Lending Club Assignment 2

GLM, RF, GBM Models for Loan Investments

Lina Quiceno Bejarano, Joshua Pollack, Lauren Sansone

All Necessary Libraries

#Importing the data set

```
lcDataSample5m <- read_csv("lcDataSample5m.csv")
lcdf_AR <- lcDataSample5m
```

#Variable Modifications

```
#Remove loans with a status other than charged off and Fully Paid
lcdf_AR <- lcdf_AR %>% filter(loan_status == "Fully Paid" | loan_status == "Charged Off")

#regrouping purpose
lcdf_AR$purpose <- fct_recode(lcdf_AR$purpose, other="wedding", other="renewable_energy")

#Filtering home ownership
lcdf_AR <- lcdf_AR %>% filter(home_ownership == "MORTGAGE"
                        | home_ownership == "OWN"
                        | home_ownership == "RENT")

lcdf_AR <- lcdf_AR %>% mutate_if(is.character, as.factor)

lcdf_AR <- lcdf_AR %>% mutate(loan_status=as.factor(loan_status)) #this is a redundancy

##Building annRet
lcdf_AR$annRet <- ((lcdf_AR$total_pymnt -lcdf_AR$funded_amnt)/lcdf_AR$funded_amnt)*(12/36)*100
```

Calculating actual loan returns

```
#  First step is to past "01-" to the character string, to get something like "01-Dec-2018", i.e. first
lcdf_AR$last_pymnt_d<-paste(lcdf_AR$last_pymnt_d, "-01", sep = "")
#     Then convert this character to a date type variable
lcdf_AR$last_pymnt_d<-parse_date_time(lcdf_AR$last_pymnt_d,  "myd")

#Determining Acutal Term or setting term to 3 years
lcdf_AR$actualTerm <- ifelse(lcdf_AR$loan_status=="Fully Paid", as.duration(lcdf_AR$issue_d  %--% lcdf_A

#Then, considering this actual term, the actual annual return is
lcdf_AR$actualReturn <- ifelse(lcdf_AR$actualTerm>0, ((lcdf_AR$total_pymnt -lcdf_AR$funded_amnt)/lcdf_AR
```

#Removing Variables due to leakage

```
## Removing variables for data leakage
lcdf_AR <- lcdf_AR %>% select(-c(acc_now_delinq, collection_recovery_fee, debt_settlement_flag, debt_se
```

#Removing variables for other reasons

```
## Removing variables for other reasons
lcdf_AR <- lcdf_AR %>% select(-c(addr_state, all_util, annual_inc_joint, application_type, desc, dti_jo
```

#Replacing Some Missing Values

```
lcdf_AR<- lcdf_AR %>% replace_na(list(mths_since_last_delinq=500, bc_open_to_buy=median(lcdf_AR$bc_open
```

#Removing Variables with >60% missing values

```
    #remove variables which have more than 60% missing values
nm<-names(lcdf_AR)[colMeans(is.na(lcdf_AR))>0.6]
lcdf_AR <- lcdf_AR %>% select(-nm)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(nm)` instead of `nm` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

#Removing Variables with lots of zeros

```
lcdf_AR <- lcdf_AR %>% select(-c(collections_12_mths_ex_med, chargeoff_within_12_mths, delinq_amnt, num
```

#Spliting Data into Training, Validation, and Test Sets

```
set.seed(123)

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

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(lcdf_AR))
sampleSizeValidation <- floor(fractionValidation * nrow(lcdf_AR))
sampleSizeTest       <- floor(fractionTest       * nrow(lcdf_AR))

# Create the randomly-sampled indices for the dataframe. Use setdiff() to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(lcdf_AR)), size=sampleSizeTraining))
indicesNotTraining <- setdiff(seq_len(nrow(lcdf_AR)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeValidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

# Finally, output the three dataframes for training, validation and test.
lcdfTrn_AR <- lcdf_AR[indicesTraining, ]
lcdfVal_AR <- lcdf_AR[indicesValidation, ]
lcdfTst_AR <- lcdf_AR[indicesTest, ]
```

Question 1

#Linear model turning into factor

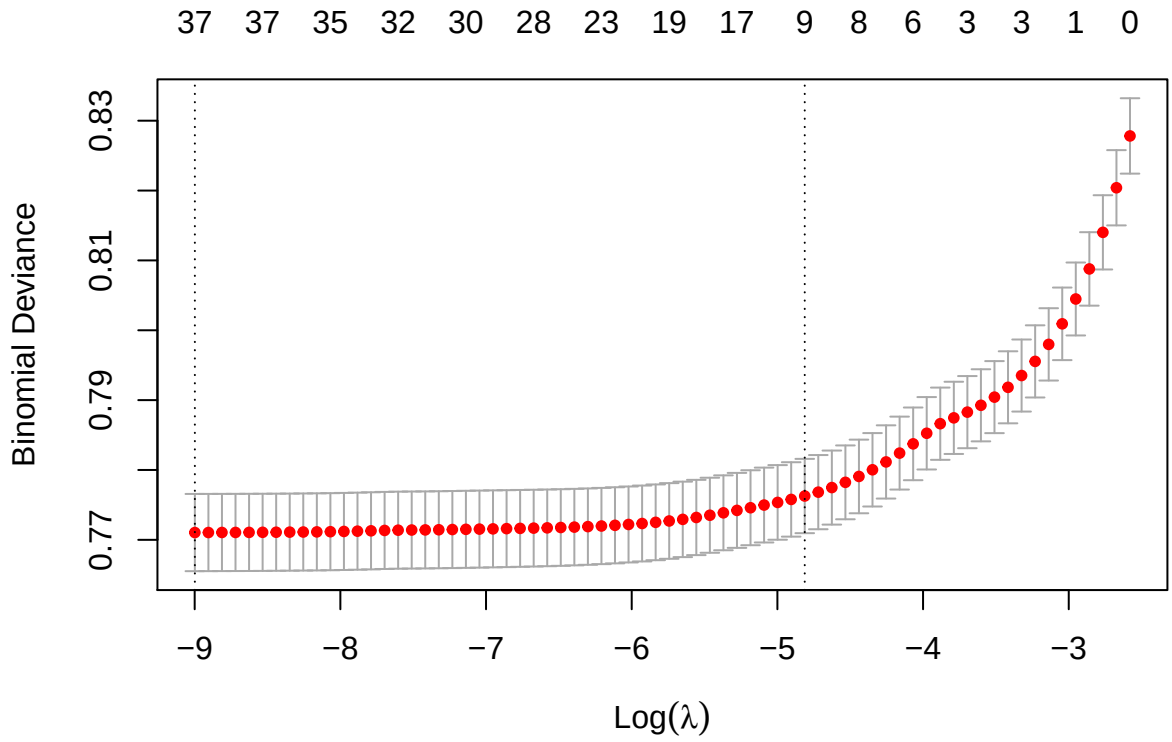
```
#Make sure that "fully paid" is 1 in the factor variable
levels(lcdfTrn_AR$loan_status)
```

```
## [1] "Charged Off" "Fully Paid"
```

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

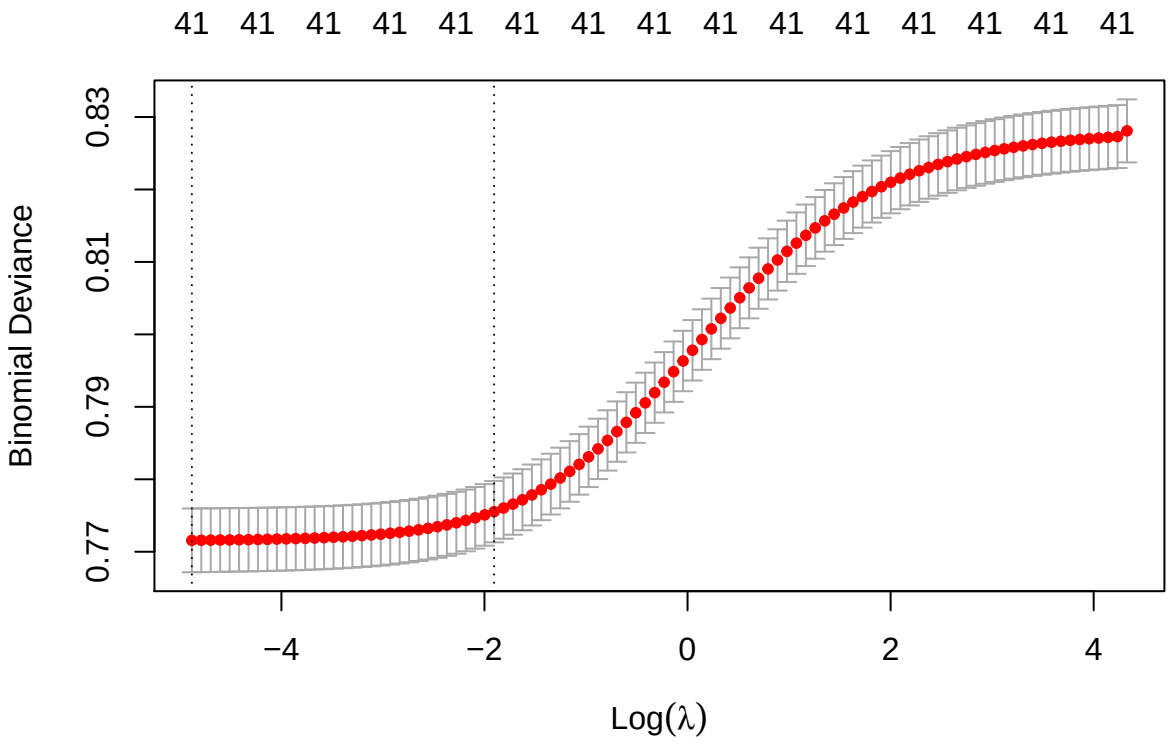
#Removing variables and running cv

```
#remove variables that would not be x variables
xDTrn<-lcdfTrn_AR %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)
#running cross validation
glmls_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial")

#plotting the model
plot(glmls_cv)
```

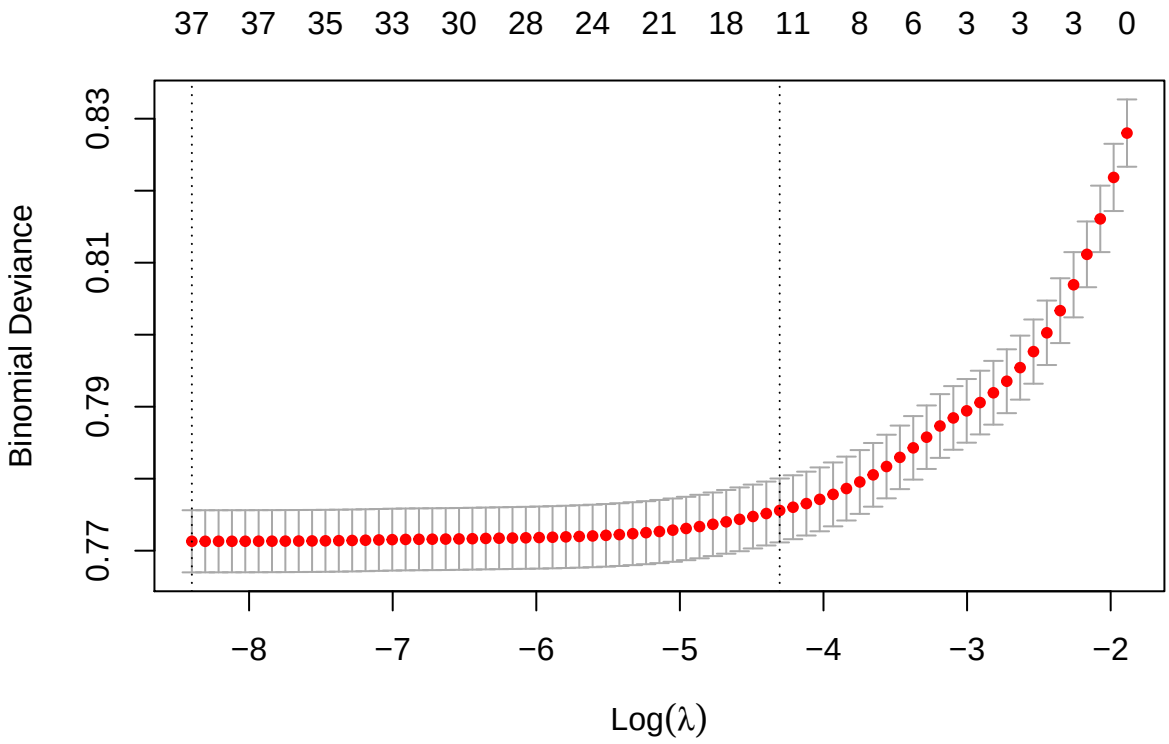


#Ridge and Lasso

```
#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] 0.0001235625
```

```r
#finding within 1 standard error
glmls_cv$lambda.1se
```

```
## [1] 0.008129587
```

#Getting lambda values

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

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                3.719035e+00
## loan_amnt                  .
## int_rate                  -4.742287e-02
## installment               -3.718904e-04
## grade                     -1.206520e-01
## sub_grade                 -1.983042e-02
## emp_length                -1.321321e-02
## home_ownership            -8.319472e-02
## annual_inc                 3.897650e-07
## verification_status       -3.965618e-02
## purpose                    7.990464e-03
## dti                       -1.665200e-02
## earliest_cr_line           6.158517e-05
## initial_list_status        2.186276e-02
## total_rev_hi_lim           4.173962e-06
## acc_open_past_24mths      -4.454970e-02
## avg_cur_bal                .
## bc_open_to_buy            -8.040990e-06
## bc_util                   -4.161641e-04
## mo_sin_old_il_acct        -9.690272e-05
## mo_sin_old_rev_tl_op       4.502810e-04
## mo_sin_rcnt_rev_tl_op      1.245176e-04
## mo_sin_rcnt_tl             1.064185e-03
## mths_since_recent_bc      -1.369398e-04
## mths_since_recent_inq      3.817099e-03
## num_accts_ever_120_pd     -1.966169e-02
## num_actv_bc_tl            -2.542256e-02
## num_actv_rev_tl            .
## num_bc_sats                .
## num_bc_tl                 -1.778553e-02
## num_il_tl                  2.691431e-03
## num_op_rev_tl             -5.192013e-03
## num_rev_accts              9.696162e-03
## num_rev_tl_bal_gt_0       -2.175282e-02
## num_sats                   5.629635e-03
## num_tl_op_past_12m         7.569597e-03
## pct_tl_nvr_dlq             3.520365e-04
## percent_bc_gt_75          -2.420667e-03
## tot_hi_cred_lim            8.056042e-07
## total_bal_ex_mort         -3.812187e-06
## total_bc_limit             7.733130e-06
## total_il_high_credit_limit 4.092791e-06
```

```
#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")
```

```
##                     row column         value
## 1           (Intercept)      1  3.277036e+00
## 2              int_rate      1 -3.545125e-02
## 3                 grade      1 -8.297652e-02
## 4             sub_grade      1 -4.212040e-02
## 5        home_ownership      1 -3.485478e-02
## 6                   dti      1 -8.132383e-03
## 7  acc_open_past_24mths      1 -2.251697e-02
## 8   num_rev_tl_bal_gt_0      1 -1.473326e-02
## 9      percent_bc_gt_75      1 -6.466641e-04
## 10      tot_hi_cred_lim      1  6.010087e-07
```

```
#values corresponding to graph
glmls_cv$glmnet.fit
```

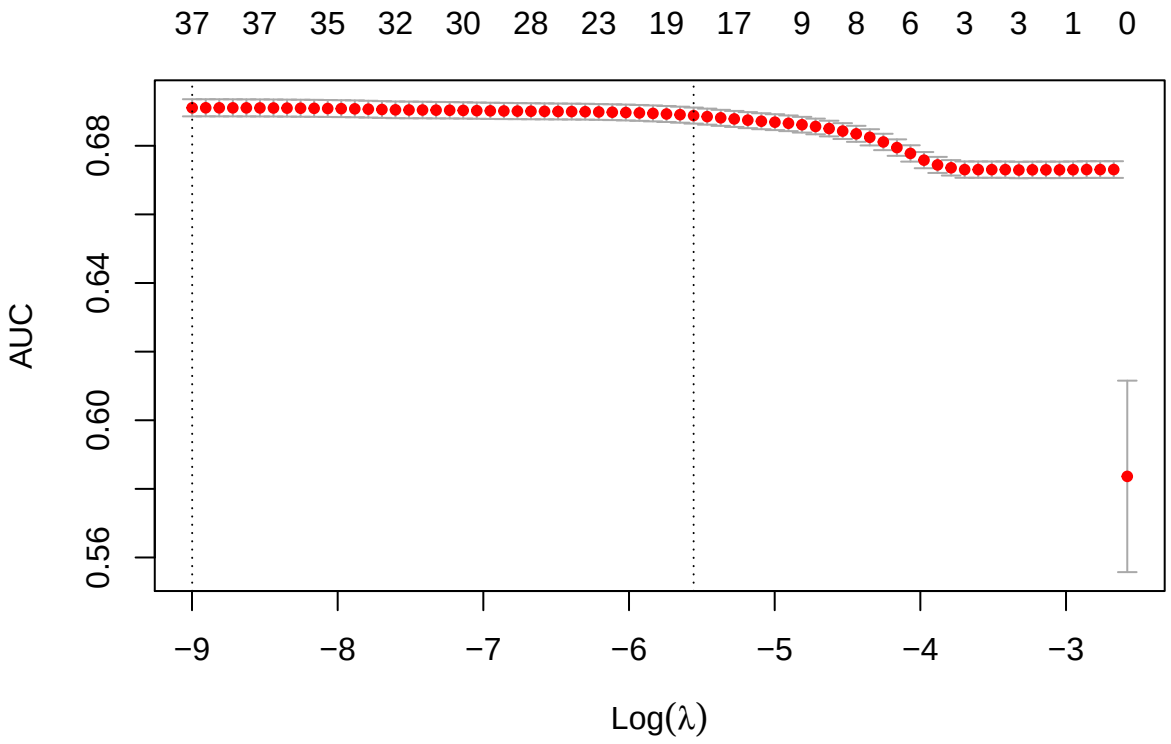
```
##
## Call:  glmnet(x = data.matrix(xDTrn), y = yTrn, family = "binomial")
##
##    Df %Dev   Lambda
## 1   0 0.00 0.075820
## 2   1 0.94 0.069080
## 3   1 1.71 0.062940
## 4   1 2.34 0.057350
## 5   1 2.86 0.052260
## 6   1 3.29 0.047620
## 7   1 3.65 0.043390
## 8   3 3.94 0.039530
## 9   3 4.19 0.036020
## 10  3 4.40 0.032820
## 11  3 4.57 0.029900
## 12  3 4.71 0.027250
## 13  3 4.83 0.024830
## 14  3 4.93 0.022620
## 15  4 5.03 0.020610
## 16  6 5.20 0.018780
## 17  6 5.38 0.017110
## 18  7 5.54 0.015590
## 19  8 5.70 0.014210
## 20  8 5.84 0.012940
## 21  8 5.95 0.011790
## 22  9 6.05 0.010750
## 23  9 6.15 0.009792
## 24  9 6.23 0.008922
## 25  9 6.30 0.008130
## 26 10 6.36 0.007407
## 27 10 6.41 0.006749
## 28 12 6.46 0.006150
```

```
## 29 16 6.51 0.005603
## 30 17 6.57 0.005106
## 31 18 6.61 0.004652
## 32 17 6.66 0.004239
## 33 17 6.70 0.003862
## 34 18 6.73 0.003519
## 35 19 6.76 0.003206
## 36 20 6.79 0.002922
## 37 20 6.81 0.002662
## 38 21 6.83 0.002426
## 39 22 6.85 0.002210
## 40 23 6.86 0.002014
## 41 23 6.88 0.001835
## 42 24 6.89 0.001672
## 43 25 6.90 0.001523
## 44 26 6.91 0.001388
## 45 28 6.92 0.001265
## 46 28 6.93 0.001152
...
```

```
#finding lambda that creates the optimal deviance
which(glmls_cv$lambda == glmls_cv$lambda.1se)
```

```
## [1] 25
```

```
#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)
```



#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)
```

```
##             1
## [1,] 1.523926
## [2,] 1.538123
## [3,] 2.101886
## [4,] 1.667319
## [5,] 1.349526
## [6,] 2.496988
```

```
#making predictions with probability for Lambda min Training
glmPredls_pMin=predict(glmls_cv,data.matrix(xDTrn), s="lambda.min", type="response" )
head(glmPredls_pMin)
```

```
##              1
## [1,] 0.8211159
## [2,] 0.8231917
## [3,] 0.8910864
## [4,] 0.8412181
## [5,] 0.7940521
## [6,] 0.9239304
```

```
#making predictions for Lambda.se
glmPredls_SE=predict ( glmls_cv,data.matrix(xDTrn), s="lambda.1se" )
head(glmPredls_SE)
```

```
##             1
## [1,] 1.640010
## [2,] 1.660560
## [3,] 2.228307
## [4,] 1.624277
## [5,] 1.410952
## [6,] 2.058022
```

```
#making predictions with probability for Lambda.1se
glmPredls_pSE=predict(glmls_cv,data.matrix(xDTrn), s="lambda.1se", type="response" )
head(glmPredls_pSE)
```

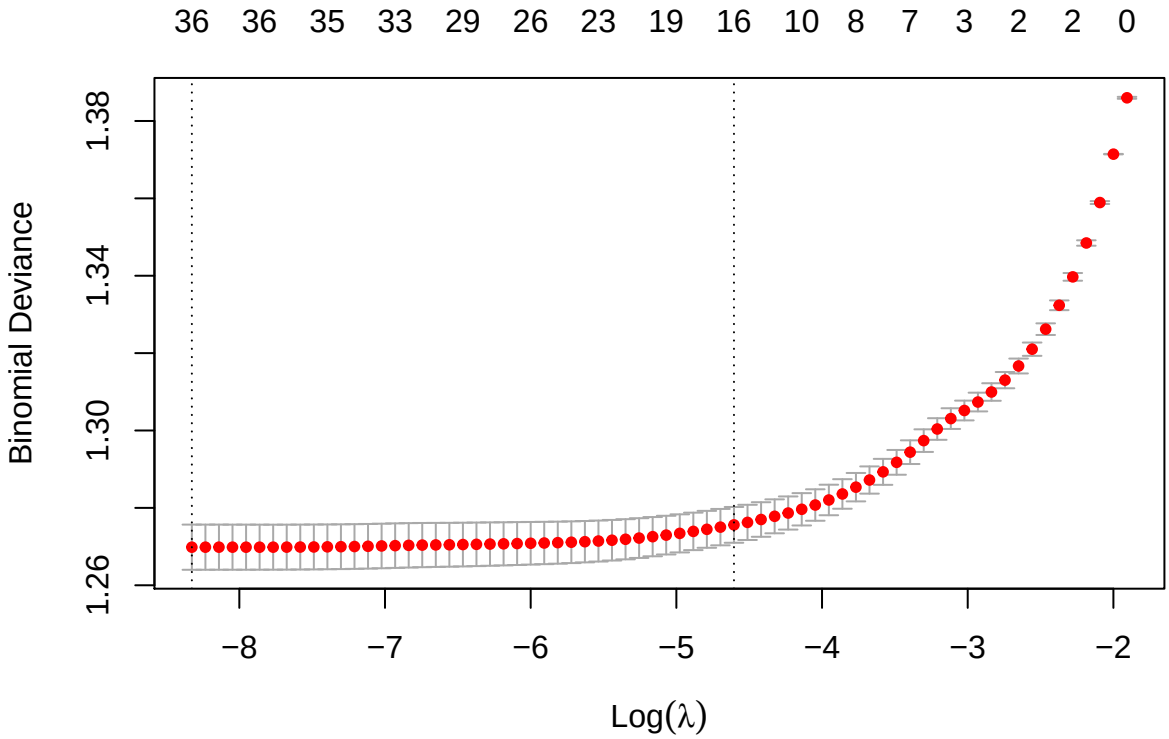
```
##              1
## [1,] 0.8375363
## [2,] 0.8403131
## [3,] 0.9027629
## [4,] 0.8353841
## [5,] 0.8039161
## [6,] 0.8867557
```

#auc values

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


#auc value for lambda.se
predsaucSE <- prediction(glmPredls_pSE, lcdfTrn_AR$loan_status, label.ordering = c("Charged Off", "Fully
aucPerfSE <- performance(predsaucSE, "auc")
```

