About Dataset:

The college dataset consists of 18 variables and 777 records. The 'Private' variable has categorial data whereas all other variables have continuous data. The variables include, number of applications, number of accepted applications, number of full-time undergraduates, room and board costs, student to faculty ratio and others.

Exploratory Data Analysis of College Dataset

Below are top 5 records of the dataset.

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	Terminal
Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	3300	450	2200	70	78
Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	6450	750	1500	29	30
Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	3750	400	1165	53	66
Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	5450	450	875	92	97
Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	4120	800	1500	76	72
Albertson College	Yes	587	479	158	38	62	678	41	13500	3335	500	675	67	73
Albertus Magnus College	Yes	353	340	103	17	45	416	230	13290	5720	500	1500	90	93
Albion College	Yes	1899	1720	489	37	68	1594	32	13868	4826	450	850	89	100
Albright College	Yes	1038	839	227	30	63	973	306	15595	4400	300	500	79	84
Alderson-Broaddus College	Yes	582	498	172	21	44	799	78	10468	3380	660	1800	40	41

Figure 1: Head of Data

Below structure shows that only one variable has categorical data whereas all other variables have continuous data.

```
> str(College)
'data.frame':   777 obs. of  18 variables:
 $ Private    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
 $ Apps       : num  1660 2186 1428 417 193 ...
 $ Accept     : num  1232 1924 1097 349 146 ...
 $ Enroll     : num  721 512 336 137 55 158 103 489 227 172 ...
 $ Top10perc  : num  23 16 22 60 16 38 17 37 30 21 ...
 $ Top25perc  : num  52 29 50 89 44 62 45 68 63 44 ...
 $ F.Undergrad: num  2885 2683 1036 510 249 ...
 $ P.Undergrad: num  537 1227 99 63 869 ...
 $ Outstate   : num  7440 12280 11250 12960 7560 ...
 $ Room.Board : num  3300 6450 3750 5450 4120 ...
 $ Books      : num  450 750 400 450 800 500 500 450 300 660 ...
 $ Personal   : num  2200 1500 1165 875 1500 ...
 $ PhD        : num  70 29 53 92 76 67 90 89 79 40 ...
 $ Terminal   : num  78 30 66 97 72 73 93 100 84 41 ...
 $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
 $ perc.alumni: num  12 16 30 37 2 11 26 37 23 15 ...
 $ Expend     : num  7041 10527 8735 19016 10922 ...
 $ Grad.Rate  : num  60 56 54 59 15 55 63 73 80 52 ...
```

Figure 2: Structure of Dataset

Using the summary() command, we got to know that there are total 565 Private Universities f which data is available whereas 212 universities are public. The summary() tells us that there no missing values and hence we will proceed for further exploratory analysis.

```
> summary(College)
 Private        Apps           Accept          Enroll       Top10perc       Top25perc
 No :212   Min.   :   81   Min.   :   72   Min.   :  35   Min.   : 1.00   Min.   :  9.0
 Yes:565   1st Qu.:  776   1st Qu.:  604   1st Qu.: 242   1st Qu.:15.00   1st Qu.: 41.0
           Median : 1558   Median : 1110   Median : 434   Median :23.00   Median : 54.0
           Mean   : 3002   Mean   : 2019   Mean   : 780   Mean   :27.56   Mean   : 55.8
           3rd Qu.: 3624   3rd Qu.: 2424   3rd Qu.: 902   3rd Qu.:35.00   3rd Qu.: 69.0
           Max.   :48094   Max.   :26330   Max.   :6392   Max.   :96.00   Max.   :100.0
  F.Undergrad     P.Undergrad         Outstate       Room.Board       Books
 Min.   :  139   Min.   :    1.0   Min.   : 2340   Min.   :1780   Min.   :  96.0
 1st Qu.:  992   1st Qu.:   95.0   1st Qu.: 7320   1st Qu.:3597   1st Qu.: 470.0
 Median : 1707   Median :  353.0   Median : 9990   Median :4200   Median : 500.0
 Mean   : 3700   Mean   :  855.3   Mean   :10441   Mean   :4358   Mean   : 549.4
 3rd Qu.: 4005   3rd Qu.:  967.0   3rd Qu.:12925   3rd Qu.:5050   3rd Qu.: 600.0
 Max.   :31643   Max.   :21836.0   Max.   :21700   Max.   :8124   Max.   :2340.0
    Personal         PhD            Terminal       S.F.Ratio      perc.alumni
 Min.   : 250   Min.   :  8.00   Min.   : 24.0   Min.   : 2.50   Min.   : 0.00
 1st Qu.: 850   1st Qu.: 62.00   1st Qu.: 71.0   1st Qu.:11.50   1st Qu.:13.00
 Median :1200   Median : 75.00   Median : 82.0   Median :13.60   Median :21.00
 Mean   :1341   Mean   : 72.66   Mean   : 79.7   Mean   :14.09   Mean   :22.74
 3rd Qu.:1700   3rd Qu.: 85.00   3rd Qu.: 92.0   3rd Qu.:16.50   3rd Qu.:31.00
 Max.   :6800   Max.   :103.00   Max.   :100.0   Max.   :39.80   Max.   :64.00
     Expend        Grad.Rate
 Min.   : 3186   Min.   : 10.00
 1st Qu.: 6751   1st Qu.: 53.00
 Median : 8377   Median : 65.00
 Mean   : 9660   Mean   : 65.46
 3rd Qu.:10830   3rd Qu.: 78.00
 Max.   :56233   Max.   :118.00
```

Figure 3: Summary of Dataset

Using the ggpair() command we plotted scatter plot, density plot and correlation plot of the different variables of the dataset. A strong relation can be observed for variables application, enrollment of the

students and the acceptance. For variables room boarding, top25% and graduation rate the density plots are distributed normally, whereas for other variables it is rightly skewed.

