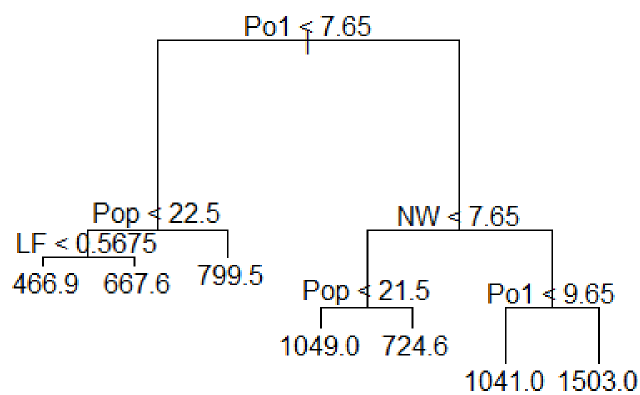
Question 10.1

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).

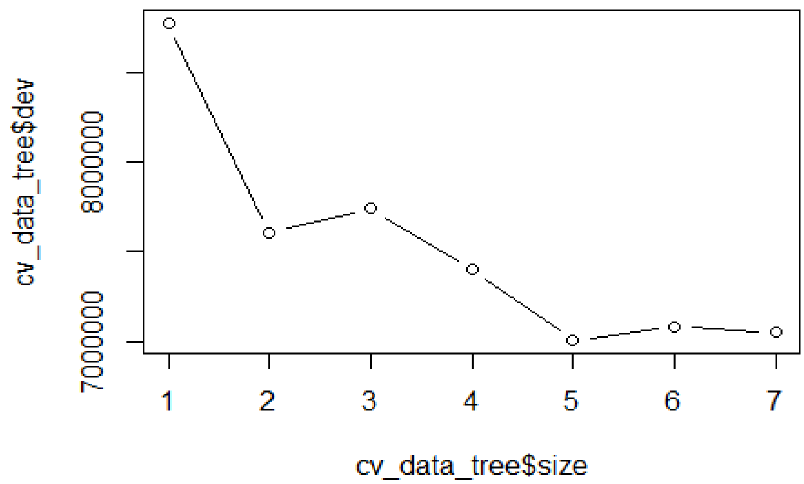
Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using

(a) a regression tree model, and

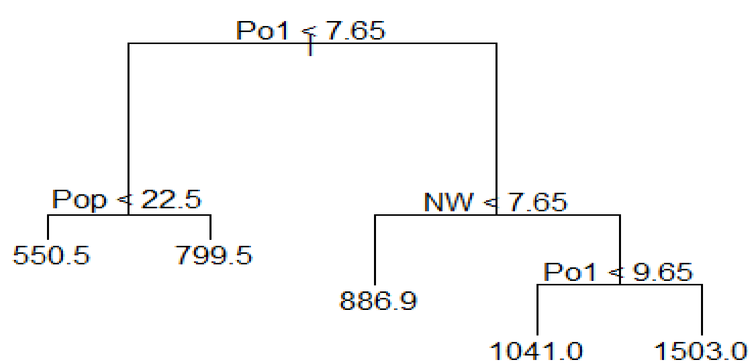
The code for my approach can be found in *appendix 10.1A*. In my approach, I discovered that there were four significant factors used in branching. They were Po1, Pop, LF, and NW. The regression tree generated is as follows:



As can be seen here, the factors Pop and Po1 are used twice in the tree. I also wanted to investigate further and see if pruning the tree could make a better model. For this, I plotted the tree deviance against the tree size (in terms of terminal nodes):



As can be seen here, it seems that having 5 or 7 terminal nodes produces the least deviance. Because I had already generated the original tree with seven terminal nodes, I decided to create another tree pruned to five nodes:



Here, I see that Po1 appears twice like in the model with seven terminal nodes, but LF is no longer present. Pop no longer appears twice either. To compare both models, I decided to look at the residual mean deviances. Surprisingly, I found that pruning the tree increased the residual mean deviance to 54210 as compared to the 47390 for the original tree with seven terminal nodes. Therefore, it seems that pruning the tree is a wrong decision here. I felt that I should investigate this more too though, and I found the R-squared values for the original tree (0.7245) and the pruned tree (0.6691), which indicates that, again, the original tree is likely better. Finally, I wanted to use cross-validation to check model quality. As such, I looked at the SSE for each size of the trees without and with cross-validation:

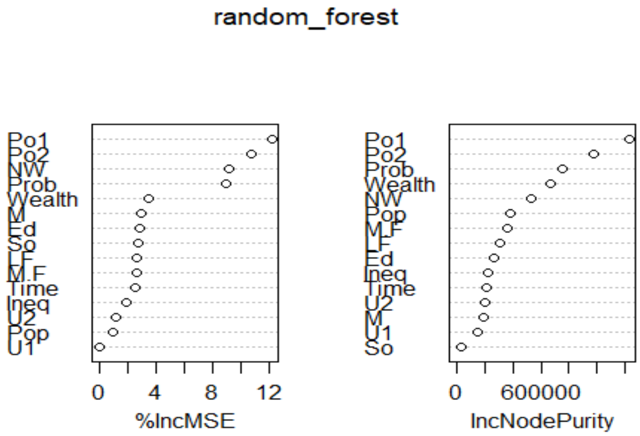
```
#Without CV

prune.tree(data_tree)$size

## [1] 7 6 5 4 3 2 1

prune.tree(data_tree)$dev

## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928
```

```
#With CV

cv_data_tree$size

## [1] 7 6 5 4 3 2 1

cv_data_tree$dev

## [1] 7049681 7083012 7008434 7402184 7739300 7607425 8773298
```

When I look at this information, the errors are much higher in the cross-validation model, which indicates to me that there is some pretty serious overfitting going on in the original model. Additionally, when I consider the factors in the pruned and unpruned trees, it appears to me that Po1 is an extremely important factor since it is present in both trees and is used several times in branching decisions. However, there is a possibility that LF is less important than the others factors (as it was dropped in the pruned tree).

(b) a random forest model.

The code for this approach can be found in *appendix 10.1B*. Because of the way random forest works (i.e., generating many trees and then aggregating them), it is able to reduce overfitting. In the regression tree model, I observed that there was a decent bit of overfitting going on, so I investigated the random forest model to see if it would address that. In this process, I obtained an R-squared value of 0.4108 with the random forest. This would make sense if there was less overfitting in the random forest model as compared to the regression tree model. I also investigated the importance of different predictors in the random forest model:



Here, when I compared it to the tree model, I also saw that the random forest model thought the Po1 factor was very important too. Ultimately, I think the random forest model is a good model that prioritizes similar factors while decreasing the risks of overfitting present in the tree model.

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.

I think a situation where a logistic regression model would be appropriate is if you were investigating how certain variables affect admission into university. For this, the response variable would be a binary one: admittance or rejection. I think some predictors that could be used are: prestige of the student's high school, student's SAT scores, student's ACT scores, length of student's extracurricular involvement, and ethnicity. Using these predictors, a model could be created to predict whether or not a student is likely to be accepted to a university.

