Robert Stephenson

Mr. Shelley

IS 412-01

1 April 2024

Algorithm to Help Lenders Mitigate Risk

Introduction

Businesses have always needed to leverage the latest technology to gain a competitive advantage. Data has been around for as long as humanity could write and keep track of things. There are ancient records of merchants keeping track of their inventories carved into stone tablets. While this is very basic data, data today has become much more detailed. As well as being more detailed and accessible, modern technology has allowed us to manipulate data in ways we previously couldn't. Machine learning and artificial intelligence are very hot technological topics of the time. While Chatgpt and artificial video and images may currently be taking the world by storm, Machine Learning and AI has been helping us to work with data to make better decisions. Almost every business uses data in one way or another, and some of them are making use of this technology to make their data more useful.

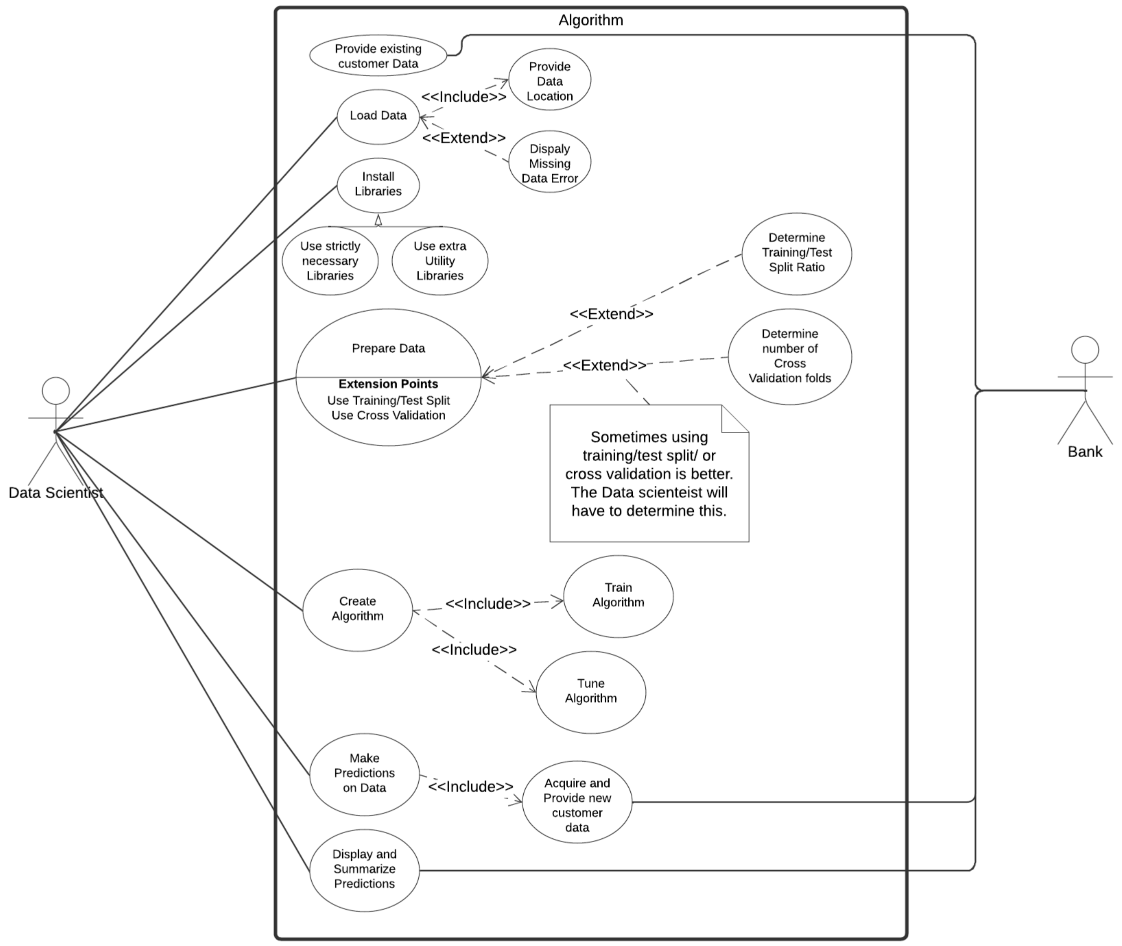
N.R. Bank Inc.

In this project, a fictional bank, N. R. Bank Inc., is trying to make use of some of their customer data. As with any bank or credit system, people are going to default on payments for one reason or another. Defaults are very costly for banks and N. R. Bank is currently feeling the effects. They have all this data, and they're hoping they can make use of it to solve this problem. Fortunately for N. R. Bank, we can do just that. We'll make use of algorithms with machine learning techniques to help them be able to identify low and high risk borrowers. This link can further help us understand exactly what data N.R. Bank is giving us: https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset.

ERD Diagrams

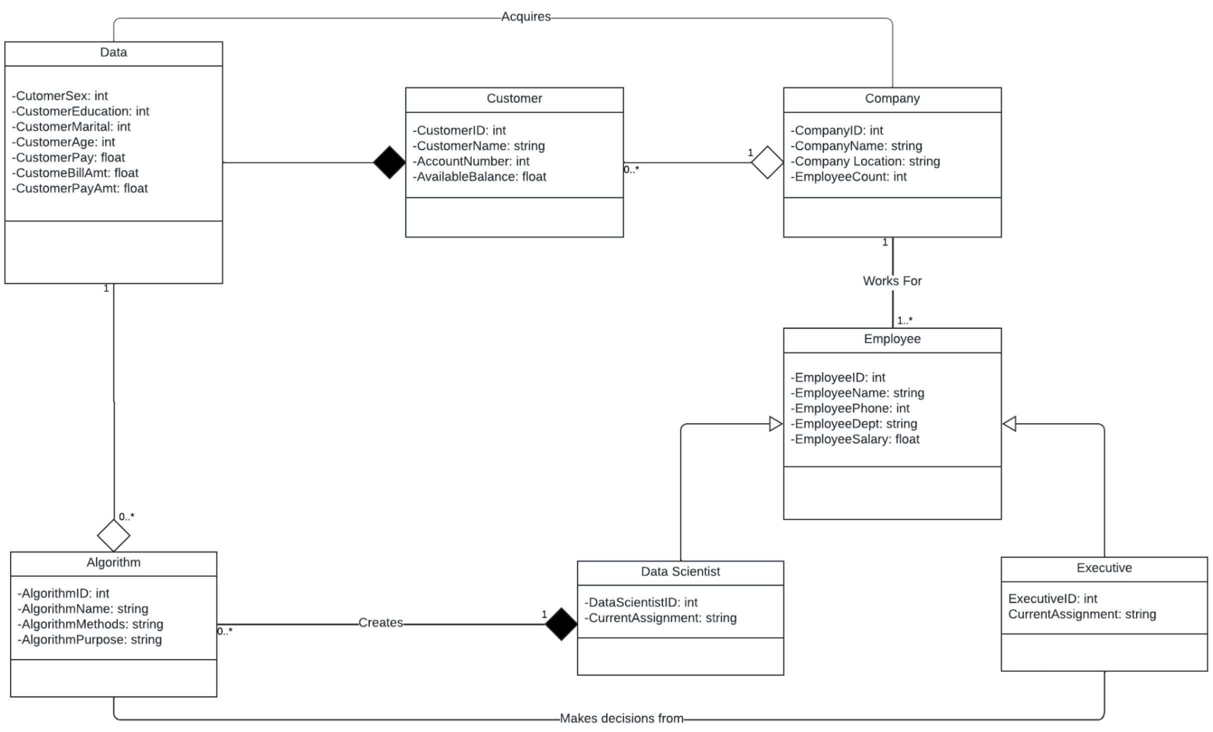
To get a better idea of how we're going to make this algorithm, we will make use of ERD diagrams so that N.R. Bank can be clued in on the inner workings. As we go through the diagrams, commentary will be added to explain what each diagram is showing.

Use Case Diagram


Algorithm
Provide existing customer Data
Provide Data Location
<<Include>>
Load Data
<<Extend>>
Dispaly Missing Data Error
Install Libraries
Use strictly necessary Libraries
Use extra Utility Libraries
Determine Training/Test Split Ratio
<<Extend>>
Determine number of Cross Validation folds
Prepare Data
<<Extend>>
Extension Points
Use Training/Test Split
Use Cross Validation
Sometimes using training/test split/ or cross validation is better. The Data scienteist will have to determine this.
Data Scientist
Bank
Create Algorithm
<<Include>>
Train Algorithm
<<Include>>
Tune Algorithm
Make Predictions on Data
<<Include>>
Acquire and Provide new customer data
Display and Summarize Predictions