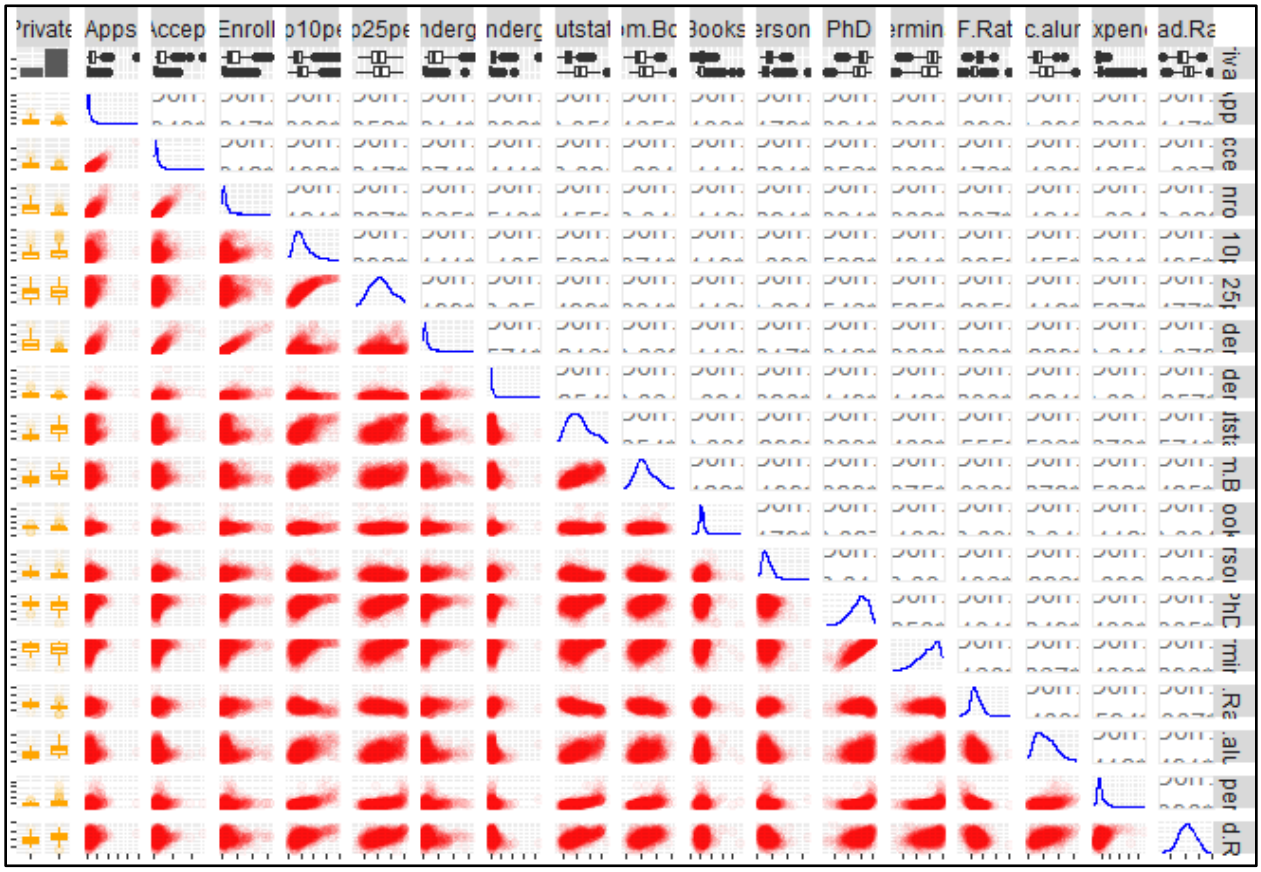

Figure 4: GGPair

Considering the variable of private universities, we again ran command ggpair() to better understand it's relationship with other variables. The other variables which were considered are applications,

acceptances, enrollment, Top10%, Top25%, undergraduate (FL), undergraduate (PT), outstate, room boarding and books.

Figure 5: GGPair (Private)

With categorical data of variable private taking into consideration, we used box plot and arranged using grid.arrange() to better understand it's relation with other variables like outstate, Top10%, applications, fulltime undergraduate, acceptance, part-time undergraduate, enrollment, student-to-faculty ratio. This visualization gives us a chance to observe means of variables, so that critical variable could be identified.

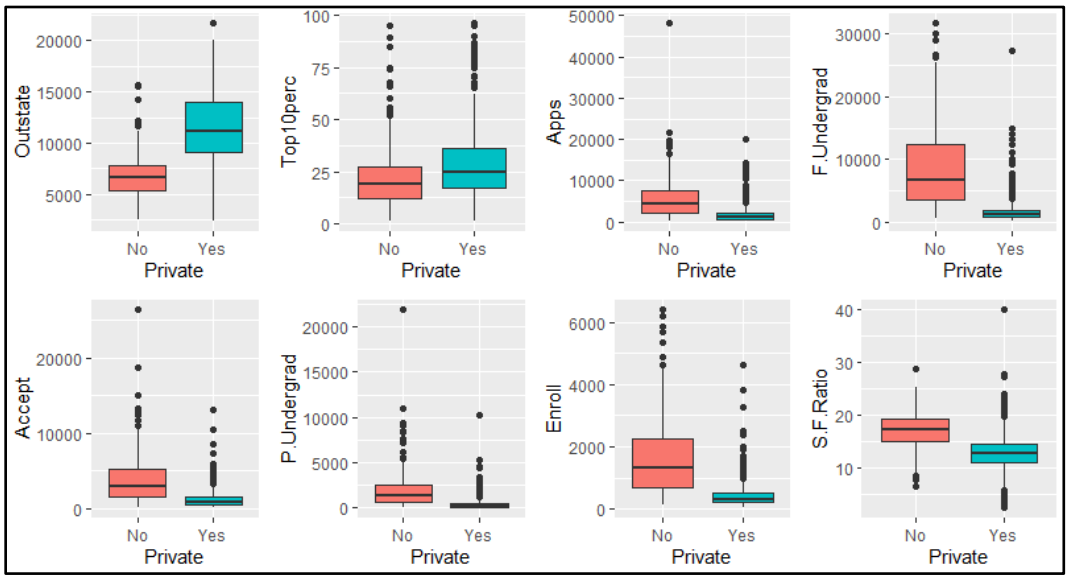

Figure 6: Boxplot

Part I

Splitting Dataset

Using caret package we split the dataset into 80%-20% and assigned 80% data to training and 20% for testing purpose. The train dataset has 622 records whereas the testing dataset has 155 records.

```
> ########################################
> #Task 2: Splitting of data-set
> ########################################
> set.seed(113)
> ## doing 80% - 20% proportion
> trainindex = createDataPartition(College$Private, p=0.80, list = F)
> ## 80% for training purpose
> train = College[trainindex,]
> ## 20% for testing purpose
> test = College[-trainindex,]
```