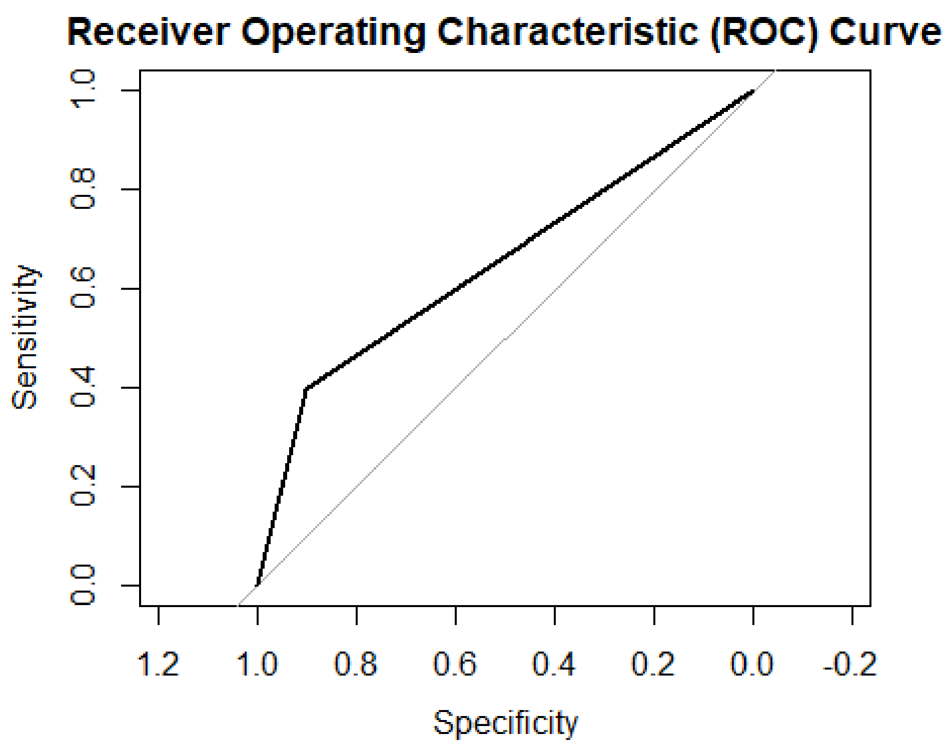
Question 10.3

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-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.

The code for this question can be found in *appendix 10.3.1*. Because there were several variables in the data set and not all of them were significant, I wanted to develop the best model possible. In order to do this, I iterated through several regression models choosing only the significant variables from each one. In the end, I was able to develop the following model:

Variable, Value	Coefficient
Intercept	-4.529e-01
V1, A13	-9.802e-01
V1, A14	-1.637e+00
V2	3.102e-02
V3, A34	-6.940e-01
V4, A41	-1.715e+00
V4, A410	-2.311e+00
V4, A42	-6.579e-01
V4, A43	-9.249e-01
V4, A49	-8.619e-01
V5	1.295e-04
V6, A64	-1.294e+00
V6, A65	-7.435e-01
V8	3.301e-01
V10, A103	-1.325e+00
V14, A143	-8.612e-01
V15, A152	-4.660e-01

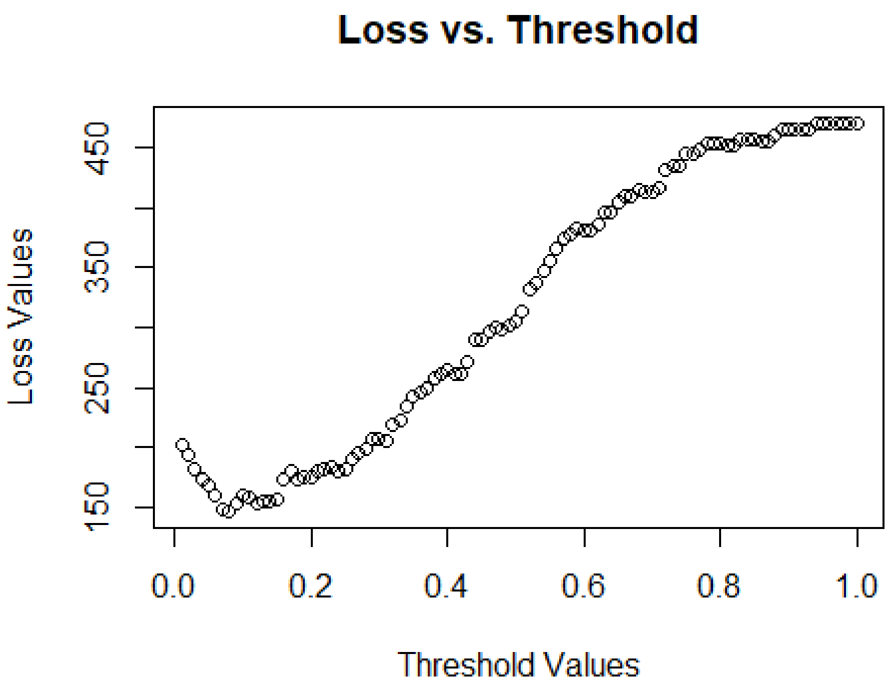
The Akaike Information Criterion value I obtained for this was 683.02. When I validated the model, I obtained an accuracy of 0.7433333 (threshold = 0.5) and a receiver operating characteristic (ROC) curve as follows:



My value for area under the curve (AUC) was 0.6483.

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.

