Homework #7

2024-10-09

Question 10.1 Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too)

```
#housekeeping
library(pacman)
pacman::p_load(rio, tree, randomForest, caret, car, GGally, pROC)

crime <- import("D:/…/uscrime.txt")
str(crime)
```

```
## 'data.frame':    47 obs. of  16 variables:
##  $ M     : num  15.1 14.3 14.2 13.6 14.1 12.1 12.7 13.1 15.7 14 ...
##  $ So    : int  1 0 1 0 0 0 1 1 1 0 ...
##  $ Ed    : num  9.1 11.3 8.9 12.1 12.1 11 11.1 10.9 9 11.8 ...
##  $ Po1   : num  5.8 10.3 4.5 14.9 10.9 11.8 8.2 11.5 6.5 7.1 ...
##  $ Po2   : num  5.6 9.5 4.4 14.1 10.1 11.5 7.9 10.9 6.2 6.8 ...
##  $ LF    : num  0.51 0.583 0.533 0.577 0.591 0.547 0.519 0.542 0.553 0.632
...
##  $ M.F   : num  95 101.2 96.9 99.4 98.5 ...
##  $ Pop   : int  33 13 18 157 18 25 4 50 39 7 ...
##  $ NW    : num  30.1 10.2 21.9 8 3 4.4 13.9 17.9 28.6 1.5 ...
##  $ U1    : num  0.108 0.096 0.094 0.102 0.091 0.084 0.097 0.079 0.081 0.1
...
##  $ U2    : num  4.1 3.6 3.3 3.9 2 2.9 3.8 3.5 2.8 2.4 ...
##  $ Wealth: int  3940 5570 3180 6730 5780 6890 6200 4720 4210 5260 ...
##  $ Ineq  : num  26.1 19.4 25 16.7 17.4 12.6 16.8 20.6 23.9 17.4 ...
##  $ Prob  : num  0.0846 0.0296 0.0834 0.0158 0.0414 ...
##  $ Time  : num  26.2 25.3 24.3 29.9 21.3 ...
##  $ Crime : int  791 1635 578 1969 1234 682 963 1555 856 705 ...
```

```
summary(crime)
```

```
##        M               So               Ed             Po1
##  Min.   :11.90   Min.   :0.0000   Min.   : 8.70   Min.   : 4.50
##  1st Qu.:13.00   1st Qu.:0.0000   1st Qu.: 9.75   1st Qu.: 6.25
##  Median :13.60   Median :0.0000   Median :10.80   Median : 7.80
##  Mean   :13.86   Mean   :0.3404   Mean   :10.56   Mean   : 8.50
##  3rd Qu.:14.60   3rd Qu.:1.0000   3rd Qu.:11.45   3rd Qu.:10.45
##  Max.   :17.70   Max.   :1.0000   Max.   :12.20   Max.   :16.60
##       Po2              LF              M.F              Pop
##  Min.   : 4.100   Min.   :0.4800   Min.   : 93.40   Min.   :  3.00
```

```
##  1st Qu.: 5.850   1st Qu.:0.5305   1st Qu.: 96.45   1st Qu.: 10.00
##  Median : 7.300   Median :0.5600   Median : 97.70   Median : 25.00
##  Mean   : 8.023   Mean   :0.5612   Mean   : 98.30   Mean   : 36.62
##  3rd Qu.: 9.700   3rd Qu.:0.5930   3rd Qu.: 99.20   3rd Qu.: 41.50
##  Max.   :15.700   Max.   :0.6410   Max.   :107.10   Max.   :168.00
##        NW              U1                U2             Wealth
##  Min.   : 0.20   Min.   :0.07000   Min.   :2.000   Min.   :2880
##  1st Qu.: 2.40   1st Qu.:0.08050   1st Qu.:2.750   1st Qu.:4595
##  Median : 7.60   Median :0.09200   Median :3.400   Median :5370
##  Mean   :10.11   Mean   :0.09547   Mean   :3.398   Mean   :5254
##  3rd Qu.:13.25   3rd Qu.:0.10400   3rd Qu.:3.850   3rd Qu.:5915
##  Max.   :42.30   Max.   :0.14200   Max.   :5.800   Max.   :6890
##       Ineq            Prob              Time            Crime
##  Min.   :12.60   Min.   :0.00690   Min.   :12.20   Min.   : 342.0
##  1st Qu.:16.55   1st Qu.:0.03270   1st Qu.:21.60   1st Qu.: 658.5
##  Median :17.60   Median :0.04210   Median :25.80   Median : 831.0
##  Mean   :19.40   Mean   :0.04709   Mean   :26.60   Mean   : 905.1
##  3rd Qu.:22.75   3rd Qu.:0.05445   3rd Qu.:30.45   3rd Qu.:1057.5
##  Max.   :27.60   Max.   :0.11980   Max.   :44.00   Max.   :1993.0
```

The data seems to have a wide range of scales—some variables are as small as the hundredths decimal place, while others, including the response variable, are in the thousands.

There is no missing data, but potential outliers might be present in variables like POP (upper end), NW (lower end), Prob (lower end), and Crime (top two values).

Next, using the tree library, carry out a tree regression, and use cross-validation to determine the ideal tree size.

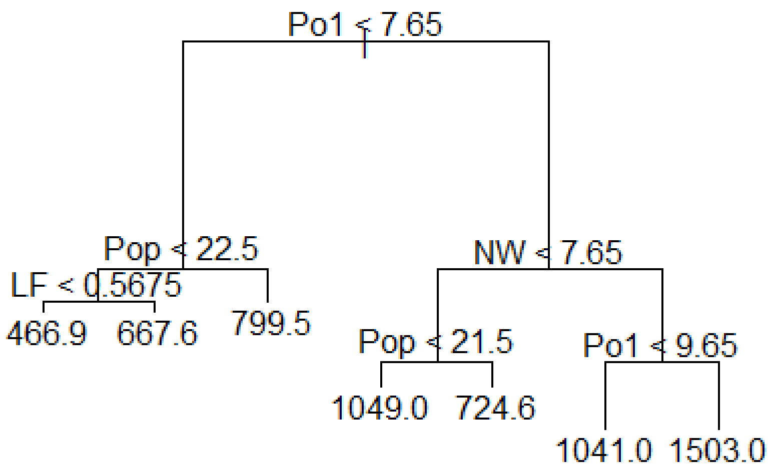
```
crimetree<-tree(Crime~.,data=crime)
summary(crimetree)

## 
## Regression tree:
## tree(formula = Crime ~ ., data = crime)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7 
## Residual mean deviance:  47390 = 1896000 / 40 
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

There are 7 terminal nodes, with a residual mean deviance of 47,390.

Let’s visualize the tree to examine the interactions and assess whether the splits are logical.

```
plot(crimetree)
text(crimetree ,pretty =0)
```




```
crimetree

## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 47 6881000  905.1
##    2) Po1 < 7.65 23  779200  669.6
##      4) Pop < 22.5 12  243800  550.5
##        8) LF < 0.5675 7   48520  466.9 *
##        9) LF > 0.5675 5   77760  667.6 *
##      5) Pop > 22.5 11  179500  799.5 *
##    3) Po1 > 7.65 24 3604000 1131.0
##      6) NW < 7.65 10  557600  886.9
##       12) Pop < 21.5 5  146400 1049.0 *
##       13) Pop > 21.5 5  147800  724.6 *
##      7) NW > 7.65 14 2027000 1305.0
##       14) Po1 < 9.65 6  170800 1041.0 *
##       15) Po1 > 9.65 8 1125000 1503.0 *
```

Po1 is identified as the most significant variable, which is why it is used for the initial split.

Now, let's compute both the R^2 and adjusted R^2 values for the tree model to evaluate its performance.

