# IDS 572 Assignment 2

# Models for investment decisions in Lending Club loans

Submitted by –

Dhananjay Singh (668437546)

Srinanda Kurapati (663244158)

Sunny Patel (676645654)

**Q1. (a) Develop boosted tree models (using either gbm or xgBoost) to predict loan_status. Experiment with different parameters using a grid of parameter values. Use cross-validation. Explain the rationale for your experimentation. How does performance vary with parameters, and which parameter setting you use for the 'best' model.**

**Model performance should be evaluated through use of same set of criteria as for the earlier models - confusion matrix based, ROC analyses and AUC, cost-based performance.**

**Provide a table with comparative evaluation of all the best models from each methods; show their ROC curves in a combined plot. Also provide profit-curves and 'best' profit' and associated cutoff. At this cutoff, what are the accuracy values for the different models?**

We have used xgBoost as a boosting tree model to predict loan_status. We have experimented with different parameter values such as max_depth, eta, lambda, subsample, colsample_bytree. We have also used cross validation with different values of Nrounds, and nfold to make sure our model is not overfitting, and to also get a good estimate of how our model is performing(low variance) on unseen or test data.

Parameter description and how we experimented with them –

- Eta – This is also known as learning rate. We get the weights of features after each boosting round and eta is sued to shrink the weights of the features. This step size shrinkage is used to make the model more robust to noise(overfitting). We have experimented with different values of eta – 0.1, 0.01, 0.001, 0.5. We had to increase the number of rounds for lower values of eta.
- Max_depth – This defines the maximum depth of the tree. Increasing this value can lead to overfitting since it would result in a more complex model. Large data sets require more depth. So we need to select adequate depth in our model to control the model complexity. We experimented with values of max_depth – 6, 8, 10.
- Nrounds – This is defined as the number of iterations our model runs. We fine tuned this parameter using cross validation to find the best iteration.
- Lambda – This is the L2 regularization term whose default value is 1. We just experimented with the value as 0.05 in one of our models.
- Subsample – This defines the ratio of the training set supplied to the model. We experimented with the value of subsample by setting it as 0.7.
- Colsample_bytree – This ration defines the number of columns(features) provided to the model during tree construction. We experimented with the values of 0.6, and 0.7.

We also fine tuned our parameters using a parameter grid to find the best combination of paramters to train our model with.

```
xgbParamGrid <- expand.grid( max_depth = c(2, 6, 10),

eta = c(0.001, 0.01, 0.1),subsample = c(0.5,0.7),colsample_bytree=c(0.5,0.7))
```

The resulting combination which gave us best AUC on training set was as follows –

Eta = 0.1, subsample = 0.7, colsample_bytree = 0.7, max_depth = 10.

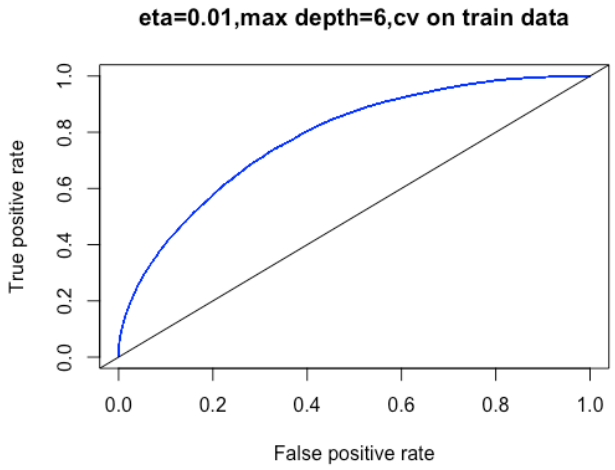
**Comparative Analysis of xgboost models with different parameter values**

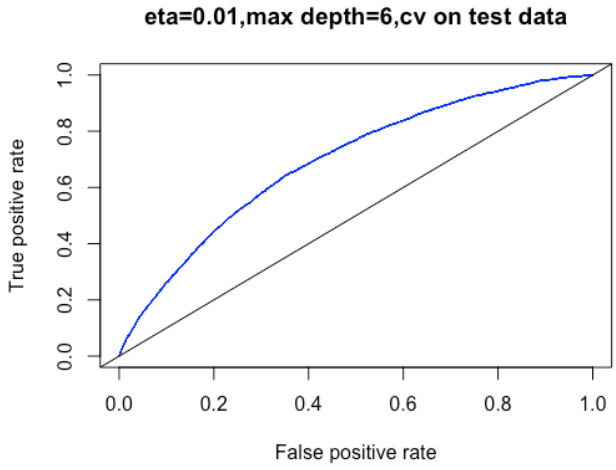
| Models | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Parameters | Max_depth = 6<br>Eta = 0.01<br>Nrounds = 500<br>early_stoppings=10<br>nfold = 5 | Max_depth = 8<br>Eta = 0.5<br>Nrounds = 500<br>early_stoppings=10<br>nfold = 5 | Max_depth = 6<br>Eta = 0.1<br>Nrounds = 800<br>early_stoppings=10<br>subsample = 0.7<br>nfold = 5 | Max_depth = 8<br>Eta = 0.01<br>Nrounds = 1000<br>early_stoppings=10<br>lambda=0.05,<br>subsample = 0.7<br>nfold = 10 | Max_depth = 10<br>Eta = 0.001<br>Nrounds = 1000<br>early_stoppings=10.<br>subsample=0.7,<br>colsample_bytree=0.6<br>nfold = 5 | Max_depth = 10<br>Eta = 0.01<br>Nrounds = 200<br>early_stoppings=10.<br>subsample=0.7,<br>colsample_bytree=0.7<br>nfold = 10 |
| Train auc | 0.72811 | 0.761139 | 0.739077 | 0.784506 | 0.790116 | 0.828695 |
| Test auc | 0.688637 | 0.67562 | 0.692133 | 0.69071 | 0.691006 | 0.691245 |
| Train auc after cv | 0.7808171 | 0.7611378 | 0.7278388 | 0.8701403 | 0.719812 | 0.8938674 |
| Test auc after cv | 0.6932381 | 0.6756216 | 0.6806303 | 0.6925267 | 0.662334 | 0.6860607 |
| Top 5 variables by importance | Int_rate<br>Dti<br>Installment<br>Acc_open_past_24mnths<br>annual_inc | Int_rate<br>Dti<br>Installment<br>annual_inc<br>Grade B | Int_rate<br>Acc_open_past_24mnths<br>Dti<br>annual_inc<br>Grade B | Int_rate<br>Dti<br>Installment<br>annual_inc<br>Mnths_since_old_rev_tl_op | Int_rate<br>Dti<br>Acc_open_past_24mnths<br>Acg_cur_bal<br>Bc_util | Int_rate<br>Dti<br>annual_inc<br>Installment<br>tot_hi_cred_lim |

**The best model gave us an AUC value of 0.6932381 on the test data set. The parameters used in this model were – eta = 0.01, max_depth = 6, using 5-fold cross validation.**

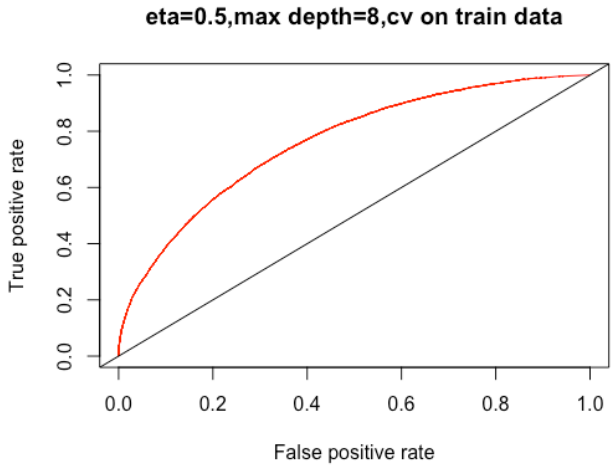
**ROC curves for all the 6 models –**

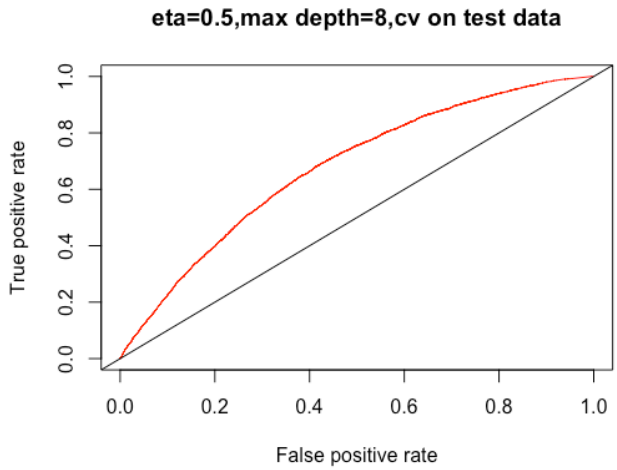
Model 1 - Max_depth = 6, Eta = 0.01, Nrounds = 500, early_stoppings=10, nfold = 5



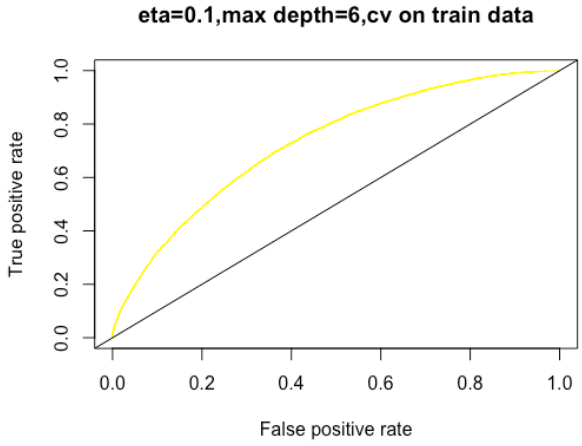



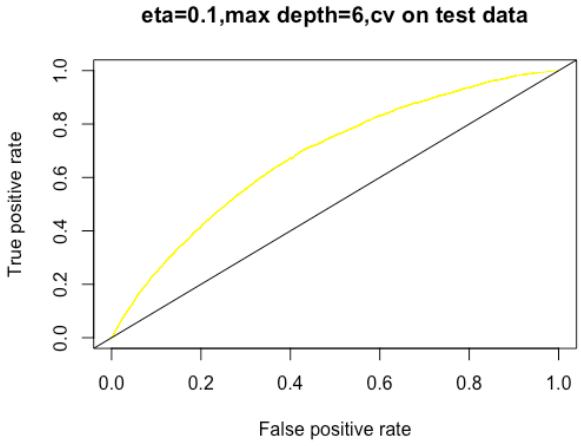

Model 2 - Max_depth = 8, Eta = 0.5, Nrounds = 500, early_stoppings=10, nfold = 5



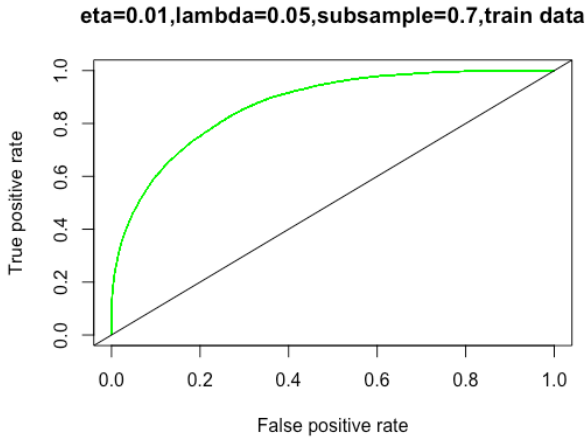



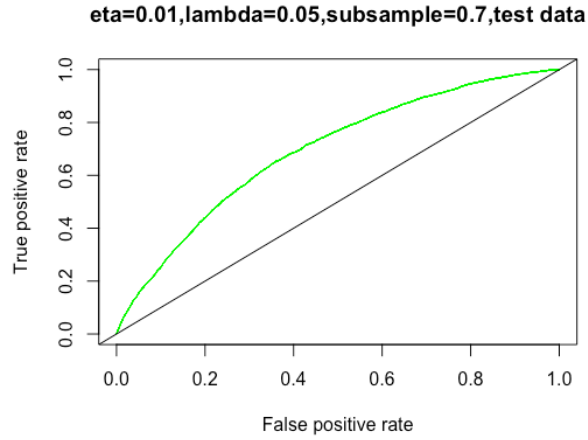
Model 3 - Max_depth = 6, Eta = 0.1, Nrounds = 800, subsample = 0.7, early_stoppings=10, nfold = 5



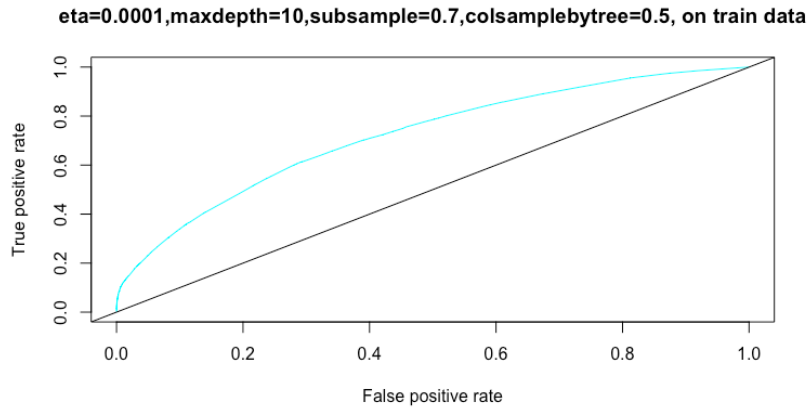



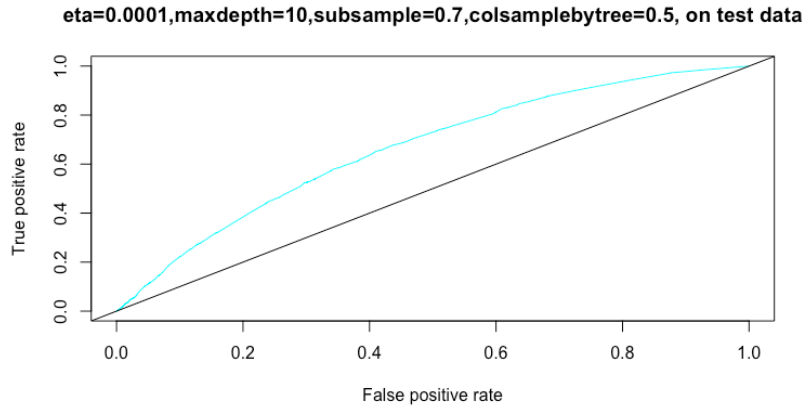

Model 4 - Max_depth = 8, Eta = 0.01, Nrounds = 1000, lambda=0.05, subsample = 0.7
nfold = 10, early_stoppings=10



