Assignment 2

Models for Investment Decisions in Lending Club Loans

IDS 572 | CRN 38886 | Fall 2021

Team

Giovanni Alvin Prasetya (gprase2@uic.edu)
Karishma Mulchandani (kmulch4@uic.edu)
Rajaram Ramesh (rrames8@uic.edu)

Q.1(a) Develop boosted tree models (using either gbm or xgBoost) to predict loan_status. Experiment with different parameters using a grid of parameter values. Use cross-validation. Explain the rationale for your experimentation. How does performance vary with parameters, and which parameter setting you use for the 'best' model. Model performance should be evaluated through use of same set of criteria as for the earlier models - confusion matrix based, ROC analyses and AUC, cost-based performance. Provide a table with comparative evaluation of all the best models from each methods; show their ROC curves in a combined plot. Also provide profit-curves and 'best' profit' and associated cutoff. At this cutoff, what are the accuracy values for the different models?

We have used the xgBoost boosted tree model for binary classification. The xgBoost models with manipulating various parameters, is determined wherein eta(learning rate), max_depth(maximum depth of a tree), subsample (subsample ratio of the training instance. Setting it to 0.5 means that xgBoost randomly collected half of the data instances to grow trees and this will prevent overfitting), lambda(regularization term). Each model is trained and by making various changes in its parameters, auc performance is measured and best prediction is estimated and plotted.

Model 1 has the highest accuracy at 86.35% and the best iteration at the 174^{th} sample. Thus, it can be considered as the 'best' model

Model	**Accuracy**
Model 1	86.35%
Model 2	86.18%
Model 3	86.33%
Model 4	86.28%
Model 5	86.35%
Model 6	86.31%
Model 7	86.31%
Model 8	86.22%
Model 9	86.34%

R Code and Output:

```
lcdf <- read.csv("lcData100K.csv")
dim(lcdf)

#Drop variables with 100% NA values
lcdf <- lcdf %>% select_if(function(x){!all(is.na(x))})
dim(lcdf)
```

```
#Columns where there are missing values
colMeans(is.na(lcdf))[colMeans(is.na(lcdf))>0]
dim(lcdf)

#Remove variables which have more than 60% missing values
colMeans(is.na(lcdf))>0.6

finalnona<-names(lcdf)[colMeans(is.na(lcdf))>0.6]
final_lcdf <- lcdf %>% select(-finalnona)
dim(final_lcdf)

#Columns with remaining missing values
colMeans(is.na(final_lcdf))[colMeans(is.na(final_lcdf))>0]

#Summary of data in these columns final_lcdf
nm<- names(final_lcdf)[colSums(is.na(final_lcdf))>0]
summary(final_lcdf[, nm])

#Replace missing values with some value
NoNAlcdf <- final_lcdf %>% replace_na(list(mths_since_last_delinq=500,
revol_util=median(final_lcdf$revol_util, na.rm=TRUE),
bc_open_to_buy=median(final_lcdf$bc_open_to_buy, na.rm=TRUE), mo_sin_old_il_acct=1000,
mths_since_recent_bc=1000, mths_since_recent_inq=50, num_tl_120dpd_2m =
median(lcdf$num_tl_120dpd_2m, na.rm=TRUE),percent_bc_gt_75 =
median(final_lcdf$percent_bc_gt_75, na.rm=TRUE), bc_util=median(final_lcdf$bc_util, na.rm=TRUE),
avg_cur_bal=median(final_lcdf$avg_cur_bal,na.rm = TRUE),
num_rev_accts=mean(final_lcdf$num_rev_accts,na.rm = TRUE),
emp_length=median(final_lcdf$emp_length,na.rm = TRUE),
pct_tl_nvr_dlq=mean(final_lcdf$pct_tl_nvr_dlq, na.rm = TRUE)))

#To check if we have no more NA values
colMeans(is.na(NoNAlcdf))[colMeans(is.na(NoNAlcdf))>0]
dim(NoNAlcdf)

```

Dropping variables which cause data leakage
```{r}
varsOmit <- c("issue_d","last_pymnt_d", "zip_code", "emp_title", "last_credit_pull_d", "pymnt_plan",
"addr_state", "policy_code", "disbursement_method", "title", "term", "funded_amnt_inv", "out_prncp",
"out_prncp_inv", "total_pymnt_inv", "total_rec_prncp", "total_rec_int", "debt_settlement_flag",
"hardship_flag", "application_type", "last_pymnt_amnt", "last_pymnt_d", "funded_amnt_inv",
"mths_since_last_delinq", "last_pymnt_amnt", "total_pymnt", "issue_d", "funded_amnt", "last_pymnt_d",
"recoveries", "num_tl_op_past_12m", "collection_recovery_fee", "total_rec_late_fee",
"num_tl_120dpd_2m", "num_tl_30dpd", "num_tl_90g_dpd_24m", "earliest_cr_line",
"num_tl_op_past_12m", "earliest_cr_line")

mydata <- NoNAlcdf %>% select(-varsOmit)

#Change chr to factors:
mydata$grade <- factor(mydata$grade, levels=c("A", "B","C","D", "E","F","G"))
```

````
mydata$sub_grade <- factor(mydata$sub_grade, levels=c("A1", "A2", "A3", "A4", "A5", "B1", "B2",
"B3", "B4", "B5", "C1", "C2", "C3", "C4", "C5", "D1", "D2", "D3", "D4", "D5", "E1", "E2", "E3", "E4",
"E5", "F1", "F2", "F3", "F4", "F5", "G1", "G2", "G3", "G4", "G5"))

mydata$initial_list_status <- factor(mydata$initial_list_status, levels=c("w", "f"))

mydata$loan_status <- factor(mydata$loan_status, levels=c("Fully Paid", "Charged Off"))

mydata$emp_length <- factor(mydata$emp_length, levels=c("n/a", "< 1 year","1 year","2 years", "3
years" ,  "4 years",   "5 years",   "6 years",   "7 years" ,  "8 years", "9 years", "10+ years" ))

mydata$purpose <- fct_recode(mydata$purpose)

mydata$home_ownership <- as.factor((mydata$home_ownership))
mydata$verification_status<- as.factor(mydata$verification_status)

dim(mydata)

```


Split Train and Test Data
```{r }
#Split the data into trn, tst subsets
nr=nrow(mydata)
mydata
trnIndex = sample(1:nr, size = round(0.7*nr), replace=FALSE)
lcdfTrn=mydata[trnIndex,]
lcdfTst = mydata[-trnIndex,]

dim(lcdfTrn)
dim(lcdfTst)

str(lcdfTrn)

```


```{r}
set.seed(12345)

mydata2<-mydata
fdum<-dummyVars(~.,data=mydata2 %>% select(-loan_status))
dxlcdf <- predict(fdum, mydata2)
dylcdf <- class2ind(mydata2$loan_status, drop2nd = FALSE)
# and then decide which one to keep
fplcdf <- dylcdf [ , 1] # or,
colcdf <- dylcdf [ , 2]

#Training subsets
dxlcdfTrn <- dxlcdf[trnIndex,]
colcdfTrn <- colcdf[trnIndex]
````

```
dxlcdfTst <- dxlcdf[-trnIndex,]
colcdfTst <- colcdf[-trnIndex]
dxTrn <- xgb.DMatrix( dxlcdfTrn, label=colcdfTrn)
dxTst <- xgb.DMatrix(dxlcdfTst, label=colcdfTst)

#Model 1(xgb_lsm1) with eta = 0.01

#Add watchlist to watch the progress of learning through the performance on these data sets
xgbWatchlist <- list(train = dxTrn,eval = dxTst)

#Training the model and getting predictions
#List of parameters
xgbParam <- list (max_depth = 5, eta = 0.01,
objective = "binary:logistic",
eval_metric="error", eval_metric = "auc")

#Iterative predictions
require(xgboost)
xgb_lsM1 <- xgb.train ( xgbParam, dxTrn, nrounds = 500, xgbWatchlist, early_stopping_rounds = 10)
xgb_lsM1$best_iteration
xpredTrg1<-predict(xgb_lsM1, dxTst) # best_iteration is used
head(xpredTrg1)
```

```
[1]     train-error:0.138229    train-auc:0.677140      eval-error:0.136900     eval-
auc:0.667923
Multiple eval metrics are present. Will use eval_auc for early stopping.
Will train until eval_auc hasn't improved in 10 rounds.

[2]     train-error:0.138229    train-auc:0.677213      eval-error:0.136900     eval-
auc:0.668132
[3]     train-error:0.138300    train-auc:0.678921      eval-error:0.136833     eval-
auc:0.668712
[4]     train-error:0.138286    train-auc:0.678690      eval-error:0.136767     eval-
auc:0.668744
[5]     train-error:0.138343    train-auc:0.679281      eval-error:0.136833     eval-
auc:0.668747
[6]     train-error:0.138243    train-auc:0.680109      eval-error:0.136733     eval-
auc:0.669357
[7]     train-error:0.138471    train-auc:0.681671      eval-error:0.136467     eval-
auc:0.670092
[8]     train-error:0.138357    train-auc:0.681719      eval-error:0.136533     eval-
auc:0.670076
```

```
[1] 500
[1] 0.13621391 0.11729860 0.14232948 0.08502362 0.14578173 0.10427569
```

```
#Cross-validation
xgbParam <- list (
max_depth = 3, eta = 0.1,
objective = "binary:logistic",
eval_metric="error", eval_metric = "auc")
xgb_lscv <- xgb.cv( xgbParam, dxTrn, nrounds = 500, nfold=5, early_stopping_rounds = 10 )
```

#Best iteration
xgb_lscv$best_iteration

#or for the best iteration based on performance measure (among those specified in xgbParam)
best_cvIter <- which.max(xgb_lscv$evaluation_log$test_auc_mean)

#Learn the best model without xval
xgb_lsbest <- xgb.train(xgbParam, dxTrn, nrounds = xgb_lscv$best_iteration)

#Variable importance
xgb.importance(model = xgb_lsbest) %>% view()

```

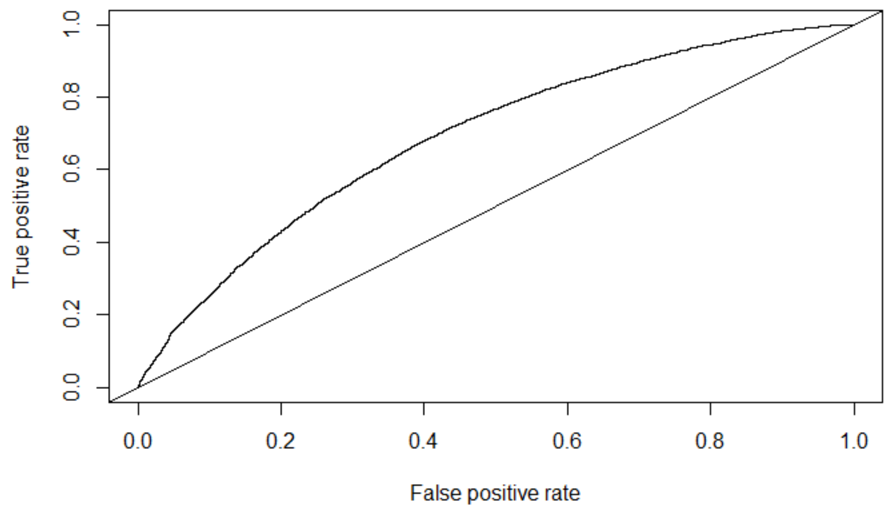
```
error:0.138357+0.003436 test-auc:0.691003+0.005166
Stopping. Best iteration:
[174]   train-error:0.137586+0.000904   train-auc:0.732310+0.001338   test-
error:0.138257+0.003467 test-auc:0.691050+0.005213

[1] 174
```

```{r}
#Performance on test data for model 1
require(ROCR)
pred_xgb_lsM1=prediction(xpredTrg1,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged Off"))
aucPerf_xgb_lsM1=performance(pred_xgb_lsM1, "tpr", "fpr")
plot(aucPerf_xgb_lsM1)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg1>0.5), act=colcdfTst)

```
    act
pred     0     1
   0 25906  4092
   1     1     1
```



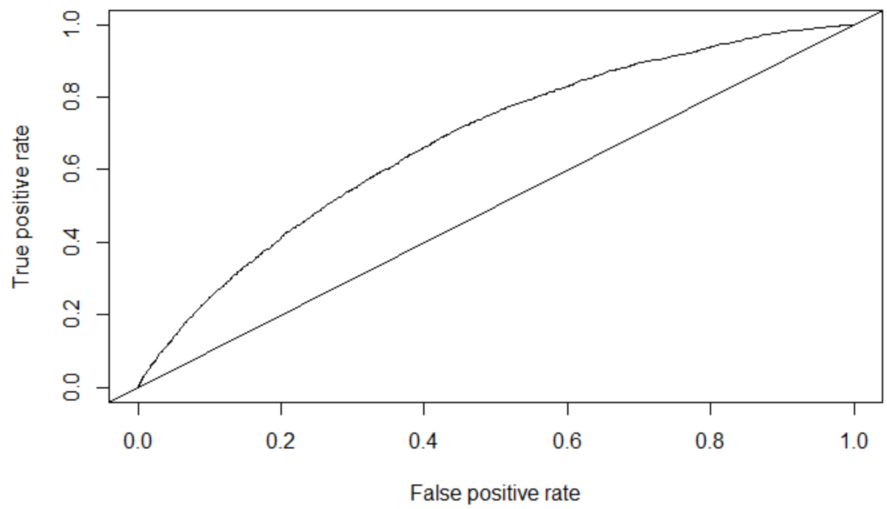
xgbParam <- list (
max_depth = 4,
objective = "binary:logistic",
eval_metric="error", eval_metric = "auc")

#Model 2 with eta = 1
xgb_lsM2 <- xgb.train(xgbParam, dxTrn, nrounds = 500,xgbWatchlist, early_stopping_rounds = 10, eta=1)
xgb_lsM2$best_iteration
xpredTrg2<-predict(xgb_lsM2, dxTst)

#Performance on test data for model 2
pred_xgb_lsM2=prediction(xpredTrg2,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged Off"))
aucPerf_xgb_lsM2=performance(pred_xgb_lsM2, "tpr", "fpr")
plot(aucPerf_xgb_lsM2)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg2>0.5), act=colcdfTst)

```
      act
pred      0     1
   0 25856  4052
   1    51    41
```

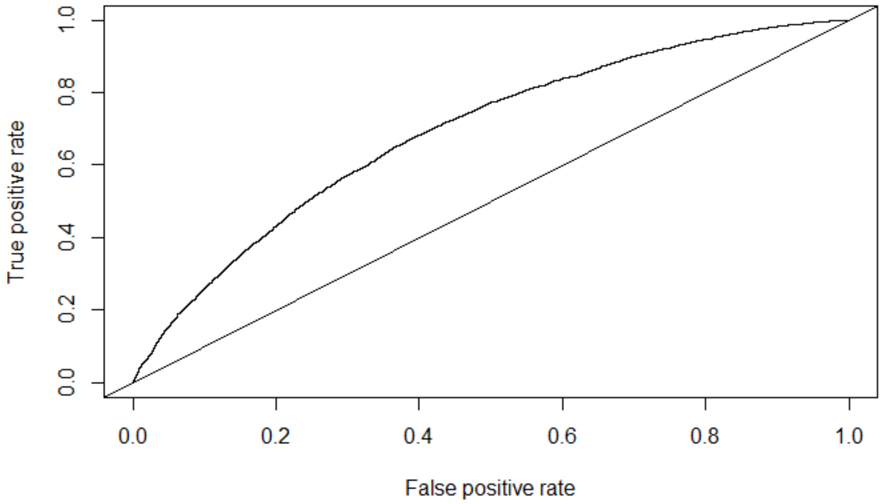



```
#Model 3 with eta = 0.1
xgb_lsM3 <- xgb.train( xgbParam, dxTrn, nrounds = 500,
xgbWatchlist, early_stopping_rounds = 10, eta=0.1 )
xgb_lsM3$best_iteration
xpredTrg3<-predict(xgb_lsM3, dxTst)

#Performance on test data for model 3
pred_xgb_lsM3=prediction(xpredTrg3,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged
Off"))
aucPerf_xgb_lsM3=performance(pred_xgb_lsM3, "tpr", "fpr")
plot(aucPerf_xgb_lsM3)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg3>0.5), act=colcdfTst)
```

```
     act
pred     0     1
   0 25901  4088
   1     6     5
```

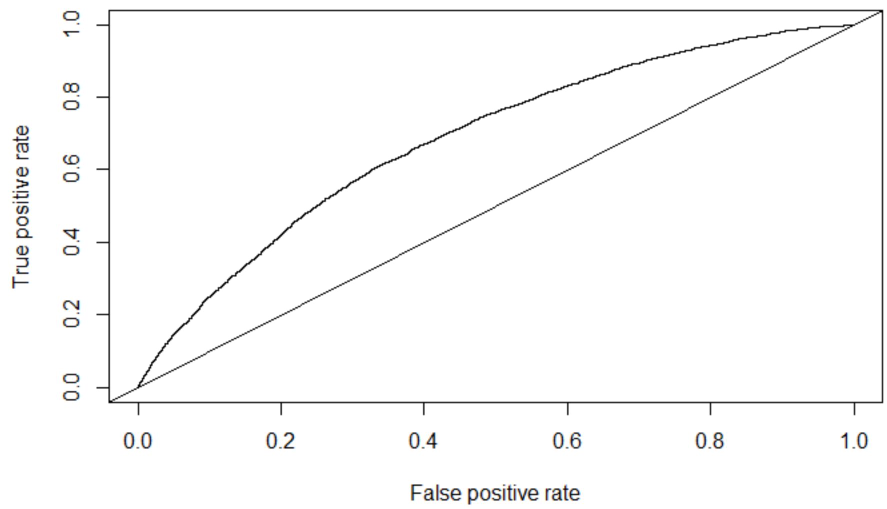


#Model 4 with eta = 0.5
xgb_lsM4 <- xgb.train(xgbParam, dxTrn, nrounds = 500,
xgbWatchlist, early_stopping_rounds = 10, eta=0.5)
xgb_lsM4$best_iteration
xpredTrg4<-predict(xgb_lsM4, dxTst)

#Performance on test data for model 4
pred_xgb_lsM4=prediction(xpredTrg4,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged Off"))
aucPerf_xgb_lsM4=performance(pred_xgb_lsM4, "tpr", "fpr")
plot(aucPerf_xgb_lsM4)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg4>0.5), act=colcdfTst)

```
     act
pred      0     1
   0 25885  4079
   1    22    14
```

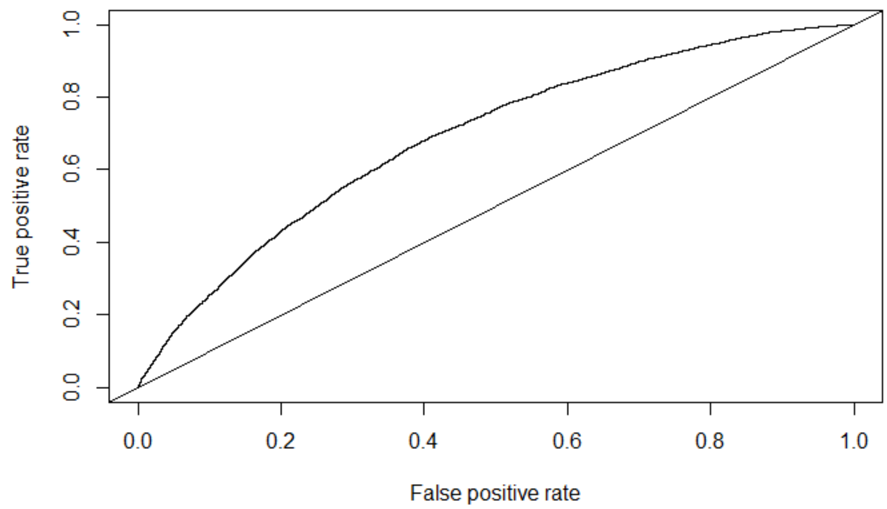



```
#Model 5 with eta = 0.01
xgb_lsM5 <- xgb.train( xgbParam, dxTrn, nrounds = 500,
xgbWatchlist, early_stopping_rounds = 10, eta=0.01 )
xgb_lsM5$best_iteration
xpredTrg5<-predict(xgb_lsM5, dxTst)

#Performance on test data for model 5
pred_xgb_lsM5=prediction(xpredTrg5,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged
Off"))
aucPerf_xgb_lsM5=performance(pred_xgb_lsM5, "tpr", "fpr")
plot(aucPerf_xgb_lsM5)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg5>0.5), act=colcdfTst)
```

```
     act
pred      0     1
   0 25906  4093
   1     1     0
```