The code for this approach can be found in *appendix 10.3.2*. To determine a good threshold probability, I iterated through different threshold values from (0.01 to 1) and performed cost of loss calculations for each threshold. The plot of the cost of loss against threshold values is as follows:



The specific cost of loss values are as follows (each index is a threshold value; e.g., the first value shows the loss for threshold 0.01; the second shows loss for threshold 0.02, …, all the way to a threshold of 1):

```
##[1] 202 194 182 173 169 160 149 147 154 160 158 154 156 156 157 173 180 174

##[19] 176 176 180 182 184 181 183 190 195 199 207 207 206 219 222 234 243 24
6

##[37] 250 258 262 265 262 262 272 290 289 296 300 298 302 305 314 332 337 34
7

##[55] 356 365 374 377 382 381 381 386 396 395 405 409 409 414 412 412 416 43
1
```

```
##[73] 435 434 444 444 448 453 453 453 452 451 456 456 456 455 455 460 465 46
5

##[91] 465 465 465 470 470 470 470 470 470 470
```

As can be seen, it appears that the lowest loss occurs when threshold is 0.08 (loss = 147). With a threshold of 0.08, the accuracy is 0.5366667 and the AUC of its ROC curve is 0.6568. In general, the losses are relatively low from a threshold of 0.07 (loss = 149) until a threshold of 0.15 (loss = 157) but, after that, they begin to rise relatively steeply. It can also be seen that if we chose a threshold of 0.5 like in question 10.3.1, the loss would be much higher at 305. Therefore, it's important not to just choose the threshold with the highest accuracy but to also consider the cost of loss associated with it.

Appendix

Question 10.1A

```
#This is 10.1A

library(DAAG); library(tree)

set.seed(7)

#Load data
data <- read.table("C:\\Users\\User\\OneDrive\\Desktop\\Data 10.1\\uscrime.tx
t",
stringsAsFactors = F, header = T)

#Create regression tree
data_tree <- tree(Crime~., data = data)

#Only four factors were used to create the tree (Po1, Pop, LF, and NW)
summary(data_tree)

##
## Regression tree:
## tree(formula = Crime ~ ., data = data)
## 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

data_tree$frame

##       var  n        dev      yval splits.cutleft splits.cutright
## 1     Po1 47 6880927.66  905.0851          <7.65           >7.65
## 2     Pop 23  779243.48  669.6087          <22.5           >22.5
## 4      LF 12  243811.00  550.5000        <0.5675         >0.5675
## 8  <leaf>  7   48518.86  466.8571
## 9  <leaf>  5   77757.20  667.6000
## 5  <leaf> 11  179470.73  799.5455
## 3      NW 24 3604162.50 1130.7500          <7.65           >7.65
## 6     Pop 10  557574.90  886.9000          <21.5           >21.5
## 12 <leaf>  5  146390.80 1049.2000
## 13 <leaf>  5  147771.20  724.6000
## 7     Po1 14 2027224.93 1304.9286          <9.65           >9.65
## 14 <leaf>  6  170828.00 1041.0000
## 15 <leaf>  8 1124984.88 1502.8750
```

```
#Plot the tree
plot(data_tree)
text(data_tree)

#We can consider the deviance of trees with different numbers of terminal nod
es
#Based on the values, we can decide how to prune the tree if we want

cv_data_tree <- cv.tree(data_tree)
plot(cv_data_tree$size, cv_data_tree$dev, type= "b")

#Based on the plot, the lowest deviation is with 5 or 7 terminal nodes

#Pruning the tree
terminal_nodes <- 5
prune_data_tree <- prune.tree(data_tree, best = terminal_nodes)
plot(prune_data_tree)
text(prune_data_tree)

summary(prune_data_tree)

##
## Regression tree:
## snip.tree(tree = data_tree, nodes = c(4L, 6L))
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "NW"
## Number of terminal nodes:  5
## Residual mean deviance:  54210 = 2277000 / 42
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -573.9  -107.5    15.5     0.0   122.8   490.1

#If we compare the residual mean deviance, pruning the tree increased it
# from 47390 to 54210
#Therefore, let's stick with the unaltered model

#Let's calculate the quality of fit of the model
data_tree_pred <- predict(data_tree, data=data[,1:15])
RSS <- sum((data_tree_pred - data[,16])^2)
TSS <- sum((data[,16] - mean(data[,16]))^2)
R2 <- 1 - RSS/TSS
R2

## [1] 0.7244962

#The R-squared is therefore 0.7245

#We can also investigate the R-squared value if we used the pruned tree
data_tree_pred2 <- predict(prune_data_tree, data=data[,1:15])
RSS <- sum((data_tree_pred2 - data[,16])^2)
TSS <- sum((data[,16] - mean(data[,16]))^2)
```

```
R2 <- 1 - RSS/TSS
R2

## [1] 0.6691333

#We see that it is lower than the R-squared of the unaltered model

#As we used the training data above, we can also use CV to check model qualit
y
#We can check the SSE for each size of tree without cross-validation

prune.tree(data_tree)$size

## [1] 7 6 5 4 3 2 1

prune.tree(data_tree)$dev

## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928

#Let's check the cross validation results now
cv_data_tree$size

## [1] 7 6 5 4 3 2 1

cv_data_tree$dev

## [1] 7049681 7083012 7008434 7402184 7739300 7607425 8773298

#These errors are much, much larger, which indicates overfitting in orig. mod
el
```