#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")
#need auc@y.values. need to get that package
```

#standard error

```
#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.1s
##
##   Df %Dev  Lambda
## 1  9  6.3 0.00813
```

```
tidy(glmls_1)
```

```
## # A tibble: 10 x 5
##    term                 step     estimate  lambda dev.ratio
##    <chr>               <dbl>        <dbl>   <dbl>     <dbl>
##  1 (Intercept)             1  3.23        0.00813    0.0630
##  2 int_rate                1 -0.0267      0.00813    0.0630
##  3 grade                   1 -0.0836      0.00813    0.0630
##  4 sub_grade               1 -0.0477      0.00813    0.0630
```

```
##  5 home_ownership           1 -0.0348      0.00813    0.0630
##  6 dti                      1 -0.00813     0.00813    0.0630
##  7 acc_open_past_24mths     1 -0.0225      0.00813    0.0630
##  8 num_rev_tl_bal_gt_0      1 -0.0147      0.00813    0.0630
##  9 percent_bc_gt_75         1 -0.000649    0.00813    0.0630
## 10 tot_hi_cred_lim          1  0.000000602 0.00813    0.0630
```

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

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                     row column         value
## 1           (Intercept)      1  3.277036e+00
## 2              int_rate      1 -3.545125e-02
## 3                 grade      1 -8.297652e-02
## 4             sub_grade      1 -4.212040e-02
## 5        home_ownership      1 -3.485478e-02
## 6                   dti      1 -8.132383e-03
## 7  acc_open_past_24mths      1 -2.251697e-02
## 8   num_rev_tl_bal_gt_0      1 -1.473326e-02
## 9      percent_bc_gt_75      1 -6.466641e-04
## 10      tot_hi_cred_lim      1  6.010087e-07
```

#without regularization

```
#linear model without regularization
#get the variables with non-zero coefficients from the regularized model
nzCoef<-tidy(coef(glmls_cv, s= glmls_cv$lambda.1se))
nzCoefVars <- nzCoef[-1,1]

#the model without regularization
glmls_nzv_2 <- glm(yTrn ~ data.matrix(xDTrn %>% select(nzCoefVars)), family=binomial())

summary(glmls_nzv_2)
```

```
##
## Call:
## glm(formula = yTrn ~ data.matrix(xDTrn %>% select(nzCoefVars)),
##     family = binomial())
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7861   0.3452   0.4653   0.6009   1.6226
##
## Coefficients:
##                                                                 Estimate
## (Intercept)                                                    3.786e+00
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate             -5.288e-02
## data.matrix(xDTrn %>% select(nzCoefVars))grade                -1.169e-01
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade            -2.738e-02
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership       -7.235e-02
```

```
## data.matrix(xDTrn %>% select(nzCoefVars))dti                   -1.232e-02
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths -4.037e-02
## data.matrix(xDTrn %>% select(nzCoefVars))num_rev_tl_bal_gt_0  -2.886e-02
## data.matrix(xDTrn %>% select(nzCoefVars))percent_bc_gt_75     -2.472e-03
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim       1.134e-06
##                                                               Std. Error
## (Intercept)                                                    1.931e-01
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate              3.433e-02
## data.matrix(xDTrn %>% select(nzCoefVars))grade                 4.318e-02
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade             2.441e-02
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership        1.559e-02
## data.matrix(xDTrn %>% select(nzCoefVars))dti                   1.470e-03
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths  4.220e-03
## data.matrix(xDTrn %>% select(nzCoefVars))num_rev_tl_bal_gt_0   4.009e-03
## data.matrix(xDTrn %>% select(nzCoefVars))percent_bc_gt_75      3.690e-04
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim       1.102e-07
##                                                               z value Pr(>|z|)
## (Intercept)                                                    19.605  < 2e-16
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate              -1.540  0.12347
## data.matrix(xDTrn %>% select(nzCoefVars))grade                 -2.708  0.00677
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade             -1.122  0.26193
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership        -4.641 3.47e-06
## data.matrix(xDTrn %>% select(nzCoefVars))dti                   -8.380  < 2e-16
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths  -9.567  < 2e-16
## data.matrix(xDTrn %>% select(nzCoefVars))num_rev_tl_bal_gt_0   -7.197 6.15e-13
## data.matrix(xDTrn %>% select(nzCoefVars))percent_bc_gt_75      -6.699 2.10e-11
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim       10.290  < 2e-16
##
## (Intercept)                                                   ***
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate
## data.matrix(xDTrn %>% select(nzCoefVars))grade                **
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership       ***
## data.matrix(xDTrn %>% select(nzCoefVars))dti                  ***
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths ***
## data.matrix(xDTrn %>% select(nzCoefVars))num_rev_tl_bal_gt_0  ***
## data.matrix(xDTrn %>% select(nzCoefVars))percent_bc_gt_75     ***
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim      ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 46964  on 56713  degrees of freedom
## Residual deviance: 43848  on 56704  degrees of freedom
## AIC: 43868
##
## Number of Fisher Scoring iterations: 5
```

```
glm(formula = yTrn ~ data.matrix(xDTrn %>% select(nzCoefVars)),
    family = binomial())
```

```
##
## Call:  glm(formula = yTrn ~ data.matrix(xDTrn %>% select(nzCoefVars)),
##     family = binomial())
```

```
##
## Coefficients:
##                                                    (Intercept)
##                                                      3.786e+00
##             data.matrix(xDTrn %>% select(nzCoefVars))int_rate
##                                                     -5.288e-02
##                data.matrix(xDTrn %>% select(nzCoefVars))grade
##                                                     -1.169e-01
##            data.matrix(xDTrn %>% select(nzCoefVars))sub_grade
##                                                     -2.738e-02
##       data.matrix(xDTrn %>% select(nzCoefVars))home_ownership
##                                                     -7.235e-02
##                  data.matrix(xDTrn %>% select(nzCoefVars))dti
##                                                     -1.232e-02
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths
##                                                     -4.037e-02
##  data.matrix(xDTrn %>% select(nzCoefVars))num_rev_tl_bal_gt_0
##                                                     -2.886e-02
##     data.matrix(xDTrn %>% select(nzCoefVars))percent_bc_gt_75
##                                                     -2.472e-03
##      data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim
##                                                      1.134e-06
##
## Degrees of Freedom: 56713 Total (i.e. Null);  56704 Residual
## Null Deviance:       46960
## Residual Deviance: 43850     AIC: 43870
```

Question 2

#Building a Random Forest Model to Predict Actual Returns

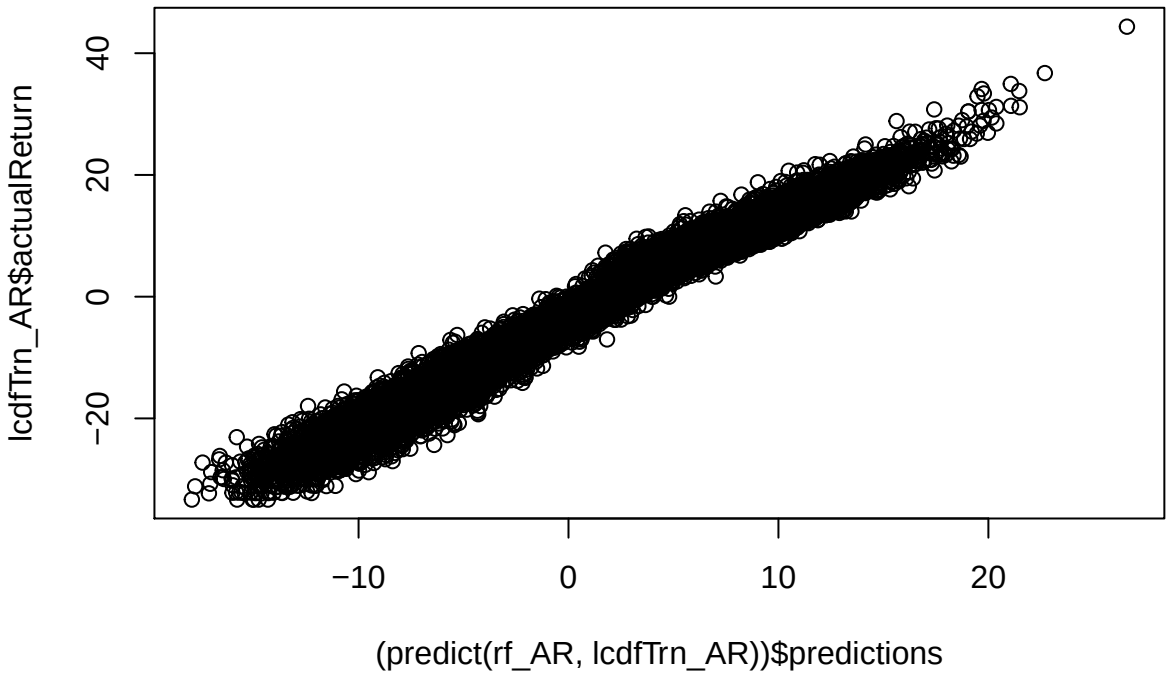
```
rf_AR <- ranger(actualReturn ~., data=subset(lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status)), nu
```

#RF – Training data

```
rfPredRet_trn_AR<- predict(rf_AR,  lcdfTrn_AR)
sqrt(mean((rfPredRet_trn_AR$predictions- lcdfTrn_AR$actualReturn)^2))
```

```
## [1] 3.752462
```

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



#RF – Performance by Deciles for Training Data

```
predRet_Trn_AR<- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta

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),    numDefaul
```

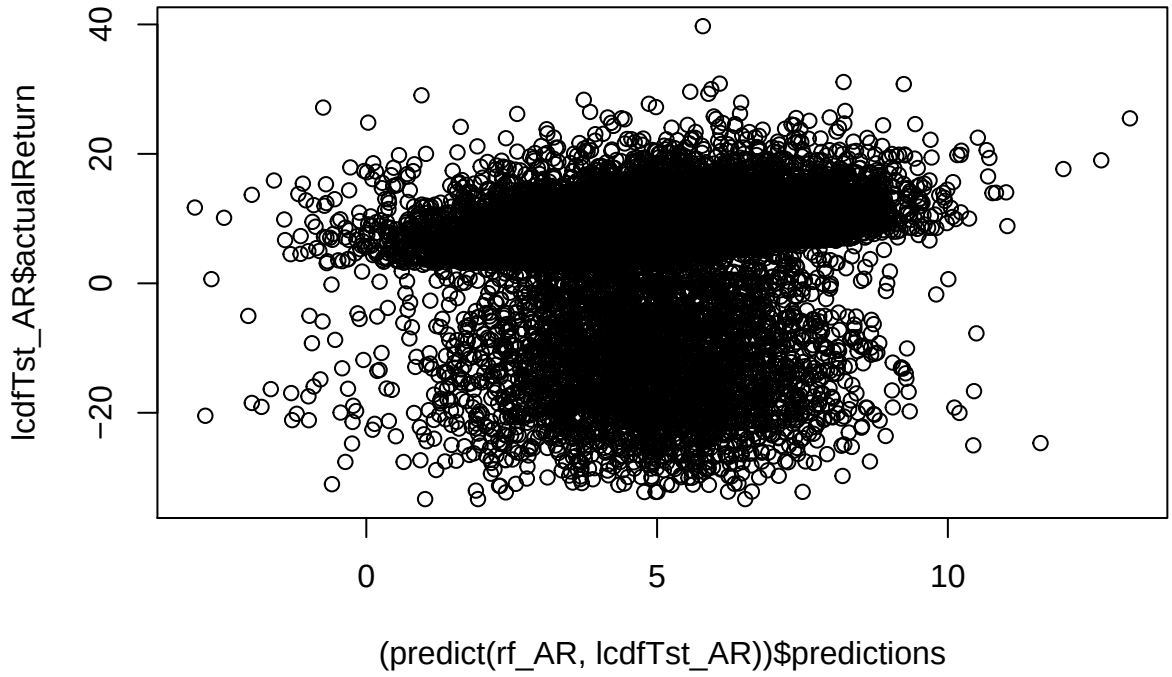
```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672      11.3            5     14.8    9.45  44.4    0.997     3
##  2     2  5672       8.82           5     11.0    7.39  18.8    1.59     11
##  3     3  5672       7.68          19      9.34   5.91  15.7    2.05     46
##  4     4  5672       6.83          39      8.16   3.32  14.0    2.24    255
##  5     5  5671       6.11          55      7.25   3.43  12.7    2.31    752
##  6     6  5671       5.43          94      6.42   2.50  13.4    2.27   1668
##  7     7  5671       4.72         136      5.38   0     10.7    2.36   3026
##  8     8  5671       4.04         165      4.44  -1.36   9.90   2.60   4526
##  9     9  5671       2.30        2038      1.46 -11.8    9.52   2.85   3575
## 10    10  5671      -7.02        5671    -16.6  -33.3   -2.86   3       516
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#RF – Test data

```
rfPredRet_Tst_AR<- predict(rf_AR,  lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_AR$predictions- lcdfTst_AR$actualReturn)^2))
```

```
## [1] 8.370825
```

```
plot ((predict(rf_AR,  lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
```



#RF – Performance by Deciles for Test Data

```
predRet_Tst_AR<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta

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),    numDefaul
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2431       7.67         412      7.41  -32.2   31.1    2.07     0
##  2     2  2431       6.47         389      6.43  -33.3   27.9    2.13     0
##  3     3  2431       5.84         399      5.80  -31.1   39.7    2.15    10
##  4     4  2431       5.34         380      5.41  -31.2   29.6    2.21   177
##  5     5  2431       4.92         346      5.08  -32.2   27.7    2.22   466
##  6     6  2431       4.53         351      4.56  -30.9   25.5    2.24   846
##  7     7  2431       4.16         282      4.74  -31.1   25.6    2.27  1098
##  8     8  2430       3.79         271      4.22  -30.9   28.4    2.27  1312
##  9     9  2430       3.31         319      4.03  -30.8   23.8    2.33  1244
## 10    10  2430       2.11         451      3.46  -33.3   29.0    2.42   871
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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
```

```
os_lcdfTrn_AR<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn_Bal), na.action= na.pass, method=

bs_lcdfTrn_AR<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn_Bal), na.action= na.pass, method=

bs_lcdfTrn_AR%>% group_by(loan_status) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   loan_status [2]
##   loan_status     n
##   <fct>       <int>
## 1 Fully Paid  28405
## 2 Charged Off 28310
```

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

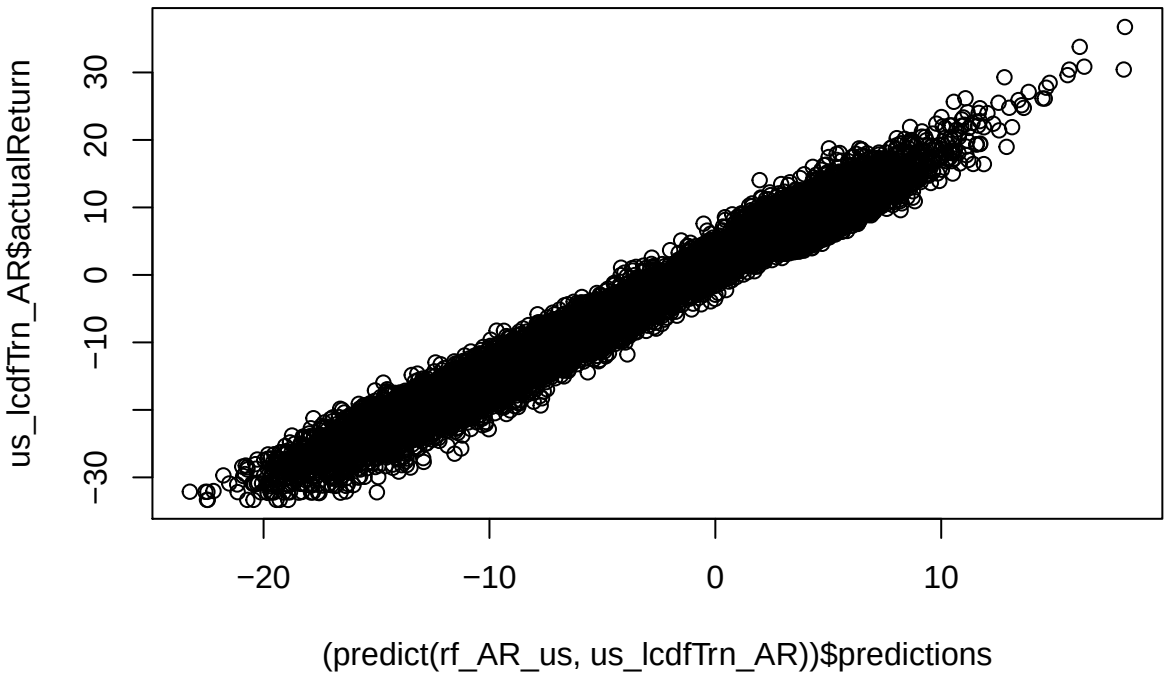
```
rf_AR_us <- ranger(actualReturn ~., data=subset(us_lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status
```

#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))
```

```
## [1] 5.008316
```

```
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) %>% mu

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),    numDe
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet  minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>   <dbl>  <dbl>   <dbl> <int>
```

```
##  1     1  1646      6.95            4    12.8     6.39   36.7     1.15    27
##  2     2  1645      4.77           12     8.83    3.97   18.8     1.88   228
##  3     3  1645      3.78           38     7.17    3.37   14.9     2.29   433
##  4     4  1645      3.01           75     6.12    2.50   13.7     2.57   628
##  5     5  1645      2.08          205     5.22    0.406  14.0     2.73   804
##  6     6  1645     -0.714        1378     0.712  -7.96    9.14    2.94   330
##  7     7  1645     -4.88         1644    -6.64  -14.4     1.37    3.00   205
##  8     8  1645     -8.29         1645   -12.4   -20.6    -5.14    3      175
##  9     9  1645    -11.4          1645   -17.5   -27.7    -9.59    3      131
## 10    10  1645    -15.4          1645   -23.5   -33.3   -14.6     3       57
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#RF Over Sampling

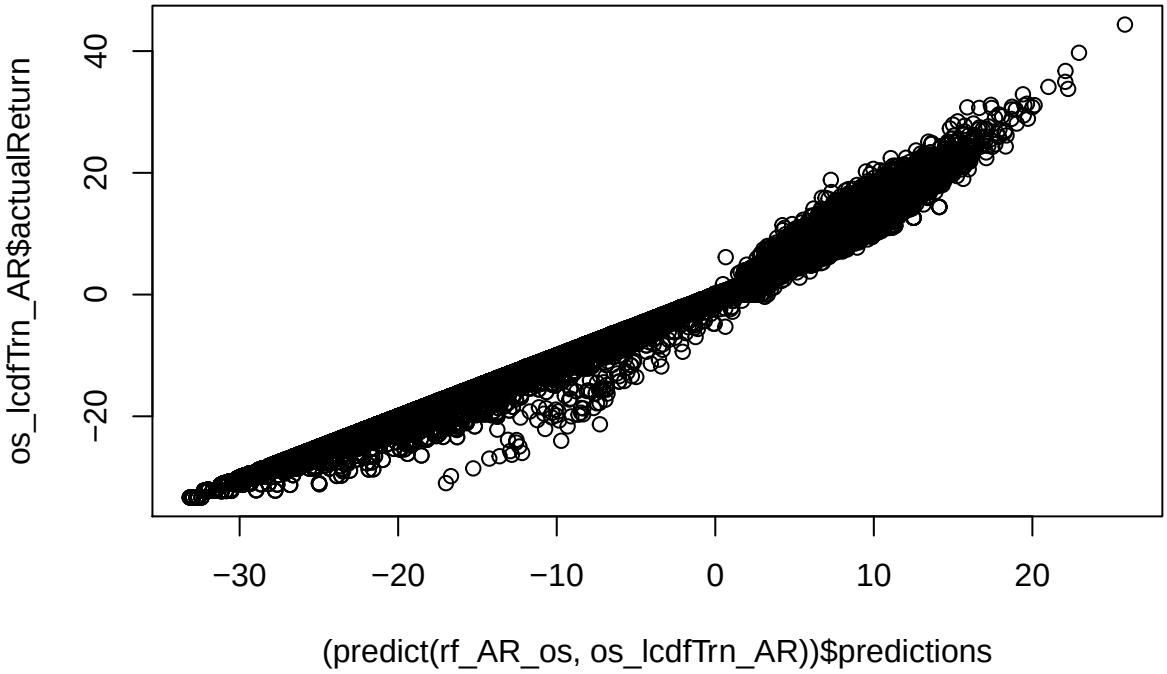
```
rf_AR_os <- ranger(actualReturn ~., data=subset(os_lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status
```

#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))
```

```
## [1] 1.692999
```

```
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) %>% mu

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),    numDe
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet  minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>   <dbl>  <dbl>   <dbl> <int>
```