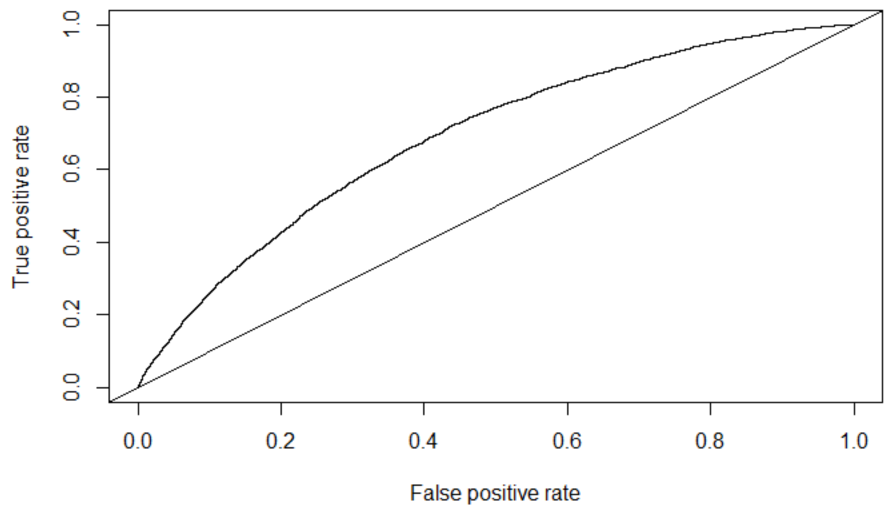



```
#Model 6 with eta = 0.1, max_depth=0.6
xgbParam <- list (
max_depth = 6,
objective = "binary:logistic",
eval_metric="error", eval_metric = "auc")
xgb_lsM6 <- xgb.train( xgbParam, dxTrn, nrounds = 500, xgbWatchlist,
early_stopping_rounds = 10, eta=0.1 )
xgb_lsM6$best_iteration
xpredTrg6<-predict(xgb_lsM6, dxTst)

#Performance on test data for model 6
pred_xgb_lsM6=prediction(xpredTrg6,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged
Off"))
aucPerf_xgb_lsM6=performance(pred_xgb_lsM6, "tpr", "fpr")
plot(aucPerf_xgb_lsM6)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg6>0.5), act=colcdfTst)
```

```
     act
pred      0     1
   0  25893  4073
   1     14    20
```

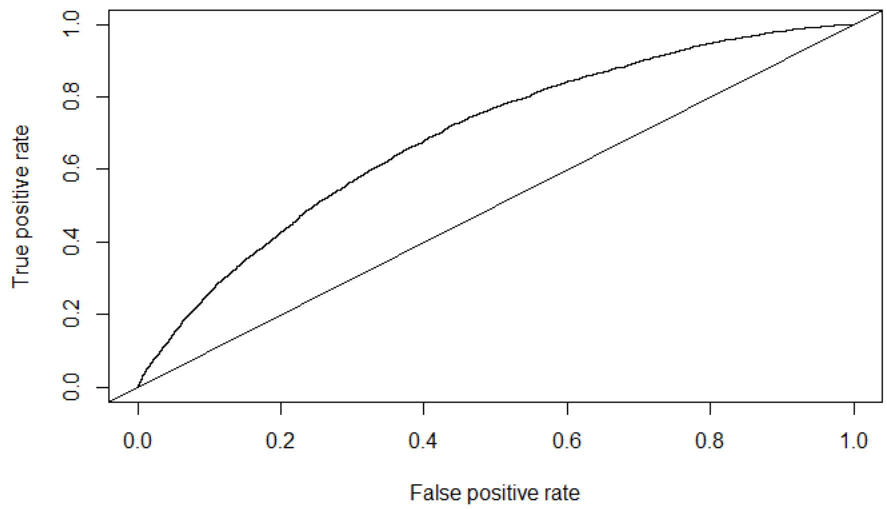



```
#Model 7 same as 6 but with nrounds = 1000
xgb_lsM7 <- xgb.train( xgbParam, dxTrn, nrounds = 1000, xgbWatchlist,
early_stopping_rounds = 10, eta=0.1)
xgb_lsM7$best_iteration
xpredTrg7<-predict(xgb_lsM7, dxTst)

#Performance on test data for model 7
pred_xgb_lsM7=prediction(xpredTrg7,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged
Off"))
aucPerf_xgb_lsM7=performance(pred_xgb_lsM7, "tpr", "fpr")
plot(aucPerf_xgb_lsM7)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg7>0.5), act=colcdfTst)
```

```
     act
pred      0     1
   0 25893  4073
   1    14    20
```

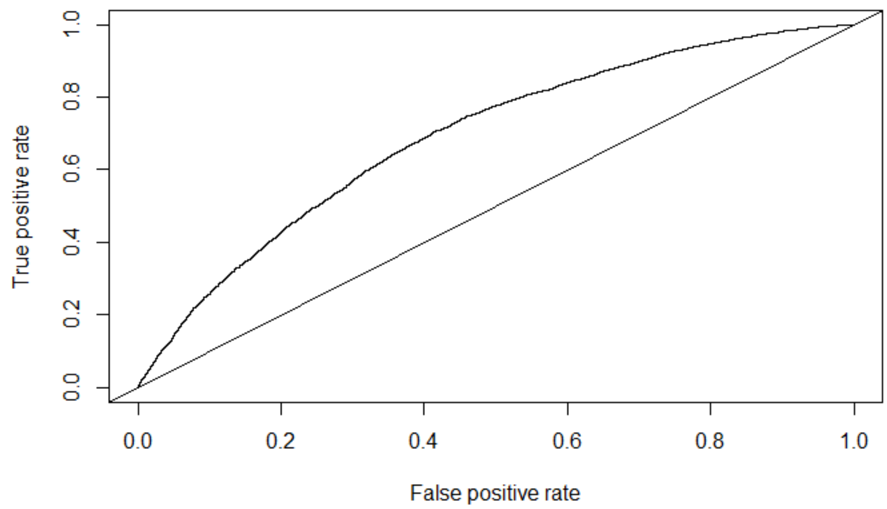



```
#Model 8 same as 7 but with lambda=0.05, subsample=0.7, colsample_bytree=0.5
xgb_lsM8 <- xgb.train( xgbParam, dxTrn, nrounds = 1000, xgbWatchlist, early_stopping_rounds = 10, eta=0.1, lambda=0.05, subsample=0.7, colsample_bytree=0.5 )
xgb_lsM8$best_iteration
xpredTrg8<-predict(xgb_lsM8, dxTst)

#Performance on test data for model 8
pred_xgb_lsM8=prediction(xpredTrg8,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged Off"))
aucPerf_xgb_lsM8=performance(pred_xgb_lsM8, "tpr", "fpr")
plot(aucPerf_xgb_lsM8)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg8>0.5), act=colcdfTst)
```

```
     act
pred      0     1
   0 25868  4062
   1    39    31
```

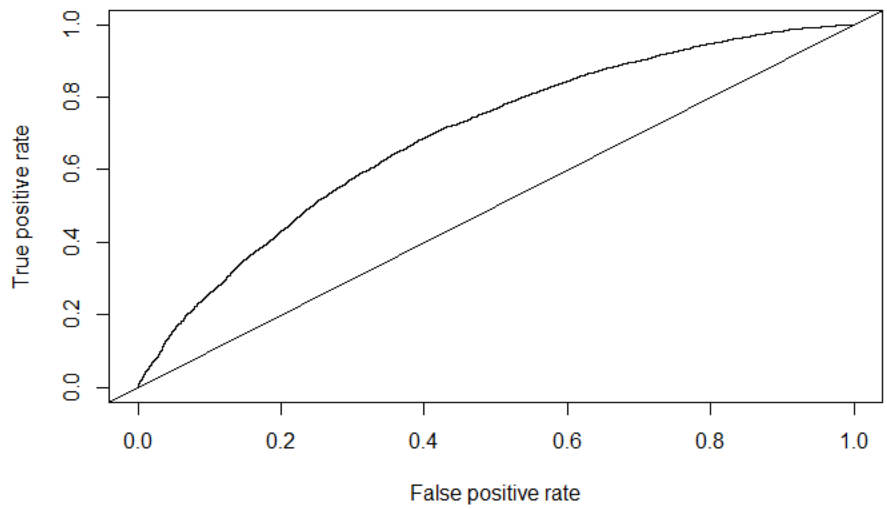



```
#Model 9 same as 8 but with eta=0.01
xgb_lsM9 <- xgb.train( xgbParam, dxTrn, nrounds = 1000, xgbWatchlist, early_stopping_rounds
= 10, eta=0.01, subsample=0.7, colsample_bytree=0.5 )
xgb_lsM9$best_iteration
xpredTrg9<-predict(xgb_lsM9, dxTst)

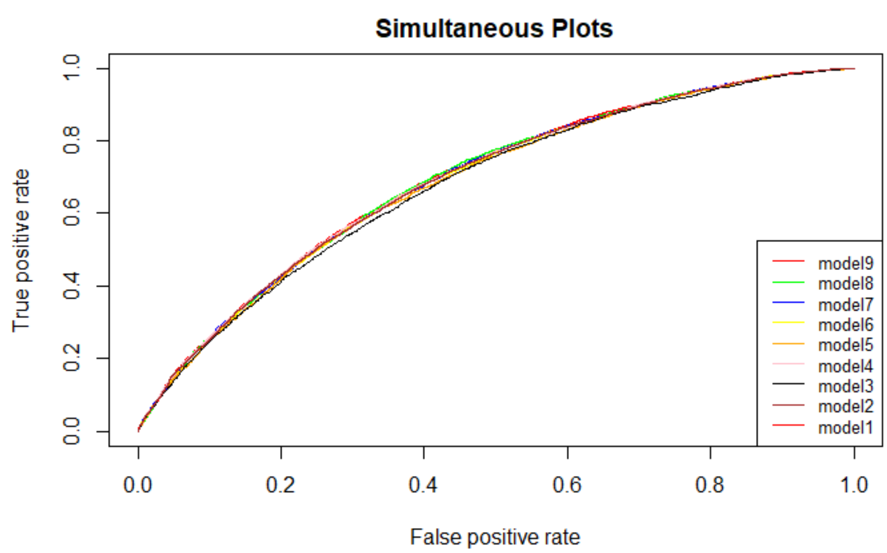
#Performance on test data for model 9
pred_xgb_lsM9=prediction(xpredTrg9,lcdfTst$loan_status,label.ordering = c("Fully Paid", "Charged
Off"))
aucPerf_xgb_lsM9=performance(pred_xgb_lsM9, "tpr", "fpr")
plot(aucPerf_xgb_lsM9)
abline(a=0, b= 1)

#Confusion matrix
table(pred=as.numeric(xpredTrg9>0.5), act=colcdfTst)
```

```
     act
pred     0     1
   0 25903  4090
   1     4     3
```




```
Combination of all Plots
```{r}
plot(aucPerf_xgb_lsM9, col="red", main = "Simultaneous Plots", cex = 0.6)
plot(aucPerf_xgb_lsM8, col="green", add=TRUE)
plot(aucPerf_xgb_lsM6, col="blue", add=TRUE)
plot(aucPerf_xgb_lsM5, col="yellow", add=TRUE)
plot(aucPerf_xgb_lsM4, col="orange", add=TRUE)
plot(aucPerf_xgb_lsM3, col="pink", add=TRUE)
plot(aucPerf_xgb_lsM2, col="black", add=TRUE)
plot(aucPerf_xgb_lsM1, col="brown", add=TRUE)
legend("bottomright",
c("model9","model8","model7","model6","model5","model4","model3","model2","model1" ), lty=1,
col=c("red","green","blue","yellow","orange","pink","black","brown"), cex = 0.8)
```


```

Q.2(a) Develop linear (glm) models to predict loan_status. Experiment with different parameter values, and identify which gives 'best' performance. Use cross-validation. Describe how you determine 'best' performance. How do you handle variable selection? Experiment with Ridge and Lasso, and show how you vary these parameters, and what performance is observed.

Q.2(b) For the linear model, what is the loss function, and link function you use? (Write the expression for these, and briefly describe).

Q.2(c) Compare performance of models with that of random forests (from last assignment) and gradient boosted tree models.

Q.2(d) Examine which variables are found to be important by the best models from the different methods, and comment on similarities, difference. What do you conclude?

Q.2(e) In developing models above, do you find larger training samples to give better models? Do you find balancing the training data examples across classes to give better models?

R Code and Output:

Linear model turning into factor

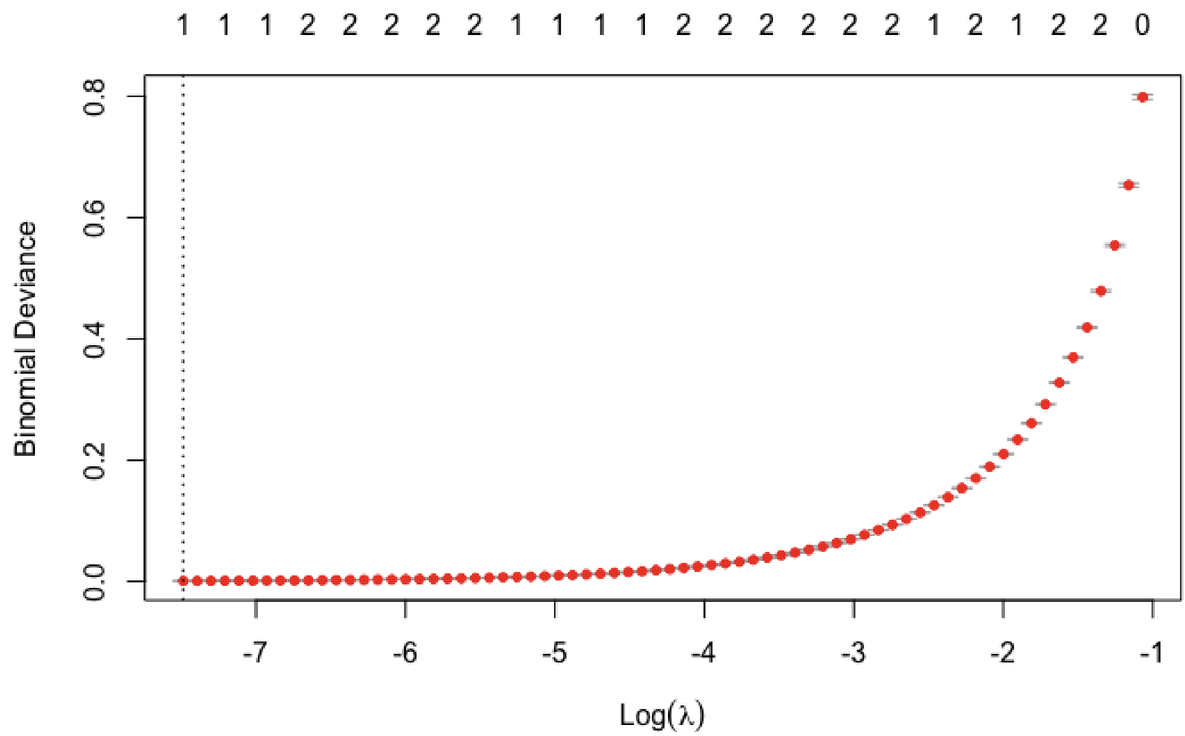
```
#Make sure that "fully paid" is 1 in the factor variable
levels(lcdfTrn_AR$loan_status)

[1] "Charged Off" "Fully Paid"

yTrn<-factor(if_else(lcdfTrn_AR$loan_status=="Fully Paid", '1', '0') )
xDTrn<-model.matrix( ~ loan_status+ actualTerm + annRet + actualReturn - 1, lcdfTrn_AR)
#the Test set
glmls_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial")
```

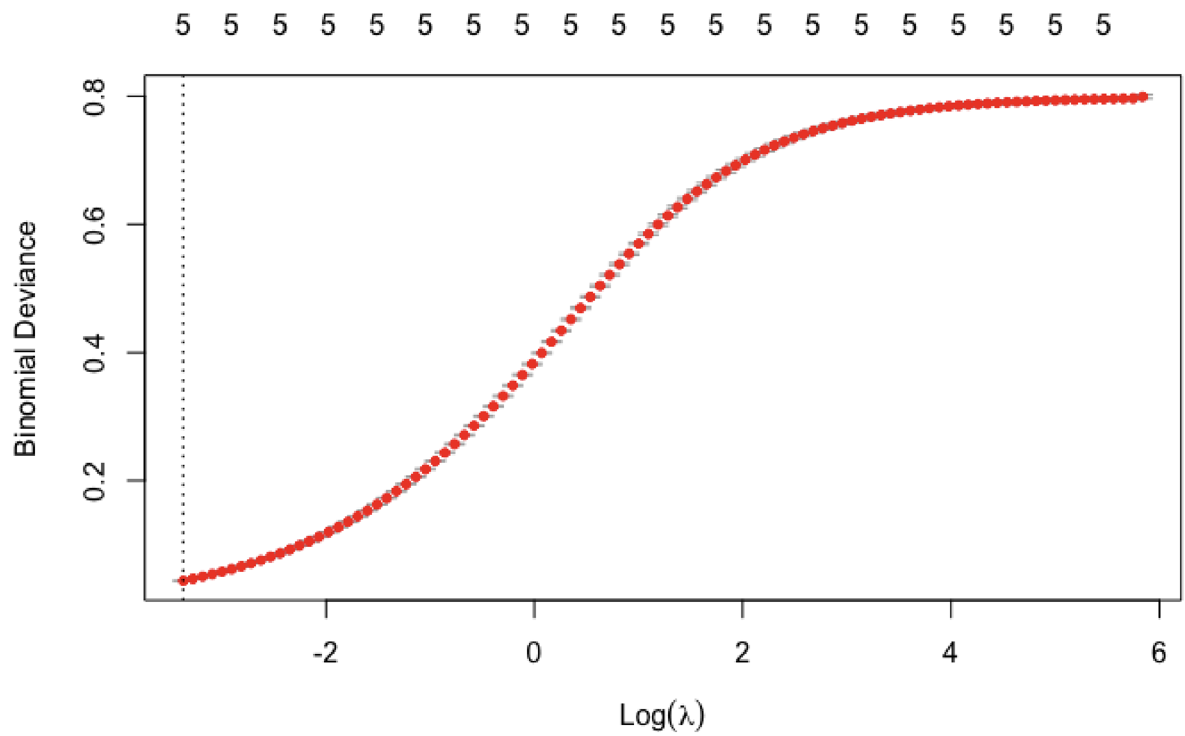
Removing variables and running cv

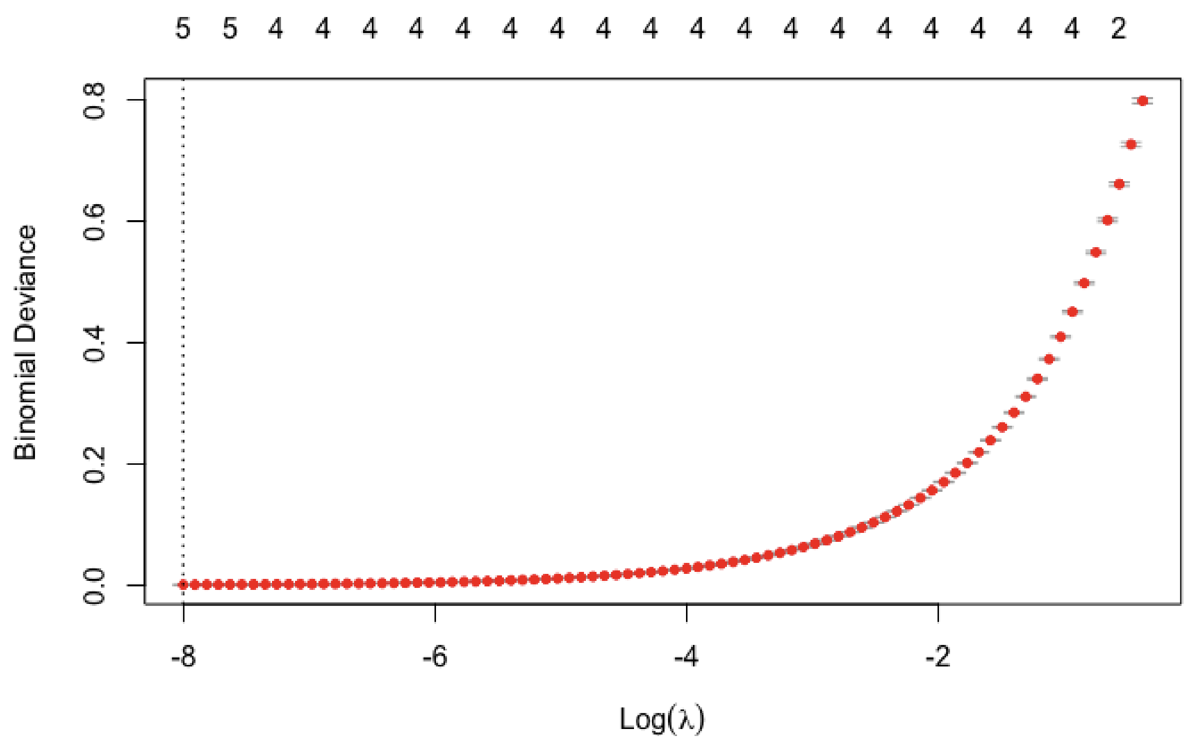
```
#Remove variables that would not be x variables
xDTrn<-model.matrix( ~ loan_status+ actualTerm + annRet + actualReturn - 1, lcdfTrn_AR)
#Running cross validation
glmls_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial")
#Plotting the model
plot(glmls_cv)
```