Question 10.1B

```
#This is 10.1B

library(DAAG); library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

set.seed(8)

#Load data
data <- read.table("C:\\Users\\User\\OneDrive\\Desktop\\Data 10.1\\uscrime.tx
t",
stringsAsFactors = F, header = T)

#Generate the random forest
ntry <- 4
random_forest <- randomForest(Crime~., data = data, mtry = ntry, importance=T
RUE)
summary(random_forest)
```

```
##                 Length Class  Mode
## call              5    -none- call
## type              1    -none- character
## predicted        47    -none- numeric
## mse             500    -none- numeric
## rsq             500    -none- numeric
## oob.times        47    -none- numeric
## importance       30    -none- numeric
## importanceSD     15    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y                47    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

```r
random_forest
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = data, mtry = ntry, importance =
TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 86264.36
##                     % Var explained: 41.08
```

```r
#Check quality of model
pred_data <- predict(random_forest)
RSS <- sum((pred_data - data[,16])^2)
TSS <- sum((data[,16] - mean(data[,16]))^2)
R2 <- 1 - RSS/TSS
R2
```

```
## [1] 0.4107735
```

```r
#Look at importance of the model
importance(random_forest)
```

```
##            %IncMSE IncNodePurity
## M       2.96976642     187734.53
## So      2.74235655      27134.61
## Ed      2.87314506     263528.42
## Po1    12.15681006    1229751.02
## Po2    10.72902168     974488.89
```

```
## LF      2.67065010     306597.09
## M.F     2.65719740     358124.59
## Pop     1.00552773     379656.81
## NW      9.17355993     526536.45
## U1      0.04785962     143931.84
## U2      1.15847648     205050.56
## Wealth  3.44995938     672172.18
## Ineq    1.94745173     226982.78
## Prob    8.93316256     758513.30
## Time    2.58330174     213746.67

varImpPlot(random_forest)
```

Question 10.3.1

```
#Load Libraries
library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

#Load Data
data <- read.table("C:\\Users\\User\\OneDrive\\Desktop\\Data 10.3\\germancred
it.txt",
stringsAsFactors = F, header = F)

#Convert the 1s and 2s to 0s and 1s for the logistic regression
data$V21[data$V21==1] <- 0
data$V21[data$V21==2] <- 1

set.seed(10)

#Use a 70-30 split of training and testing data
nrows <- nrow(data)
train_set <- sample(1:nrows, size = round(nrows*0.7))
train <- data[train_set,]
validate <- data[-train_set,]

#Perform iterations to create logistic regression model
#Use all variables first
lreg <- glm(V21~., family=binomial(link = "logit"), data = train)
summary(lreg)