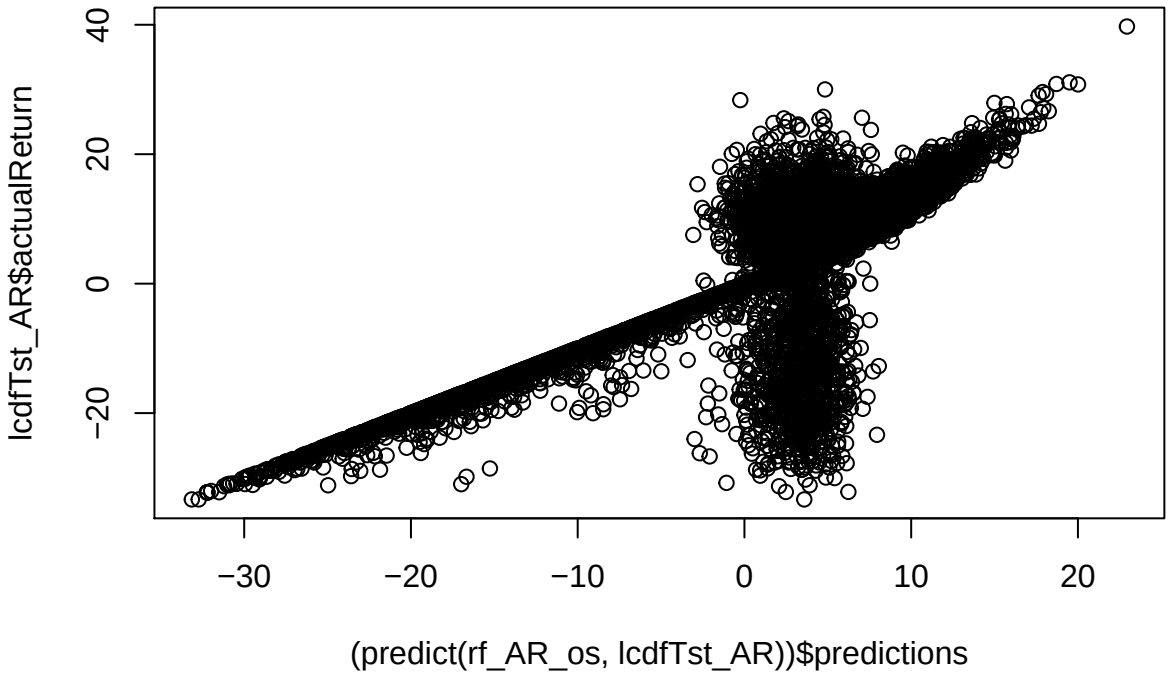
```
##  1     1  9692      9.78          198    13.2     7.42   44.4     1.24    12
##  2     2  9692      7.38          328     9.29    5.28   18.8     2.03   106
##  3     3  9692      6.18          426     7.52    3.79   15.9     2.34   910
##  4     4  9692      5.15          694     6.09    2.77   12.3     2.33  3135
##  5     5  9691      4.09          947     4.50    0.918  11.4     2.60  6993
##  6     6  9691      0.953        7133     0.948  -9.40    7.99    2.93  3166
##  7     7  9691     -5.99         9691    -6.31  -21.3    -2.69    3     1146
##  8     8  9691    -11.8          9691   -12.2   -26.9    -8.98    3      925
##  9     9  9691    -17.0          9691   -17.4   -31.0   -14.4     3      886
## 10    10  9691    -23.4          9691   -23.8   -33.3   -19.5     3      591
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#RF Over Sampling Test Results

```
rfPredRet_Tst_os<- predict(rf_AR_os,  lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_os$predictions- lcdfTst_AR$actualReturn)^2))
```

```
## [1] 5.029708
```

```
plot ((predict(rf_AR_os,  lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
```




```
predRet_Tst_os<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta

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),    numDe
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2431      10.0            7     13.7    6.47   39.7    1.12     4
##  2     2  2431       7.70          22      9.83 -23.3    23.8    1.88    13
##  3     3  2431       6.63          38      8.20 -32.2    25.6    2.23   101
```

```
##  4     4  2431       5.83          88      7.03 -27.7    22.4    2.29   295
##  5     5  2431       5.16         105      6.20 -30.0    22.3    2.23   625
##  6     6  2431       4.57         152      5.37 -27.6    30.0    2.28  1011
##  7     7  2431       4.05         179      4.58 -31.1    20.9    2.43  1409
##  8     8  2430       3.50         204      4.22 -33.3    24.6    2.48  1533
##  9     9  2430       2.30         520      4.00 -32.2    25.5    2.41   790
## 10    10  2430     -11.7         2285    -12.0  -33.3    28.4    2.95   243
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#Both #Building RF for Both (combination of over and under sampling)

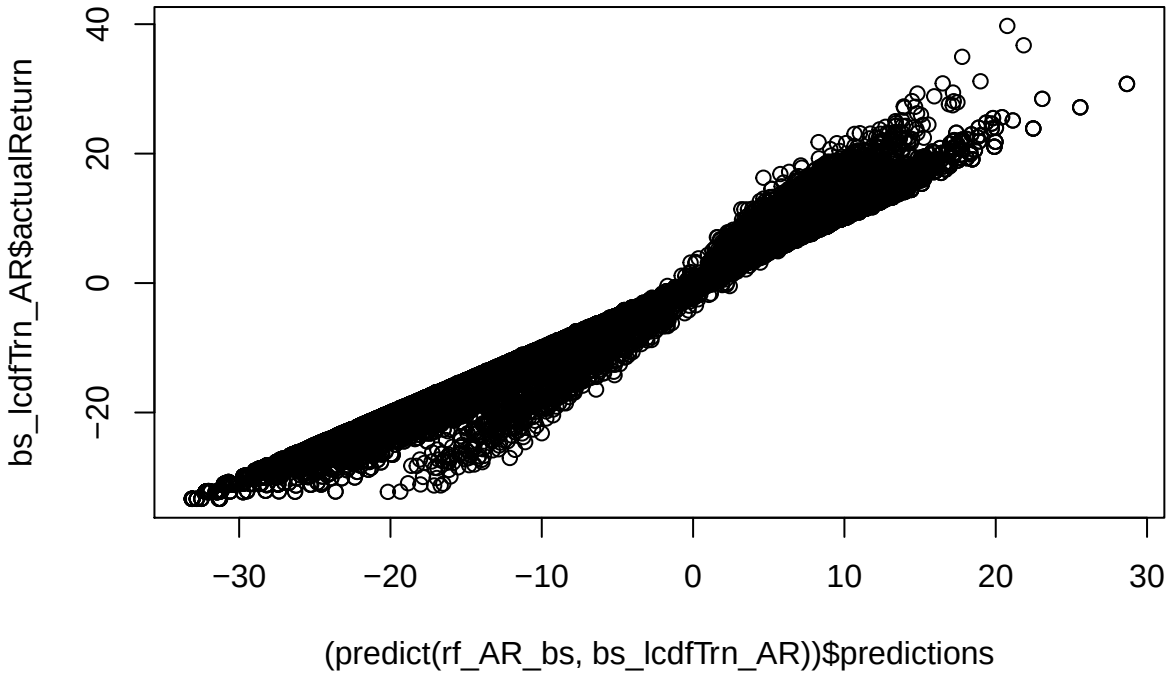
```
rf_AR_bs <- ranger(actualReturn ~., data=subset(bs_lcdfTrn_AR, select=-c(annRet, actualTerm, loan_status
```

#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))
```

```
## [1] 2.105647
```

```
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) %>% mu

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),    numDe
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672     10.3            83    12.9     8.07  39.7     1.27    21
##  2     2  5672      7.13          194     9.23    6.31  18.2     1.96   258
##  3     3  5672      5.78          308     7.52    4.92  17.2     2.26   851
```

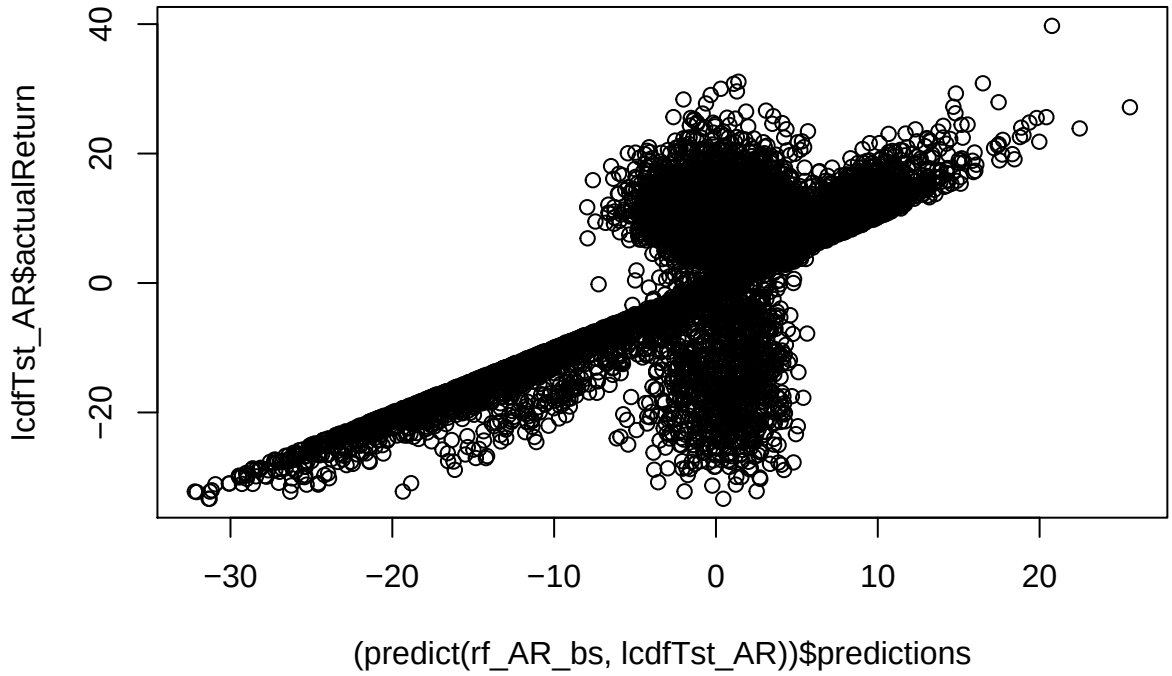
```
##  4     4  5672      4.73          436     6.15    3.16  16.3     2.38  1907
##  5     5  5672      3.73          526     4.66    2.11  11.5     2.68  3687
##  6     6  5671      0.717        4079     0.904  -7.40   8.59    2.91  1764
##  7     7  5671     -5.80         5671    -6.54  -18.6   -2.25    3      659
##  8     8  5671    -11.3          5671   -12.3   -27.0   -8.75    3      589
##  9     9  5671    -16.3          5671   -17.4   -31.3  -13.6     3      502
## 10    10  5671    -22.7          5671   -23.7   -33.3  -19.0     3      357
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#RF Both Test results

```
rfPredRet_Tst_bs<- predict(rf_AR_bs,  lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_bs$predictions- lcdfTst_AR$actualReturn)^2))
```

```
## [1] 7.054103
```

```
plot ((predict(rf_AR_bs,  lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
```



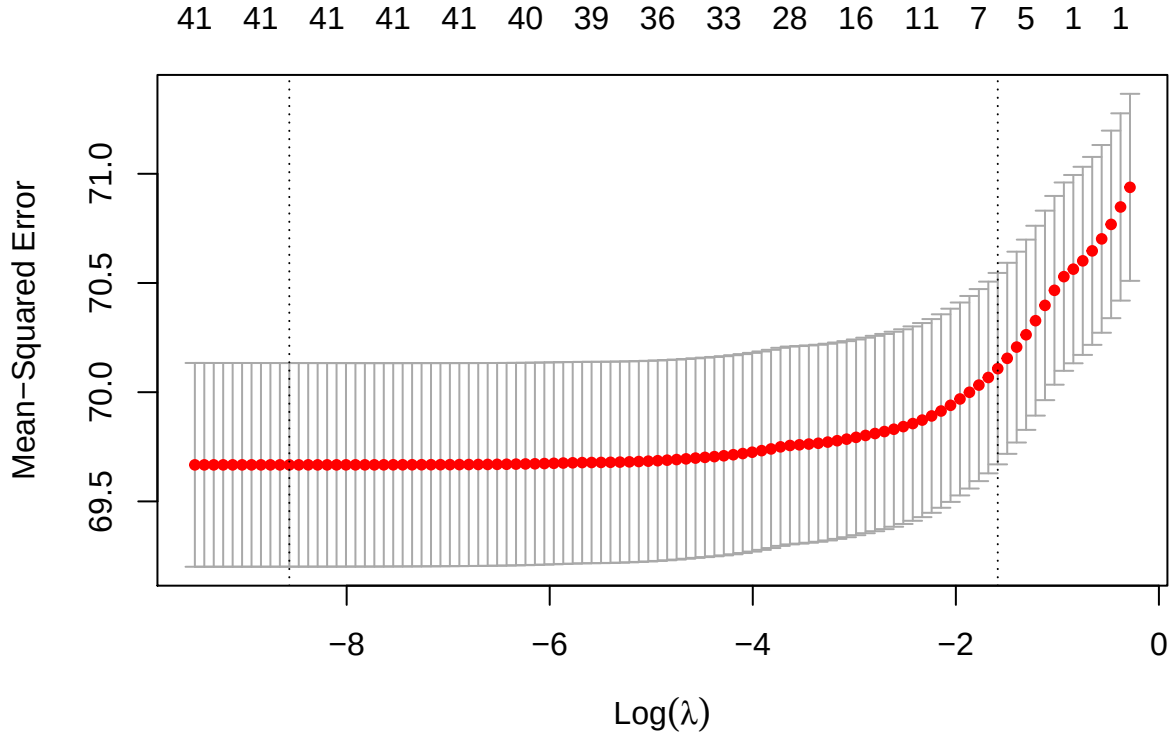

```
predRet_Tst_bs<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta

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),    numDe
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2431      8.26           32     11.2    6.15   39.7    1.56    56
##  2     2  2431      5.27           61      7.33 -27.7    23.4    2.24   463
##  3     3  2431      4.07           81      5.97 -28.9    24.7    2.30  1087
##  4     4  2431      3.36           87      5.77 -27.9    26.6    2.25  1226
##  5     5  2431      2.72          136      5.85 -32.2    21.7    2.22  1049
##  6     6  2431      2.06          178      6.24 -28.8    26.5    2.16   803
```

```
##  7     7  2431      1.32          213      6.36 -31.1    31.1    2.19   613
##  8     8  2430      0.395         313      6.43 -33.3    30.0    2.25   371
##  9     9  2430     -1.15          483      6.40 -32.2    29.0    2.29   164
## 10    10  2430    -11.3          2016    -10.4  -33.3    25.6    2.85   192
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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
```

```
## [1] 0.0001907208
```

```
glmRet_cv$lambda.1se
```

```
## [1] 0.2045036
```

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

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                   row column         value
```

```
## 1         (Intercept)      1  4.003159e+00
## 2            int_rate      1  1.615811e-01
## 3      home_ownership      1 -8.375471e-02
## 4                 dti      1 -2.341256e-02
## 5         avg_cur_bal      1  2.108409e-06
## 6      num_actv_bc_tl      1 -4.939848e-02
## 7 num_rev_tl_bal_gt_0      1 -1.181398e-02
## 8     tot_hi_cred_lim      1  3.503166e-07
```

```
coef(glmRet_cv, s="lambda.min")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                2.915360e+00
## loan_amnt                  2.272776e-05
## int_rate                   6.844757e-01
## installment               -1.583412e-03
## grade                     -5.425341e-01
## sub_grade                 -1.485621e-01
## emp_length                -4.675459e-02
## home_ownership            -2.741406e-01
## annual_inc                -3.102492e-07
## verification_status       -1.784089e-01
## purpose                    2.311312e-02
## dti                       -5.980435e-02
## earliest_cr_line           3.955694e-04
## initial_list_status       -4.008842e-02
## total_rev_hi_lim           7.892476e-06
## acc_open_past_24mths      -7.215334e-02
## avg_cur_bal               -1.073943e-06
## bc_open_to_buy            -3.534707e-05
## bc_util                   -3.671538e-03
## mo_sin_old_il_acct        -3.100169e-04
## mo_sin_old_rev_tl_op       5.578787e-04
## mo_sin_rcnt_rev_tl_op     -1.542609e-03
## mo_sin_rcnt_tl            -1.447704e-03
## mths_since_recent_bc      -4.615475e-04
## mths_since_recent_inq      3.982255e-03
## num_accts_ever_120_pd     -2.321585e-02
## num_actv_bc_tl            -1.022037e-01
## num_actv_rev_tl            1.045263e-01
## num_bc_sats                3.623803e-02
## num_bc_tl                 -3.797350e-02
## num_il_tl                  1.435028e-02
## num_op_rev_tl             -1.365540e-02
## num_rev_accts              4.022284e-02
## num_rev_tl_bal_gt_0       -1.513533e-01
## num_sats                  -1.965856e-02
## num_tl_op_past_12m         4.817557e-02
## pct_tl_nvr_dlq            -1.176149e-03
## percent_bc_gt_75          -5.188140e-03
## tot_hi_cred_lim            1.505965e-06
## total_bal_ex_mort         -9.002107e-06
## total_bc_limit             1.984131e-05
## total_il_high_credit_limit  1.209001e-05
```

#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) %>%

####May want to run predictions with lambda.1se and compare

#spliting 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),    numDefa
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672       7.33        1097      7.21  -33.3   44.4    2.10    60
##  2     2  5672       6.36         920      6.53  -33.3   30.8    2.10   157
##  3     3  5672       5.91         906      6.03  -33.3   31.4    2.16   392
##  4     4  5672       5.57         845      5.87  -31.0   25.0    2.17   685
##  5     5  5671       5.26         846      5.14  -32.2   24.8    2.20  1059
##  6     6  5671       4.97         817      4.87  -33.3   27.3    2.23  1413
##  7     7  5671       4.68         728      4.62  -31.2   22.7    2.28  1933
##  8     8  5671       4.37         717      4.26  -32.3   19.9    2.28  2317
##  9     9  5671       3.98         701      3.72  -32.2   21.0    2.33  2934
## 10    10  5671       3.25         650      3.43  -32.3   22.9    2.41  3428
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

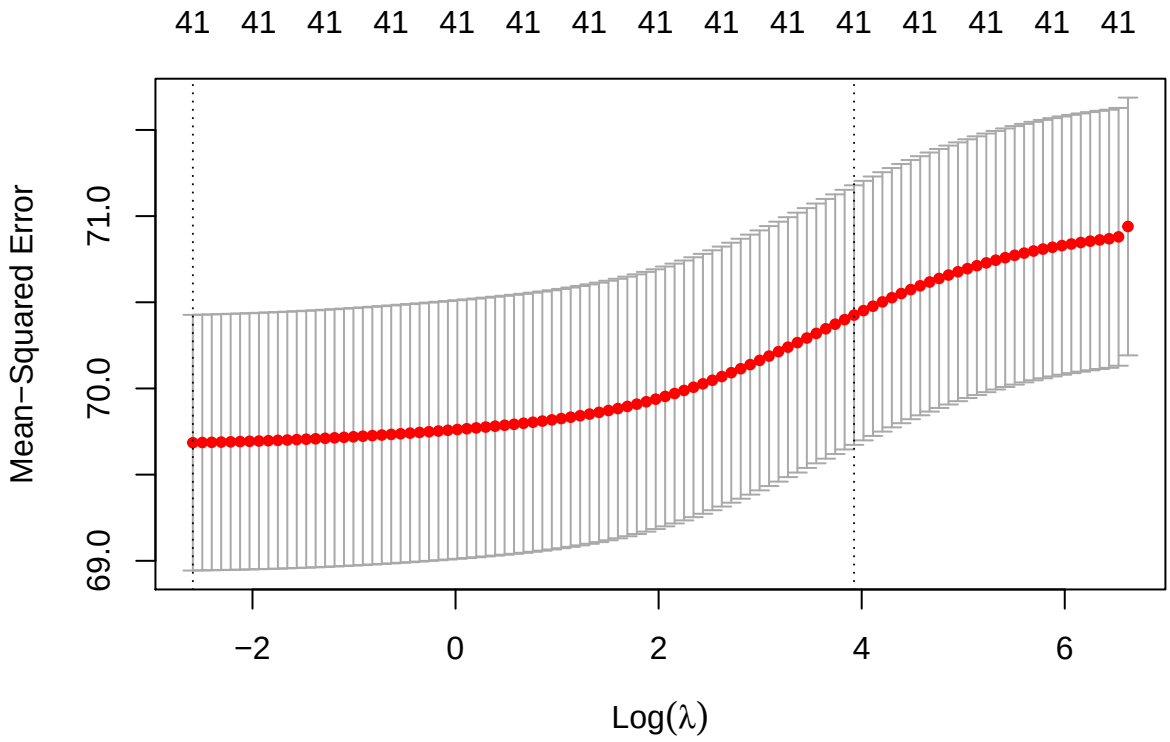
#Predictions of glm Model for Test Data

```
#Performance of glm model for lambda.min
predRet_Tst_AR_glm <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%

####May want to run predictions with lambda.1se and compare

#spliting 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),    numDefa
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2431       7.35         476      7.19  -32.2   31.1    2.10    33
##  2     2  2431       6.35         402      6.58  -32.2   28.4    2.10    75
##  3     3  2431       5.91         430      5.59  -33.3   30.9    2.18   146
##  4     4  2431       5.58         376      5.69  -32.2   25.3    2.16   284
##  5     5  2431       5.28         361      5.07  -31.2   39.7    2.24   439
##  6     6  2431       4.99         356      4.66  -31.1   23.8    2.26   591
##  7     7  2431       4.70         279      5.07  -28.7   26.2    2.25   827
##  8     8  2430       4.39         310      4.20  -32.2   17.7    2.29   991
##  9     9  2430       3.99         298      3.91  -30.2   23.0    2.31  1183
## 10    10  2430       3.27         312      3.17  -32.3   21.7    2.41  1455
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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
```

```
## [1] 50.65711
```

```
#min
glmRet_cv_a0$lambda.min
```

```
## [1] 0.0752242
```

```
coef(glmRet_cv_a0, s="lambda.1se")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                5.347271e+00
## loan_amnt                 -2.026873e-06
## int_rate                   2.227185e-02
## installment               -3.499554e-05
## grade                      6.144129e-02
## sub_grade                  1.406563e-02
## emp_length                -7.401085e-03
## home_ownership            -3.911690e-02
## annual_inc                 3.606064e-07
## verification_status       -1.444097e-02
## purpose                    1.991915e-02
## dti                       -5.267693e-03
## earliest_cr_line           5.524346e-05
## initial_list_status       -2.221584e-02
```

```
## total_rev_hi_lim           -1.744525e-07
## acc_open_past_24mths        8.075483e-04
## avg_cur_bal                 2.159092e-06
## bc_open_to_buy             -1.184233e-06
## bc_util                    -3.677205e-04
## mo_sin_old_il_acct         -3.765717e-06
## mo_sin_old_rev_tl_op       -1.832558e-05
## mo_sin_rcnt_rev_tl_op      -6.951602e-04
## mo_sin_rcnt_tl             -1.586810e-03
## mths_since_recent_bc       -7.689132e-07
## mths_since_recent_inq      -8.729770e-04
## num_accts_ever_120_pd       5.413930e-03
## num_actv_bc_tl             -1.693521e-02
## num_actv_rev_tl            -9.160758e-03
## num_bc_sats                -9.296279e-03
## num_bc_tl                  -3.050633e-03
## num_il_tl                   7.457646e-04
## num_op_rev_tl              -4.824649e-03
## num_rev_accts              -4.517338e-04
## num_rev_tl_bal_gt_0        -1.010265e-02
## num_sats                   -3.131688e-03
## num_tl_op_past_12m          1.157120e-02
## pct_tl_nvr_dlq             -2.321679e-03
## percent_bc_gt_75           -3.094522e-04
## tot_hi_cred_lim             1.861700e-07
## total_bal_ex_mort           1.265222e-07
## total_bc_limit             -8.006020e-07
## total_il_high_credit_limit  1.598687e-07
```