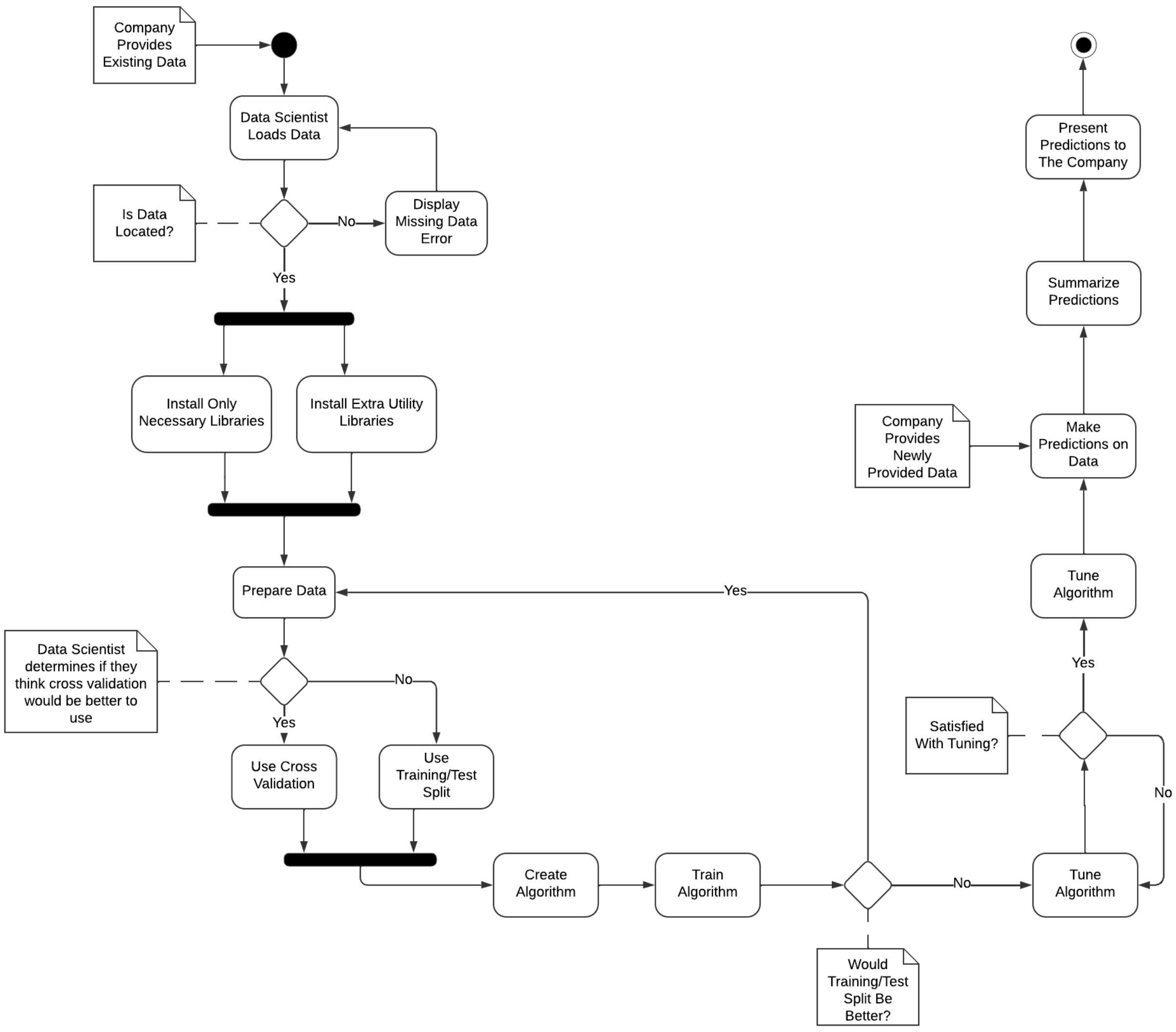
This diagram is called a use case diagram as it shows the use case of a system, which in this case is our algorithm. The actors of this system are the data scientist and the bank. The bank is really just providing us with data to train our algorithm on, data to then use our algorithm for predictions, and then we will both look at the results and see what to make of them. The data scientist (me, in this case) is the one who really does all the work with the algorithm. The data scientist will train, tune, and predict with the algorithm and find the right solution.

Domain Model Class Diagram



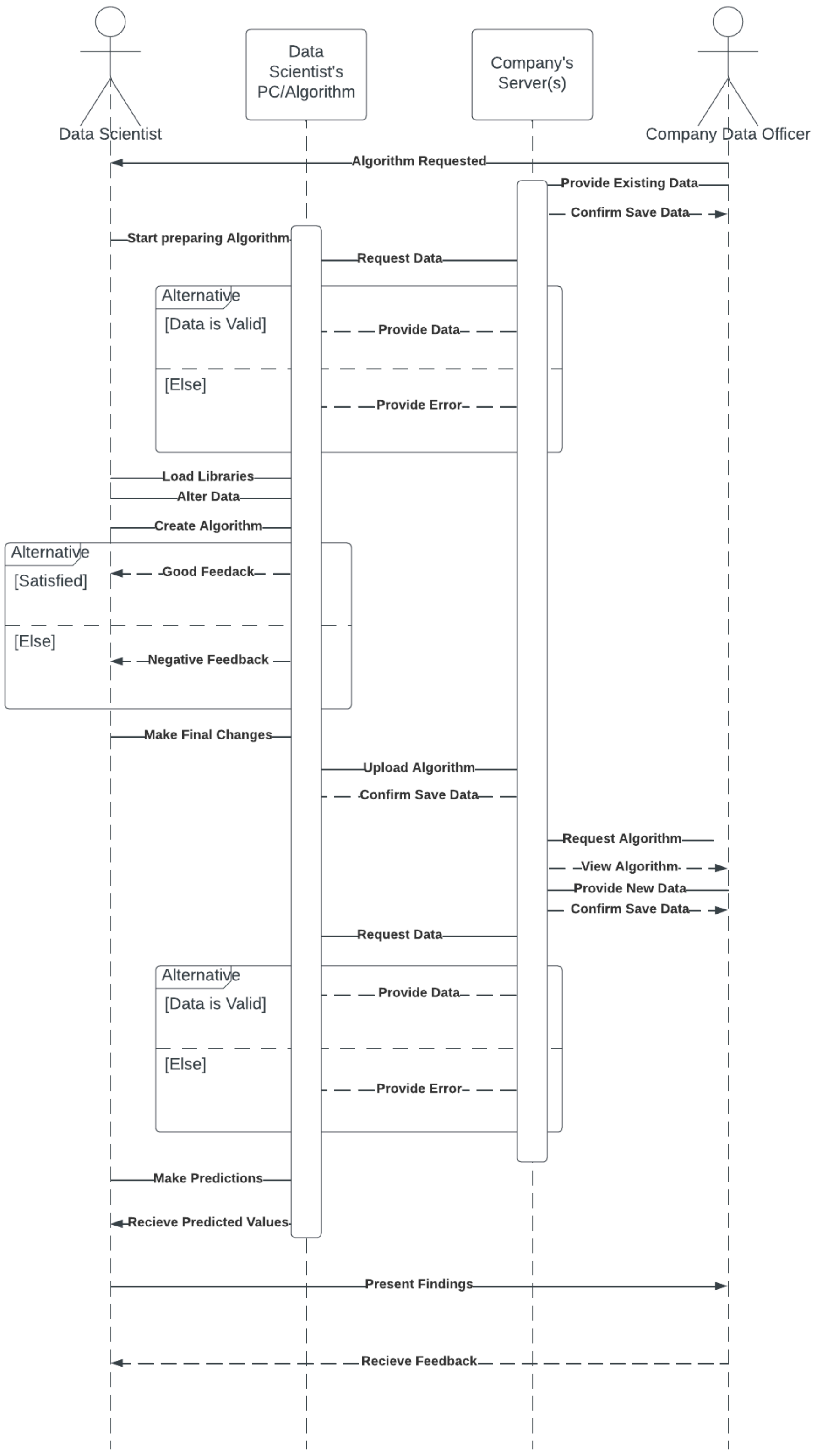
The domain model class diagram shows how you would keep track of everything we're doing in this project inside a database. It shows every table we would make like Data Scientist, Customer, Company, and so on. It also shows all of the columns there would be in each table like CompanyID, CompanyName, and CompanyLocation. The lines in the graph show how each table interacts with each other. We do not make a database in this project, but if we were to this diagram would be incredibly useful.

Activity Diagram


Company Provides Existing Data
Data Scientist Loads Data
Is Data Located?
No
Display Missing Data Error
Yes
Install Only Necessary Libraries
Install Extra Utility Libraries
Prepare Data
Data Scientist determines if they think cross validation would be better to use
No
Yes
Use Cross Validation
Use Training/Test Split
Create Algorithm
Train Algorithm
Yes
No
Would Training/Test Split Be Better?
Tune Algorithm
Satisfied With Tuning?
No
Yes
Tune Algorithm
Company Provides Newly Provided Data
Make Predictions on Data
Summarize Predictions
Present Predictions to The Company

The activity diagram shows the activity of the entire process we're doing here. It shows the entire process of making this algorithm from the beginning to the end. As we can see this project starts with N.R. Bank providing us with the data, and ends with us presenting our model's predictions back to N.R. Bank. This is again to again help them with separating high-risk borrowers (likely to default) from low-risk borrowers. This also shows some of the steps of making the algorithm we'll see in greater detail in the algorithm process section.

System Sequence Diagram


Data Scientist
Data Scientist's PC/Algorithm
Company's Server(s)
Company Data Officer
Algorithm Requested
Provide Existing Data
Confirm Save Data
Start preparing Algorithm
Request Data
Alternative
[Data is Valid]
Provide Data
[Else]
Provide Error
Load Libraries
Alter Data
Create Algorithm
Alternative
[Satisfied]
Good Feedack
[Else]
Negative Feedback
Make Final Changes
Upload Algorithm
Confirm Save Data
Request Algorithm
View Algorithm
Provide New Data
Confirm Save Data
Request Data
Alternative
[Data is Valid]
Provide Data
[Else]
Provide Error
Make Predictions
Recieve Predicted Values
Present Findings
Recieve Feedback

The system sequence diagram shows the sequence of the system which in this case is the algorithm and N.R. Bank's servers. This graph shows how the algorithm and the company's server interact with each other throughout the overall sequence of the system. This graph not only helps in showing the chronological sequence of these activities in this system, but also shows us how active each piece of the system is.

The Algorithm Process

We start the algorithm by loading our libraries and removing the ID variable, as it is unnecessary. We then split the data 80/20 for our training and testing data. The tools we will use to analyze our models every iteration will be the confusion matrix, the ROC curve, and the AUC score. The specific metric we want to optimize for is recall as we want to miss as few defaulters as possible.

