

# Advanced Data Mining and Analytics

## Group Project Report

### **Group 3**

Rishi Sai Reddy Sudireddy  
Monalisa Singh  
Zi Yang

### **\*Author Note**

This report was prepared for Advanced Data Mining and Analytics, MIS 64037,  
taught by Dr. Rouzbeh Razavi

## Table of Contents

Project Goal -----	2
Overview of Data -----	3
Modeling Strategy -----	8
Estimation of Model's Performance -----	12
Insights and Conclusions -----	13

## **Project Goal**

Machine learning has had fruitful applications in finance well before the advent of mobile banking apps, proficient chat bots, or search engines. Given high volume, accurate historical records, and quantitative nature of the finance world, few industries are better suited for artificial intelligence. There are more uses cases of machine learning in finance than ever before, a trend perpetuated by more accessible computing power and more accessible machine learning tools. Today, machine learning has come to play an integral role in many phases of the financial ecosystem, from approving loans, to managing assets, to assessing risks.

There is a great deal of worry in the industry that machines will replace a large swath of the underwriting positions that exist today. Especially at large companies (big banks and publicly traded insurance firms), machine learning algorithms can be trained on millions of examples of consumer data (age, job, marital status, etc...) and financial lending or insurance results. The underlying trends that can be assessed with algorithms, and continuously analyzed to detect trends that might influence lending and insuring into the future.

In this project, Our task is to determine whether a loan will default, as well as the loss incurred if it does default. Unlike traditional finance-based approaches to this problem, where one distinguishes between good or bad counterparties in a binary way, we seek to anticipate and incorporate both the default and the severity of the losses that result. In doing so, we are building a bridge between traditional banking, where we are looking at reducing the consumption of economic capital, to an asset-management perspective, where we optimize on the risk to the financial investor.

## Overview of Data

### a)Data Exploration

The data that was given to us was split in two parts; train\_v3.csv and test\_\_no\_lossv3.csv. Both the data had columns named as id, f1, f2 and so on to ensure privacy of the customers to save it from fraud usage. The number of columns and rows in the train csv were 778 and 80000 respectively. The number of rows and columns for the test csv were 778 and 25472 respectively. Both these csv's were loaded into R for the modelling.

```
{r}
My.Data<-read.csv('D:\\ADM\\train_v3.csv')

{r}
My.TestData<-read.csv('D:\\ADM\\test__no_lossv3.csv')
```

We used R notebook for this project as it also enables us to save our file as an html file and can be viewed on the web along with all the outputs.

### b)Data Cleaning

On opening the train csv file, we noticed that columns like f33, f34, f35, f37 and f38 had 0 throughout these columns. To check it further we applied filter on the csv for 0 and it gave all the rows as 0. Hence, when we loaded the csv as a dataframe, we directly removed those columns from our data.

```
My.Data1<-My.Data
View(My.Data1)
My.Data1$f33<-NULL
My.Data1$f34<-NULL
My.Data1$f35<-NULL
My.Data1$f37<-NULL
My.Data1$f38<-NULL
```

Another strange thing that we noticed in the dataframe was that it seemed that some columns were being repeated in a certain pattern. To verify this, we removed the duplicated columns and the number of columns reduced in our dataframe.

```
My.Data1[!duplicated(as.list(My.Data1))]
My.Data2<-My.Data1[!duplicated(as.list(My.Data1))]
```

After removing duplicate columns.

My.Data3 39430 obs. of 674 variables

Most importantly, it was necessary to deal with the various missing values in the data. On checking upon the data we saw that the missing values were at random and did not indicate anything. We ran the Amelia package on the missing data and found that the influence that the missing data had on the prediction was negligible.

```
library(Amelia)
complete_data<-amelia(My.Data3)
My.Data4<-complete_data

install.packages("zelig", dependencies = TRUE)
library(zelig)
library(zeligverse)
z.out<- zelig(loss~., data=complete_data$imputations, model="ls",cite = FALSE)
view(complete_data)
loss<- reformulate(setdiff(colnames(My.Data4)))
```

We initially imputed the missing values with the median imputation that has no side effects on the data. However, as we progressed in our model, we noticed that there were no difference in the prediction when we removed the missing values and when we imputed them. Hence, we decided to go back and remove all the missing values.

```
My.Data3<-My.Data2[complete.cases(My.Data2),]
```

X <int>	f1 <int>	f3 <dbl>	f4 <int>	f5 <int>
78539	120	0.146251127	2200	4
61541	154	0.349399073	4200	4
76531	126	0.975969168	1500	10
22066	127	0.951302949	3100	7
45589	162	0.518485643	3500	7
99812	131	0.321643369	2200	7
43515	122	0.487797882	1600	16
1429	128	0.533352314	1600	4
32243	131	0.585260305	1500	17
73119	131	0.229810848	3500	16

1-10 of 80,000 rows | 1-5 of 674 columns

Previous 1 2 3 4 5 6 ... 100 Next

After dealing with missing values we wanted to standardise/normalise the data.