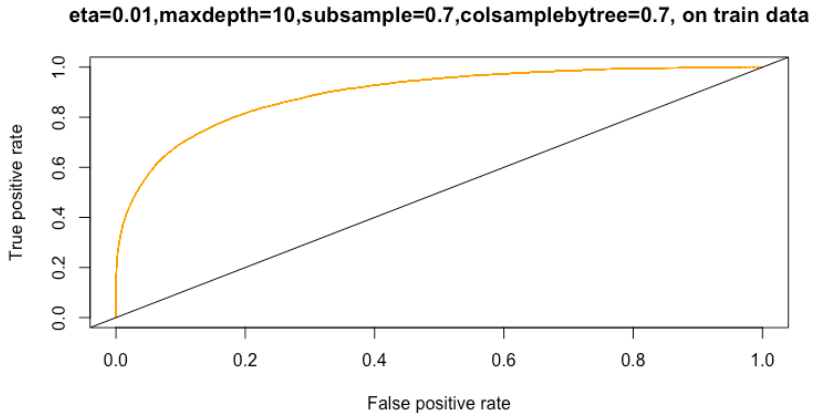



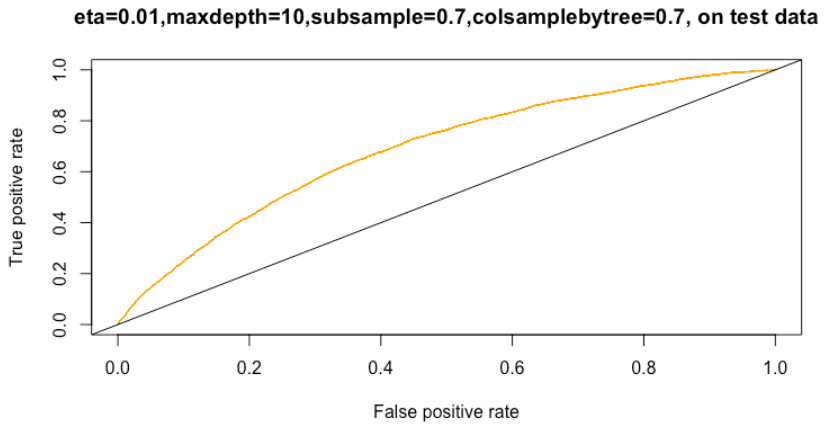
Model 5 - Max_depth = 10, Eta = 0.001, Nrounds = 1000, subsample=0.7, colsample_bytree=0.6 nfold = 5, early_stoppings=10



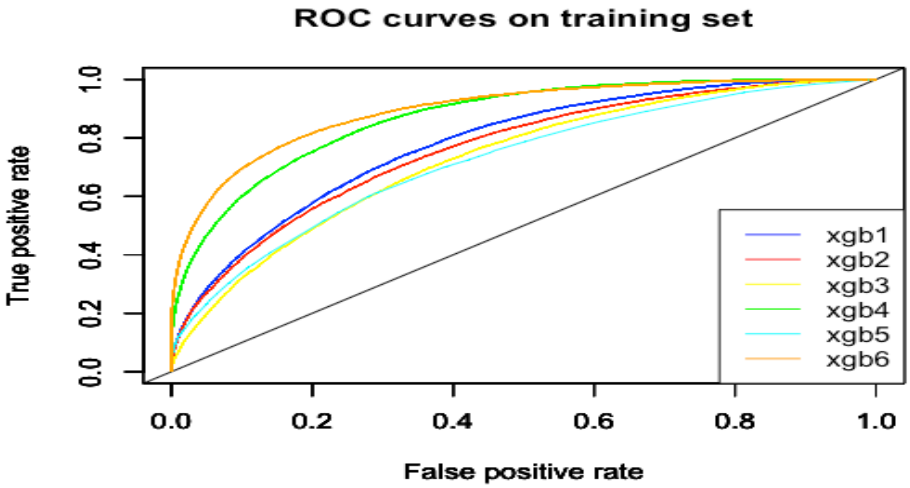



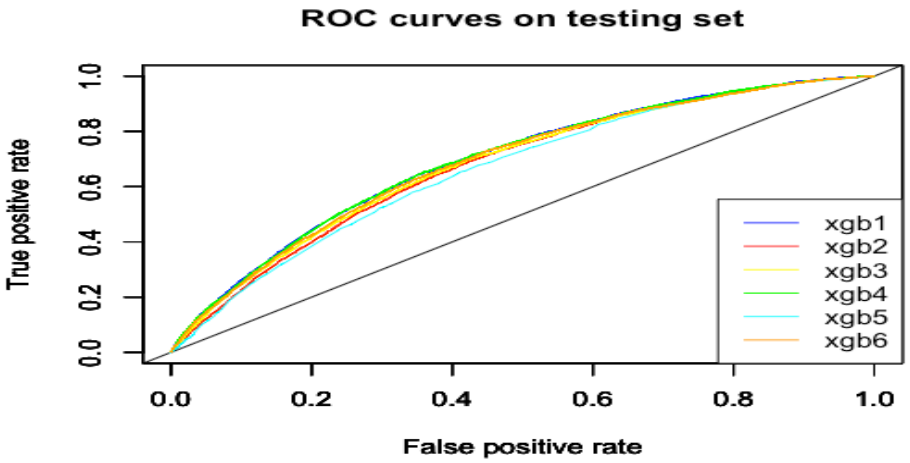

Model 6 - Max_depth = 10, Eta = 0.01, Nrounds = 200, subsample=0.7, colsample_bytree=0.7 nfold = 10, early_stoppings=10






**ROC curves of all models on the same plot –**

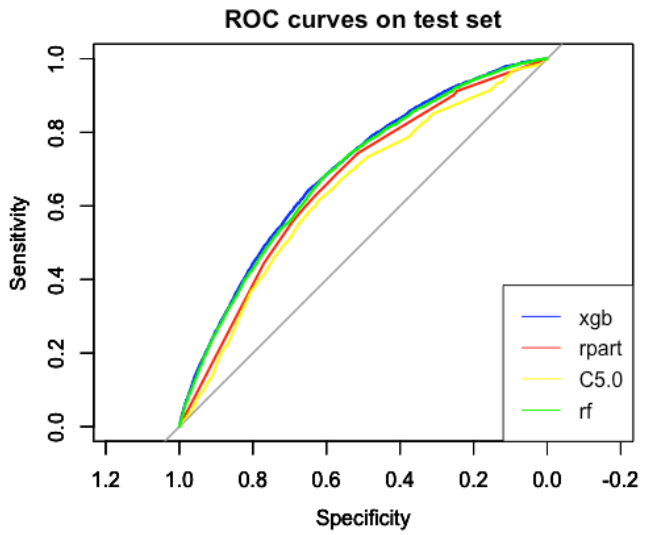


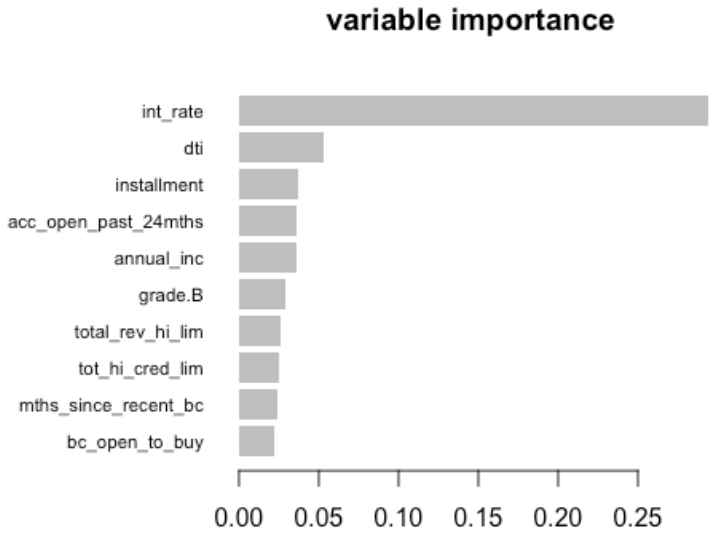
We also did a comparative analysis among rpart, C5.0, rf, and xgBoost models to predict loan_status.

| Models | rpart | C5.0 | rf | xgb |
|---|---|---|---|---|
| AUC on test set | 0.6629409 | 0.6389534 | 0.687 | 0.6932381 |

ROC curves of these best models on same plot –



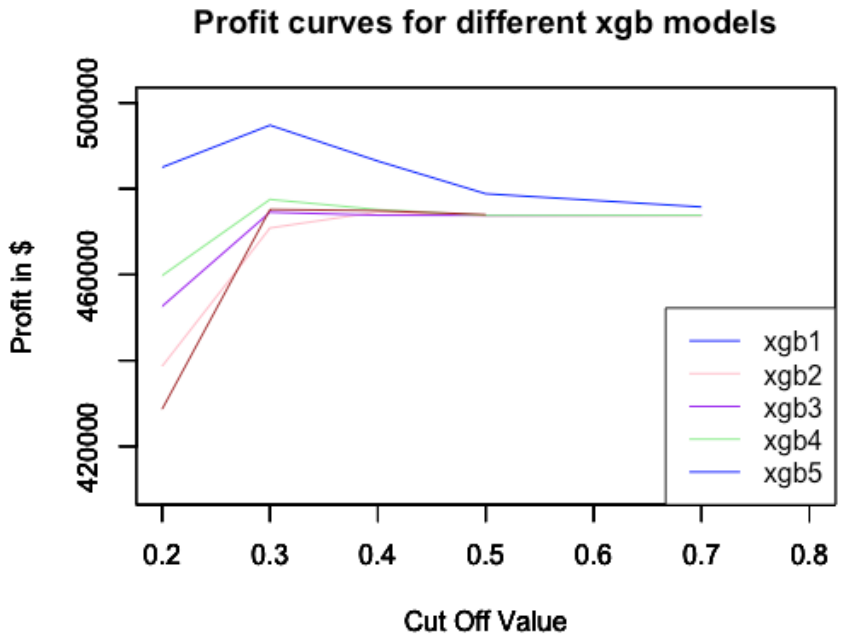

**Plot of Variable importance for the best xgBoost model –**




We did cut off analysis on the different xgBoost models using values – 0.2, 0.3, 0.4, 0.5, 0.6.

We generated the confusion matrix for all models based on these cut off values.
Based on the confusion matrix, we created a profit matrix – Profit of $24 for predicting a fully paid loan correctly, and loss of $35 for prediciting a charged off loan as fully paid. We assumed of investing in a small safe CD to get a profit of $6 on those loans predicted as charged off.
We multiplied the profit matrix with the confusion matrix at different threshold values to get the best profit value.

We got the highest profit for the cut off value = 0.3
The highest profit we got was $494796.3
The Profit Curves of the different xgBoost models on the same plot –




| Accuracy of xgBoost models for the cut off value of 0.3 which gave best profit | | | | | |
|---|---|---|---|---|---|
| **Models** | **Model 1** | **Model 2** | **Model 3** | **Model 4** | **Model 5** |
| **Accuracy** | 84.84% | 83.18% | 84.37% | 84.69% | 83.53% |

**Q2. (a) Develop linear (glm) models to predict loan_status. Experiment with different parameter values, and identify which gives 'best' performance. Use cross-validation. Describe how you determine 'best' performance.**
**How do you handle variable selection?**

**Experiment with Ridge and Lasso, and show how you vary these parameters, and what performance is observed.**

We used cv.glmnet function to train our logistic regression models to predict loan_status. We used family = "binomial" because our dependent variable has two responses. We experiment with different values of alpha which is an elastic net mixing parameter. We did this using cross validation to minimize overfitting.

Alpha = 1 is lasso regression which uses L1 norm to estimate sparse coefficients. This type of regression sets coefficients to zero and eliminates non useful variables.

Alpha = 0 is ridge regression which uses L2 regularization. This model retains all the variables and shrinks the coefficients of corelated variables.

The elastic net mixes these two kinds of regularization and so, we have created different models when predicting loan_status by experimenting with different alpha values like 1, 0, 0.2, 0.5, 0.8.

We have also created one extra model using weights parameter to model on a more balanced data set.

**Comparative analysis of glmnet models for different parameter values**

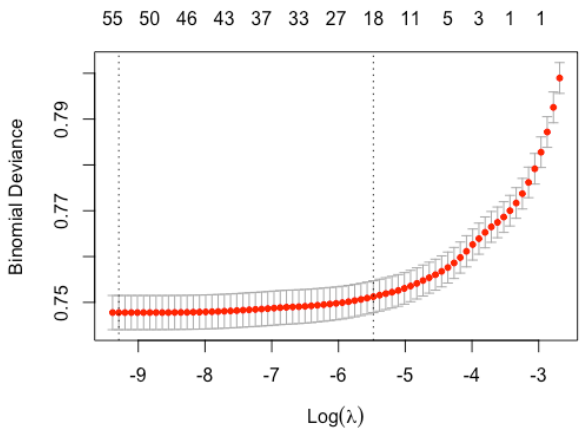
| Model | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Parameters | Family=. binomial Alpha=1 Lasso regression | Family=. binomial Alpha=0 Ridge regression | Family=. binomial Alpha=0.2 | Family=. binomial Alpha=0.5 | Family=. binomial Alpha=0.8 | Family=. binomial Alpha=1 Weights=wts Weighted model |
| AUC train | 0.6882238 | 0.6865674 | 0.6881915 | 0.6881471 | 0.688085 | 0.68818 |
| AUC test | 0.6931996 | 0.691595 | 0.6931424 | 0.6931602 | 0.6931382 | 0.6929799 |

**Best model – Model with parameters family=binomial, and alpha=1(lasso regression) gave us the best AUC of 0.6931996 on test data.**

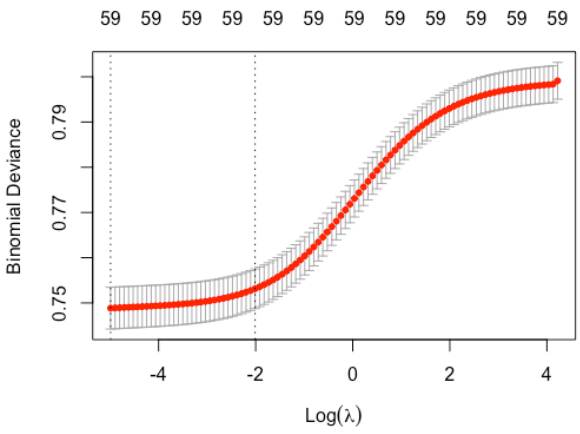
**Plot of Mean Squared Error Vs Log Lambda for all models –**

This plots displays the cross validated mean squared error as a function of log lambda. The vertical dotted lines show us the two values of lambda – lambda.min which gives us minimum cross validation error, and lamda.1se which cross validated error within 1 standard deviation of minimum error. The numbers on the top indicate the number of variables the model is using.