Ridge and Lasso
#Experimenting with Ridge
glmls_Ridge<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", alpha=0)
plot(glmls_Ridge)
#Experimenting between Ridge and Lasso
glmls_Mid<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", alpha=.5)
plot(glmls_Mid)
#Finding the minimum lambda
glmls_cv$lambda.min
[1] 0.0005604592

#Finding within 1 standard error
glmls_cv$lambda.1se
[1] 0.0005604592





Getting lambda values
#Values for all coefficients
coef(glmls_cv, s = glmls_cv$lambda.min)

```
6 x 1 sparse Matrix of class "dgCMatrix"
                                  s1
(Intercept)                 8.404607
loan_statusCharged Off    -14.967957
loan_statusFully Paid          .
actualTerm                     .
annRet                         .
actualReturn                   .
```

#Showing coefficients using tidy getting rid of all zeros
tidy(coef(glmls_cv, s = glmls_cv$lambda.1se))
Warning: 'tidy.dgCMatrix' is deprecated.
See help("Deprecated")
Warning: 'tidy.dgTMatrix' is deprecated.
See help("Deprecated")

#Values corresponding to graph
glmls_cv$glmnet.fit

Call: glmnet(x = data.matrix(xDTrn), y = yTrn, family = "binomial")

```
   Df  %Dev  Lambda
1   0  0.00 0.34390
2   1 18.26 0.31330
3   2 30.68 0.28550
4   2 40.08 0.26010
5   2 47.59 0.23700
6   1 53.78 0.21600
7   2 59.01 0.19680
8   2 63.49 0.17930
9   2 67.36 0.16340
10  1 70.75 0.14890
11  2 73.73 0.13560
12  1 76.36 0.12360
13  2 78.69 0.11260
14  2 80.77 0.10260
15  2 82.63 0.09349
16  1 84.29 0.08518
17  2 85.78 0.07762
18  2 87.12 0.07072
19  2 88.33 0.06444
20  2 89.42 0.05871
21  2 90.40 0.05350
22  2 91.29 0.04875
23  2 92.09 0.04442
```

24	2	92.82	0.04047
25	2	93.47	0.03687
26	2	94.07	0.03360
27	2	94.61	0.03061
28	2	95.10	0.02789
29	2	95.54	0.02542
30	1	95.94	0.02316
31	2	96.31	0.02110
32	2	96.64	0.01923
33	2	96.94	0.01752
34	2	97.22	0.01596
35	2	97.47	0.01454
36	1	97.70	0.01325
37	1	97.90	0.01207
38	1	98.09	0.01100
39	2	98.26	0.01002
40	1	98.42	0.00913
41	1	98.56	0.00832
42	2	98.69	0.00758
43	1	98.80	0.00691
44	2	98.91	0.00630
45	2	99.01	0.00574
46	1	99.10	0.00523
47	2	99.18	0.00476
48	1	99.25	0.00434
49	2	99.32	0.00395
50	2	99.38	0.00360
51	1	99.43	0.00328
52	2	99.48	0.00299
53	2	99.53	0.00272
54	1	99.57	0.00248
55	2	99.61	0.00226
56	2	99.64	0.00206
57	2	99.68	0.00188
58	2	99.70	0.00171
59	2	99.73	0.00156
60	2	99.75	0.00142
61	2	99.78	0.00130
62	2	99.80	0.00118
63	2	99.81	0.00108
64	1	99.83	0.00098
65	2	99.85	0.00089
66	2	99.86	0.00081
67	1	99.87	0.00074

68 2 99.88 0.00068
69 1 99.89 0.00062
70 1 99.90 0.00056

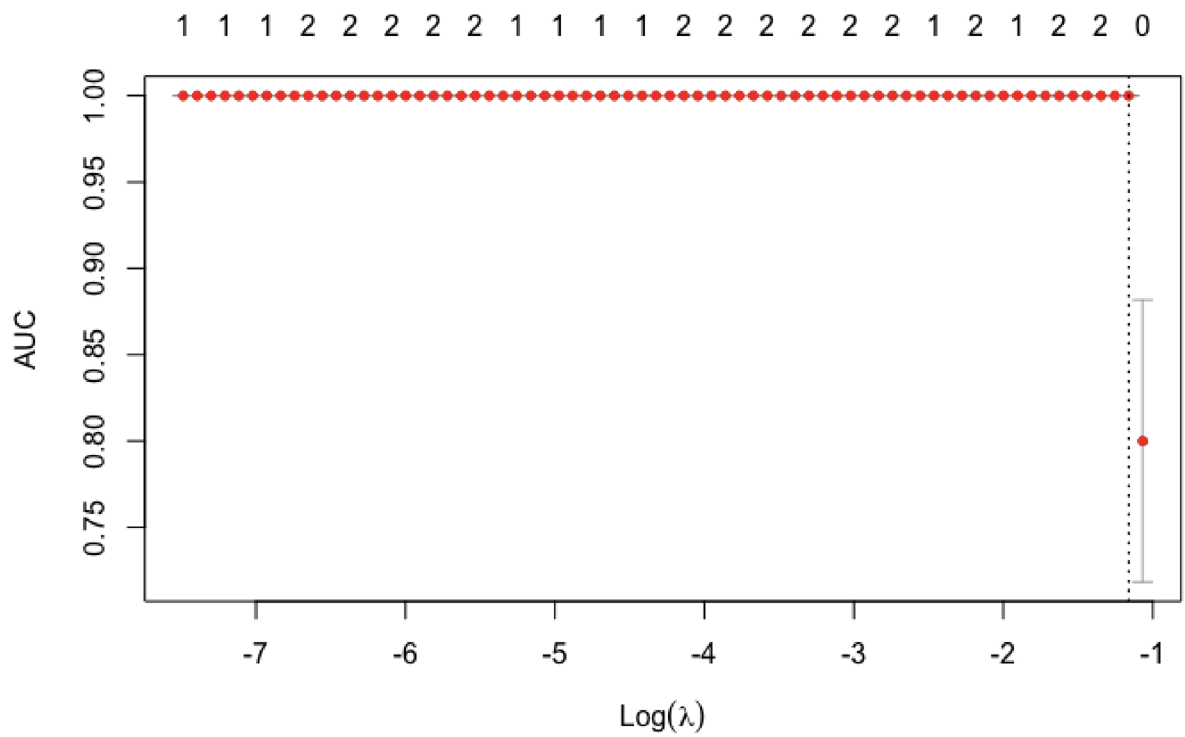
#Finding lambda that creates the optimal deviance
which(glmls_cv$lambda == glmls_cv$lambda.1se)
[1] 70

#Finding the lambda that has the best auc level
glmls_cv_auc<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", type.measure = "auc")
plot(glmls_cv_auc)

Description: df [2 x 3]

row <chr>	column <chr>	value <dbl>
(Intercept)	s1	8.404607
loan_statusCharged Off	s1	-14.967957

2 rows



Making Predictions
#Making predictions for Lambda min with Training and Test
glmPredls_Min=predict (glmls_cv,data.matrix(xDTrn), s="lambda.min")
head(glmPredls_Min)
 lambda.min

```
1   8.404607
2   8.404607
3   8.404607
4   8.404607
5   8.404607
6   8.404607

#Making predictions with probability for Lambda min Training
glmPredls_pMin=predict(glmls_cv,data.matrix(xDTrn), s="lambda.min", type="response" )
head(glmPredls_pMin)
  lambda.min
1  0.9997762
2  0.9997762
3  0.9997762
4  0.9997762
5  0.9997762
6  0.9997762

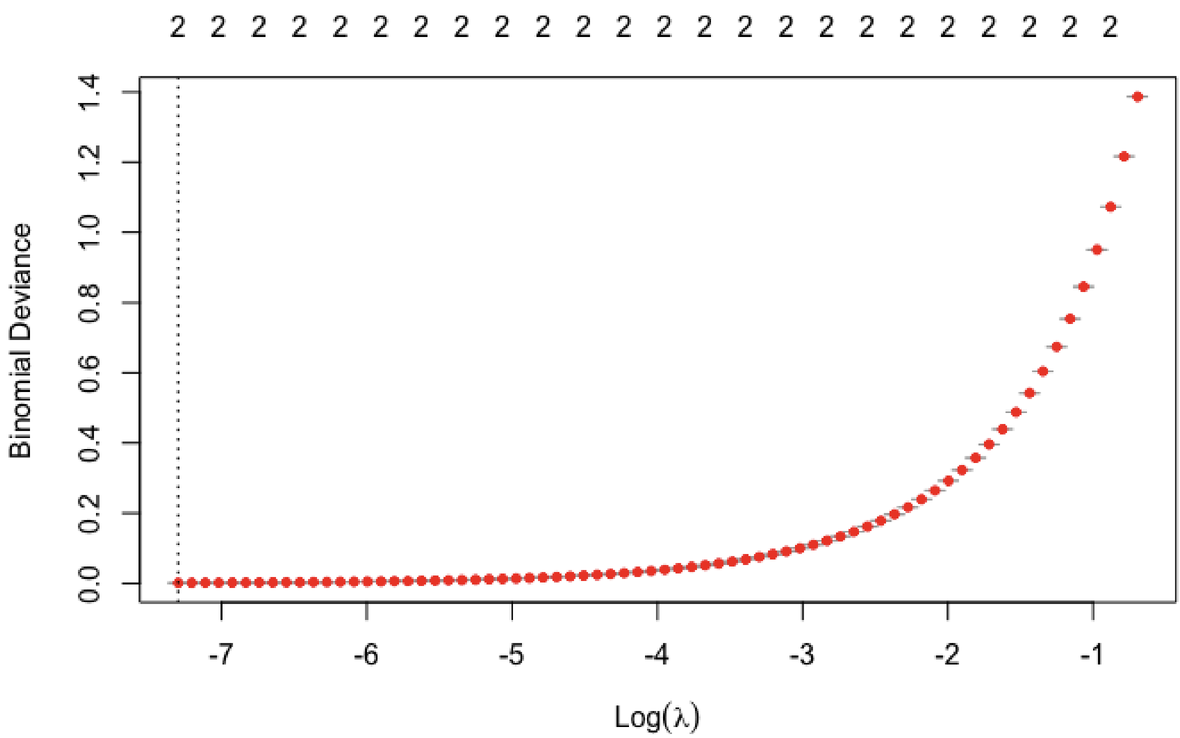
#Making predictions for Lambda.se
glmPredls_SE=predict ( glmls_cv,data.matrix(xDTrn), s="lambda.1se" )
head(glmPredls_SE)
  lambda.1se
1   8.404607
2   8.404607
3   8.404607
4   8.404607
5   8.404607
6   8.404607

#Making predictions with probability for Lambda.1se
glmPredls_pSE=predict(glmls_cv,data.matrix(xDTrn), s="lambda.1se", type="response" )
head(glmPredls_pSE)
  lambda.1se
1  0.9997762
2  0.9997762
3  0.9997762
4  0.9997762
5  0.9997762
6  0.9997762
```

Balancing data

```
#To consider a more balanced data, we can include example weights
wts=if_else(yTrn==0, 1-sum(yTrn==0)/length(yTrn), 1-sum(yTrn==1)/length(yTrn) )
glmsw_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family = "binomial", weights= wts )
```

plot(glmsw_cv)
predsauc <- prediction(glmPredls_pMin, lcdfTrn_AR$loan_status, label.ordering =
c("Charged Off", "Fully Paid"))
aucPerf <- performance(predsauc, "auc")



Model for standard error
glmls_1 <- glmnet(data.matrix(xDTrn), yTrn, family="binomial", lambda = glmls_cv$lambda.1se)
glmls_1

Call: glmnet(x = data.matrix(xDTrn), y = yTrn, family = "binomial", lambda = glmls_cv$lambda.1se)

	Df	%Dev	Lambda
1	1	99.9	0.0005605

tidy(glmls_1)
glmls_1b <- glmnet(data.matrix(xDTrn), yTrn, family="binomial",lambda = glmls_cv$lambda)
tidy(coef(glmls_1b, s= glmls_cv$lambda.1se))

A tibble: 2 × 5

term <chr>	step <dbl>	estimate <dbl>	lambda <dbl>	dev.ratio <dbl>
(Intercept)	1	8.406192	0.0005604592	0.9990345
loan_statusCharged Off	1	-14.971129	0.0005604592	0.9990345

2 rows

Description: df [2 × 3]

row <chr>	column <chr>	value <dbl>
(Intercept)	s1	8.404607
loan_statusCharged Off	s1	-14.967957

2 rows

Q.3 Develop models to identify loans which provide the best returns. Explain how you define returns? Does it include Lending Club's service costs? Develop glm, rf, gbm/xgb models for this. Show how you systematically experiment with different parameters to find the best models. Compare model performance – explain what performance criteria do you use, and why.

R Code and Output

Building a Random Forest Model to Predict Actual Returns

rf_AR <- ranger(actualReturn ~., data=subset(lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status)), num.trees = 200, importance = 'permutation')

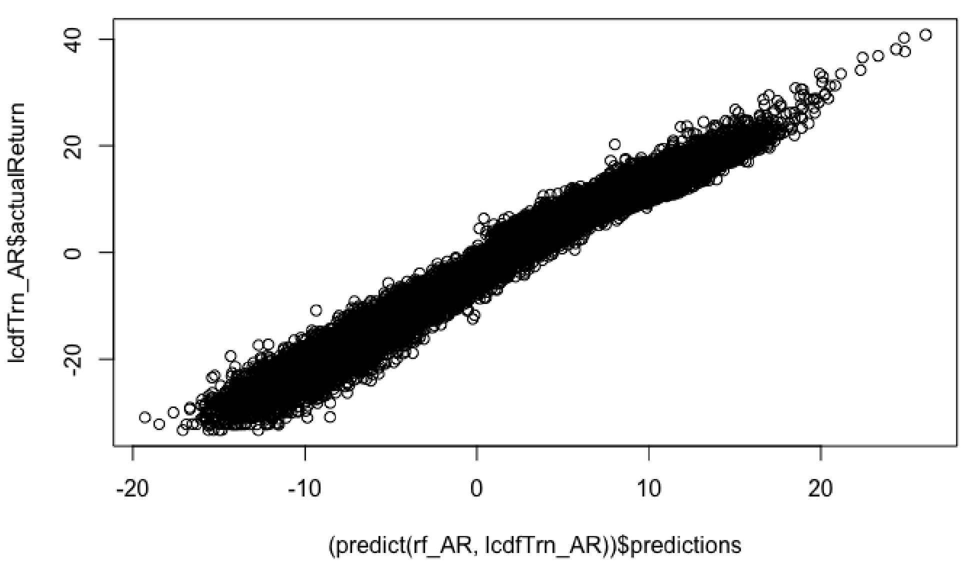
RF Training data

rfPredRet_trn_AR<- predict(rf_AR, lcdfTrn_AR)

sqrt(mean((rfPredRet_trn_AR$predictions- lcdfTrn_AR$actualReturn)^2))

plot ((predict(rf_AR, lcdfTrn_AR))$predictions, lcdfTrn_AR$actualReturn)

[1] 3.703739



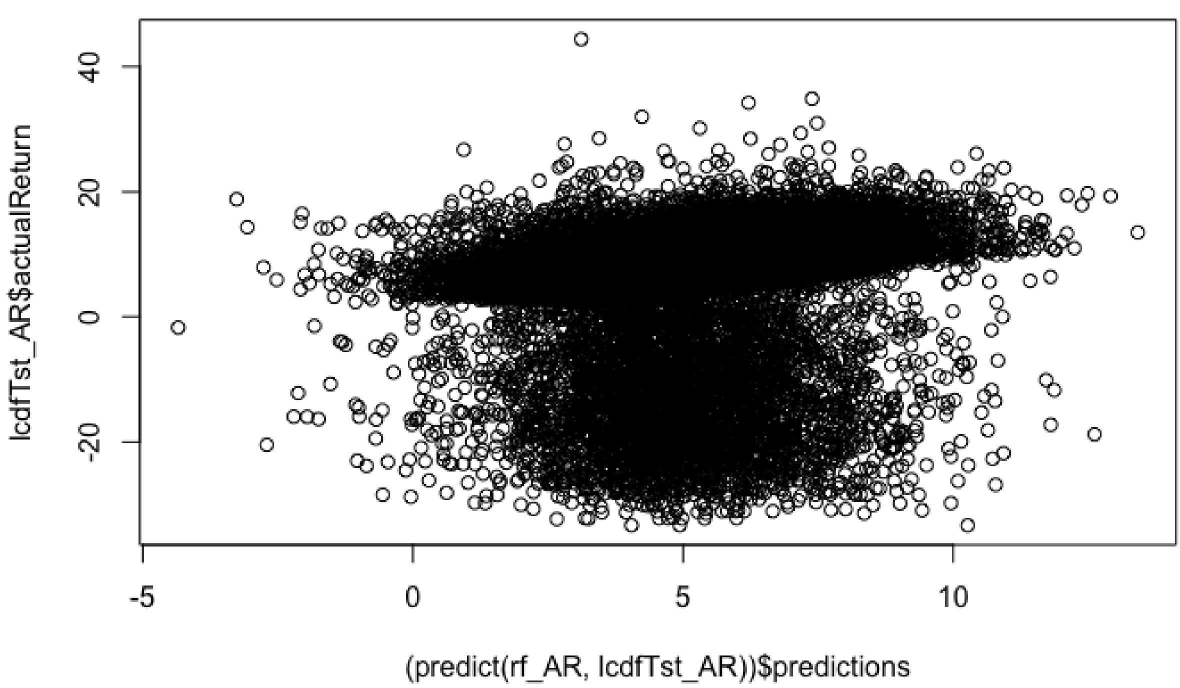
RF Performance by Deciles for Training Data

predRet_Trn_AR<- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_ARtrn=(predict(rf_AR, lcdfTrn_AR))$predictions)
predRet_Trn_AR<- predRet_Trn_AR %>% mutate(tile=ntile(-predRet_ARtrn, 10))
predRet_Trn_AR %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_ARtrn), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

tile <int>	count <int>	avgpredRet <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTerm <dbl>	totA <int>	totB <int>
1	7000	11.459257	8	14.598976	8.4048767	40.8154152	1.213971	2	268
2	7000	8.919240	15	11.018392	6.9463966	17.7625022	1.690217	7	1597
3	7000	7.765493	35	9.409129	5.2217222	20.2760823	1.996949	21	2520
4	7000	6.900850	42	8.212828	4.3849878	13.6093904	2.218635	195	3254
5	7000	6.165181	68	7.261014	2.3443526	12.4004522	2.335295	701	3890
6	7000	5.483443	106	6.467019	0.5092222	12.1281712	2.299710	1665	4076
7	7000	4.793582	146	5.514302	0.7324874	11.4981818	2.367788	3141	3324
8	7000	4.074396	186	4.528975	-1.4550833	10.7999775	2.573398	4973	1757
9	7000	2.590678	1987	2.088197	-12.4708169	8.3766897	2.769394	4584	1168
10	7000	-6.715925	7000	-16.171140	-33.3333333	-0.9448517	3.000000	600	1949

RF Test data

rfPredRet_Tst_AR<- predict(rf_AR, lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_AR$predictions- lcdfTst_AR$actualReturn)^2))
plot ((predict(rf_AR, lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
[1] 8.366654



RF Performance by Deciles for Test Data

```
predRet_Tst_AR<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet_ARtst=(predict(rf_AR, lcdfTst_AR))$predictions)
predRet_Tst_AR<- predRet_Tst_AR %>% mutate(tile=ntile(-predRet_ARtst, 10))
predRet_Tst_AR %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(predRet_ARtst),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),    maxRet=max(actualReturn), avgTerm=mean(actualTerm),
totA=sum(grade=="A"), totB=sum(grade=="B" ),   totC=sum(grade=="C"), totD=sum(grade=="D"),
totE=sum(grade=="E"), totF=sum(grade=="F"))
```

tile	count	avgpredRet	numDefaults	avgActRet	minRet	maxRet	avgTerm	totA	totB
<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>
1	3000	8.222095	511	7.836744	-33.33333	34.84624	2.148854	0	192
2	3000	6.724924	459	6.737833	-31.08100	27.51648	2.150261	0	838
3	3000	6.016469	465	5.940008	-32.19230	34.20363	2.236589	1	1264
4	3000	5.463725	399	5.800514	-32.22487	30.14516	2.225064	109	1488
5	3000	4.989860	403	5.164607	-33.33333	23.65092	2.225301	455	1418
6	3000	4.586126	365	4.674285	-32.18920	26.48408	2.260161	898	1261
7	3000	4.199768	358	4.402199	-33.33333	31.97116	2.283562	1209	1055
8	3000	3.793758	321	4.259976	-31.25387	24.53655	2.275021	1492	861
9	3000	3.286661	374	3.791127	-32.24854	44.35949	2.316634	1493	800
10	3000	2.017287	537	3.436664	-32.31356	27.62635	2.415366	1042	927