Before doing so, we have intentionally removed the loss column and placed it into a new dataframe. The intention being that we do not want to normalise the loss function since we need to use it to create the default column. We also removed the column X (that represents the id) since we did not want to transform that either.

```
My.Data3a<-My.Data3$loss
My.Data3$loss<-NULL
My.Data3b<-My.Data3$X
My.Data3$X<-NULL
```

Once that was done, We ran normalisation using min-max on our code initially as shown below;

```

{r}
maxx= apply(Train , 2 , max)
minn= apply(Train, 2 , min)
trainTransformed = as.data.frame(scale(Train, center = minn, scale = maxx - minn))
testTransformed= as.data.frame(scale(Validation, center = minn, scale = maxx - minn))

```

On further research we found that using **z-transformation** (that is subtracting the mean and divide by the standard deviation) is considered better for better probabilities in our prediction. It also is unaffected by outlier values, unlike the normalisation using min-max. Normalizing our data will certainly scale the “normal” data to a very small interval. And generally, most of data sets have outliers. Hence, we decided to go with z-transformation.

```

{r}
My.Data3[] <- lapply(My.Data3, scale)

```

We also wanted to get rid of those columns that had the least or zero variance. We applied the nearZeroVar function in R. The issue that we saw here was removing these NZV variables was also removing the ‘loss’ column in our data. After a lot of contemplation, we decided to only remove those variables that had **no variance** in them at all. This ensured that our ‘loss’ column remained intact in our data.

```

My.Data3Transformed1<-My.Data3Transformed
zv<-apply(My.Data3Transformed1,2,function(x)length(unique(x))==1)
zv
My.Data3Transformed2<-My.Data3Transformed1[,!zv]
str(My.Data3Transformed2)

```

This reduced our columns even further;

My.Data3Transfo... 39430 obs. of 671 variables

The next step was to deal with the correlated variables in the data. Since the data was so huge, it was difficult to check the correlation between all variables and manually remove each one.


Our approach to solving this was to find the **correlation** between all the variables and remove those; keeping only one of those correlated columns. This helps in reducing our data and better prediction. We took 95% as threshold to remove the correlated columns.

```

{r}
NewTrain<-cor(My.Data3Transformed2, use='complete.obs')
hc=findCorrelation(NewTrain, cutoff = 0.95)
reduced_train<-My.Data3Transformed2[, -hc]
dim(reduced_train)

```

By running this code, this is what our data looked like:

reduced\_train 39430 obs. of 317 variables 

As you can see, we now have a lot less column and rows than we originally had.

One last step in this process was to create a default column in this data. We added back the loss column to the data. We also added back the X column.

```
#Restoring loss and X and creating default column from loss
reduced_train$loss<-My.Data3a
reduced_train$X<-My.Data3b
reduced_train$default<-ifelse(reduced_train$loss>0, 1, 0) #default HERE is loss binary
````
```

Since we had the loss column, we wanted to determine that according to the amount of loss, so and so person(according to the id column) was defaulting or not. This column was a binary column with values of 0 or 1.

|    | v1 |  |
|----|----|--|
| 1  | 0  |  |
| 2  | 1  |  |
| 3  | 0  |  |
| 4  | 0  |  |
| 5  | 0  |  |
| 6  | 1  |  |
| 7  | 0  |  |
| 8  | 0  |  |
| 9  | 1  |  |
| 10 | 0  |  |
| 11 | 1  |  |
| 12 | 0  |  |
| 13 | 0  |  |
| 14 | 0  |  |
| 15 | 0  |  |
| 16 | 0  |  |

After thoroughly cleaning our data in this order we decided to pursue towards building the model.

## Modeling Strategy

The first step was to split the data. We kept the split ratio as 0.8 i.e 80% data into the training data and rest 20% into the test data.