Figure 7: Code chunk for splitting of data

Logistic Regression Model-1

Using glm() command we found that the variables full-time undergraduate, outstate, PhD, student-to-faculty ratio, percent alumni are more significant as compared to the other variables and now we will modify our model by selecting these variables for an another model. Also to be considered is the AIC value which is 203.48.

```
glm(formula = Private ~ ., family = binomial(link = "logit"),
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.8423  -0.0146   0.0417   0.1453   2.7473

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.3664086  2.3175009   0.590  0.55546
Apps        -0.0004748  0.0002909  -1.632  0.10271
Accept       0.0003361  0.0005760   0.584  0.55950
Enroll       0.0009537  0.0010366   0.920  0.35755
Top10perc    0.0056277  0.0339213   0.166  0.86823
Top25perc    0.0107587  0.0224072   0.480  0.63112
F.Undergrad -0.0006064  0.0002102  -2.885  0.00392 **
P.Undergrad  0.0001159  0.0002036   0.569  0.56929
Outstate     0.0007047  0.0001373   5.132 2.86e-07 ***
Room.Board   0.0003317  0.0003279   1.012  0.31176
Books        0.0026860  0.0018117   1.483  0.13819
Personal    -0.0004718  0.0003286  -1.436  0.15102
PhD         -0.0632071  0.0346766  -1.823  0.06834 .
Terminal    -0.0391011  0.0345437  -1.132  0.25766
S.F.Ratio   -0.1417346  0.0825530  -1.717  0.08600 .
perc.alumni  0.0434273  0.0261892   1.658  0.09727 .
Expend       0.0001086  0.0001490   0.729  0.46600
Grad.Rate    0.0165886  0.0146219   1.135  0.25658
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 729.64  on 621  degrees of freedom
Residual deviance: 167.48  on 604  degrees of freedom
AIC: 203.48

Number of Fisher Scoring iterations: 8
```

Figure 8: GLM

Second Model

The difference between null deviance and the residual deviance has been increased as compared to the values of previous model. The AIC of previous model was 203.48 whereas the AIC of this model have been decreased to 191.22, therefore we are confident to say that the second model namely modely is more preferred since the second model explains the variations in a better way.

```
> modely = glm(Private ~ F.Undergrad + Outstate + PhD +
+                        S.F.Ratio + perc.alumni,
+              family = binomial(link = "logit"), data = train)
> summary(modely)

Call:
glm(formula = Private ~ F.Undergrad + Outstate + PhD + S.F.Ratio +
    perc.alumni, family = binomial(link = "logit"), data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6856  -0.0108   0.0399   0.1493   3.2420

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.020e+00  1.785e+00   1.692   0.0907 .
F.Undergrad -5.790e-04  8.516e-05  -6.799 1.06e-11 ***
Outstate     8.335e-04  1.058e-04   7.875 3.42e-15 ***
PhD         -8.081e-02  1.800e-02  -4.490 7.12e-06 ***
S.F.Ratio   -1.436e-01  6.533e-02  -2.198   0.0279 *
perc.alumni  5.502e-02  2.335e-02   2.356   0.0185 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 729.64  on 621  degrees of freedom
Residual deviance: 179.22  on 616  degrees of freedom
AIC: 191.22

Number of Fisher Scoring iterations: 8
```

Figure 9: Model(y) 2

We used coef() command to extract coefficients of the model. These coefficients are of logits and to observe a sigmoidal function we can plot this model since the sigmoidal function is the feature of a logistic regression model so as to fit binomial data.

```
> #Display regression coefficients (log-odds)
> coef(modely)
  (Intercept)   F.Undergrad      Outstate           PhD     S.F.Ratio
 3.0201165976 -0.0005790021  0.0008335470 -0.0808109297 -0.1436054839
  perc.alumni
 0.0550179411
```

Figure 10: Regression Co-efficient (log-odds)

We converted the log-odds to the odds using the command exp().
Odds of College increase by a factor of 0.999 for each 1 unit increase in full-time undergraduate.
Odds of College increase by a factor of 1 for each 1 unit increase in outstate.
Odds of College increase by a factor of 0.92 for each 1 unit increase in PhD.
Odds of College increase by a factor of 0.866 for each 1 unit increase in student-to-faculty ratio.
Odds of College increase by a factor of 1 for each 1 unit increase in percent alumni.

```
> #Display regression coefficients (odds)
> exp(coef(modely))
(Intercept) F.Undergrad    Outstate         PhD   S.F.Ratio perc.alumni
 20.4936811   0.9994212   1.0008339   0.9223681   0.8662294   1.0565596
```

Figure 11: Regression Co-efficient (odds)

Confusion Matrix (Train Data)

True Positive (TP): 434
True Negative (TN): 150
False Positive (FP): 20
False Negative (FN): 18

It is good that accuracy of the model is 93.89% and FP are low as compared to the TP and the FN are low as compared to the TN.

```
> confusionMatrix(predicted.class.min, train$Private, positive = 'Ye
s')
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  150  18
       Yes  20 434

               Accuracy : 0.9389
                 95% CI : (0.9171, 0.9564)
    No Information Rate : 0.7267
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8456

 Mcnemar's Test P-Value : 0.8711

            Sensitivity : 0.9602
            Specificity : 0.8824
         Pos Pred Value : 0.9559
         Neg Pred Value : 0.8929
             Prevalence : 0.7267
         Detection Rate : 0.6977
   Detection Prevalence : 0.7299
      Balanced Accuracy : 0.9213

       'Positive' Class : Yes
```

Figure 12: Confusion Matrix (Train Data)

Analysis of damage done by misclassifications

The false negative (or the type 2 error) is causing the more damage since the Negative Predictive Value (NPV) is 0.8929. However, if we look at the Positive Predictive Value (PPV), it is 0.9559. Since the PPV > NPV (0.9559 > 0.8929), therefore we are confident to say that Type 2 error is causing more damage in this data model to the accuracy.