```
SST<-sum((crime$Crime-mean(crime$Crime))^2)
SSE<-sum((crime$Crime-predict(crimetree,newdata=crime))^2)
SST
```

```
## [1] 6880928
```

```
SSE
```

```
## [1] 1895722
```

```
r2<-1-SSE/SST
r2
```

```
## [1] 0.7244962
```

```
n=nrow(crime)
k=7#number of terminal nodes
adjR2<-1-(1-r2)*(n-1)/(n-k-1)
adjR2
```

```
## [1] 0.6750468
```

Sum of Square Total: 6880928 Sum of Square Error: 1895722 R^2: 0.7244962 Adjusted R^2: 0.6750468

Let’s try building a more refined tree model, aiming to reduce the sum of squared residuals and improve overall performance.

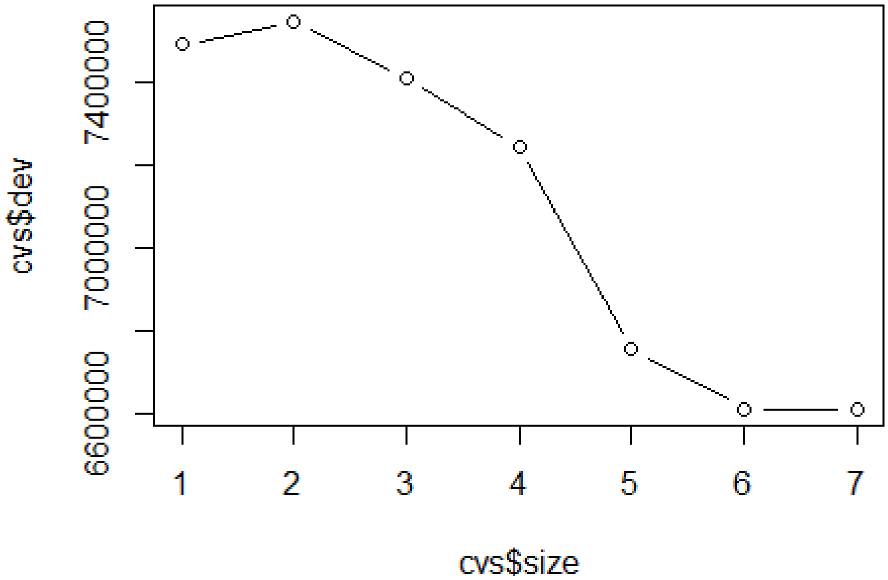
```
sample(1:1000,1)
```

```
## [1] 562
```

```
set.seed(123)
cvs<-cv.tree(crimetree, FUN = prune.tree, K=5)
cvs
```

```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1] 6609119 6609119 6752943 7245720 7410677 7547625 7491120
##
## $k
## [1]      -Inf  117534.9  263412.9  355961.8  731412.1 1019362.7 2497521.7
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cvs$size ,cvs$dev ,type="b")
```



A 5-fold cross-validation shows that the optimal tree size is 2, as it results in the lowest error, with a deviance of 6,609,119.

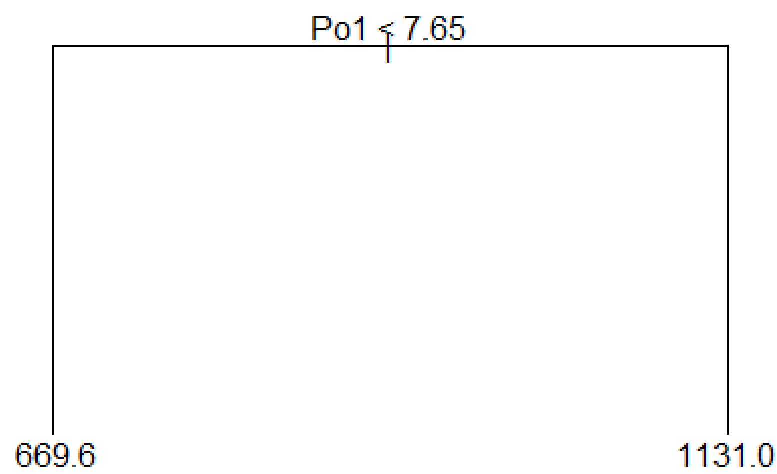
Since we now know that the optimal number of terminal nodes is 2, we can set this as a limit in the tree.control option to prevent overfitting if the tree exceeds 2 terminal nodes.

To create the recommended pruned tree, we should set the minimum number of observations at each node to 23. Let's rebuild the tree with mincut = 23, which will give us the desired 2 terminal nodes.

```
crimetree2<-tree(Crime~.,data=crime, control = tree.control(nobs = 47, mincut
= 23))
crimetree2

## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 47 6881000  905.1
##   2) Po1 < 7.65 23  779200  669.6 *
##   3) Po1 > 7.65 24 3604000 1131.0 *

plot(crimetree2)
text(crimetree2 ,pretty =0)
```



This approach results in a tree with 2 terminal nodes (or leaves).

Now, let's evaluate the in-sample performance of the pruned tree to see how well it fits the training data.

```
summary(crimetree2)

##
## Regression tree:
## tree(formula = Crime ~ ., data = crime, control = tree.control(nobs = 47,
##     mincut = 23))
## Variables actually used in tree construction:
## [1] "Po1"
## Number of terminal nodes:  2
## Residual mean deviance:  97410 = 4383000 / 45
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -622.800 -193.200   -5.609    0.000  147.300  862.200
```

The in-sample residual mean deviance has now increased to 97,410, which is double what it was with 7 terminal nodes. This is expected, as the simpler model has fewer splits and thus explains less of the variance in the training data, leading to a higher residual deviance.

Let's now calculate the R^2 value for the pruned model to assess its explanatory power.

```
SSE2<-sum((crime$Crime-predict(crimetree2,newdata=crime))^2)
SSE2
```

```
## [1] 4383406
r2<-1-SSE2/SST
r2
## [1] 0.3629629
n=nrow(crime)
k=2#number of terminal nodes
adjR2<-1-(1-r2)*(n-1)/(n-k-1)
adjR2
## [1] 0.3340067
```

Sum of Square Error: 4383406 R^2: 0.3629629 Adjusted R^2: 0.3340067

The in-sample adjusted R^2 has dropped significantly, as expected for a simpler model. However, we anticipate that the cross-validated R^2 will be lower than that of the full model.

Next, we'll use the randomForest package to build a random forest model. For the number of variables to split on at each node (mtry), we will use the recommended value of sqrt(15), which rounds to 4. Let's proceed with mtry = 4 to train the random forest model.