```
```{r}
library(caTools)
set.seed(1000)
split = sample.split(reduced_train, splitRatio = 0.8)
Traindata = subset(reduced_train, split == TRUE)
Testdata = subset(reduced_train, split == FALSE)
```
```

We then decided to build the data on our training set and test it over the test set.

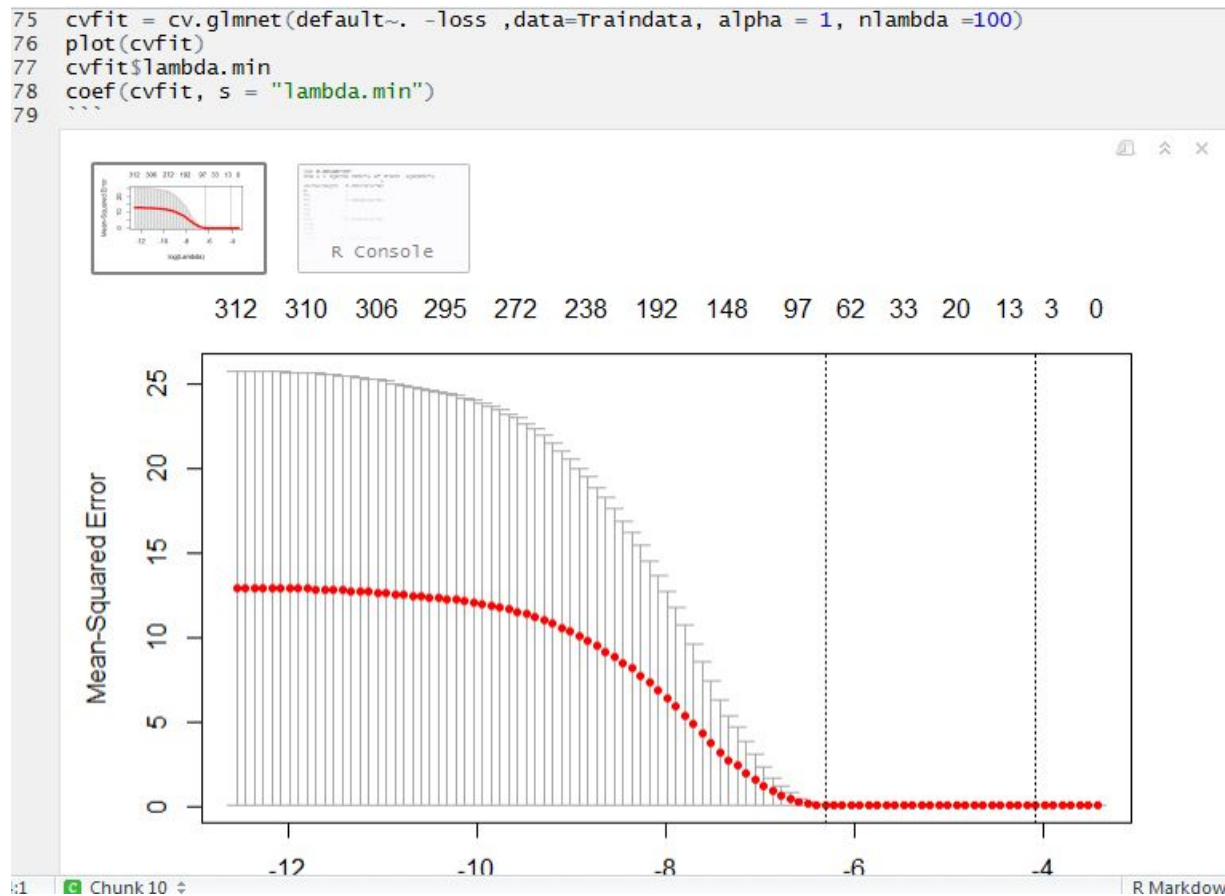
The first model we built was a **lasso regression** model. We did this in order to regularize the penalties over the coefficients of the variables. This could help in providing better predictions.

We used the glmnet model that is computed for the lasso or elastic net penalty at a grid of values for the regularization parameter lambda. It Can deal with all shapes of data, including very large sparse data matrices. Fits linear, logistic and multinomial, poisson, and Cox regression models.

Here,

1. alpha=1 implements Lasso penalty.
2. Cross Validation is run for selecting Lambda.
3. lambda.min selects the best lambda.





The other appreciable advantage of using Lasso Regression is that it gives us coefficients for those variables that are important. We determine the importance of the variable as to how far it is from 1.