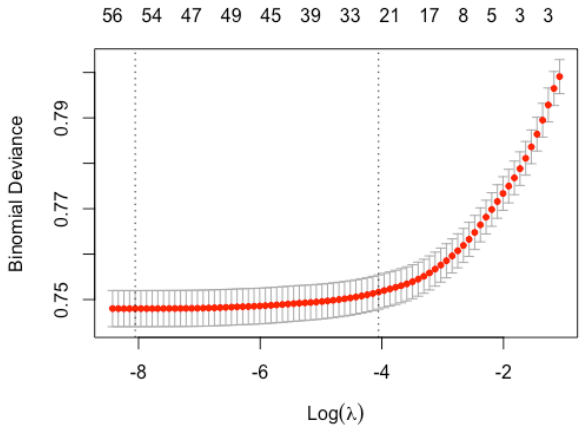
Model 1 –



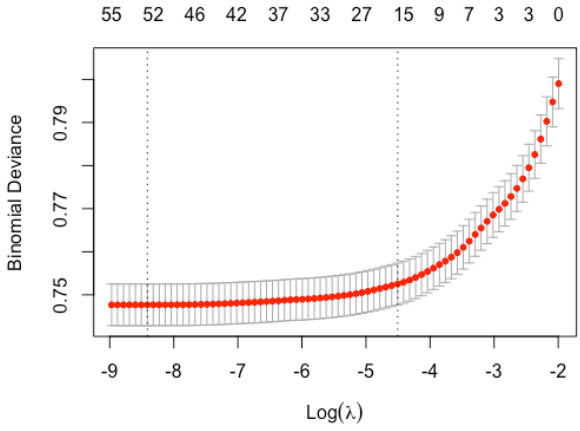

Model 2 -



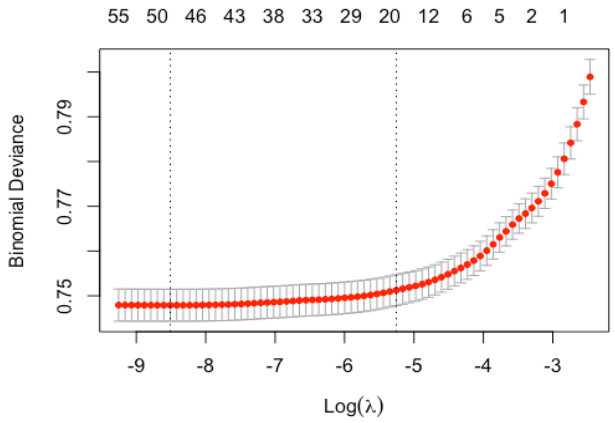

Model 3 -



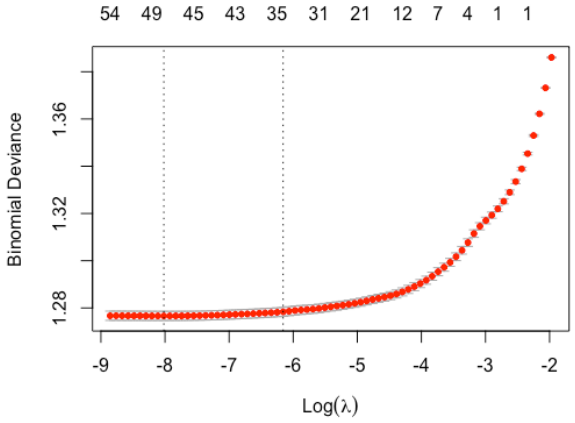

Model 4 -



Model 5 -



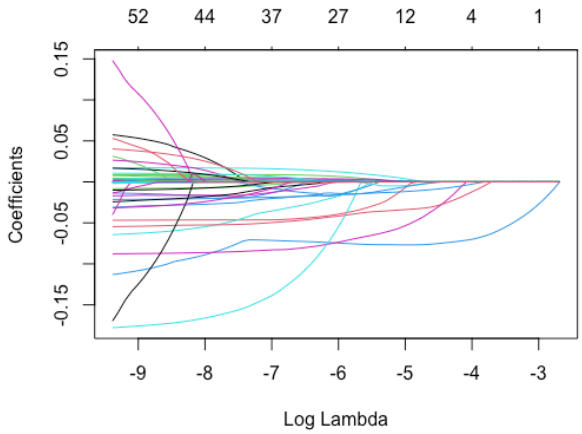

Model 6 –



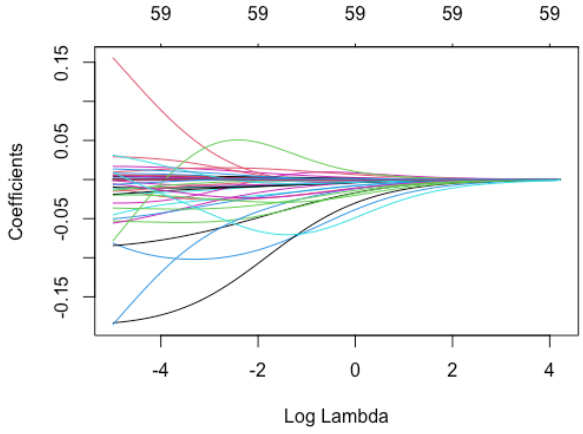

**Plot of Coefficients scores Vs Log Lambda for all models –**

This plot indicates the shrinkage of variables for larger values of log lambda.

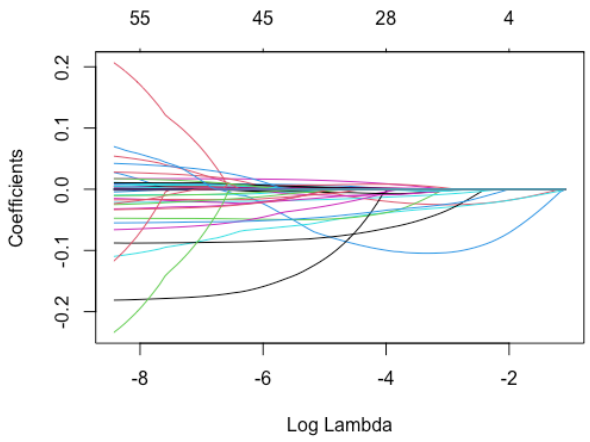
Model 1 -



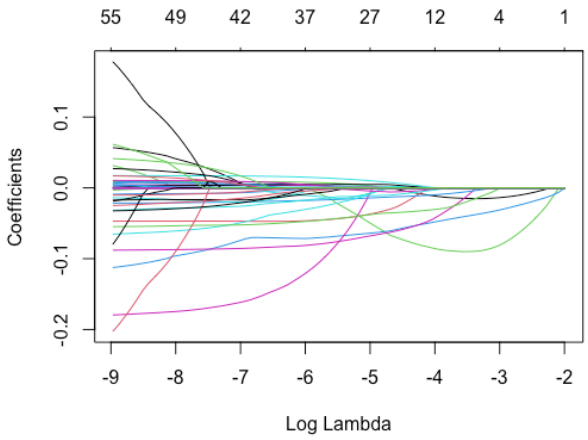

Model 2 –



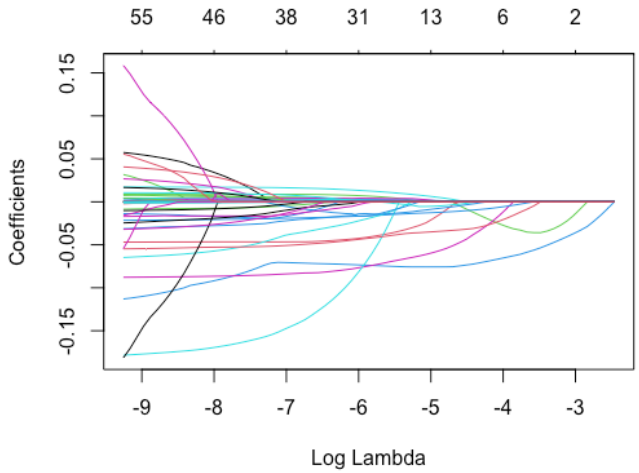
Model 3 -




Model 4 –



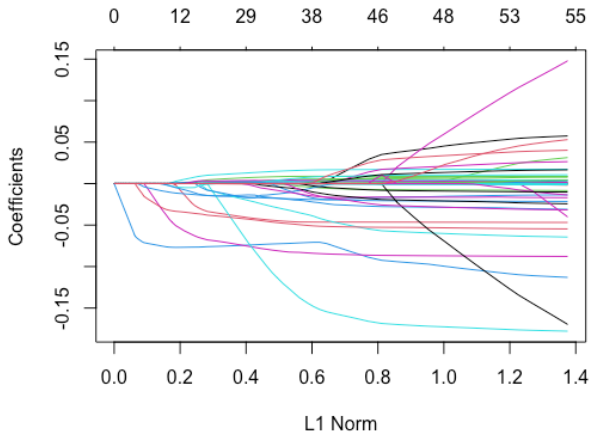

Model 5 -



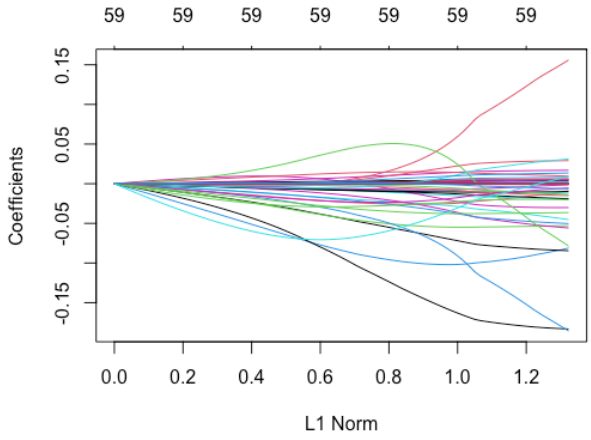

**Plot of Coefficients scores Vs L1 norm for all models –**

The curves above the axis indicate positive coefficients. This graph shows the path of coefficients against the L1 norm of the entire coefficient vector.

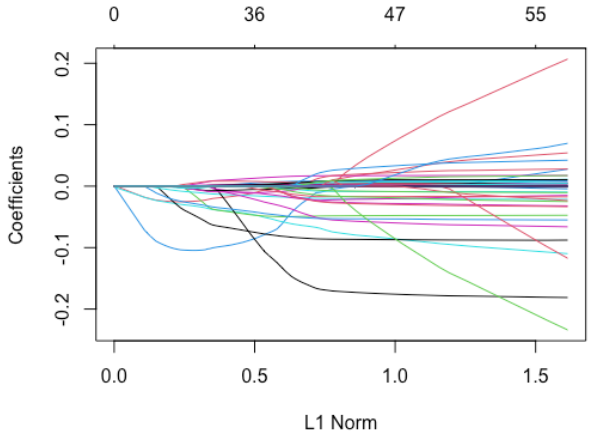
Model 1 -



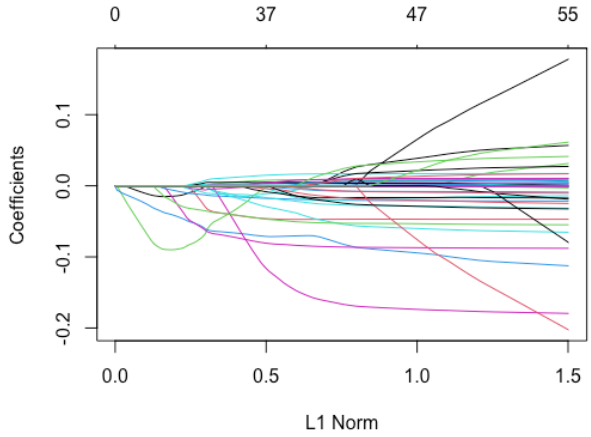

Model 2 –



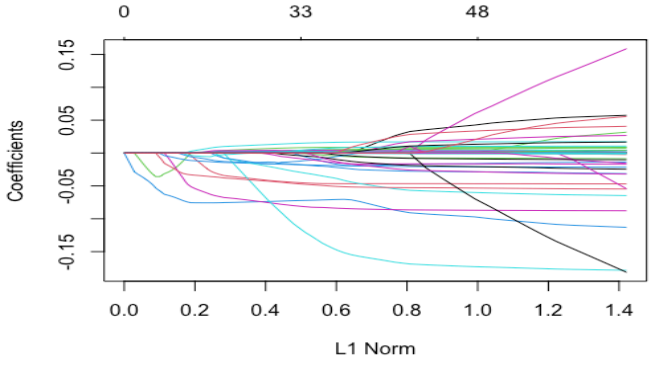

Model 3 -




Model 4 –



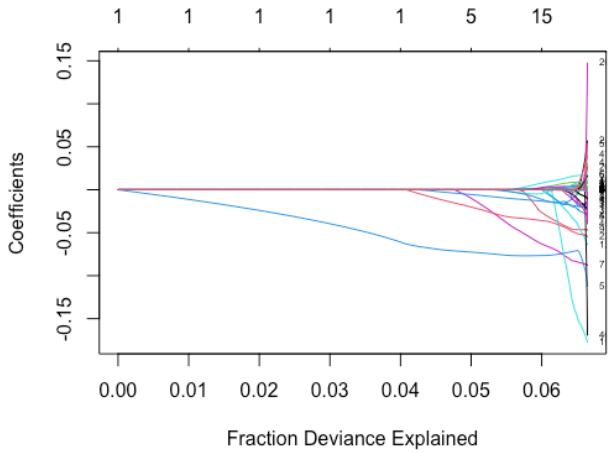

Model 5 –



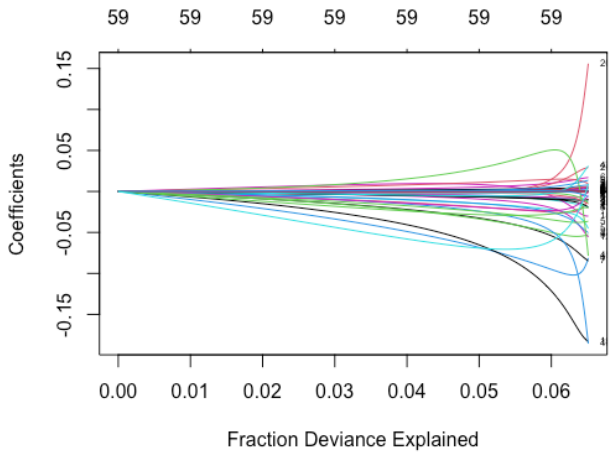
**Plot of Coefficients scores Vs Fraction Deviance Explained for all models –**

The colored lines in this graph indicate the variables and their coefficient scores.