```
coef(glmRet_cv_a0, s="lambda.min")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                4.620596e+00
## loan_amnt                 -4.608868e-06
## int_rate                   3.624230e-01
## installment               -7.382300e-04
## grade                     -4.588488e-01
## sub_grade                  3.955867e-02
## emp_length                -4.607556e-02
## home_ownership            -2.728075e-01
## annual_inc                 1.255134e-09
## verification_status       -1.807049e-01
## purpose                    2.466083e-02
## dti                       -5.726318e-02
## earliest_cr_line           3.960348e-04
## initial_list_status       -3.429721e-02
## total_rev_hi_lim           6.219061e-06
## acc_open_past_24mths      -6.973469e-02
## avg_cur_bal               -1.210600e-06
## bc_open_to_buy            -3.037043e-05
## bc_util                   -3.403210e-03
## mo_sin_old_il_acct        -2.998103e-04
## mo_sin_old_rev_tl_op       5.650212e-04
## mo_sin_rcnt_rev_tl_op     -1.702153e-03
```

```
## mo_sin_rcnt_tl             -1.479016e-03
## mths_since_recent_bc       -4.766423e-04
## mths_since_recent_inq       3.709269e-03
## num_accts_ever_120_pd      -2.729368e-02
## num_actv_bc_tl             -9.583090e-02
## num_actv_rev_tl             5.429455e-02
## num_bc_sats                 3.368203e-02
## num_bc_tl                  -3.538024e-02
## num_il_tl                   1.381021e-02
## num_op_rev_tl              -1.046255e-02
## num_rev_accts               3.813871e-02
## num_rev_tl_bal_gt_0        -1.040919e-01
## num_sats                   -1.966481e-02
## num_tl_op_past_12m          4.632336e-02
## pct_tl_nvr_dlq             -1.872413e-03
## percent_bc_gt_75           -4.993007e-03
## tot_hi_cred_lim             1.500699e-06
## total_bal_ex_mort          -6.745801e-06
## total_bc_limit              1.695373e-05
## total_il_high_credit_limit  9.696218e-06
```

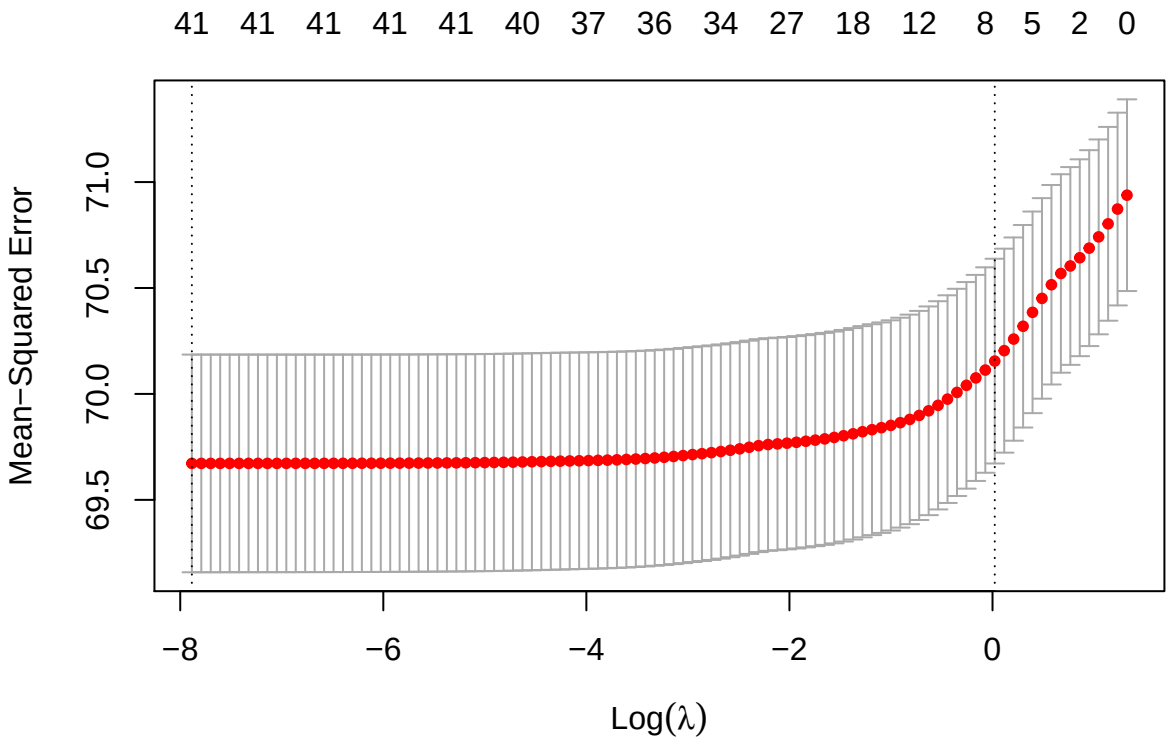
#Predictions for alpha=0 Model for Training Data

```
#Performance of glm model for lambda min
predRet_Trn_AR_a0 <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% r

####May want to run predictions with lambda.1se and compare


#spliting 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),    numDefaul
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672       7.33        1097      7.21  -33.3   44.4    2.10    60
##  2     2  5672       6.36         920      6.53  -33.3   30.8    2.10   157
##  3     3  5672       5.91         906      6.03  -33.3   31.4    2.16   392
##  4     4  5672       5.57         845      5.87  -31.0   25.0    2.17   685
##  5     5  5671       5.26         846      5.14  -32.2   24.8    2.20  1059
##  6     6  5671       4.97         817      4.87  -33.3   27.3    2.23  1413
##  7     7  5671       4.68         728      4.62  -31.2   22.7    2.28  1933
##  8     8  5671       4.37         717      4.26  -32.3   19.9    2.28  2317
##  9     9  5671       3.98         701      3.72  -32.2   21.0    2.33  2934
## 10    10  5671       3.25         650      3.43  -32.3   22.9    2.41  3428
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#alpha = x for glm model

```
glmRet_cv_a2<- cv.glmnet(data.matrix(df4_glm),   lcdfTrn_AR$actualReturn, family="gaussian",    alpha=0

plot(glmRet_cv_a2)
```




```
#1se
glmRet_cv_a2$lambda.1se
```

```
## [1] 1.022518
```

```
#min
glmRet_cv_a2$lambda.min
```

```
## [1] 0.000376121
```

```
coef(glmRet_cv_a2, s="lambda.1se")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                4.314429e+00
## loan_amnt                  .
## int_rate                   1.019532e-01
## installment                .
## grade                      .
## sub_grade                  3.153415e-02
## emp_length                 .
## home_ownership            -7.407729e-02
## annual_inc                 .
## verification_status        .
## purpose                    .
## dti                       -2.029078e-02
## earliest_cr_line           .
## initial_list_status        .
## total_rev_hi_lim           .
## acc_open_past_24mths       .
## avg_cur_bal                2.824329e-06
## bc_open_to_buy             .
```

```
## bc_util                      .
## mo_sin_old_il_acct           .
## mo_sin_old_rev_tl_op         .
## mo_sin_rcnt_rev_tl_op        .
## mo_sin_rcnt_tl               .
## mths_since_recent_bc         .
## mths_since_recent_inq        .
## num_accts_ever_120_pd        .
## num_actv_bc_tl              -4.486843e-02
## num_actv_rev_tl              .
## num_bc_sats                  .
## num_bc_tl                    .
## num_il_tl                    .
## num_op_rev_tl                .
## num_rev_accts                .
## num_rev_tl_bal_gt_0         -1.314398e-02
## num_sats                     .
## num_tl_op_past_12m           .
## pct_tl_nvr_dlq               .
## percent_bc_gt_75             .
## tot_hi_cred_lim              2.428879e-07
## total_bal_ex_mort            .
## total_bc_limit               .
## total_il_high_credit_limit   .
```

```
coef(glmRet_cv_a2, s="lambda.min")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                2.877661e+00
## loan_amnt                  3.107277e-05
## int_rate                   6.888089e-01
## installment               -1.829235e-03
## grade                     -5.242163e-01
## sub_grade                 -1.542728e-01
## emp_length                -4.682192e-02
## home_ownership            -2.743410e-01
## annual_inc                -3.350809e-07
## verification_status       -1.785844e-01
## purpose                    2.331336e-02
## dti                       -5.990578e-02
## earliest_cr_line           3.955944e-04
## initial_list_status       -4.011880e-02
## total_rev_hi_lim           8.031314e-06
## acc_open_past_24mths      -7.239405e-02
## avg_cur_bal               -1.084186e-06
## bc_open_to_buy            -3.553705e-05
## bc_util                   -3.687582e-03
## mo_sin_old_il_acct        -3.106517e-04
## mo_sin_old_rev_tl_op       5.564035e-04
## mo_sin_rcnt_rev_tl_op     -1.539437e-03
## mo_sin_rcnt_tl            -1.449437e-03
## mths_since_recent_bc      -4.628947e-04
## mths_since_recent_inq      3.991208e-03
## num_accts_ever_120_pd     -2.298864e-02
```

```
## num_actv_bc_tl             -1.034780e-01
## num_actv_rev_tl             1.159707e-01
## num_bc_sats                 3.661916e-02
## num_bc_tl                  -3.763655e-02
## num_il_tl                   1.446496e-02
## num_op_rev_tl              -1.314313e-02
## num_rev_accts               3.998321e-02
## num_rev_tl_bal_gt_0        -1.623106e-01
## num_sats                   -2.009981e-02
## num_tl_op_past_12m          4.873078e-02
## pct_tl_nvr_dlq             -1.171617e-03
## percent_bc_gt_75           -5.164141e-03
## tot_hi_cred_lim             1.504621e-06
## total_bal_ex_mort          -9.133730e-06
## total_bc_limit              1.980839e-05
## total_il_high_credit_limit  1.223500e-05
```

#Predictions for alpha=2 Model for Training Data

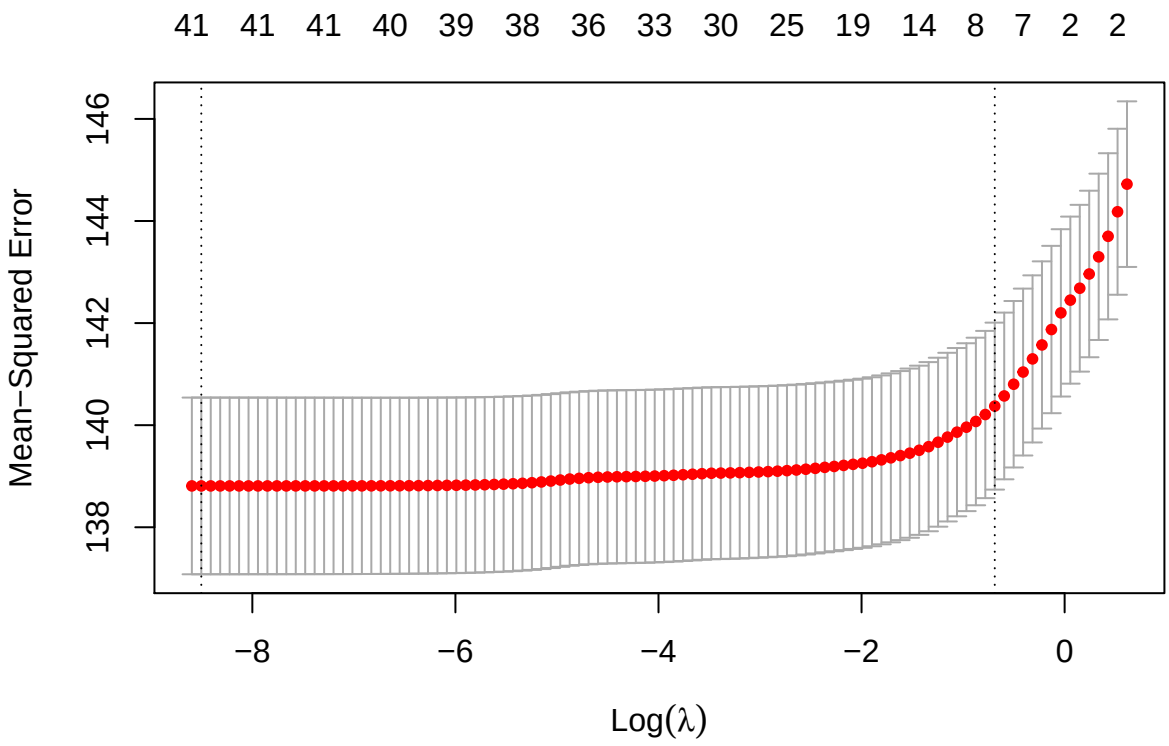
```
#Performance of glm model for lambda min
predRet_Trn_AR_a2 <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% r

####May want to run predictions with lambda.1se and compare

#spliting 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),    numDefaul
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672       7.33        1097      7.21  -33.3   44.4    2.10    60
##  2     2  5672       6.36         920      6.53  -33.3   30.8    2.10   157
##  3     3  5672       5.91         906      6.03  -33.3   31.4    2.16   392
##  4     4  5672       5.57         845      5.87  -31.0   25.0    2.17   685
##  5     5  5671       5.26         846      5.14  -32.2   24.8    2.20  1059
##  6     6  5671       4.97         817      4.87  -33.3   27.3    2.23  1413
##  7     7  5671       4.68         728      4.62  -31.2   22.7    2.28  1933
##  8     8  5671       4.37         717      4.26  -32.3   19.9    2.28  2317
##  9     9  5671       3.98         701      3.72  -32.2   21.0    2.33  2934
## 10    10  5671       3.25         650      3.43  -32.3   22.9    2.41  3428
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

##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
```

```
## [1] 0.0002028862
```

```
glmRet_cv_us$lambda.1se
```

```
## [1] 0.5025645
```

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

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                     row column         value
## 1           (Intercept)      1  1.756434e+00
## 2                 grade      1 -4.937687e-01
## 3             sub_grade      1 -8.817004e-02
## 4       home_ownership      1 -7.690732e-02
## 5  verification_status      1 -1.669717e-01
## 6                   dti      1 -4.235990e-02
## 7 acc_open_past_24mths      1 -7.293647e-02
## 8           avg_cur_bal      1  8.905874e-06
## 9      tot_hi_cred_lim      1  1.544018e-06
```

```
coef(glmRet_cv_us, s="lambda.min")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                4.251374e-01
```

```
## loan_amnt                   7.035579e-04
## int_rate                    1.006919e+00
## installment                -2.301049e-02
## grade                      -6.522767e-01
## sub_grade                  -6.137106e-01
## emp_length                 -5.919707e-02
## home_ownership             -4.474751e-01
## annual_inc                  3.612285e-07
## verification_status        -4.349977e-01
## purpose                     3.696513e-02
## dti                        -9.912444e-02
## earliest_cr_line            1.256506e-03
## initial_list_status         7.182176e-02
## total_rev_hi_lim            2.818272e-05
## acc_open_past_24mths       -1.616336e-01
## avg_cur_bal                 7.049416e-06
## bc_open_to_buy             -4.660811e-05
## bc_util                     8.904183e-03
## mo_sin_old_il_acct          7.119662e-04
## mo_sin_old_rev_tl_op        1.457886e-03
## mo_sin_rcnt_rev_tl_op       1.498826e-02
## mo_sin_rcnt_tl             -2.548140e-03
## mths_since_recent_bc       -1.748643e-03
## mths_since_recent_inq       1.312268e-02
## num_accts_ever_120_pd      -5.181044e-02
## num_actv_bc_tl             -2.779512e-01
## num_actv_rev_tl             4.714783e-01
## num_bc_sats                 4.772895e-03
## num_bc_tl                  -1.912987e-02
## num_il_tl                   9.918647e-03
## num_op_rev_tl               6.178322e-03
## num_rev_accts              -3.433970e-03
## num_rev_tl_bal_gt_0        -3.088907e-01
## num_sats                   -2.319460e-02
## num_tl_op_past_12m         -1.025040e-01
## pct_tl_nvr_dlq             -2.486594e-02
## percent_bc_gt_75           -1.894951e-02
## tot_hi_cred_lim             3.363225e-06
## total_bal_ex_mort          -2.946409e-05
## total_bc_limit              2.057380e-05
## total_il_high_credit_limit  3.459497e-05
```

#Predictions for glm under sampling

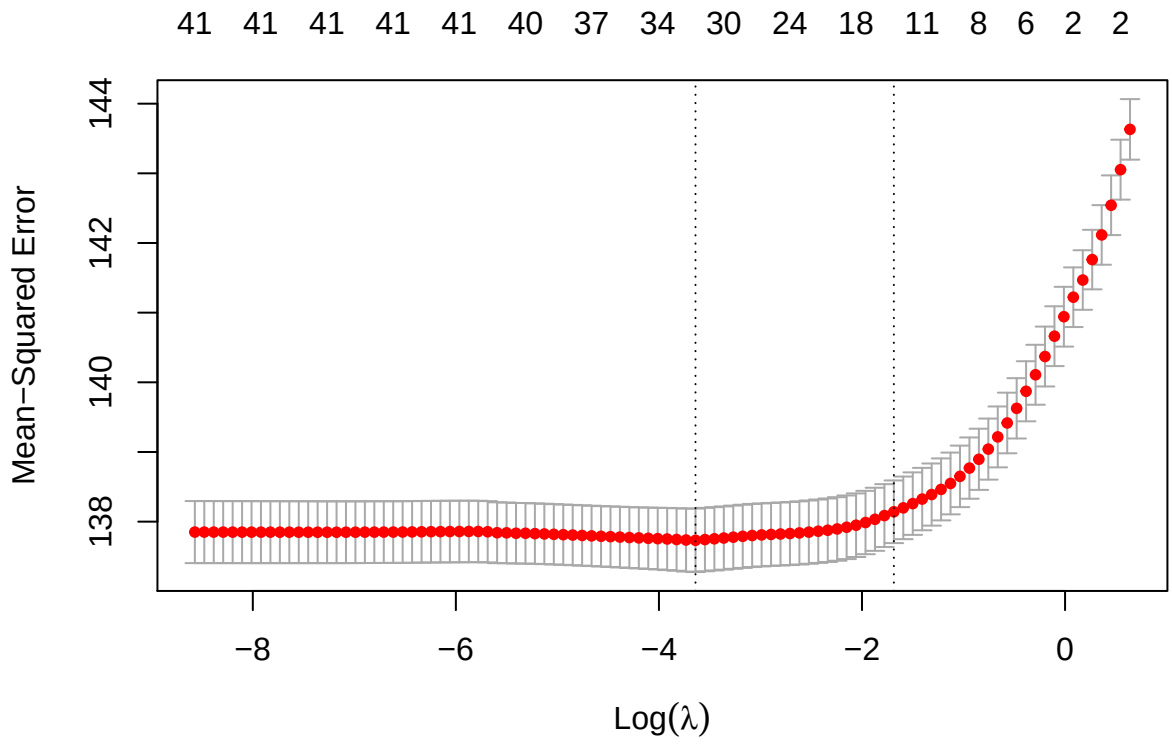
```
#Performance of glm model for lambda min
predRet_Trn_us_glm <- us_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %

#spliting 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),    numl
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
```

```
##  1     1  1646      2.53          372     1.87   -33.3   30.9    2.35  1059
##  2     2  1645      0.733         521     0.950  -32.3   19.4    2.37   742
##  3     3  1645     -0.172         592     0.601  -31.3   23.3    2.42   511
##  4     4  1645     -0.907         742    -1.07   -33.3   28.5    2.53   336
##  5     5  1645     -1.56          799    -1.22   -33.3   22.2    2.51   203
##  6     6  1645     -2.19          893    -2.00   -32.3   30.4    2.59   115
##  7     7  1645     -2.87          925    -2.32   -32.2   24.0    2.60    35
##  8     8  1645     -3.66         1061    -4.25   -33.3   24.7    2.68    14
##  9     9  1645     -4.62         1144    -5.00   -31.1   30.4    2.72     2
## 10    10  1645     -6.46         1242    -6.74   -33.3   36.7    2.79     1
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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
```

```
## [1] 0.02625397
```

```
glmRet_cv_os$lambda.1se
```

```
## [1] 0.1852165
```

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

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                       row column         value
## 1           (Intercept)      1  4.487589e+00
## 2           installment      1 -3.167530e-04
## 3                 grade      1 -5.938138e-01
## 4             sub_grade      1 -9.164084e-02
## 5            emp_length      1 -1.326221e-02
## 6        home_ownership      1 -3.806716e-01
## 7   verification_status      1 -3.434070e-01
## 8                   dti      1 -7.018060e-02
## 9  acc_open_past_24mths      1 -1.366643e-01
## 10          avg_cur_bal      1  8.709156e-06
## 11 mo_sin_old_rev_tl_op      1  2.083957e-04
## 12       num_actv_bc_tl      1 -3.512874e-02
## 13          num_bc_sats      1 -7.777194e-04
## 14     percent_bc_gt_75      1 -5.680250e-03
## 15      tot_hi_cred_lim      1  2.117258e-06
```

```
coef(glmRet_cv_os, s="lambda.min")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                6.456766e+00
## loan_amnt                  .
## int_rate                   .
## installment               -1.417098e-03
## grade                     -6.839616e-01
## sub_grade                 -6.268098e-02
## emp_length                -5.711728e-02
## home_ownership            -5.235484e-01
## annual_inc                 5.917655e-07
## verification_status       -3.779846e-01
## purpose                    2.368450e-02
## dti                       -9.014291e-02
## earliest_cr_line           9.497962e-04
## initial_list_status        9.799235e-02
## total_rev_hi_lim           .
## acc_open_past_24mths      -1.751218e-01
## avg_cur_bal                6.579614e-06
## bc_open_to_buy             .
## bc_util                    2.878751e-03
## mo_sin_old_il_acct         2.548556e-04
## mo_sin_old_rev_tl_op       2.060932e-03
## mo_sin_rcnt_rev_tl_op      2.135507e-03
## mo_sin_rcnt_tl             .
## mths_since_recent_bc      -4.460670e-04
## mths_since_recent_inq      6.582279e-03
## num_accts_ever_120_pd     -6.145925e-02
## num_actv_bc_tl            -3.622656e-02
## num_actv_rev_tl            4.135284e-02
## num_bc_sats               -8.924868e-02
```

```
## num_bc_tl                  -3.058744e-02
## num_il_tl                  -1.157553e-04
## num_op_rev_tl               .
## num_rev_accts               7.562344e-03
## num_rev_tl_bal_gt_0         .
## num_sats                    .
## num_tl_op_past_12m          .
## pct_tl_nvr_dlq             -1.432340e-02
## percent_bc_gt_75           -1.298496e-02
## tot_hi_cred_lim             2.358945e-06
## total_bal_ex_mort           .
## total_bc_limit              1.037437e-05
## total_il_high_credit_limit  5.894719e-06
```