Balancing the Training Data with over and under sampling or combination of both

```
lcdfSplit_AR <- initial_split(lcdf_AR, prop=0.7)
lcdfTrn_Bal <- training(lcdfSplit_AR)
lcdfTst_Bal <- testing(lcdfSplit_AR)
us_lcdfTrn_AR<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn_Bal), na.action= na.pass,
method="under", p=0.5)$data
os_lcdfTrn_AR<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn_Bal), na.action= na.pass,
method="over", p=0.5)$data
bs_lcdfTrn_AR<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn_Bal), na.action= na.pass,
method="both", p=0.5)$data
bs_lcdfTrn_AR%>% group_by(loan_status) %>% count()
```

loan_status	n
<fctr>	<int>
Fully Paid	35046
Charged Off	34954

Building A RF w/ balanced data \#Building RF Under Sampling

```
rf_AR_us <- ranger(actualReturn ~., data=subset(us_lcdfTrn_AR, select=-c(annRet, actualTerm,
loan_status)), num.trees = 200, importance='permutation')
```

RF Under Sampling Results

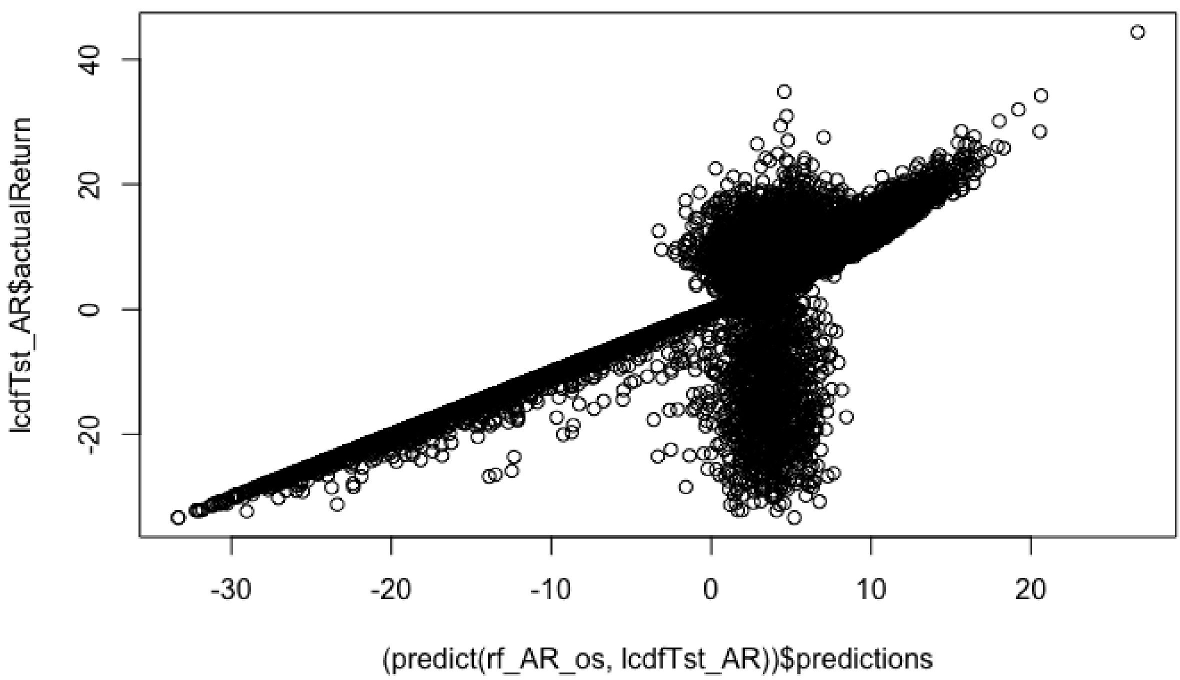
```
rfPredRet_trn_us<- predict(rf_AR_us,  us_lcdfTrn_AR)
sqrt(mean((rfPredRet_trn_us$predictions- us_lcdfTrn_AR$actualReturn)^2))
plot ((predict(rf_AR_us,  us_lcdfTrn_AR))$predictions, us_lcdfTrn_AR$actualReturn)
```

predRet_Trn_us<- us_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_ARtrn_us=(predict(rf_AR_us, us_lcdfTrn_AR))$predictions)
predRet_Trn_us<- predRet_Trn_us %>% mutate(tile=ntile(-predRet_ARtrn_us, 10))
predRet_Trn_us %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_ARtrn_us),   numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn),   maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"),   totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

RF Over Sampling
rf_AR_os <- ranger(actualReturn ~., data=subset(os_lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status)), num.trees = 200, importance='permutation')

RF Train Results for Over Sampling
rfPredRet_trn_os<- predict(rf_AR_os, os_lcdfTrn_AR)
sqrt(mean((rfPredRet_trn_os$predictions- os_lcdfTrn_AR$actualReturn)^2))
plot ((predict(rf_AR_os, os_lcdfTrn_AR))$predictions, os_lcdfTrn_AR$actualReturn)
predRet_Trn_os<- os_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_ARtrn_os=(predict(rf_AR_os, os_lcdfTrn_AR))$predictions)
predRet_Trn_os<- predRet_Trn_os %>% mutate(tile=ntile(-predRet_ARtrn_os, 10))
predRet_Trn_os %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_ARtrn_os),   numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn),   maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"),   totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

RF Over Sampling Test Results
rfPredRet_Tst_os<- predict(rf_AR_os, lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_os$predictions- lcdfTst_AR$actualReturn)^2))
plot ((predict(rf_AR_os, lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
predRet_Tst_os<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_ARTst_os=(predict(rf_AR_os, lcdfTst_AR))$predictions)
predRet_Tst_os<- predRet_Tst_os %>% mutate(tile=ntile(-predRet_ARTst_os, 10))
predRet_Tst_os %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_ARTst_os),   numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn),   maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"),   totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))
[1] 4.930287



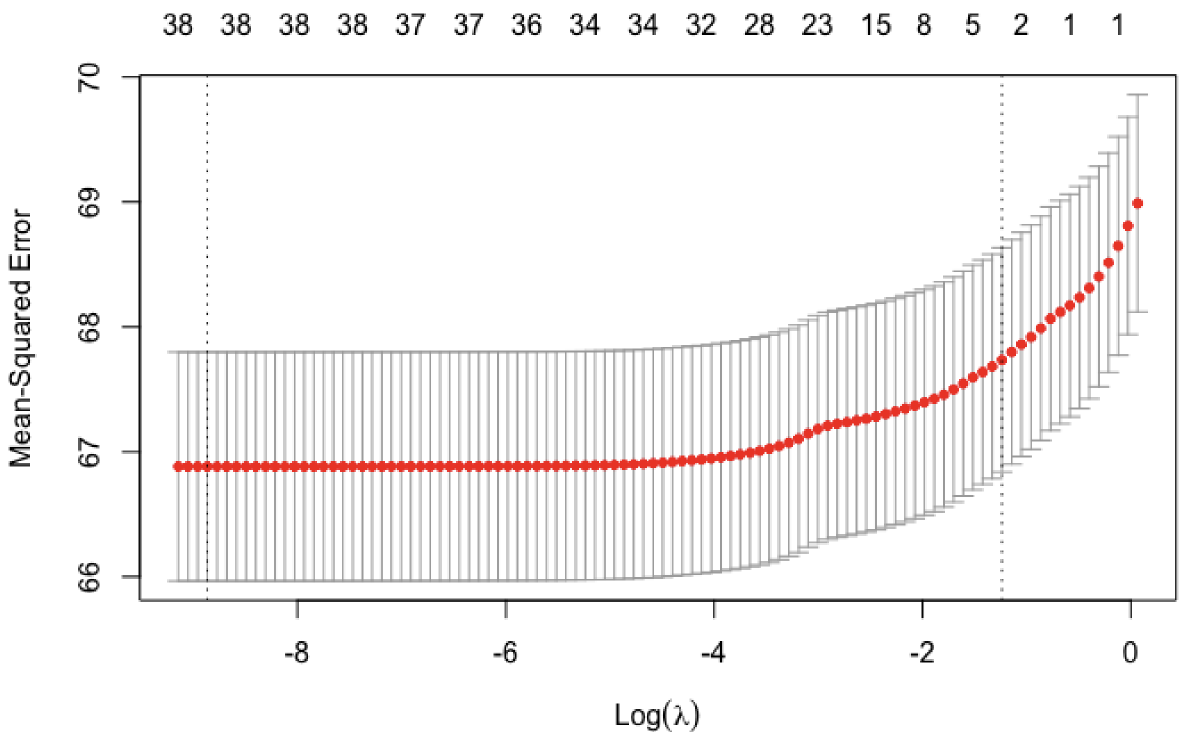
Building RF for Both (combination of over and under sampling)
rf_AR_bs <- ranger(actualReturn ~., data=subset(bs_lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status)), num.trees = 200, importance='permutation')

RF Both Train results
rfPredRet_Trn_bs<- predict(rf_AR_bs, bs_lcdfTrn_AR)
sqrt(mean((rfPredRet_Trn_bs$predictions- bs_lcdfTrn_AR$actualReturn)^2))
plot ((predict(rf_AR_bs, bs_lcdfTrn_AR))$predictions, bs_lcdfTrn_AR$actualReturn)
predRet_Trn_bs<- bs_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_ARTrn_bs=(predict(rf_AR_bs, bs_lcdfTrn_AR))$predictions)
predRet_Trn_bs<- predRet_Trn_bs %>% mutate(tile=ntile(-predRet_ARTrn_bs, 10))
predRet_Trn_bs %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_ARTrn_bs),   numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn),   maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"),   totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

RF Both Test results
rfPredRet_Tst_bs<- predict(rf_AR_bs, lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_bs$predictions- lcdfTst_AR$actualReturn)^2))
plot ((predict(rf_AR_bs, lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
predRet_Tst_bs<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_ARTst_bs=(predict(rf_AR_bs, lcdfTst_AR))$predictions)

predRet_Tst_bs<- predRet_Tst_bs %>% mutate(tile=ntile(-predRet_ARTst_bs, 10))
predRet_Tst_bs %>% group_by(tile) %>% summarise(count=n(),
avgpredRet=mean(predRet_ARTst_bs),   numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn),   maxRet=max(actualReturn),
avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"),   totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

Building GLM Model
#Creating data set for glm model
df4_glm <-lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)
glmRet_cv <- cv.glmnet(data.matrix(df4_glm), lcdfTrn_AR$actualReturn, family="gaussian")
plot(glmRet_cv)



Some Metrics for GLM model
glmRet_cv$lambda.min
glmRet_cv$lambda.1se
coef(glmRet_cv, s="lambda.1se") %>% tidy()
coef(glmRet_cv, s="lambda.min")

row <chr>	column <chr>	value <dbl>
(Intercept)	s1	3.32584955
int_rate	s1	0.20992071
home_ownership	s1	-0.05295093
dti	s1	-0.02498593

Predictions for GLM Model for Training Data

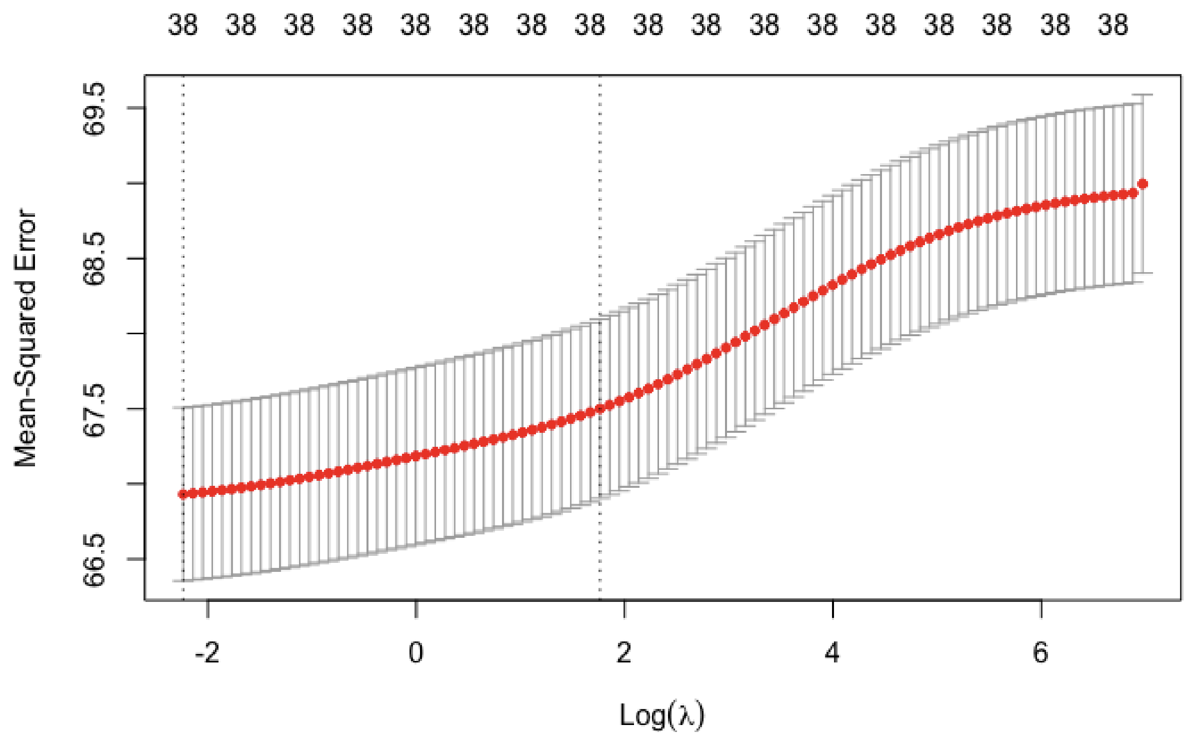
```
#Performance of GLM model for lambda min
predRet_Trn_AR_glm <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate)
%>% mutate(predRet_glm=  predict(glmRet_cv, data.matrix(lcdfTrn_AR %>% select(-loan_status,
-actualTerm, -annRet, -actualReturn)),s="lambda.min" ))
#Splitting into deciles
predRet_Trn_AR_glm<- predRet_Trn_AR_glm%>% mutate(tile=ntile(-predRet_glm, 10))
predRet_Trn_AR_glm %>% group_by(tile) %>%  summarise(count=n(),
avgpredRet=mean(predRet_glm),    numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn),    maxRet=max(actualReturn),
avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

Predictions of GLM Model for Test Data

```
#Performance of glm model for lambda.min
predRet_Tst_AR_glm <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate)
%>% mutate(predRet_glm=  predict(glmRet_cv, data.matrix(lcdfTst_AR %>% select(-loan_status,
-actualTerm, -annRet, -actualReturn)),s="lambda.min" ) )
#Splitting into deciles
predRet_Tst_AR_glm<- predRet_Tst_AR_glm%>% mutate(tile=ntile(-predRet_glm, 10))
predRet_Tst_AR_glm %>% group_by(tile) %>%  summarise(count=n(),
avgpredRet=mean(predRet_glm),    numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn),    maxRet=max(actualReturn),
avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

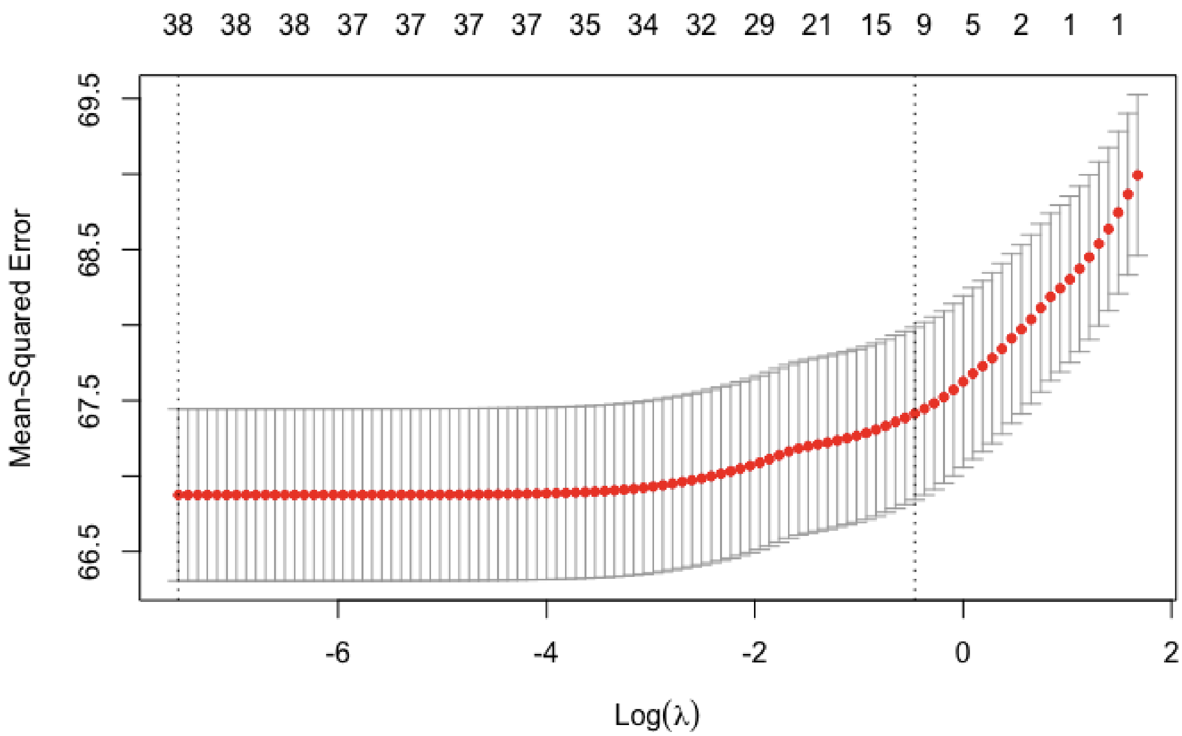
alpha=0 (Ridge Regression) for glm model

```
glmRet_cv_a0<- cv.glmnet(data.matrix(df4_glm),   lcdfTrn_AR$actualReturn, family="gaussian",
alpha=0)
plot(glmRet_cv_a0)
#1se
glmRet_cv_a0$lambda.1se
#min
glmRet_cv_a0$lambda.min
coef(glmRet_cv_a0, s="lambda.1se")
coef(glmRet_cv_a0, s="lambda.min")
```



Predictions for alpha=0 Model for Training Data
#Performance of glm model for lambda min
predRet_Trn_AR_a0 <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_a0= predict(glmRet_cv, data.matrix(lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)),s="lambda.min"))
#Splitting into deciles
predRet_Trn_AR_a0<- predRet_Trn_AR_a0 %>% mutate(tile=ntile(-predRet_a0, 10))
predRet_Trn_AR_a0 %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_a0), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

alpha = x for glm model
glmRet_cv_a2<- cv.glmnet(data.matrix(df4_glm), lcdfTrn_AR$actualReturn, family="gaussian", alpha=0.2)
plot(glmRet_cv_a2)
#1se
glmRet_cv_a2$lambda.1se
#min
glmRet_cv_a2$lambda.min
coef(glmRet_cv_a2, s="lambda.1se")
coef(glmRet_cv_a2, s="lambda.min")



Predictions for alpha=2 Model for Training Data

#Performance of GLM model for lambda min

predRet_Trn_AR_a2 <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_a2= predict(glmRet_cv, data.matrix(lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)),s="lambda.min"))

#Splitting into deciles

predRet_Trn_AR_a2<- predRet_Trn_AR_a2 %>% mutate(tile=ntile(-predRet_a2, 10))

predRet_Trn_AR_a2 %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_a2), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

Building GLM model with balanced data

Building GLM Model with under sampled data

#Creating data set for GLM model

df4_glm_us <-us_lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)

glmRet_cv_us <- cv.glmnet(data.matrix(df4_glm_us), us_lcdfTrn_AR$actualReturn, family="gaussian")

plot(glmRet_cv_us)

Some Metrics for GLM under sampling

glmRet_cv_us$lambda.min

glmRet_cv_us$lambda.1se

```
coef(glmRet_cv_us, s="lambda.1se") %>% tidy()
coef(glmRet_cv_us, s="lambda.min")
```

Predictions for GLM under sampling

```
#Performance of GLM model for lambda min
predRet_Trn_us_glm <- us_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm,
int_rate) %>% mutate(predRet_glm_us=  predict(glmRet_cv_us, data.matrix(us_lcdfTrn_AR %>%
select(-loan_status, -actualTerm, -annRet, -actualReturn)),s="lambda.min" ))
#Splitting into deciles
predRet_Trn_us_glm<- predRet_Trn_us_glm%>% mutate(tile=ntile(-predRet_glm_us, 10))
predRet_Trn_us_glm %>% group_by(tile) %>%  summarise(count=n(),
avgpredRet=mean(predRet_glm_us),    numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn),    maxRet=max(actualReturn),
avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

Building GLM Model with over sampled data

```
#Creating data set for glm model
df4_glm_os <-os_lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)
glmRet_cv_os <- cv.glmnet(data.matrix(df4_glm_os), os_lcdfTrn_AR$actualReturn, family="gaussian")
plot(glmRet_cv_os)
```

Some Metrics for GLM over sampling

```
glmRet_cv_os$lambda.min
glmRet_cv_os$lambda.1se
coef(glmRet_cv_os, s="lambda.1se") %>% tidy()
coef(glmRet_cv_os, s="lambda.min")
```

Predictions for GLM over sampling