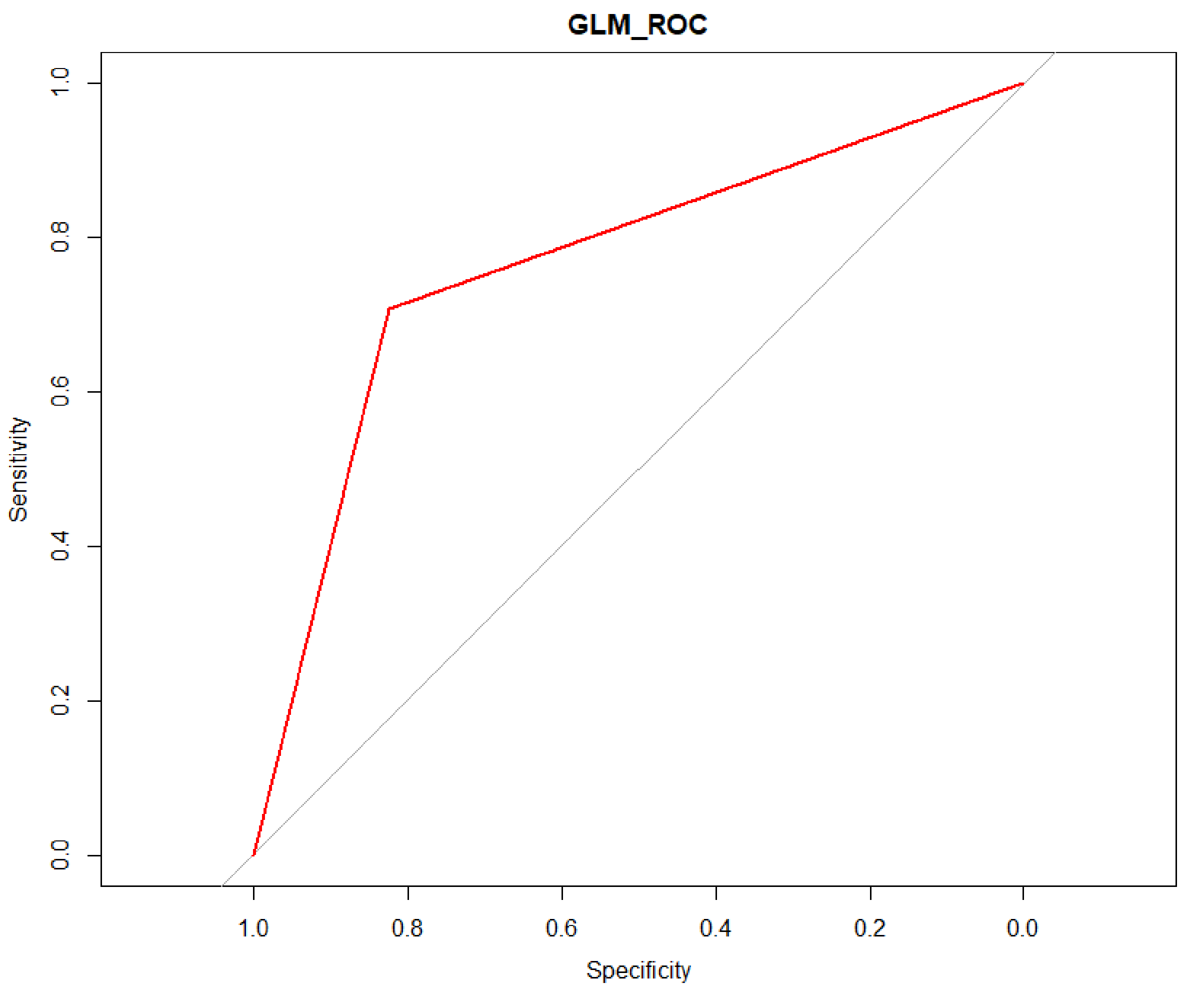
Iteration 1

In the first iteration of our algorithm we made a very basic GLM (logistic regression) model on the data. The prefix "GLM_" in the code is for code relating to this model iteration. The confusion matrix gives us the following results:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4584  975
         1  129  312
                                          
               Accuracy : 0.816           
                 95% CI : (0.806, 0.8257)
    No Information Rate : 0.7855          
    P-Value [Acc > NIR] : 2.567e-09       
                                          
                  Kappa : 0.2826          
                                          
 Mcnemar's Test P-Value : < 2.2e-16       
                                          
            Sensitivity : 0.2424          
            Specificity : 0.9726          
         Pos Pred Value : 0.7075          
```

```
        Neg Pred Value : 0.8246
            Prevalence : 0.2145
        Detection Rate : 0.0520
  Detection Prevalence : 0.0735
     Balanced Accuracy : 0.6075

       'Positive' Class : 1
```

As we see, our recall (called sensitivity in the confusion matrix) is really low at only 0.2424. While our very rudimentary model gave us an accuracy of 0.81, the low recall makes this model really quite useless. Next we take a look at the ROC curve and AUC score:



```
Area under the curve: 0.766
```

The horizontal gray line in these ROC curve graphs represents the performance of someone randomly guessing the classification. The AUC score of this random guessing is 0.5 and this model's AUC score is 0.766 which means it's better than randomly guessing and isn't a bad number for our first model. The low recall again really leaves something to be desired. To try and remedy this we will first check to see if there's a data imbalance that's causing our low recall:

```
> > # Checking for imbalance and getting minority percentage.
> maj_class <- sum(Train_Data$default.payment.next.month == 0)
> print(maj_class)
[1] 18651
> min_class <- sum(Train_Data$default.payment.next.month == 1)
> print(min_class)
[1] 5349
> imbal <- min_class/(min_class+maj_class)
> print(imbal)
[1] 0.222875
```

As we see from this check it appears the data is quite unbalanced as 1 is the minority class that only makes up roughly 22.29% of our training data.

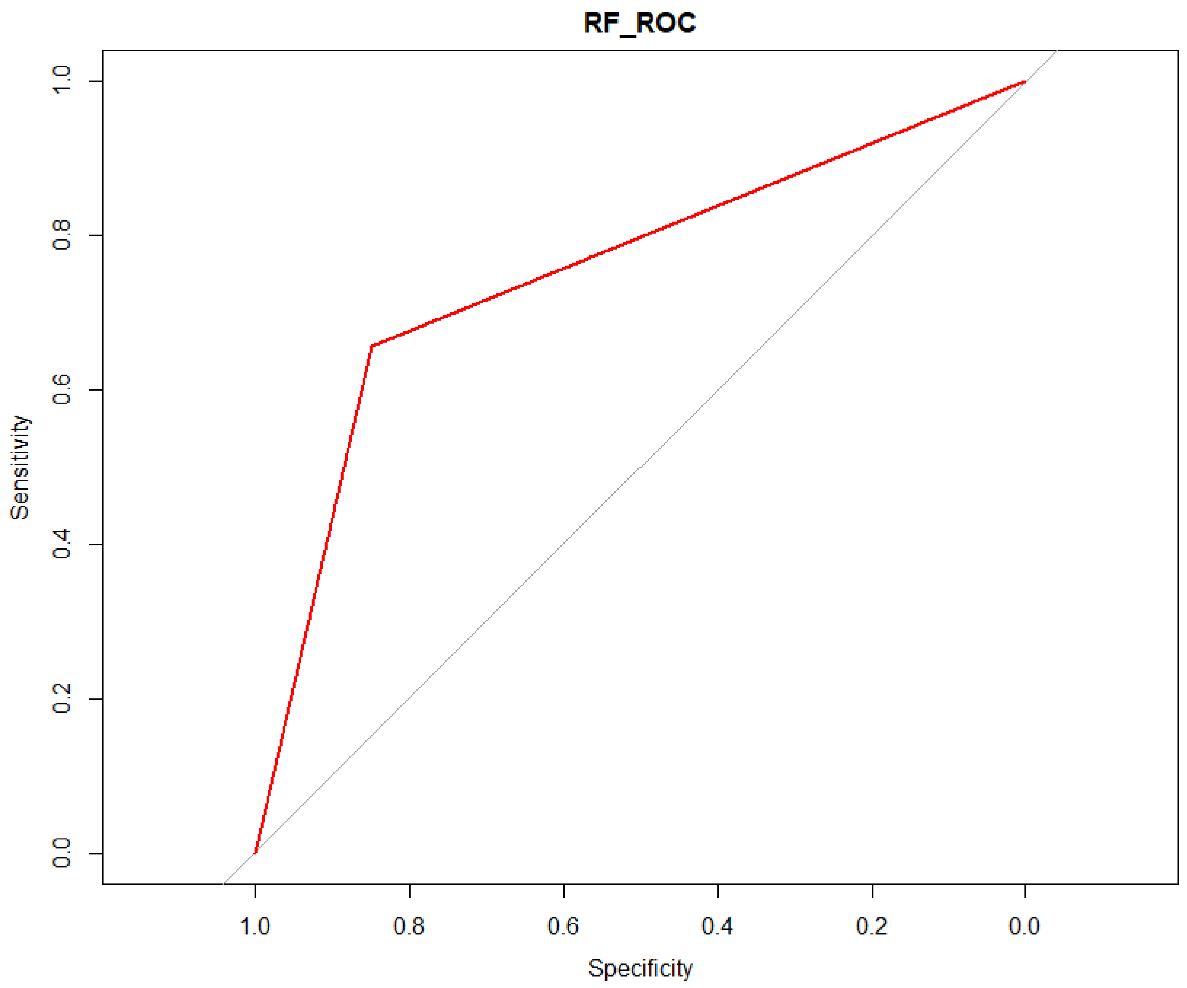
Iteration 2

We now know that our data is imbalanced. We are going to try and make a random forest algorithm as they can oftentimes handle class imbalance better. We are also going to implement a 5-fold cross validation method with the rest of our iterations as it should perform better with unusual data than just the training/test split. This is the confusion matrix for our RF_Model:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4457  799
         1  256  488

               Accuracy : 0.8242
                 95% CI : (0.8143, 0.8337)
    No Information Rate : 0.7855
    P-Value [Acc > NIR] : 4.597e-14

                  Kappa : 0.3837

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.37918
            Specificity : 0.94568
         Pos Pred Value : 0.65591
         Neg Pred Value : 0.84798
             Prevalence : 0.21450
         Detection Rate : 0.08133
   Detection Prevalence : 0.12400
      Balanced Accuracy : 0.66243

       'Positive' Class : 1
```

This confusion matrix tells us that progress is being made here as our accuracy and recall have increased! Our accuracy is now 0.8242 and our sensitivity increased to 0.37. This is better, but still not good enough, as this means we would only be 37% accurate in predicting defaulters.