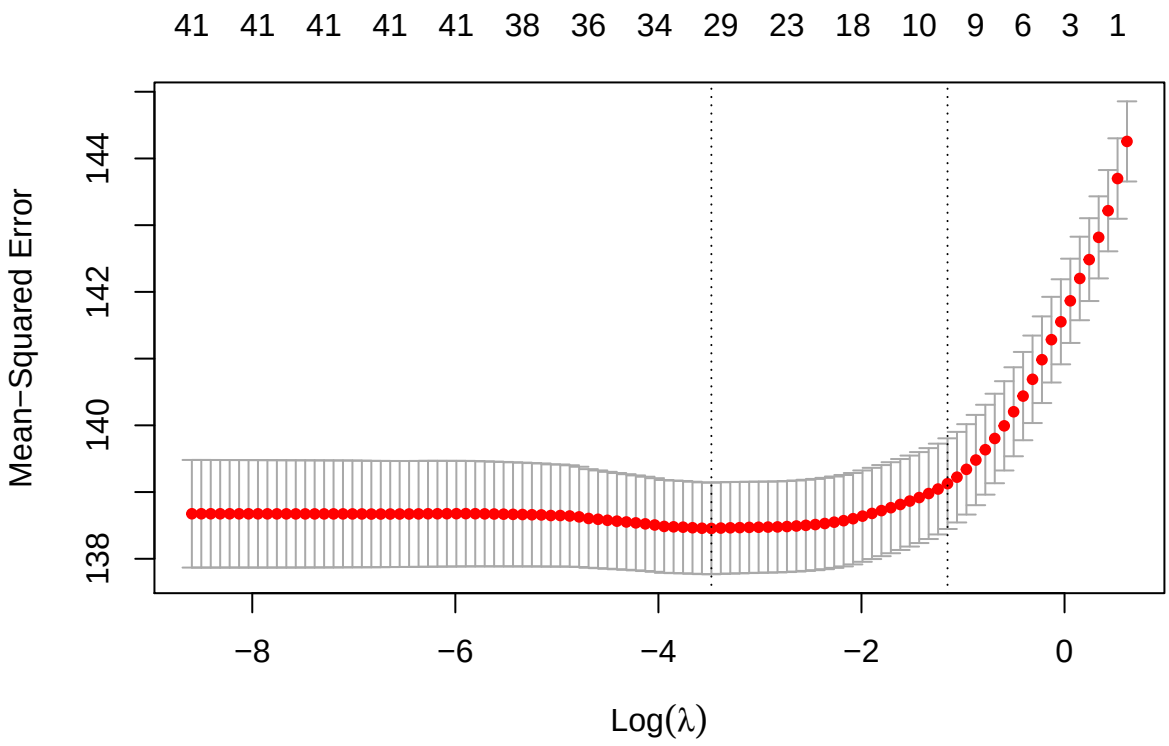
#Predictions for glm over sampling

```
#Performance of glm model for lambda min
predRet_Trn_os_glm <- os_lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %

#spliting 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),    numl
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  9692      2.31         1989     2.05   -32.3   30.9    2.29  7377
##  2     2  9692      0.739        2944     0.956  -33.3   23.4    2.40  4824
##  3     3  9692     -0.139        3610     0.157  -31.3   23.9    2.45  3013
##  4     4  9692     -0.863        4409    -0.865  -33.3   24.4    2.51  1552
##  5     5  9691     -1.50         4760    -1.07   -31.2   24.3    2.53   759
##  6     6  9691     -2.10         5245    -2.06   -33.3   28.5    2.58   241
##  7     7  9691     -2.74         5566    -2.42   -32.3   30.8    2.60    83
##  8     8  9691     -3.46         6037    -3.48   -33.3   34.1    2.65    17
##  9     9  9691     -4.33         6625    -5.02   -32.2   39.7    2.71     1
## 10    10  9691     -5.99         7305    -6.32   -33.3   44.4    2.77     3
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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
```

```
## [1] 0.03086273
```

```
glmRet_cv_bs$lambda.1se
```

```
## [1] 0.3158896
```

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

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                     row column         value
## 1           (Intercept)      1  3.383051e+00
## 2                 grade      1 -3.817168e-01
## 3             sub_grade      1 -1.180329e-01
## 4        home_ownership      1 -2.624640e-01
## 5   verification_status      1 -2.461224e-01
## 6                   dti      1 -6.706766e-02
## 7  acc_open_past_24mths      1 -1.409954e-01
## 8           avg_cur_bal      1  9.214818e-06
## 9      percent_bc_gt_75      1 -3.559439e-03
## 10      tot_hi_cred_lim      1  1.312566e-06
```

```
coef(glmRet_cv_bs, s="lambda.min")
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
```

```
## (Intercept)                 5.463369e+00
## loan_amnt                   .
## int_rate                    .
## installment                -8.287858e-04
## grade                      -5.350717e-01
## sub_grade                  -8.339062e-02
## emp_length                 -4.980715e-02
## home_ownership             -5.066250e-01
## annual_inc                  3.186334e-07
## verification_status        -4.224710e-01
## purpose                     5.947365e-02
## dti                        -9.522205e-02
## earliest_cr_line            7.803539e-04
## initial_list_status         3.509539e-02
## total_rev_hi_lim            .
## acc_open_past_24mths       -2.072998e-01
## avg_cur_bal                 3.606831e-06
## bc_open_to_buy              .
## bc_util                     5.039211e-03
## mo_sin_old_il_acct          .
## mo_sin_old_rev_tl_op        2.373902e-03
## mo_sin_rcnt_rev_tl_op       6.403237e-04
## mo_sin_rcnt_tl              .
## mths_since_recent_bc       -1.213659e-03
## mths_since_recent_inq       9.000303e-03
## num_accts_ever_120_pd      -3.634043e-02
## num_actv_bc_tl             -2.348944e-02
## num_actv_rev_tl             3.433046e-02
## num_bc_sats                -1.023880e-01
## num_bc_tl                  -1.968260e-02
## num_il_tl                  -1.456465e-02
## num_op_rev_tl               .
## num_rev_accts               .
## num_rev_tl_bal_gt_0         .
## num_sats                    .
## num_tl_op_past_12m          .
## pct_tl_nvr_dlq             -3.492982e-03
## percent_bc_gt_75           -1.555782e-02
## tot_hi_cred_lim             2.265405e-06
## total_bal_ex_mort           .
## total_bc_limit              7.005881e-06
## total_il_high_credit_limit  8.367344e-06
```

#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) %

#spliting 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),    numD
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
```

```
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672      2.20         1152    2.08    -33.3   16.5    2.29  4240
##  2     2  5672      0.662        1674    1.09    -32.1   30.9    2.39  2883
##  3     3  5672     -0.196        2126    0.0734  -32.2   23.9    2.46  1743
##  4     4  5672     -0.896        2567   -0.651   -31.3   24.4    2.51   917
##  5     5  5672     -1.53         2881   -1.87    -33.3   24.3    2.56   494
##  6     6  5671     -2.13         3010   -1.91    -33.3   28.5    2.57   196
##  7     7  5671     -2.78         3203   -2.58    -32.3   28.1    2.58    90
##  8     8  5671     -3.51         3550   -3.45    -33.3   30.8    2.63    19
##  9     9  5671     -4.38         3862   -5.08    -32.2   39.7    2.70     8
## 10    10  5671     -6.07         4285   -6.33    -33.3   36.7    2.76     5
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

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

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
```

#XGBOOST Finding which hyper-parameters work best – experiment with a grid of parameter values

```
#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:9.627043 eval-rmse:9.610718
## 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:9.619272 eval-rmse:9.602957
## [3]  train-rmse:9.611520 eval-rmse:9.595201
## [4]  train-rmse:9.603771 eval-rmse:9.587458
## [5]  train-rmse:9.596032 eval-rmse:9.579720
## [6]  train-rmse:9.588303 eval-rmse:9.571995
## [7]  train-rmse:9.580580 eval-rmse:9.564279
## [8]  train-rmse:9.572877 eval-rmse:9.556571
## [9]  train-rmse:9.565176 eval-rmse:9.548869
## [10] train-rmse:9.562856 eval-rmse:9.546564
## [11] train-rmse:9.555510 eval-rmse:9.539234
## [12] train-rmse:9.547930 eval-rmse:9.531649
## [13] train-rmse:9.540357 eval-rmse:9.524071
## [14] train-rmse:9.538060 eval-rmse:9.521781
## [15] train-rmse:9.530407 eval-rmse:9.514127
## [16] train-rmse:9.522751 eval-rmse:9.506479
## [17] train-rmse:9.515114 eval-rmse:9.498840
## [18] train-rmse:9.512826 eval-rmse:9.496570
## [19] train-rmse:9.505294 eval-rmse:9.489032
## [20] train-rmse:9.497966 eval-rmse:9.481710
## [21] train-rmse:9.490359 eval-rmse:9.474100
## [22] train-rmse:9.482754 eval-rmse:9.466499
## [23] train-rmse:9.475162 eval-rmse:9.458907
## [24] train-rmse:9.467572 eval-rmse:9.451325
## [25] train-rmse:9.460097 eval-rmse:9.443842
## [26] train-rmse:9.452638 eval-rmse:9.436366
## [27] train-rmse:9.445074 eval-rmse:9.428810
## [28] train-rmse:9.437518 eval-rmse:9.421265
## [29] train-rmse:9.430270 eval-rmse:9.414020
## [30] train-rmse:9.422840 eval-rmse:9.406580
## [31] train-rmse:9.420592 eval-rmse:9.404361
## [32] train-rmse:9.413074 eval-rmse:9.396844
## [33] train-rmse:9.405563 eval-rmse:9.389338
## [34] train-rmse:9.398065 eval-rmse:9.381838
## [35] train-rmse:9.390667 eval-rmse:9.374433
## [36] train-rmse:9.388427 eval-rmse:9.372211
## [37] train-rmse:9.380939 eval-rmse:9.364732
## [38] train-rmse:9.373466 eval-rmse:9.357270
## [39] train-rmse:9.371287 eval-rmse:9.355104
```

```
## [40] train-rmse:9.369054 eval-rmse:9.352894
## [41] train-rmse:9.361699 eval-rmse:9.345526
## [42] train-rmse:9.354534 eval-rmse:9.338371
## [43] train-rmse:9.347082 eval-rmse:9.330934
## [44] train-rmse:9.339761 eval-rmse:9.323589
## [45] train-rmse:9.332336 eval-rmse:9.316180
## [46] train-rmse:9.324917 eval-rmse:9.308771
## [47] train-rmse:9.317603 eval-rmse:9.301455
...
```

#Plugin the tune parameters

```
xgbParamGrid2
```

```
##   max_depth   eta bestTree bestPerf
## 1         2 0.001     1000 5.527935
## 2         5 0.001     1000 5.382647
## 3         2 0.010     1000 4.006492
## 4         5 0.010      625 3.991658
## 5         2 0.100      130 4.004863
## 6         5 0.100       63 3.993993
```

```
# max_depth eta bestTree bestPerf
# 2 0.001   1000    5.534117
# 5 0.001   1000    5.395167
# 2 0.010   1000    4.006708
# 5 0.010   536 3.990832
# 2 0.100   154 4.004697
# 5 0.100   66  3.996803    <- almost the lower error this is better than 3.990832 because it use less

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_lsM2 <- xgb.train(xgbParam3, dxTrn2, nrounds = xgb_tune$best_iteration)
```

#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)
```

```
## [1] 8.805379 8.425661 4.817416 8.810807 9.789893 7.610485
```

```
#Using the predicting function to get the scores in the test data set
xpredTst2<-predict(xgb_lsM2, dxTst2)

#Error
sqrt(mean((xpredTst2- colcdfTst2)^2))
```

```
## [1] 3.997903
```

```
#Performance by deciles
scoreTst_xgb_ls2 <- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mu

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_sta
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm
totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=
```

```
## # A tibble: 10 x 15
##     tile count  avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>  <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431  12.8            0     13.0    0      31.1   1.93     0     0
##  2     2  2431  10.5            0     10.6    0      39.7   2.02     0     0
##  3     3  2431   9.45           0      9.48   2.81   24.4   2.01     0    28
##  4     4  2431   8.63           0      8.64   0      20.0   2.09     0  1070
##  5     5  2431   7.76           0      7.80   0      30.9   2.13     0  2363
##  6     6  2431   6.62           0      6.53   0      17.0   2.15    39  2391
##  7     7  2431   5.51           0      5.50   1.48   14.0   2.14  1953   478
##  8     8  2430   4.69           0      4.65   1.98   12.2   2.20  2428     2
##  9     9  2430  -2.96        1170     -3.01 -33.3    15.9   2.63  1401   405
## 10    10  2430 -12.2         2430    -12.1  -33.3    10.1   3      203   390
## # ... with 5 more variables: totC <int>, totD <int>, totE <int>, totF <int>,
## #   totG <int>
```

#3 #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))
#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") )
```

```
## # A tibble: 10 x 14
##     tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431 0.974          73      4.38  -30.3   17.0   2.18  2196   234
##  2     2  2431 0.942         170      4.43  -30.3   30.9   2.19  1583   804
##  3     3  2431 0.915         218      4.88  -30.0   23.4   2.19   988  1195
##  4     4  2431 0.889         235      5.53  -32.2   24.6   2.18   586  1272
##  5     5  2431 0.863         291      5.79  -33.3   24.0   2.18   317  1162
##  6     6  2431 0.836         363      5.51  -32.2   25.8   2.21   183   964
##  7     7  2431 0.809         410      5.67  -32.3   39.7   2.25    87   709
##  8     8  2430 0.777         491      5.46  -32.2   26.2   2.26    52   463
##  9     9  2430 0.736         571      5.16  -32.2   30.8   2.29    24   251
## 10    10  2430 0.653         778      4.31  -33.3   31.1   2.37     8    73
```

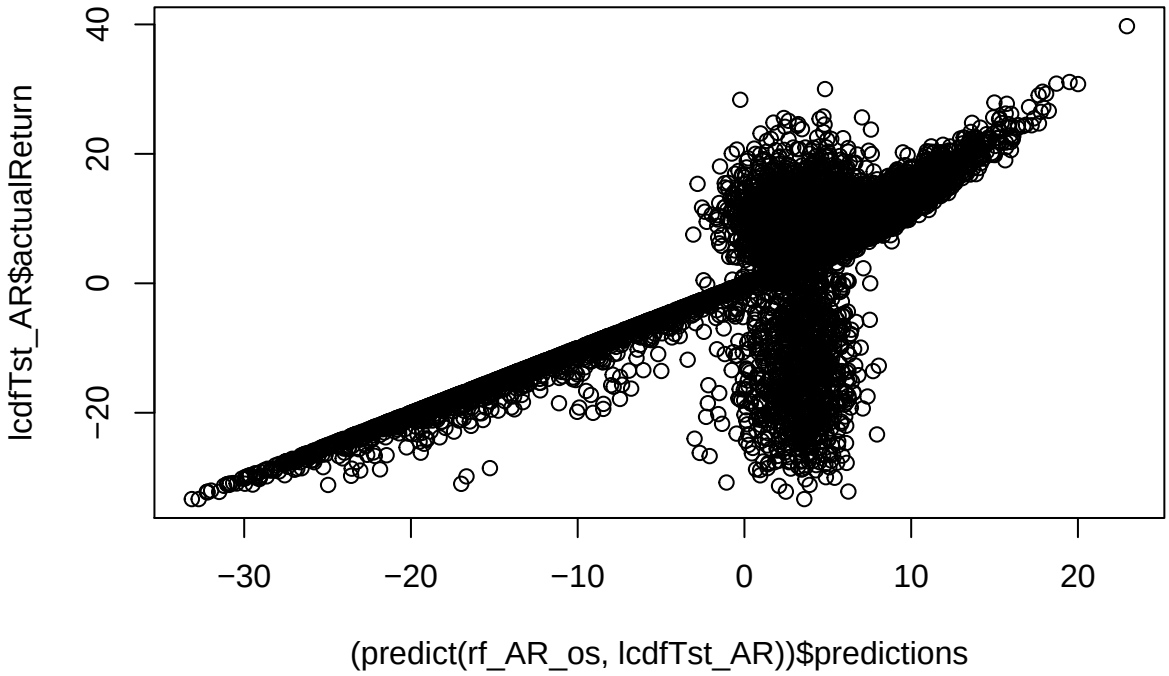
```
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

#Rf Returns selected by grade M2 - all grades

```
#Choose the Actual Returns Random forest we want for here
rfPredRet_Tst_ag<- predict(rf_AR_os,  lcdfTst_AR)
sqrt(mean((rfPredRet_Tst_ag$predictions- lcdfTst_AR$actualReturn)^2))
```

```
## [1] 5.029708
```

```
plot ((predict(rf_AR_os,  lcdfTst_AR))$predictions, lcdfTst_AR$actualReturn)
```



#Rf – Performance by Deciles for Test Data - all grades

```
ag_predRet_Tst<- lcdfTst_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta

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),    numDefau
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2431      10.0            7     13.7    6.47   39.7    1.12     4
##  2     2  2431       7.70          22      9.83 -23.3    23.8    1.88    13
##  3     3  2431       6.63          38      8.20 -32.2    25.6    2.23   101
##  4     4  2431       5.83          88      7.03 -27.7    22.4    2.29   295
##  5     5  2431       5.16         105      6.20 -30.0    22.3    2.23   625
##  6     6  2431       4.57         152      5.37 -27.6    30.0    2.28  1011
##  7     7  2431       4.05         179      4.58 -31.1    20.9    2.43  1409
##  8     8  2430       3.50         204      4.22 -33.3    24.6    2.48  1533
##  9     9  2430       2.30         520      4.00 -32.2    25.5    2.41   790
## 10    10  2430     -11.7         2285    -12.0  -33.3    28.4    2.95   243
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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") )
```

```
## # A tibble: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   122       9.48           0      12.2   8.26   30.9  0.713     3    70
##  2     2   122       9.78           0      12.6   9.05   24.6  0.863     1    33
##  3     3   122       9.77           0      12.4   7.43   21.7  0.961     0    25
##  4     4   122       9.71           0      12.5   8.82   22.2  1.02      0    17
##  5     5   122       9.51           0      12.2   6.47   24.0  1.11      0    11
##  6     6   122       9.76           0      12.7   9.15   21.2  1.07      0    11
##  7     7   122       9.91           0      13.1   8.79   21.3  0.949     0    13
##  8     8   122      10.0            1      13.6   9.13   39.7  1.06      0     5
##  9     9   122       9.81           0      13.1   9.02   20.7  1.11      0     7
## 10    10   122       9.97           0      13.5   8.59   22.4  1.06      0     6
## 11    11   122       9.81           1      13.1   8.69   22.5  1.24      0     6
## 12    12   121      10.1            0      14.0   9.20   26.2  1.05      0     3
## 13    13   121      10.1            0      13.7   9.25   30.8  1.32      0     1
## 14    14   121      10.3            1      14.3   9.25   24.5  1.07      0     2
## 15    15   121       9.99           1      13.9   8.98   22.5  1.29      0     0
## 16    16   121      10.3            1      14.5   9.76   25.5  1.24      0     0
## 17    17   121      10.3            0      15.0   9.24   31.1  1.23      0     1
## 18    18   121      10.8            1      15.7   8.96   29.3  1.21      0     0
## 19    19   121      10.7            0      15.8  10.1    27.7  1.43      0     0
## 20    20   121      10.8            1      16.7   8.75   29.0  1.41      0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

##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
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)
```

```
## [1] "Charged Off" "Fully Paid"
```

```
#"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
```

```
## [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_byt

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]  train-error:0.145061    train-auc:0.647772  eval-error:0.148105 eval-auc:0.649417
## 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:0.145061    train-auc:0.662740  eval-error:0.148105 eval-auc:0.662860
## [3]  train-error:0.145061    train-auc:0.660282  eval-error:0.148105 eval-auc:0.660339
## [4]  train-error:0.145061    train-auc:0.668248  eval-error:0.148105 eval-auc:0.668491
## [5]  train-error:0.145061    train-auc:0.668405  eval-error:0.148105 eval-auc:0.668809
## [6]  train-error:0.145061    train-auc:0.669267  eval-error:0.148105 eval-auc:0.669722
## [7]  train-error:0.145061    train-auc:0.669294  eval-error:0.148105 eval-auc:0.669818
## [8]  train-error:0.145061    train-auc:0.669312  eval-error:0.148105 eval-auc:0.669827
## [9]  train-error:0.145061    train-auc:0.669320  eval-error:0.148105 eval-auc:0.669831
## [10] train-error:0.145061    train-auc:0.669322  eval-error:0.148105 eval-auc:0.669831
## [11] train-error:0.145061    train-auc:0.669322  eval-error:0.148105 eval-auc:0.669831
## [12] train-error:0.145061    train-auc:0.669322  eval-error:0.148105 eval-auc:0.669831
## [13] train-error:0.145061    train-auc:0.669322  eval-error:0.148105 eval-auc:0.669831
## [14] train-error:0.145061    train-auc:0.669236  eval-error:0.148105 eval-auc:0.669877
## [15] train-error:0.145061    train-auc:0.670753  eval-error:0.148105 eval-auc:0.671571
## [16] train-error:0.145061    train-auc:0.671327  eval-error:0.148105 eval-auc:0.671742
## [17] train-error:0.145061    train-auc:0.671327  eval-error:0.148105 eval-auc:0.671742
## [18] train-error:0.145061    train-auc:0.671796  eval-error:0.148105 eval-auc:0.672020
## [19] train-error:0.145061    train-auc:0.672815  eval-error:0.148105 eval-auc:0.672932
## [20] train-error:0.145061    train-auc:0.672964  eval-error:0.148105 eval-auc:0.672850
## [21] train-error:0.145061    train-auc:0.673041  eval-error:0.148105 eval-auc:0.672899
## [22] train-error:0.145061    train-auc:0.673069  eval-error:0.148105 eval-auc:0.673002
## [23] train-error:0.145061    train-auc:0.673070  eval-error:0.148105 eval-auc:0.673002
```