Accuracy, Precision, Recall, and Specificity

Accuracy = (TN + TP)/ (TN + FP + FN + TP)
= 584/ 622
= 0.9389

93.89% of accuracy has been shown when predicting the correct values of Yes and No.

Precision = TP/ (FP + TP)
= 434/ (20 + 434)
= 0.9559

If the value of precision is 1, then it means that no false positive values were predicted, however this is not the case since the precision value is 0.9559.

Recall = TP/ (TP + FN)
= 434/ (434 + 18)
= 0.96

This value tells us about the proportion which was correctly predicted from total true values. Here the recall value is 0.96, that means 96% (434) of the values were predicted correctly from a total of 100% (452).

Specificity = TN/ (TN + FP)
= 150/ (150 + 20)
= 0.88

To measure the accuracy of the model pertaining to negative values is known as specificity. Here the specificity value is 0.88, which means that 88% of the negative values were predicted correctly.

Confusion Matrix (Test Data)

True Positive (TP): 109
True Negative (TN): 38
False Positive (FP): 04
False Negative (FN): 04

Although accuracy is not the only parameter to evaluate the model, but still we can say that the accuracy of the model is increased to 94.84% from 93.89% (when we ran the model for train data). Also the FP and FN are low as compared to the TP and TN.

Here we can also see that the model predicted positive values with 96.46% whereas the negative values were predicted with 90.48% accuracy. Also, the specificity of the model is 90.48%.

```
> confusionMatrix(predicted.class.min1, test$Private, positive = 'Ye
s')
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No   38   4
       Yes   4 109

               Accuracy : 0.9484
                 95% CI : (0.9008, 0.9775)
    No Information Rate : 0.729
    P-Value [Acc > NIR] : 1.553e-12

                  Kappa : 0.8694

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9646
            Specificity : 0.9048
         Pos Pred Value : 0.9646
         Neg Pred Value : 0.9048
             Prevalence : 0.7290
         Detection Rate : 0.7032
   Detection Prevalence : 0.7290
      Balanced Accuracy : 0.9347

       'Positive' Class : Yes
```

Figure 13: Confusion Matrix (Test Data)

Receiver Operator Characteristic Curve

The receiver operator characteristic curve tells as how good our model is by visualizing sensitivity and specificity. The curve of ideal model would be a straight vertical line from the grey line and then bending at 90 degree to continue as straight horizontal line and meeting the grey line. The pink line shown tells us that our model is not ideal but it is close to be an ideal model.

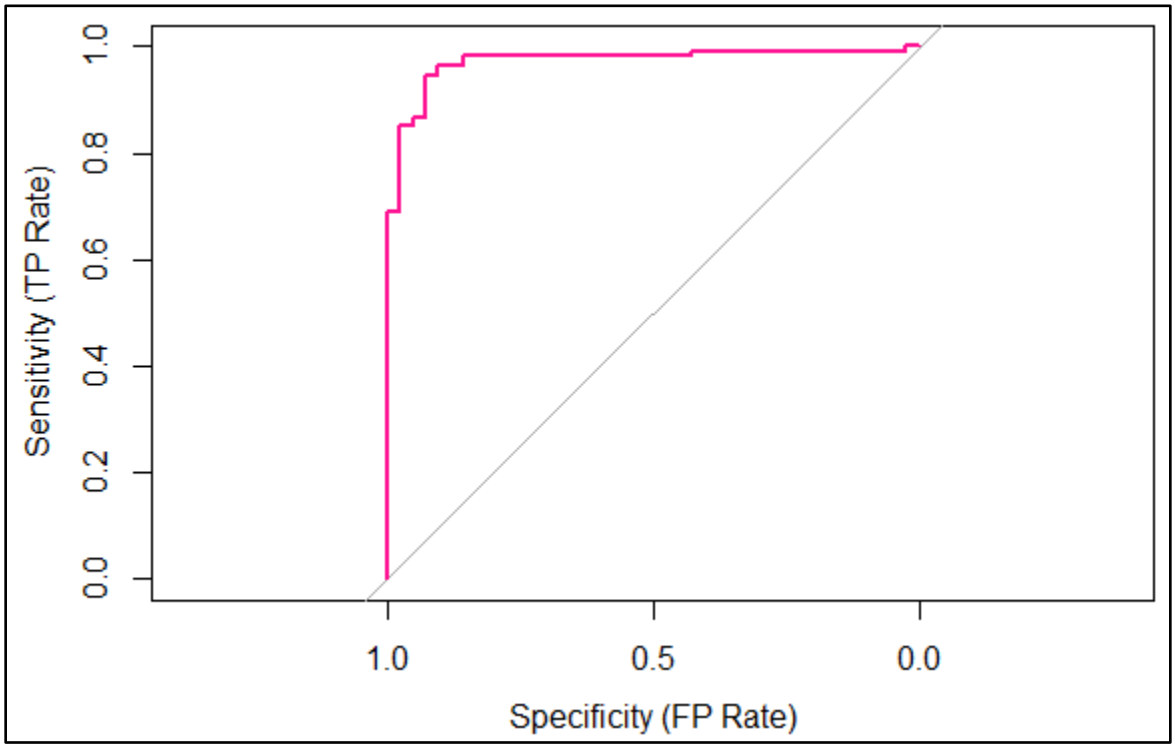

Figure 14: ROC Curve

Area Under the Curve (AUC)

The ideal value of area under curve (AUC) will be 1, whereas the AUC value of the model is 0.9718. This parameter tells us the accuracy of the model and 97.18% is a good predictive accuracy.

```
> ## Calculate area under the curve
> auc <- auc(ROC1)
> auc
Area under the curve: 0.9718
```

Figure 15: Calculation for area under the curve

In part I, we learnt and experimented about how a generalized linear model and logistic regression can be implemented. About how to modify the model so as to increase the accuracy of prediction. Then we further evaluated the chosen using multiple parameters for example area under the curve, (different parameters associated with confusion matrix) accuracy, precision, recall, specificity.

This exercise helped us to understand more deeper about the generalized linear model where dependent variable mandatory to be continuous or normally distributed, also we explored and experimented about the logistic regression using college dataset.