```
library(randomForest)
rm<-randomForest(Crime~., data=crime, mtry=4)
rm
##
## Call:
##  randomForest(formula = Crime ~ ., data = crime, mtry = 4)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 86008.71
##                     % Var explained: 41.25
```

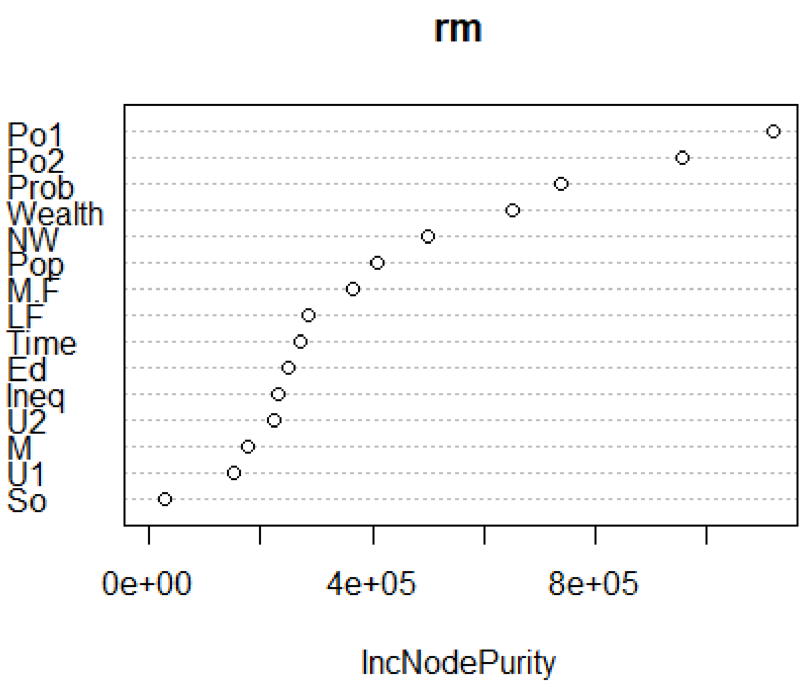
We observe that the random forest model's R^2 is 41.25%, which is an improvement over the selected tree model's R^2 of 36.2%.

Next, let's evaluate the importance of the predictors using the Importance function and visualize it with VarImpPlot to identify the most influential variables in the model.

```
importance(rm)
##        IncNodePurity
## M          175616.94
## So          29094.31
## Ed         247982.57
## Po1       1117782.90
## Po2        956521.24
## LF         284317.91
```

```
## M.F         366642.57
## Pop         409715.40
## NW          500903.62
## U1          152274.19
## U2          222715.63
## Wealth      650623.69
## Ineq        230891.71
## Prob        739138.31
## Time        271831.04
```

```r
varImpPlot(rm)
```



In Random Forest regression, the increase in node purity represents the average increase in RSS across all trees from splitting on a given variable. Once again, we find that Po1 is the most important variable.

Now, let's apply cross-validation to test different mtry levels and see if we can improve the model's performance further. By tuning mtry, we aim to find the optimal number of variables to split at each node, potentially boosting the model's accuracy.

```r
cvrm<-train(Crime~., data=crime,
            trControl=trainControl(method = 'cv', number = 5, savePredictions
= TRUE),
            tuneGrid=expand.grid(mtry=1:15))
cvrm
```

```
## Random Forest
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 38, 38, 37, 37, 38
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    1    297.6471  0.4652406  231.4440
##    2    282.9032  0.5010611  216.1343
##    3    283.1173  0.4807975  217.4543
##    4    283.9964  0.4744617  215.1611
##    5    285.3789  0.4640639  216.1700
##    6    288.9224  0.4495705  218.3318
##    7    294.8726  0.4486273  222.2781
##    8    291.2056  0.4440580  217.0495
##    9    296.8684  0.4332951  221.0979
##   10    300.0878  0.4218781  224.8335
##   11    303.5372  0.4145929  226.3279
##   12    296.8611  0.4288516  221.5051
##   13    302.1244  0.4221979  225.4605
##   14    301.0151  0.4124426  226.3083
##   15    304.2244  0.4037450  227.9303
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
SSErm<-sum((cvrm$pred[,2]-cvrm$pred[,1])^2)
SSErm
```

```
## [1] 62272516
```

```
SSTrm<-sum((cvrm$pred[,2]-mean(cvrm$pred[,2]))^2)
SSTrm
```

```
## [1] 103213915
```

```
r2<-1-(SSErm/SSTrm)
r2
```

```
## [1] 0.3966655
```

Now, let's try reducing the number of trees built from 500 to 400 and compare the cross-validated performance. This could help prevent overfitting while maintaining model accuracy.

```
cvrm2<-train(Crime~., data=crime,
             trControl=trainControl(method = 'cv', number = 5, savePredictions
```

```
= TRUE),
              tuneGrid=expand.grid(mtry=1:15), ntree=400)
cvrm2

## Random Forest
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 38, 38, 36, 37, 39
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    1    307.5985  0.5127369  234.3739
##    2    296.6861  0.5200015  226.0615
##    3    291.4671  0.5447273  223.0628
##    4    287.2255  0.5245885  217.5861
##    5    292.0182  0.5071527  219.0340
##    6    288.0024  0.5181279  214.9951
##    7    288.1204  0.5028579  213.7320
##    8    284.8795  0.5177826  212.3652
##    9    287.4861  0.5102370  215.0251
##   10    292.9740  0.5069761  218.3567
##   11    292.1732  0.5042289  218.0302
##   12    293.4688  0.4953211  218.1189
##   13    296.2153  0.4810546  220.7496
##   14    294.0484  0.5042943  217.9525
##   15    293.2615  0.4949142  216.6381
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 8.

SSErm<-sum((cvrm2$pred[,2]-cvrm2$pred[,1])^2)
SSErm

## [1] 64442310

SSTrm<-sum((cvrm2$pred[,2]-mean(cvrm2$pred[,2]))^2)
SSTrm

## [1] 103213915

r2<-1-(SSErm/SSTrm)
r2

## [1] 0.3756432
```

We've observed a slight improvement in cross-validated performance by reducing the number of trees to 400.

Let's now reduce the number of trees further, from 400 to 300, and compare the cross-validated performance to see if this helps prevent overfitting while maintaining or improving model accuracy.

```
set.seed(478)
cvrm3<-train(Crime~., data=crime,
            trControl=trainControl(method = 'cv', number = 5, savePredictions
= TRUE),
            tuneGrid=expand.grid(mtry=1:15), ntree=300)
cvrm3
```

```
## Random Forest
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 36, 39, 35, 39, 39
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    1    291.9857  0.5106459  228.5653
##    2    275.7679  0.5424265  212.1414
##    3    270.5985  0.5616933  208.8549
##    4    268.0824  0.5812400  206.5469
##    5    267.5825  0.5652140  205.2438
##    6    266.1166  0.5824725  205.7723
##    7    269.1086  0.5597965  206.6419
##    8    264.8420  0.5825996  206.4396
##    9    267.0515  0.5680341  207.4476
##   10    268.2813  0.5673642  208.9184
##   11    274.4136  0.5351932  214.6021
##   12    268.8706  0.5578022  208.3199
##   13    270.7862  0.5528106  205.5508
##   14    274.7376  0.5360487  213.1964
##   15    269.8778  0.5659305  209.4186
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 8.
```

```
SSErm<-sum((cvrm3$pred[,2]-cvrm3$pred[,1])^2)
SSErm
```

```
## [1] 57281709
```

```
SSTrm<-sum((cvrm3$pred[,2]-mean(cvrm3$pred[,2]))^2)
SSTrm
```

```
## [1] 103213915
```

```
r2<-1-(SSErm/SSTrm)
r2

## [1] 0.4450195
```

Since reducing the number of trees to 300 didn't lead to any performance improvement, we'll stick with the earlier model, cvrm2, which was built using 400 trees. This model strikes a better balance between accuracy and overfitting.

Question 10.2 Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

A situation where a logistic regression model would be appropriate is predicting whether a customer will default on a loan (yes/no). Since the outcome is binary, logistic regression is a suitable approach.

5 potential predictors:

Income: The customer's annual income, which can help assess their ability to repay the loan.

Credit Score: A score indicating the customer's creditworthiness, an important factor in predicting loan defaults.

Debt-to-Income Ratio: The ratio of a customer's monthly debt payments to their monthly income.

Employment Status: Whether the customer is employed, unemployed, or self-employed, which can impact their financial stability.

Loan Amount: The total amount of the loan, as larger loans might be harder to repay depending on other factors.

Question 10.3.1 Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machinelearning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.