Next we will look at the ROC curve and AUC score:



```
Area under the curve: 0.7519
```

We've lost some of our AUC score from the last model, but again our recall has increased which is the most important.

Iteration 3

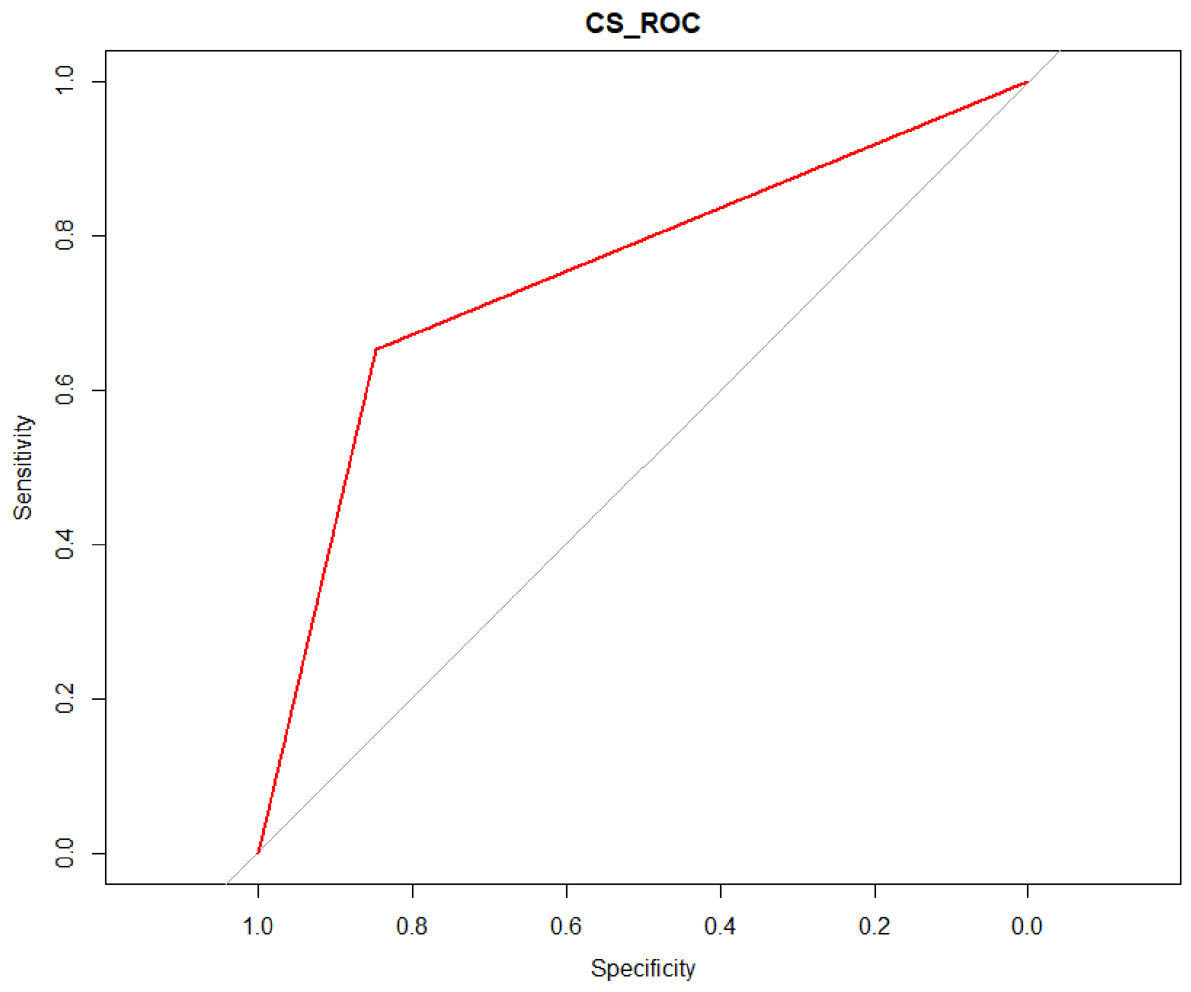
This time around we will again use a random forest model with cross validation, but this time we will include class weights. Class weights basically will tell the algorithm that it's much more important to correctly guess the minority class, which should increase recall. The idea of this is called cost sensitive learning, hence the "CS_" prefix in the related code. We do this with the following code:

```
# Defining class weights
CW <- ifelse(Train_Data$default.payment.next.month == 0, 1, 10)
```

Now that we have given the CS_Model it's class weights, let's see it's confusion matrix:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4455  801
         1  258  486

               Accuracy : 0.8235
                 95% CI : (0.8136, 0.8331)
    No Information Rate : 0.7855
    P-Value [Acc > NIR] : 1.24e-13

                  Kappa : 0.3814

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3776
            Specificity : 0.9453
         Pos Pred Value : 0.6532
         Neg Pred Value : 0.8476
             Prevalence : 0.2145
         Detection Rate : 0.0810
   Detection Prevalence : 0.1240
      Balanced Accuracy : 0.6614

       'Positive' Class : 1
```

This confusion matrix lets us see that this method was rather ineffective at increasing recall or accuracy. I had high hopes for cost sensitive learning, but in this case it didn't really seem to help us, but let's see the ROC curve and AUC score too:



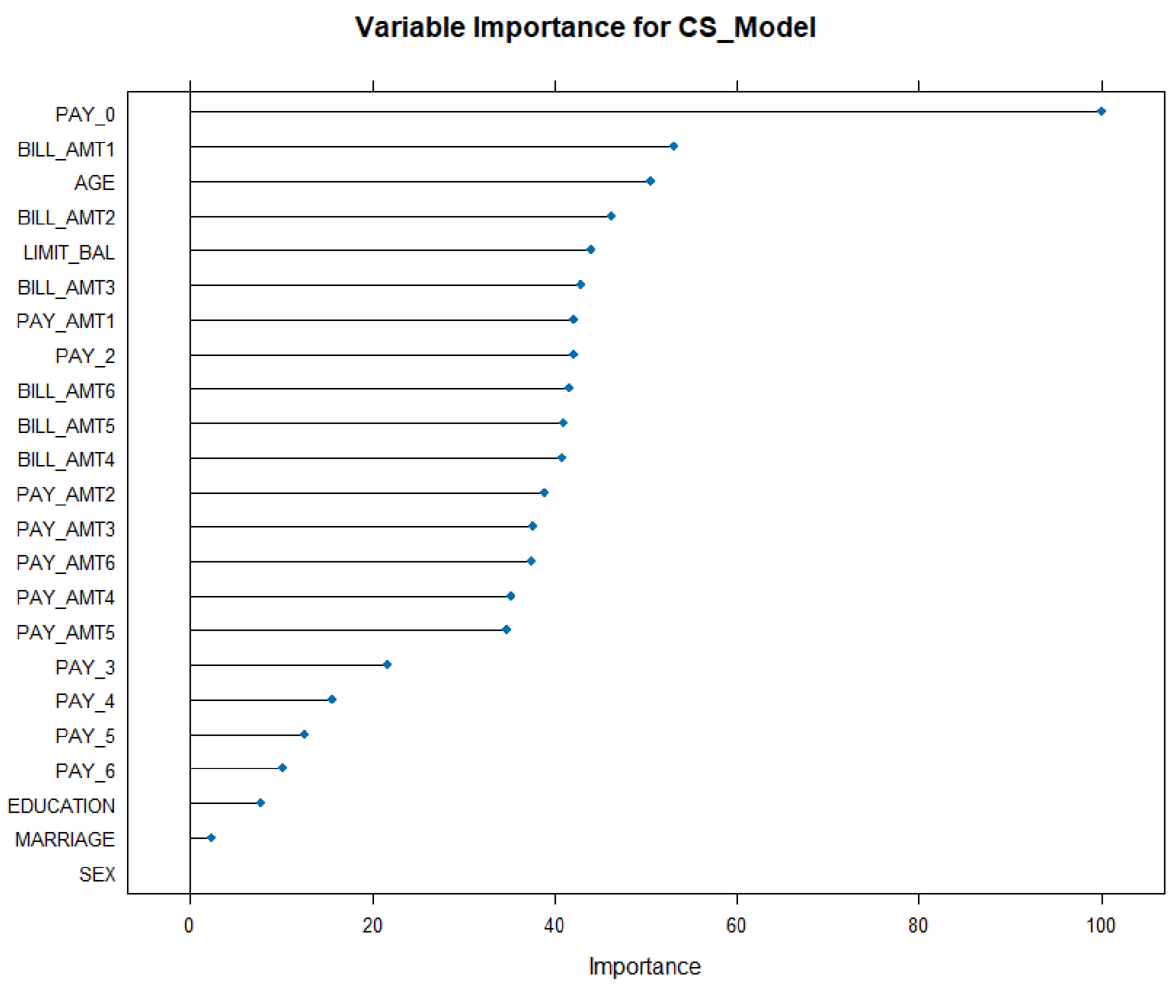
```
Area under the curve: 0.7504
```

Again, the ROC curve and AUC score also show us that the solution for this iteration was rather unsuccessful.

Iteration 4

A possible explanation for the ineffectiveness of the last iteration could suggest there’s a lot of noise in our data. We will therefore, try to see if there’s any prectors that are not important as we do have a lot of them. We can see the variable importance of the last iteration with the following code and output:

```
# Testing for what variables may not be important
test_imp <- varImp(CS_Model)
plot(test_imp, main = "Variable Importance for CS_Model")
```



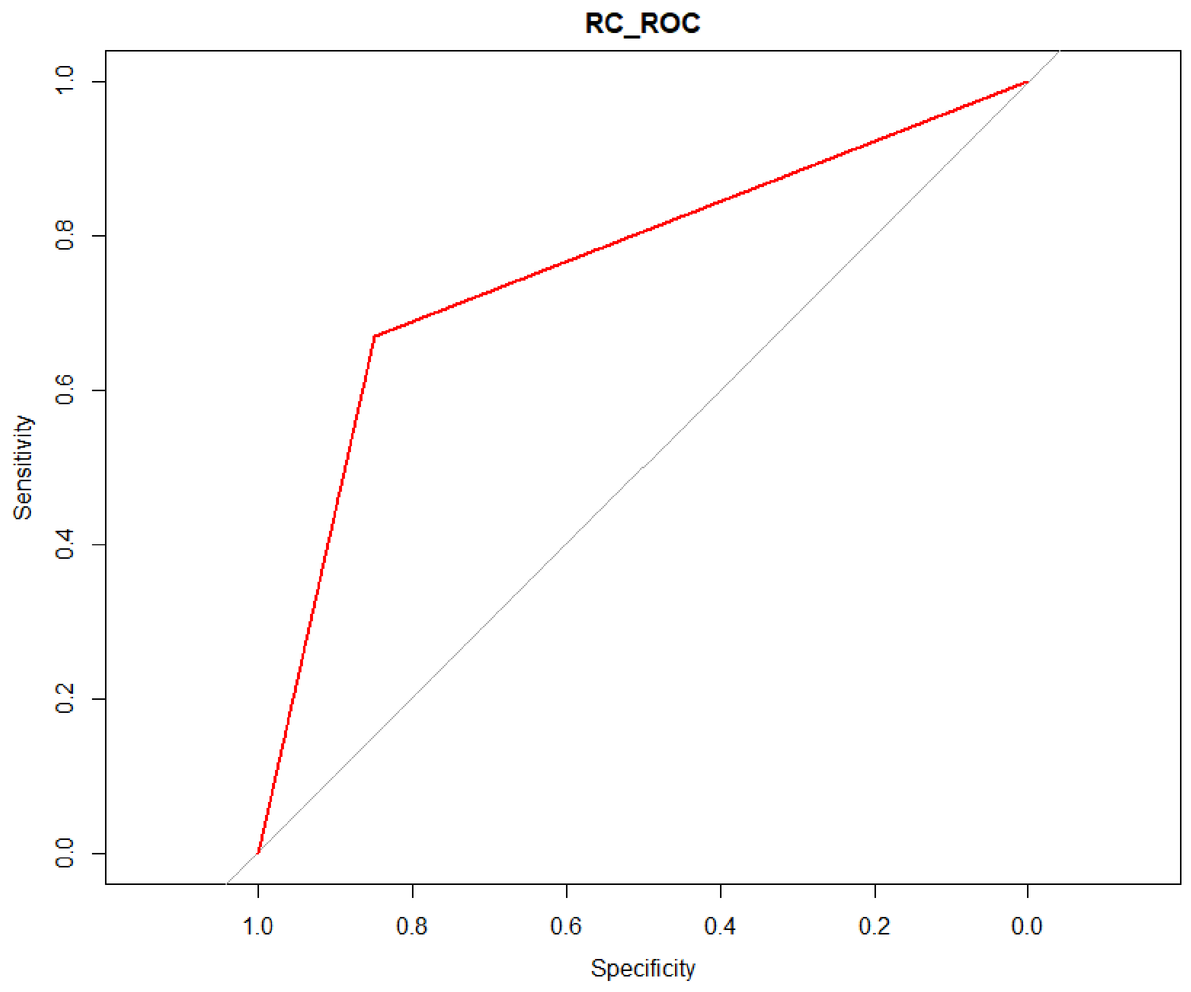
This shows us that there's sort of a cutoff of average importance below the variable PAY_AMT5. Therefore, we will remove the variables less important than it with the following code:

```
# Removing Variables of low importance
Train_Data <- Train_Data %>%
  dplyr::select(-SEX, -MARRIAGE, -EDUCATION, -PAY_6, -PAY_4,
                -PAY_5, -PAY_3)
Test_Data <- Test_Data %>%
  dplyr::select(-SEX, -MARRIAGE, -EDUCATION, -PAY_6, -PAY_4,
                -PAY_5, -PAY_3)
```

With our new data, we are also going to try a different solution in this model than class weights. We are going to set the metric parameter in the train() function specifically to recall. By doing this we should see a higher recall as we're telling the model to optimize for it. Let's take a look at RC_Model's confusion matrix:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4469  794
         1  244  493

               Accuracy : 0.827
                 95% CI : (0.8172, 0.8365)
    No Information Rate : 0.7855
    P-Value [Acc > NIR] : 5.523e-16

                  Kappa : 0.3922

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.38306
            Specificity : 0.94823
         Pos Pred Value : 0.66893
         Neg Pred Value : 0.84914
             Prevalence : 0.21450
         Detection Rate : 0.08217
   Detection Prevalence : 0.12283
      Balanced Accuracy : 0.66564

       'Positive' Class : 1
```

This confusion matrix shows that recall has again slightly increased to 0.38. This still suggests only very slight progress. Let’s check the ROC curve and AUC score:



```
Area under the curve: 0.759
```

The change in the ROC curve and AUC score is also almost negligible. This iteration was only a slight success, which is still an improvement from our last iteration.

Iteration 5

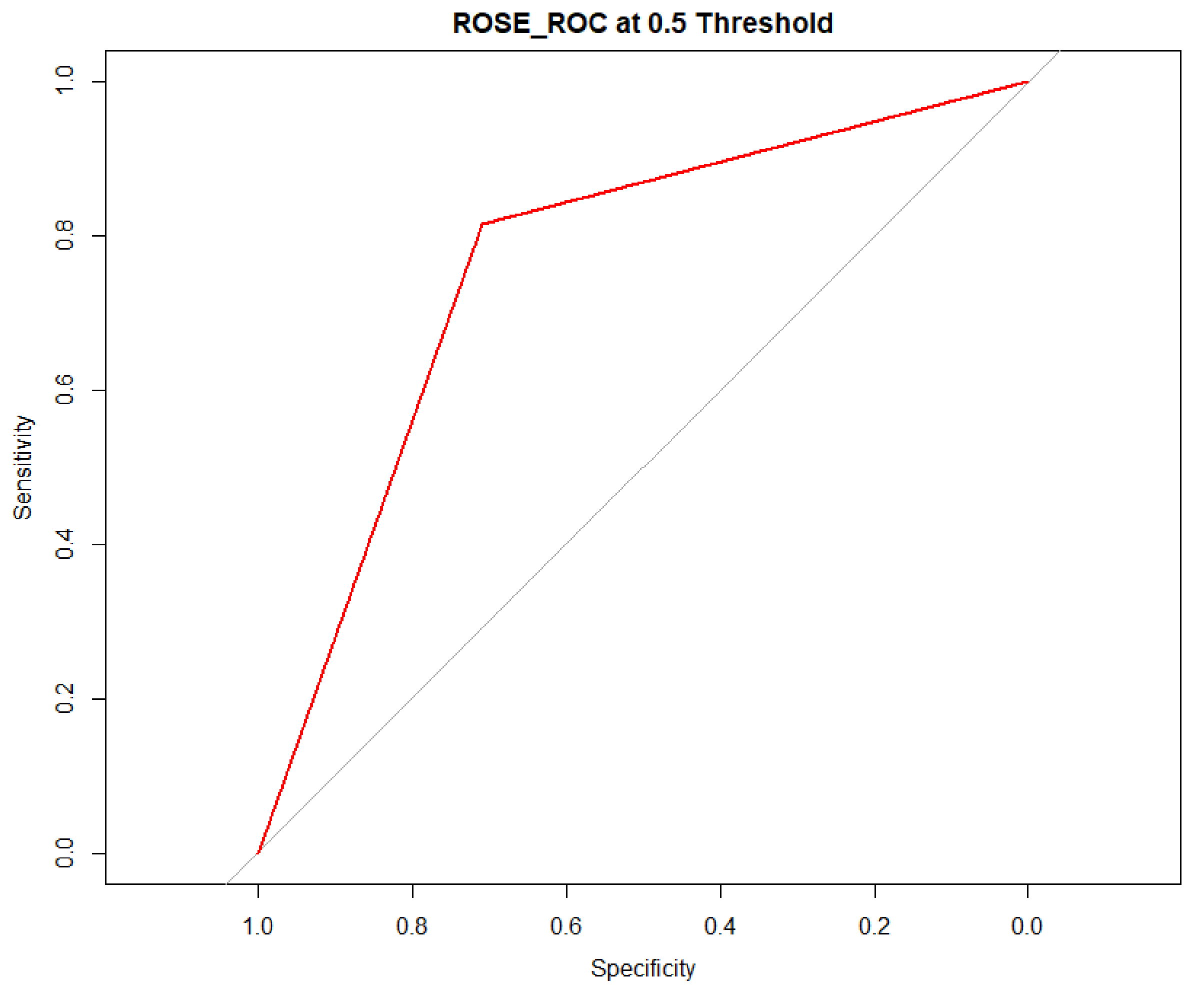
In this iteration, I thought of another way to possibly solve the problem of imbalanced data. There's a method called upsampling where you basically make data similar to that already existing in your minority class. There are various techniques to essentially make up more of this data while trying to keep it as similar to your already existing data as possible. I chose to utilize the ROSE package. We upsample the data with ROSe with the following code:

```
TRAIN_R = ROSE(default.payment.next.month ~ .,
                    data = Train_Data, seed = 2112)$data

      TEST_R = ROSE(default.payment.next.month ~ .,
                   data = Test_Data, seed = 2112)$data
```