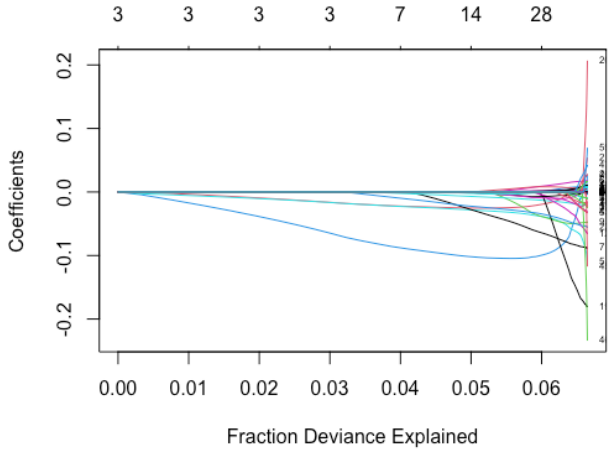
Model 1 -



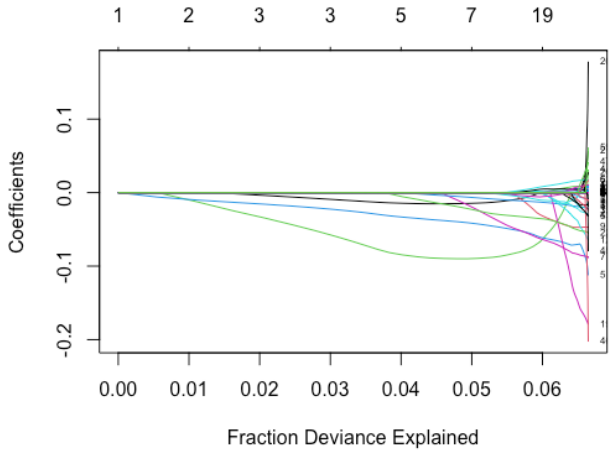

Model 2 –




Model 3 -

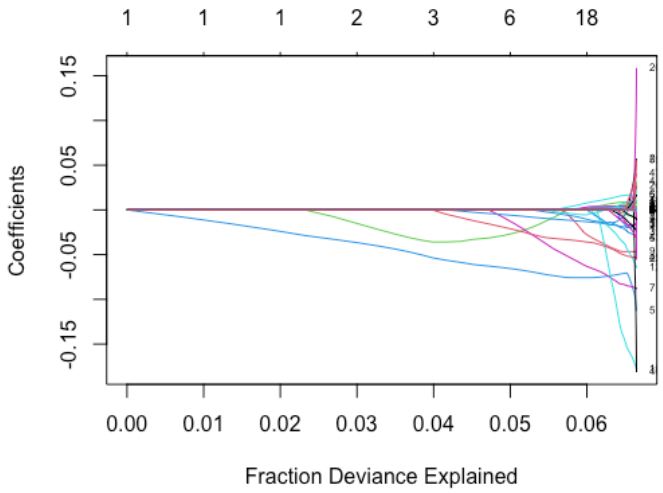



Model 4 –




*For the model with weights we only generated the Plot of Mean Squared Error Vs Log Lambda.

Model 5 –




**When it comes to selecting predictors, there are two primary ways of doing it:**

**1) Forward stepwise selection:**

This is the way of selection in which it begins with an empty model and adds variables one by one. In each step you add a variable that improves your model's performance. This begins with a model that contains no variables and this is called a null model. Then we start adding the significant variables one by one. This is done until an earlier condition about variables is reached / until all variables under consideration have been added.

**2) Backward stepwise selection:**

This is also called backward elimination and this is a way of selecting variables that begins with a model that contains all variables under consideration. Then it starts removing less significant variables one by one until a pre-specified stopping rule is encountered or when no variable is left behind in the model

Source : https://quantifyinghealth.com/stepwise-selection/

**Q2. (b) For the linear model, what is the loss function, and link function you use ? (Write the expression for these, and briefly describe).**

**We have used the following link functions –**

1. link = logit, when family = Binomial
2. link = identity, when family = Gaussian

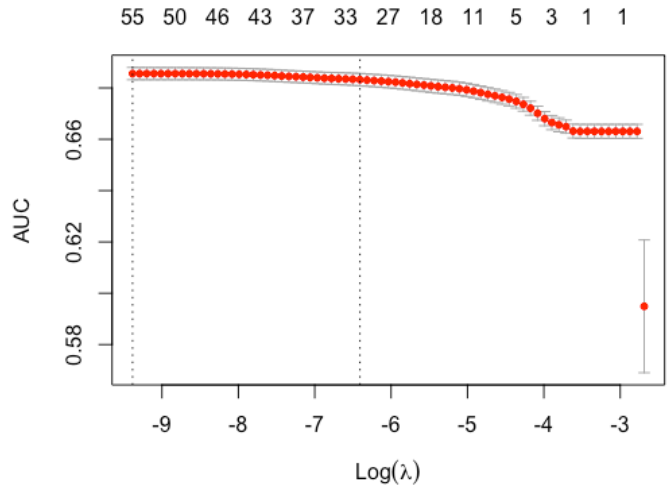
**We have used following loss functions –**

1. type.measure = deviance (logistic regression)
2. type.measure = auc (two-class logistic regression)
3. type.measure = class (binomial logistic regression)

We made 4 models experimenting with loss functions when predicting loan_status.

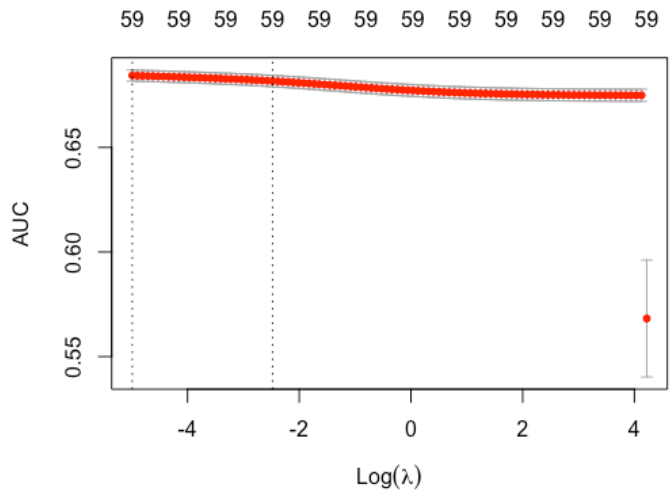
We used type.measure as auc, and class in our model and used did both lasso, as well as ridge regression.

**Model 1 -** type.measure = auc (two-class logistic regression), alpha = 1



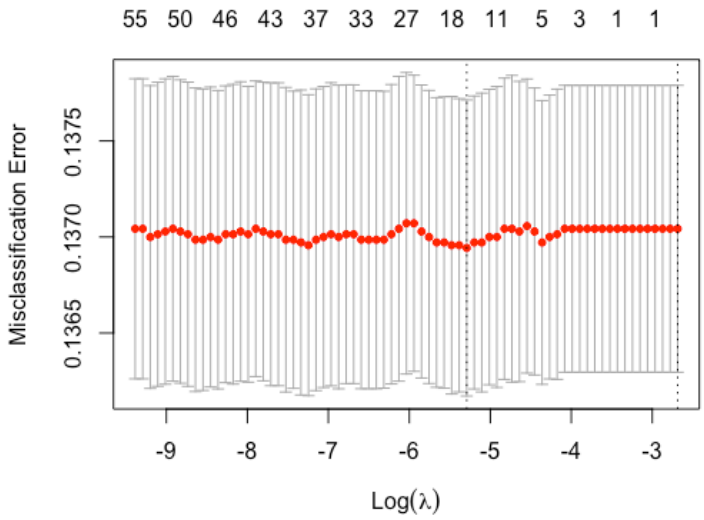

AUC value on training data set = 0.6882395
AUC value on testing data set = 0.693201

**Model 2 -** type.measure = auc (two-class logistic regression), alpha = 0



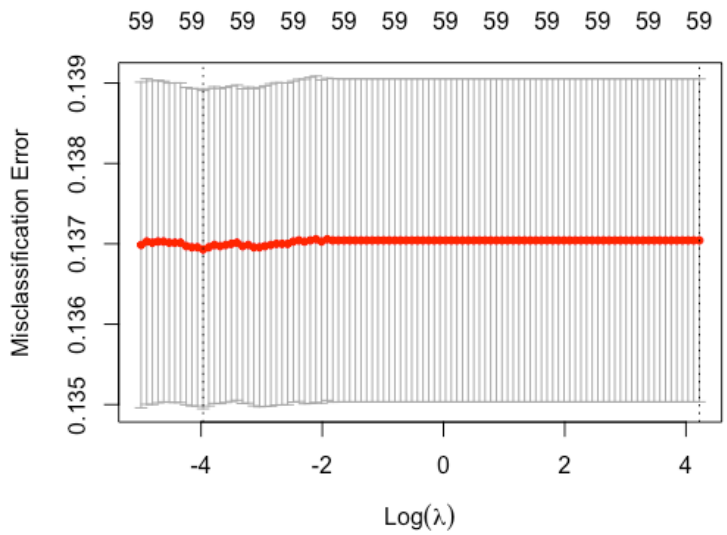

AUC value on training data set = 0.6865674
AUC value on testing data set = 0.691595

**Model 3 -** type.measure = class (binomial logistic regression), alpha = 1




AUC value on training data set = 0.6809814
AUC value on testing data set = 0.6892568

**Model 4 -** type.measure = class (binomial logistic regression), alpha = 0




AUC value on training data set = 0.6853718
AUC value on testing data set = 0.6906039

**Q2. (c) Compare performance of models with that of random forests (from last assignment) and gradient boosted tree models.**

We have used the best random forest model from last assignment to do a comparative analysis with xgBoost and linear models. There is not a very significant difference in the performance of these three models. Out of all three, the best model so far is xgBoost model with an AUC value of 0.6932381.

| Model | glm | rf | xgb |
|---|---|---|---|
| AUC values | 0.6931996 | 0.6872 | 0.6932381 |

**Q2. (d) Examine which variables are found to be important by the best models from the different methods, and comment on similarities, difference. What do you conclude?**

**We did a significance test to find out the most important variables in the linear(glm) models.**

| Variables with highest importance | | | | | |
|---|---|---|---|---|---|
| **glm model 1** | **glm model 2** | **glm model 3** | **glm model 4** | **glm model 5** | **glm model 6** |
| sub_grade | sub_grade | sub_grade | dti | dti | dti |
| dti | dti | dti | sub_grade | sub_grade | sub_grade |
| acc_open_past_24mths | acc_open_past_24mths | acc_open_past_24mths | acc_open_past_24mths | acc_open_past_24mths | acc_open_past_24mths |
| installment | installment | installment | installment | tot_hi_cred_lim | installment |
| emp_length | emp_length | tot_hi_cred_lim | tot_hi_cred_lim | installment | tot_hi_cred_lim |
| tot_hi_cred_lim | tot_hi_cred_lim | emp_length | emp_length | emp_length | home_ownership |
| home_ownership | total_bc_limit | home_ownership | home_ownership | home_ownership | emp_length |
| total_bc_limit | home_ownership | total_bc_limit | total_bc_limit | total_bc_limit | total_bc_limit |

**We had found out the most important variables in our xgBoost models using xgb.importance() function.**

Variable importance from our best xgBoost model is as follows –

| | Feature | Gain | Cover | Frequency |
|---|---|---|---|---|
| 1 | int_rate | 2.936933e-01 | 1.840466e-01 | 4.631262e-02 |
| 2 | dti | 5.255601e-02 | 6.668140e-02 | 5.955306e-02 |
| 3 | installment | 3.730509e-02 | 5.980312e-02 | 4.304588e-02 |
| 4 | acc_open_past_24mths | 3.615371e-02 | 6.774466e-02 | 2.974762e-02 |
| 5 | annual_inc | 3.556991e-02 | 5.228457e-02 | 5.244139e-02 |
| 6 | grade.B | 2.885707e-02 | 3.814263e-02 | 7.458587e-03 |
| 7 | total_rev_hi_lim | 2.633056e-02 | 2.821123e-02 | 3.894076e-02 |
| 8 | tot_hi_cred_lim | 2.457350e-02 | 2.658649e-02 | 3.813131e-02 |
| 9 | mths_since_recent_bc | 2.451823e-02 | 2.924781e-02 | 3.454656e-02 |
| 10 | bc_open_to_buy | 2.182387e-02 | 2.737146e-02 | 3.417074e-02 |

**Random forest model in the previous assignment gave the variables as important –**

int_rate, installment, sub_grade, annual_inc, openAccRatio, borrHistory.

The variables which are identified as important by all the three (glm, xgb, rf) models are dti, installment, and annual income.

**Similarities –**

xgBoost and glm model, both identified acc_open_past_24mths as significant.

Glm and rf model identified int_rate as the most important variable.

xgBoost and rf identified subgrade as one of the important variables.

**Differences –**

There are certain variables like openAccRatio, borrHistory, which are identified as important by rf model but not by either of glm or xgBoost models.

**Conclusion –**

Overall, the xgBoost, glm, and random forest models identify some common variables as important.

**Q2. (e) In developing models above, do you find larger training samples to give better models ? Do you find balancing the training data examples across classes to give better models ?**

We used the ovun.sample() function from ROSE library to obtain undersampled, oversampled, and both (under and over) sampled data sets. After balancing our data set, we created linear glm models and performed AUC evaluation to compare with unbalanced data set model.

**AUC value for over sampled data set = 0.6884384**
**AUC value for under sampled data set = 0.6898371**
**AUC value for both sampled data set = 0.6882363**

We conclude that balancing of the data set across classes did not have any effect on the accuracy of the models. All the three models had pretty much the same AUC value. These AUC values were also not different from the AUC value that we got for the unbalanced data model.

**Q3. Develop models to identify loans which provide the best returns. Explain how you define returns? Does it include Lending Club's service costs?**

**Develop glm, rf, gbm/xgb models for this. Show how you systematically experiment with different parameters to find the best models. Compare model performance – explain what performance criteria do you use, and why.**

Returns or Return on Investment (ROI) measures the performance of the investment. It demonstrates the profit as a percentage of the initial investment amount.

$$Returns\ or\ R = \frac{Final\ Amount - Initial\ Amount}{Initial\ Amount}$$

With increase in the final amount, our returns also increase. For the best returns, we expect maximum final amount.
In the case of Lending Club, when a buyer invests in a loan, the interest rate determines the final amount. Higher the interest rate on the Initial amount, higher the value of the final amount, thereby leading to higher returns.

Lending Club's service cost was 1% on all amounts the borrower's pays. Actual return does not include this service cost, and this may reduce the investor's actual return by 1%.

In our case, increasing interest rates also correspond to high occurrences of 'charged off' loans, whereas lower loans have fewer instances of charged off loans. This means that with increase in interest rates, the investor must get exposed to higher chances of risk. In simple words, it is high returns or no returns. Whereas, at lower interest rates, most of the loans are paid off. This would mean, moderate returns at moderate risk.