Part II

(Ridge vs Lasso)

Splitting Dataset (70% - 30% ratio)

The dataset was split into a 70% - 30% ratio, using the function createDataPartition(). The 70% of the rows were assigned to the training dataset for training the model and the remaining 30% were assigned for testing the model. For fixing the randomization I used seed value as 113. The train dataset has 543 records whereas the testing dataset has 234 records.

```
#########################################################
# Split Data-set
#########################################################

#setting seed value so that randomization could be fixed
set.seed(113)

# Creating the Train and Test set - with proportion as (70/30 split)
traindata.mine <- createDataPartition(datamine$Private, p=0.70, list = F)

# Including 70% rows to training data
train.mine <- datamine[traindata.mine,]

# Including remaining 30% rows to testing data
test.mine  <- datamine[-traindata.mine,]
```

Figure 166: Code for splitting of data

Below is the code for converting data frame into matrix so that we could use the glmnet function. Moving further I also excluded the output variable from both, the training and the testing set, then created the vector variables accordingly.

```
# Converting data frame to matrix and separating output variable
train.mine_x <- model.matrix(Grad.Rate~.,train.mine)[,-18]
test.mine_x <- model.matrix(Grad.Rate~.,test.mine)[,-18]

# For training and testing set, created vector variable
train.mine_y <- train.mine$Grad.Rate
test.mine_y <- test.mine$Grad.Rate
```

Figure 17: Code chunk for separating output variable & for vector variable

Ridge Regression

Estimating Lambda values

To find the optimum values of Lambda, I used cv.glmnet function first for ridge regression for the purpose of cross validation and then used function cv.ridge$lambda.min and cv.ridge$lambda.1se for the values.

The minimum mean cross validated error is lambda.min, whereas the lambda.1se gives the most regularized model, where value of cross validated error is within one standard error of the minimum (Hastie, Qian, and Tay &2021).

```
> cv.ridge.mine$lambda.min
[1] 2.193105
```

Figure 18: Lambda.min Value

The value of lambda.min is 2.193105.

```
> cv.ridge.mine$lambda.1se
[1] 32.56691
```

Figure 19: Lambda.1se value

The value of lambda.1se is 32.56691. Now we know that when lambda is 2.193105 then the minimum mean squared error will be achieved.

cv.glmnet Plot

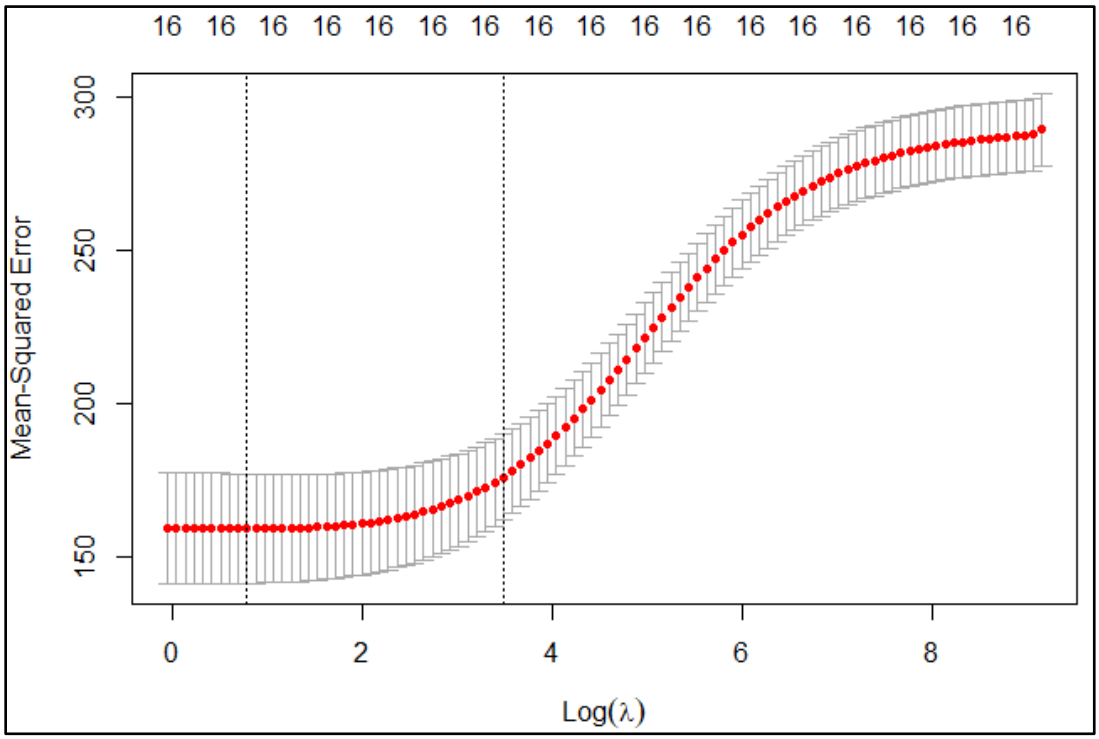

Figure 20: Plot of cv.glmnet

The x-axis shows the log values of lambda whereas the y-axis shows the mean squared error values. Non-zero coefficients numbers written on axis also. There could be two dashed lines be seen. These lines depicts lambda values. The line on the left side is the lambda.min value whereas the line on the right side shows the lambda.1se value. However the red dots in the plot are calculated using the cross validation and are also called loss matrics and the line which is consistently above and below these red dots are actually the upper and lower (ranges of) standard deviation curves.

