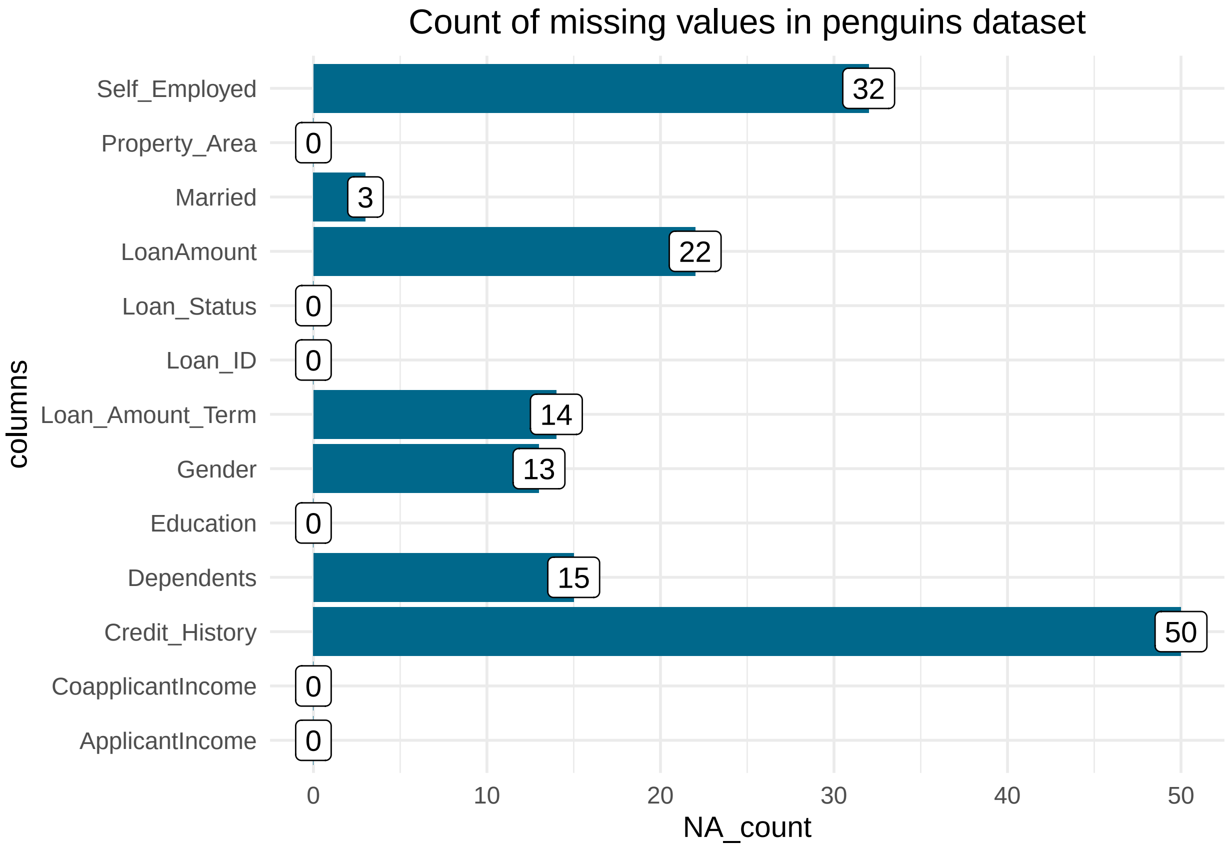
```
summary(loan)
```

```
##      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
```

We can see that there are a fair amount of things we'll need to do to clean the dataset before being able to run our decision tree algorithm. First, we can see that there quite a few missing values (NAs) in our `LoanAmount`, `Loan_Amount_Term` and `Credit_History` features. Also, we noticed that there were a lot of blank values, which needed to be recoded to NAs. Therefore, we used the `naniar` package below to handle this:

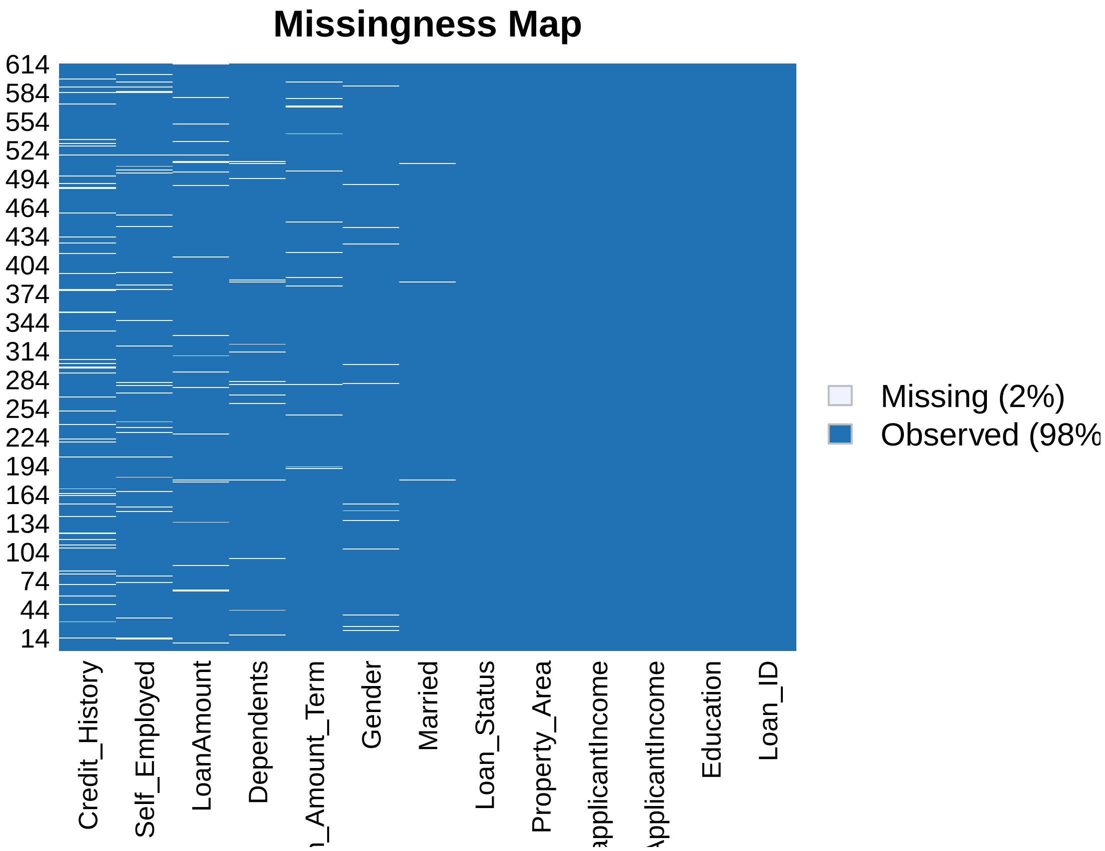
```
loan <- loan %>% replace_with_na_all(condition = ~. == "")
```

```
loan %>%
  summarise_all(funs(sum(is.na(.)))) %>%
  pivot_longer(cols = 1:13, names_to = 'columns', values_to = 'NA_count') %>%
  arrange(desc(NA_count)) %>%
  ggplot(aes(y = columns, x = NA_count)) + geom_col(fill = 'deepskyblue4') +
  geom_label(aes(label = NA_count)) +
  theme_minimal() +
  labs(title = 'Count of missing values in penguins dataset') +
  theme(plot.title = element_text(hjust = 0.5))
```



This can be further visualized by the “Missingness Map” below:

```
missmap(loan)
```