For this question, the predictive modelling methods we use are – Linear Regression, Random Forest, XgBoost. In the previous question, performance could be assessed by calculating accuracy – as it was a classification problem (predicting loan status). Since we are working on regression here, the performance must be analysed in terms of the error in prediction or how close the predicted values were to the actual/expected values. In this question, we will be analysing performance using two metrics:

1. **Root Mean Square Error (RMSE) –** This is an extension of Mean Square Error (MSE). The advantage of RMSE over MSE is that the final units are retained in the former while the latter has squared units which tends to cause ambiguity. RMSE tends to penalize large differences due to squaring of differences.
2. **Mean Absolute Error (MAE) –** This takes the difference between absolute values of actual and expected values, thereby treating large or small differences neutrally.

## 1) General Linear Model (GLM)

GLM on training data with Alpha = 0

```
# A tibble: 10 x 14
```

| | tile | count | avgpredRet | numDefaults | avgActRet | minRet | maxRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 7000 | 11.3 | 86 | 10.4 | -0.265 | 28.5 | 2.59 | 3 | 767 | 2308 | 2477 | 1090 | 319 |
| 2 | 2 | 7000 | 8.55 | 117 | 9.60 | -26.9 | 40.2 | 2.42 | 81 | 1685 | 2736 | 1968 | 485 | 43 |
| 3 | 3 | 7000 | 7.48 | 149 | 8.92 | -25.1 | 44.4 | 2.29 | 325 | 2369 | 2878 | 1227 | 175 | 23 |
| 4 | 4 | 7000 | 6.72 | 151 | 8.34 | -30.1 | 36.9 | 2.21 | 640 | 2954 | 2670 | 634 | 94 | 7 |
| 5 | 5 | 7000 | 6.10 | 222 | 7.69 | -27.2 | 31.3 | 2.15 | 1085 | 3351 | 2126 | 373 | 55 | 8 |
| 6 | 6 | 7000 | 5.53 | 253 | 7.11 | -29.8 | 28.8 | 2.06 | 1707 | 3443 | 1563 | 229 | 44 | 14 |
| 7 | 7 | 7000 | 4.95 | 312 | 6.34 | -26.5 | 30.1 | 2.02 | 2736 | 3102 | 951 | 163 | 41 | 3 |
| 8 | 8 | 7000 | 4.32 | 416 | 5.51 | -30.8 | 26.8 | 1.95 | 3885 | 2398 | 528 | 138 | 46 | 4 |
| 9 | 9 | 7000 | 3.34 | 1198 | 3.57 | -31.0 | 34.8 | 1.87 | 4495 | 1506 | 637 | 269 | 74 | 17 |
| 10 | 10 | 7000 | -5.96 | 6846 | -15.1 | -33.3 | 19.1 | 2.95 | 829 | 2083 | 2343 | 1269 | 395 | 73 |

GLM on testing data with Alpha = 0

```
# A tibble: 10 x 14
```

| | tile | count | avgpredRet | numDefaults | avgActRet | minRet | maxRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 3000 | 7.93 | 492 | 8.44 | -33.3 | 40.2 | 2.17 | 9 | 279 | 811 | 1145 | 545 | 192 |
| 2 | 2 | 3000 | 6.74 | 472 | 6.74 | -33.3 | 30.9 | 2.17 | 34 | 767 | 1145 | 840 | 204 | 8 |
| 3 | 3 | 3000 | 6.15 | 474 | 5.96 | -33.3 | 28.7 | 2.21 | 133 | 1023 | 1113 | 598 | 130 | 2 |
| 4 | 4 | 3000 | 5.72 | 456 | 5.43 | -32.1 | 32.9 | 2.22 | 277 | 1180 | 1048 | 402 | 92 | 1 |
| 5 | 5 | 3000 | 5.35 | 432 | 5.33 | -32.3 | 36.9 | 2.22 | 431 | 1188 | 1013 | 316 | 51 | 1 |
| 6 | 6 | 3000 | 4.99 | 409 | 4.80 | -32.2 | 29.5 | 2.24 | 665 | 1215 | 891 | 193 | 36 | 0 |
| 7 | 7 | 3000 | 4.64 | 370 | 4.52 | -33.3 | 22.6 | 2.29 | 910 | 1182 | 754 | 143 | 11 | 0 |
| 8 | 8 | 3000 | 4.27 | 371 | 4.12 | -32.2 | 24.9 | 2.30 | 1116 | 1232 | 584 | 63 | 5 | 0 |
| 9 | 9 | 3000 | 3.83 | 337 | 3.91 | -30.9 | 19.2 | 2.31 | 1434 | 1129 | 396 | 38 | 3 | 0 |
| 10 | 10 | 3000 | 3.04 | 333 | 3.34 | -31.3 | 23.7 | 2.36 | 1757 | 1024 | 206 | 13 | 0 | 0 |

GLM on training data with Alpha = 0.2

| | tile | count | avgpredRet | numDefaults | avgActRet | minRet | maxRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 7000 | 8.09 | 1187 | 8.21 | -32.2 | 44.4 | 2.18 | 15 | 886 | 1935 | 2550 | 1114 | 446 |
| 2 | 2 | 7000 | 6.82 | 1029 | 6.93 | -32.2 | 36.6 | 2.20 | 65 | 2075 | 2494 | 1825 | 496 | 41 |
| 3 | 3 | 7000 | 6.20 | 1069 | 6.24 | -33.3 | 40.8 | 2.20 | 257 | 2491 | 2495 | 1384 | 360 | 13 |
| 4 | 4 | 7000 | 5.74 | 1005 | 5.84 | -33.3 | 33.6 | 2.20 | 580 | 2612 | 2493 | 1096 | 217 | 2 |
| 5 | 5 | 7000 | 5.33 | 1003 | 5.21 | -32.3 | 34.8 | 2.24 | 1098 | 2682 | 2351 | 746 | 122 | 1 |
| 6 | 6 | 7000 | 4.96 | 934 | 4.81 | -32.2 | 31.3 | 2.24 | 1562 | 2768 | 2085 | 508 | 76 | 1 |
| 7 | 7 | 7000 | 4.60 | 917 | 4.39 | -32.2 | 29.6 | 2.28 | 2214 | 2613 | 1785 | 325 | 63 | 0 |
| 8 | 8 | 7000 | 4.22 | 819 | 4.22 | -33.3 | 24.6 | 2.28 | 2751 | 2604 | 1426 | 183 | 36 | 0 |
| 9 | 9 | 7000 | 3.76 | 855 | 3.65 | -32.3 | 26.8 | 2.33 | 3283 | 2544 | 1072 | 90 | 11 | 0 |
| 0 | 10 | 7000 | 2.95 | 821 | 3.20 | -32.3 | 22.2 | 2.35 | 3997 | 2413 | 548 | 35 | 7 | 0 |

GLM on testing data with Alpha = 0.2

```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3000       8.08         475      8.45  -33.3   40.2   2.18     8   357   861  1089   480   187
 2     2  3000       6.82         473      6.71  -32.2   33.5   2.18    29   874  1109   770   203    13
 3     3  3000       6.19         460      6.03  -33.3   30.9   2.21   130  1095  1038   609   125     1
 4     4  3000       5.74         456      5.48  -31.1   30.9   2.22   266  1168  1024   429   111     2
 5     5  3000       5.34         442      5.24  -32.3   32.9   2.22   450  1106  1005   364    75     0
 6     6  3000       4.96         414      4.84  -32.2   36.9   2.22   686  1170   891   211    41     1
 7     7  3000       4.60         351      4.67  -32.2   29.5   2.29   935  1115   772   150    28     0
 8     8  3000       4.22         388      4.03  -33.3   24.9   2.30  1110  1167   641    76     6     0
 9     9  3000       3.76         337      3.87  -30.9   17.3   2.30  1470  1090   395    38     7     0
10    10  3000       2.96         350      3.29  -31.3   23.7   2.37  1682  1077   225    15     1     0
```

GLM on Training data with Alpha = 0.5

```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  7000       8.09        1187      8.21  -32.2   44.4   2.18    15   886  1935  2550  1114   446
 2     2  7000       6.82        1029      6.93  -32.2   36.6   2.20    65  2075  2494  1825   496    41
 3     3  7000       6.20        1069      6.24  -33.3   40.8   2.20   257  2491  2495  1384   360    13
 4     4  7000       5.74        1005      5.84  -33.3   33.6   2.20   580  2612  2493  1096   217     2
 5     5  7000       5.33        1003      5.21  -32.3   34.8   2.24  1098  2682  2351   746   122     1
 6     6  7000       4.96         934      4.81  -32.2   31.3   2.24  1562  2768  2085   508    76     1
 7     7  7000       4.60         917      4.39  -32.2   29.6   2.28  2214  2613  1785   325    63     0
 8     8  7000       4.22         819      4.22  -33.3   24.6   2.28  2751  2604  1426   183    36     0
 9     9  7000       3.76         855      3.65  -32.3   26.8   2.33  3283  2544  1072    90    11     0
10    10  7000       2.95         821      3.20  -32.3   22.2   2.35  3997  2413   548    35     7     0
```

GLM on Testing data with Alpha = 0.5

```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3000       8.08         475      8.45  -33.3   40.2   2.18     8   357   861  1089   480   187
 2     2  3000       6.82         473      6.71  -32.2   33.5   2.18    29   874  1109   770   203    13
 3     3  3000       6.19         460      6.03  -33.3   30.9   2.21   130  1095  1038   609   125     1
 4     4  3000       5.74         456      5.48  -31.1   30.9   2.22   266  1168  1024   429   111     2
 5     5  3000       5.34         442      5.24  -32.3   32.9   2.22   450  1106  1005   364    75     0
 6     6  3000       4.96         414      4.84  -32.2   36.9   2.22   686  1170   891   211    41     1
 7     7  3000       4.60         351      4.67  -32.2   29.5   2.29   935  1115   772   150    28     0
 8     8  3000       4.22         388      4.03  -33.3   24.9   2.30  1110  1167   641    76     6     0
 9     9  3000       3.76         337      3.87  -30.9   17.3   2.30  1470  1090   395    38     7     0
10    10  3000       2.96         350      3.29  -31.3   23.7   2.37  1682  1077   225    15     1     0
```

GLM on training data with Alpha = 0.8

```
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  7000       8.07        1184      8.25  -32.2   44.4   2.18    14   867  1908  2577  1130   450
 2     2  7000       6.81        1040      6.89  -32.2   36.6   2.20    63  2024  2519  1844   507    39
 3     3  7000       6.20        1066      6.24  -33.3   40.8   2.20   252  2507  2504  1375   350    12
 4     4  7000       5.73        1017      5.81  -33.3   33.6   2.20   568  2626  2508  1085   212     1
 5     5  7000       5.33         995      5.24  -32.3   34.8   2.24  1080  2698  2369   734   118     1
 6     6  7000       4.96         930      4.82  -32.2   31.3   2.24  1544  2784  2085   511    75     1
 7     7  7000       4.61         920      4.37  -32.2   29.6   2.27  2232  2631  1763   312    62     0
 8     8  7000       4.23         813      4.23  -33.3   24.6   2.28  2763  2594  1430   182    31     0
 9     9  7000       3.77         862      3.63  -32.3   26.8   2.33  3283  2549  1068    90    10     0
10    10  7000       2.97         812      3.21  -32.3   19.6   2.35  4023  2408   530    32     7     0
```

GLM on testing data with Alpha = 0.8

```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3000       8.06         475      8.47  -33.3   40.2   2.18     8   347   853  1099   487   188
 2     2  3000       6.81         479      6.71  -32.2   33.5   2.18    27   862  1116   780   201    12
 3     3  3000       6.19         463      5.99  -33.3   30.9   2.21   125  1082  1061   604   125     1
 4     4  3000       5.74         451      5.50  -31.1   30.9   2.22   262  1177  1023   427   109     2
 5     5  3000       5.34         441      5.28  -32.3   36.9   2.22   446  1110  1007   360    76     1
 6     6  3000       4.97         410      4.84  -32.2   22.5   2.22   688  1175   893   206    38     0
 7     7  3000       4.60         356      4.64  -32.2   29.5   2.29   931  1134   756   150    29     0
 8     8  3000       4.23         380      4.07  -33.3   24.9   2.30  1111  1173   638    74     4     0
 9     9  3000       3.77         340      3.84  -30.9   17.3   2.30  1480  1082   395    36     7     0
10    10  3000       2.98         351      3.27  -31.3   23.7   2.37  1688  1077   219    15     1     0
>
```

GLM on training data with Alpha = 1

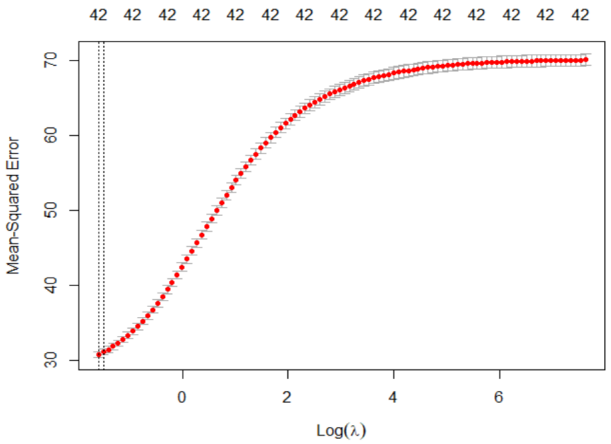
```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  7000       8.09        1146      8.34  -33.3   44.4   2.18    14   930  1989  2478  1078   458
 2     2  7000       6.83        1083      6.83  -32.2   38.1   2.18    62  2090  2531  1742   531    40
 3     3  7000       6.21        1005      6.34  -33.3   30.1   2.20   256  2595  2429  1380   329     8
 4     4  7000       5.74        1044      5.66  -32.2   33.6   2.21   555  2628  2498  1067   248     3
 5     5  7000       5.34        1011      5.15  -33.3   32.9   2.25  1138  2655  2266   784   156     1
 6     6  7000       4.97         979      4.76  -32.2   30.7   2.24  1629  2650  2070   557    94     0
 7     7  7000       4.61         884      4.49  -32.2   29.6   2.27  2208  2633  1732   347    80     0
 8     8  7000       4.23         813      4.24  -33.3   29.8   2.27  2782  2604  1408   183    23     0
 9     9  7000       3.79         807      3.76  -30.6   29.2   2.31  3388  2444  1036   110    22     0
10    10  7000       2.98         830      3.21  -32.3   19.0   2.37  3916  2484   554    37     9     0
```