Fitting Ridge Regression Model:

To fit the ridge regression model, we have used the glmnet() finction. For ridge model the alpha value used is 0 and then we calculated the lambda.min and the lambda.1se values.

```
> model.L2.min

Call:  glmnet(x = train.mine_x, y = train.mine_y,
 alpha = 0, lambda = cv.ridge.mine$lambda.min)

  Df  %Dev Lambda
1 16 47.74  2.193
```

Figure 21: Fitting final model using lambda.min

```
> model.L2.1se

Call:  glmnet(x = train.mine_x, y = train.mine_y,
 alpha = 0, lambda = cv.ridge.mine$lambda.1se)

  Df  %Dev Lambda
1 16 40.05  32.57
```

Figure 22: Fitting final model using lambda.1se

For lambda.min and for lambda.1se the number of non-zero coefficients are 16 whereas the in terms of deviance 1se is 40.05% whereas for min it is 47.74%. Since the model with lower deviance is good therefore to fit the ridge model we will use the lambda.1se value.

```
> coef(model.L2.min)
18 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) 31.6554227859
(Intercept)  .
PrivateYes   5.2334145402
Apps         0.0007351354
Accept       0.0003027258
Enroll       0.0006809637
Top10perc    0.0724757514
Top25perc    0.1246139837
F.Undergrad -0.0003085953
P.Undergrad -0.0015631277
Outstate     0.0005771604
Room.Board   0.0013038744
Books       -0.0046301801
Personal    -0.0018125552
PhD          0.0672672606
Terminal    -0.0218974126
S.F.Ratio    0.2980869099
perc.alumni  0.2694775319
```

```
> coef(model.L2.1se)
18 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)  4.458529e+01
(Intercept)  .
PrivateYes   2.603361e+00
Apps         1.882459e-04
Accept       1.437133e-04
Enroll       9.667888e-05
Top10perc    7.542458e-02
Top25perc    7.268442e-02
F.Undergrad -5.616342e-05
P.Undergrad -8.040846e-04
Outstate     3.887647e-04
Room.Board   9.323583e-04
Books       -1.456238e-03
Personal    -1.307606e-03
PhD          4.137114e-02
Terminal     3.198404e-02
S.F.Ratio   -9.605996e-02
perc.alumni  1.352851e-01
```

Figure 23: Displaying Regression coefficients

Above is the comparison of the coefficient values, here we can easily observe how the coefficient values are being moved towards the zero value by the model. The intercept of the lambda.1se model as compared to the lambda.min is reduced significantly.

Checking Performance of the Training set

Using lambda.1se so as to fit the ridge regression prediction model against the training set. To measure the performance of the training set I used rmse() function. The rmse value is 13.15157.

```
> train.rmse = rmse(train.mine_y, preds.trains)
> train.rmse
[1] 13.15157
```

Figure24: Checking Performance of Train set

Checking Performance of the Testing set

Using again rmse() function, we found the rmse value for the testing set, which is 14.01189. However the rmse value for the training set was 13.15157, that means there is not significant difference between the rmse values, therefore the model is not overfitting.

```
> test.rmse
[1] 14.01189
```

Figure 25: Checking Performance of Testing set

Lasso Regression

Estimating Lambda values

To find the optimum values of Lambda, I have again used the cv.glmnet function first for lasso regression for the purpose of cross validation and then used function cv.lasso$lambda.min and cv.lasso$lambda.1se for the values.

```
> cv.lasso.mine$lambda.min
[1] 0.2093425
```

Figure 26: Lambda.min Value

The value of lambda.min is 0.2093425.

```
> cv.lasso.mine$lambda.1se
[1] 1.47687
```

Figure 27: Lambda.1se value

The value of lambda.1se is 1.47687.

Now we know that when lambda is 0.2093425 then the minimum mean squared error will be achieved.

cv.glmnet Plot

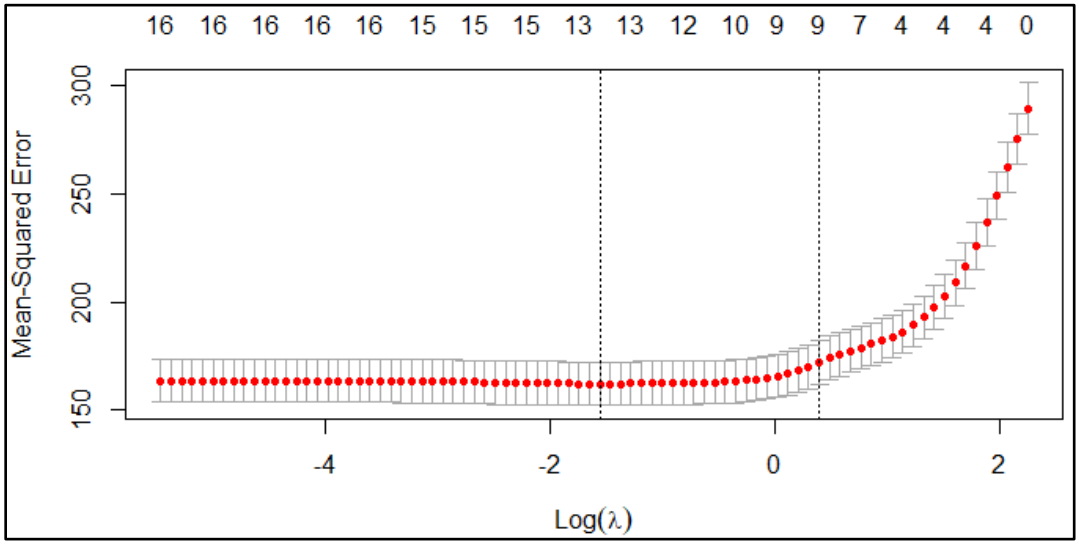

Figure 28: Plot of cv.glmnet(lasso)

Non-zero coefficients numbers written on axis (the number of non-zero coefficients moving towards zero). The x-axis shows the log values of lambda whereas the y-axis shows the mean squared error values.. There could be two dashed lines be seen. These lines represent lambda values. The line on the left side is the lambda.min value whereas the line on the right side shows the lambda.1se value. However the red dots in the plot are calculated using the cross validation and are also called loss matrics and the light grey line which is consistently above and below these red dots are actually the upper and lower (ranges

of) standard deviation curves. We need to fit the model in a way that the rmse value of test set remains low and correspondingly with the change in rmse values of the train set.

Fitting Ridge Regression Model:

```
> model.L2.min = glmnet(train.mine_x,train.mine_
y, alpha =1,
+                       lambda =cv.lasso.mine$lam
bda.min)
> model.L2.min

Call:  glmnet(x = train.mine_x, y = train.mine_y,
 alpha = 1, lambda = cv.lasso.mine$lambda.min)

  Df  %Dev Lambda
1 13 47.89 0.2093
```

Figure 29: Fitting final model using lambda.min

```
> model.L2.1se = glmnet(train.mine_x,train.mine_
y, alpha =1,
+                       lambda =cv.lasso.mine$lam
bda.1se)
> model.L2.1se

Call:  glmnet(x = train.mine_x, y = train.mine_y,
 alpha = 1, lambda = cv.lasso.mine$lambda.1se)

  Df  %Dev Lambda
1  9 42.74  1.477
```

Figure 30: Fitting final model using lambda.1se

To fit the lasso regression model, we have againused the glmnet() finction. For lasso model the alpha value used is 1 and then we have calculated the lambda.min and the lambda.1se values.