Now that we're using our upsampled data let's take a look at the confusion matrix:

```
      Confusion Matrix and Statistics

                Reference
      Prediction    0    1
               0 2541 1045
               1  447 1967

                     Accuracy : 0.7513
                       95% CI : (0.7402, 0.7622)
          No Information Rate : 0.502
          P-Value [Acc > NIR] : < 2.2e-16

                        Kappa : 0.5031

       Mcnemar's Test P-Value : < 2.2e-16

                  Sensitivity : 0.6531
                  Specificity : 0.8504
               Pos Pred Value : 0.8148
               Neg Pred Value : 0.7086
                   Prevalence : 0.5020
               Detection Rate : 0.3278
         Detection Prevalence : 0.4023
            Balanced Accuracy : 0.7517

             'Positive' Class : 1
```

While our accuracy decreased, our recall increased dramatically! This is definitely the best model we've had so far for recall, which is our main evaluation metric. Let's take a look at our ROC curve and AUC score:



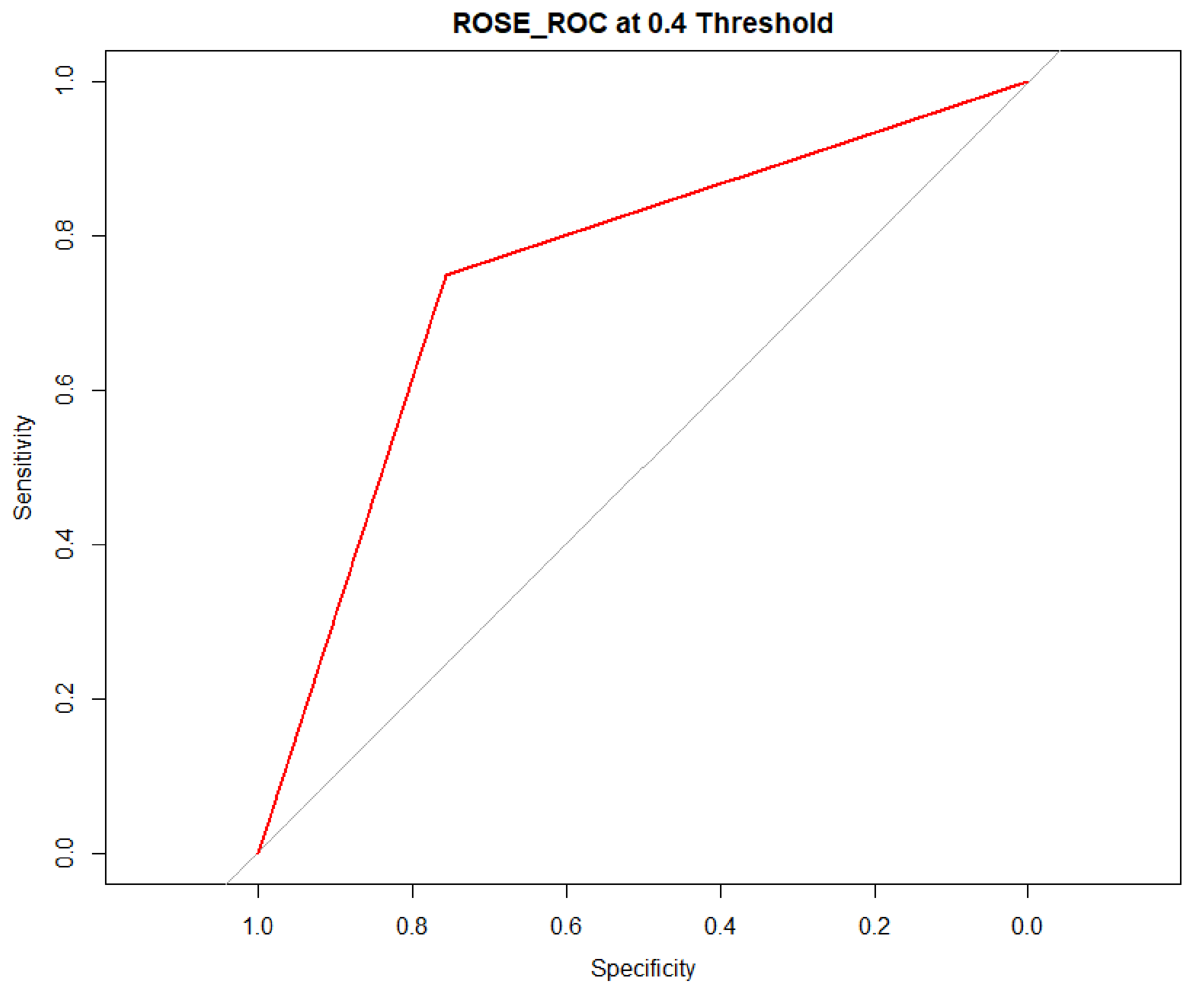
```
Area under the curve: 0.7617
```

Our AUC score is still pretty good and the curve looks normal still although it's a little curved upwards. For all of our other models we've been making predictions at a 0.5 threshold. This means that while the model predicts 0 to 1, instead of giving just 0 or 1 it gives a value of between 0 and 1. A 0.5 threshold means that if the value increases above 0.5 it will be counted as a one. We'll make predictions with this same model at a threshold of 0.4. Let's look at the confusion matrix:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2215  712
         1  773 2300
```

```
               Accuracy : 0.7525
                 95% CI : (0.7414, 0.7634)
    No Information Rate : 0.502
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.505

 Mcnemar's Test P-Value : 0.1195

            Sensitivity : 0.7636
            Specificity : 0.7413
         Pos Pred Value : 0.7485
         Neg Pred Value : 0.7567
             Prevalence : 0.5020
         Detection Rate : 0.3833
   Detection Prevalence : 0.5122
      Balanced Accuracy : 0.7525

       'Positive' Class : 1
```

And our ROC curve and AUC score:



```
Area under the curve: 0.7526
```

This change in our threshold again significantly increased our recall. It's increased all the way up to 0.76 which is the highest yet! Our ROC curve is again good and pretty normal shaped and the AUC score is also good still with minimal loss. This means that we can be fairly confident in predicting our minority class (defaulters) much more than the under 40% of our initial models. This model at a 0.4 threshold will be our final model.

The Final Model

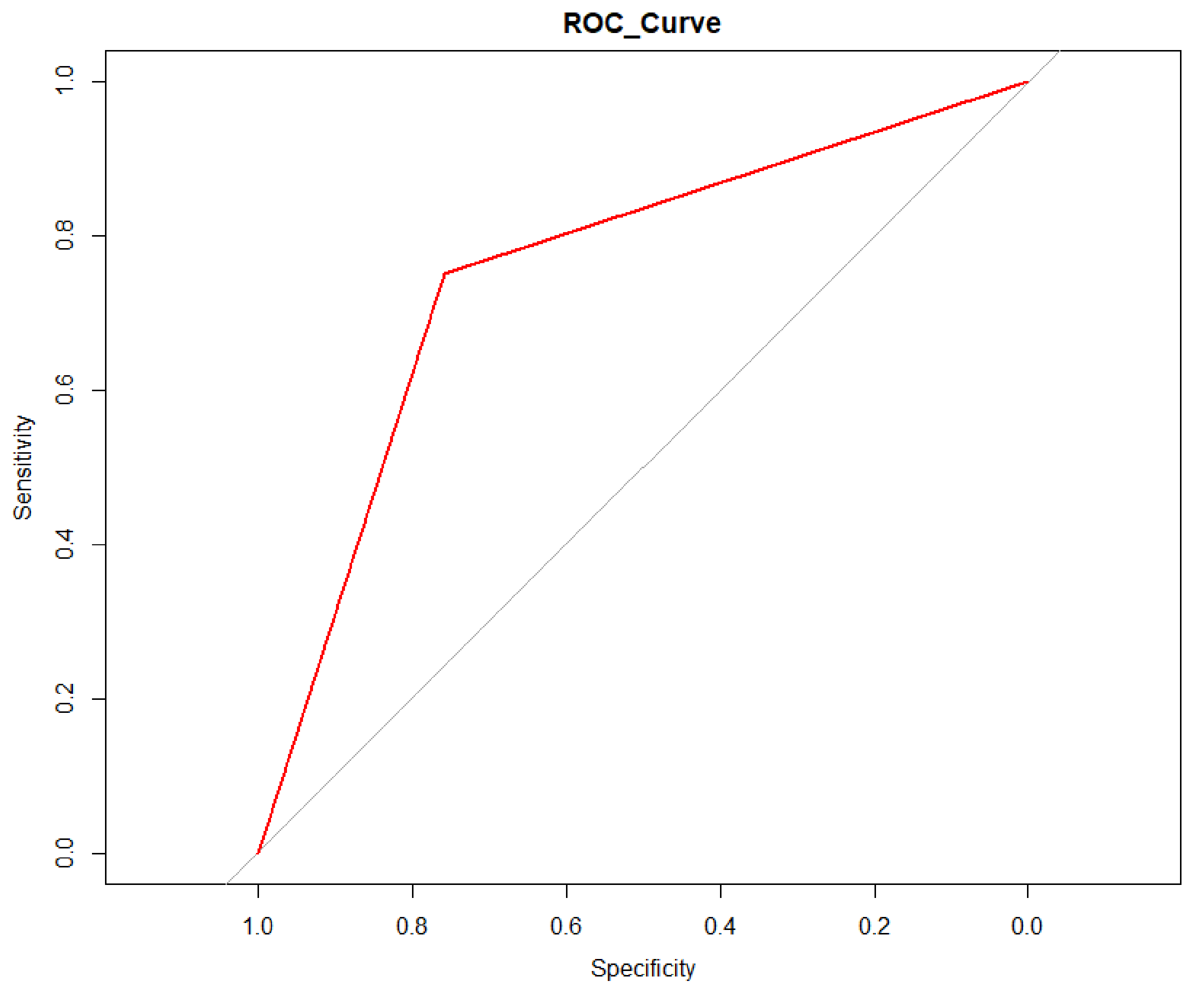
Here's the code for our final model:

```
# Finalizing the model
Training_Data <- TRAIN_R
Testing_Data <- TEST_R
```

```
Model <- train(default.payment.next.month ~ ., data = Training_Data,
                    method = "rf", trControl = TrCtrl,
                    tuneGrid = data.frame(.mtry = 5))
print(Model)