GLM on testing data with Alpha = 1

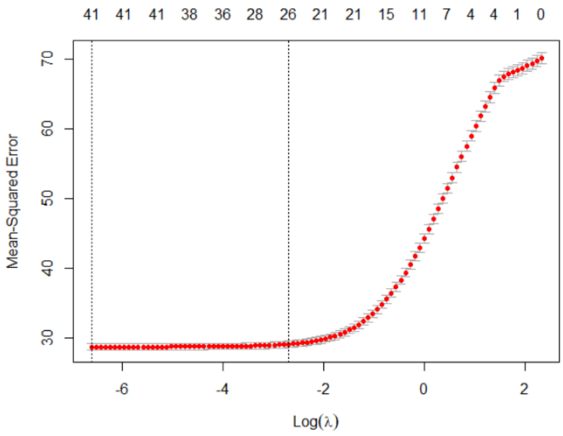
```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3000       8.05         482      8.32  -32.2   37.7   2.19     4   400   862  1130   415   171
 2     2  3000       6.81         474      6.53  -32.1   40.8   2.20    19   915  1070   777   202    16
 3     3  3000       6.20         452      6.23  -33.3   27.0   2.18    98  1104  1043   614   137     4
 4     4  3000       5.74         452      5.71  -33.3   36.9   2.21   240  1115  1082   466    92     5
 5     5  3000       5.35         403      5.44  -31.1   34.8   2.20   462  1152  1006   317    62     1
 6     6  3000       4.98         410      4.81  -32.3   31.3   2.24   696  1112   931   209    52     0
 7     7  3000       4.62         378      4.48  -30.0   22.6   2.25   937  1128   775   136    23     1
 8     8  3000       4.24         401      3.97  -33.3   23.8   2.31  1143  1079   654   101    23     0
 9     9  3000       3.80         360      3.72  -31.3   19.9   2.34  1416  1083   462    36     3     0
10    10  3000       2.99         371      3.21  -30.2   19.6   2.38  1625  1106   247    22     0     0
```

**Plot of MSE vs log(lambda) for each value of Alpha –**

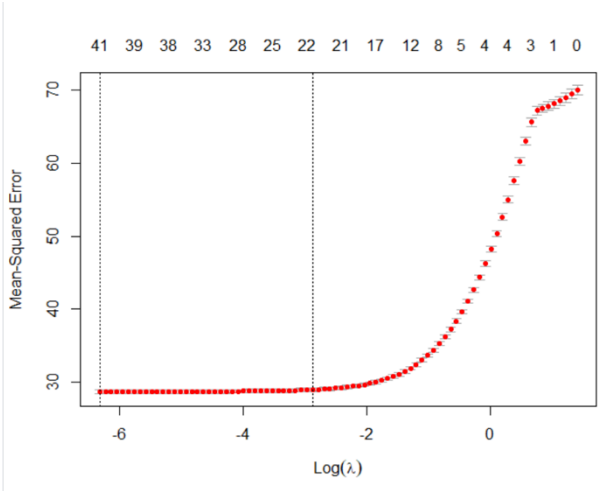
**Alpha = 0**



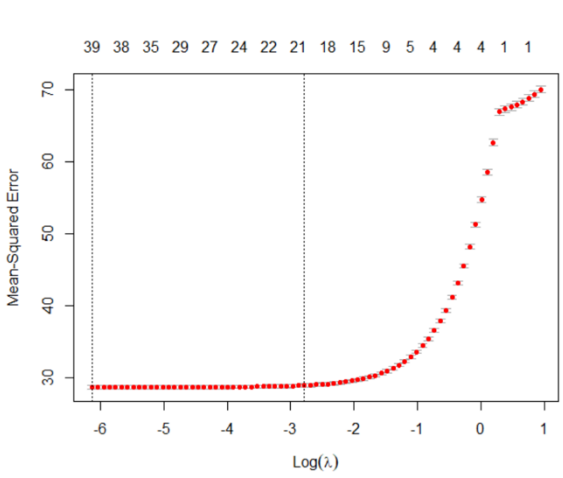

**Alpha = 0.2**



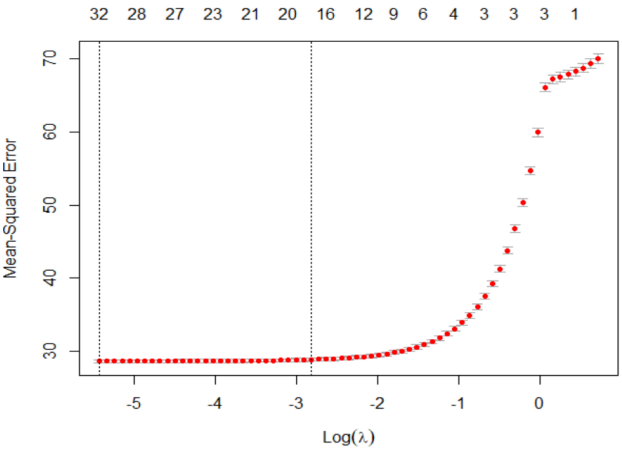

**Alpha = 0.5**




**Alpha = 0.8**




**Alpha = 1**



As one can observe, every plot of Mean Square Error vs Log(lambda) there are two dotted lines, the first corresponds to the value of lambda that gives minimum value of cross validated error, whereas the second dotted line corresponds to the value of lambda that has cross validated error within one standard error of the minimum

Here, alpha determines the value of weighing of the predictors we are using. Alpha = 0 corresponds to Ridge regression, while Alpha = 1 corresponds to Lasso regression. Alpha is the penalty parameter, and we see that as the value varies from 0 to 1, the mean square error for various values of lambda remains low for a long time until the predictor variable count comes to an average of 4. After which, with decrease in the number of predictors, the value of MSE rises for all values of alpha.

**Table of various values of Alpha, RMSE, MAE for the linear model –**

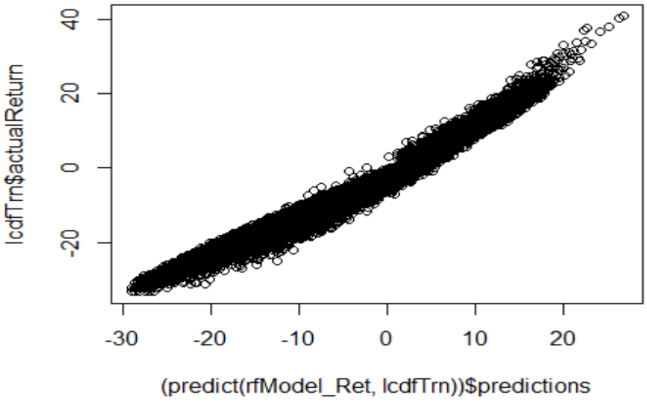
| # | Alpha | RMSE (Training, Testing) | MAE (Training, Testing) |
|---|---|---|---|
| 1 | 0 | 5.549436, 5.554813 | 3.533587, 3.543513 |
| 2 | 0.2 | 5.35092, 5.339394 | 3.461226, 3.543513 |
| 3 | 0.5 | 5.35089, 5.339214 | 3.460933, 3.462398 |
| 4 | 0.8 | 5.350864, 5.338935 | 3.460811, 3.461828 |
| 5 | 1 | 5.351482, 5.339509 | 3.4622, 3.462321 |

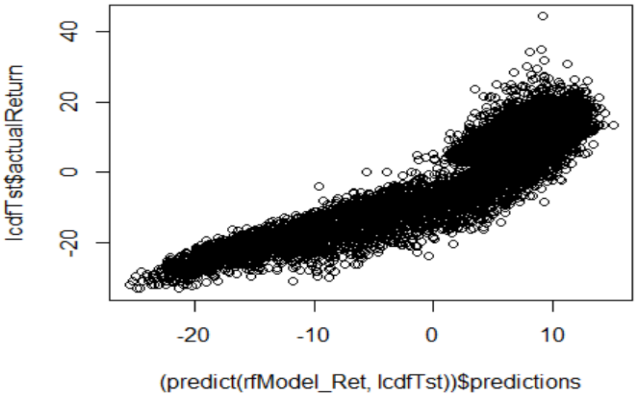
**The combination of least RMSE and MAE values exist when Alpha = 0.8. Increasing the value of Alpha brings the linear regression model closer to Lasso regression.**

**The top deciles created for alpha = 0.8 on test data shows that our model can identify the loans with highest actual returns. This model, however, is not able to completely identify charged off loans since it has 475 defaults out of a total of 3000 loans.**

## 2) Random Forest

- When number of trees = 200




**Performance by deciles –**

Training Set

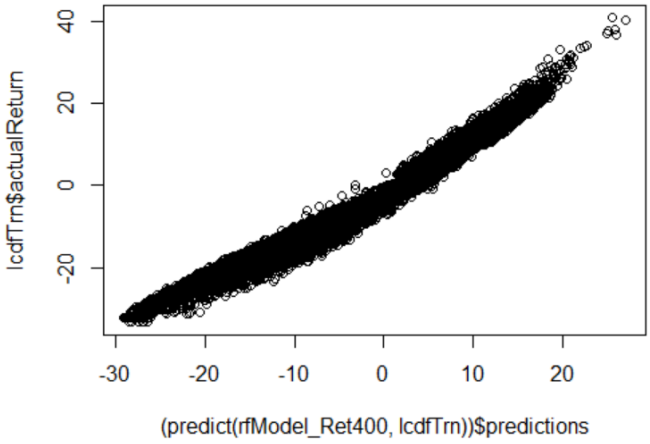
```
# A tibble: 10 x 14
    tile count avgpredRet numDefaults avgActRet  minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>      <dbl>       <int>     <dbl>   <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  7000       12.7          10      14.7    9.81   40.8   1.30     0    83  1895  3278  1417   302
 2     2  7000       10.1          24      11.0    7.59   17.1   1.77     4  1058  3386  2204   334    12
 3     3  7000       8.81          51      9.46    5.22   16.0   2.00     3  2254  3603  1075    54     9
 4     4  7000       7.85          54      8.22    3.43   13.2   2.24    33  3202  3559   186    15     5
 5     5  7000       7.02         106      7.26    2.67   12.3   2.38   273  4288  2358    57    23     1
 6     6  7000       6.19         141      6.43    1.51   12.4   2.29  1244  5147   546    49    11     3
 7     7  7000       5.30         199      5.49  -0.764   10.6   2.24  3136  3643   139    61    18     3
 8     8  7000       4.45         207      4.47  -0.733   8.32   2.50  5416  1428    97    37    19     3
 9     9  7000       2.83        1802      2.04   -9.68   8.66   2.74  5195   752   626   325    79    18
10    10  7000      -12.6        6999     -16.2   -33.3   0      3.00   585  1948  2382  1425   514   126
```

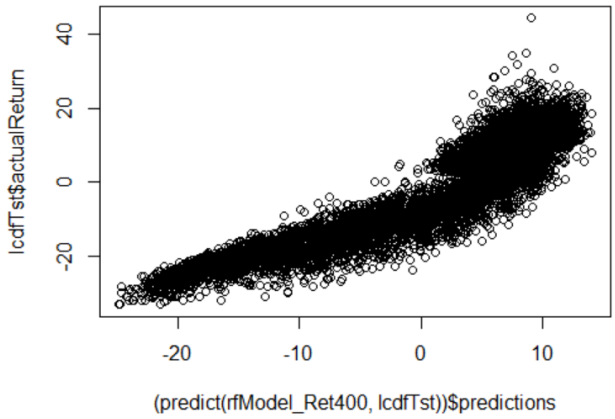
Testing Set

```
# A tibble: 10 x 14
    tile count avgpredRetTst numDefaults avgActRetTst minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>         <dbl>       <int>        <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3000         10.6          101         12.4   -2.63   30.9   2.10     0     0   384  1832   642   136
 2     2  3000          8.90         109         10.4   -8.49   44.4   2.08     0   161  1947   760   107    18
 3     3  3000          8.02          92          9.27  -7.00   34.2   2.11     0   733  1959   253    43     9
 4     4  3000          7.34          83          8.49  -8.63   24.5   2.12     0  1554  1320   100    22     3
 5     5  3000          6.67         117          7.66  -9.37   28.5   2.20     0  2151   736    89    17     6
 6     6  3000          5.86         142          6.63 -14.3    24.1   2.19   166  2448   317    51    13     2
 7     7  3000          5.10         117          5.60 -12.4    24.9   2.19  1414  1437   118    24     6     1
 8     8  3000          4.42         133          4.71 -16.4    21.5   2.23  2319   577    78    17     7     2
 9     9  3000          3.50         343          3.16 -16.4    23.7   2.34  2528   273   132    48    13     6
10    10  3000         -8.96        2955        -16.3  -33.3    10.8   2.98   272   770  1063   622   225    43
```

**RMSE for training set: 1.752642**
**RMSE for testing set: 3.995073**

- **When number of trees = 400**




## Training Set

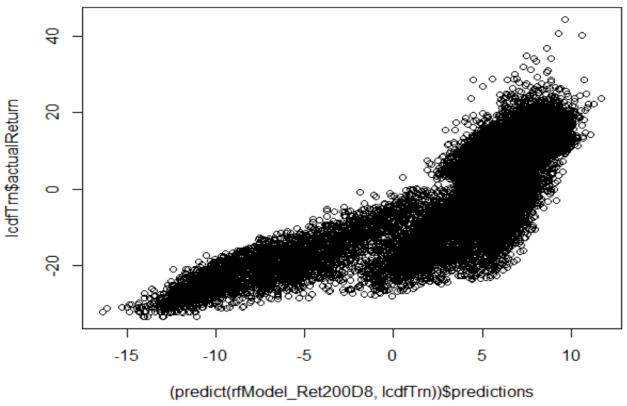
```
# A tibble: 10 x 14
    tile count avgpredRet400 numDefaults avgActRet  minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>         <dbl>       <int>     <dbl>   <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  7000          12.7          10      14.7    9.76   40.8   1.29     0    88  1892  3275  1416   303
 2     2  7000          10.1          26      11.1    6.95   17.1   1.77     3  1059  3386  2202   337    11
 3     3  7000          8.81          47      9.46    5.24   14.0   2.00     5  2226  3625  1079    57     7
 4     4  7000          7.84          57      8.21    4.26   13.3   2.26    32  3148  3611   189    15     5
 5     5  7000          7.01          89      7.26    2.20   11.8   2.38   252  4325  2348    57    14     4
 6     6  7000          6.18         169      6.43    1.57   10.1   2.28  1266  5167   495    54    16     2
 7     7  7000          5.30         185      5.49  -0.764   10.6   2.24  3120  3677   128    51    20     4
 8     8  7000          4.45         223      4.46  -0.700   7.34   2.50  5407  1415   108    53    16     1
 9     9  7000          2.84        1788      2.04   -8.71   7.46   2.74  5217   749   620   308    83    18
10    10  7000         -12.6        6999     -16.2  -33.3    0      3.00   587  1949  2378  1429   510   127
```