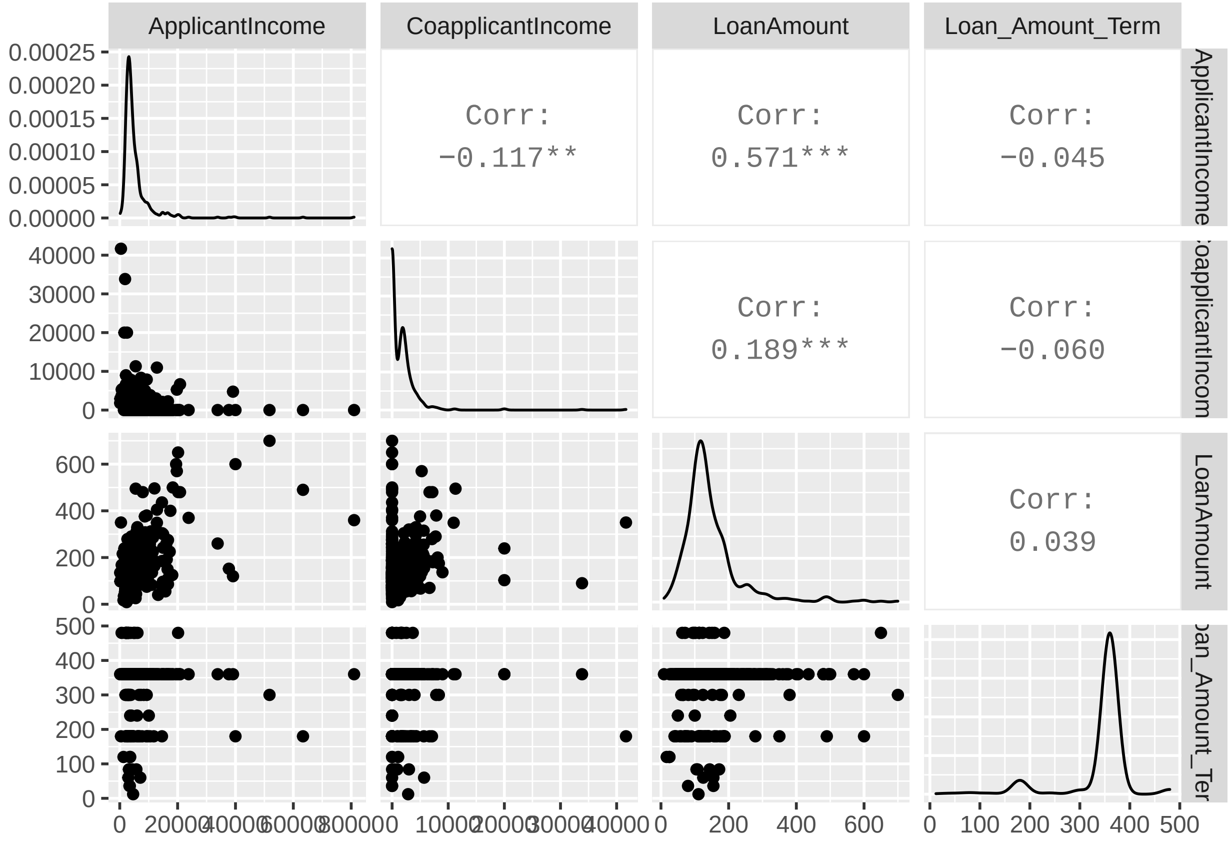


Before progressing, we thought it would be helpful to conduct some transformations on this data, as well as account for our N/As that we’ve discovered. To do this, we used the `factor` function on each feature:

```
loan$Loan_Status<-factor(loan$Loan_Status)


loan<-loan %>%
           mutate(Gender = factor(Gender),
                  Married = factor(Married),
                  Dependents=factor(Dependents),
                  Education=factor(Education),
                  Self_Employed=factor(Self_Employed),
                  Property_Area=factor(Property_Area),
                  Loan_Status=factor(Loan_Status))

summary(loan)
```

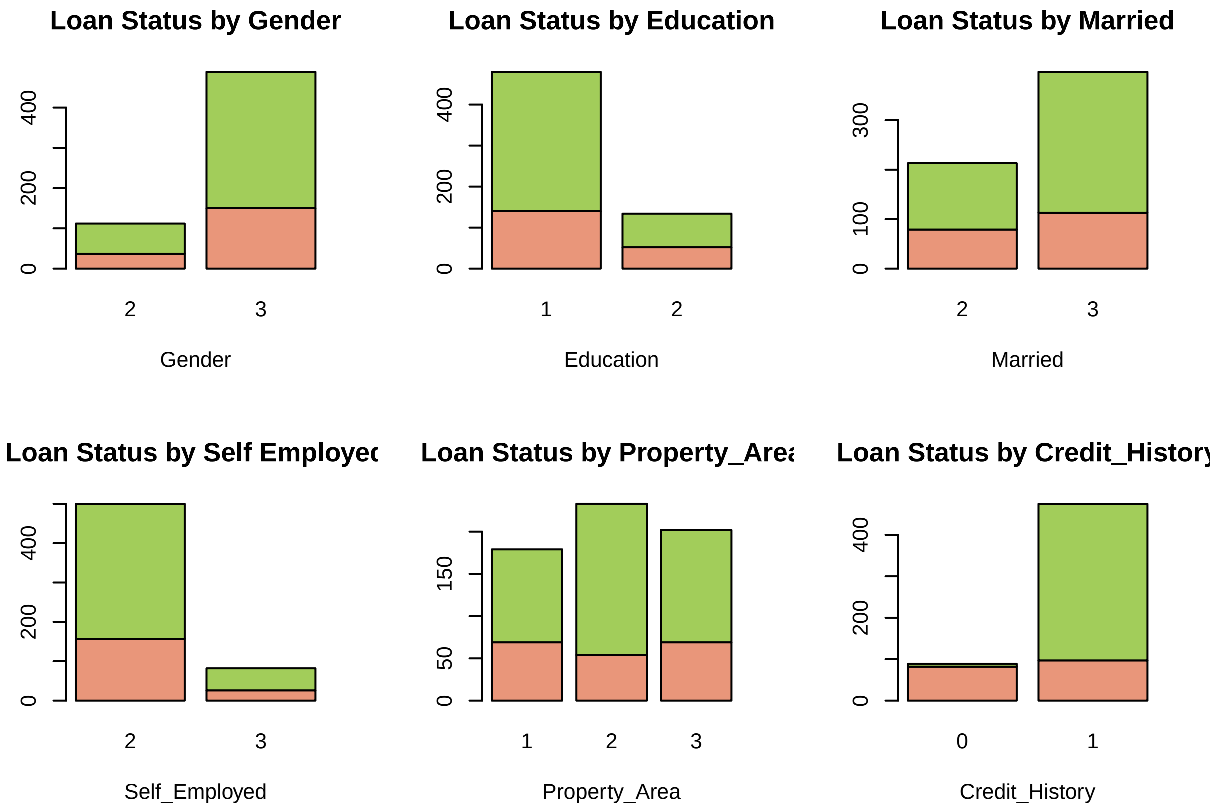
```
##     Loan_ID       Gender     Married     Dependents Education Self_Employed
##  Min.   :  1.0   2    :112   2    :213   2    :345   1:480     2    :500
##  1st Qu.:154.2   3    :489   3    :398   3    :102   2:134     3    : 82
##  Median :307.5   NA's: 13    NA's:   3   4    :101             NA's: 32
##  Mean   :307.5                         5   : 51
##  3rd Qu.:460.8                         NA's: 15
##  Max.   :614.0
##
##  ApplicantIncome CoapplicantIncome   LoanAmount    Loan_Amount_Term
##  Min.   :  150   Min.   :    0     Min.   :  9.0   Min.   : 12
##  1st Qu.: 2878   1st Qu.:    0     1st Qu.:100.0   1st Qu.:360
##  Median : 3812   Median : 1188     Median :128.0   Median :360
##  Mean   : 5403   Mean   : 1621     Mean   :146.4   Mean   :342
##  3rd Qu.: 5795   3rd Qu.: 2297     3rd Qu.:168.0   3rd Qu.:360
##  Max.   :81000   Max.   :41667     Max.   :700.0   Max.   :480
##                                    NA's   :22      NA's   :14
##  Credit_History   Property_Area Loan_Status
##  Min.   :0.0000   1:179         1:192
##  1st Qu.:1.0000   2:233         2:422
##  Median :1.0000   3:202
##  Mean   :0.8422
##  3rd Qu.:1.0000
##  Max.   :1.0000
##  NA's   :50
```

Next, we'll want to take a look at the distributions of the continuous features, which appear to be `ApplicantIncome`, `CoapplicantIncome`, `LoanAmount`, and `Loan_Amount_Term`:



We can see that `ApplicantIncome`, `CoapplicantIncome` and `LoanAmount` are highly right skewed, with long right tails. Conversely, it looks like `Loan_Amount_Term` is highly left skewed, with a long left tail.

Additionally, we can also take a look at the categorical variables, broken out by our target variable of whether or not a loan was issued to the applicant. From the green bars below, we can see the total amount of loan approvals, relative to the red bars below documenting the loan denials.



From the categorical variable splits based on loan approval status, we can see a few interesting things:

- The number of loans applied for by males was significantly higher than loans applied for by females in this dataset.
- The same phenomenon is true for graduate students, where a much higher number of loan applications were coming from graduate students relative to undergraduate students.
- Whether or not the applicant was self employed or married also showed a pretty large class imbalance, where those that identified as not self employed and those that identified as married had higher proportions of applicants in this dataset than those that did not identify by those two characteristics.
- There was a pretty even split in the number of applicants based on property area, where someone identifying that they live in rural, semiurban, or urban settings were pretty evenly distributed.
- We can see that applicants that did not pass the credit history criteria were almost always likely to be denied a loan.

These factors will be interesting to examine and discuss as we look to build our decision tree model and random forest models, since these class imbalances could affect model performance and interpretation.

Imputing our data to reconcile missing values (NAs) and adjust transformations

Since there were a large amount of missing values that we identified above, we can use KNN() to help with our imputations. With the kNN method, a categorical missing value is imputed with the majority among its k nearest neighbors, and the average value (mean) of the k nearest neighbors is regarded as the prediction for a numerical missing value.

```
loan<-kNN(loan)%>%
          subset(select = Loan_ID:Loan_Status)
```

Now, we can double check that this worked correctly by running syntax to find any missing values:

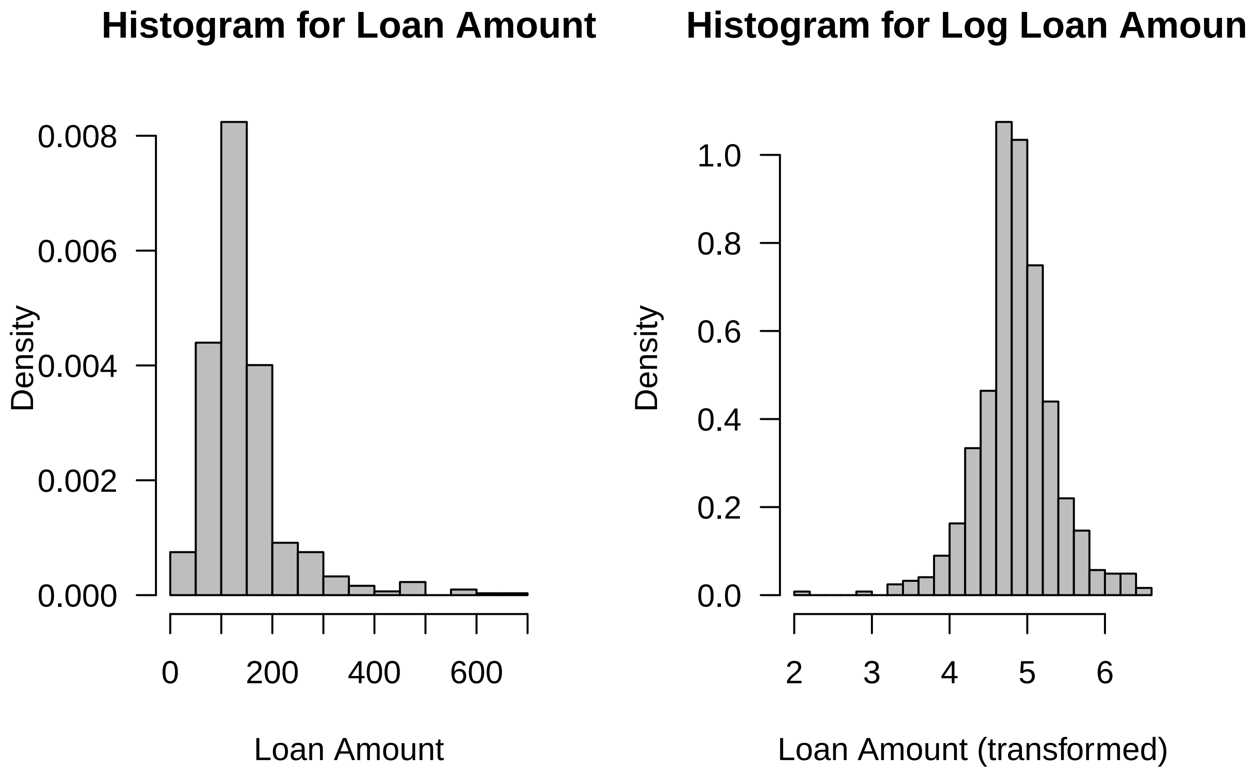
```
sapply(loan, function(x) sum(is.na(x)))
```

```
##           Loan_ID            Gender           Married        Dependents
##                 0                 0                 0                 0
##         Education     Self_Employed   ApplicantIncome CoapplicantIncome
##                 0                 0                 0                 0
##        LoanAmount  Loan_Amount_Term    Credit_History     Property_Area
##                 0                 0                 0                 0
##       Loan_Status
##                 0
```