Predictions <- predict(Model, Testing_Data, type = "raw")
Predictions <- ifelse(Predictions >= 0.4, 1, 0)

Observations <- Testing_Data$default.payment.next.month
Confusion_Matrix <- confusionMatrix(as.factor(Predictions),
                                    as.factor(Observations), positive =
"1")
print(Confusion_Matrix)

ROC_Curve <- roc(Predictions, Observations)
plot(ROC_Curve, main = "ROC_Curve", col = "red")
auc(ROC_Curve)
```

Again, here is the confusion matrix and ROC curve / AUC score:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2225  712
         1  763 2300

               Accuracy : 0.7542
                 95% CI : (0.7431, 0.765)
    No Information Rate : 0.502
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.5083

 Mcnemar's Test P-Value : 0.193

            Sensitivity : 0.7636
            Specificity : 0.7446
         Pos Pred Value : 0.7509
         Neg Pred Value : 0.7576
             Prevalence : 0.5020
         Detection Rate : 0.3833
   Detection Prevalence : 0.5105
      Balanced Accuracy : 0.7541

       'Positive' Class : 1
```



```
Area under the curve: 0.7542
```

The Complete Algorithm Code

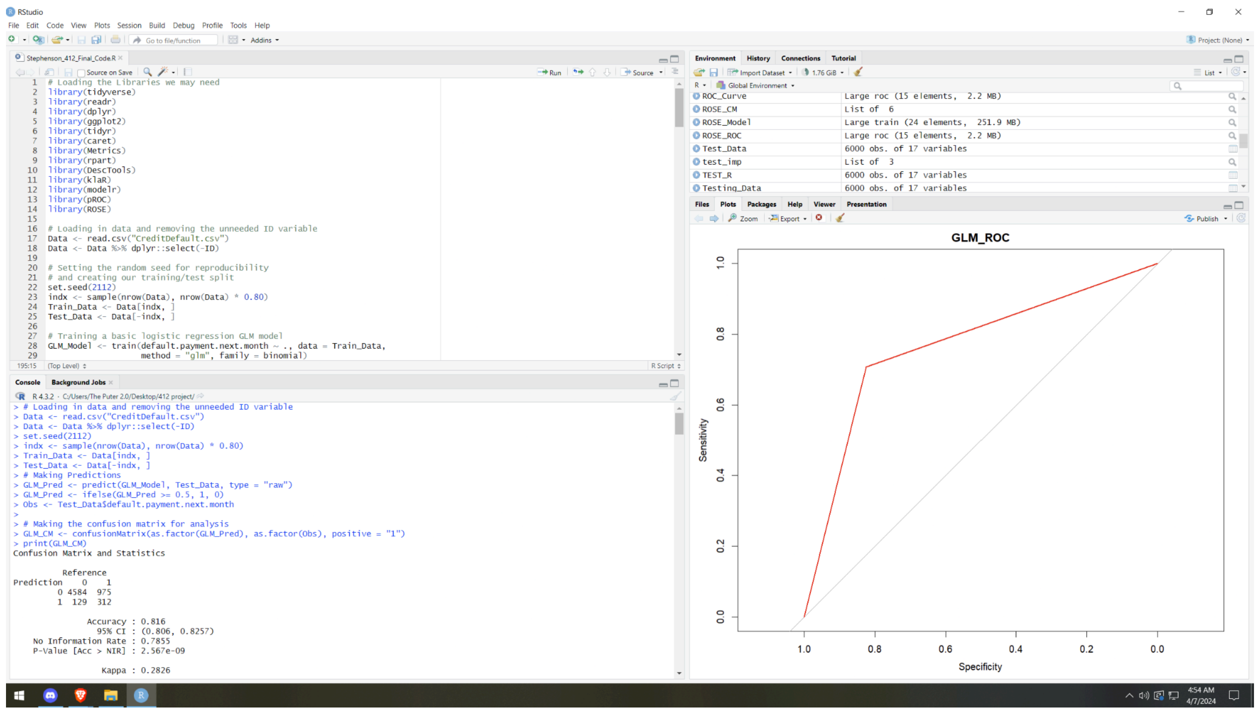
```
# Loading the Libraries we may need
library(tidyverse)
library(readr)
library(dplyr)
library(ggplot2)
library(tidyr)
library(caret)
library(Metrics)
library(rpart)
library(DescTools)
library(klaR)
library(modelr)
library(pROC)
library(ROSE)

# Loading in data and removing the unneeded ID variable
Data <- read.csv("CreditDefault.csv")
Data <- Data %>% dplyr::select(-ID)

# Setting the random seed for reproducibility
# and creating our training/test split
set.seed(2112)
indx <- sample(nrow(Data), nrow(Data) * 0.80)
Train_Data <- Data[indx, ]
Test_Data <- Data[-indx, ]

# Training a basic logistic regression GLM model
GLM_Model <- train(default.payment.next.month ~ ., data = Train_Data,
                   method = "glm", family = binomial)

# Making Predictions
GLM_Pred <- predict(GLM_Model, Test_Data, type = "raw")
GLM_Pred <- ifelse(GLM_Pred >= 0.5, 1, 0)
Obs <- Test_Data$default.payment.next.month

# Making the confusion matrix for analysis
GLM_CM <- confusionMatrix(as.factor(GLM_Pred), as.factor(Obs), positive = "1")
print(GLM_CM)

# Making the ROC curve and AUC score for analysis
GLM_ROC <- roc(GLM_Pred, Obs)
plot(GLM_ROC, main = "GLM_ROC", col = "red")
auc(GLM_ROC)

# Checking for imbalance and getting minority percentage.
maj_class <- sum(Train_Data$default.payment.next.month == 0)
print(maj_class)
min_class <- sum(Train_Data$default.payment.next.month == 1)
print(min_class)
imbal <- min_class/(min_class+maj_class)
print(imbal)


#Setting train control for 5 fold cross validation
TrCtrl <- trainControl(method = "cv", number = 5)

# Training random forest model.
RF_Model <- train(default.payment.next.month ~ ., data = Train_Data,
                  method = "rf", trControl = TrCtrl,
                  tuneGrid = data.frame(.mtry = 5))

print(RF_Model)

# Making predictions
RF_Pred <- predict(RF_Model, Test_Data, type = "raw")
RF_Pred <- ifelse(RF_Pred >= 0.5, 1, 0)

# Making the confusion matrix for analysis
RF_CM <- confusionMatrix(as.factor(RF_Pred), as.factor(Obs), positive = "1")
print(RF_CM)
```

```
# Making the ROC Curve and getting AUC score for analysis
RF_ROC <- roc(RF_Pred, Obs)
plot(RF_ROC, main ="RF_ROC", col = "red")
auc(RF_ROC)

# Defining class weights
CW <- ifelse(Train_Data$default.payment.next.month == 0, 1, 10)

# Creating a random forest model with class weights
CS_Model <- train(default.payment.next.month ~ ., data = Train_Data,
                  method = "rf", trControl = TrCtrl,
                  tuneGrid = data.frame(.mtry = 5),
                  weights = CW)
print(CS_Model)

# Making predictions
CS_Pred <- predict(CS_Model, Test_Data, type = "raw")
CS_Pred <- ifelse(CS_Pred >= 0.5, 1, 0)

# Making the confusion matrix for analysis
CS_CM <- confusionMatrix(as.factor(CS_Pred), as.factor(Obs), positive = "1")
print(CS_CM)

# Making the ROC Curve and getting AUC score for analysis
CS_ROC <- roc(CS_Pred, Obs)
plot(CS_ROC, main ="CS_ROC", col = "red")
auc(CS_ROC)

# Testing for what variables may not be important
test_imp <- varImp(CS_Model)
plot(test_imp, main = "Variable Importance for CS_Model")

# Removing Variables of low importance
Train_Data <- Train_Data %>%
  dplyr::select(-SEX, -MARRIAGE, -EDUCATION, -PAY_6, -PAY_4,
                -PAY_5, -PAY_3)
Test_Data <- Test_Data %>%
  dplyr::select(-SEX, -MARRIAGE, -EDUCATION, -PAY_6, -PAY_4,
                -PAY_5, -PAY_3)


# Training a random forest model specifically for recall
RC_Model <- train(default.payment.next.month ~ ., data = Train_Data,
                  method = "rf", trControl = TrCtrl, metric = "recall",
                  tuneGrid = data.frame(.mtry = 5))
print(RC_Model)

# Making Predictions
RC_Pred <- predict(RC_Model, Test_Data, type ="raw")
RC_Pred <- ifelse(RC_Pred >= 0.5, 1, 0)

# Making the confusion matrix
RC_CM <- confusionMatrix(as.factor(RC_Pred), as.factor(Obs), positive = "1")
print(RC_CM)
```

```
# Making the ROC Curve and getting AUC score for analysis
RC_ROC <- roc(RC_Pred, Obs)
plot(RC_ROC, main = "RC_ROC", col = "red")
auc(RC_ROC)

# upsampling our training and test data
TRAIN_R = ROSE(default.payment.next.month ~ .,
               data = Train_Data, seed = 2112)$data

TEST_R = ROSE(default.payment.next.month ~ .,
              data = Test_Data, seed = 2112)$data

# Training a random forest model with our upsampled data from ROSE
ROSE_Model <- train(default.payment.next.month ~ ., data = TRAIN_R,
                    method = "rf", trControl = TrCtrl,
                    tuneGrid = data.frame(.mtry = 5))
print(ROSE_Model)

# Making predictions
ROSE_Pred <- predict(ROSE_Model, TEST_R, type = "raw")
ROSE_Pred <- ifelse(ROSE_Pred >= 0.5, 1, 0)

# Making the confusion matrix for analysis
ROSE_CM <- confusionMatrix(as.factor(ROSE_Pred),
                           as.factor(TEST_R$default.payment.next.month),
                           positive = "1")
print(ROSE_CM)

# Making the ROC Curve and getting AUC score for analysis
ROSE_ROC <- roc(ROSE_Pred, TEST_R$default.payment.next.month)
plot(ROSE_ROC, main = "ROSE_ROC at 0.5 Threshold", col = "red")
auc(ROSE_ROC)

# Analysis of model at 0.4 threshold

# Making predictions
ROSE_Pred <- predict(ROSE_Model, TEST_R, type = "raw")
ROSE_Pred <- ifelse(ROSE_Pred >= 0.4, 1, 0)
R_Obs <- TEST_R$default.payment.next.month

# Making the confusion matrix for analysis
ROSE_CM <- confusionMatrix(as.factor(ROSE_Pred), as.factor(R_Obs),
                           positive = "1")
print(ROSE_CM)

# Making the ROC Curve and getting AUC score for analysis
ROSE_ROC <- roc(ROSE_Pred, R_Obs)
plot(ROSE_ROC, main = "ROSE_ROC at 0.4 Threshold", col = "red")
auc(ROSE_ROC)

# Finalizing the model
Training_Data <- TRAIN_R
Testing_Data <- TEST_R
```

```
Model <- train(default.payment.next.month ~ ., data = TRAIN_R,
                    method = "rf", trControl = TrCtrl,
                    tuneGrid = data.frame(.mtry = 5))
print(Model)

Predictions <- predict(Model, Testing_Data, type = "raw")
Predictions <- ifelse(Predictions >= 0.4, 1, 0)

Observations <- Testing_Data$default.payment.next.month
Confusion_Matrix <- confusionMatrix(as.factor(Predictions),
                                    as.factor(Observations), positive = "1")
print(Confusion_Matrix)

ROC_Curve <- roc(Predictions, Observations)
plot(ROC_Curve, main = "ROC_Curve", col = "red")
auc(ROC_Curve)
```