##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2454  -0.6866  -0.3350   0.6271   2.7315
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.709e-01  1.388e+00   0.267 0.789277
## V1A12       -3.518e-02  2.664e-01  -0.132 0.894918
## V1A13       -1.025e+00  4.638e-01  -2.209 0.027150 *
## V1A14       -1.598e+00  2.848e-01  -5.611 2.01e-08 ***
## V2           3.026e-02  1.143e-02   2.648 0.008091 **
## V3A31       -4.026e-01  6.930e-01  -0.581 0.561304
## V3A32       -6.337e-01  5.518e-01  -1.148 0.250800
## V3A33       -1.038e+00  5.980e-01  -1.735 0.082652 .
## V3A34       -1.439e+00  5.609e-01  -2.565 0.010308 *
## V4A41       -1.865e+00  4.660e-01  -4.003 6.26e-05 ***
## V4A410      -2.376e+00  1.067e+00  -2.227 0.025961 *
## V4A42       -1.008e+00  3.243e-01  -3.110 0.001874 **
## V4A43       -1.041e+00  3.035e-01  -3.429 0.000606 ***
## V4A44       -8.000e-01  1.360e+00  -0.588 0.556372
## V4A45       -1.948e-01  6.255e-01  -0.311 0.755457
## V4A46       -9.934e-02  4.589e-01  -0.216 0.828610
## V4A48       -2.125e+00  1.239e+00  -1.714 0.086467 .
## V4A49       -9.695e-01  4.179e-01  -2.320 0.020349 *
## V5           1.541e-04  5.538e-05   2.783 0.005385 **
## V6A62       -7.091e-01  3.689e-01  -1.922 0.054561 .
## V6A63       -9.984e-01  5.695e-01  -1.753 0.079581 .
## V6A64       -1.523e+00  6.378e-01  -2.388 0.016959 *
## V6A65       -8.158e-01  3.109e-01  -2.624 0.008692 **
## V7A72       -2.747e-01  5.797e-01  -0.474 0.635653
## V7A73       -9.896e-02  5.532e-01  -0.179 0.858011
## V7A74       -7.037e-01  5.818e-01  -1.209 0.226505
## V7A75       -1.353e-01  5.505e-01  -0.246 0.805883
## V8           4.166e-01  1.128e-01   3.692 0.000222 ***
## V9A92       -1.219e-01  4.772e-01  -0.255 0.798446
## V9A93       -7.147e-01  4.688e-01  -1.525 0.127355
## V9A94       -1.837e-01  5.538e-01  -0.332 0.740136
## V10A102     -3.486e-03  5.052e-01  -0.007 0.994494
## V10A103     -1.457e+00  5.152e-01  -2.828 0.004686 **
## V11          1.035e-02  1.049e-01   0.099 0.921417
## V12A122      6.786e-01  3.209e-01   2.115 0.034449 *
## V12A123      4.051e-01  3.002e-01   1.350 0.177146
## V12A124      1.025e+00  5.380e-01   1.906 0.056681 .
## V13         -1.514e-02  1.144e-02  -1.323 0.185806
## V14A142     -2.349e-01  5.100e-01  -0.461 0.645129
## V14A143     -9.993e-01  2.856e-01  -3.499 0.000467 ***
## V15A152     -6.193e-01  2.985e-01  -2.075 0.038017 *
## V15A153     -9.986e-01  5.829e-01  -1.713 0.086704 .
## V16          2.001e-01  2.219e-01   0.902 0.367146
```

```
## V17A172      3.168e-01  9.030e-01   0.351 0.725716
## V17A173      3.490e-01  8.764e-01   0.398 0.690480
## V17A174      3.083e-01  8.873e-01   0.347 0.728216
## V18          3.457e-01  3.063e-01   1.129 0.259092
## V19A192     -3.343e-01  2.469e-01  -1.354 0.175839
## V20A202     -1.439e+00  8.745e-01  -1.646 0.099772 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 848.32  on 699  degrees of freedom
## Residual deviance: 607.47  on 651  degrees of freedom
## AIC: 705.47
##
## Number of Fisher Scoring iterations: 5
```