```
#Performance of GLM model for lambda min
predRet_Trn_os_glm <- os_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm,
int_rate) %>% mutate(predRet_glm_os=  predict(glmRet_cv_os, data.matrix(os_lcdfTrn_AR %>%
select(-loan_status, -actualTerm, -annRet, -actualReturn)),s="lambda.min" ))
#Splitting into deciles
predRet_Trn_os_glm<- predRet_Trn_os_glm%>% mutate(tile=ntile(-predRet_glm_os, 10))
predRet_Trn_os_glm %>% group_by(tile) %>%  summarise(count=n(),
avgpredRet=mean(predRet_glm_os),    numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn),    maxRet=max(actualReturn),
avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

Building GLM Model with both sampled data

```
#Creating data set for GLM model
df4_glm_bs <-bs_lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)
```

glmRet_cv_bs <- cv.glmnet(data.matrix(df4_glm_bs), bs_lcdfTrn_AR$actualReturn, family="gaussian")
plot(glmRet_cv_bs)

Some Metrics for GLM both sampling
glmRet_cv_bs$lambda.min
glmRet_cv_bs$lambda.1se
coef(glmRet_cv_bs, s="lambda.1se") %>% tidy()
coef(glmRet_cv_bs, s="lambda.min")

Predictions for GLM both sampling
#Performance of glm model for lambda min
predRet_Trn_bs_glm <- bs_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(predRet_glm_bs= predict(glmRet_cv_bs, data.matrix(bs_lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)),s="lambda.min"))
#Splitting into deciles
predRet_Trn_bs_glm<- predRet_Trn_bs_glm%>% mutate(tile=ntile(-predRet_glm_bs, 10))
predRet_Trn_bs_glm %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet_glm_bs), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

XGBOOST Building the model
we are predicting actualReturn a numeric variable
#converting factor variables to dummy-variables
lcdf_ARdum2<-dummyVars(~.,data=lcdf_AR %>% select(-actualReturn))
dxlcdf2 <- predict(lcdf_ARdum2, lcdf_AR)
#Training, test subsets
dxlcdfTrn2 <- dxlcdf2[indicesTraining,]
colcdfTrn2 <- lcdf_AR$actualReturn[indicesTraining]
dxlcdfTst2 <- dxlcdf2[-indicesTraining,]
colcdfTst2 <- lcdf_AR$actualReturn[indicesTest]
#Creating Training and Test xgb matrices need it to run the model
dxTrn2 <- xgb.DMatrix(subset(dxlcdfTrn2, select=-c(annRet, actualTerm)), label=colcdfTrn2)
dxTst2 <- xgb.DMatrix(subset(dxlcdfTst2,select=-c(annRet, actualTerm)), label=colcdfTst2)
# (annRet, actualTerm) These variables are useful for performance assessment, but should not be used in the model
xgbWatchlist2 <- list(train = dxTrn2, eval = dxTst2)
#we can watch the progress of learning thru performance on these datasets
Including a xgbWatchlist so the early_stopping_rounds = 10 that we are going to use in the model can use it as a base to know when to stop

XGBOOST

````
#Parameter with the grid of options we want to test to find the best for the model
xgbParamGrid2 <- expand.grid(
max_depth = c(2, 5),
eta = c(0.001, 0.01, 0.1) )
#Parameter list
xgbParam2 <- list (
booster = "gbtree",
objective = "reg:squarederror",

min_child_weight=1,
colsample_bytree=0.6)
for(i in 1:nrow(xgbParamGrid2)) {
xgb_tune<- xgb.train(data=dxTrn2,xgbParam2,
nrounds=1000, early_stopping_rounds = 10, xgbWatchlist2,
eta=xgbParamGrid2$eta[i], max_depth=xgbParamGrid2$max_depth[i] )
xgbParamGrid2$bestTree[i] <- xgb_tune$evaluation_log[xgb_tune$best_iteration]$iter
xgbParamGrid2$bestPerf[i] <- xgb_tune$evaluation_log[xgb_tune$best_iteration]$eval_rmse
}

Plugin the tune parameters
```{r, output.lenght=32}
xgbParamGrid2

xgbParam3 <- list(
max_depth = 5,
eta = 0.100,
booster = "gbtree",
objective = "reg:squarederror",

min_child_weight=1,
colsample_bytree=0.6)

#XGBOOST running the model with the best parameters found with the for loop
xgb_lsM2 <- xgb.train(xgbParam3, dxTrn2, nrounds = xgb_tune$best_iteration)

XGBOOST evaluation of the model
```{r, output.lenght=32}
#Using the predicting function to get the scores in the training dataset
xpredTrn2<-predict(xgb_lsM2, dxTrn2)
head(xpredTrn2)
#Using the predicting function to get the scores in the test data set
xpredTst2<-predict(xgb_lsM2, dxTst2)
#Error
sqrt(mean((xpredTst2- colcdfTst2)^2))
````

#Performance by deciles
scoreTst_xgb_ls2 <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(score=xpredTst2)
scoreTst_xgb_ls2 <- scoreTst_xgb_ls2 %>% mutate(tile=ntile(-score, 10))
scoreTst_xgb_ls2 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score), numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"),
totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"), totG=sum(grade=="G"))

Q.4 Considering results from Questions 1 and 2 above – that is, considering the best model for predicting loan-status and that for predicting loan returns -- how would you select loans for investment? There can be multiple approaches for combining information from the two models - describe your approach, and show performance. How does performance here compare with use of single models?

R Code and Output:

XGBOOST Building the model
#We are predicting actualReturn a numeric variable
#Converting factor variables to dummy-variables
lcdf_ARdum2<-dummyVars(~.,data=lcdf_AR %>% select(-actualReturn))
dxlcdf2 <- predict(lcdf_ARdum2, lcdf_AR)
#Training, test subsets
dxlcdfTrn2 <- dxlcdf2[indicesTraining,]
colcdfTrn2 <- lcdf_AR$actualReturn[indicesTraining]
dxlcdfTst2 <- dxlcdf2[-indicesTraining,]
colcdfTst2 <- lcdf_AR$actualReturn[indicesTest]
#Creating Training and Test xgb matrices need it to run the model
dxTrn2 <- xgb.DMatrix(subset(dxlcdfTrn2, select=-c(annRet, actualTerm)), label=colcdfTrn2)
dxTst2 <- xgb.DMatrix(subset(dxlcdfTst2,select=-c(annRet, actualTerm)), label=colcdfTst2)
# (annRet, actualTerm) These variables are useful for performance assessment, but should not be used in the model
xgbWatchlist2 <- list(train = dxTrn2, eval = dxTst2)
#To watch the progress of learning thru performance on these datasets
#Including a xgbWatchlist so the early_stopping_rounds = 10 that we are going to use in the model can use it as a base to know when to stop

XGBOOST
#Weight Calculations
#sqrt(sum(dxlcdfTrn==0) / sum(dxlcdfTrn==1))   #7.950299
#Parameter with the grid of options we want to test to find the best for the model

```
xgbParamGrid2 <- expand.grid(
max_depth = c(2, 5),
eta = c(0.001, 0.01, 0.1) )
#Parameter list
xgbParam2 <- list (
booster = "gbtree",
objective = "reg:squarederror",
#scale_pos_weight = 7.950299,
min_child_weight=1,
colsample_bytree=0.6)
for(i in 1:nrow(xgbParamGrid2)) {
xgb_tune<- xgb.train(data=dxTrn2,xgbParam2,
nrounds=1000, early_stopping_rounds = 10, xgbWatchlist2,
eta=xgbParamGrid2$eta[i], max_depth=xgbParamGrid2$max_depth[i] )
xgbParamGrid2$bestTree[i] <- xgb_tune$evaluation_log[xgb_tune$best_iteration]$iter
xgbParamGrid2$bestPerf[i] <- xgb_tune$evaluation_log[xgb_tune$best_iteration]$eval_rmse
}
```

```
[1]	train-rmse:9.582130	eval-rmse:9.665909
Multiple eval metrics are present. Will use eval_rmse for early stopping.
Will train until eval_rmse hasn't improved in 10 rounds.

[2]	train-rmse:9.574399	eval-rmse:9.658188
[3]	train-rmse:9.571939	eval-rmse:9.655753
[4]	train-rmse:9.564423	eval-rmse:9.648134
[5]	train-rmse:9.557072	eval-rmse:9.640767
[6]	train-rmse:9.549667	eval-rmse:9.633330
[7]	train-rmse:9.542109	eval-rmse:9.625722
[8]	train-rmse:9.534453	eval-rmse:9.617990
[9]	train-rmse:9.526955	eval-rmse:9.610441
[10]	train-rmse:9.519296	eval-rmse:9.602722
[11]	train-rmse:9.511628	eval-rmse:9.595071
[12]	train-rmse:9.504107	eval-rmse:9.587543
[13]	train-rmse:9.496489	eval-rmse:9.579850
[14]	train-rmse:9.488967	eval-rmse:9.572345
[15]	train-rmse:9.481599	eval-rmse:9.564974
[16]	train-rmse:9.474005	eval-rmse:9.557269
[17]	train-rmse:9.466392	eval-rmse:9.549668
[18]	train-rmse:9.458826	eval-rmse:9.542000
[19]	train-rmse:9.451247	eval-rmse:9.534438
[20]	train-rmse:9.448858	eval-rmse:9.532122
[21]	train-rmse:9.441441	eval-rmse:9.524658
[22]	train-rmse:9.434045	eval-rmse:9.517211
[23]	train-rmse:9.426576	eval-rmse:9.509744
```

```
[24]	train-rmse:9.419087	eval-rmse:9.502189
[25]	train-rmse:9.411630	eval-rmse:9.494691
[26]	train-rmse:9.404066	eval-rmse:9.487155
[27]	train-rmse:9.396564	eval-rmse:9.479609
[28]	train-rmse:9.389105	eval-rmse:9.472081
[29]	train-rmse:9.381597	eval-rmse:9.464545
[30]	train-rmse:9.374214	eval-rmse:9.457141
[31]	train-rmse:9.366926	eval-rmse:9.449775
[32]	train-rmse:9.359511	eval-rmse:9.442265
[33]	train-rmse:9.351937	eval-rmse:9.434759
[34]	train-rmse:9.344440	eval-rmse:9.427208
[35]	train-rmse:9.342107	eval-rmse:9.424901
[36]	train-rmse:9.334813	eval-rmse:9.417576
[37]	train-rmse:9.327347	eval-rmse:9.410121
[38]	train-rmse:9.319934	eval-rmse:9.402643
[39]	train-rmse:9.312514	eval-rmse:9.395209
[40]	train-rmse:9.305219	eval-rmse:9.387917
[41]	train-rmse:9.297903	eval-rmse:9.380457
[42]	train-rmse:9.290418	eval-rmse:9.373019
[43]	train-rmse:9.283134	eval-rmse:9.365731
[44]	train-rmse:9.276119	eval-rmse:9.358677
[45]	train-rmse:9.268970	eval-rmse:9.351387
[46]	train-rmse:9.261511	eval-rmse:9.344004
[47]	train-rmse:9.254226	eval-rmse:9.336630
[48]	train-rmse:9.247039	eval-rmse:9.329401
[49]	train-rmse:9.239662	eval-rmse:9.322045
[50]	train-rmse:9.232432	eval-rmse:9.314692
…
```

Plugin the tune parameters

```
xgbParamGrid2

xgbParam3 <- list(
max_depth = 5,
eta = 0.100,
booster = "gbtree",
objective = "reg:squarederror",
#scale_pos_weight = 7.950299,
min_child_weight=1,
colsample_bytree=0.6)
#XGBOOST running the model with the best parameters found with the for loop
xgb_tune <- xgb.train(data=dxTrn2,xgbParam2)
xgb_lsM2 <- xgb.train(xgbParam3, dxTrn2, nrounds = xgb_tune$best_iteration)
```

max_depth <dbl>	eta <dbl>	bestTree <dbl>	bestPerf <dbl>
2	0.001	1000	5.509290
5	0.001	1000	5.380741
2	0.010	1000	3.925343
5	0.010	602	3.907755
2	0.100	177	3.919853
5	0.100	69	3.910070

XGBOOST evaluation of the model
#Using the predicting function to get the scores in the training data set
xpredTrn2<-predict(xgb_lsM2, dxTrn2)
head(xpredTrn2)
#Using the predicting function to get the scores in the test data set
xpredTst2<-predict(xgb_lsM2, dxTst2)
#Error
sqrt(mean((xpredTst2- colcdfTst2)^2))
#Performance by deciles
scoreTst_xgb_ls2 <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(score=xpredTst2)
scoreTst_xgb_ls2 <- scoreTst_xgb_ls2 %>% mutate(tile=ntile(-score, 10))
scoreTst_xgb_ls2 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score),
numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn),
avgTer=mean(actualTerm), totA=sum(grade=="A"),
totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"),
totF=sum(grade=="F"), totG=sum(grade=="G"))

Creating the RF based on Assignment 1 for all Grade loans
#myweights = ifelse(lcdfTrn_AR$loan_status == "Charged Off", 5, 1)
RF_Asst1 <- ranger(loan_status ~., data=subset(lcdfTrn_AR, select=-c(annRet, actualTerm,
actualReturn)), num.trees =200, min.node.size=1, importance='impurity', probability=TRUE)
#case.weights= myweights)

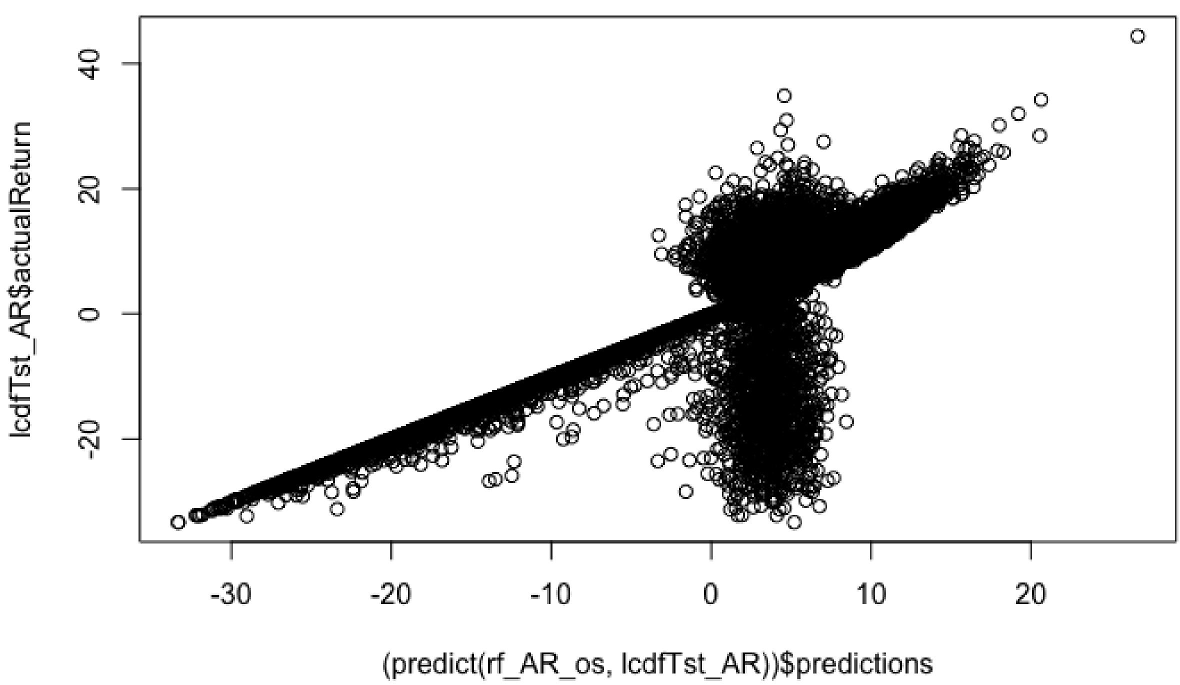
Performance of RF for all grades in Deciles M1
ag_scoreTstRF <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(score=(predict(RF_Asst1,lcdfTst_AR))$predictions[,"Fully Paid"])
ag_scoreTstRF <- ag_scoreTstRF %>% mutate(tile=ntile(-score, 10))
ag_scoreTstRF %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"
),
totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

tile <int>	count <int>	avgSc <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTer <dbl>	totA <int>	totB <int>
1	3000	0.9744600	113	4.226684	-29.14592	13.22594	2.205999	2597	394
2	3000	0.9442667	191	4.492962	-32.31356	16.40321	2.241992	1712	1192
3	3000	0.9192978	258	5.033545	-31.25387	25.79776	2.231890	993	1658
4	3000	0.8946076	256	5.718098	-31.27686	26.09333	2.210089	599	1710
5	3000	0.8697894	368	5.506807	-32.22487	23.33842	2.216427	385	1551
6	3000	0.8445544	398	5.970973	-32.22428	23.41193	2.210336	183	1263
7	3000	0.8176700	489	5.757093	-30.32713	34.84624	2.236097	119	1000
8	3000	0.7872978	561	5.609815	-33.33333	29.36088	2.289612	68	703
9	3000	0.7482450	670	5.261891	-33.33333	30.14516	2.308978	29	441
10	3000	0.6674772	888	4.466090	-32.19230	44.35949	2.385394	14	192

Rf Returns selected by grade M2 - all grades

```
#Choose the Actual Returns Random forest we want for here
lcdfSplit_AR <- initial_split(lcdf_AR, prop=0.7)
lcdfTrn_Bal <- training(lcdfSplit_AR)
os_lcdfTrn_AR<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn_Bal), na.action= na.pass,
method="over", p=0.5)$data
rf_AR_os <- ranger(actualReturn ~., data=subset(os_lcdfTrn_AR, select=-c(annRet, actualTerm,
loan_status)), num.trees = 200, importance='permutation')
rfPredRet_Tst_ag<- predict(rf_AR_os,  lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_ag$predictions- lcdfTst_AR$actualReturn)^2))
plot ((predict(rf_AR_os,  lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
```

[1] 4.930287



Rf Performance by Deciles for Test Data - all grades

```
ag_predRet_Tst<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet_agTst=(predict(rf_AR_os, lcdfTst_AR))$predictions)
ag_predRet_Tst<- ag_predRet_Tst %>% mutate(tile=ntile(-predRet_agTst, 10))
ag_predRet_Tst %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(predRet_agTst),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),    maxRet=max(actualReturn), avgTerm=mean(actualTerm),
totA=sum(grade=="A"), totB=sum(grade=="B" ),   totC=sum(grade=="C"), totD=sum(grade=="D"),
totE=sum(grade=="E"), totF=sum(grade=="F"))
```

tile <int>	count <int>	avgpredRet <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTerm <dbl>	totA <int>	totB <int>
1	3000	10.248020	4	13.736934	8.475877	44.35949	1.312339	2	243
2	3000	7.855640	28	9.948265	-26.691333	20.69885	1.884026	6	926
3	3000	6.749990	61	8.279634	-30.756133	27.51648	2.219319	73	1240
4	3000	5.953158	117	6.997805	-29.802708	24.14531	2.310514	260	1527
5	3000	5.276368	123	6.445048	-33.333333	22.13129	2.282129	538	1693
6	3000	4.672834	150	5.717288	-31.050366	34.84624	2.298853	992	1407
7	3000	4.117090	198	4.783145	-32.224276	29.36088	2.420585	1557	965
8	3000	3.545134	228	4.348729	-30.080300	24.22094	2.437479	1827	648
9	3000	2.501245	542	3.681154	-32.192296	26.48408	2.428589	1179	728
10	3000	-11.797970	2741	-11.894043	-33.333333	22.53423	2.942981	265	727

Consider Top d deciles from Model 2(actualReturn), ranked by M1 scores(loan_Status) all grades

```
d=1
pRetSc_RF_ag <- ag_predRet_Tst %>% mutate(poScore=ag_scoreTstRF$score)
pRet_d_RF_ag <- pRetSc_RF_ag %>% filter(tile<=d)
pRet_d_RF_ag<- pRet_d_RF_ag %>% mutate(tile2=ntile(-poScore, 20))
pRet_d_RF_ag %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(predRet_agTst),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"
),
totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

tile2 <int>	count <int>	avgPredRet <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTer <dbl>	totA <int>	totB <int>
1	150	9.585386	0	11.83893	8.475877	25.79776	1.015487	1	67
2	150	9.894872	0	12.40863	8.581450	26.09333	1.027369	0	47
3	150	9.938695	0	12.48361	8.483169	22.30971	1.135204	0	31
4	150	10.070057	0	12.88853	9.402708	23.33842	1.139092	0	20
5	150	9.906231	0	12.68308	9.353654	18.60215	1.194543	0	21
6	150	10.037913	2	13.13574	8.675167	20.28881	1.266156	1	11
7	150	10.049254	0	13.07460	9.410012	20.27814	1.212083	0	12
8	150	10.132540	0	13.26988	9.264268	20.34229	1.358029	0	7
9	150	10.168243	0	13.35644	9.380745	23.35950	1.317563	0	3
10	150	10.251903	0	13.63563	8.822596	21.36442	1.228638	0	4

Boosting - Creating the Boosting Model based on Assignment 1 for all Grade loans

```
# use the dummyVars function in the 'caret' package to convert factor variables to dummy-variables, do
not include loan_status for this
fdum1<-dummyVars(~.,data=lcdf_AR %>% select(-loan_status))
dxlcdf1 <- predict(fdum1, lcdf_AR)
```

```
# for loan_status, check levels and convert to dummy vars and keep the class label of interest
levels(lcdf_AR$loan_status)
#"Charged Off" "Fully Paid"
#converting to dummy variables
dylcdf1 <- class2ind(lcdf_AR$loan_status, drop2nd = FALSE)
# we decided we want to keep "Fully Paid"
colcdf1 <- dylcdf1 [ ,2]
#Training, test subsets
dxlcdfTrn1 <- dxlcdf1[indicesTraining,]
colcdfTrn1 <- colcdf1[indicesTraining]
dxlcdfTst1 <- dxlcdf1[indicesTest,]
colcdfTst1 <- colcdf1[indicesTest]
#Creating of xgb.DMatrix
dxTrn1 <- xgb.DMatrix(subset(dxlcdfTrn1, select=-c(annRet, actualTerm, actualReturn)),
label=colcdfTrn1)
dxTst1 <- xgb.DMatrix(subset(dxlcdfTst1, select=-c(annRet, actualTerm, actualReturn)),
label=colcdfTst1)
#we can watch the progress of learning thru performance on these datasets
xgbWatchlist1 <- list(train = dxTrn1, eval = dxTst1)
#defining weights
sqrt(sum(dxlcdfTrn1==0) / sum(dxlcdfTrn1==1))  #8.536599
#use cross-validation on training dataset to determine best model
xgbParamGrid1 <- expand.grid( max_depth = c(2, 5), eta = c(0.001, 0.01, 0.1) )
xgbParam1 <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1,
colsample_bytree=0.6,eval_metric="error", eval_metric = "auc")
for(i in 1:nrow(xgbParamGrid1)) {
xgb_tune1<- xgb.train(data=dxTrn1,xgbParam1,
nrounds=1000, early_stopping_rounds = 10, xgbWatchlist1,
eta=xgbParamGrid1$eta[i], max_depth=xgbParamGrid1$max_depth[i] )
xgbParamGrid1$bestTree[i] <- xgb_tune1$evaluation_log[xgb_tune1$best_iteration]$iter
xgbParamGrid1$bestPerf[i] <- xgb_tune1$evaluation_log[xgb_tune1$best_iteration]$eval_auc
}
```

```
[1] "Charged Off" "Fully Paid"
[1] 8.743602
[1]	train-error:0.137043	train-auc:0.608211	eval-error:0.139733	eval-auc:0.608633
Multiple eval metrics are present. Will use eval_auc for early stopping.
Will train until eval_auc hasn't improved in 10 rounds.