Fortunately, we can see that there are now no missing data points!

Next, we'll have to work through a few transformations for our highly skewed continuous data. For our `LoanAmount` feature, we can conduct a log transformation:

```
loan$LogLoanAmount <- log(loan$LoanAmount)
loan$LogLoan_Amount_Term <- log(loan$Loan_Amount_Term)
```

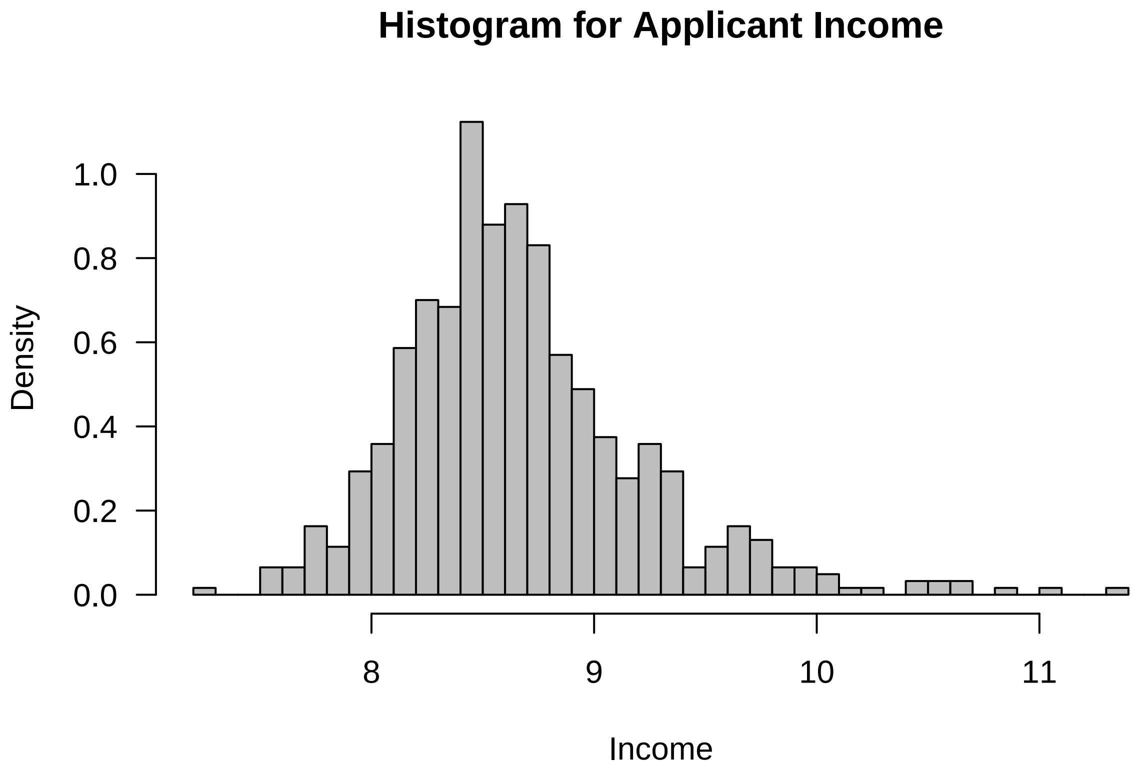

Histogram for Loan Amount

Histogram for Log Loan Amoun

Next, since there is a lot of relateability between the `ApplicantIncome` variable and `CoapplicantIncome` variable, we thought it would be best to combine these two variables to create an `Income` variable, and then perform a log transformation to obtain the best results.

```
loan$Income <- loan$ApplicantIncome + loan$CoapplicantIncome
loan$ApplicantIncome <- NULL
loan$CoapplicantIncome <- NULL
loan$LogIncome <- log(loan$Income)
```

When checking the new distribution of our `Income` variable, we can see that it is much more normally distributed:



With the exploratory data analysis behind us, and our data tidy'd, we are now ready to run our decision tree!

Splitting loan dataset into training and testing datasets

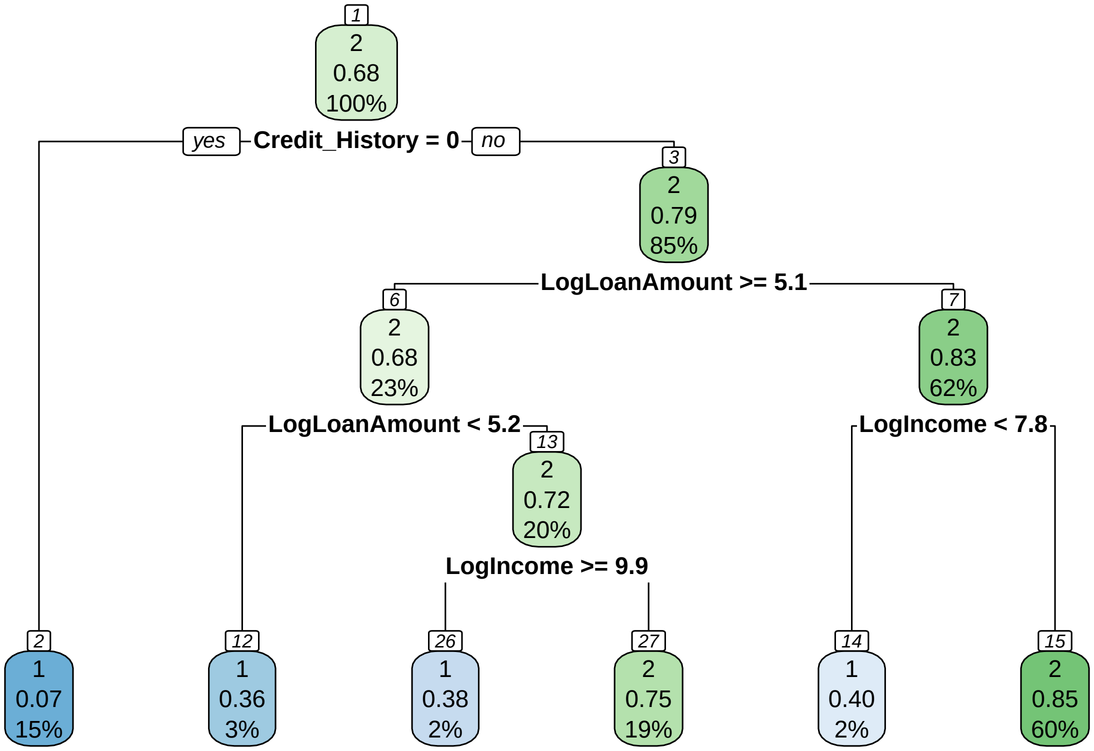
Similar to our KNN model process above, we'll determine an 80/20 split for our testing and training datasets:

```
set.seed(123)
train_sample<-sample(1:nrow(loan),size = floor(0.80*nrow(loan)))
train_loan<-loan[train_sample,]
test_loan<-loan[-train_sample,]
```