```
credit<-import("D:/…/germancredit.txt")
str(credit)

## 'data.frame':    1000 obs. of  21 variables:
##  $ V1 : chr  "A11" "A12" "A14" "A11" ...
##  $ V2 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ V3 : chr  "A34" "A32" "A34" "A32" ...
##  $ V4 : chr  "A43" "A43" "A46" "A42" ...
```

```
##  $ V5 : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ V6 : chr  "A65" "A61" "A61" "A61" ...
##  $ V7 : chr  "A75" "A73" "A74" "A74" ...
##  $ V8 : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ V9 : chr  "A93" "A92" "A93" "A93" ...
##  $ V10: chr  "A101" "A101" "A101" "A103" ...
##  $ V11: int  4 2 3 4 4 4 4 2 4 2 ...
##  $ V12: chr  "A121" "A121" "A121" "A122" ...
##  $ V13: int  67 22 49 45 53 35 53 35 61 28 ...
##  $ V14: chr  "A143" "A143" "A143" "A143" ...
##  $ V15: chr  "A152" "A152" "A152" "A153" ...
##  $ V16: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ V17: chr  "A173" "A173" "A172" "A173" ...
##  $ V18: int  1 1 2 2 2 2 1 1 1 1 ...
##  $ V19: chr  "A192" "A191" "A191" "A191" ...
##  $ V20: chr  "A201" "A201" "A201" "A201" ...
##  $ V21: int  1 2 1 1 2 1 1 1 1 2 ...
```

```
summary(credit)
```

```
##       V1                  V2            V3                  V4
##  Length:1000        Min.   : 4.0   Length:1000        Length:1000
##  Class :character   1st Qu.:12.0   Class :character   Class :character
##  Mode  :character   Median :18.0   Mode  :character   Mode  :character
##                     Mean   :20.9
##                     3rd Qu.:24.0
##                     Max.   :72.0
##        V5              V6                  V7                  V8
##  Min.   :  250   Length:1000        Length:1000        Min.   :1.000
##  1st Qu.: 1366   Class :character   Class :character   1st Qu.:2.000
##  Median : 2320   Mode  :character   Mode  :character   Median :3.000
##  Mean   : 3271                                         Mean   :2.973
##  3rd Qu.: 3972                                         3rd Qu.:4.000
##  Max.   :18424                                         Max.   :4.000
##       V9                V10                  V11            V12
##  Length:1000        Length:1000        Min.   :1.000   Length:1000
##  Class :character   Class :character   1st Qu.:2.000   Class :character
##  Mode  :character   Mode  :character   Median :3.000   Mode  :character
##                                        Mean   :2.845
##                                        3rd Qu.:4.000
##                                        Max.   :4.000
##       V13            V14                  V15                  V16
##  Min.   :19.00   Length:1000        Length:1000        Min.   :1.000
##  1st Qu.:27.00   Class :character   Class :character   1st Qu.:1.000
##  Median :33.00   Mode  :character   Mode  :character   Median :1.000
##  Mean   :35.55                                         Mean   :1.407
##  3rd Qu.:42.00                                         3rd Qu.:2.000
##  Max.   :75.00                                         Max.   :4.000
##      V17                 V18             V19                  V20
##  Length:1000        Min.   :1.000   Length:1000        Length:1000
```

```
##   Class :character   1st Qu.:1.000   Class :character   Class :character  
##   Mode  :character   Median :1.000   Mode  :character   Mode  :character  
##                      Mean   :1.155                                        
##                      3rd Qu.:1.000                                        
##                      Max.   :2.000                                        
##       V21     
##   Min.   :1.0  
##   1st Qu.:1.0  
##   Median :1.0  
##   Mean   :1.3  
##   3rd Qu.:2.0  
##   Max.   :2.0
```

One of the variables, V5 (credit amount), operates on a very different scale compared to the other variables. Thankfully, since we are using regression, this difference in scale won't disrupt the model. Additionally, there are no missing values in the data.

We will relabel the response variable to 0 and 1, where 0 represents 'Good' and 1 represents 'Bad,' with 'Bad' being treated as the positive class.

```
credit$V21<-ifelse(credit$V21==1,0,ifelse(credit$V21==2,1,credit$V21))
summary(credit$V21)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##     0.0     0.0     0.0     0.3     1.0     1.0
```

```
prop.table(table(credit$V21))
```

```
## 
##   0   1 
## 0.7 0.3
```

The data shows that 70% of the cases are good credit risks, while 30% are bad.

Next, we'll split the dataset into 70% for training (roughly 700 observations) and 30% for testing. To ensure the splits maintain the same proportions of the response variable (good/bad credit risks), we will use stratified sampling. Additionally, the data will be randomized before splitting to avoid any ordering bias.

```
head(credit)
```

```
##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17
V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173
1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173
1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172
2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173
2
```

```
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173
2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172
2
##    V19  V20 V21
## 1 A192 A201   0
## 2 A191 A201   1
## 3 A191 A201   0
## 4 A191 A201   0
## 5 A191 A201   1
## 6 A192 A201   0
```

```
set.seed(1)
credit2<-credit[sample(1:nrow(credit)),]
head(credit2)
```

```
##      V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16
V17 V18
## 836 A11 12 A30 A40 1082 A61 A73  4 A93 A101   4 A123  48 A141 A152   2
A173   1
## 679 A11 24 A32 A43 2384 A61 A75  4 A93 A101   4 A121  64 A141 A151   1
A172   1
## 129 A12 12 A34 A41 1860 A61 A71  4 A93 A101   2 A123  34 A143 A152   2
A174   1
## 930 A11 12 A33 A40 1344 A61 A73  4 A93 A101   2 A121  43 A143 A152   2
A172   2
## 509 A14 24 A32 A43 1413 A61 A73  4 A94 A101   2 A122  28 A143 A152   1
A173   1
## 471 A12 24 A32 A43 3092 A62 A72  3 A94 A101   2 A123  22 A143 A151   1
A173   1
##      V19  V20 V21
## 836 A191 A201   1
## 679 A191 A201   0
## 129 A192 A201   0
## 930 A191 A201   0
## 509 A191 A201   0
## 471 A192 A201   1
```

The dataset rows have now been randomized.

We can proceed by creating the building (training) set, ensuring that 70% of the data is allocated for model training while maintaining the stratified proportions of the response variable.