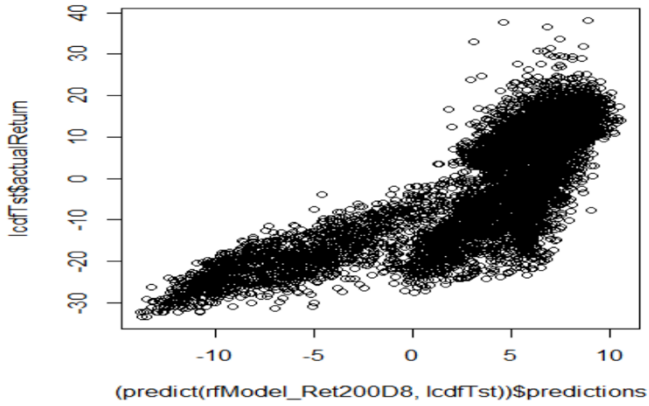
## Testing Set

```
# A tibble: 10 x 14
    tile count avgpredRet400Tst numDefaults avgActRetTst minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>            <dbl>       <int>        <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  3000             10.6          92         12.5  -2.35   30.9   2.10     0     1   353  1838   661   138
 2     2  3000              8.90        110         10.4  -8.63   44.4   2.07     0   111  1970   799    96    19
 3     3  3000              8.01         90          9.28 -7.72   32.0   2.11     0   720  2006   223    42     7
 4     4  3000              7.34        104          8.40 -11.7   34.2   2.14     0  1545  1316   110    23     6
 5     5  3000              6.67        111          7.64 -10.1   30.1   2.19     0  2199   721    63    13     1
 6     6  3000              5.85        120          6.73 -14.3   28.5   2.18   164  2480   300    44     6     5
 7     7  3000              5.08        131          5.52 -16.1   21.5   2.20  1412  1424   120    30    10     3
 8     8  3000              4.41        122          4.79 -13.5   23.7   2.22  2324   587    66    18     4     1
 9     9  3000              3.51        357          3.07 -16.8   16.9   2.35  2522   275   132    51    18     2
10    10  3000             -8.85       2955        -16.3  -33.3   10.8   2.98   277   762  1070   620   222    44
```

**RMSE for training set: 1.759196**
**RMSE for testing set: 4.019605**

- **When number of trees = 200, depth = 8**




## Training Set

```
# A tibble: 10 x 14
    tile count avgpredRet200D8 numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC  totD  totE  totF
   <int> <int>           <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int> <int>
 1     1  7000            8.34         287      12.4  -10.6   44.4   2.10     0     0   827  4235  1585   318
 2     2  7000            7.36         303      10.1  -16.3   34.8   2.12     0   226  4658  1863   219    30
 3     3  7000            6.91         384       8.98 -18.9   30.1   2.14     0  1372  4888   621    97    20
 4     4  7000            6.53         421       8.05 -20.5   28.5   2.16     0  3417  3233   298    46     5
 5     5  7000            6.15         518       7.14 -21.2   24.9   2.19     0  5191  1576   197    24    12
 6     6  7000            5.64         653       5.96 -23.3   28.9   2.22     0  5894   907   158    36     5
 7     7  7000            5.05         610       5.01 -23.3   26.8   2.24  1335  5090   439   107    22     7
 8     8  7000            4.46         353       4.62 -23.8   28.5   2.21  6218   588   120    53    17     3
 9     9  7000            4.07         339       3.72 -23.1   18.7   2.26  6621   239    88    36    14     2
10    10  7000           -2.06        5882     -13.6  -33.3   17.4   2.87  1612  1641  2004  1179   439   109
```

## Testing Set

```
# A tibble: 10 x 14
    tile count avgpredRet200D8Tst numDefaults avgActRetTst minRet maxRet avgTer  totA  totB  totC  totD  totE
   <int> <int>              <dbl>       <int>        <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int> <int> <int>
 1     1  3000               8.25         120         12.3  -10.0   38.1   2.11     0     1   386  1854   648
 2     2  3000               7.32         142         10.0  -17.4   33.6   2.13     0   125  2013   734   108
 3     3  3000               6.87         158          9.01 -19.6   36.6   2.13     0   670  2027   248    41
 4     4  3000               6.49         172          8.14 -21.5   30.9   2.14     0  1514  1295   145    40
 5     5  3000               6.12         219          7.09 -21.7   23.6   2.21     0  2229   642   104    23
 6     6  3000               5.60         301          5.88 -23.3   26.2   2.22     0  2451   430    87    27
 7     7  3000               5.03         214          5.35 -22.8   27.7   2.22   597  2173   177    38    13
 8     8  3000               4.44         162          4.71 -23.8   37.7   2.22  2625   277    57    29    11
 9     9  3000               4.06         129          3.78 -22.9   17.1   2.24  2827   112    32    28     1
10    10  3000              -1.64        2418        -12.9  -33.3   32.9   2.83   753   697   846   479   168
# ... with 1 more variable: totF <int>
```

**RMSE for training set: 5.924254**
**RMSE for testing set: 6.01261**

Based on the graphs, we see that in general, the variance is high for all the models. The overall performance for the model with the combination of 200 trees and tree depth of 8 is the least, whereas the random forest model with 200 trees is the best. If one were to compare the RMSE values for the 3 random forest models, we would observe that this model performs relatively better. This model can distinguish the higher actual returns which we can see in the top deciles.

The number of charged off loans is also very low (101), compared to the total (3000) in the top decile.

### 3) xgBoost

xgBoost being very sensitive to parameters, we had to experiment with many different combinations to get the best model.

**Model 1**

```
xgbParam <- list (max_depth = 6, objective = "reg:linear",eta = 0.1, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 3.226965
RMSE for test data: 3.255951

**Model 2**

```
xgbParam <- list (max_depth = 4, objective = "reg:linear",eta = 0.1, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 8.681218
RMSE for test data: 8.681218

**Model 3**

```
xgbParam <- list (max_depth = 4, objective = "reg:linear",eta = 0.5, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 1.205086
RMSE for test data: 1.204125

**Model 4**

```
xgbParam <- list (max_depth = 2, objective = "reg:linear", eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 1000, xgbWatchlist, early_stopping_rounds = 10, eta = 0.001 )
```

RMSE for training data: 8.458475
RMSE for test data: 8.531373

**Model 5**

```
xgbParam <- list (max_depth = 5, objective = "reg:linear", eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 1000, xgbWatchlist, early_stopping_rounds = 10, eta = 0.1 )
```

RMSE for training data: 8.662857
RMSE for test data: 8.739311

**Model 6**

```
xgbParam <- list (max_depth = 8, objective = "reg:linear",eta = 0.5, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 0.9707548
RMSE for test data: 1.014542

**Model 7**

```
xgbParam <- list (max_depth = 7, objective = "reg:linear",eta = 0.5, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 0.8671739
RMSE for test data: 0.9847829

**Model 8**

```
xgbParam <- list (max_depth = 6, objective = "reg:linear",eta = 0.5, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 4.961738
RMSE for test data: 5.007139

**Model 9**

```
xgbParam <- list (max_depth = 5, objective = "reg:linear",eta = 0.5, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 1.101874
RMSE for test data: 1.105213

**Model 10**