[2]	train-error:0.137043	train-auc:0.648336	eval-error:0.139733	eval-auc:0.654023
[3]	train-error:0.137043	train-auc:0.648336	eval-error:0.139733	eval-auc:0.654023
[4]	train-error:0.137043	train-auc:0.648336	eval-error:0.139733	eval-auc:0.654023
[5]	train-error:0.137043	train-auc:0.648344	eval-error:0.139733	eval-auc:0.654029
[6]	train-error:0.137043	train-auc:0.648341	eval-error:0.139733	eval-auc:0.654030
```

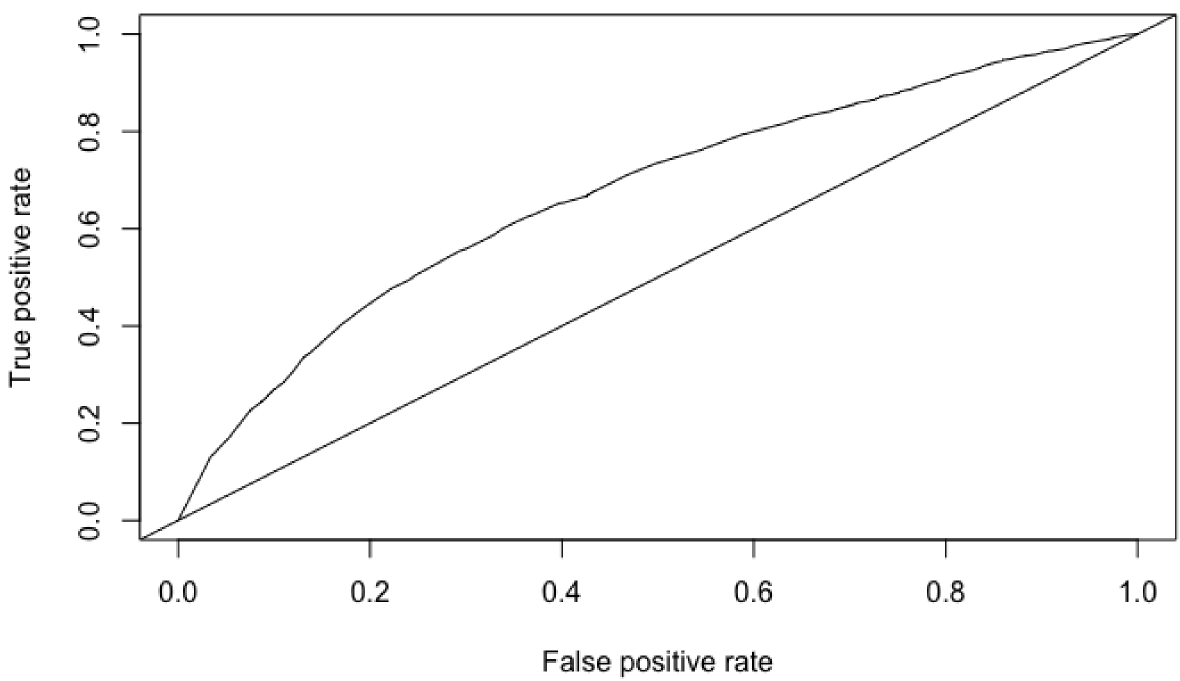
[7]	train-error:0.137043	train-auc:0.648341	eval-error:0.139733	eval-auc:0.654030
[8]	train-error:0.137043	train-auc:0.652393	eval-error:0.139733	eval-auc:0.659533
[9]	train-error:0.137043	train-auc:0.652418	eval-error:0.139733	eval-auc:0.659590
[10]	train-error:0.137043	train-auc:0.653413	eval-error:0.139733	eval-auc:0.660746
[11]	train-error:0.137043	train-auc:0.653413	eval-error:0.139733	eval-auc:0.660751
[12]	train-error:0.137043	train-auc:0.653416	eval-error:0.139733	eval-auc:0.660725
[13]	train-error:0.137043	train-auc:0.653405	eval-error:0.139733	eval-auc:0.660751
[14]	train-error:0.137043	train-auc:0.653406	eval-error:0.139733	eval-auc:0.660752
[15]	train-error:0.137043	train-auc:0.653406	eval-error:0.139733	eval-auc:0.660752
[16]	train-error:0.137043	train-auc:0.653417	eval-error:0.139733	eval-auc:0.660727
[17]	train-error:0.137043	train-auc:0.657834	eval-error:0.139733	eval-auc:0.665258
[18]	train-error:0.137043	train-auc:0.657828	eval-error:0.139733	eval-auc:0.665271
[19]	train-error:0.137043	train-auc:0.657808	eval-error:0.139733	eval-auc:0.665266
[20]	train-error:0.137043	train-auc:0.657814	eval-error:0.139733	eval-auc:0.665253
[21]	train-error:0.137043	train-auc:0.657812	eval-error:0.139733	eval-auc:0.665251
[22]	train-error:0.137043	train-auc:0.657812	eval-error:0.139733	eval-auc:0.665251
[23]	train-error:0.137043	train-auc:0.657808	eval-error:0.139733	eval-auc:0.665251
[24]	train-error:0.137043	train-auc:0.658478	eval-error:0.139733	eval-auc:0.665760
[25]	train-error:0.137043	train-auc:0.658473	eval-error:0.139733	eval-auc:0.665757
[26]	train-error:0.137043	train-auc:0.658476	eval-error:0.139733	eval-auc:0.665756
[27]	train-error:0.137043	train-auc:0.658477	eval-error:0.139733	eval-auc:0.665738
[28]	train-error:0.137043	train-auc:0.658795	eval-error:0.139733	eval-auc:0.665939
[29]	train-error:0.137043	train-auc:0.659023	eval-error:0.139733	eval-auc:0.666035
[30]	train-error:0.137043	train-auc:0.659027	eval-error:0.139733	eval-auc:0.666002
[31]	train-error:0.137043	train-auc:0.659022	eval-error:0.139733	eval-auc:0.665996
[32]	train-error:0.137043	train-auc:0.660655	eval-error:0.139733	eval-auc:0.668163
[33]	train-error:0.137043	train-auc:0.660653	eval-error:0.139733	eval-auc:0.668167
[34]	train-error:0.137043	train-auc:0.660649	eval-error:0.139733	eval-auc:0.668172
[35]	train-error:0.137043	train-auc:0.660646	eval-error:0.139733	eval-auc:0.668171
[36]	train-error:0.137043	train-auc:0.660689	eval-error:0.139733	eval-auc:0.668261
[37]	train-error:0.137043	train-auc:0.660846	eval-error:0.139733	eval-auc:0.668378
[38]	train-error:0.137043	train-auc:0.660849	eval-error:0.139733	eval-auc:0.668381
[39]	train-error:0.137043	train-auc:0.660848	eval-error:0.139733	eval-auc:0.668382
[40]	train-error:0.137043	train-auc:0.660847	eval-error:0.139733	eval-auc:0.668378
[41]	train-error:0.137043	train-auc:0.660846	eval-error:0.139733	eval-auc:0.668381
[42]	train-error:0.137043	train-auc:0.660958	eval-error:0.139733	eval-auc:0.668572
[43]	train-error:0.137043	train-auc:0.660958	eval-error:0.139733	eval-auc:0.668573
[44]	train-error:0.137043	train-auc:0.660959	eval-error:0.139733	eval-auc:0.668576
[45]	train-error:0.137043	train-auc:0.660961	eval-error:0.139733	eval-auc:0.668578
[46]	train-error:0.137043	train-auc:0.661137	eval-error:0.139733	eval-auc:0.668783
[47]	train-error:0.137043	train-auc:0.661135	eval-error:0.139733	eval-auc:0.668779
[48]	train-error:0.137043	train-auc:0.661136	eval-error:0.139733	eval-auc:0.668782
[49]	train-error:0.137043	train-auc:0.661138	eval-error:0.139733	eval-auc:0.668783
[50]	train-error:0.137043	train-auc:0.661139	eval-error:0.139733	eval-auc:0.668784

```
[51]	train-error:0.137043	train-auc:0.661120	eval-error:0.139733	eval-auc:0.668704
[52]	train-error:0.137043	train-auc:0.661121	eval-error:0.139733	eval-auc:0.668707
[53]	train-error:0.137043	train-auc:0.661181	eval-error:0.139733	eval-auc:0.668880
[54]	train-error:0.137043	train-auc:0.661181	eval-error:0.139733	eval-auc:0.668880
[55]	train-error:0.137043	train-auc:0.661181	eval-error:0.139733	eval-auc:0.668880
[56]	train-error:0.137043	train-auc:0.661183	eval-error:0.139733	eval-auc:0.668878
[57]	train-error:0.137043	train-auc:0.661184	eval-error:0.139733	eval-auc:0.668873
[58]	train-error:0.137043	train-auc:0.661185	eval-error:0.139733	eval-auc:0.668871
[59]	train-error:0.137043	train-auc:0.661185	eval-error:0.139733	eval-auc:0.668871
[60]	train-error:0.137043	train-auc:0.661193	eval-error:0.139733	eval-auc:0.668808
[61]	train-error:0.137043	train-auc:0.660897	eval-error:0.139733	eval-auc:0.668465
[62]	train-error:0.137043	train-auc:0.661364	eval-error:0.139733	eval-auc:0.668802
[63]	train-error:0.137043	train-auc:0.661364	eval-error:0.139733	eval-auc:0.668802
Stopping. Best iteration:
[53]	train-error:0.137043	train-auc:0.661181	eval-error:0.139733	eval-auc:0.668880
```

```
xgbParamGrid1
xgbParam_Best1 <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1,
colsample_bytree=0.6, scale_pos_weight = 8.53, max_depth = 2, eta = 0.001)
# XGBOOST running the model with the best parameters found with the for loop
xgb_lsM1 <- xgb.train(xgbParam_Best1, dxTrn1, nrounds = xgb_tune1$best_iteration)
#XGBOOST evaluation of the model
#Using the predicting function to get the scores in the training data  set
xpredTrn1<-predict(xgb_lsM1, dxTrn1)
head(xpredTrn1)
#Using the predicting function to get the scores in the test data set
xpredTst1<-predict(xgb_lsM1, dxTst1)
head(xpredTst1)
#Auc
#ROC, AUC performance
#confusion matrix
table(pred=as.numeric(xpredTst1>0.5), act=colcdfTst1)
#ROC, AUC performance
pred_xgb_lsM1<-prediction(xpredTst1, lcdfTst_AR$loan_status,
label.ordering = c("Charged Off", ("Fully Paid")))
aucPerf_xgb_lsM1<-performance(pred_xgb_lsM1, "tpr", "fpr")
plot(aucPerf_xgb_lsM1)
abline(a=0, b= 1)
#variable importance
xgb.importance(model = xgb_lsM1) %>% view()
```

max_depth <dbl>	eta <dbl>	bestTree <dbl>	bestPerf <dbl>
2	0.001	53	0.668880
5	0.001	51	0.684453
2	0.010	130	0.677105
5	0.010	29	0.684844
2	0.100	142	0.693089
5	0.100	70	0.693284

```
[1] 0.5332605 0.5321483 0.5324648 0.5321177 0.5321068 0.5326806
[1] 0.5321483 0.5321483 0.5321372 0.5321177 0.5326617 0.5321563
    act
pred     0     1
   1  4192 25808
```



XGBOOST Performance of Boosting A-G in Deciles M1
xpredTstM1<-predict(xgb_lsM1, dxTst1)
scoreTst_xgb_ls1 <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(score=xpredTst1)
scoreTst_xgb_ls1 <- scoreTst_xgb_ls1 %>% mutate(tile=ntile(-score, 10))
scoreTst_xgb_ls1 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score), numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn),
avgTer=mean(actualTerm), totA=sum(grade=="A"),
totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"),
totF=sum(grade=="F"), totE=sum(grade=="G"),)

tile <int>	count <int>	avgSc <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTer <dbl>	totA <int>	totB <int>
1	3000	0.5332605	122	3.953400	-31.27686	14.03120	2.281764	3000	0
2	3000	0.5332436	189	3.718988	-32.31356	15.41828	2.231961	3000	0
3	3000	0.5331005	224	4.334127	-31.25387	14.14902	2.258669	699	2301
4	3000	0.5327499	268	5.712965	-31.20100	18.40389	2.223498	0	2993
5	3000	0.5326900	351	5.409525	-31.10341	21.27937	2.249873	0	3000
6	3000	0.5325824	432	5.129059	-33.33333	23.33842	2.240108	0	1542
7	3000	0.5322713	495	6.093815	-32.22487	34.84624	2.235217	0	268
8	3000	0.5321205	621	5.365459	-32.18920	28.51641	2.248968	0	0
9	3000	0.5318263	676	6.348609	-33.33333	44.35949	2.282271	0	0
10	3000	0.5315362	814	5.978010	-33.33333	34.20363	2.284487	0	0

XGBOOST Performance of Boosting A-G in Deciles M2
xpredTstM2<-predict(xgb_lsM2, dxTst2)
scoreTst_xgb_ls2 <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(score2=xpredTst2)
scoreTst_xgb_ls2 <- scoreTst_xgb_ls2 %>% mutate(tile=ntile(-score2, 10))
scoreTst_xgb_ls2 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score2), numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"),
totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"), totE=sum(grade=="G"),)

```{r, output.lenght=32}
d=1
xpredTst2<-predict(xgb_lsM2, dxTst2)
scoreTst_xgb_ls2 <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(score=xpredTst2)
pRetSc <- scoreTst_xgb_ls2 %>% mutate(poScore=scoreTst_xgb_ls1$score)
pRet_d <- pRetSc %>% filter(tile<=d)
pRet_d<- pRet_d %>% mutate(tile2=ntile(-poScore, 10))
pRet_d %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(score2), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ),
totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )

Performance of glm model for all grade loans in deciles for M1
```{r, output.lenght=32}
xDTrn<-lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)
yTrn<-factor(if_else(lcdfTrn_AR$loan_status=="Fully Paid", '1', '0') )
glmls_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial")
predLS_glm <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(glmPredls_pMin=predict(glmls_cv,data.matrix(xDTrn), s="lambda.min", type="response" ))
predLS_glm<- predLS_glm%>% mutate(tile=ntile(-glmPredls_pMin, 10))

```
predLS_glm %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(glmPredls_pMin),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),    maxRet=max(actualReturn), avgTerm=mean(actualTerm),
totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"),
totE=sum(grade=="E"), totF=sum(grade=="F") )
```

tile <int>	count <int>	avgpredRet <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTerm <dbl>	totA <int>	totB <int>
1	7000	0.9602597	216	4.232955	-32.30987	17.81924	2.205535	6039	890
2	7000	0.9378507	391	4.676768	-32.22635	18.50286	2.228800	4313	2479
3	7000	0.9217279	542	5.035678	-32.24217	30.85477	2.225452	2682	3848
4	7000	0.9063543	616	5.448636	-32.22634	22.85197	2.215567	1578	4341
5	7000	0.8903458	808	5.421337	-32.25540	24.61920	2.233900	806	4309
6	7000	0.8722539	932	5.812767	-32.24854	26.79731	2.219232	312	3646
7	7000	0.8506081	1097	5.746110	-33.33333	24.85824	2.252177	117	2540
8	7000	0.8226186	1364	5.672011	-33.33333	34.20051	2.259498	36	1270
9	7000	0.7820259	1563	5.645914	-31.25383	29.49576	2.286207	6	426
10	7000	0.6855264	2064	5.235514	-33.33333	40.81542	2.338990	0	54

Actual Returns selected by grade M2 all grades

```
#Performance of glm model for lambda min
df4_glm <-lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)
glmRet_cv <- cv.glmnet(data.matrix(df4_glm), lcdfTrn_AR$actualReturn, family="gaussian")
predRet_Trn_glm <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate)
%>% mutate(predRet_glm=  predict(glmRet_cv, data.matrix(lcdfTrn_AR %>% select(-loan_status,
-actualTerm, -annRet, -actualReturn)),s="lambda.min" ))
predRet_Trn_glm<- predRet_Trn_glm%>% mutate(tile=ntile(-predRet_glm, 10))
predRet_Trn_glm %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(predRet_glm),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),    maxRet=max(actualReturn), avgTerm=mean(actualTerm),
totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"),
totE=sum(grade=="E"), totF=sum(grade=="F") )
```

tile <int>	count <int>	avgpredRet <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTerm <dbl>	totA <int>	totB <int>
1	7000	8.149745	1153	8.310403	-32.17619	40.24976	2.212328	12	916
2	7000	6.852539	1040	6.952767	-32.22635	40.81542	2.177340	63	2199
3	7000	6.224443	1098	6.178427	-33.33333	38.13311	2.192476	178	2556
4	7000	5.748064	1066	5.591385	-33.33333	36.86044	2.219362	543	2619
5	7000	5.346003	1018	5.200736	-32.30987	28.84105	2.236752	987	2683
6	7000	4.969548	937	4.934296	-32.25500	31.32790	2.236533	1636	2624
7	7000	4.609981	879	4.561291	-33.33333	29.77180	2.255522	2188	2651
8	7000	4.236242	849	4.104033	-32.28293	24.67951	2.279141	2836	2550
9	7000	3.788537	759	3.831002	-31.29376	26.79875	2.293794	3420	2514
10	7000	3.002589	794	3.263351	-32.29490	22.60164	2.362109	4026	2491

GLM - Consider Top d deciles from Model 2(actualReturn), ranked by M1 scores(loan_Status) all grades