Splitting loan dataset into training and testing datasets

With the data split, we can now run our first decision tree:

```
tree1 <- rpart(Loan_Status~Gender+Married+Dependents+Education+Self_Employed+LogIncome+LogLoanAmount+Loa
rpart.plot(tree1,nn=TRUE)
```



```r
summary(tree1)
```

```
## Call:
## rpart(formula = Loan_Status ~ Gender + Married + Dependents +
##     Education + Self_Employed + LogIncome + LogLoanAmount + Loan_Amount_Term +
##     Credit_History + Property_Area, data = train_loan)
##   n= 491
##
##           CP nsplit rel error    xerror       xstd
## 1 0.40384615      0 1.0000000 1.0000000 0.06613317
## 2 0.01282051      1 0.5961538 0.5961538 0.05565673
## 3 0.01000000      5 0.5448718 0.6794872 0.05844104
##
## Variable importance
## Credit_History      LogIncome  LogLoanAmount
##             80             10              9
##
## Node number 1: 491 observations,    complexity param=0.4038462
##   predicted class=2  expected loss=0.3177189  P(node) =1
##     class counts:   156   335
##    probabilities: 0.318 0.682
##   left son=2 (73 obs) right son=3 (418 obs)
##   Primary splits:
##       Credit_History < 0.5      to the left,  improve=64.609250, (0 missing)
##       LogIncome      < 7.77862  to the left,  improve= 3.774721, (0 missing)
##       LogLoanAmount  < 5.093731 to the right, improve= 3.169404, (0 missing)
##       Property_Area  splits as  LRL,          improve= 2.078251, (0 missing)
##       Dependents     splits as  RLRL,         improve= 1.252302, (0 missing)
##
## Node number 2: 73 observations
##   predicted class=1  expected loss=0.06849315  P(node) =0.1486762
##     class counts:    68     5
##    probabilities: 0.932 0.068
##
## Node number 3: 418 observations,    complexity param=0.01282051
##   predicted class=2  expected loss=0.2105263  P(node) =0.8513238
##     class counts:    88   330
##    probabilities: 0.211 0.789
##   left son=6 (114 obs) right son=7 (304 obs)
##   Primary splits:
##       LogLoanAmount    < 5.093731 to the right, improve=4.0767540, (0 missing)
##       LogIncome        < 7.806656 to the left,  improve=3.1081530, (0 missing)
##       Property_Area    splits as  LRL,          improve=1.3899880, (0 missing)
##       Married          splits as  LR,           improve=0.8900091, (0 missing)
##       Loan_Amount_Term < 420      to the right, improve=0.7356037, (0 missing)
##   Surrogate splits:
##       LogIncome < 8.886468 to the right, agree=0.859, adj=0.482, (0 split)
##
## Node number 6: 114 observations,    complexity param=0.01282051
##   predicted class=2  expected loss=0.3245614  P(node) =0.2321792
##     class counts:    37    77
##    probabilities: 0.325 0.675
##   left son=12 (14 obs) right son=13 (100 obs)
##   Primary splits:
##       LogLoanAmount < 5.159039 to the left,  improve=3.2338850, (0 missing)
##       LogIncome     < 8.747586 to the left,  improve=2.4820750, (0 missing)
##       Married       splits as  LR,           improve=1.1905700, (0 missing)
##       Self_Employed splits as  LR,           improve=0.6618637, (0 missing)
##       Dependents    splits as  LLRR,         improve=0.5850202, (0 missing)
##   Surrogate splits:
##       LogIncome < 8.359914 to the left,  agree=0.895, adj=0.143, (0 split)
##
## Node number 7: 304 observations,    complexity param=0.01282051
##   predicted class=2  expected loss=0.1677632  P(node) =0.6191446
##     class counts:    51   253
##    probabilities: 0.168 0.832
##   left son=14 (10 obs) right son=15 (294 obs)
##   Primary splits:
##       LogIncome        < 7.806656 to the left,  improve=3.8636680, (0 missing)
##       Property_Area    splits as  LRL,          improve=1.7836780, (0 missing)
##       LogLoanAmount    < 4.740565 to the left,  improve=1.4148430, (0 missing)
##       Loan_Amount_Term < 420      to the right, improve=0.7057255, (0 missing)
##       Married          splits as  LR,           improve=0.6873981, (0 missing)
##
## Node number 12: 14 observations
##   predicted class=1  expected loss=0.3571429  P(node) =0.02851324
##     class counts:     9     5
##    probabilities: 0.643 0.357
##
```

```
## Node number 13: 100 observations,    complexity param=0.01282051
##   predicted class=2  expected loss=0.28  P(node) =0.203666
##     class counts:    28    72
##    probabilities: 0.280 0.720
##   left son=26 (8 obs) right son=27 (92 obs)
##   Primary splits:
##       LogIncome     < 9.913412 to the right, improve=2.0700000, (0 missing)
##       Married       splits as  LR,           improve=1.4385030, (0 missing)
##       LogLoanAmount < 5.470148 to the right, improve=1.0244230, (0 missing)
##       Dependents    splits as  LLRR,         improve=0.4538761, (0 missing)
##       Education     splits as  RL,           improve=0.3322888, (0 missing)
##
## Node number 14: 10 observations
##   predicted class=1  expected loss=0.4  P(node) =0.0203666
##     class counts:     6     4
##    probabilities: 0.600 0.400
##
## Node number 15: 294 observations
##   predicted class=2  expected loss=0.1530612  P(node) =0.598778
##     class counts:    45   249
##    probabilities: 0.153 0.847
##
## Node number 26: 8 observations
##   predicted class=1  expected loss=0.375  P(node) =0.01629328
##     class counts:     5     3
##    probabilities: 0.625 0.375
##
## Node number 27: 92 observations
##   predicted class=2  expected loss=0.25  P(node) =0.1873727
##     class counts:    23    69
##    probabilities: 0.250 0.750
```

From our decision tree summary and plot, we can see that a few variables, such as `Credit_History`, `Income` and `LoanAmount`, were important features in this classification method. We can use this decision tree to make predictions on our holdout test set.

```
loanPre<-predict(tree1,test_loan,type="class")
table(loanPre,test_loan$Loan_Status,dnn=c("Actual", "Predicted"))
```

```
##       Predicted
## Actual  1  2
##      1 20  9
##      2 16 78
```

```
accuracy_tree1<-mean(loanPre==test_loan$Loan_Status)
accuracy_tree1
```

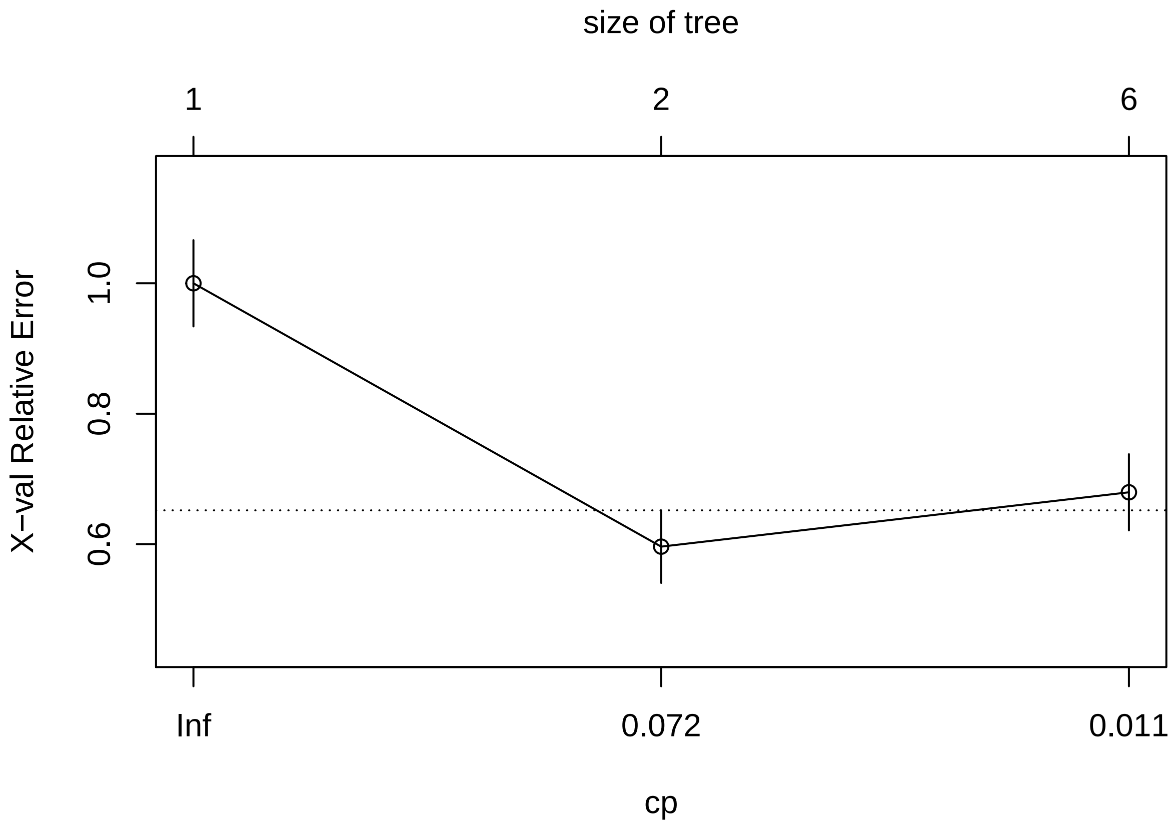
```
## [1] 0.796748
```

As we can see from our predictions and the ensuing confusion matrix, we were able to obtain about 79% accuracy using this decision tree. To achieve higher accuracy, we'll use the `prune()` function in the `rpart` package to examine a predicted optimal tree size.

```r
dtree.cm_train <- confusionMatrix(loanPre, test_loan$Loan_Status)
dtree.cm_train
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction  1  2
##          1 20  9
##          2 16 78
## 
##                Accuracy : 0.7967
##                  95% CI : (0.7148, 0.8639)
##     No Information Rate : 0.7073
##     P-Value [Acc > NIR] : 0.01632
## 
##                   Kappa : 0.4794
## 
##  Mcnemar's Test P-Value : 0.23014
## 
##             Sensitivity : 0.5556
##             Specificity : 0.8966
##          Pos Pred Value : 0.6897
##          Neg Pred Value : 0.8298
##              Prevalence : 0.2927
##          Detection Rate : 0.1626
##    Detection Prevalence : 0.2358
##       Balanced Accuracy : 0.7261
## 
##        'Positive' Class : 1
## 
```

```r
plotcp(tree1)
```



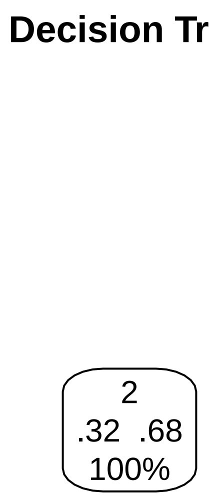
By plotting the cross-validated error against the complexity parameter, we can see that the relationship between the yields the best optimal outcome for our tree at a tree size of 2.

```
tree1$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.40384615      0 1.0000000 1.0000000 0.06613317
## 2 0.01282051      1 0.5961538 0.5961538 0.05565673
## 3 0.01000000      5 0.5448718 0.6794872 0.05844104
```

Although this is the case, when using this cp value in our `prune()` function, we still obtain accuracy ratings of around 79%. In order to test difference complexity parameters, we also decided to use our cp value of 0.41025641, which is a tree size of 1. When we use this tree size, and prune our initial tree accordingly, we can see the resulting decision tree below:

```
tree1.pruned <- prune(tree1, cp=0.41025641)
prp(tree1.pruned, type = 2, extra = 104, fallen.leaves = TRUE, main = "Decision Tree")
```



Interestingly, from this pruning process, the `Credit_History` feature alone seems to do a fairly good job of classifying applicants into the approval vs. disapproval status. The accuracy is about 83% on our test set, which is slightly higher than our initial tree.