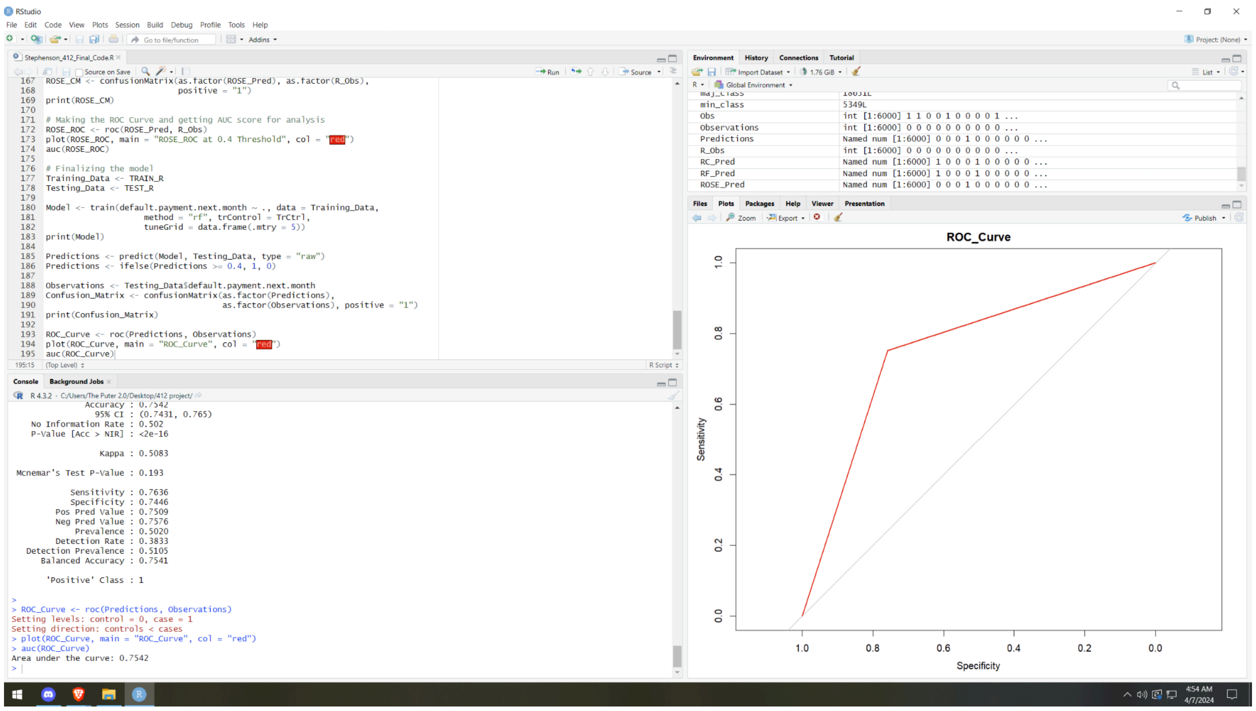
Screenshots of Code Running in RStudio



```
# Loading the Libraries we may need
library(tidyverse)
library(readr)
library(dplyr)
library(ggplot2)
library(tidyr)
library(caret)
library(Metrics)
library(rpart)
library(DescTools)
library(klaR)
library(modelr)
library(pROC)
library(ROSE)

# Loading in data and removing the unneeded ID variable
Data <- read.csv("CreditDefault.csv")
Data <- Data %>% dplyr::select(-ID)

# Setting the random seed for reproducibility
# and creating our training/test split
set.seed(2112)
indx <- sample(nrow(Data), nrow(Data) * 0.80)
Train_Data <- Data[indx, ]
Test_Data <- Data[-indx, ]

# Training a basic logistic regression GLM model
GLM_Model <- train(default.payment.next.month ~ ., data = Train_Data,
                   method = "glm", family = binomial)
```

```
> # Loading in data and removing the unneeded ID variable
> Data <- read.csv("CreditDefault.csv")
> Data <- Data %>% dplyr::select(-ID)
> set.seed(2112)
> indx <- sample(nrow(Data), nrow(Data) * 0.80)
> Train_Data <- Data[indx, ]
> Test_Data <- Data[-indx, ]
> # Making Predictions
> GLM_Pred <- predict(GLM_Model, Test_Data, type = "raw")
> GLM_Pred <- ifelse(GLM_Pred >= 0.5, 1, 0)
> Obs <- Test_Data$default.payment.next.month
>
> # Making the confusion matrix for analysis
> GLM_CM <- confusionMatrix(as.factor(GLM_Pred), as.factor(Obs), positive = "1")
> print(GLM_CM)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4584  975
         1  129  312

               Accuracy : 0.816
                 95% CI : (0.806, 0.8257)
    No Information Rate : 0.7855
    P-Value [Acc > NIR] : 2.567e-09

                  Kappa : 0.2826
```

```
ROC_Curve        Large roc (15 elements,  2.2 MB)
ROSE_CM          List of  6
ROSE_Model       Large train (24 elements,  251.9 MB)
ROSE_ROC         Large roc (15 elements,  2.2 MB)
Test_Data        6000 obs. of 17 variables
test_imp         List of  3
TEST_R           6000 obs. of 17 variables
Testing_Data     6000 obs. of 17 variables
```



```r
ROSE_CM <- confusionMatrix(as.factor(ROSE_Pred), as.factor(R_Obs),
                           positive = "1")
print(ROSE_CM)

# Making the ROC Curve and getting AUC score for analysis
ROSE_ROC <- roc(ROSE_Pred, R_Obs)
plot(ROSE_ROC, main = "ROSE_ROC at 0.4 Threshold", col = "red")
auc(ROSE_ROC)

# Finalizing the model
Training_Data <- TRAIN_R
Testing_Data <- TEST_R

Model <- train(default.payment.next.month ~ ., data = Training_Data,
                    method = "rf", trControl = TrCtrl,
                    tuneGrid = data.frame(.mtry = 5))
print(Model)

Predictions <- predict(Model, Testing_Data, type = "raw")
Predictions <- ifelse(Predictions >= 0.4, 1, 0)

Observations <- Testing_Data$default.payment.next.month
Confusion_Matrix <- confusionMatrix(as.factor(Predictions),
                                    as.factor(Observations), positive = "1")
print(Confusion_Matrix)

ROC_Curve <- roc(Predictions, Observations)
plot(ROC_Curve, main = "ROC_Curve", col = "red")
auc(ROC_Curve)
```

```
               Accuracy : 0.7542
                 95% CI : (0.7431, 0.765)
    No Information Rate : 0.502
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.5083

 Mcnemar's Test P-Value : 0.193

            Sensitivity : 0.7636
            Specificity : 0.7446
         Pos Pred Value : 0.7509
         Neg Pred Value : 0.7576
             Prevalence : 0.5020
         Detection Rate : 0.3833
   Detection Prevalence : 0.5105
      Balanced Accuracy : 0.7541

       'Positive' Class : 1

>
> ROC_Curve <- roc(Predictions, Observations)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(ROC_Curve, main = "ROC_Curve", col = "red")
> auc(ROC_Curve)
Area under the curve: 0.7542
> |
```

```
maj_class      18651L
min_class      5349L
Obs            int [1:6000] 1 1 0 0 1 0 0 0 0 1 ...
Observations   int [1:6000] 0 0 0 0 0 0 0 0 0 0 ...
Predictions    Named num [1:6000] 0 0 0 1 0 0 0 0 0 0 ...
R_Obs          int [1:6000] 0 0 0 0 0 0 0 0 0 0 ...
RC_Pred        Named num [1:6000] 1 0 0 0 1 0 0 0 0 0 ...
RF_Pred        Named num [1:6000] 1 0 0 0 1 0 0 0 0 0 ...
ROSE_Pred      Named num [1:6000] 0 0 0 1 0 0 0 0 0 0 ...
```

Conclusion

Overall, while this model is not perfect, as few often are, this model is performing pretty well for our imbalance and noisy dataset. This algorithm can give N.R. Bank a better idea on predicting its customer base. This will be especially helpful for providing insight on those who may be likely to default so N.R Bank can be more careful with them without labeling too many people as high-risk who aren't. This is a good balance between caution and over-caution which should lead to high customer satisfaction and peace of mind for N.R. Bank.