```
d=1
pRetSc_glm <- predRet_Trn_glm %>% mutate(poScore=predLS_glm$glmPredls_pMin)
pRet_d_glm <- pRetSc_glm %>% filter(tile<=d)
pRet_d_glm<- pRet_d_glm %>% mutate(tile2=ntile(-poScore, 20))
pRet_d_glm %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(predRet_glm),
```

numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B"
),
totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"))

tile2 <int>	count <int>	avgPredRet <dbl>	numDefaults <int>	avgActRet <dbl>	minRet <dbl>	maxRet <dbl>	avgTer <dbl>	totA <int>	totB <int>
1	350	8.092517	25	6.863417	-26.86533	17.81924	2.219417	12	255
2	350	7.932308	36	6.778535	-30.35203	16.47523	2.207467	0	222
3	350	7.973038	27	8.102244	-31.04505	16.69297	2.176593	0	181
4	350	7.913367	27	8.239050	-25.09637	17.38997	2.226086	0	126
5	350	7.961018	39	7.890863	-31.01210	18.87776	2.174888	0	74
6	350	7.986183	35	8.558284	-29.93133	24.61920	2.219405	0	46
7	350	8.073536	39	8.891786	-28.03776	22.34097	2.118989	0	12
8	350	8.161267	53	8.367374	-32.17619	26.79731	2.210328	0	0
9	350	8.153369	59	8.086641	-31.41667	19.16581	2.234886	0	0
10	350	8.218660	44	9.398014	-30.70667	24.85824	2.169629	0	0

Q.5 As seen in data summaries and your work in the first assignment, higher grade loans are less likely to default, but also carry lower interest rates; many lower grad loans are fully paid, and these can yield higher returns. One approach may be to focus on lower grade loans (C and below), and try to identify those which are likely to be paid off. Develop models from the data on lower grade loans, and check if this can provide an effective investment approach – for this, you can use one of the methods (glm, rf, or gbm/xgb) which you find to give superior performance from earlier questions. Can this provide a useful approach for investment? Compare performance with that in Question 4.

By using xgboost model, grades from c-lower are shown clearly whether it is fully paid/charged off. The last table below shows that some loans have a higher actual return than predicted return which means it's a good approach for investment. The xgBoost model could picture the results fairly clearly about which class has better returns, and it's shown on the maxRet column.

<u>R Code and Output:</u>

```
#Creating a GLM model for grade C-G loans M1

#Selecting only Grades C-G
lg_lcdfTrn<-Trainingdf %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')
lg_lcdfTst<-Testdf %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')

#XGBOOST Creating a data set for grade C and lower
lcdf_AR_LowGrades<-lcdf_AR %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade==
'G')

##Start building XGBOOST model by Splitting Data into Training, Validation, and Test Sets
set.seed(123)

fractionTraining   <- 0.70
fractionValidation <- 0.00
fractionTest       <- 0.30
```

```
# Compute sample sizes.
sampleSizeTraining2   <- floor(fractionTraining   * nrow(lcdf_AR_LowGrades))
sampleSizeValidation2 <- floor(fractionValidation * nrow(lcdf_AR_LowGrades))
sampleSizeTest2       <- floor(fractionTest       * nrow(lcdf_AR_LowGrades))

# Create the randomly-sampled indices for the dataframe.
indicesTraining2    <- sort(sample(seq_len(nrow(lcdf_AR_LowGrades)), size=sampleSizeTraining2))
indicesNotTraining2 <- setdiff(seq_len(nrow(lcdf_AR_LowGrades)), indicesTraining2)
indicesValidation2  <- sort(sample(indicesNotTraining2, size=sampleSizeValidation2))
indicesTest2        <- setdiff(indicesNotTraining2, indicesValidation2)

# Deploy the dataframes for training, and validation
Trainingdf2 <- lcdf_AR_LowGrades[indicesTraining2, ]
Validationdf2 <- lcdf_AR_LowGrades[indicesValidation2, ]
Testdf2 <- lcdf_AR_LowGrades[indicesTest2, ]


#Building XGBOOST model for grades C and lower

#All data should be numeric, therefore we convert categorial variables  using one-hot encoding
# we use the dummyVars function from 'caret' package to convert the data type
#values_count <- sapply(lapply(lcdf_AR_LowGrades, unique), length)
#fdum4<-dummyVars(~.,data=lcdf_AR_LowGrades[ , values_count > 1] %>% select(-loan_status))
#lcdf_AR_LowGrades<- na.omit(lcdf_AR_LowGrades)
fdum4<-dummyVars(~.,data=lcdf_AR_LowGrades %>% select(-loan_status))
dxlcdf4 <- predict(fdum4, lcdf_AR_LowGrades)

# for loan_status, check levels and convert to dummy vars and keep the class label of interest
levels(lcdf_AR_LowGrades$loan_status)
dylcdf4 <- class2ind(lcdf_AR_LowGrades$loan_status, drop2nd = FALSE)
colcdf4 <- dylcdf4 [ , 1]

#Creating Training and test subsets
dxlcdfTrn4 <- dxlcdf4[indicesTraining2,]
colcdfTrn4 <- colcdf4[indicesTraining2]
dxlcdfTst4 <- dxlcdf4[indicesTest2,]
colcdfTst4 <- colcdf4[indicesTest2]

dxTrn4 <- xgb.DMatrix(subset(dxlcdfTrn4, select=-c(annRet, actualTerm, actualReturn)),
label=colcdfTrn4)
dxTst4 <- xgb.DMatrix(subset(dxlcdfTst4,select=-c(annRet, actualTerm, actualReturn)), label=colcdfTst4)
#Pick (annRet, actualTerm, actualReturn, total_pymnt) for performance assessment,
#but don't used it in the model

#watch the progress of learning through performance on these datasets
xgbWatchlist4 <- list(train = dxTrn4, eval = dxTst4)

#Perform cross-validation on training dataset to determine best model
xgbParamGrid4 <- expand.grid( max_depth = c(2, 5), eta = c(0.001, 0.01, 0.1))
```

```
xgbParam4 <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1,
colsample_bytree=0.6,eval_metric="error", eval_metric = "auc",scale_pos_weight = 8.50)

for(i in 1:nrow(xgbParamGrid4)) {
  xgb_tune4<- xgb.train(data=dxTrn4,xgbParam4,
                nrounds=1000, early_stopping_rounds = 10, xgbWatchlist4,
                eta=xgbParamGrid4$eta[i], max_depth=xgbParamGrid4$max_depth[i] )
  xgbParamGrid4$bestTree[i] <- xgb_tune4$evaluation_log[xgb_tune4$best_iteration]$iter
  xgbParamGrid4$bestPerf[i] <- xgb_tune4$evaluation_log[xgb_tune4$best_iteration]$eval_auc
}
```

Note : Results are simplified

```
train-error:0.795948	train-auc:0.565014	eval-error:0.796123	eval-auc:0.564330
Multiple eval metrics are present. Will use eval_auc for early stopping.
Will train until eval_auc hasn't improved in 10 rounds.

[2]	train-error:0.795948	train-auc:0.582212	eval-error:0.796123	eval-auc:0.587073
[3]	train-error:0.795948	train-auc:0.583721	eval-error:0.796123	eval-auc:0.589682
[4]	train-error:0.795948	train-auc:0.581070	eval-error:0.796123	eval-auc:0.587500
[5]	train-error:0.795948	train-auc:0.582989	eval-error:0.796123	eval-auc:0.587396
[6]	train-error:0.795948	train-auc:0.584745	eval-error:0.796123	eval-auc:0.589055
[7]	train-error:0.795948	train-auc:0.585393	eval-error:0.796123	eval-auc:0.588914
[8]	train-error:0.795948	train-auc:0.589275	eval-error:0.796123	eval-auc:0.592610
[9]	train-error:0.795948	train-auc:0.588552	eval-error:0.796123	eval-auc:0.591767
[10]	train-error:0.795948	train-auc:0.588995	eval-error:0.796123	eval-auc:0.593936
[11]	train-error:0.795948	train-auc:0.588373	eval-error:0.796123	eval-auc:0.592722
[12]	train-error:0.795948	train-auc:0.588133	eval-error:0.796123	eval-auc:0.593109
[13]	train-error:0.795948	train-auc:0.587640	eval-error:0.796123	eval-auc:0.592248
[14]	train-error:0.795948	train-auc:0.587215	eval-error:0.796123	eval-auc:0.592542
[15]	train-error:0.795948	train-auc:0.587921	eval-error:0.796123	eval-auc:0.593246
[16]	train-error:0.795948	train-auc:0.587923	eval-error:0.796123	eval-auc:0.592981
[17]	train-error:0.795948	train-auc:0.587779	eval-error:0.796123	eval-auc:0.592720
[18]	train-error:0.795948	train-auc:0.589170	eval-error:0.796123	eval-auc:0.593464
[19]	train-error:0.795948	train-auc:0.588357	eval-error:0.796123	eval-auc:0.593937
[20]	train-error:0.795948	train-auc:0.588890	eval-error:0.796123	eval-auc:0.593684
[21]	train-error:0.795948	train-auc:0.588778	eval-error:0.796123	eval-auc:0.593746
[22]	train-error:0.795948	train-auc:0.590229	eval-error:0.796123	eval-auc:0.594573
[23]	train-error:0.795948	train-auc:0.590700	eval-error:0.796123	eval-auc:0.594698
[24]	train-error:0.795948	train-auc:0.590670	eval-error:0.796123	eval-auc:0.595102
[25]	train-error:0.795948	train-auc:0.590947	eval-error:0.796123	eval-auc:0.595005
[26]	train-error:0.795948	train-auc:0.590911	eval-error:0.796123	eval-auc:0.595252
[27]	train-error:0.795948	train-auc:0.592564	eval-error:0.796123	eval-auc:0.596635
[28]	train-error:0.795948	train-auc:0.593231	eval-error:0.796123	eval-auc:0.596578
[29]	train-error:0.795948	train-auc:0.592674	eval-error:0.796123	eval-auc:0.596579
[30]	train-error:0.795948	train-auc:0.592332	eval-error:0.796123	eval-auc:0.596438
[31]	train-error:0.795948	train-auc:0.592337	eval-error:0.796123	eval-auc:0.596493
[32]	train-error:0.795948	train-auc:0.592009	eval-error:0.796123	eval-auc:0.596178
[33]	train-error:0.795948	train-auc:0.592008	eval-error:0.796123	eval-auc:0.595701
[34]	train-error:0.795948	train-auc:0.592074	eval-error:0.796123	eval-auc:0.595888
[35]	train-error:0.795948	train-auc:0.592311	eval-error:0.796123	eval-auc:0.595841
[36]	train-error:0.795948	train-auc:0.593109	eval-error:0.796123	eval-auc:0.596286
[37]	train-error:0.795948	train-auc:0.592589	eval-error:0.796123	eval-auc:0.595965
```

Stopping. Best iteration:
[27]	train-error:0.795948	train-auc:0.592564	eval-error:0.796123	eval-auc:0.596635

[1]	train-error:0.752668	train-auc:0.608876	eval-error:0.760343	eval-auc:0.588570
Multiple eval metrics are present. Will use eval_auc for early stopping.
Will train until eval_auc hasn't improved in 10 rounds.

[2]	train-error:0.753522	train-auc:0.627714	eval-error:0.763791	eval-auc:0.594017
[3]	train-error:0.758842	train-auc:0.637091	eval-error:0.767315	eval-auc:0.599624
[4]	train-error:0.762881	train-auc:0.639404	eval-error:0.770840	eval-auc:0.604844
[5]	train-error:0.762519	train-auc:0.643398	eval-error:0.770610	eval-auc:0.607324
[6]	train-error:0.760615	train-auc:0.647386	eval-error:0.768158	eval-auc:0.608004
[7]	train-error:0.762322	train-auc:0.650696	eval-error:0.769690	eval-auc:0.609067
[8]	train-error:0.763275	train-auc:0.655009	eval-error:0.770993	eval-auc:0.609810
[9]	train-error:0.761600	train-auc:0.658877	eval-error:0.768235	eval-auc:0.609308
[10]	train-error:0.762289	train-auc:0.661537	eval-error:0.769078	eval-auc:0.609086
[11]	train-error:0.762749	train-auc:0.664374	eval-error:0.768465	eval-auc:0.609543
[12]	train-error:0.761074	train-auc:0.667316	eval-error:0.767545	eval-auc:0.610382
[13]	train-error:0.759301	train-auc:0.668727	eval-error:0.767469	eval-auc:0.610364
[14]	train-error:0.759728	train-auc:0.671235	eval-error:0.767162	eval-auc:0.611536
[15]	train-error:0.759268	train-auc:0.673622	eval-error:0.766319	eval-auc:0.612268
[16]	train-error:0.758382	train-auc:0.675480	eval-error:0.766626	eval-auc:0.613771
[17]	train-error:0.756510	train-auc:0.677283	eval-error:0.764940	eval-auc:0.613730
[18]	train-error:0.754540	train-auc:0.678158	eval-error:0.763638	eval-auc:0.613705
[19]	train-error:0.754244	train-auc:0.680723	eval-error:0.763714	eval-auc:0.613243
[20]	train-error:0.752373	train-auc:0.681861	eval-error:0.762642	eval-auc:0.612508
[21]	train-error:0.750041	train-auc:0.682920	eval-error:0.761569	eval-auc:0.612600
[22]	train-error:0.748235	train-auc:0.684903	eval-error:0.760037	eval-auc:0.613110
[23]	train-error:0.748596	train-auc:0.685772	eval-error:0.760037	eval-auc:0.613620
[24]	train-error:0.747283	train-auc:0.687846	eval-error:0.759041	eval-auc:0.614001
[25]	train-error:0.746429	train-auc:0.689999	eval-error:0.758198	eval-auc:0.613501
[26]	train-error:0.746298	train-auc:0.690768	eval-error:0.758045	eval-auc:0.613702
[27]	train-error:0.744557	train-auc:0.692299	eval-error:0.756589	eval-auc:0.613236
[28]	train-error:0.743375	train-auc:0.694707	eval-error:0.756512	eval-auc:0.613413
[29]	train-error:0.741668	train-auc:0.696180	eval-error:0.755363	eval-auc:0.613984
[30]	train-error:0.739697	train-auc:0.698813	eval-error:0.754061	eval-auc:0.613547
[31]	train-error:0.738581	train-auc:0.700531	eval-error:0.752758	eval-auc:0.613599
[32]	train-error:0.738187	train-auc:0.701174	eval-error:0.752145	eval-auc:0.614208
[33]	train-error:0.736446	train-auc:0.704218	eval-error:0.751226	eval-auc:0.614223
[34]	train-error:0.735494	train-auc:0.704805	eval-error:0.750843	eval-auc:0.614343
[35]	train-error:0.732933	train-auc:0.706459	eval-error:0.748391	eval-auc:0.614569
[36]	train-error:0.731948	train-auc:0.707176	eval-error:0.747778	eval-auc:0.614773
[37]	train-error:0.730995	train-auc:0.708172	eval-error:0.747548	eval-auc:0.614734
[38]	train-error:0.730207	train-auc:0.709030	eval-error:0.746552	eval-auc:0.614707
[39]	train-error:0.728073	train-auc:0.710077	eval-error:0.744254	eval-auc:0.614708
[40]	train-error:0.726332	train-auc:0.710787	eval-error:0.742798	eval-auc:0.614994
[41]	train-error:0.725544	train-auc:0.711676	eval-error:0.742415	eval-auc:0.615146
[42]	train-error:0.724494	train-auc:0.712919	eval-error:0.741112	eval-auc:0.614536
[43]	train-error:0.722425	train-auc:0.714051	eval-error:0.739427	eval-auc:0.614848
[44]	train-error:0.721571	train-auc:0.714560	eval-error:0.738814	eval-auc:0.614943

[45]	train-error:0.720093	train-auc:0.716536	eval-error:0.737128	eval-auc:0.615136
[46]	train-error:0.719371	train-auc:0.717413	eval-error:0.736745	eval-auc:0.614991
[47]	train-error:0.717236	train-auc:0.718648	eval-error:0.735060	eval-auc:0.614878
[48]	train-error:0.716711	train-auc:0.719053	eval-error:0.734370	eval-auc:0.614894
[49]	train-error:0.715989	train-auc:0.719532	eval-error:0.733987	eval-auc:0.614978
[50]	train-error:0.714544	train-auc:0.720760	eval-error:0.732378	eval-auc:0.614524
[51]	train-error:0.712836	train-auc:0.721890	eval-error:0.732608	eval-auc:0.614358

Stopping. Best iteration:

[41]	train-error:0.725544	train-auc:0.711676	eval-error:0.742415	eval-auc:0.615146

#See the tune variable with parameter grid
xgbParamGrid4

```
> xgbParamGrid4
  max_depth   eta bestTree bestPerf
1         2 0.001       27 0.596635
2         5 0.001       15 0.609757
3         2 0.010       14 0.603004
4         5 0.010       56 0.613993
5         2 0.100       56 0.621286
6         5 0.100       41 0.615146
```

xgbParam_Best4 <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1, colsample_bytree=0.6, max_depth = 2, eta = 0.001,scale_pos_weight = 8.50)

# XGBOOST running the model with the best parameters found with the for loop
xgb_lsM4 <- xgb.train(xgbParam_Best4, dxTrn4, nrounds = xgb_tune4$best_iteration)

#XGBOOST evaluation of the model
#Using the predicting function to get the scores in the training data set
xpredTrn4<-predict(xgb_lsM4, dxTrn4)
head(xpredTrn4)

```
> #XGBOOST evaluation of the model
> #Using the predicting function to get the scores in the training data  set
> xpredTrn4<-predict(xgb_lsM4, dxTrn4)
> head(xpredTrn4)
[1] 0.5065721 0.5053604 0.5067171 0.5070637 0.5070215 0.5051140
```

#Using the predicting function to get the scores in the test data set
xpredTst4<-predict(xgb_lsM4, dxTst4)

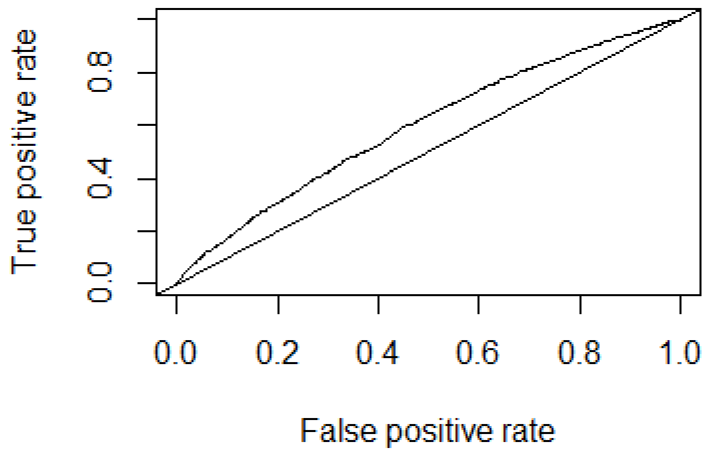
#confusion matrix
table(pred=as.numeric(xpredTst4>0.5), act=colcdfTst4)

#ROC, AUC performance
pred_xgb_lsM4<-prediction(xpredTst4, Testdf2$loan_status,
label.ordering = c("Fully Paid", ("Charged Off")))

aucPerf_xgb_lsM4<-performance(pred_xgb_lsM4, "tpr", "fpr")

#Plot ROC, AUC performance
plot(aucPerf_xgb_lsM4)

abline(a=0, b= 1)



#XGBOOST Deciles for grades C and lower M1
xpredTstM4<-predict(xgb_lsM4, dxTst4)

scoreTst_xgb_ls4 <- Testdf2 %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(score4=xpredTstM4)
scoreTst_xgb_ls4 <- scoreTst_xgb_ls4 %>% mutate(tile=ntile(-score4, 10))
scoreTst_xgb_ls4 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score4), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm),totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"), totE=sum(grade=="G"),)

```
> scoreTst_xgb_ls4 <- scoreTst_xgb_ls4 %>% mutate(tile=ntile(-score4, 10))
> scoreTst_xgb_ls4 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score4), numDefaults=sum(loan_status=="Ch
arged Off"),
+                                                   avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=
max(actualReturn), avgTer=mean(actualTerm),totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=s
um(grade=="F"), totE=sum(grade=="G"), )
# A tibble: 10 x 12
    tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totC  totD  totE  totF
   <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int>
 1     1  1306 0.510         406      4.66  -32.2   36.9   2.38     0   907     9    72
 2     2  1306 0.509         345      6.07  -33.3   40.2   2.32     0   968     3    54
 3     3  1305 0.508         315      6.24  -33.3   37.7   2.26   293   811     3    40
 4     4  1305 0.508         269      7.29  -33.3   31.3   2.23   228   828     3    37
 5     5  1305 0.507         288      5.11  -32.2   23.7   2.26  1238    60     0     1
 6     6  1305 0.507         251      5.67  -33.3   27.2   2.18  1131   123     0     8
 7     7  1305 0.507         244      6.25  -30.0   26.1   2.25  1185   114     0     5
 8     8  1305 0.506         199      6.37  -33.3   24.3   2.23  1305     0     0     0
 9     9  1305 0.505         187      6.18  -32.2   34.8   2.19  1305     0     0     0
10    10  1305 0.504         157      6.67  -31.1   24.2   2.21  1305     0     0     0
```

#XGBOOST Model 2 Actual returns for lower grades
#Finding which parameters work best with this model
we are predicting actualReturn as a numeric variable
#converting factor variables to dummy-variables to convert dataset to numerical
fdum5<-dummyVars(~.,lcdf_AR_LowGrades %>% select(-actualReturn))

dxlcdf5 <- predict(fdum5, lcdf_AR_LowGrades)

#Create Training and test subsets
dxlcdfTrn5 <- dxlcdf5[indicesTraining2,]

```
colcdfTrn5 <- lcdf_AR_LowGrades$actualReturn[indicesTraining2]
dxlcdfTst5 <- dxlcdf5[-indicesTraining2,]
colcdfTst5 <- lcdf_AR_LowGrades$actualReturn[indicesTest2]

#Creating Training and Test xgb matrices need it to run the model
dxTrn5 <- xgb.DMatrix(subset(dxlcdfTrn5, select=-c(annRet, actualTerm)), label=colcdfTrn5)
dxTst5 <- xgb.DMatrix(subset(dxlcdfTst5,select=-c(annRet, actualTerm)), label=colcdfTst5)

#watch the progress of learning through performance on these datasets
xgbWatchlist5 <- list(train = dxTrn5, eval = dxTst5)

#Expand the parameter grid to find the best for the model
xgbParamGrid5 <- expand.grid(
  max_depth = c(2, 5),
  eta = c(0.001, 0.01, 0.1) )

#Parameter list
xgbParam5 <- list (
  booster = "gbtree",
  objective = "reg:squarederror",
  #scale_pos_weight = 7.950299,
  min_child_weight=1,
  colsample_bytree=0.6
)
#Perform xgb training
for(i in 1:nrow(xgbParamGrid5)) {
  xgb_tune5<- xgb.train(data=dxTrn5,xgbParam5,
                        nrounds=1000, early_stopping_rounds = 10, xgbWatchlist5,
                        eta=xgbParamGrid5$eta[i], max_depth=xgbParamGrid5$max_depth[i] )
  xgbParamGrid5$bestTree[i] <- xgb_tune5$evaluation_log[xgb_tune5$best_iteration]$iter
  xgbParamGrid5$bestPerf[i] <- xgb_tune5$evaluation_log[xgb_tune5$best_iteration]$eval_rmse
}
```

Note : Results are simplified

```
train-rmse:11.700222    eval-rmse:11.747152
Multiple eval metrics are present. Will use eval_rmse for early stopping.
Will train until eval_rmse hasn't improved in 10 rounds.