However, it's important to think critically about whether or not this pruned tree would perform better on another dataset, and if it is applicable for real-life events.

```
tree1.pruned.pred <- predict(tree1.pruned, test_loan, type = "class" )
tree1.pruned.perf <- table(test_loan$Loan_Status, tree1.pruned.pred, dnn = c("Actual", "Predicted"))

tree1.pruned.perf
```

```
##       Predicted
## Actual  1  2
##      1  0 36
##      2  0 87
```

```
accuracy_tree2<-mean(tree1.pruned.pred==test_loan$Loan_Status)
accuracy_tree2
```

```
## [1] 0.7073171
```

```
dtree.cm_pruned <- confusionMatrix(tree1.pruned.pred, test_loan$Loan_Status)
dtree.cm_pruned
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  0  0
##          2 36 87
##
##                Accuracy : 0.7073
##                  95% CI : (0.6185, 0.7859)
##     No Information Rate : 0.7073
##     P-Value [Acc > NIR] : 0.5448
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 5.433e-09
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7073
##              Prevalence : 0.2927
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```

In the end, we'd likely want to go with a more robust decision tree, than just one feature, but not getting too large in size – which is why our initial tree seems to be a good balance between these two phenomenon.

Part 3: Random Forests on loan approval dataset

Using the same dataset on Loan Approval Status, please use Random Forests to predict on loan approval status. Again, please be sure to walk us through the steps you took to get to your final model. (50 points)

```
fit.forest <- randomForest(Loan_Status~Gender+Married+Dependents+Education+Self_Employed+LogIncome+LogLo

fit.forest
```

```
##
## Call:
##  randomForest(formula = Loan_Status ~ Gender + Married + Dependents +      Education + Self_Employed
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
```

```
##         OOB estimate of  error rate: 20.57%
## Confusion matrix:
##    1   2 class.error
## 1 75  81  0.51923077
## 2 20 315  0.05970149
```

```
importance(fit.forest)
```

```
##                  MeanDecreaseGini
## Gender                   4.410974
## Married                  5.061851
## Dependents              10.026353
## Education                3.998800
## Self_Employed            3.725412
## LogIncome               44.954358
## LogLoanAmount           40.013028
## Loan_Amount_Term         8.895708
## Credit_History          59.546064
## Property_Area            9.584900
```

```
forest.pred <- predict(fit.forest, newdata = test_loan)
forest.cm <- table(test_loan$Loan_Status, forest.pred,
                     dnn=c("Actual", "Predicted"))
forest.cm
```

```
##       Predicted
## Actual  1  2
##      1 20 16
##      2  4 83
```

```
accuracy_forest1 <- mean(forest.pred==test_loan$Loan_Status)
accuracy_forest1
```

```
## [1] 0.8373984
```

Here, we notice slight improvements on both samples where accuracy for the training sample is 83.74%.

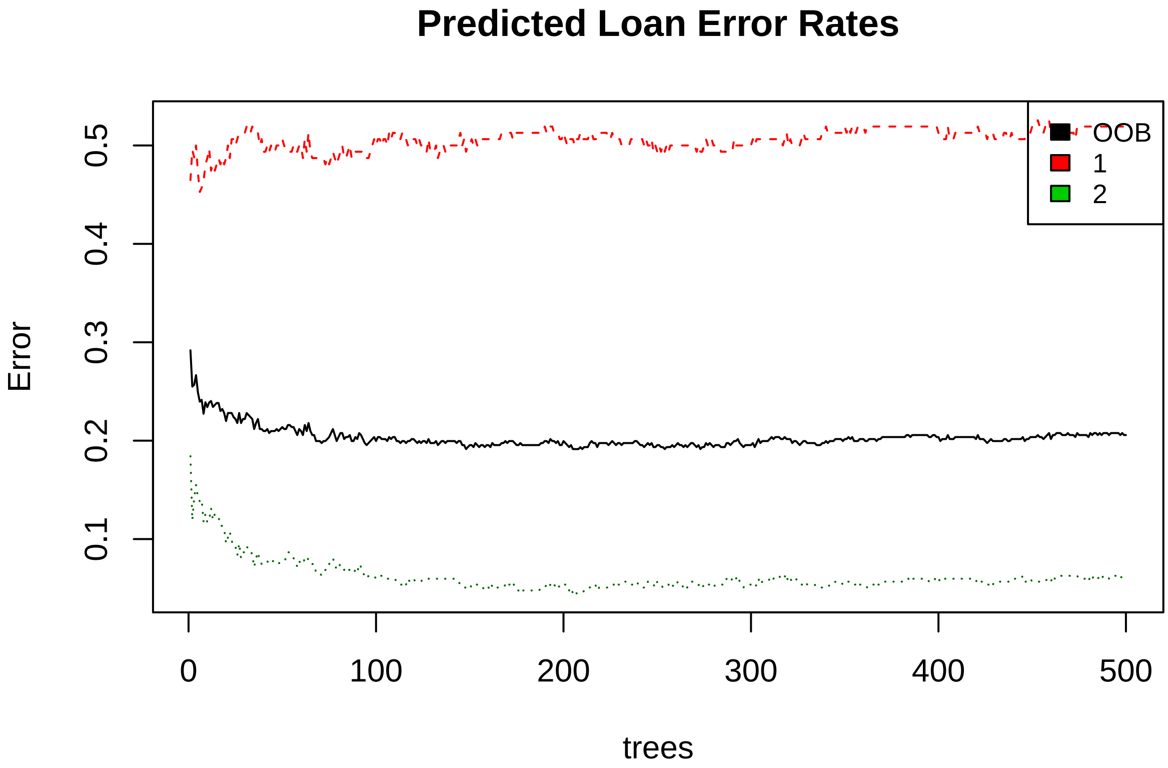
```
forest.cm_train <- confusionMatrix(forest.pred, test_loan$Loan_Status)
forest.cm_train
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 20  4
##          2 16 83
##
##                Accuracy : 0.8374
##                  95% CI : (0.7601, 0.8978)
##     No Information Rate : 0.7073
##     P-Value [Acc > NIR] : 0.0006253
```

```
##
##                   Kappa : 0.5648
##
##  Mcnemar's Test P-Value : 0.0139063
##
##             Sensitivity : 0.5556
##             Specificity : 0.9540
##          Pos Pred Value : 0.8333
##          Neg Pred Value : 0.8384
##              Prevalence : 0.2927
##          Detection Rate : 0.1626
##    Detection Prevalence : 0.1951
##       Balanced Accuracy : 0.7548
##
##        'Positive' Class : 1
##
```

Next we want to see if we have generated enough trees so that the Out Of Bag (OOB Error) error rates are minimum. From the below we see that the OOB error rate is decreasing with 1-20 trees, and rate stabilizes that at around 100 trees.

```
plot(fit.forest, col = c("black", "red", "dark green"), main = "Predicted Loan Error Rates")
legend("topright", colnames(fit.forest$err.rate),col = 1:6, cex = 0.8, fill = 1:6)
```

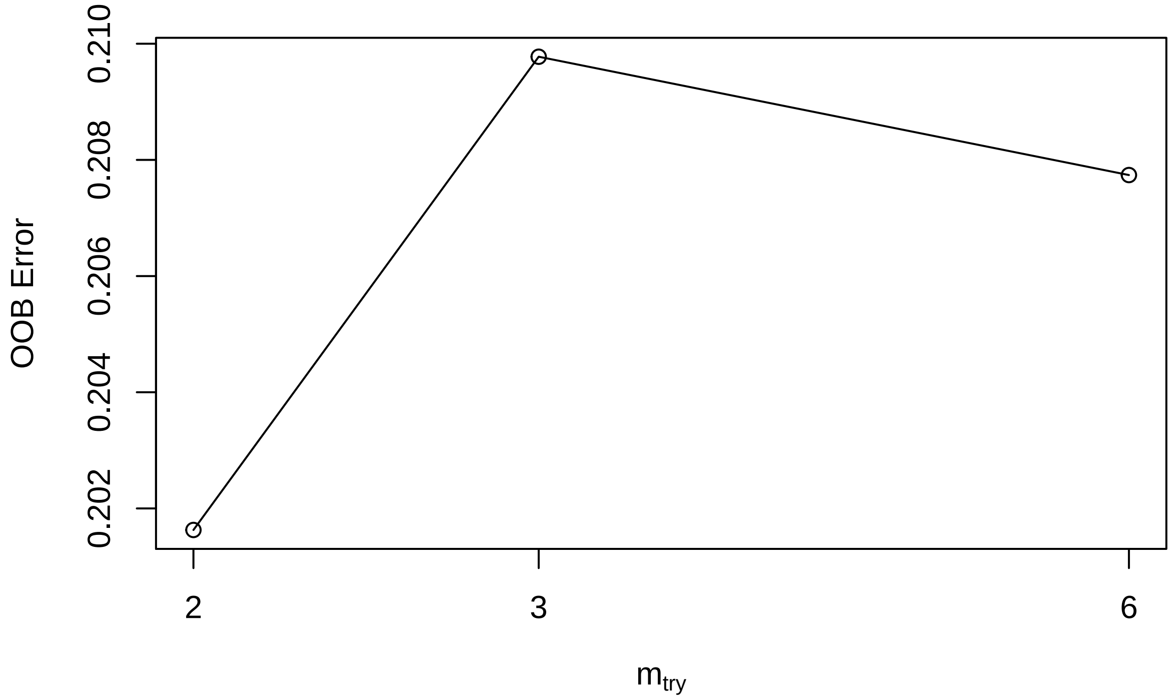


In order to see whether the Random forest model can be improved, We will run the same model but this time we will use tuneRF function. X request for the variable of the dataset, but the target value, while Y is the target value. Stepfactor increases or decreases the Mtry at each iteration. The plot is whether to plot the OOB error as a function of Mtry. Ntreetry is the number of the number of trees used at the tuning step. Trace is whether to print the progress of the search and Improve the (relative) improvement in OOB error must be by this much for the search to continue.

The values were assigned randomly initially, and they have tweaked until the optimal was found.