```
## [24] train-error:0.145061    train-auc:0.673070  eval-error:0.148105 eval-auc:0.673002
## [25] train-error:0.145061    train-auc:0.673070  eval-error:0.148105 eval-auc:0.673002
## [26] train-error:0.145061    train-auc:0.673070  eval-error:0.148105 eval-auc:0.673002
## [27] train-error:0.145061    train-auc:0.673071  eval-error:0.148105 eval-auc:0.672996
## [28] train-error:0.145061    train-auc:0.673071  eval-error:0.148105 eval-auc:0.672996
## [29] train-error:0.145061    train-auc:0.673071  eval-error:0.148105 eval-auc:0.672997
## [30] train-error:0.145061    train-auc:0.673071  eval-error:0.148105 eval-auc:0.672997
## [31] train-error:0.145061    train-auc:0.673071  eval-error:0.148105 eval-auc:0.672997
## [32] train-error:0.145061    train-auc:0.673062  eval-error:0.148105 eval-auc:0.672999
## Stopping. Best iteration:
## [22] train-error:0.145061    train-auc:0.673069  eval-error:0.148105 eval-auc:0.673002
##
## [1]  train-error:0.145026    train-auc:0.678447  eval-error:0.148352 eval-auc:0.670429
## 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:0.145061    train-auc:0.688491  eval-error:0.148105 eval-auc:0.681110
## [3]  train-error:0.145061    train-auc:0.690068  eval-error:0.148105 eval-auc:0.680143
## [4]  train-error:0.145061    train-auc:0.692707  eval-error:0.148105 eval-auc:0.682682
## [5]  train-error:0.145061    train-auc:0.694914  eval-error:0.148105 eval-auc:0.684117
## [6]  train-error:0.145061    train-auc:0.694760  eval-error:0.148105 eval-auc:0.684112
## [7]  train-error:0.145061    train-auc:0.697721  eval-error:0.148105 eval-auc:0.686720
## [8]  train-error:0.145061    train-auc:0.699750  eval-error:0.148105 eval-auc:0.688512
## [9]  train-error:0.145061    train-auc:0.699466  eval-error:0.148105 eval-auc:0.687739
## [10] train-error:0.145061    train-auc:0.699009  eval-error:0.148105 eval-auc:0.687100
## [11] train-error:0.145061    train-auc:0.700316  eval-error:0.148105 eval-auc:0.687948
## [12] train-error:0.145061    train-auc:0.700083  eval-error:0.148105 eval-auc:0.687375
## [13] train-error:0.145061    train-auc:0.699982  eval-error:0.148105 eval-auc:0.687463
## [14] train-error:0.145061    train-auc:0.700452  eval-error:0.148105 eval-auc:0.687852
## [15] train-error:0.145061    train-auc:0.699977  eval-error:0.148105 eval-auc:0.687261
## [16] train-error:0.145061    train-auc:0.700609  eval-error:0.148105 eval-auc:0.687874
## [17] train-error:0.145061    train-auc:0.701182  eval-error:0.148105 eval-auc:0.688918
## [18] train-error:0.145061    train-auc:0.701411  eval-error:0.148105 eval-auc:0.689272
## [19] train-error:0.145061    train-auc:0.701122  eval-error:0.148105 eval-auc:0.688772
## [20] train-error:0.145061    train-auc:0.700722  eval-error:0.148105 eval-auc:0.688355
## [21] train-error:0.145061    train-auc:0.700965  eval-error:0.148105 eval-auc:0.688867
## [22] train-error:0.145061    train-auc:0.700894  eval-error:0.148105 eval-auc:0.688632
## [23] train-error:0.145061    train-auc:0.700518  eval-error:0.148105 eval-auc:0.688354
## [24] train-error:0.145061    train-auc:0.700247  eval-error:0.148105 eval-auc:0.688172
## [25] train-error:0.145061    train-auc:0.700396  eval-error:0.148105 eval-auc:0.688035
## [26] train-error:0.145061    train-auc:0.700575  eval-error:0.148105 eval-auc:0.688367
## [27] train-error:0.145061    train-auc:0.700392  eval-error:0.148105 eval-auc:0.688465
## [28] train-error:0.145061    train-auc:0.700316  eval-error:0.148105 eval-auc:0.688205
## Stopping. Best iteration:
...
```

```
xgbParamGrid1
```

```
##   max_depth   eta bestTree bestPerf
## 1         2 0.001       22 0.673002
## 2         5 0.001       18 0.689272
## 3         2 0.010       23 0.681698
## 4         5 0.010      533 0.700684
## 5         2 0.100      124 0.699873
## 6         5 0.100       67 0.700586
```

```
# max_depth eta bestTree bestPerf
# 2 0.001   7   0.677577
# 5 0.001   26  0.688194
# 2 0.010   58  0.682594
# 5 0.010   66  0.691801
# 2 0.100   124 0.699748
# 5 0.100   62  0.699370

xgbParam_Best1 <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1, colsample

# 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)
```

```
## [1] 0.5311583 0.5311792 0.5318380 0.5306311 0.5306578 0.5313295
```

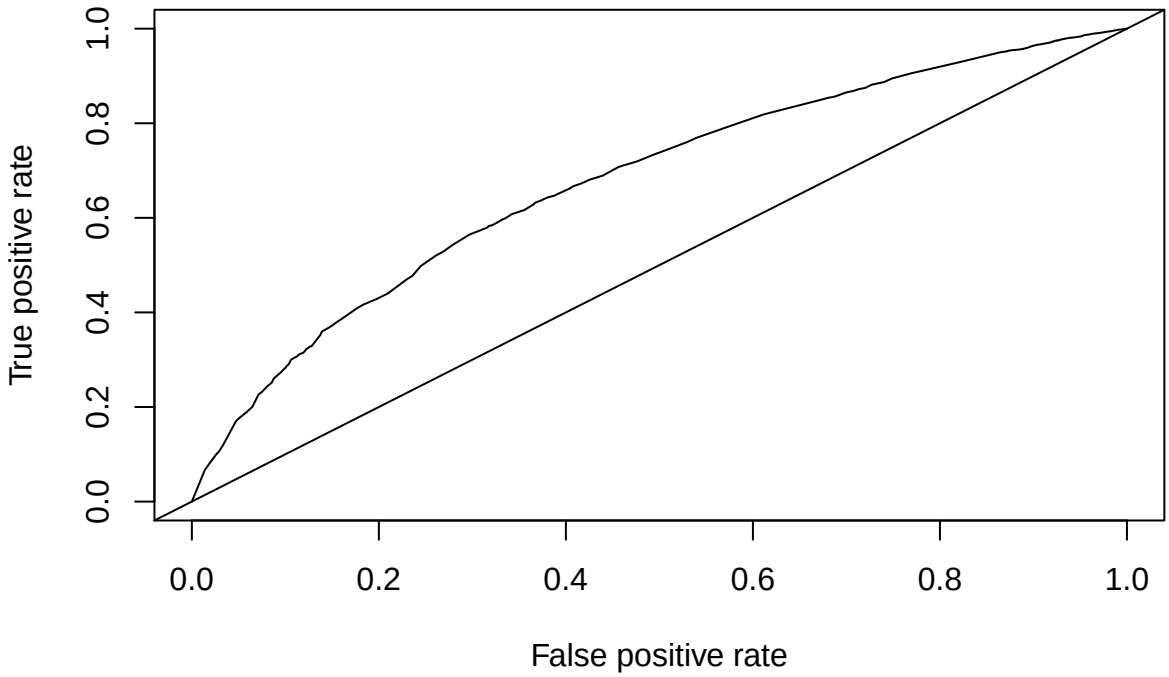
```
#Using the predicting function to get the scores in the test data set
xpredTst1<-predict(xgb_lsM1, dxTst1)
head(xpredTst1)
```

```
## [1] 0.5313393 0.5312767 0.5306311 0.5306311 0.5306578 0.5313295
```

```
#Auc
#ROC, AUC performance
#confusion matrix
table(pred=as.numeric(xpredTst1>0.5), act=colcdfTst1)
```

```
##     act
## pred     0     1
##    1  3600 20707
```

```
#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()
```

#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) %>% mu

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_sta
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm
totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=
```

```
## # A tibble: 10 x 14
##     tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431 0.532         111      3.94  -26.6   12.2   2.24  2431     0
##  2     2  2431 0.532         143      3.86  -31.3   14.0   2.17  2431     0
##  3     3  2431 0.532         209      4.12  -30.3   15.4   2.28  1154  1275
##  4     4  2431 0.531         271      5.52  -32.2   22.4   2.17     0  2431
##  5     5  2431 0.531         254      5.57  -30.0   30.9   2.25     0  2431
##  6     6  2431 0.531         364      5.58  -32.3   21.3   2.19     7   990
##  7     7  2431 0.531         434      5.83  -32.2   24.4   2.19     0     0
##  8     8  2430 0.531         487      5.63  -33.3   39.7   2.25     0     0
##  9     9  2430 0.531         596      5.48  -32.2   24.8   2.27     0     0
## 10    10  2430 0.530         731      5.60  -33.3   31.1   2.30     1     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

#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) %>% mu
```

```
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_s
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTer
totB=sum(grade=="B" ), totC=sum(grade=="C"), totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=
```

```
## # A tibble: 10 x 14
##     tile count  avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>  <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431  12.8            0     13.0    0      31.1   1.93     0     0
##  2     2  2431  10.5            0     10.6    0      39.7   2.02     0     0
##  3     3  2431   9.45           0      9.48   2.81   24.4   2.01     0    28
##  4     4  2431   8.63           0      8.64   0      20.0   2.09     0  1070
##  5     5  2431   7.76           0      7.80   0      30.9   2.13     0  2363
##  6     6  2431   6.62           0      6.53   0      17.0   2.15    39  2391
##  7     7  2431   5.51           0      5.50   1.48   14.0   2.14  1953   478
##  8     8  2430   4.69           0      4.65   1.98   12.2   2.20  2428     2
##  9     9  2430  -2.96        1170     -3.01 -33.3    15.9   2.63  1401   405
## 10    10  2430 -12.2         2430    -12.1  -33.3    10.1   3      203   390
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

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

```
d=1

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") )
```

```
## # A tibble: 10 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   244       11.9           0      12.4   7.56   23.8   1.70     0     0
##  2     2   243       12.1           0      12.2   5.16   24.8   1.85     0     0
##  3     3   243       11.9           0      12.1   4.46   23.0   1.92     0     0
##  4     4   243       11.9           0      11.7   3.40   20.8   2.05     0     0
##  5     5   243       12.1           0      12.2   0      24.8   1.82     0     0
##  6     6   243       13.0           0      13.1   3.88   29.0   2.01     0     0
##  7     7   243       13.7           0      14.2   5.13   30.0   2.02     0     0
##  8     8   243       13.4           0      13.5   5.28   27.2   2.05     0     0
##  9     9   243       14.2           0      14.7   8.86   31.1   1.87     0     0
## 10    10   243       14.3           0      14.4   6.90   29.6   1.99     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

#Performance of glm model for all grade loans in deciles for M1

```
predLS_glm <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mutate(


predLS_glm<- predLS_glm%>% mutate(tile=ntile(-glmPredls_pMin, 10))
```

```
predLS_glm %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(glmPredls_pMin),    numDefaults=
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672      0.958         184      4.27  -32.3   18.5    2.15  5202
##  2     2  5672      0.938         282      4.52  -33.3   19.4    2.20  4381
##  3     3  5672      0.921         436      4.79  -32.3   23.8    2.19  2940
##  4     4  5672      0.903         575      5.19  -32.3   23.5    2.19  1308
##  5     5  5671      0.884         662      5.62  -30.0   24.3    2.20   425
##  6     6  5671      0.863         825      5.69  -33.3   23.9    2.22    99
##  7     7  5671      0.839        1002      5.67  -32.3   25.0    2.23    20
##  8     8  5671      0.810        1117      5.64  -33.3   29.0    2.24     2
##  9     9  5671      0.768        1337      5.45  -32.2   30.8    2.27     1
## 10    10  5671      0.665        1807      4.83  -33.3   44.4    2.37     0
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#Actual Returns selected by grade M2 all grades

```
#Performance of glm model for lambda min
predRet_Trn_glm <- lcdfTrn_AR %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% mu

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),    numDefaul
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  5672       7.33        1097      7.21  -33.3   44.4    2.10    60
##  2     2  5672       6.36         920      6.53  -33.3   30.8    2.10   157
##  3     3  5672       5.91         906      6.03  -33.3   31.4    2.16   392
##  4     4  5672       5.57         845      5.87  -31.0   25.0    2.17   685
##  5     5  5671       5.26         846      5.14  -32.2   24.8    2.20  1059
##  6     6  5671       4.97         817      4.87  -33.3   27.3    2.23  1413
##  7     7  5671       4.68         728      4.62  -31.2   22.7    2.28  1933
##  8     8  5671       4.37         717      4.26  -32.3   19.9    2.28  2317
##  9     9  5671       3.98         701      3.72  -32.2   21.0    2.33  2934
## 10    10  5671       3.25         650      3.43  -32.3   22.9    2.41  3428
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#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") )
```

```
## # A tibble: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   284       7.69          17      6.89  -24.6   18.5   2.09    60   159
##  2     2   284       7.23          24      7.21  -33.3   23.8   1.95     0   204
##  3     3   284       7.13          36      6.74  -25.6   17.9   2.09     0   168
##  4     4   284       7.12          30      7.31  -27.9   21.7   1.94     0   119
##  5     5   284       7.18          38      6.93  -31.1   20.5   1.99     0    62
##  6     6   284       7.16          37      7.35  -28.6   20.0   1.96     0    17
##  7     7   284       7.16          27      8.51  -33.3   24.3   1.94     0    11
##  8     8   284       7.16          36      7.74  -28.7   21.9   2.12     0     1
##  9     9   284       7.15          50      6.99  -28.3   21.2   2.15     0     0
## 10    10   284       7.17          39      8.75  -28.6   24.8   1.97     0     1
## 11    11   284       7.23          56      7.66  -30.9   22.4   2.04     0     0
## 12    12   284       7.26          51      8.09  -28.6   29.0   2.01     0     0
## 13    13   283       7.24          56      7.30  -33.3   19.9   2.12     0     0
## 14    14   283       7.33          65      7.57  -29.7   26.8   2.09     0     0
## 15    15   283       7.28          69      7.32  -26.1   30.4   2.17     0     0
## 16    16   283       7.36          76      7.11  -28.5   34.1   2.24     0     0
## 17    17   283       7.42          92      5.18  -32.1   30.4   2.27     0     0
## 18    18   283       7.65          94      6.05  -31.0   36.7   2.21     0     0
## 19    19   283       7.94          94      6.86  -27.4   31.1   2.35     0     0
## 20    20   283       7.75         110      6.64  -32.1   44.4   2.40     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

#4 Modeling Loan Status on lower grade loans

#Create a RF model for low grade C-G loans M1 Loan Status

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

#Creating RF for loans in C - G
rf_M1_lg <- ranger(loan_status ~., data=subset(lg_lcdfTrn, select=-c(annRet, actualTerm, actualReturn)),
probability=TRUE, importance='permutation')
```

#Performance of RF C-G in Deciles M1

```
lg_scoreTstRF <- lg_lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(score=(predict(rf_M1_lg,lg_lcdfTst))$predictions[,"Fully Paid"])

lg_scoreTstRF <- lg_scoreTstRF %>% mutate(tile=ntile(-score, 10))

lg_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") )
```

```
## # A tibble: 10 x 14
##    tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  1116 0.891         153      7.05  -33.3   24.6   2.07     0     0
##  2     2  1116 0.853         176      6.68  -31.1   25.8   2.15     0     0
##  3     3  1116 0.829         194      6.16  -32.2   30.8   2.17     0     0
##  4     4  1116 0.809         205      6.01  -30.0   39.7   2.21     0     0
```

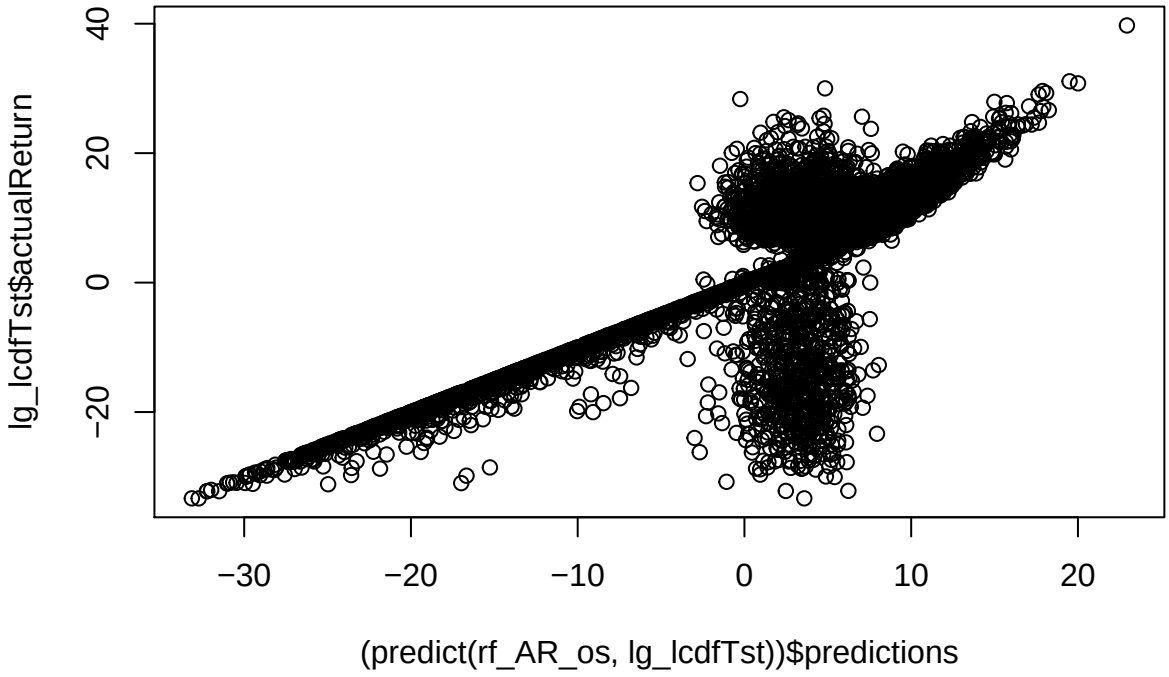
```
##  5     5  1116 0.789         236      5.74  -28.4   26.2   2.24     0     0
##  6     6  1116 0.769         254      5.54  -32.2   25.4   2.27     0     0
##  7     7  1115 0.747         260      5.17  -31.0   24.8   2.30     0     0
##  8     8  1115 0.721         276      5.38  -30.9   31.1   2.33     0     0
##  9     9  1115 0.687         306      5.13  -33.3   30.0   2.29     0     0
## 10    10  1115 0.614         401      4.00  -33.3   29.0   2.39     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

#RF Actual Returns selected by grade M2 low grades

```
#Choose the Actual Returns Random forest we want for here
rfPredRet_Tst_lg<- predict(rf_AR_os,  lg_lcdfTst)
sqrt(mean((rfPredRet_Tst_lg$predictions- lg_lcdfTst$actualReturn)^2))
```

```
## [1] 6.459928
```

```
plot ((predict(rf_AR_os,  lg_lcdfTst))$predictions, lg_lcdfTst$actualReturn)
```



#Rf – Performance by Deciles for Test Data low grades

```
lg_predRet_Tst<- lg_lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta

lg_predRet_Tst<- lg_predRet_Tst %>% mutate(tile=ntile(-predRet_lgTst, 10))

lg_predRet_Tst %>% group_by(tile) %>%  summarise(count=n(), avgpredRet=mean(predRet_lgTst),    numDefau
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  1116     11.2             2     15.6    9.72  39.7    0.907     0
##  2     2  1116      9.01            5     12.1    6.47  20.2    1.45      0
##  3     3  1116      7.92           16     10.2  -23.3   23.8    2.08      0
##  4     4  1116      7.13           13      9.01 -19.3   25.6    2.51      0
##  5     5  1116      6.44           39      8.00 -32.2   20.9    2.66      0
##  6     6  1116      5.66          100      7.32 -30.0   22.3    2.53      0
```

```
##  7     7  1115      4.54          211      6.13 -30.0   30.0    2.30      0
##  8     8  1115      3.13          315      4.71 -33.3   25.5    2.34      0
##  9     9  1115     -0.453         645      1.09 -30.8   28.4    2.63      0
## 10    10  1115    -16.7          1115    -17.3  -33.3   -7.15   3         0
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

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

```
d=1
pRetSc_RF <- lg_predRet_Tst %>% mutate(poScore=lg_scoreTstRF$score)
pRet_d_RF <- pRetSc_RF %>% filter(tile<=d)
pRet_d_RF<- pRet_d_RF %>% mutate(tile2=ntile(-poScore, 20))

pRet_d_RF %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(predRet_lgTst),
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") )
```

```
## # A tibble: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1    56       11.1           0      14.3  10.7    24.4  0.691     0     0
##  2     2    56       11.0           0      14.4  11.1    24.6  0.644     0     0
##  3     3    56       11.1           0      14.5  10.3    24.0  0.774     0     0
##  4     4    56       10.8           0      14.3  10.3    20.7  0.731     0     0
##  5     5    56       10.8           0      14.0  10.5    20.0  0.784     0     0
##  6     6    56       10.9           0      14.6  11.4    22.9  0.745     0     0
##  7     7    56       11.2           0      15.3  11.0    39.7  0.748     0     0
##  8     8    56       10.9           0      14.4  11.2    21.0  0.930     0     0
##  9     9    56       11.4           0      15.6  12.0    22.5  0.791     0     0
## 10    10    56       10.9           1      15.2   9.72   26.2  0.898     0     0
## 11    11    56       11.3           0      15.7  11.6    22.5  0.900     0     0
## 12    12    56       11.1           0      15.7  11.7    24.8  0.934     0     0
## 13    13    56       11.0           0      15.3  10.5    19.6  1.19      0     0
## 14    14    56       11.5           0      16.4  12.3    25.5  1.00      0     0
## 15    15    56       11.2           0      16.0  11.2    31.1  0.996     0     0
## 16    16    56       11.3           0      16.4  12.6    26.5  1.03      0     0
## 17    17    55       11.6           0      16.8  11.8    24.8  1.01      0     0
## 18    18    55       11.9           1      17.7   9.79   29.3  1.09      0     0
## 19    19    55       11.8           0      18.0  13.0    27.7  1.05      0     0
## 20    20    55       12.0           0      18.4  11.9    29.0  1.22      0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

#4 GLM Lower Grade Models #Modeling Loan Status on lower grade loans

#Creating a GLM model for C-G loans M1

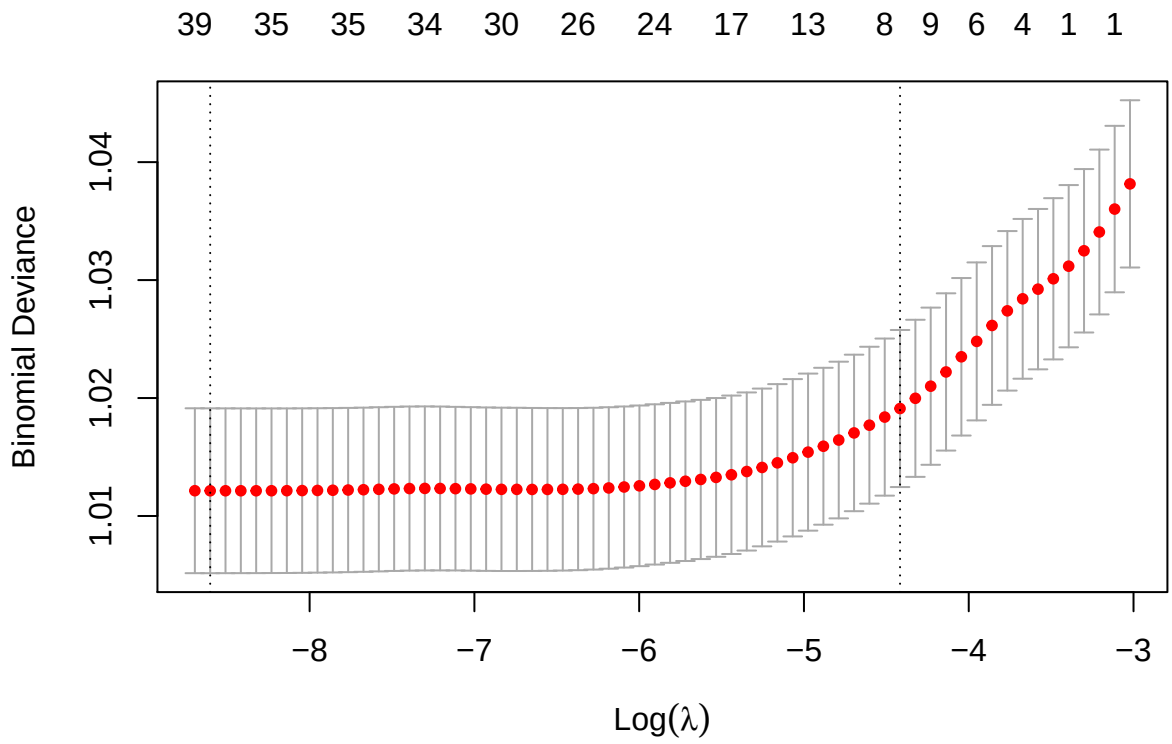
```
#Selecting only Grades C-G
lg_lcdfTrn<-lcdfTrn_AR %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')
lg_lcdfTst<-lcdfTst_AR %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')
```