```
credit2$V21<-as.factor(credit2$V21)
sample(1:nrow(credit2),1)
```

```
## [1] 752
```

```
set.seed(266)
buildIndex <- createDataPartition(credit2$V21, p = .7, list=FALSE, times = 1)
```

```r
build<-credit2[buildIndex,]
str(build)#confirm number of records is 70% of entire data set
```

```
## 'data.frame':    700 obs. of  21 variables:
##  $ V1 : chr  "A11" "A11" "A12" "A14" ...
##  $ V2 : int  12 24 12 24 18 18 30 24 15 24 ...
##  $ V3 : chr  "A30" "A32" "A34" "A32" ...
##  $ V4 : chr  "A40" "A43" "A41" "A43" ...
##  $ V5 : int  1082 2384 1860 1413 2515 2427 4811 1442 874 3621 ...
##  $ V6 : chr  "A61" "A61" "A61" "A61" ...
##  $ V7 : chr  "A73" "A75" "A71" "A73" ...
##  $ V8 : int  4 4 4 4 3 4 2 4 4 2 ...
##  $ V9 : chr  "A93" "A93" "A93" "A94" ...
##  $ V10: chr  "A101" "A101" "A101" "A101" ...
##  $ V11: int  4 4 2 2 4 2 4 4 1 4 ...
##  $ V12: chr  "A123" "A121" "A123" "A122" ...
##  $ V13: int  48 64 34 28 43 42 24 23 24 31 ...
##  $ V14: chr  "A141" "A141" "A143" "A143" ...
##  $ V15: chr  "A152" "A151" "A152" "A152" ...
##  $ V16: int  2 1 2 1 1 2 1 2 1 2 ...
##  $ V17: chr  "A173" "A172" "A174" "A173" ...
##  $ V18: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ V19: chr  "A191" "A191" "A192" "A191" ...
##  $ V20: chr  "A201" "A201" "A201" "A201" ...
##  $ V21: Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 2 1 2 ...
```

Now, let’s verify if the proportion of 0s and 1s in the training set aligns with the original 70% “Good” (0) and 30% “Bad” (1) split observed in the full dataset. This check will confirm that the stratification worked correctly during the data splitting process.

```r
prop.table(table(build$V21))
```

```
## 
##   0   1 
## 0.7 0.3
```

The stratified partition of the build set has been successfully created.

Now, we’ll create a validation set using the remaining 30% of the data, ensuring that this set also maintains the same stratified proportions of the response variable (0s and 1s). This will be used for evaluating model performance.

```r
test<-credit2[-buildIndex,]
str(test)#confirm number of records is 50% of test and validation data
```

```
## 'data.frame':    300 obs. of  21 variables:
##  $ V1 : chr  "A11" "A12" "A14" "A12" ...
##  $ V2 : int  12 24 24 9 18 6 6 12 27 18 ...
##  $ V3 : chr  "A33" "A32" "A32" "A31" ...
##  $ V4 : chr  "A40" "A43" "A43" "A41" ...
##  $ V5 : int  1344 3092 999 5129 2404 368 1068 385 2570 1795 ...
```

```
##  $ V6 : chr  "A61" "A62" "A65" "A61" ...
##  $ V7 : chr  "A73" "A72" "A75" "A75" ...
##  $ V8 : int  4 3 4 2 2 4 4 4 3 3 ...
##  $ V9 : chr  "A93" "A94" "A93" "A92" ...
##  $ V10: chr  "A101" "A101" "A101" "A101" ...
##  $ V11: int  2 2 2 4 2 4 4 3 3 4 ...
##  $ V12: chr  "A121" "A123" "A123" "A124" ...
##  $ V13: int  43 22 25 74 26 38 28 58 21 48 ...
##  $ V14: chr  "A143" "A143" "A143" "A141" ...
##  $ V15: chr  "A152" "A151" "A152" "A153" ...
##  $ V16: int  2 1 2 1 2 1 1 4 1 2 ...
##  $ V17: chr  "A172" "A173" "A173" "A174" ...
##  $ V18: int  2 1 1 2 1 1 2 1 1 1 ...
##  $ V19: chr  "A191" "A192" "A191" "A192" ...
##  $ V20: chr  "A201" "A201" "A201" "A201" ...
##  $ V21: Factor w/ 2 levels "0","1": 1 2 1 2 1 1 1 1 2 1 ...
```

Let's now verify that the proportions of 0s and 1s in the test set match the original 70% "Good" (0) and 30% "Bad" (1) split observed in the full dataset. This will confirm that the stratification in the test set was applied correctly.

```
prop.table(table(test$V21))

## 
##   0   1 
## 0.7 0.3
```

The test set successfully matches the original 70-30% proportions.

Now, let's proceed with fitting a generalized linear model (GLM) using all the features on the build (training) set. This model will help us analyze the relationship between the predictors and the response variable.

```
glm1<-glm(V21~., data=build, family = binomial(link='logit'))
summary(glm1)

## 
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = build)
## 
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -6.093e-01  1.456e+00  -0.418 0.675662    
## V1A12       -2.537e-01  2.695e-01  -0.941 0.346555    
## V1A13       -7.945e-01  4.397e-01  -1.807 0.070779 .  
## V1A14       -1.920e+00  2.984e-01  -6.434 1.24e-10 ***
## V2           4.573e-02  1.226e-02   3.731 0.000191 ***
## V3A31        3.289e-01  6.810e-01   0.483 0.629116    
## V3A32       -6.354e-01  5.472e-01  -1.161 0.245570    
## V3A33       -9.101e-01  6.051e-01  -1.504 0.132595    
## V3A34       -1.522e+00  5.684e-01  -2.678 0.007404 ** 
```

```
## V4A41       -1.346e+00  4.672e-01  -2.881 0.003967 **
## V4A410      -1.405e+00  1.013e+00  -1.387 0.165519
## V4A42       -7.308e-01  3.308e-01  -2.209 0.027148 *
## V4A43       -1.162e+00  3.188e-01  -3.646 0.000266 ***
## V4A44       -9.327e-01  9.217e-01  -1.012 0.311606
## V4A45       -3.024e-01  7.040e-01  -0.430 0.667494
## V4A46        3.180e-01  5.185e-01   0.613 0.539609
## V4A48       -1.475e+01  5.199e+02  -0.028 0.977368
## V4A49       -8.913e-01  4.336e-01  -2.055 0.039838 *
## V5           9.189e-05  5.343e-05   1.720 0.085452 .
## V6A62       -4.411e-01  3.616e-01  -1.220 0.222523
## V6A63       -6.655e-01  5.573e-01  -1.194 0.232443
## V6A64       -2.053e+00  7.501e-01  -2.737 0.006191 **
## V6A65       -1.046e+00  3.363e-01  -3.110 0.001874 **
## V7A72        1.345e-01  5.498e-01   0.245 0.806785
## V7A73       -7.085e-02  5.304e-01  -0.134 0.893732
## V7A74       -7.997e-01  5.731e-01  -1.395 0.162876
## V7A75       -3.136e-01  5.248e-01  -0.597 0.550182
## V8           3.207e-01  1.076e-01   2.980 0.002886 **
## V9A92       -1.672e-01  4.735e-01  -0.353 0.723938
## V9A93       -6.027e-01  4.658e-01  -1.294 0.195685
## V9A94        1.628e-01  5.537e-01   0.294 0.768767
## V10A102      2.320e-01  4.790e-01   0.484 0.628179
## V10A103     -6.746e-01  5.182e-01  -1.302 0.192944
## V11         -5.896e-02  1.085e-01  -0.543 0.586956
## V12A122      1.673e-01  3.197e-01   0.524 0.600622
## V12A123      4.178e-01  2.898e-01   1.441 0.149444
## V12A124      7.226e-01  5.483e-01   1.318 0.187529
## V13         -2.583e-03  1.191e-02  -0.217 0.828265
## V14A142      2.878e-01  5.032e-01   0.572 0.567315
## V14A143     -4.091e-01  3.012e-01  -1.358 0.174345
## V15A152     -4.432e-01  2.858e-01  -1.551 0.120894
## V15A153     -9.597e-01  6.099e-01  -1.574 0.115597
## V16          3.717e-01  2.355e-01   1.578 0.114506
## V17A172      8.836e-01  8.888e-01   0.994 0.320133
## V17A173      8.121e-01  8.641e-01   0.940 0.347310
## V17A174      6.554e-01  8.719e-01   0.752 0.452252
## V18          3.294e-02  3.174e-01   0.104 0.917348
## V19A192     -2.985e-01  2.568e-01  -1.162 0.245071
## V20A202     -7.501e-01  6.976e-01  -1.075 0.282283
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 855.21  on 699  degrees of freedom
## Residual deviance: 590.96  on 651  degrees of freedom
## AIC: 688.96
##
## Number of Fisher Scoring iterations: 14
```