```
tree_var <- tuneRF(x = subset(train_loan, select = -Loan_Status), y=train_loan$Loan_Status, ntreeTry = 
```

```
## mtry = 3  OOB error = 20.98%
## Searching left ...
## mtry = 2     OOB error = 20.16%
## 0.03883495 0.05
## Searching right ...
## mtry = 6     OOB error = 20.77%
## 0.009708738 0.05
```



The Random Forest model is re-run with the new parameter Ntree= 1000 and Mtry=3

```
val_opt <- tree_var [,"mtry"][which.min(tree_var [,"OOBError"])]
fit.forest2 <- randomForest(Loan_Status ~Gender+Married+Dependents+Education+Self_Employed+LogIncome+Lo

fit.forest2
```

```
##
## Call:
```

```
##  randomForest(formula = Loan_Status ~ Gender + Married + Dependents +      Education + Self_Employed
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 19.96%
## Confusion matrix:
##    1   2 class.error
## 1 68  88  0.56410256
## 2 10 325  0.02985075
```

The results of the trained Random Forest model are an out of bag error of 21.18 %, which is higher than the original model 19.35%, Although, it still a good result it has got worse with the tune.

```
forest.pred2 <- predict(fit.forest2, newdata = test_loan)
forest.cm2 <- table(test_loan$Loan_Status, forest.pred2,
                     dnn=c("Actual", "Predicted"))
forest.cm2
```

```
##       Predicted
## Actual  1  2
##      1 18 18
##      2  2 85
```

```
accuracy_forest2 <- mean(forest.pred2==test_loan$Loan_Status)
accuracy_forest2
```

```
## [1] 0.8373984
```

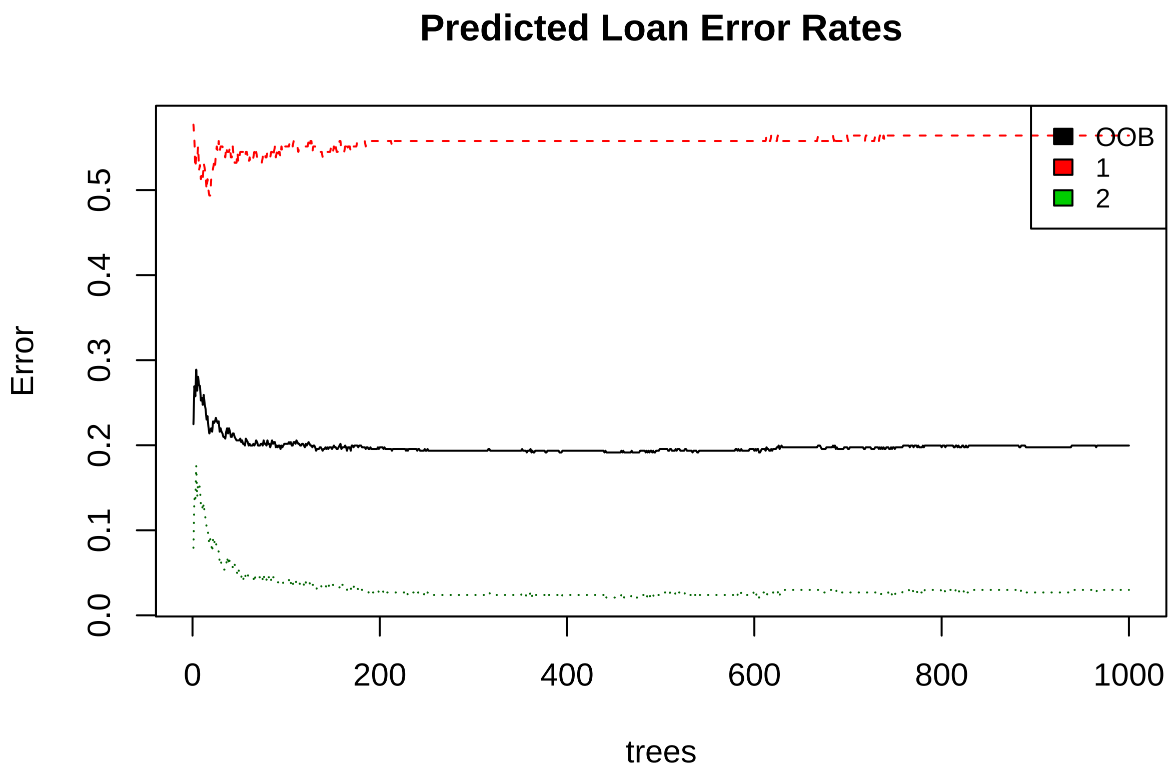
```
forest.cm_tune <- confusionMatrix(forest.pred2, test_loan$Loan_Status)
forest.cm_tune
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 18  2
##          2 18 85
##
##                Accuracy : 0.8374
##                  95% CI : (0.7601, 0.8978)
##     No Information Rate : 0.7073
##     P-Value [Acc > NIR] : 0.0006253
##
##                   Kappa : 0.5485
##
##  Mcnemar's Test P-Value : 0.0007962
##
##             Sensitivity : 0.5000
##             Specificity : 0.9770
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.8252
##               Prevalence : 0.2927
##           Detection Rate : 0.1463
##     Detection Prevalence : 0.1626
##        Balanced Accuracy : 0.7385
##
##         'Positive' Class : 1
##
```

Here, we notice there has the same accuracy as previous one for the training sample is 82.11%. The tuned model has performed worse than the test data set than the original Random Forest. The accuracy is slightly decreased, and the 95 % CI has decreased a bit too.

Next we want to see if we have generated enough trees so that the Out Of Bag (OOB Error) error rates are minimum. From the below we see that the OOB error rate is decreasing with 1-20 trees, and rate stabilizes that at around 20 trees.

```r
plot(fit.forest2, col = c("black", "red", "dark green"), main = "Predicted Loan Error Rates")
legend("topright", colnames(fit.forest2$err.rate),col = 1:6, cex = 0.8, fill = 1:6)
```



Part 4: Gradient Boosting

Using the Loan Approval Status data, please use Gradient Boosting to predict on the loan approval status. Please use whatever boosting approach you deem appropriate;but please be sure to walk us through your steps. (50 points)

```
set.seed(1)
boost<-gbm(Loan_Status~., data=train_loan[,-1],distribution = "gaussian",n.trees = 1000,cv.folds = 3)
```

gbm uses a default number of trees of 100, which is rarely sufficient. Consequently, I crank it up to 1,000 trees. The default depth of each tree (interaction.depth) is 1, which means we are ensembling a bunch of stumps. Lastly, I also include cv.folds to perform a 3 fold cross validation.

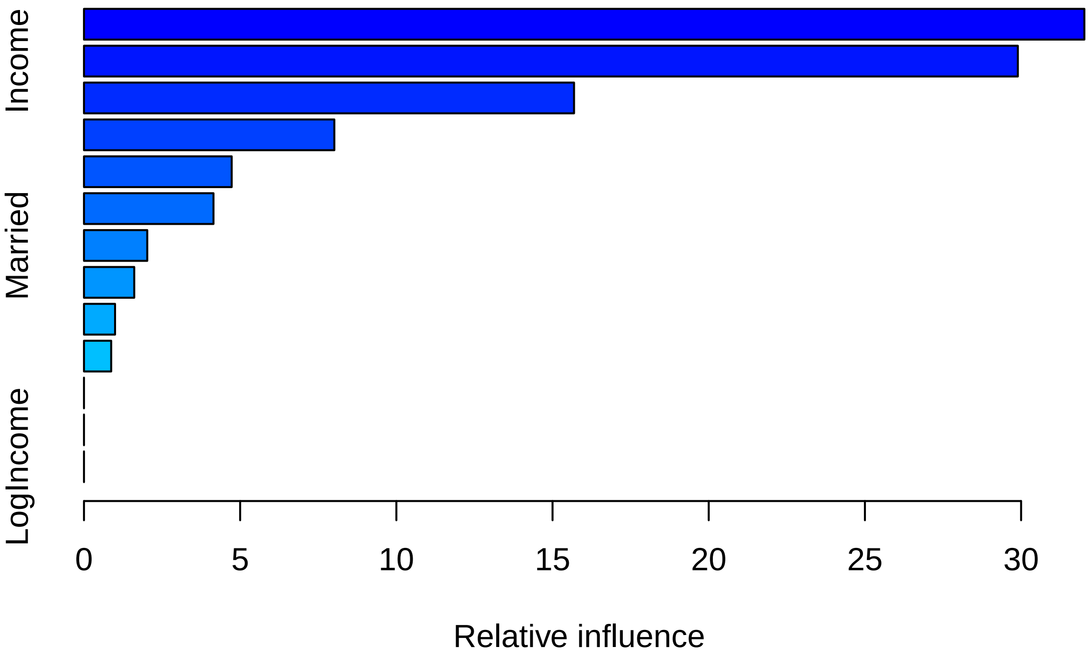
```
boost
```