For lambda.min and for lambda.1se the number of non-zero coefficients are 13 and 9 correspondingly whereas the in terms of deviance 1se is 42.74% whereas for min it is 47.89%. Since the model with lower deviance is good therefore to fit the lasso model we will use the lambda.1se value.

```
> coef(model.L2.min)
18 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept) 30.132118879
(Intercept)  .
PrivateYes   5.678137956
Apps         0.001126231
Accept       .
Enroll       .
Top10perc    0.012058954
Top25perc    0.158792332
F.Undergrad -0.000249889
P.Undergrad -0.001584721
Outstate     0.000602192
Room.Board   0.001062134
Books       -0.003774854
Personal    -0.001668545
PhD          0.046978691
Terminal     .
S.F.Ratio    0.295053717
perc.alumni  0.297494855
```

```
> coef(model.L2.1se)
18 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)  3.986059e+01
(Intercept)  .
PrivateYes   6.407426e-01
Apps         9.410521e-05
Accept       .
Enroll       .
Top10perc    4.017636e-02
Top25perc    1.409686e-01
F.Undergrad  .
P.Undergrad -6.098275e-04
Outstate     9.050227e-04
Room.Board   5.985686e-04
Books        .
Personal    -8.773282e-04
PhD          .
Terminal     .
S.F.Ratio    .
perc.alumni  2.446443e-01
```

Figure 31: Displaying Regression coefficients

Above is the comparison of the coefficient values, here we can easily observe how the coefficient values are being moved towards the zero value by the model. The intercept of the lambda.1se model as compared to the lambda.min is reduced significantly from 30.13 to 3.98e+01.

Checking Performance of the Training set

Using lambda.1se so as to fit the lasso regression prediction model against the training set. To measure the performance of the training set I used rmse() function. The rmse value is 12.8537.

```
> train.rmse.L2 = rmse(train.mine_y, preds.trains.L2)
> train.rmse.L2
[1] 12.8537
```

Figure 32: Checking Performance of Train set

Checking Performance of the Testing set

Using again rmse() function, we found the rmse value for the testing set, which is 13.73793. However the rmse value for the training set was 12.8537, that means there is not significant difference between the rmse values, therefore the model is not overfitting.

```
> test.rmse.L2 = rmse(test.mine_y, preds.test.L2)
> test.rmse.L2
[1] 13.73793
```

Figure 33: Checking Performance of Testing set

Comparing Models

```
> Final_t222
       RMSE Train Value RMSE Test Value
Ridge          13.15157         14.01189
Lasso          12.85370         13.73793
```

Figure 34: RMSE Values

Using the RMSE values, we compared the models that we have worked on during this assignment so far and found that Lasso Regression Model is good one when compared to the Ridge Regression Model, since from both of these models the Lasso Regression Model has the lowest rmse value which is 13.7 (& 12.8). Hence Lasso Regression Model is better than the Ridge Regression Model. This is as I expected since despite Ridge model being most popular but Lasso Model tends to do well as compared to the Ridge Model (Oleszak, 2019).

Step-wise Regression

The step-wise approach involves backward and forward selection methodology. It is an iterative approach to add variable in a step wise manner so as to achieve the optimum model. The performance of each step is measured using the AIC value which estimates the prediction error (Hayes, 2022).

```
> model_back <- step(lm(Grad.Rate~.,data=train.mine), direction
 = 'backward')
Start:  AIC=2743.19
Grad.Rate ~ Private + Apps + Accept + Enroll + Top10perc + Top25
perc +
    F.Undergrad + P.Undergrad + Outstate + Room.Board + Books +
    Personal + PhD + Terminal + S.F.Ratio + perc.alumni + Expend

              Df Sum of Sq   RSS    AIC
- Top10perc    1       2.6 79440 2741.2
- Terminal     1     100.5 79538 2741.9
- Books        1     177.5 79614 2742.4
<none>                    79437 2743.2
- S.F.Ratio    1     348.8 79786 2743.6
- Accept       1     418.6 79856 2744.0
- Personal     1     427.2 79864 2744.1
- PhD          1     462.3 79899 2744.3
- Room.Board   1     616.9 80054 2745.4
- Enroll       1     677.0 80114 2745.8
- Expend       1     901.1 80338 2747.3
- P.Undergrad  1     968.1 80405 2747.8
- Top25perc    1    1088.7 80526 2748.6
- F.Undergrad  1    1170.0 80607 2749.1
- Private      1    1280.2 80717 2749.9
- Outstate     1    1446.2 80883 2751.0
- Apps         1    2332.3 81769 2756.9
- perc.alumni  1    4192.3 83629 2769.1
```

Figure 35: Step 1

```
Step:  AIC=2741.2
Grad.Rate ~ Private + Apps + Accept + Enroll + Top25perc + F.Und
ergrad +
    P.Undergrad + Outstate + Room.Board + Books + Personal +
    PhD + Terminal + S.F.Ratio + perc.alumni + Expend

             Df Sum of Sq   RSS    AIC
- Terminal    1     105.3 79545 2739.9
- Books       1     175.6 79615 2740.4
<none>                    79440 2741.2
- S.F.Ratio   1     349.4 79789 2741.6
- Personal    1     426.4 79866 2742.1
- PhD         1     481.8 79921 2742.5
- Accept      1     490.2 79930 2742.5
- Room.Board  1     614.4 80054 2743.4
- Enroll      1     718.3 80158 2744.1
- Expend      1     944.7 80384 2745.6
- P.Undergrad 1     973.9 80413 2745.8
- F.Undergrad 1    1195.8 80635 2747.3
- Private     1    1287.9 80727 2747.9
- Outstate    1    1467.4 80907 2749.1
- Apps        1    2613.5 82053 2756.8
- Top25perc   1    3050.7 82490 2759.7
- perc.alumni 1    4213.9 83654 2767.3
```