```r
anova(glm1, test="Chi")#type I test
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: V21
##
## Terms added sequentially (first to last)
##
##
##      Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                   699     855.21
## V1    3  126.472       696     728.74 < 2.2e-16 ***
## V2    1   30.045       695     698.69 4.221e-08 ***
## V3    4   19.919       691     678.77 0.0005182 ***
## V4    9   24.864       682     653.91 0.0031251 **
## V5    1    0.389       681     653.52 0.5328840
## V6    4   17.960       677     635.56 0.0012567 **
## V7    4   11.815       673     623.75 0.0187828 *
## V8    1    7.234       672     616.51 0.0071527 **
## V9    3    6.004       669     610.51 0.1114327
## V10   2    2.838       667     607.67 0.2419908
## V11   1    0.002       666     607.67 0.9626747
## V12   3    2.567       663     605.10 0.4632270
## V13   1    0.202       662     604.90 0.6527227
## V14   2    3.977       660     600.92 0.1369079
## V15   2    3.465       658     597.46 0.1768303
## V16   1    2.322       657     595.13 0.1275229
## V17   3    1.714       654     593.42 0.6338794
## V18   1    0.007       653     593.41 0.9342698
## V19   1    1.194       652     592.22 0.2745787
## V20   1    1.258       651     590.96 0.2621147
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Based on the Type I ANOVA test, we've identified several insignificant variables with p-values greater than 0.05, and we can drop the following: V20, V19, V18, V17, V15, V13, V12, V11, V10, V7, and V5.

Next, let's check for multicollinearity among the remaining predictors to ensure that no high correlations are distorting the model's estimates. This can be done by calculating the Variance Inflation Factor (VIF) for each variable to detect multicollinearity.

```r
vif(glm1)
```

```
##         GVIF Df GVIF^(1/(2*Df))
## V1  1.537492  3        1.074324
## V2  1.950812  1        1.396715
## V3  2.663751  4        1.130282
## V4  3.778548  9        1.076648
```

```
## V5  2.429089  1        1.558554
## V6  1.653463  4        1.064877
## V7  2.764675  4        1.135548
## V8  1.346400  1        1.160345
## V9  1.759161  3        1.098713
## V10 1.259315  2        1.059336
## V11 1.433695  1        1.197370
## V12 4.675313  3        1.293109
## V13 1.609002  1        1.268464
## V14 1.364883  2        1.080871
## V15 4.485618  2        1.455310
## V16 1.763185  1        1.327850
## V17 2.933300  3        1.196445
## V18 1.242080  1        1.114486
## V19 1.469455  1        1.212211
## V20 1.127144  1        1.061670
```

```r
alias(glm1)
```

```
## Model :
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20
```

The Variance Inflation Factor (VIF) indicates that variables V2 (loan duration) and V5 (credit amount) have slightly elevated levels, suggesting potential multicollinearity. This could be due to a positive relationship between the duration of the loan and the loan amount, as larger loans may tend to have longer durations.

To explore this relationship further, we can create a scatter plot of V2 (loan duration) against V5 (credit amount) to visually assess the extent of their correlation. This will help us determine if there's a strong linear relationship between the two variables.

```r
ggpairs(credit, columns = c('V2', 'V5'), mapping=ggplot2::aes(color=
'#3366FF'))
```

The scatter plot confirms a positive correlation between V2 (loan duration) and V5 (credit amount), and since the ANOVA test indicated that V5 can be dropped, we'll retain V2.

Now, we'll rerun the model, excluding the insignificant predictors: V20, V19, V18, V17, V15, V13, V12, V11, V10, V7, and V5. This will give us a more refined model based on the significant predictors.

```
glm2<-update(glm1,.~.-V20-V19-V18-V17-V15-V13-V12-V11-V10-V7-V5)
summary(glm2)

##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V6 + V8 + V9 + V14 +
##     V16, family = binomial(link = "logit"), data = build)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.321261   0.865875   0.371 0.710619
## V1A12       -0.224350   0.252630  -0.888 0.374509
## V1A13       -0.811500   0.421439  -1.926 0.054161 .
## V1A14       -1.888124   0.284197  -6.644 3.06e-11 ***
## V2           0.053450   0.009659   5.534 3.14e-08 ***
## V3A31       -0.037748   0.646336  -0.058 0.953427
## V3A32       -0.923920   0.521953  -1.770 0.076707 .
## V3A33       -1.157653   0.581363  -1.991 0.046451 *
## V3A34       -1.849084   0.544090  -3.398 0.000678 ***
## V4A41       -1.165924   0.434173  -2.685 0.007245 **
```

```
## V4A410       -1.391403   0.891563  -1.561 0.118610
## V4A42        -0.548035   0.307107  -1.785 0.074341 .
## V4A43        -1.186527   0.296972  -3.995 6.46e-05 ***
## V4A44        -0.782282   0.893089  -0.876 0.381069
## V4A45        -0.229949   0.647040  -0.355 0.722301
## V4A46         0.503749   0.496017   1.016 0.309825
## V4A48       -14.781920 545.259460  -0.027 0.978372
## V4A49        -0.950098   0.408022  -2.329 0.019883 *
## V6A62        -0.276723   0.334854  -0.826 0.408578
## V6A63        -0.592588   0.543414  -1.090 0.275497
## V6A64        -1.852736   0.714181  -2.594 0.009481 **
## V6A65        -1.001513   0.318204  -3.147 0.001647 **
## V8            0.221999   0.094778   2.342 0.019165 *
## V9A92        -0.212221   0.440190  -0.482 0.629726
## V9A93        -0.782688   0.431170  -1.815 0.069483 .
## V9A94        -0.025271   0.520422  -0.049 0.961270
## V14A142       0.418859   0.487310   0.860 0.390047
## V14A143      -0.386378   0.290892  -1.328 0.184095
## V16           0.300935   0.219125   1.373 0.169644
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 855.21  on 699  degrees of freedom
## Residual deviance: 616.50  on 671  degrees of freedom
## AIC: 674.5
##
## Number of Fisher Scoring iterations: 14
```

With the new model based on the selected terms, we've observed an improvement in the AIC, which has decreased to 674.5—indicating a better fit.