```

t680      5.222627e-04
f682      .
f693     -1.146772e-02
f699      .
f715      .
f716      .
f717      .
f724     1.426157e-03
f725      .
f726      .
f733      .
f734      .
f735      .
f737      .
f739      .
f740      .
f742      .
f743      .
f744      .
f746      .
f755      .
f756      .
f760      .
f763      .
f765      .

```

This gave us around 55 variables that seemed to be important for the prediction of default.

We tested our lasso regression model over the test data to check their MSE and MAE which will be discussed in the next section.

Our next step was to build a classification model using these variables. We manually picked the important variables to build the model.

```
{R}  
#Classification model using important features according to lasso  
cvfitglm1 = glm(default~ f3 +f13 +f26 +f44 +f45 +f55 +f57+f67 +f69 +f71 +f73 +f75 +f81  
+f91 +f102 +f123 +f129+f132+f134+f139+f143+f146+f151+f163+f172+f190+f191+f198  
+f221+f240+f251+f261+f268+f298+f301+f306+  
f317+f322+f333+f340+f359+f361+f364+f383+f403+f406+f411+f419+f422+f428+f433+f442+f461+f471+  
f489+f512+f514+f518+f523+f530+f533+f536+f546+f598+f604+f614+f619+f628+f639+f654+f669+f671+  
f672+f674+f677+f679+f680+f693+f724+f776,data=Traindata,family = binomial())  
summary(cvfitglm1)  
  
predicts_glm<-predict(cvfitglm1, newdata = Testdata, type="response")  
residuals(cvfitglm1, type="deviance")  
view(predicts_glm)  
library(pROC)  
roc(Testdata$default, predicts_glm)  
plot(roc(Testdata$default, predicts_glm), col='red', lwd=3)
```

We tested this model over our sample test data.

The issue with this was that running the Lasso model again and again was giving different important features most of the time and we found this to be unreliable.

We built a basic classification model to check if it gave a better performance.

```
#trying simple classification model to see if it works better  
glmmodel<-glm(default~. -X -loss, family = "binomial", data = Traindata)  
summary(glmmodel)  
  
predicts_glm1<-predict(glmmodel, newdata = Testdata, type="response")  
residuals(glmmodel, type="deviance")  
library(pROC)  
roc(Testdata$default, predicts_glm1)  
plot(roc(Testdata$default, predicts_glm1), col='red', lwd=3)  
  
predicts_glm2<-predict(glmmodel, newdata = Traindata, type="response")
```

The Roc for this model was better and we decided to use this. We then created the columns to show the original default variables vs the predicted default probabilities for both the train and test data.

```
#Taking predictions from this model and applying it on train and test data
library(dplyr)
Data=Traindata
Dataa=Testdata
Data$predDefault=predicts_glm2
Dataa$predDefault=predicts_glm1
Data1=Data[,c("default","predDefault")]
Data1
Dataa1=Dataa[,c("default","predDefault")]
Dataa1
...
```

This is what the real vs predicted looked like.

|     | <b>default</b><br><dbl> | <b>predDefault</b><br><dbl> |
|-----|-------------------------|-----------------------------|
| 31  | 0                       | 0.043055016                 |
| 41  | 0                       | 0.236383461                 |
| 49  | 0                       | 0.057330989                 |
| 66  | 0                       | 0.113319374                 |
| 79  | 0                       | 0.058770342                 |
| 93  | 0                       | 0.071801544                 |
| 114 | 0                       | 0.148465904                 |
| 126 | 0                       | 0.094875994                 |
| 129 | 0                       | 0.294394397                 |
| 133 | 0                       | 0.045606310                 |

We next created a Regression model only for those people who were defaulting.

```
#Regression Model
#Taking only those who had default 1 (i.e loss binary=1)
Traindata1<-subset(Traindata, Traindata$default==1)
Traindata1$default
Testdata1<-subset(Testdata, Testdata$default==1)
Testdata1$default

cvfit1<-cv.glmnet(loss~. -X=default,data =Traindata1, alpha=1, nlambda=100)
plot(cvfit1)
cvfit1$lambda.min
coef(cvfit1, s = "lambda.min")
summary(cvfit1)

predicts_4<-predict(cvfit1, newdata= Testdata1 , s = "lambda.min")
view(predicts_4)
view(Dataa)

Td<-Testdata1
Td$predloss<-predicts_4
c1<-Td[,c("X","loss","predloss")]
...
```

We further took different thresholds to come to the one that suited our model the best and gave the least MAE. This threshold helped us in connecting the probabilities of default for those who were defaulting i.e connecting the classification model to the regression model.