```
## gbm(formula = Loan_Status ~ ., distribution = "gaussian", data = train_loan[,
##     -1], n.trees = 1000, cv.folds = 3)
## A gradient boosted model with gaussian loss function.
## 1000 iterations were performed.
## The best cross-validation iteration was 26.
## There were 13 predictors of which 5 had non-zero influence.
```

```
sqrt(min(boost$cv.error))
```

```
## [1] 0.3929355
```

```
summary(boost)
```



```
##                                         var    rel.inf
## LoanAmount                       LoanAmount 32.0250342
## Income                               Income 29.8962392
## Credit_History               Credit_History 15.6880081
## Dependents                       Dependents  8.0146185
## Loan_Amount_Term           Loan_Amount_Term  4.7291612
## Property_Area                 Property_Area  4.1460619
## Married                             Married  2.0261409
## Self_Employed                 Self_Employed  1.6090044
## Gender                               Gender  0.9947729
## Education                         Education  0.8709587
## LogLoanAmount                 LogLoanAmount  0.0000000
## LogLoan_Amount_Term LogLoan_Amount_Term  0.0000000
## LogIncome                         LogIncome  0.0000000
```

The best cross-validation iteration was 34 and the RMSE is 0.3913894 The summary of output creates a new data set with var, a factor variable with the variables in our model, and rel.inf, the relative influence each variable had on our model predictions.From the table, we can see LoanAmount, income and Credit_History is the top three most important vaiable for our model.

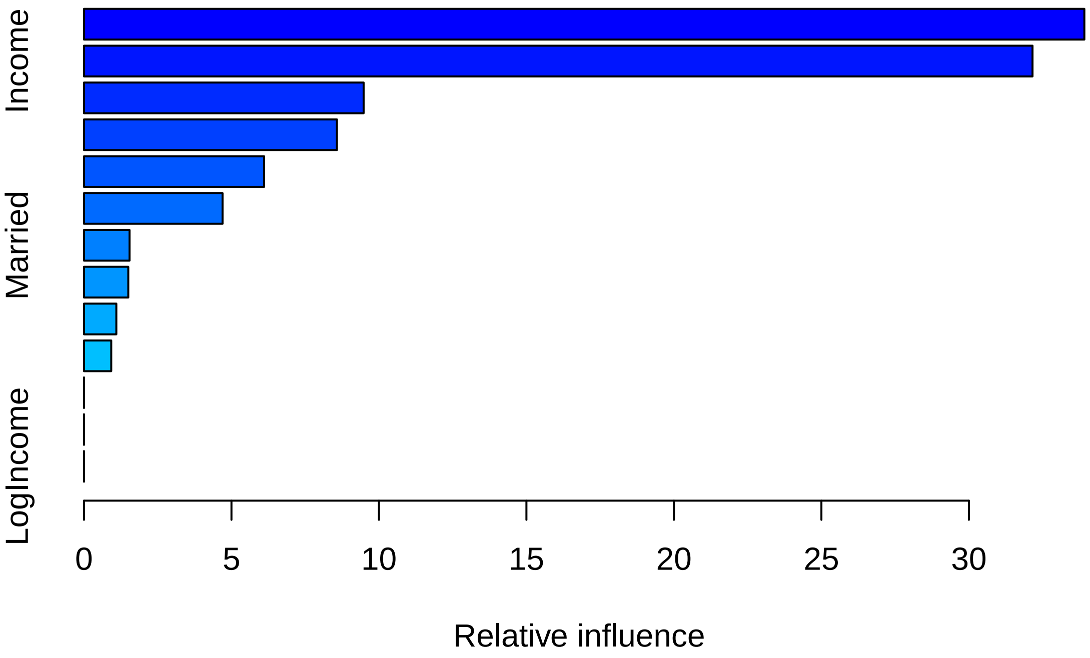
*** Tuning ***

We could tune parameters to see how the results change. Here, I increase the number of trees and also perform a 5 fold cross validation.

```
set.seed(1)
boost2<-gbm(Loan_Status~., data=train_loan[,-1],distribution = "gaussian",n.trees = 2000,cv.folds = 5)
sqrt(min(boost2$cv.error))
```

```
## [1] 0.3900931
```

```
summary(boost2)
```

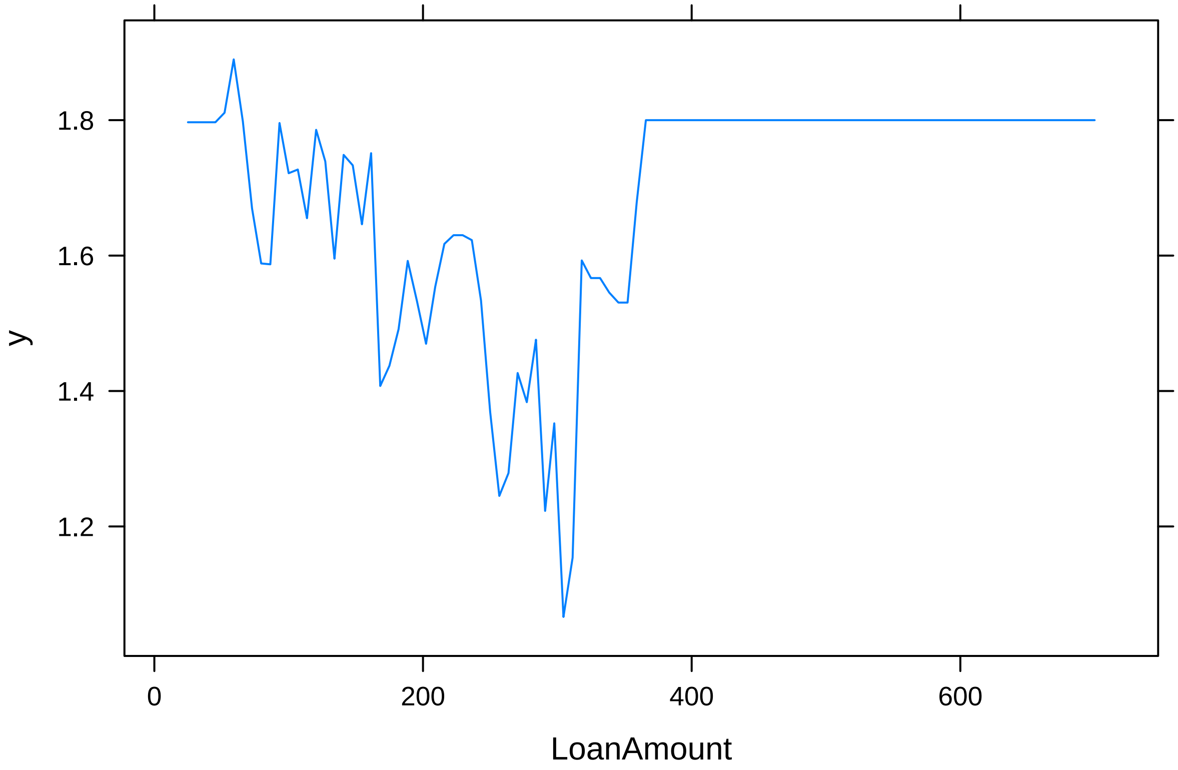


```
##                                     var    rel.inf
## LoanAmount                   LoanAmount 33.9146944
## Income                           Income 32.1569851
## Credit_History           Credit_History  9.4808038
## Dependents                   Dependents  8.5747007
## Loan_Amount_Term       Loan_Amount_Term  6.1069720
## Property_Area             Property_Area  4.6975610
## Married                         Married  1.5438725
## Self_Employed             Self_Employed  1.5027080
## Education                     Education  1.0973188
## Gender                           Gender  0.9243836
## LogLoanAmount             LogLoanAmount  0.0000000
## LogLoan_Amount_Term LogLoan_Amount_Term  0.0000000
## LogIncome                     LogIncome  0.0000000
```

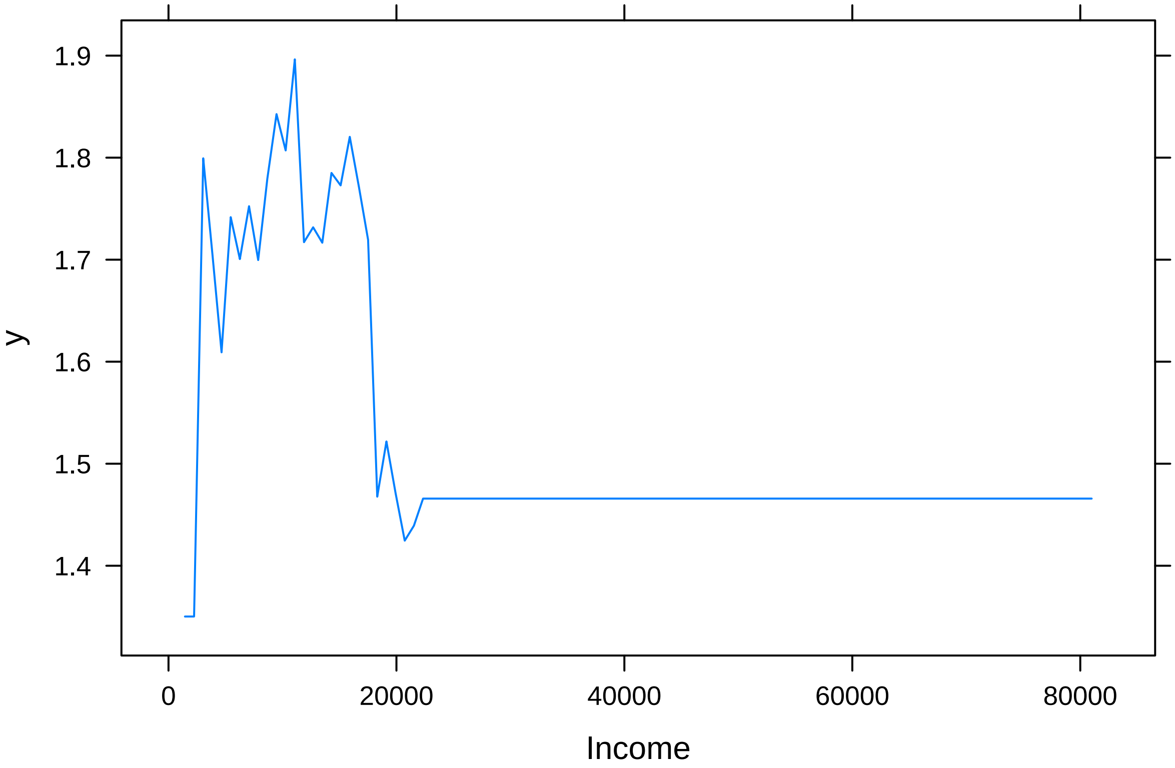
We can see the number of relative influence of each variable change and also the RMSE change to 0.3900092, lower than our previous model.

The partial Dependence Plots tell us the relationship and dependence of the variables X with the Response variable Y.