[2]     train-rmse:10.741508    eval-rmse:10.797511
[3]     train-rmse:9.896646     eval-rmse:9.960797
[4]     train-rmse:9.153386     eval-rmse:9.226575
[5]     train-rmse:8.503037     eval-rmse:8.586250
[6]     train-rmse:7.937613     eval-rmse:8.031483
[7]     train-rmse:7.446745     eval-rmse:7.551983
[8]     train-rmse:7.343415     eval-rmse:7.457603
[9]     train-rmse:6.931310     eval-rmse:7.057893
[10]    train-rmse:6.579497     eval-rmse:6.716870
[11]    train-rmse:6.515373     eval-rmse:6.659767
[12]    train-rmse:6.221575     eval-rmse:6.375267
[13]    train-rmse:5.973438     eval-rmse:6.138580
[14]    train-rmse:5.765673     eval-rmse:5.940746
[15]    train-rmse:5.590408     eval-rmse:5.773128
```

```
[16]	train-rmse:5.440906	eval-rmse:5.632814
[17]	train-rmse:5.417535	eval-rmse:5.612856
[18]	train-rmse:5.295611	eval-rmse:5.500779
[19]	train-rmse:5.194365	eval-rmse:5.408567
[20]	train-rmse:5.110200	eval-rmse:5.333051
[21]	train-rmse:5.037741	eval-rmse:5.268813
[22]	train-rmse:4.978216	eval-rmse:5.215764
[23]	train-rmse:4.929973	eval-rmse:5.172333
[24]	train-rmse:4.889657	eval-rmse:5.138330
[25]	train-rmse:4.855788	eval-rmse:5.110473
[26]	train-rmse:4.827390	eval-rmse:5.087634
[27]	train-rmse:4.804537	eval-rmse:5.068079
[28]	train-rmse:4.782790	eval-rmse:5.051438
[29]	train-rmse:4.762628	eval-rmse:5.036211
[30]	train-rmse:4.743316	eval-rmse:5.024541
[31]	train-rmse:4.737270	eval-rmse:5.023823
[32]	train-rmse:4.725092	eval-rmse:5.016268
[33]	train-rmse:4.712137	eval-rmse:5.008268
[34]	train-rmse:4.702710	eval-rmse:5.002246
[35]	train-rmse:4.691658	eval-rmse:4.997258
[36]	train-rmse:4.681256	eval-rmse:4.993438
[37]	train-rmse:4.674147	eval-rmse:4.989196
[38]	train-rmse:4.668905	eval-rmse:4.987413
[39]	train-rmse:4.662926	eval-rmse:4.986079
[40]	train-rmse:4.658926	eval-rmse:4.985519
[41]	train-rmse:4.651308	eval-rmse:4.981722
[42]	train-rmse:4.648942	eval-rmse:4.981273
[43]	train-rmse:4.645028	eval-rmse:4.979413
[44]	train-rmse:4.638552	eval-rmse:4.979047
[45]	train-rmse:4.635851	eval-rmse:4.978304
[46]	train-rmse:4.634536	eval-rmse:4.978135
[47]	train-rmse:4.629727	eval-rmse:4.976712
[48]	train-rmse:4.626838	eval-rmse:4.977036
[49]	train-rmse:4.625494	eval-rmse:4.977039
[50]	train-rmse:4.619664	eval-rmse:4.977470
[51]	train-rmse:4.616091	eval-rmse:4.977742
[52]	train-rmse:4.611224	eval-rmse:4.975496
[53]	train-rmse:4.608206	eval-rmse:4.975776
[54]	train-rmse:4.604103	eval-rmse:4.974890
[55]	train-rmse:4.601810	eval-rmse:4.974590
[56]	train-rmse:4.600368	eval-rmse:4.974729
[57]	train-rmse:4.598578	eval-rmse:4.975019
[58]	train-rmse:4.597795	eval-rmse:4.974772
[59]	train-rmse:4.597065	eval-rmse:4.975035
[60]	train-rmse:4.596221	eval-rmse:4.974833
[61]	train-rmse:4.595340	eval-rmse:4.975124
[62]	train-rmse:4.594477	eval-rmse:4.974759
[63]	train-rmse:4.594092	eval-rmse:4.974795
[64]	train-rmse:4.592149	eval-rmse:4.974081
[65]	train-rmse:4.590569	eval-rmse:4.974404
[66]	train-rmse:4.587722	eval-rmse:4.974605
```

[67] train-rmse:4.584843 eval-rmse:4.974868
[68] train-rmse:4.583748 eval-rmse:4.975747
[69] train-rmse:4.581357 eval-rmse:4.975651
[70] train-rmse:4.576166 eval-rmse:4.975430
[71] train-rmse:4.574084 eval-rmse:4.975587
[72] train-rmse:4.573107 eval-rmse:4.975377
[73] train-rmse:4.571788 eval-rmse:4.974962
[74] train-rmse:4.570568 eval-rmse:4.974908
Stopping. Best iteration:
[64] train-rmse:4.592149 eval-rmse:4.974081

#Deploy the tune parameters
xgbParamGrid5

```
> xgbParamGrid5
  max_depth   eta bestTree bestPerf
1         2 0.001     1000 6.780442
2         5 0.001     1000 6.755215
3         2 0.010      797 4.991000
4         5 0.010      585 4.967330
5         2 0.100      106 4.989126
6         5 0.100       64 4.974081
```

```
xgbParam6 <- list(
 max_depth = 2,
 eta = 0.100,
 booster = "gbtree",
 objective = "reg:squarederror",
 #scale_pos_weight = 7.950299,
 min_child_weight=1,
 colsample_bytree=0.6
)
```

# XGBOOST running the model with the best parameters found with the for loop
xgb_lsM5 <- xgb.train(xgbParam6, dxTrn5, nrounds = xgb_tune5$best_iteration)

#XGBOOST evaluation of the model
#Using the predicting function to get the scores in the training data set
xpredTrn5<-predict(xgb_lsM5, dxTrn5)
head(xpredTrn5)

```
> #Using the predicting function to get the scores in the training data  set
> xpredTrn5<-predict(xgb_lsM5, dxTrn5)
> head(xpredTrn5)
[1] 10.147585 10.166124 10.378598  9.111142 10.147690  8.967637
```

#Using the predicting function to get the scores in the test data set
xpredTst5<-predict(xgb_lsM5, dxTst5)

#Error
sqrt(mean((xpredTst5- colcdfTst5)^2)) #4.994001

#XGBOOST M2 Actual Returns Performance of grades C and lower (Boosted) xpredTstM5<-predict(xgb_lsM5, dxTst5)

scoreTst_xgb_ls5 <- Testdf2 %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(score5=xpredTstM5)

scoreTst_xgb_ls5 <- scoreTst_xgb_ls5 %>% mutate(tile=ntile(-score5, 10))
scoreTst_xgb_ls5 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score5), numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm),totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"), totE=sum(grade=="G"))

```
> scoreTst_xgb_ls5 <- scoreTst_xgb_ls5 %>% mutate(tile=ntile(-score5, 10))
> scoreTst_xgb_ls5 %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score5), numDefaults=sum(loan_status=="Ch
arged Off"),
+                                                   avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=
max(actualReturn), avgTer=mean(actualTerm),totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=s
um(grade=="F"), totE=sum(grade=="G"))
# A tibble: 10 x 12
    tile count  avgSc numDefaults avgActRet minRet maxRet avgTer  totC  totD  totE  totF
   <int> <int>  <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int>
 1     1  1306  13.9            0     14.3     0     40.2   2.01     0   514     9   141
 2     2  1306  12.3            0     12.4     0     33.6   2.03     0  1214     0     0
 3     3  1305  11.4            0     11.4     0     31.3   2.07   214  1091     0     0
 4     4  1305  10.4            0     10.7     0     27.8   2.01  1209    96     0     0
 5     5  1305   9.94           0     10.0     0     34.8   2.08  1305     0     0     0
 6     6  1305   9.49           0      9.46    0     24.3   2.09  1304     1     0     0
 7     7  1305   9.06           0      8.99    0     24.2   2.05  1305     0     0     0
 8     8  1305   7.97          51      7.75  -30.7   20.0   2.18  1260    12     2    23
 9     9  1305 -11.6         1305    -11.8   -33.3   11.6   3       612   457     6    49
10    10  1305 -12.6         1305    -12.8   -33.3   10.2   3       781   426     1     4
```

#XGBOOST -Consider Top d deciles from Model 2(actualReturn), ranked by M1 scores(loan_Status) all grades
#Initiate decile (d) as 1
d=1
pRetSc6 <- scoreTst_xgb_ls5 %>% mutate(poScore=scoreTst_xgb_ls4$score4)
pRet_d6 <- pRetSc6 %>% filter(tile<=d)
pRet_d6<- pRet_d6 %>% mutate(tile2=ntile(-poScore, 10))

pRet_d6 %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(score5),numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"), totE=sum(grade=="G"))

```
> pRet_d6 %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(score5),
+                                            numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualRetur
n), minRet=min(actualReturn),
+                                            maxRet=max(actualReturn), avgTer=mean(actualTerm), totC=sum(grade=
="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F"), totE=sum(grade=="G"))
# A tibble: 10 x 12
   tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int>
 1     1   131       14.0           0      15.1  10.3    36.9   1.99     0    21     2    19
 2     2   131       14.0           0      14.5   9.37   23.6   2.10     0    34     2    18
 3     3   131       14.1           0      14.5   0      40.2   2.05     0    36     1    16
 4     4   131       13.9           0      14.4   9.81   23.0   2.03     0    61     0    13
 5     5   131       14.0           0      13.8   8.22   24.4   2.05     0    48     0    15
 6     6   131       13.8           0      14.5   2.70   37.7   1.85     0    77     1    11
 7     7   130       13.9           0      14.5   0      25.2   1.89     0    69     1    14
 8     8   130       14.1           0      14.4   5.24   23.9   1.90     0    14     2    10
 9     9   130       13.7           0      13.7   9.83   20.9   2.07     0    80     0    11
10    10   130       13.9           0      13.6   2.10   22.2   2.16     0    74     0    14
```

#performance of glm model for low grade loans in deciles for M1
#according to question number 1
#yTrn<-factor(if_else(Trainingdf$loan_status=="Fully Paid", '1', '0'))
#xcdftrn<- model.matrix(~ loan_status+ actualTerm + annRet + actualReturn - 1, Trainingdf)
glmls_cv<- cv.glmnet(data.matrix(xcdftrn), yTrn, family="binomial")

lg_xDTrn<-model.matrix(~ loan_status+ actualTerm + annRet + actualReturn - 1, lg_lcdfTrn)
lg_predLS_glm <- lg_lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
 mutate(lg_score=predict(glmls_cv,data.matrix(lg_xDTrn), s="lambda.min", type="response"))

lg_predLS_glm<- lg_predLS_glm%>% mutate(tile=ntile(-lg_score, 10))

lg_predLS_glm %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(lg_score),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn),   maxRet=max(actualReturn), avgTerm=mean(actualTerm),
totA=sum(grade=="A"), totB=sum(grade=="B"), totC=sum(grade=="C"), totD=sum(grade=="D"),
totE=sum(grade=="E"), totF=sum(grade=="F"))

```
> lg_predLS_glm %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(lg_score),    numDefaults=sum(loan_sta
tus=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualReturn),    maxRet=max(actualReturn), avgTerm=m
ean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum
(grade=="E"), totF=sum(grade=="F") )
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3052    1.00              0     10.8     0     40.8    2.05     0     0  1968   826   218    35
 2     2  3052    1.00              0     10.8     0     28.8    2.06     0     0  1926   833   247    38
 3     3  3052    1.00              0     10.7     0     34.2    2.08     0     0  1896   879   232    37
 4     4  3051    1.00              0     10.8     0     44.4    2.05     0     0  1889   875   235    49
 5     5  3051    1.00              0     10.6     0     37.7    2.07     0     0  1982   812   211    41
 6     6  3051    1.00              0     10.7     0     34.8    2.08     0     0  1923   857   226    43
 7     7  3051    1.00              0     10.8     0     36.6    2.04     0     0  1891   901   206    51
 8     8  3051    0.947           161      9.54  -27.6   29.6    2.14     0     0  1900   850   261    33
 9     9  3051    0.00141        3051    -12.0   -33.3   13.7    3        0     0  1631   993   337    78
10    10  3051    0.00141        3051    -12.2   -33.3   12.4    3        0     0  1625   965   366    88
```

#Actual Returns selected by grade M2 low grades

#Performance of glm model for lambda min
#According to question number 2
df4_glm <-model.matrix(~ loan_status+ actualTerm + annRet + actualReturn - 1, Trainingdf)
glmRet_cv <- cv.glmnet(data.matrix(df4_glm), Trainingdf$actualReturn, family="gaussian")

lgXDTRN2<- model.matrix(~ loan_status+ actualTerm + annRet + actualReturn - 1, lg_lcdfTrn)

```
lg_predRet_Trn_glm <- lg_lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate)
%>%
  mutate(lg_predRet_glm = predict(glmRet_cv, data.matrix(lgXDTRN2),s="lambda.min" ))

lg_predRet_Trn_glm<- lg_predRet_Trn_glm%>% mutate(tile=ntile(-lg_predRet_glm, 10))

lg_predRet_Trn_glm %>% group_by(tile) %>%  summarise(count=n(),
avgpredRet=mean(lg_predRet_glm),  numDefaults=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn),   maxRet=max(actualReturn),
avgTerm=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="B" ), totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

```
> lg_predRet_Trn_glm %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(lg_predRet_glm),
+                       numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actua
lReturn),
+                       maxRet=max(actualReturn), avgTerm=mean(actualTerm), totA=sum(grade=="A"),
+                       totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), to
tF=sum(grade=="F") )
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3052      16.7            0     17.0   14.5   44.4    0.835     0     0   485  1604   740   187
 2     2  3052      13.2            2     13.4   12.5   14.5    1.28      0     0  1459  1186   283   122
 3     3  3052      11.6            6     11.8   11.2   12.5    1.63      0     0  1853   790   397    12
 4     4  3051      10.5            7     10.6   10.1   11.2    2.07      0     0  1734   991   324     2
 5     5  3051       9.51          13      9.64   9.14  10.1    2.41      0     0  1687  1288    71     5
 6     6  3051       8.58          20      8.67   8.24   9.14   2.68      0     0  2182   860     7     2
 7     7  3051       7.72          29      7.79   7.36   8.24   2.85      0     0  2990    52     6     3
 8     8  3051       6.80         189      6.84   4.92   7.36   2.91      0     0  2934    84    24     9
 9     9  3051      -4.74        2946     -5.04 -13.1    4.90   2.91      0     0  1755   901   322    63
10    10  3051     -19.5         3051    -20.3  -33.3  -13.1    3         0     0  1552  1035   365    88
```

```
#glm - Consider Top d deciles from Model 2(actualReturn), ranked by M1 scores(loan_Status) low grades
#Initiate decile (d) as 1
d=1
lg_pRetSc_glm <- lg_predRet_Trn_glm %>% mutate(poScore=lg_predLS_glm$lg_score)
lg_pRet_d_glm <- lg_pRetSc_glm %>% filter(tile<=d)
lg_pRet_d_glm<- lg_pRet_d_glm %>% mutate(tile2=ntile(-poScore, 20))


lg_pRet_d_glm %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(lg_predRet_glm),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn),
minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm),
totA=sum(grade=="A"), totB=sum(grade=="B" ),totC=sum(grade=="C"), totD=sum(grade=="D"),
totE=sum(grade=="E"), totF=sum(grade=="F") )
```

```
> lg_pRet_d_glm %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(lg_predRet_glm),
+                                                numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actua
lReturn), minRet=min(actualReturn),
+                                                maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade
=="A"), totB=sum(grade=="B" ),
+                                                totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"),
 totF=sum(grade=="F") )
# A tibble: 20 x 14
   tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1   153       16.6           0      16.9   14.5   40.8  0.934     0     0    26    72    42    12
 2     2   153       16.5           0      16.8   14.5   25.2  0.882     0     0    16    95    32     8
 3     3   153       16.7           0      17.0   14.5   26.6  0.829     0     0    23    81    39     8
 4     4   153       16.8           0      17.2   14.6   26.2  0.772     0     0    22    79    42     8
 5     5   153       16.7           0      17.0   14.5   28.8  0.824     0     0    27    82    32     8
 6     6   153       16.5           0      16.9   14.5   32.9  0.847     0     0    25    79    40     8
 7     7   153       16.6           0      16.9   14.5   34.2  0.910     0     0    22    82    40     6
 8     8   153       16.8           0      17.1   14.5   33.6  0.833     0     0    23    84    30    13
 9     9   153       16.9           0      17.2   14.5   28.1  0.834     0     0    23    83    31    16
10    10   153       17.1           0      17.5   14.5   44.4  0.808     0     0    24    77    38    11
11    11   153       16.6           0      16.9   14.5   37.7  0.804     0     0    22    89    29    11
12    12   153       16.8           0      17.2   14.5   26.2  0.822     0     0    27    69    47     7
13    13   152       16.7           0      17.0   14.5   24.8  0.773     0     0    25    84    34     8
14    14   152       16.9           0      17.2   14.5   34.8  0.827     0     0    27    76    41     8
15    15   152       16.3           0      16.7   14.5   23.4  0.892     0     0    30    78    32    11
16    16   152       16.6           0      17.0   14.5   23.8  0.795     0     0    26    81    33    11
17    17   152       16.6           0      16.9   14.5   36.6  0.802     0     0    30    80    33     8
18    18   152       16.6           0      16.9   14.5   31.3  0.794     0     0    25    82    36     8
19    19   152       17.1           0      17.4   14.5   29.6  0.797     0     0    22    77    43     8
20    20   152       16.6           0      16.9   14.5   26.8  0.914     0     0    20    74    46     9
```

Q.6 Considering all your results, which approach(s) would you recommend for investing in LC loans? Explain your rationale.

Deciding to invest in LC Loans has its pros and cons. A borrower might receive the full amount they're asking for or only a portion of it. In the case of the latter, the remaining portion of the loan may be funded by one or more investors in the peer lending marketplace. It's quite typical for a loan to have multiple sources, with monthly repayments being made to each of the individual sources.

For lenders, the loans generate income in the form of interest, which can often exceed the rates that can be earned through other vehicles, such as savings accounts and CDs. In addition, the monthly interest payments a lender receives may even earn a higher return than a stock market investment. For borrowers, P2P loans represent an alternative source of financing—especially useful if they are unable to get approval from standard financial intermediaries. They often receive a more favorable interest rate or terms on the loan than from conventional sources too.

Considering the results that we earn, we can start by looking at loan grades data and the actual return data. Many peer-to-peer investors report annual investment returns of greater than 10%, and it is shown multiple times on the data. Investing in LC loans means that we have more control over the specific investments that we want to do. We can select notes based on several criteria such as loan type, credit score, payment ratio, etc. Aside from all these advantages, P2P investments are generally unsecured, so there is no collateral to go after in the event of default. It is conceivable that you could lose 100% of your investment on an early-term default

Therefore, the best approach for investing in LC Loans is to diversify the investment. Spreading, rather than fully investing only in one place, is seen as a good solution to lower the risk of losing your investment. Investing in different grades is one solution of diversification, as grades are not the major factor of your return. By mixing in positions in lower grade loans, you can increase those returns to double digits. The idea is to spread your capital across different loan grades, and to avoid those that are the highest risk.

Citations (for Q6)

Curtis, G. (2021, October 21). The Best Ways to Borrow Money. Investopedia, from https://www.investopedia.com/articles/basics/07/financing-options.asp

Moneyunder30.com. (n.d.)., from https://www.moneyunder30.com/invest-in-peer-to-peer-loans