```
#different thresholds to get better MAE
Testdata2=Dataa
Testdata3<-subset(Testdata2, predDefault>0.3)#This gave us the most reliable MAE

#Check for threshold
library(dplyr)
T1=subset(Testdata3, default>0)#Regression model using these number of people who
defaulted
nrow(T1)/nrow(Testdata3)##% of defaulters in regression model from classification model. |

predicts_5<-predict(cvfit1, newdata= Testdata3 , s = "lambda.min")
library(Metrics)
MAEnew= mae(Testdata3$loss, predicts_5)
MAEnew

#Output on Sample Tesdata
Testdata3$Predloss=abs(predicts_5)
c2=Testdata3[,c("X", "loss", "Predloss")]
````
```

For the prediction, we predicted both the models over the My.TestData file that was created from the Test\_\_no\_lossv3.csv.

Before predicting it, we scaled the test data set as well along with some other data cleaning techniques since our training set was scaled before building the model.

```
My.TestData<-read.csv('D:\\ADM\\test__no_lossv3.csv')

My.TestData1=My.TestData[complete.cases(My.TestData),]

zv <- apply(My.TestData1, 2, function(x) length(unique(x)) == 1)
My.TestData2 <- My.TestData1[, !zv]

My.Testdata3<-My.TestData2$X
My.TestData2$X<-NULL
My.TestData3<-My.TestData2
My.TestData3[] <- lapply(My.TestData2, scale)
My.TestData3$X<-My.Testdata3
````
```



Applying the classification model and displaying the output.

```
{r}  
#Applying the classification model  
library(dplyr)  
Prediction<-predict(glmmodel,My.TestData3,type='response')  
Prediction  
  
#Results  
c3=My.TestData3  
c3$Defaultprob=Prediction  
c4=c3[,c("X","Defaultprob")]
```

|    | X      | Defaultprob |
|----|--------|-------------|
| 1  | 7933   | 0.125734687 |
| 2  | 101860 | 0.068540529 |
| 5  | 48008  | 0.047990015 |
| 6  | 9308   | 0.110518515 |
| 7  | 27862  | 0.029210509 |
| 8  | 87760  | 0.034520290 |
| 9  | 53334  | 0.225490687 |
| 10 | 72303  | 0.169373374 |
| 11 | 63146  | 0.101584476 |
| 12 | 36604  | 0.069504779 |
| 13 | 39206  | 0.062727447 |
| 17 | 74349  | 0.137854721 |
| 18 | 96978  | 0.037102120 |
| 20 | 22864  | 0.016225264 |
| 24 | 101249 | 0.254962769 |
| 27 | 73152  | 0.118170985 |

ing 1 to 16 of 12,510 entries

Next, before we applied the Regression model and displayed the results of our prediction we set the threshold value again for connecting the models.

```
{r}  
#Applying threshold to regression model  
My.TestData4=subset(c3,Defaultprob>0.3)  
My.TestData5=My.TestData4  
My.TestData5$Defaultprob=NULL  
}
```

Applying Regression model and showing results.

```
#applying the Regression model
predicts_glm_lasso<-predict(cvfit1, newdata = My.TestData6,s = "lambda.min")
predicts_glm_lasso
#Output File
c5=My.TestData6
c5$Defaults=abs(predicts_glm_lasso)
c6=c5[,c("X","Defaults")]
```

Outcome.

|      | X     | Defaults   |
|------|-------|------------|
| 58   | 65249 | 0.38311518 |
| 90   | 12485 | 0.51159813 |
| 472  | 16663 | 3.68740090 |
| 794  | 64385 | 0.77849201 |
| 861  | 92270 | 1.90997565 |
| 927  | 27170 | 3.88858609 |
| 1026 | 55752 | 4.49379944 |
| 1087 | 78502 | 3.12820444 |
| 1198 | 81921 | 5.65362730 |
| 1275 | 39451 | 1.98794112 |
| 1289 | 91319 | 4.02302463 |
| 1325 | 12245 | 4.62632141 |
| 1442 | 26568 | 0.48100890 |
| 1544 | 96805 | 0.68162224 |
| 1665 | 26666 | 1.71247963 |
| 1690 | 92780 | 0.18772178 |

Showing 1 to 16 of 272 entries

This file showed the severity of loss for those who were defaulting in the test\_\_no\_lossv3.csv

We further tried models like Neural Networks, PCA and Random Forests. The issue with these algorithms were that they ran for a day and came out to be computationally expensive while not producing any results. Hence, we decided to stick with the glm model.

## Estimation of Model's Performance

We tested both the models over our sample of the test data.

We initially used the MSE(Mean squared error) and computed the MAE( Mean Absolute Error) from it.

where

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|,$$

- $n$  is the number of rows
- $\hat{y}_i$  is the predicted loss
- $y_i$  is the actual loss

### a)Lasso Regression Model