```
#partial dependence plots
plot(boost2,i="LoanAmount")
```

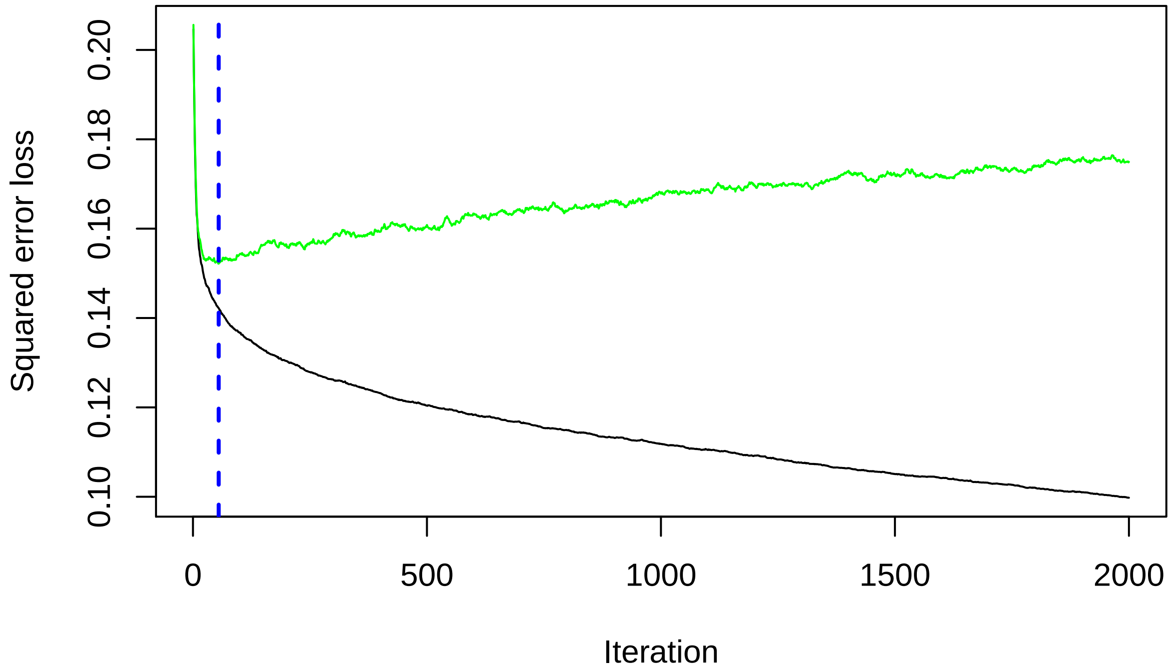


```
plot(boost2,i="Income")
```



The first plot shows loanAmount is negatively correlated with the response Loan_Status before 300K. The second plot indicate income is positively correlated with Loan_Status when it less than $20,000.

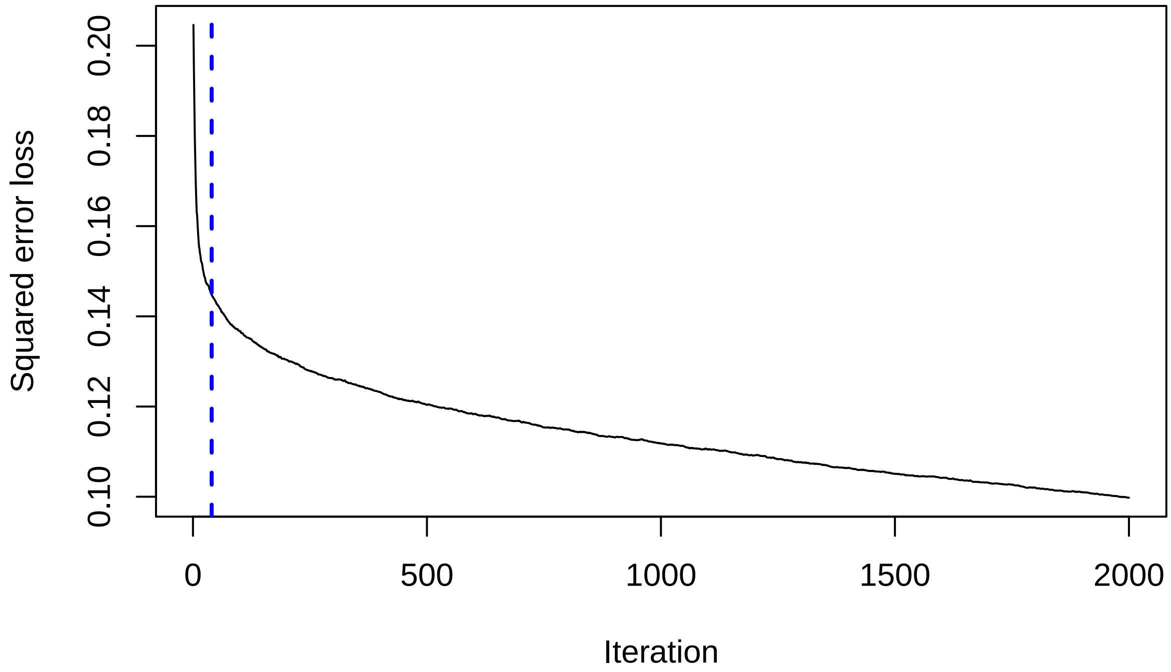
```r
# plot loss function as a result of n trees added to the ensemble
gbm.perf(boost2, method = "cv")
```



```
## [1] 55
```

```
gbm.perf(boost2,method="OOB")
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance is reas
```



```
## [1] 40
## attr(,"smoother")
## Call:
## loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,
##     length(x)/10), 50))
##
## Number of Observations: 2000
## Equivalent Number of Parameters: 39.99
## Residual Standard Error: 0.0003722
```

The plots indicating the optimum number of trees based on the technique we used. Green line indicates Error on test and the blue dotted line points the optimum number of iterations.We can observe that the beyond a certain a point (55 iterations for the "cv" method and 40 for the "OOB" method), the error on the test data appears to increase because of overfitting.

We use model 2 to forcast our data.According to the "cv" method, we choose 55 as the number of trees.

```
boostPre<-predict.gbm(boost2,test_loan, n.trees =55)
boostPre<-ifelse(boostPre<1.5,1,2)
boostPre<-factor(boostPre)
test_loan$Loan_Status<-factor(test_loan$Loan_Status)

boost.cm <- confusionMatrix(boostPre, test_loan$Loan_Status)
boost.cm
```

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

```
##
##           Reference
## Prediction  1  2
##          1 17  2
##          2 19 85
##
##                Accuracy : 0.8293
##                  95% CI : (0.7509, 0.8911)
##     No Information Rate : 0.7073
##     P-Value [Acc > NIR] : 0.0013303
##
##                   Kappa : 0.5214
##
##  Mcnemar's Test P-Value : 0.0004803
##
##             Sensitivity : 0.4722
##             Specificity : 0.9770
##          Pos Pred Value : 0.8947
##          Neg Pred Value : 0.8173
##              Prevalence : 0.2927
##          Detection Rate : 0.1382
##    Detection Prevalence : 0.1545
##       Balanced Accuracy : 0.7246
##
##        'Positive' Class : 1
##
```

According to the Confusion matrix, Our model accuracy is 0.8293.

Part 5: Model Performance

Model performance: please compare the models you settled on for problem # 2 – 4.Comment on their relative performance. Which one would you prefer the most? Why?(20 points)

```r
temp <- data.frame(dtree.cm_train$overall,
                   forest.cm_train$overall,
                   boost.cm$overall) %>%
  t() %>%
  data.frame() %>%
  dplyr::select(Accuracy) %>%
  mutate(`Classification Error Rate` = 1-Accuracy)
```

```r
eval <- data.frame(dtree.cm_train$byClass,
                   forest.cm_train$byClass,
                   boost.cm$byClass)
eval <- data.frame(t(eval)) %>%
  cbind(temp) %>%
  mutate(eval = c("Decision Tree", "Random Forest", "Gradient Boosting"))
```

```r
eval <- dplyr::select(eval, Accuracy, `Classification Error Rate`, Sensitivity, Specificity, Precision,
rownames(eval) = c("Decision Tree",  "Random Forest", "Gradient Boosting")
t_eval <- t(eval)
colnames(t_eval) <- rownames(eval)
rownames(t_eval) <- colnames(eval)
knitr::kable(t_eval)
```

	Decision Tree	Random Forest	Gradient Boosting
Accuracy	0.7967480	0.8373984	0.8292683
Classification Error Rate	0.2032520	0.1626016	0.1707317
Sensitivity	0.5555556	0.5555556	0.4722222
Specificity	0.8965517	0.9540230	0.9770115
Precision	0.6896552	0.8333333	0.8947368
Recall	0.5555556	0.5555556	0.4722222
F1	0.6153846	0.6666667	0.6181818