Figure 36: Step 2

```
Step:  AIC=2739.92
Grad.Rate ~ Private + Apps + Accept + Enroll + Top25perc + F.Und
ergrad +
    P.Undergrad + Outstate + Room.Board + Books + Personal +
    PhD + S.F.Ratio + perc.alumni + Expend

             Df Sum of Sq   RSS    AIC
- Books       1     207.0 79752 2739.3
<none>                    79545 2739.9
- S.F.Ratio   1     347.1 79892 2740.3
- Personal    1     413.6 79958 2740.7
- PhD         1     431.4 79976 2740.9
- Accept      1     507.7 80053 2741.4
- Room.Board  1     555.4 80100 2741.7
- Enroll      1     726.5 80271 2742.9
- Expend      1     959.9 80505 2744.4
- P.Undergrad 1     995.7 80541 2744.7
- F.Undergrad 1    1220.2 80765 2746.2
- Private     1    1393.2 80938 2747.3
- Outstate    1    1423.2 80968 2747.6
- Apps        1    2697.2 82242 2756.0
- Top25perc   1    2996.2 82541 2758.0
- perc.alumni 1    4140.4 83685 2765.5
```

Figure 37: Step 3

```
Step:  AIC=2739.33
Grad.Rate ~ Private + Apps + Accept + Enroll + Top25perc + F.Und
ergrad +
    P.Undergrad + Outstate + Room.Board + Personal + PhD + S.F.R
atio +
    perc.alumni + Expend

             Df Sum of Sq   RSS    AIC
<none>                    79752 2739.3
- S.F.Ratio   1     311.6 80063 2739.4
- PhD         1     429.6 80181 2740.2
- Room.Board  1     504.4 80256 2740.8
- Accept      1     536.4 80288 2741.0
- Personal    1     608.7 80361 2741.5
- Enroll      1     751.6 80503 2742.4
- P.Undergrad 1     997.5 80749 2744.1
- Expend      1    1041.5 80793 2744.4
- F.Undergrad 1    1240.4 80992 2745.7
- Private     1    1342.3 81094 2746.4
- Outstate    1    1486.1 81238 2747.4
- Apps        1    2730.7 82483 2755.6
- Top25perc   1    2834.8 82587 2756.3
- perc.alumni 1    4219.0 83971 2765.3
```

Figure 38: Step 4

We choose the backward selection method. In this initially all the predictor variables are included that mean all 17 variables. Initially the AIC value was 2743.19 but after multiple iterations as mentioned in each picture (above) the AIC value improved to 2739.33 and in final iteration you can see reduced number of variables i.e. 14. For fitting the model train data was used. Private, Apps, Accept, Enroll, Top25perc, F.Undergrad, P.Undergrad, Outstate, Room.Board, Personal, PhD, S.F.Ratio, perc.alumni and the Expend were the final variables.

```
> model_back11 = rmse(train.mine_y, model_back1)
> model_back11
[1] 12.11911
```

Figure39: RMSE train value for step wise model

```
> model_back22
[1] 20.55337
```

Figure 17: RMSE test value for step wise model

I have used the rmse function and the prediction function to calculate the rmse value of train and test set using this model. The rmse value for the train set is 12.1 whereas the rmse value for the test set is 20.55.

```
> Final_t2222
          RMSE Train Value RMSE Test Value
Ridge              13.15157         14.01189
Lasso              12.85370         13.73793
Step.Wise          12.11911         20.55337
```

Figure 40: RMSE values for all three models

Using the rmse value we have compared all three models in the table presented in the above picture and now we can clearly see that the Lasso Regression is actually the best model since the lowest rmse value i.e. 13.7 belongs to the Lasso Regression Model.

Hence **my preference will be the Lasso Regression Model** since we can clearly note that the Lasso Regression Model reduces coefficients towards zero. Hence using the Lasso Regression Model a more robust and flexible model could be created for prediction.

In part II, we created multiple models by first training the model and testing it. Multiple models include Ridge Regression Model, Least Absolute Shrinkage and Selection Operator (Lasso) and the Step-wise Regression Model. In each model we checked if there is any overfitting or not. Later we compared each of the model using root mean square error values. The model with the least root mean square error value was selected.

This projected helped me understand regularization in a much more clear sense and how it could be implemented not only using Lasso or Ridge model but also the Step-wise model. For all of the calculations and practical during this task, the College dataset was used and we found that Lasso Regression Model is the best among all considered models.

Reference:

Below are some links, from where I took help conceptually and for the purpose of correct code.

How to change correlation text size in ggpairs(). Stackoverflow.
Data retrieved on May 4th 2022 using below URL,
https://stackoverflow.com/questions/8599685/how-to-change-correlation-text-size-in-ggpairs

Confusion Matrix in R | A Complete Guide. JournalDev.
Data retrieved on May 4th 2022 using below URL,
https://www.journaldev.com/46732/confusion-matrix-in-r#:~:text=A%20confusion%20matrix%20in%20R,will%20represent%20the%20actual%20values.&text=In%20most%20of%20the%20recourses,%C3%972%20matrix%20in%20R.

College: U.S. News and World Report's College Data. Rdrr.io
Data retrieved on May 4th 2022 using below URL,
https://rdrr.io/cran/ISLR/man/College.html

Hastie, Trevor. Qian, Junyang. Tay, Kenneth. (2021 November 01st). An Introduction to glmnet. Stanford.edu
Data retrieved on May 10th 2022 using below URL,
https://glmnet.stanford.edu/articles/glmnet.html#:~:text=vertical%20dotted%20lines.-,lambda.,standard%20error%20of%20the%20minimum.

Oleszak, Michal. (2019 November 11th). Regularization Tutorial: Ridge, Lasso and Elastic Net. Datacamp.
Data retrieved on May 11th 2022 using below URL,
https://www.datacamp.com/tutorial/tutorial-ridge-lasso-elastic-net

Hayes, Adam. (2022 January 10th). Stepwise Regression. Investopedia.com. Data retrieved on May 11th 2022 using below URL, https://www.investopedia.com/terms/s/stepwise-regression.asp