```
predicts_glm_lasso<-predict(cvfit, newdata = Testdata, s = "lambda.min")
MSE_glm_lasso=mean(sqrt((predicts_glm_lasso-Testdata$default)^2))
print(MSE_glm_lasso)
summary(MSE_glm_lasso)
view(predicts_glm_lasso)
```
```

```
[1] 0.1809523
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.181  0.181   0.181   0.181  0.181   0.181
```

```
```{r}
library(Metrics)
MAE= mae(Testdata$default, MSE_glm_lasso)
print(MAE)
```
```

```
[1] 0.2489951
```

### b) GLM for Classification Model using Important Variables

```
call:
roc.default(response = Testdata$default, predictor = predicts_glm)

Data: predicts_glm in 7132 controls (Testdata$default 0) < 821 cases (Testdata$default 1).
Area under the curve: 0.6713
```

### c) Simple Classification Model

```
call:
roc.default(response = Testdata$default, predictor = predicts_glm1)

Data: predicts_glm1 in 7132 controls (Testdata$default 0) < 821 cases (Testdata$default
1).
Area under the curve: 0.6788
```

This model gave a better ROC and hence we decided to go further with this one.

### d) Regression Model (when testing on Sample Test data)

```
##{r}
library(Metrics)
MAE11= mae(Testdata1$loss, predicts_4)
print(MAE11)
```

```
[1] 5.085637
```

After applying Threshold

```
Testdata2=Dataaa
Testdata3<-subset(Testdata2, predDefault>0.3)#This gave us the most reliable MAE

#Check for threshold
library(dplyr)
T1=subset(Testdata3, default>0)#Regression model using these number of people who
defaulted
nrow(T1)/nrow(Testdata3)##% of defaulters in regression model from classification model.

predicts_5<-predict(cvfit1, newdata= Testdata3 , s = "lambda.min")
library(Metrics)
MAEnew= mae(Testdata3$loss, predicts_5)
MAEnew

#Output on Sample Tesdata
Testdata3$Predloss=abs(predicts_5)
c2=Testdata3[,c("X", "loss", "Predloss")]
```

```
[1] 2.692631
```

This was the best MAE that we got.



## Insights and Conclusions

The very fact that the data did not have the real attribute names, we cannot conclude that which variable was important for the prediction i.e if income was a factor, ROI was a factor etc. The data cleaning process was extremely important in a large data set like this. It helped us determine that most of the data was irrelevant to the target variable and they needed to be dealt with.

The lasso Regression model did help us understand that some of these variables were important. The reason behind not using the classification model made from important variables predicted from Lasso was that Lasso gave us different important variables during each execution. Most of these were some but some differed. Another problem was that every time lasso gave different features, it was manually time consuming to again extract those variables and apply on the model. Every time that new features were used, the roc was changing too.

We then built two models; first was using important features of default for classification and for loss for regression, the second were basic models without using any feature extraction. To our surprise, the MAE for both came the same.

```
> print(MAE112)
[1] 5.085637
> print(MAE11)
[1] 5.085637
```

We then we adapted to the basic glm classification model to predict the loss binary( shown as 'default' in our code) in the test set a Lasso Regression model helped us determine the severity of the loss (named as 'Defaults' in the code).

The interesting learning from this project was the connection between the Classification and Regression Model. The threshold helped us decide how many of the probabilities of default were ideal(that were actually correct) for those who were actually defaulting.

Applying both models on the Main Test data (My.TestData in the code) helped us show who was defaulting and by how much. Around 272 ids were defaulting in the Test\_\_no\_lossv3.csv.

Future Work includes further extending the use of important variables for both Default and Loss Prediction using other algorithms like neural networks, RandomForest etc.

We are extremely thankful to our Professor for giving us this project which helped us apply various concepts and learn a lot more throughout the project.