However, the 4th variable has an unusual coefficient estimate for level A48, and both levels A44 and A48 have very high p-values. We need to further investigate these levels to determine whether it makes sense to combine them with another level to simplify the model and improve its performance.

Let's explore their distribution and relationship with the response variable to make an informed decision on potential combinations.

```
View(build$V4)
```

After combining factor level A48 (loan purpose: retraining) with the base level A40 (car new), as well as combining A44 with the base due to low observation counts, let's rerun the model and check if the AIC improves. This adjustment should help stabilize the coefficient estimates and reduce any instability in the model.

```r
levels(build$V4)[levels(build$V4)==c("A48","A44")]<-"A40"
glm2.5<-update(glm2,.~.)
summary(glm2.5)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V6 + V8 + V9 + V14 +
##     V16, family = binomial(link = "logit"), data = build)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.321261   0.865875   0.371 0.710619
## V1A12        -0.224350   0.252630  -0.888 0.374509
## V1A13        -0.811500   0.421439  -1.926 0.054161 .
## V1A14        -1.888124   0.284197  -6.644 3.06e-11 ***
## V2            0.053450   0.009659   5.534 3.14e-08 ***
## V3A31        -0.037748   0.646336  -0.058 0.953427
## V3A32        -0.923920   0.521953  -1.770 0.076707 .
## V3A33        -1.157653   0.581363  -1.991 0.046451 *
## V3A34        -1.849084   0.544090  -3.398 0.000678 ***
## V4A41        -1.165924   0.434173  -2.685 0.007245 **
## V4A410       -1.391403   0.891563  -1.561 0.118610
## V4A42        -0.548035   0.307107  -1.785 0.074341 .
## V4A43        -1.186527   0.296972  -3.995 6.46e-05 ***
## V4A44        -0.782282   0.893089  -0.876 0.381069
## V4A45        -0.229949   0.647040  -0.355 0.722301
## V4A46         0.503749   0.496017   1.016 0.309825
## V4A48       -14.781920 545.259460  -0.027 0.978372
## V4A49        -0.950098   0.408022  -2.329 0.019883 *
## V6A62        -0.276723   0.334854  -0.826 0.408578
## V6A63        -0.592588   0.543414  -1.090 0.275497
## V6A64        -1.852736   0.714181  -2.594 0.009481 **
## V6A65        -1.001513   0.318204  -3.147 0.001647 **
## V8            0.221999   0.094778   2.342 0.019165 *
## V9A92        -0.212221   0.440190  -0.482 0.629726
## V9A93        -0.782688   0.431170  -1.815 0.069483 .
## V9A94        -0.025271   0.520422  -0.049 0.961270
## V14A142       0.418859   0.487310   0.860 0.390047
## V14A143      -0.386378   0.290892  -1.328 0.184095
## V16           0.300935   0.219125   1.373 0.169644
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 855.21  on 699  degrees of freedom
## Residual deviance: 616.50  on 671  degrees of freedom
## AIC: 674.5
##
## Number of Fisher Scoring iterations: 14
```

Now, let's proceed with evaluating the cross-validated performance of the model using 7 folds, with each fold containing 100 observations. This will give us a more robust measure of the model's performance beyond the in-sample results.

```
build$risk<-as.factor(ifelse(build$V21==1,'bad','good'))
buildfolds<-createFolds(build$risk,k=7)

set.seed(123)
cvrm<-train(risk ~ V1 + V2 + V3 + V4 + V6 + V8 + V9 + V14 + V16,
            data=build,
            method='glm',
            trControl=trainControl(method = 'cv', number = 7, index =
buildfolds, classProbs = TRUE, summaryFunction = twoClassSummary),
    metric='ROC')

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful
cases
```

```
cvrm
```

```
## Generalized Linear Model
##
## 700 samples
##   9 predictor
##   2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Cross-Validated (7 fold)
## Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.6880943  0.5063492  0.7792517
```

```
cvrm$finalModel
```

```
##
## Call:  NULL
##
## Coefficients:
## (Intercept)        V1A12        V1A13        V1A14           V2
V3A31
##    -0.32126      0.22435      0.81150      1.88812     -0.05345
0.03775
##       V3A32        V3A33        V3A34        V4A41       V4A410
V4A42
##     0.92392      1.15765      1.84908      1.16592      1.39140
0.54804
##       V4A43        V4A44        V4A45        V4A46        V4A48
V4A49
##     1.18653      0.78228      0.22995     -0.50375     14.78192
0.95010
```

```
##       V6A62        V6A63        V6A64        V6A65           V8
V9A92
##     0.27672      0.59259      1.85274      1.00151     -0.22200
0.21222
##       V9A93        V9A94      V14A142      V14A143          V16
##     0.78269      0.02527     -0.41886      0.38638     -0.30094
##
## Degrees of Freedom: 699 Total (i.e. Null);  671 Residual
## Null Deviance:       855.2
## Residual Deviance: 616.5     AIC: 674.5
```

The 7-fold cross-validated results show an area under the curve (AUC) of 0.6880943, sensitivity of 0.5063492, and specificity of 0.7792517 with a 50% threshold, assuming '1' as the positive class. As noted, some predicted probabilities were exactly 0 or 1, which can occur with logistic regression.

Now, let's make predictions on the test set after relabeling V4A48 and V4A44 to V4A40, just as we did in the build set, and evaluate the performance using the 50% threshold. This will help us see how well the model generalizes to unseen data.

```
levels(test$V4)[levels(test$V4)==c("A48","A44")]<-"A40"
glm2test<-predict(glm2.5, newdata=test,type='response')


#set threshold at 50%
glm2testfact<-as.factor(ifelse(glm2test>0.5,1,0))#positive class is bad risks

confusionMatrix(glm2testfact,as.factor(test$V21))#using 50% threshold

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 182  51
##          1  28  39
##
##                Accuracy : 0.7367
##                  95% CI : (0.683, 0.7856)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.09179
##
##                   Kappa : 0.3236
##
##  Mcnemar's Test P-Value : 0.01332
##
##             Sensitivity : 0.8667
##             Specificity : 0.4333
##          Pos Pred Value : 0.7811
##          Neg Pred Value : 0.5821
##              Prevalence : 0.7000
```

```
##          Detection Rate : 0.6067
##    Detection Prevalence : 0.7767
##       Balanced Accuracy : 0.6500
##
##        'Positive' Class : 0
##
```

```
library(pROC)
roc(test$V21,ifelse(glm2test>0.5,1,0))#inputs must be numeric
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
## roc.default(response = test$V21, predictor = ifelse(glm2test >     0.5, 1, 0))
##
## Data: ifelse(glm2test > 0.5, 1, 0) in 210 controls (test$V21 0) < 90 cases (test$V21 1).
## Area under the curve: 0.65
```

From the test set results, we can see that the model correctly predicted 58.21% of the bad risks, but it misclassified 51 out of 90 bad risks as good, which could lead to significant costs for the bank. On the other hand, 28 good risks were classified as bad, but this mistake has a lower cost since these risks were still good.

Test set sensitivity for the positive class ('1') is higher than the cross-validated result from the build set, which is beneficial since the cost of misclassifying bad risks as good is higher. However, test set specificity for the negative class ('0') is lower, but this is less concerning because the primary focus is on avoiding false negatives.

The test set area under the curve (AUC) is lower than the cross-validated AUC of 0.69, which suggests the model's performance on the test data isn't as strong as during cross-validation, but the sensitivity improvement offsets some of this concern, given the higher cost of misclassifying bad risks as good ones.