#Finding Lambda Values

```
levels(lg_lcdfTrn$loan_status)
```

```
## [1] "Charged Off" "Fully Paid"
```

```
yTrnGr<-factor(if_else(lg_lcdfTrn$loan_status=="Fully Paid", '1', '0') )

xDTrnGR <- lg_lcdfTrn %>%select(-loan_status, -actualTerm, -annRet, -actualReturn,)

glmls_cvGR <-cv.glmnet(data.matrix(xDTrnGR), yTrnGr, family="binomial")
plot(glmls_cvGR)
```




```
glmls_cvGR$lambda.min
```

```
## [1] 0.0001834884
```

```
glmls_cvGR$lambda.1se
```

```
## [1] 0.01207231
```

```
coef(glmls_cvGR, s=glmls_cvGR$lambda.min)
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                3.286937e+00
## loan_amnt                 -1.470937e-05
## int_rate                  -2.582633e-05
## installment                .
## grade                     -9.162664e-02
## sub_grade                 -4.088506e-02
## emp_length                -1.787727e-02
## home_ownership            -7.261339e-02
## annual_inc                 1.238407e-07
## verification_status       -2.139095e-02
## purpose                    9.547551e-03
## dti                       -1.491083e-02
## earliest_cr_line           1.248388e-04
```

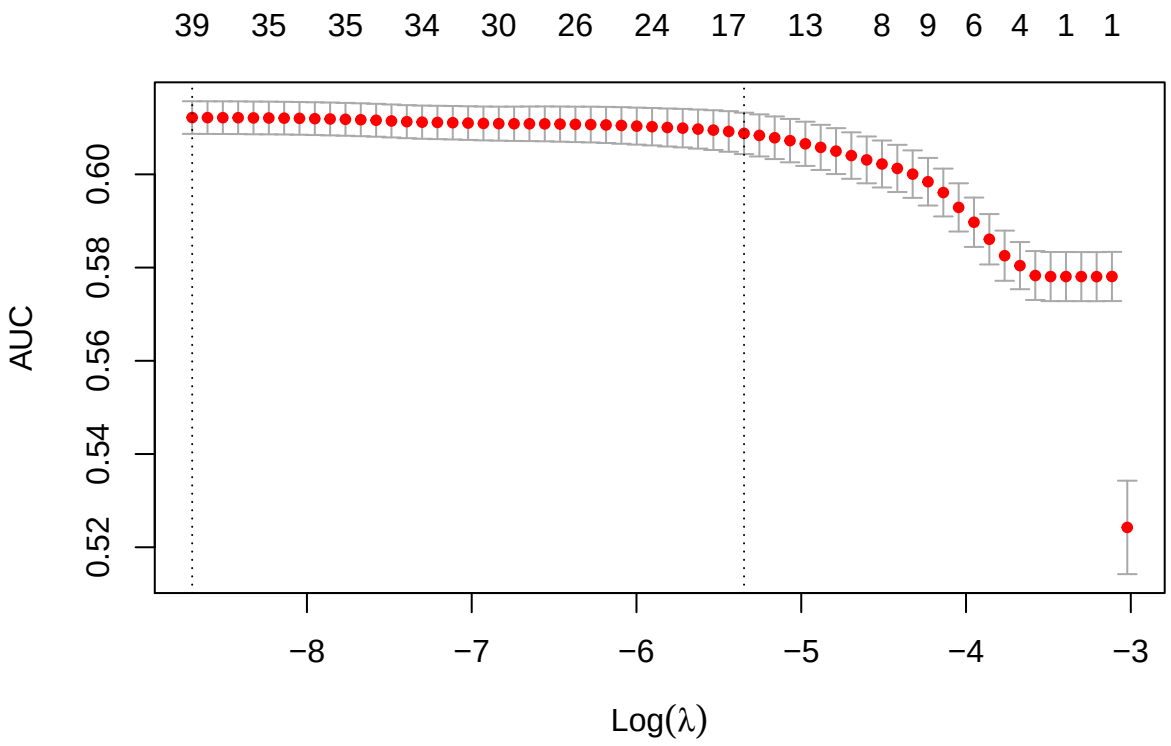
```
## initial_list_status         3.028831e-02
## total_rev_hi_lim            3.832130e-06
## acc_open_past_24mths       -3.908243e-02
## avg_cur_bal                -1.418333e-06
## bc_open_to_buy             -9.160605e-06
## bc_util                    -6.127845e-04
## mo_sin_old_il_acct          2.907767e-07
## mo_sin_old_rev_tl_op        3.357875e-04
## mo_sin_rcnt_rev_tl_op       9.304062e-05
## mo_sin_rcnt_tl             -1.547285e-03
## mths_since_recent_bc       -1.529673e-04
## mths_since_recent_inq       3.681208e-03
## num_accts_ever_120_pd      -1.158374e-02
## num_actv_bc_tl             -3.854554e-02
## num_actv_rev_tl             .
## num_bc_sats                 1.333380e-02
## num_bc_tl                  -1.405477e-02
## num_il_tl                  -1.025248e-03
## num_op_rev_tl              -6.975732e-04
## num_rev_accts               1.162117e-02
## num_rev_tl_bal_gt_0        -2.092788e-02
## num_sats                   -1.050963e-04
## num_tl_op_past_12m          9.316316e-03
## pct_tl_nvr_dlq             -2.154134e-03
## percent_bc_gt_75           -1.172728e-03
## tot_hi_cred_lim             9.676613e-07
## total_bal_ex_mort          -2.753801e-06
## total_bc_limit              5.859932e-06
## total_il_high_credit_limit  3.063930e-06
```

```
coef(glmls_cvGR, s=glmls_cvGR$lambda.1se)
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                2.399664e+00
## loan_amnt                   .
## int_rate                    .
## installment                 .
## grade                      -5.294246e-02
## sub_grade                  -4.076956e-02
## emp_length                  .
## home_ownership             -2.239738e-02
## annual_inc                  .
## verification_status         .
## purpose                     .
## dti                        -7.057016e-03
## earliest_cr_line            .
## initial_list_status         .
## total_rev_hi_lim            .
## acc_open_past_24mths       -1.386275e-02
## avg_cur_bal                 .
## bc_open_to_buy              .
## bc_util                     .
## mo_sin_old_il_acct          .
## mo_sin_old_rev_tl_op        .
```

```
## mo_sin_rcnt_rev_tl_op       .
## mo_sin_rcnt_tl              .
## mths_since_recent_bc        .
## mths_since_recent_inq       .
## num_accts_ever_120_pd       .
## num_actv_bc_tl             -4.808882e-03
## num_actv_rev_tl             .
## num_bc_sats                 .
## num_bc_tl                   .
## num_il_tl                   .
## num_op_rev_tl               .
## num_rev_accts               .
## num_rev_tl_bal_gt_0        -7.170783e-03
## num_sats                    .
## num_tl_op_past_12m          .
## pct_tl_nvr_dlq              .
## percent_bc_gt_75            .
## tot_hi_cred_lim             3.601523e-07
## total_bal_ex_mort           .
## total_bc_limit              .
## total_il_high_credit_limit  .
```

```
glmls_cvGR_auc<- cv.glmnet(data.matrix(xDTrnGR), yTrnGr, family="binomial", type.measure = "auc")
plot(glmls_cvGR_auc)
```




```
#loss Value at Lambda.SE
glmls_cvGR_auc$cvm [ which(glmls_cvGR_auc$lambda == glmls_cvGR_auc$lambda.1se) ]
```

```
## [1] 0.6087879
```

```
#loss Value at Lambda.Min
glmls_cvGR_auc$cvm [ which(glmls_cvGR_auc$lambda == glmls_cvGR_auc$lambda.min) ]
```

```
## [1] 0.6121862
```

#Predictions

```
#Log value lambda min
glmPredls_1GR=predict ( glmls_cvGR,data.matrix(xDTrnGR), s="lambda.min" )
head(glmPredls_1GR)
```

```
##             1
## [1,] 1.494349
## [2,] 1.456752
## [3,] 1.623375
## [4,] 1.285253
## [5,] 1.506315
## [6,] 1.153115
```

```
#probability lambda min
glmPredls_1GRP=predict ( glmls_cvGR,data.matrix(xDTrnGR), s="lambda.min", type="response")
head(glmPredls_1GRP)
```

```
##              1
## [1,] 0.8167302
## [2,] 0.8110354
## [3,] 0.8352601
## [4,] 0.7833426
## [5,] 0.8185144
## [6,] 0.7600795
```

```
#Log value Lambda.SE
glmPredls_1GRSE=predict ( glmls_cvGR,data.matrix(xDTrnGR), s="lambda.1se" )
head(glmPredls_1GRSE)
```

```
##             1
## [1,] 1.498948
## [2,] 1.465315
## [3,] 1.484921
## [4,] 1.297384
## [5,] 1.462754
## [6,] 1.369950
```

```
#probability value Lambda.SE
glmPredls_1GRPMin=predict ( glmls_cvGR,data.matrix(xDTrnGR), s="lambda.1se", type="response")
head(glmPredls_1GRPMin)
```

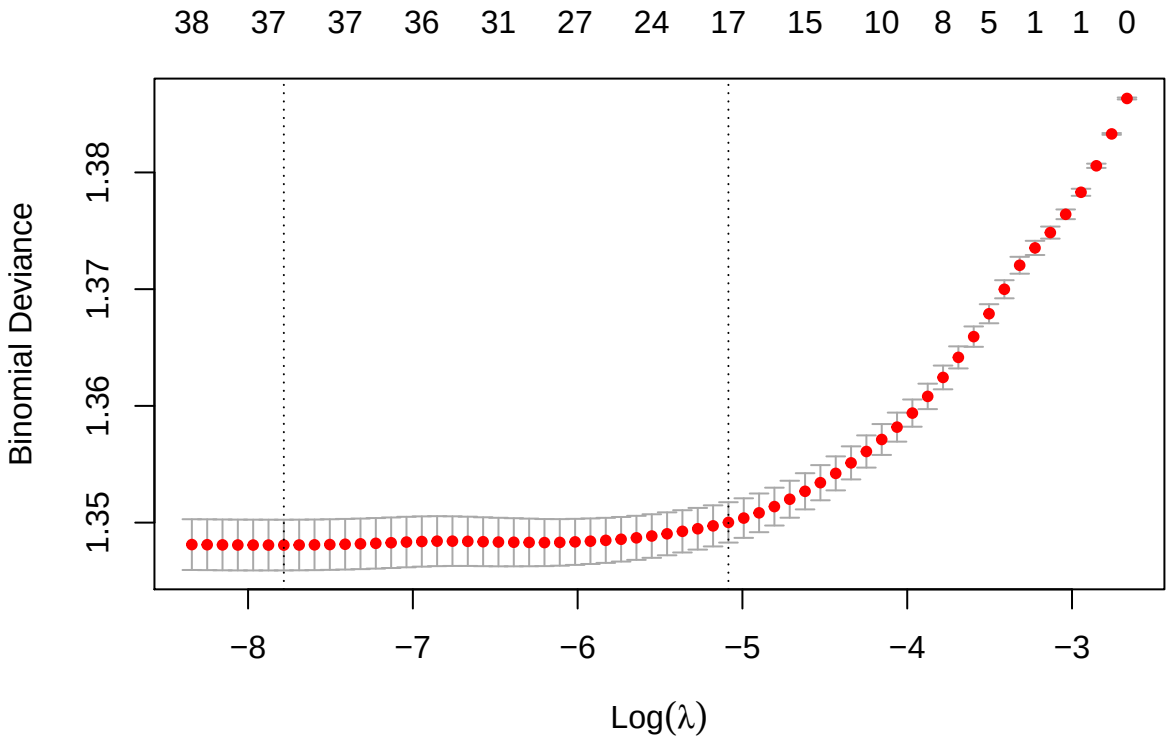
```
##              1
## [1,] 0.8174175
## [2,] 0.8123442
## [3,] 0.8153146
## [4,] 0.7853944
## [5,] 0.8119536
## [6,] 0.7973721
```

#AUC values

```
predsaucGR <- prediction(glmPredls_1GRP, lg_lcdfTrn$loan_status, label.ordering = c("Charged Off", "Full
aucPerfGR <- performance(predsaucGR, "auc")
#performance is worse with the lower grade loans. auc value of only .614
```

#Balancing Data

```
wtsGR=if_else(yTrnGr==0,1-sum(yTrnGr==0)/length(yTrnGr),1-sum(yTrnGr==1)/length(yTrnGr))

glmlswGR_cv<- cv.glmnet(data.matrix(xDTrnGR), yTrnGr, family="binomial", weights = wtsGR)
plot(glmlswGR_cv)
```




```
predsaucGRB <- prediction(glmPredls_1GRP, lg_lcdfTrn$loan_status, label.ordering =
c("Charged Off", "Fully Paid"))
aucPerfGRB <- performance(predsaucGRB, "auc")
```

4 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')
```

#4 XGBOOST Spliting 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. Use setdiff() to
# avoid overlapping subsets of indices.
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)

# Finally, output the three dataframes for training, validation and test.
lcdfTrn_AR2 <- lcdf_AR_LowGrades[indicesTraining2, ]
lcdfVal_AR2 <- lcdf_AR_LowGrades[indicesValidation2, ]
lcdfTst_AR2 <- lcdf_AR_LowGrades[indicesTest2, ]
```

4 XGBOOST Builing the model for grades C and lower

```
#Needs all data to be numeric -- so we convert categorical
#(i.e. factor) variables using one-hot encoding
# we use the dummyVars function in the 'caret' package
#to convert factor variables to # dummy-variables
fdum4<-dummyVars(~.,data=lcdf_AR_LowGrades %>% select(-loan_status)) #do not include loan_status for th
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)
```

```
## [1] "Charged Off" "Fully Paid"
```

```
dylcdf4 <- class2ind(lcdf_AR_LowGrades$loan_status, drop2nd = FALSE)
# and then decide which one to keep
#fplcdf <- dycldf [ , 2] # or,
colcdf4 <- dylcdf4 [ , 1]

#Training, 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)
#er picked in loan status to dummy variable step
# (annRet, actualTerm, actualReturn, total_pymnt)
#These variables are useful for performance assessment,
#but should not be used in the model

xgbWatchlist4 <- list(train = dxTrn4, eval = dxTst4)
#we can watch the progress of learning thru performance on these datasets
# Include a gbWatchlist so the eearly_stopping_rounds = 10
#that you are goint to use in the model can use
#it as a base to know when to stop

#Calculating weights
sqrt(sum(dxlcdfTrn4==0) / sum(dxlcdfTrn4==1))  #8.50
```

```
## [1] 8.504025
```

```r
#use 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_byt

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
}
```

```
## [1]  train-error:0.784793    train-auc:0.581618  eval-error:0.782395 eval-auc:0.556519
## 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:0.784793    train-auc:0.581618  eval-error:0.782395 eval-auc:0.556519
## [3]  train-error:0.784793    train-auc:0.581618  eval-error:0.782395 eval-auc:0.556519
## [4]  train-error:0.784793    train-auc:0.589914  eval-error:0.782395 eval-auc:0.561610
## [5]  train-error:0.784793    train-auc:0.593321  eval-error:0.782395 eval-auc:0.564567
## [6]  train-error:0.784793    train-auc:0.595921  eval-error:0.782395 eval-auc:0.569289
## [7]  train-error:0.784793    train-auc:0.597790  eval-error:0.782395 eval-auc:0.570816
## [8]  train-error:0.784793    train-auc:0.597629  eval-error:0.782395 eval-auc:0.571134
## [9]  train-error:0.784793    train-auc:0.597197  eval-error:0.782395 eval-auc:0.572419
## [10] train-error:0.784793    train-auc:0.597348  eval-error:0.782395 eval-auc:0.571032
## [11] train-error:0.784793    train-auc:0.596920  eval-error:0.782395 eval-auc:0.569906
## [12] train-error:0.784793    train-auc:0.596778  eval-error:0.782395 eval-auc:0.569724
## [13] train-error:0.784793    train-auc:0.598986  eval-error:0.782395 eval-auc:0.571339
## [14] train-error:0.784793    train-auc:0.598850  eval-error:0.782395 eval-auc:0.571228
## [15] train-error:0.784793    train-auc:0.598443  eval-error:0.782395 eval-auc:0.570994
## [16] train-error:0.784793    train-auc:0.598207  eval-error:0.782395 eval-auc:0.571025
## [17] train-error:0.784793    train-auc:0.598219  eval-error:0.782395 eval-auc:0.570973
## [18] train-error:0.784793    train-auc:0.598169  eval-error:0.782395 eval-auc:0.571016
## [19] train-error:0.784793    train-auc:0.598251  eval-error:0.782395 eval-auc:0.570531
## Stopping. Best iteration:
## [9]  train-error:0.784793    train-auc:0.597197  eval-error:0.782395 eval-auc:0.572419
##
## [1]  train-error:0.750633    train-auc:0.605096  eval-error:0.752664 eval-auc:0.571227
## 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:0.755009    train-auc:0.622636  eval-error:0.761709 eval-auc:0.575253
## [3]  train-error:0.758540    train-auc:0.630194  eval-error:0.761082 eval-auc:0.580167
## [4]  train-error:0.759807    train-auc:0.632140  eval-error:0.762246 eval-auc:0.581120
## [5]  train-error:0.758655    train-auc:0.632429  eval-error:0.762156 eval-auc:0.581466
## [6]  train-error:0.756928    train-auc:0.637322  eval-error:0.760276 eval-auc:0.583987
## [7]  train-error:0.757887    train-auc:0.638162  eval-error:0.761709 eval-auc:0.585109
## [8]  train-error:0.761035    train-auc:0.640471  eval-error:0.764126 eval-auc:0.585605
## [9]  train-error:0.762225    train-auc:0.641368  eval-error:0.765291 eval-auc:0.586655
## [10] train-error:0.763107    train-auc:0.641804  eval-error:0.766544 eval-auc:0.586494
## [11] train-error:0.764105    train-auc:0.641513  eval-error:0.766544 eval-auc:0.585826
## [12] train-error:0.764144    train-auc:0.640907  eval-error:0.767082 eval-auc:0.586019
## [13] train-error:0.766715    train-auc:0.642908  eval-error:0.768693 eval-auc:0.586344
## [14] train-error:0.766907    train-auc:0.642548  eval-error:0.768604 eval-auc:0.586204
```

```
## [15] train-error:0.766562    train-auc:0.642665  eval-error:0.768335 eval-auc:0.586470
## [16] train-error:0.765449    train-auc:0.642866  eval-error:0.768246 eval-auc:0.586315
## [17] train-error:0.765372    train-auc:0.643165  eval-error:0.768604 eval-auc:0.586681
## [18] train-error:0.765756    train-auc:0.643352  eval-error:0.768873 eval-auc:0.586549
## [19] train-error:0.766946    train-auc:0.643566  eval-error:0.769141 eval-auc:0.586475
## [20] train-error:0.767214    train-auc:0.643667  eval-error:0.769679 eval-auc:0.586486
## [21] train-error:0.766715    train-auc:0.643212  eval-error:0.769320 eval-auc:0.586262
## [22] train-error:0.766331    train-auc:0.642727  eval-error:0.768873 eval-auc:0.586116
## [23] train-error:0.766216    train-auc:0.643592  eval-error:0.769052 eval-auc:0.586878
## [24] train-error:0.766178    train-auc:0.643811  eval-error:0.769141 eval-auc:0.586882
## [25] train-error:0.766562    train-auc:0.644176  eval-error:0.769231 eval-auc:0.586628
## [26] train-error:0.766562    train-auc:0.643666  eval-error:0.769231 eval-auc:0.586459
## [27] train-error:0.766408    train-auc:0.643149  eval-error:0.768514 eval-auc:0.586393
## [28] train-error:0.766331    train-auc:0.643393  eval-error:0.769320 eval-auc:0.586408
## [29] train-error:0.767483    train-auc:0.644027  eval-error:0.769947 eval-auc:0.586830
## [30] train-error:0.766907    train-auc:0.643669  eval-error:0.770037 eval-auc:0.586707
## [31] train-error:0.766946    train-auc:0.643851  eval-error:0.769947 eval-auc:0.586922
## [32] train-error:0.766830    train-auc:0.643805  eval-error:0.770126 eval-auc:0.587060
## [33] train-error:0.767176    train-auc:0.643670  eval-error:0.769858 eval-auc:0.587050
## [34] train-error:0.766984    train-auc:0.643797  eval-error:0.769768 eval-auc:0.587353
## [35] train-error:0.767061    train-auc:0.643833  eval-error:0.769410 eval-auc:0.587462
## [36] train-error:0.766869    train-auc:0.643776  eval-error:0.769410 eval-auc:0.587554
## [37] train-error:0.766869    train-auc:0.643616  eval-error:0.768962 eval-auc:0.587517
## [38] train-error:0.766447    train-auc:0.644194  eval-error:0.769320 eval-auc:0.587823
## [39] train-error:0.766293    train-auc:0.644051  eval-error:0.769141 eval-auc:0.587964
## [40] train-error:0.766830    train-auc:0.644174  eval-error:0.769589 eval-auc:0.587783
## [41] train-error:0.766447    train-auc:0.644133  eval-error:0.769410 eval-auc:0.587602
## [42] train-error:0.766293    train-auc:0.643786  eval-error:0.769231 eval-auc:0.587449
...
```

```
xgbParamGrid4
```

```
##   max_depth   eta bestTree bestPerf
## 1         2 0.001        9 0.572419
## 2         5 0.001       73 0.589245
## 3         2 0.010       19 0.579483
## 4         5 0.010       48 0.593521
## 5         2 0.100       93 0.607490
## 6         5 0.100       40 0.600758
```

```
# max_depth eta bestTree bestPerf
# 2 0.001   9   0.572419     <--
# 5 0.001   73  0.589245
# 2 0.010   19  0.579483
# 5 0.010   48  0.593521
# 2 0.100   93  0.607490
# 5 0.100   40  0.600758

xgbParam_Best4 <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1, colsample

# 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
```