```
xgbParam <- list (max_depth = 7, objective = "reg:linear",eta = 0.1, eval_metric="error")
xgb_lsM1 <- xgb.train( xgbParam, dxTrn, nrounds = 400, xgbWatchlist, early_stopping_rounds = 10 )
```

RMSE for training data: 3.856708
RMSE for test data: 3.891654

| **Parameters** | **RMSE of Training Set** | **RMSE of Testing Set** |
|---|---|---|
| Max_depth = 6<br>eta = 0.1<br>nrounds = 400 | 3.226965 | 3.255951 |
| Max_depth = 4<br>eta = 0.1<br>nrounds = 400 | 8.681218 | 8.681218 |
| Max_depth = 4<br>eta = 0.5<br>nrounds = 400 | 1.205086 | 1.204125 |
| Max_depth = 2<br>eta = 0.1<br>nrounds = 1000 | 8.458475 | 8.531373 |
| Max_depth = 5<br>eta = 0.1<br>nrounds = 1000 | 8.662857 | 8.739311 |
| Max_depth = 8<br>eta = 0.5<br>nrounds = 400 | 0.9707548 | 1.014542 |
| Max_depth = 7<br>eta = 0.5<br>nrounds = 400 | 0.8671739 | 0.9847829 |
| Max_depth = 6<br>eta = 0.5<br>nrounds = 400 | 4.961738 | 5.007139 |
| Max_depth = 5<br>eta = 0.5<br>nrounds = 400 | 1.101874 | 1.105213 |
| Max_depth = 7<br>eta = 0.1<br>nrounds = 400 | 3.856708 | 3.891654 |

**The best xgBoost model came out to be with parameters – max_depth as 7, eta as 0.5, and Nrounds as 400, with the test set RMSE as 0.9847829.**

x**gBoost model has the least RMSE when compared to glm and random forest models.**

**Q4. Considering results from Questions 1 and 2 above – that is, considering the best model for predicting loan-status and that for predicting loan returns -- how would you select loans for investment? There can be multiple approaches for combining information from the two models - describe your approach, and show performance. How does performance here compare with use of single models?**

The first two questions dealt with finding the best model to accurately predict loan status. The best model here was the XgBoost model with parameters as follows:
**Max_Depth = 6**
**Eta = 0.01**
**Nrounds = 500**
**Nfold = 5**
From model 1, we see that loans with higher grades are less likely to default. Hence majority of them are 'Fully Paid'.

In question 3, we see that the best model that is used to predict Actual Returns is the XgBoost model with the following parameters:
**Max_depth = 7**
**Eta = 0.5**
**nrounds = 400**
We see that the highest returns are from the lower grade loans whereas the higher-grade loans have lower interest rates and hence lower returns.

A good investment is one that is the best of both model 1 and model 2. An investment on a loan should be one that gets Fully Paid and at the same time provides the best returns. In other words, the model that will tell us what a good investment will be a combination of these two XgBoost models.
We will be finding out the product of the predicted returns from a loan and the probability of that loan being fully paid. The top results of this calculation correspond to the best investment.

## Loans Model –

# A tibble: 10 × 14

| | tile | count | avgSc | numDefaults | avgActRet | minRet | maxRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 3000 | 0.297 | 915 | 4.30 | -33.3 | 44.4 | 2.43 | 3 | 78 | 857 | 1271 | 614 | 158 |
| 2 | 2 | 3000 | 0.225 | 728 | 5.27 | -33.3 | 34.2 | 2.28 | 7 | 225 | 1399 | 1085 | 242 | 38 |
| 3 | 3 | 3000 | 0.190 | 574 | 5.93 | -33.3 | 29.4 | 2.25 | 7 | 448 | 1707 | 692 | 132 | 12 |
| 4 | 4 | 3000 | 0.161 | 494 | 5.83 | -32.2 | 24.2 | 2.23 | 21 | 964 | 1564 | 385 | 57 | 8 |
| 5 | 5 | 3000 | 0.138 | 396 | 6.01 | -32.2 | 34.8 | 2.24 | 49 | 1545 | 1186 | 181 | 32 | 7 |
| 6 | 6 | 3000 | 0.118 | 307 | 5.97 | -32.2 | 22.3 | 2.20 | 123 | 1964 | 793 | 108 | 11 | 1 |
| 7 | 7 | 3000 | 0.0996 | 289 | 5.48 | -31.3 | 25.8 | 2.22 | 387 | 2171 | 387 | 51 | 2 | 2 |
| 8 | 8 | 3000 | 0.0809 | 218 | 5.04 | -32.3 | 19.5 | 2.26 | 999 | 1852 | 129 | 16 | 4 | 0 |
| 9 | 9 | 3000 | 0.0608 | 175 | 4.37 | -31.3 | 16.6 | 2.23 | 2165 | 796 | 32 | 6 | 1 | 0 |
| 10 | 10 | 3000 | 0.0348 | 96 | 3.86 | -28.1 | 14.0 | 2.21 | 2938 | 61 | 0 | 1 | 0 | 0 |

## Returns Model –

# A tibble: 10 × 14

| | tile | count | avgPredRet | numDefaults | avgActRet | minRet | maxRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 3000 | 12.8 | 0 | 13.3 | 0 | 44.4 | 1.99 | 0 | 1 | 131 | 1922 | 770 | 155 |
| 2 | 2 | 3000 | 10.3 | 0 | 10.7 | 0 | 34.8 | 2.05 | 0 | 43 | 1996 | 954 | 5 | 2 |
| 3 | 3 | 3000 | 9.17 | 0 | 9.48 | 0 | 24.2 | 2.07 | 0 | 512 | 2470 | 18 | 0 | 0 |
| 4 | 4 | 3000 | 8.32 | 0 | 8.56 | 0 | 24.9 | 2.12 | 0 | 1348 | 1651 | 1 | 0 | 0 |
| 5 | 5 | 3000 | 7.49 | 0 | 7.71 | 0 | 21.3 | 2.15 | 0 | 2690 | 310 | 0 | 0 | 0 |
| 6 | 6 | 3000 | 6.47 | 0 | 6.67 | 0 | 17.1 | 2.18 | 127 | 2869 | 2 | 2 | 0 | 0 |
| 7 | 7 | 3000 | 5.44 | 0 | 5.61 | 0 | 18.4 | 2.18 | 1511 | 1484 | 4 | 0 | 1 | 0 |
| 8 | 8 | 3000 | 4.64 | 1 | 4.78 | 0 | 10.7 | 2.22 | 2884 | 115 | 1 | 0 | 0 | 0 |
| 9 | 9 | 3000 | -1.31 | 1191 | -2.26 | -32.3 | 11.1 | 2.57 | 1909 | 345 | 387 | 257 | 79 | 23 |
| 10 | 10 | 3000 | -12.4 | 3000 | -12.5 | -33.3 | 11.4 | 3 | 268 | 697 | 1102 | 642 | 240 | 46 |

## Top scores of M2 ranked by M1 –

| | tile2 | count | avgPredRet | numDefaults | avgActRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 150 | 14.20981 | 0 | 15.30230 | 2.028985 | 0 | 0 | 2 | 33 | 62 | 44 |
| 2 | 2 | 150 | 13.17603 | 0 | 13.36227 | 2.097340 | 0 | 0 | 2 | 65 | 66 | 13 |
| 3 | 3 | 150 | 13.18480 | 0 | 13.56140 | 2.055633 | 0 | 0 | 2 | 69 | 68 | 10 |
| 4 | 4 | 150 | 12.88342 | 0 | 12.39838 | 2.230107 | 0 | 0 | 3 | 85 | 53 | 9 |
| 5 | 5 | 150 | 12.75331 | 0 | 12.75630 | 2.121177 | 0 | 0 | 7 | 91 | 47 | 5 |
| 6 | 6 | 150 | 12.86953 | 0 | 12.82615 | 2.202546 | 0 | 0 | 6 | 90 | 41 | 13 |
| 7 | 7 | 150 | 12.87981 | 0 | 13.69679 | 1.965558 | 0 | 0 | 4 | 94 | 43 | 8 |
| 8 | 8 | 150 | 12.86232 | 0 | 13.35526 | 2.044280 | 0 | 0 | 3 | 89 | 52 | 5 |
| 9 | 9 | 150 | 12.90516 | 0 | 13.03051 | 1.995638 | 0 | 0 | 2 | 104 | 34 | 9 |
| 10 | 10 | 150 | 12.57144 | 0 | 13.16765 | 1.925950 | 0 | 0 | 3 | 117 | 29 | 1 |
| 11 | 11 | 150 | 12.80167 | 0 | 13.35531 | 1.916696 | 0 | 0 | 6 | 108 | 29 | 6 |
| 12 | 12 | 150 | 12.66391 | 0 | 13.23689 | 1.928798 | 0 | 0 | 8 | 108 | 28 | 6 |
| 13 | 13 | 150 | 12.83322 | 0 | 13.57332 | 1.905033 | 0 | 0 | 9 | 103 | 33 | 5 |
| 14 | 14 | 150 | 12.68336 | 0 | 12.73277 | 1.963769 | 0 | 0 | 10 | 102 | 36 | 1 |
| 15 | 15 | 150 | 12.66978 | 0 | 13.43724 | 1.890504 | 0 | 0 | 9 | 105 | 34 | 2 |
| 16 | 16 | 150 | 12.66203 | 0 | 12.88849 | 1.982350 | 0 | 0 | 13 | 106 | 26 | 4 |
| 17 | 17 | 150 | 12.61240 | 0 | 12.94062 | 1.934383 | 0 | 0 | 9 | 113 | 24 | 3 |
| 18 | 18 | 150 | 12.58468 | 0 | 13.57328 | 1.775752 | 0 | 0 | 12 | 118 | 17 | 3 |
| 19 | 19 | 150 | 12.69730 | 0 | 13.05542 | 2.018325 | 0 | 1 | 15 | 98 | 31 | 5 |
| 20 | 20 | 150 | 12.46407 | 0 | 12.98984 | 1.880575 | 0 | 0 | 6 | 124 | 17 | 3 |

**Expected return on Investment –**

| | tile2 | count | avgPredRet | numDefaults | avgActRet | avgTer | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 150 | 15.51124 | 0 | 16.56524 | 1.968843 | 0 | 0 | 0 | 7 | 60 | 69 |
| 2 | 2 | 150 | 13.99592 | 0 | 13.82128 | 2.170714 | 0 | 0 | 0 | 36 | 90 | 23 |
| 3 | 3 | 150 | 13.42855 | 0 | 13.60272 | 2.122491 | 0 | 0 | 2 | 63 | 69 | 14 |
| 4 | 4 | 150 | 13.22749 | 0 | 13.63801 | 2.024933 | 0 | 0 | 2 | 64 | 71 | 13 |
| 5 | 5 | 150 | 13.09835 | 0 | 12.89087 | 2.074086 | 0 | 0 | 2 | 79 | 64 | 4 |
| 6 | 6 | 150 | 12.90653 | 0 | 13.37881 | 2.042017 | 0 | 0 | 2 | 97 | 48 | 3 |
| 7 | 7 | 150 | 12.79427 | 0 | 13.02966 | 2.053242 | 0 | 0 | 5 | 95 | 44 | 6 |
| 8 | 8 | 150 | 12.59190 | 0 | 12.80808 | 2.101209 | 0 | 0 | 7 | 103 | 37 | 3 |
| 9 | 9 | 150 | 12.67803 | 0 | 12.89644 | 2.007538 | 0 | 0 | 6 | 104 | 38 | 1 |
| 10 | 10 | 150 | 12.76196 | 0 | 13.23289 | 1.998777 | 0 | 0 | 1 | 105 | 41 | 3 |
| 11 | 11 | 150 | 12.63266 | 0 | 13.14273 | 1.965722 | 0 | 0 | 5 | 114 | 28 | 2 |
| 12 | 12 | 150 | 12.58173 | 0 | 13.29919 | 1.900799 | 0 | 0 | 2 | 115 | 29 | 3 |
| 13 | 13 | 150 | 12.45918 | 0 | 12.99292 | 1.930641 | 0 | 0 | 9 | 112 | 28 | 1 |
| 14 | 14 | 150 | 12.42480 | 0 | 12.75492 | 1.943308 | 0 | 0 | 10 | 111 | 28 | 1 |
| 15 | 15 | 150 | 12.50462 | 0 | 12.95583 | 2.032982 | 0 | 0 | 12 | 112 | 23 | 3 |
| 16 | 16 | 150 | 12.31080 | 0 | 13.04837 | 1.849728 | 0 | 0 | 10 | 127 | 11 | 2 |
| 17 | 17 | 150 | 12.27013 | 0 | 12.60921 | 1.990418 | 0 | 0 | 13 | 122 | 13 | 2 |
| 18 | 18 | 150 | 12.34597 | 0 | 13.15919 | 1.766972 | 0 | 0 | 13 | 120 | 17 | 0 |
| 19 | 19 | 150 | 12.19854 | 0 | 12.72788 | 1.992845 | 0 | 0 | 19 | 112 | 19 | 0 |
| 20 | 20 | 150 | 12.24539 | 0 | 12.68595 | 1.926133 | 0 | 1 | 11 | 124 | 12 | 2 |

**This shows the returns and the number of loans by grades that correspond to each return. All these 3000 loans are fully paid loans and investing in these loans have no risk.**

**When we observe the tibble, we notice that there are no grade A loans and just one grade B loans that have top returns. Grades C, D and E have the maximum returns.**

**Combining the loan status and actual return model, we can predict the highest returns on loans which do not have any number of defaults in the top 20 deciles. This model predicted the average actual return as 16.56% which is higher than both, the loan status model (4.30%), as well as returns model (13.3%).**

**Q5. As seen in data summaries and your work in the first assignment, higher grade loans are less likely to default, but also carry lower interest rates; many lower grad loans are fully paid, and these can yield higher returns. One approach may be to focus on lower grade loans (C and below), and try to identify those which are likely to be paid off. Develop models from the data on lower grade loans, and check if this can provide an effective investment approach – for this, you can use one of the methods (glm, rf, or gbm/xgb) which you find to give superior performance from earlier questions.**

**Can this provide a useful approach for investment? Compare performance with that in Question 4.**

We focus on lower grade loans and use these to predict loan status. We have created models using rf, glm, and xgBoost to predict loan status using only the lower grade loans.

- **Random Forest –**

# A tibble: 10 × 12

| | tile | count | avgScore | nDefaults | AvgReturn | AvgTerm | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 1320 | 0.900 | 143 | 7.50 | 2.13 | 0 | 0 | 1195 | 119 | 6 | 0 |
| 2 | 2 | 1320 | 0.861 | 184 | 6.94 | 2.16 | 0 | 0 | 1084 | 214 | 21 | 1 |
| 3 | 3 | 1320 | 0.838 | 229 | 6.36 | 2.21 | 0 | 0 | 999 | 289 | 30 | 2 |
| 4 | 4 | 1320 | 0.817 | 254 | 6.10 | 2.28 | 0 | 0 | 923 | 344 | 51 | 2 |
| 5 | 5 | 1320 | 0.798 | 256 | 6.30 | 2.21 | 0 | 0 | 862 | 380 | 75 | 3 |
| 6 | 6 | 1320 | 0.778 | 287 | 5.77 | 2.27 | 0 | 0 | 789 | 441 | 82 | 8 |
| 7 | 7 | 1320 | 0.757 | 315 | 5.39 | 2.28 | 0 | 0 | 718 | 477 | 113 | 12 |
| 8 | 8 | 1319 | 0.733 | 336 | 4.87 | 2.38 | 0 | 0 | 644 | 498 | 161 | 15 |
| 9 | 9 | 1319 | 0.700 | 343 | 4.97 | 2.36 | 0 | 0 | 520 | 545 | 220 | 33 |
| 10 | 10 | 1319 | 0.629 | 435 | 4.32 | 2.42 | 0 | 0 | 320 | 489 | 336 | 150 |

The random forest model is able to predict loans with a high average actual return of 7.50%. These loans are mostly concentrated in grade C. There are also 143 defaults out of 1320 loans in the top decile.

- **Linear (glm) –**

# A tibble: 10 × 12

| | tile | count | avgpredRet | nDefaults | avgActRet | avgTerm | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 1320 | 2.17 | 152 | 7.43 | 2.18 | 0 | 0 | 1176 | 136 | 8 | 0 |
| 2 | 2 | 1320 | 1.86 | 175 | 7.16 | 2.16 | 0 | 0 | 1141 | 166 | 11 | 2 |
| 3 | 3 | 1320 | 1.71 | 215 | 6.57 | 2.19 | 0 | 0 | 1066 | 232 | 20 | 2 |
| 4 | 4 | 1320 | 1.59 | 230 | 6.27 | 2.24 | 0 | 0 | 1003 | 288 | 23 | 6 |
| 5 | 5 | 1320 | 1.49 | 260 | 6.14 | 2.22 | 0 | 0 | 898 | 377 | 41 | 4 |
| 6 | 6 | 1320 | 1.39 | 271 | 5.74 | 2.28 | 0 | 0 | 857 | 416 | 44 | 3 |
| 7 | 7 | 1320 | 1.28 | 289 | 5.63 | 2.34 | 0 | 0 | 749 | 480 | 81 | 10 |
| 8 | 8 | 1319 | 1.15 | 349 | 4.74 | 2.33 | 0 | 0 | 610 | 575 | 126 | 8 |
| 9 | 9 | 1319 | 0.987 | 363 | 4.93 | 2.34 | 0 | 0 | 409 | 604 | 261 | 41 |
| 10 | 10 | 1319 | 0.641 | 478 | 3.91 | 2.42 | 0 | 0 | 145 | 522 | 480 | 150 |

Linear model is able to identify high average actual returns of 7.43% in the top decile. There are slightly greater number of defaults (152) out of 1320 loans.

- **xgBoost –**

# A tibble: 10 × 14

| | tile | count | avgSc | numDefaults | avgActRet | minRet | maxRet | avgTerm | totA | totB | totC | totD | totE | totF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <int> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | 1 | 1306 | 0.335 | 440 | 3.71 | -32.2 | 36.9 | 2.44 | 0 | 0 | 317 | 510 | 359 | 110 |
| 2 | 2 | 1306 | 0.274 | 354 | 5.02 | -33.3 | 40.2 | 2.37 | 0 | 0 | 446 | 613 | 209 | 37 |
| 3 | 3 | 1305 | 0.248 | 328 | 5.33 | -32.2 | 33.6 | 2.29 | 0 | 0 | 553 | 604 | 121 | 21 |
| 4 | 4 | 1305 | 0.228 | 306 | 5.68 | -33.3 | 23.9 | 2.26 | 0 | 0 | 669 | 535 | 87 | 14 |
| 5 | 5 | 1305 | 0.211 | 268 | 6.31 | -33.3 | 25.2 | 2.24 | 0 | 0 | 779 | 464 | 55 | 7 |
| 6 | 6 | 1305 | 0.196 | 259 | 5.96 | -32.2 | 29.2 | 2.20 | 0 | 0 | 919 | 320 | 55 | 10 |
| 7 | 7 | 1305 | 0.182 | 208 | 6.72 | -33.3 | 23.4 | 2.20 | 0 | 0 | 990 | 256 | 56 | 3 |
| 8 | 8 | 1305 | 0.166 | 200 | 6.90 | -31.1 | 31.3 | 2.18 | 0 | 0 | 1048 | 221 | 30 | 6 |
| 9 | 9 | 1305 | 0.147 | 158 | 7.41 | -32.2 | 34.8 | 2.18 | 0 | 0 | 1136 | 147 | 17 | 5 |
| 10 | 10 | 1305 | 0.116 | 140 | 7.48 | -31.4 | 27.2 | 2.17 | 0 | 0 | 1133 | 141 | 27 | 4 |

xgBoost on lower grades model did not perform well when compared to rf, and glm. It is only able to predict loans with average actual returns of 3.71%. This model is also not able to detect charged off loans very well. There are 440 defaults out of 1306 loans in the top decile.

**Random forest model had the best performance when modelling over lower grade loans. It predicted the highest average actual return out of glm, and xgBoost models, along with least number of defaults in the top deciles.**

**This approach is useful when we want to invest in lower grade loans. These loans have good average actual returns (7.50%) in the top decile. There is a little bit of risk factor involved since the number of defaults is 143 out of 1320 loans in the top decile which increases as we go to lower deciles. There is a trade off between default rate of loans and the average actual returns which must be considered when investing using this approach.**

**Compared to Q4 (Combined loan status and actual returns model), lower grade models did not perform very well. The model in Q4 has zero defaults in the top 20 deciles with very high average actual returns.**

**Q6. Considering all your results, which approach(s) would you recommend for investing in LC loans? Explain your rationale.**

We have used linear (glm) models, random forest, xgBoost, and decision trees to predict loan status and actual returns. In each of the techniques we have created multiple models by experimenting with hyper parameters.

In case of loan status (classification) we have evaluated the performance for each of the models using confusion matrix, ROC curves, and AUC values. In case of confusion matrix, we want minimum false positives, and false negatives, thereby leading to high accuracy. We also prefer those models which have high AUC values.

In case of actual returns (regression), we have calculated performance using RMSE, MAE and deciles. The lesser the value of RMSE and MAE, the better the model's performance in prediction. The deciles help us get an overall summary of the model performance by predicting high actual returns vs defaults. This can also be coupled with the total count of loan in each grade.

In both the scenarios, we observed that xgBoost model gave us the best predictions, as this technique gave us the best overall performance.

An investment in Lending club comprises of two elements - loan status and returns. We had models generated for both individually and for finding the best investment, we had to combine both the models. As stated above, xgBoost models proved to be the best. On doing so, we have been able to obtain a decile chart that gives us the best highest returns where the number of defaults for each grade has been zero.

We have also modelled the data to find the best returns among lower grade loans - C, D, E, F. The Random Forest modelling technique proved to give the best performance for this. However, when this was compared to the combined model (loan status and returns) to find the best investment approach, it turns out that the combined model performed way better.

**Out of all the models that we have made to predict loan status, and actual returns, the best approach that we would recommend for investing in LC loans is to combine the loan status and actual returns model using xgBoost. This combined model could easily predict very high average actual returns (16.56%), with zero defaults out of 150 loans in the top decile. The number of defaults remain zero in the top 20 deciles, and the average actual return decreases as we go to lower deciles, which makes this investment approach the safest one. Even in the $20^{th}$ decile the average actual return was 12.68%. So given the observed high average actual returns with zero defaults, we suggest using this approach for investing in LC loans.**