Question 10.3.2 Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

Let's explore other thresholds and apply our cost function to assess the model's performance.

We'll define the costs accordingly.

```
    mycost=function(r,pi,threshold){#bad risks
```

```r
    ifelse(r==1,ifelse(pi<threshold,5,0), #misclassifying bad risk incurs a
cost of 5 per risk

          ifelse(pi>threshold#misclass good risk incurrs a cost of 1 per risk
      ,1,0))
    }


  for(threshold in (1:100)/100){
cost= sum(mycost(test$V21,glm2test,threshold))
print(cbind(threshold,cost))}
```

```
##      threshold cost
## [1,]      0.01  209
##      threshold cost
## [1,]      0.02  202
##      threshold cost
## [1,]      0.03  202
##      threshold cost
## [1,]      0.04  194
##      threshold cost
## [1,]      0.05  193
##      threshold cost
## [1,]      0.06  195
##      threshold cost
## [1,]      0.07  194
##      threshold cost
## [1,]      0.08  202
##      threshold cost
## [1,]      0.09  204
##      threshold cost
## [1,]       0.1  212
##      threshold cost
## [1,]      0.11  212
##      threshold cost
## [1,]      0.12  207
##      threshold cost
## [1,]      0.13  204
##      threshold cost
## [1,]      0.14  202
##      threshold cost
## [1,]      0.15  197
##      threshold cost
## [1,]      0.16  197
##      threshold cost
## [1,]      0.17  198
##      threshold cost
## [1,]      0.18  197
##      threshold cost
## [1,]      0.19  195
```

```
##      threshold cost
## [1,]       0.2  201
##      threshold cost
## [1,]      0.21  205
##      threshold cost
## [1,]      0.22  200
##      threshold cost
## [1,]      0.23  206
##      threshold cost
## [1,]      0.24  211
##      threshold cost
## [1,]      0.25  211
##      threshold cost
## [1,]      0.26  207
##      threshold cost
## [1,]      0.27  205
##      threshold cost
## [1,]      0.28  200
##      threshold cost
## [1,]      0.29  215
##      threshold cost
## [1,]       0.3  214
##      threshold cost
## [1,]      0.31  218
##      threshold cost
## [1,]      0.32  217
##      threshold cost
## [1,]      0.33  216
##      threshold cost
## [1,]      0.34  231
##      threshold cost
## [1,]      0.35  229
##      threshold cost
## [1,]      0.36  232
##      threshold cost
## [1,]      0.37  236
##      threshold cost
## [1,]      0.38  243
##      threshold cost
## [1,]      0.39  250
##      threshold cost
## [1,]       0.4  253
##      threshold cost
## [1,]      0.41  252
##      threshold cost
## [1,]      0.42  253
##      threshold cost
## [1,]      0.43  256
##      threshold cost
## [1,]      0.44  265
```

```
##      threshold cost
## [1,]      0.45  269
##      threshold cost
## [1,]      0.46  277
##      threshold cost
## [1,]      0.47  275
##      threshold cost
## [1,]      0.48  279
##      threshold cost
## [1,]      0.49  278
##      threshold cost
## [1,]       0.5  283
##      threshold cost
## [1,]      0.51  288
##      threshold cost
## [1,]      0.52  297
##      threshold cost
## [1,]      0.53  300
##      threshold cost
## [1,]      0.54  303
##      threshold cost
## [1,]      0.55  308
##      threshold cost
## [1,]      0.56  318
##      threshold cost
## [1,]      0.57  322
##      threshold cost
## [1,]      0.58  321
##      threshold cost
## [1,]      0.59  325
##      threshold cost
## [1,]       0.6  330
##      threshold cost
## [1,]      0.61  338
##      threshold cost
## [1,]      0.62  341
##      threshold cost
## [1,]      0.63  339
##      threshold cost
## [1,]      0.64  339
##      threshold cost
## [1,]      0.65  339
##      threshold cost
## [1,]      0.66  353
##      threshold cost
## [1,]      0.67  365
##      threshold cost
## [1,]      0.68  369
##      threshold cost
## [1,]      0.69  374
```

```
##      threshold cost
## [1,]       0.7  379
##      threshold cost
## [1,]      0.71  378
##      threshold cost
## [1,]      0.72  377
##      threshold cost
## [1,]      0.73  376
##      threshold cost
## [1,]      0.74  381
##      threshold cost
## [1,]      0.75  380
##      threshold cost
## [1,]      0.76  385
##      threshold cost
## [1,]      0.77  385
##      threshold cost
## [1,]      0.78  384
##      threshold cost
## [1,]      0.79  384
##      threshold cost
## [1,]       0.8  384
##      threshold cost
## [1,]      0.81  394
##      threshold cost
## [1,]      0.82  404
##      threshold cost
## [1,]      0.83  414
##      threshold cost
## [1,]      0.84  414
##      threshold cost
## [1,]      0.85  418
##      threshold cost
## [1,]      0.86  423
##      threshold cost
## [1,]      0.87  427
##      threshold cost
## [1,]      0.88  432
##      threshold cost
## [1,]      0.89  436
##      threshold cost
## [1,]       0.9  436
##      threshold cost
## [1,]      0.91  435
##      threshold cost
## [1,]      0.92  445
##      threshold cost
## [1,]      0.93  445
##      threshold cost
## [1,]      0.94  445
```

```
##      threshold cost
## [1,]      0.95  450
##      threshold cost
## [1,]      0.96  450
##      threshold cost
## [1,]      0.97  450
##      threshold cost
## [1,]      0.98  450
##      threshold cost
## [1,]      0.99  450
##      threshold cost
## [1,]         1  450
```

The minimum cost is 193, which occurs at a threshold of 0.05.

Now, let's examine the confusion matrix at this threshold to evaluate the model's classification performance.

```
  #set threshold at 5%
glm2testfact2<-as.factor(ifelse(glm2test>0.05,1,0))#positive class is bad risks

confusionMatrix(glm2testfact2,as.factor(test$V21))#using 5% threshold
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1
##          0  37   4
##          1 173  86
## 
##                Accuracy : 0.41
##                  95% CI : (0.3538, 0.468)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 1
## 
##                   Kappa : 0.0857
## 
##  Mcnemar's Test P-Value : <2e-16
## 
##             Sensitivity : 0.1762
##             Specificity : 0.9556
##          Pos Pred Value : 0.9024
##          Neg Pred Value : 0.3320
##              Prevalence : 0.7000
##          Detection Rate : 0.1233
##    Detection Prevalence : 0.1367
##       Balanced Accuracy : 0.5659
## 
```

```
##        'Positive' Class : 0
##
```

```
roc(test$V21,ifelse(glm2test>0.05,1,0))#inputs must be numeric
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
## roc.default(response = test$V21, predictor = ifelse(glm2test >     0.05,
1, 0))
##
## Data: ifelse(glm2test > 0.05, 1, 0) in 210 controls (test$V21 0) < 90
cases (test$V21 1).
## Area under the curve: 0.5659
```

Lowering the threshold to 0.05 reduced the number of false positives to 4. However, this also increased the number of false negatives to 173.

It's important to note that overall accuracy isn't the key metric here, due to the class imbalance (70% good, 30% bad) and the 5:1 cost ratio for misclassification. In fact, overall accuracy decreased, but this is expected given our focus on minimizing costs, not maximizing accuracy.