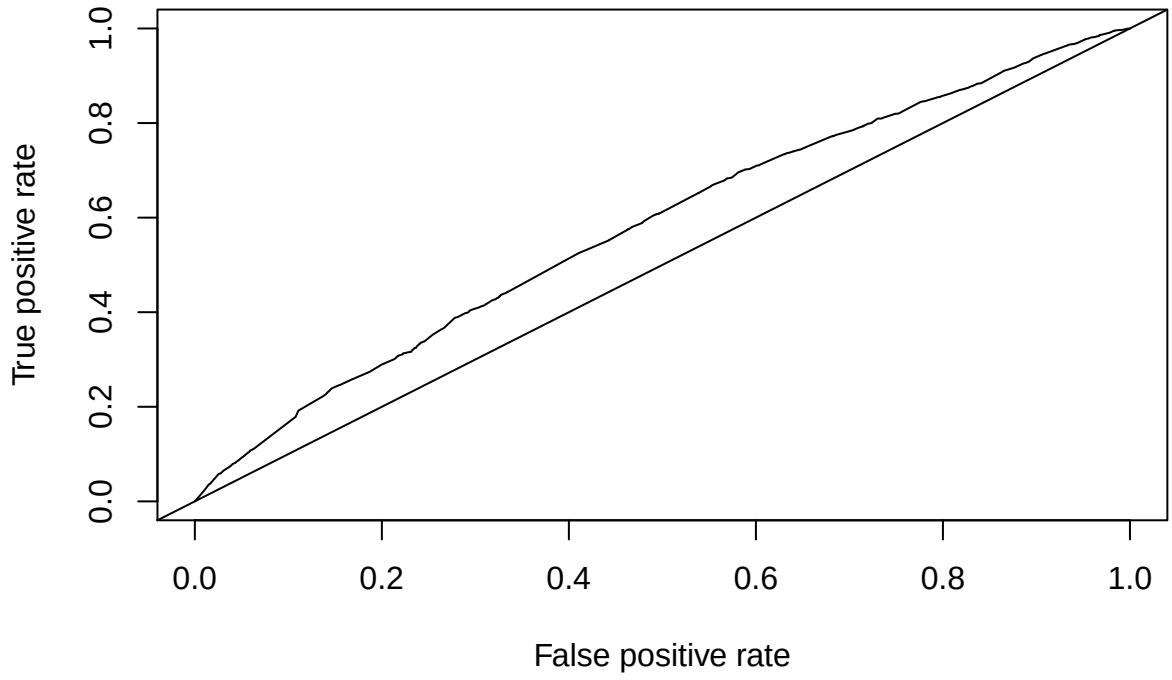
```
xpredTrn4<-predict(xgb_lsM4, dxTrn4)
head(xpredTrn4)
```

```
## [1] 0.5070234 0.5057521 0.5068929 0.5071743 0.5065823 0.5069507
```

```
#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)
```

```
##     act
## pred    0    1
##    1 8737 2430
```

```
#ROC, AUC performance
pred_xgb_lsM4<-prediction(xpredTst4, lcdfTst_AR2$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)
```



4

XGBOOST Deciles for grades C and lower M1

```
xpredTstM4<-predict(xgb_lsM4, dxTst4)

scoreTst_xgb_ls4 <- lcdfTst_AR2 %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% m

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_st
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTerm
```

```
## # A tibble: 10 x 12
##     tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totC  totD
```

```
##   * <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##   1     1  1117 0.511         361      4.97  -32.2   30.8   2.29     0   296
##   2     2  1117 0.510         289      6.25  -33.3   33.8   2.23     0  1019
##   3     3  1117 0.509         289      6.08  -31.0   34.1   2.28     0   889
##   4     4  1117 0.508         244      5.79  -31.0   25.9   2.25   464   635
##   5     5  1117 0.507         245      4.96  -30.0   22.2   2.26  1045    72
##   6     6  1117 0.507         240      5.18  -29.9   39.7   2.19   992   125
##   7     7  1117 0.506         206      5.75  -33.3   22.9   2.18  1061    55
##   8     8  1116 0.506         189      6.14  -33.3   19.3   2.20  1058    57
##   9     9  1116 0.506         197      5.53  -32.2   23.8   2.30  1086    30
## 10    10  1116 0.505         170      6.20  -33.3   23.5   2.22  1104    12
## # ... with 2 more variables: totE <int>, totF <int>
```

#XGBOOST Model 2 Actual returns for lower grades #Finding which hyper-parameters work best – experiment with a grid of parameter values

```
# we are predicting actualReturn a numeric variable

#converting factor variables to dummy-variables
fdum5<-dummyVars(~.,lcdf_AR_LowGrades %>% select(-actualReturn))

dxlcdf5 <- predict(fdum5, lcdf_AR_LowGrades)

#Training, 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)
# (annRet, actualTerm) These variables are useful
#for performance assessment, but should not be used in the model

xgbWatchlist5 <- list(train = dxTrn5, eval = dxTst5)
#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
```

```
#Parameter with the grid of options we want
#to test 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
)
```

```
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
}
```

```
## [1]  train-rmse:11.896250    eval-rmse:11.945131
## 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:11.886497    eval-rmse:11.935389
## [3]  train-rmse:11.876825    eval-rmse:11.925717
## [4]  train-rmse:11.867113    eval-rmse:11.916010
## [5]  train-rmse:11.857397    eval-rmse:11.906307
## [6]  train-rmse:11.847689    eval-rmse:11.896608
## [7]  train-rmse:11.837992    eval-rmse:11.886926
## [8]  train-rmse:11.828320    eval-rmse:11.877269
## [9]  train-rmse:11.818649    eval-rmse:11.867606
## [10] train-rmse:11.808993    eval-rmse:11.857956
## [11] train-rmse:11.799455    eval-rmse:11.848433
## [12] train-rmse:11.797049    eval-rmse:11.846118
## [13] train-rmse:11.787426    eval-rmse:11.836511
## [14] train-rmse:11.777813    eval-rmse:11.826911
## [15] train-rmse:11.768217    eval-rmse:11.817325
## [16] train-rmse:11.758711    eval-rmse:11.807827
## [17] train-rmse:11.749124    eval-rmse:11.798244
## [18] train-rmse:11.739547    eval-rmse:11.788681
## [19] train-rmse:11.729977    eval-rmse:11.779121
## [20] train-rmse:11.720433    eval-rmse:11.769592
## [21] train-rmse:11.710888    eval-rmse:11.760060
## [22] train-rmse:11.701355    eval-rmse:11.750538
## [23] train-rmse:11.691836    eval-rmse:11.741024
## [24] train-rmse:11.682324    eval-rmse:11.731521
## [25] train-rmse:11.672824    eval-rmse:11.722037
## [26] train-rmse:11.663405    eval-rmse:11.712614
## [27] train-rmse:11.653944    eval-rmse:11.703163
...
```

#Plugin the tune parameters

```
xgbParamGrid5
```

```
##   max_depth   eta bestTree bestPerf
## 1         2 0.001     1000 6.825973
## 2         5 0.001     1000 6.742201
## 3         2 0.010      786 5.004888
## 4         5 0.010      590 4.986000
## 5         2 0.100      125 5.002567
## 6         5 0.100       59 5.011881
```

```
# max_depth eta bestTree bestPerf
# 2 0.001   1000    6.764726
# 5 0.001   1000    6.805881
# 2 0.010   824 5.004813
```

```
# 5 0.010   640 4.988406
# 2 0.100   132 5.001203    <.
# 5 0.100   60  4.999652

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)
```

```
## [1]   9.199468   8.846642   9.141410   9.949840 -11.183322   9.455641
```

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

#Error
sqrt(mean((xpredTst5- colcdfTst5)^2)) #5.00
```

```
## [1] 5.00919
```

#XGBOOST M2 Actual Returns Performance of Boosting grades C and lower

```
xpredTstM5<-predict(xgb_lsM5, dxTst5)

scoreTst_xgb_ls5 <- lcdfTst_AR2 %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% r

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_s
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(actualTer
```

```
## # A tibble: 10 x 12
##     tile count  avgSc numDefaults avgActRet minRet maxRet avgTer  totC  totD
##  * <int> <int>  <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  1117  13.7            0     14.2    0      34.1   1.96     0   335
##  2     2  1117  12.0            0     12.0    0      25.3   2.00     0  1115
##  3     3  1117  11.1            0     11.2    0      27.7   2.00   172   945
##  4     4  1117  10.3            0     10.6    3.96   39.7   1.95  1101    16
##  5     5  1117   9.83           0      9.89   0      24.4   1.98  1117     0
##  6     6  1117   9.52           0      9.42   4.47   23.8   2.01  1117     0
##  7     7  1117   9.25           0      8.98   0      23.5   2.12  1117     0
##  8     8  1116   5.42         198      5.09 -33.3    22.3   2.39  1057    42
##  9     9  1116 -11.5         1116    -11.8  -33.3    12.5   3      635   304
## 10    10  1116 -12.1         1116    -12.7  -33.3    10.2   3      494   433
## # ... with 2 more variables: totE <int>, totF <int>
```

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

```
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
```

```
## # A tibble: 10 x 12
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totC  totD
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   112       13.8           0      14.4   8.86   25.3   1.93     0     0
##  2     2   112       13.8           0      13.6   0      23.3   2.07     0     0
##  3     3   112       13.7           0      14.5   0      26.5   1.80     0     0
##  4     4   112       13.6           0      14.6   5.66   24.0   1.90     0     0
##  5     5   112       13.6           0      14.4   7.61   30.8   1.94     0    11
##  6     6   112       13.5           0      13.4   0      23.2   2.05     0    86
##  7     7   112       13.8           0      14.7   8.47   33.8   1.87     0    81
##  8     8   111       13.8           0      14.0   8.69   30.0   2.07     0    66
##  9     9   111       13.5           0      14.6   5.28   34.1   1.81     0    21
## 10    10   111       14.1           0      13.9   8.24   26.9   2.13     0    70
## # ... with 2 more variables: totE <int>, totF <int>
```

#performance of glm model for low grade loans in deciles for M1

```
lg_xDTrn<-lg_lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn)

lg_predLS_glm <- lg_lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>% muta


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=su
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2607      0.896         287      7.36  -33.3   24.3    2.06     0
##  2     2  2607      0.864         356      6.68  -33.3   23.9    2.15     0
##  3     3  2607      0.846         462      5.91  -31.2   24.8    2.21     0
##  4     4  2607      0.830         470      6.22  -32.2   25.0    2.21     0
##  5     5  2607      0.814         503      5.74  -32.2   27.1    2.22     0
##  6     6  2606      0.797         574      5.57  -33.3   29.0    2.25     0
##  7     7  2606      0.775         588      5.72  -32.2   27.1    2.24     0
##  8     8  2606      0.748         642      5.41  -32.2   30.8    2.29     0
##  9     9  2606      0.708         768      4.80  -33.3   34.1    2.36     0
## 10    10  2606      0.610         926      4.74  -33.3   44.4    2.39     0
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#Actual Retruns selected by grade M2 low grades

```
#Performance of glm model for lambda min
lg_predRet_Trn_glm <- lg_lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%

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),    numl
```

```
## # A tibble: 10 x 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTerm  totA
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>   <dbl> <int>
##  1     1  2607       7.81         582      7.55  -33.3   44.4    2.12     0
##  2     2  2607       6.83         514      7.03  -33.3   35.0    2.11     0
##  3     3  2607       6.42         496      6.67  -33.3   30.8    2.10     0
##  4     4  2607       6.09         507      6.32  -32.2   25.9    2.19     0
##  5     5  2607       5.82         517      6.21  -33.3   31.4    2.19     0
##  6     6  2606       5.56         498      6.10  -31.0   25.0    2.22     0
##  7     7  2606       5.30         584      5.14  -32.2   24.8    2.27     0
##  8     8  2606       5.02         586      4.97  -33.3   27.3    2.33     0
##  9     9  2606       4.66         604      4.54  -30.0   22.7    2.39     0
## 10    10  2606       3.97         688      3.61  -32.2   22.9    2.46     0
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

#glm - Consider Top d deciles from Model 2(actualReturn), ranked by M1 scores(loan_Status) low grades

```
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") )
```

```
## # A tibble: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##  * <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   131       8.28          11      8.94  -33.3   22.4   1.85     0     0
##  2     2   131       7.67          15      8.25  -27.4   17.9   1.90     0     0
##  3     3   131       7.61          16      7.24  -28.8   21.7   2.06     0     0
##  4     4   131       7.58          17      7.63  -28.6   22.3   1.92     0     0
##  5     5   131       7.55          12      8.91  -24.0   24.3   1.98     0     0
##  6     6   131       7.55          20      8.16  -28.7   21.9   2.01     0     0
##  7     7   131       7.64          23      7.61  -28.6   24.8   2.17     0     0
##  8     8   130       7.69          25      7.91  -30.9   22.1   2.14     0     0
##  9     9   130       7.62          24      8.27  -29.7   19.4   2.07     0     0
## 10    10   130       7.70          30      7.13  -33.3   27.1   2.01     0     0
## 11    11   130       7.63          25      7.97  -27.3   23.3   2.12     0     0
## 12    12   130       7.75          36      7.03  -29.7   26.8   2.16     0     0
## 13    13   130       7.73          28      8.23  -26.1   30.4   2.05     0     0
## 14    14   130       7.77          36      7.08  -28.5   28.5   2.19     0     0
## 15    15   130       7.85          37      7.21  -32.1   28.1   2.23     0     0
## 16    16   130       8.16          47      5.78  -31.0   36.7   2.16     0     0
```

```
## 17    17   130       8.18          44      5.78  -30.8   26.8   2.25     0     0
## 18    18   130       8.22          41      7.71  -27.3   31.1   2.28     0     0
## 19    19   130       8.20          39      8.67  -32.0   33.8   2.38     0     0
## 20    20   130       7.83          56      5.50  -29.4   44.4   2.46     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

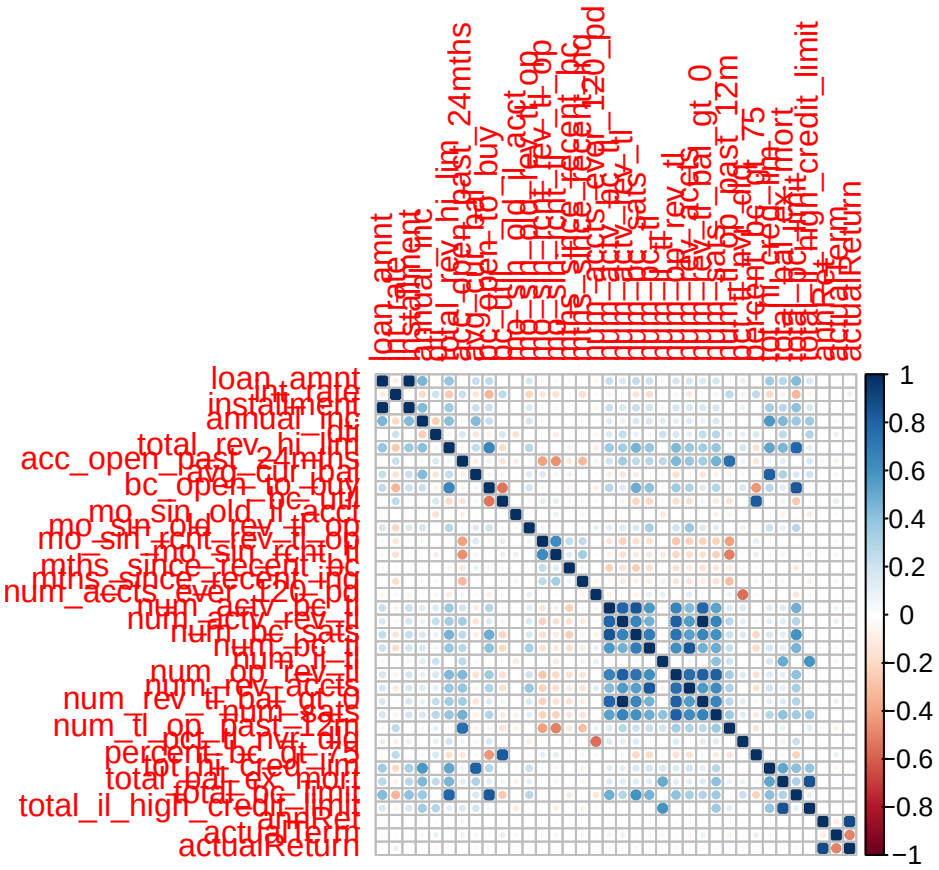
Bonus points

Correlations

```
library(corrplot)
```

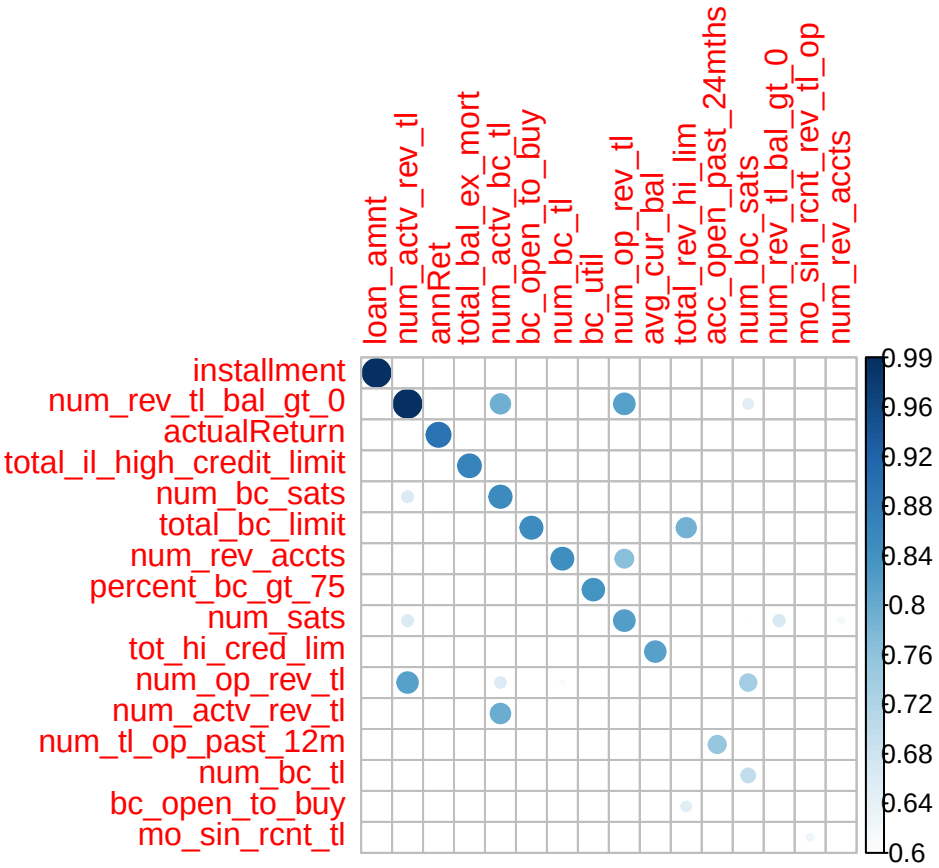
```
## corrplot 0.84 loaded
```

```
xCorr <- lcdfTrn_AR %>% select_if(is.numeric) %>%cor()

corrplot(xCorr, method="circle")
```




```
#Setting threshold to 0.6
corrTH = 0.6
xCorr[upper.tri(xCorr, diag=TRUE)] <- NA

#Creating data fame
xCorr <- as.data.frame(as.table(xCorr))
#Remove rows corresponding to NA values
xCorr <- na.omit(xCorr)
#remove the rows with abs(values) < corrTH
```

```
xCorr_th <- xCorr %>% filter(abs(Freq) > corrTH)
#order by the corr values
xCorr_th <- xCorr_th[order(-abs(xCorr_th$Freq)),]
#Convert back to matrix form to use with corrPlot
xCorrMat <- xCorr_th %>% pivot_wider(names_from = Var2, values_from = Freq)
#convert first column to rownames
xCorrMat<-column_to_rownames(xCorrMat, var="Var1")

corrplot(as.matrix(xCorrMat), is.corr=FALSE, na.label=" ", method="circle")
```



#AUC

```
library(pROC)

#Or considering both numeric and factor variables:
aucAR<- sapply(lcdfTrn_AR %>% mutate_if(is.factor, as.numeric) %>% select_if(is.numeric), auc, response=

#TO determine which variables have auc > 0.5
aucAR[aucAR>0.5]
```

```
##                  loan_amnt                   int_rate
##                  0.6666667                  1.0000000
##                      grade                  sub_grade
##                  1.0000000                  1.0000000
##                 emp_length                 annual_inc
##                  0.7500000                  0.6666667
##        verification_status                    purpose
```

```
##                  0.8333333                  1.0000000
##                        dti           earliest_cr_line
##                  0.6666667                  0.6666667
##        initial_list_status           total_rev_hi_lim
##                  0.9166667                  0.6666667
##       acc_open_past_24mths                avg_cur_bal
##                  1.0000000                  0.6666667
##             bc_open_to_buy                     bc_util
##                  1.0000000                  1.0000000
##         mo_sin_old_il_acct       mo_sin_old_rev_tl_op
##                  0.6666667                  0.6666667
##      mo_sin_rcnt_rev_tl_op             mo_sin_rcnt_tl
##                  0.8333333                  1.0000000
##       mths_since_recent_bc             num_actv_bc_tl
##                  0.8333333                  0.8333333
##            num_actv_rev_tl                num_bc_sats
##                  0.8333333                  0.6666667
##                  num_bc_tl                  num_il_tl
##                  0.7500000                  0.5833333
##              num_rev_accts        num_rev_tl_bal_gt_0
##                  0.8333333                  0.8333333
##                   num_sats         num_tl_op_past_12m
##                  0.7500000                  1.0000000
##             pct_tl_nvr_dlq           percent_bc_gt_75
##                  0.6666667                  1.0000000
##          total_bal_ex_mort             total_bc_limit
##                  0.8333333                  0.8333333
## total_il_high_credit_limit                     annRet
##                  0.6666667                  1.0000000
##                actualReturn
##                  1.0000000
```

```
#Or, we can use the tidy(..) function from the broom package - which converts the 'messy' output into a
library(broom)

tidy(aucAR[aucAR > 0.5]) %>% view()

# or  in any range of values like, tidy(aucAll[aucAll >=0.5 & aucAll < 0.6])
# or in sorted order
tidy(aucAR) %>% arrange(desc(aucAR))
```

```
## # A tibble: 45 x 2
##    names                    x
##    <chr>                <dbl>
##  1 int_rate                 1
##  2 grade                    1
##  3 sub_grade                1
##  4 purpose                  1
##  5 acc_open_past_24mths     1
##  6 bc_open_to_buy           1
##  7 bc_util                  1
##  8 mo_sin_rcnt_tl           1
##  9 num_tl_op_past_12m       1
## 10 percent_bc_gt_75         1
## # ... with 35 more rows
```

#Removing variables based on auc score

```
lcdf_AR_auc <- lcdfTrn_AR %>% select(-c(int_rate, grade, sub_grade, purpose, acc_open_past_24mths, bc_o
```

#Building A Random Forest with removed auc

```
rf_auc <- ranger(actualReturn ~., data=subset(lcdf_AR_auc, select=-c(actualTerm, loan_status)), num.tre
```

#Rf AUC – Training data

```
rfPredRet_trn_auc<- predict(rf_auc,  lcdf_AR_auc)
sqrt(mean((rfPredRet_trn_auc$predictions- lcdf_AR_auc$actualReturn)^2))
```

```
## [1] 3.801919
```

```
plot ((predict(rf_auc,  lcdf_AR_auc))$predictions, lcdf_AR_auc$actualReturn)
```