```
#Create second iteration with all variables with p < 0.05
lreg <- glm(V21~ V1+V2+V3+V4+V5+V6+V8+V10+V12+V14+V15, family=binomial(link =
"logit"), data = train)
summary(lreg)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V10 +
##     V12 + V14 + V15, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1960  -0.7111  -0.3650   0.7109   2.6298
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.321e-01  7.313e-01   0.728  0.46686
## V1A12        3.874e-03  2.560e-01   0.015  0.98793
## V1A13       -1.049e+00  4.451e-01  -2.357  0.01844 *
## V1A14       -1.532e+00  2.741e-01  -5.590 2.27e-08 ***
## V2           3.230e-02  1.096e-02   2.947  0.00321 **
## V3A31       -6.505e-01  6.520e-01  -0.998  0.31843
## V3A32       -7.683e-01  5.220e-01  -1.472  0.14105
## V3A33       -1.145e+00  5.819e-01  -1.968  0.04906 *
## V3A34       -1.507e+00  5.471e-01  -2.755  0.00587 **
## V4A41       -1.800e+00  4.467e-01  -4.030 5.59e-05 ***
## V4A410      -2.110e+00  9.760e-01  -2.162  0.03063 *
## V4A42       -8.574e-01  3.089e-01  -2.775  0.00551 **
## V4A43       -9.497e-01  2.890e-01  -3.286  0.00102 **
## V4A44       -4.515e-01  1.256e+00  -0.360  0.71917
## V4A45       -1.150e-01  6.174e-01  -0.186  0.85219
## V4A46       -3.654e-03  4.541e-01  -0.008  0.99358
## V4A48       -1.914e+00  1.189e+00  -1.609  0.10756
```

```
## V4A49       -1.013e+00  4.005e-01  -2.529  0.01145 *
## V5           1.186e-04  5.073e-05   2.339  0.01936 *
## V6A62       -7.112e-01  3.599e-01  -1.976  0.04814 *
## V6A63       -1.069e+00  5.512e-01  -1.939  0.05254 .
## V6A64       -1.446e+00  6.272e-01  -2.306  0.02113 *
## V6A65       -8.754e-01  2.974e-01  -2.943  0.00325 **
## V8           3.194e-01  1.033e-01   3.093  0.00198 **
## V10A102     -1.839e-01  4.901e-01  -0.375  0.70750
## V10A103     -1.454e+00  5.117e-01  -2.841  0.00450 **
## V12A122      5.488e-01  3.054e-01   1.797  0.07233 .
## V12A123      3.835e-01  2.864e-01   1.339  0.18049
## V12A124      8.913e-01  5.049e-01   1.765  0.07754 .
## V14A142     -7.430e-02  4.928e-01  -0.151  0.88017
## V14A143     -8.980e-01  2.756e-01  -3.258  0.00112 **
## V15A152     -7.350e-01  2.712e-01  -2.710  0.00673 **
## V15A153     -1.151e+00  5.427e-01  -2.121  0.03394 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 848.32  on 699  degrees of freedom
## Residual deviance: 627.16  on 667  degrees of freedom
## AIC: 693.16
##
## Number of Fisher Scoring iterations: 5
```

```
#Create third iteration with all variables with p < 0.05
lreg <- glm(V21~ V1+V2+V3+V4+V5+V6+V8+V10+V14+V15, family=binomial(link = "lo
git"), data = train)
summary(lreg)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V10 +
##     V14 + V15, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2644  -0.7100  -0.3733   0.7302   2.7433
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.691e-01  7.171e-01   1.073 0.283478
## V1A12       -3.122e-02  2.529e-01  -0.123 0.901779
## V1A13       -1.052e+00  4.422e-01  -2.380 0.017334 *
## V1A14       -1.562e+00  2.721e-01  -5.739 9.50e-09 ***
## V2           3.319e-02  1.083e-02   3.065 0.002177 **
## V3A31       -6.128e-01  6.453e-01  -0.950 0.342254
## V3A32       -7.830e-01  5.200e-01  -1.506 0.132089
```

```
## V3A33       -1.130e+00  5.793e-01  -1.951 0.051071 .
## V3A34       -1.506e+00  5.456e-01  -2.760 0.005784 **
## V4A41       -1.775e+00  4.451e-01  -3.988 6.68e-05 ***
## V4A410      -2.168e+00  9.852e-01  -2.201 0.027771 *
## V4A42       -7.973e-01  3.045e-01  -2.619 0.008830 **
## V4A43       -9.597e-01  2.876e-01  -3.337 0.000848 ***
## V4A44       -5.494e-01  1.249e+00  -0.440 0.659893
## V4A45       -1.146e-01  6.051e-01  -0.189 0.849831
## V4A46        1.456e-01  4.453e-01   0.327 0.743697
## V4A48       -1.859e+00  1.185e+00  -1.569 0.116733
## V4A49       -1.025e+00  3.988e-01  -2.570 0.010170 *
## V5           1.352e-04  4.947e-05   2.733 0.006279 **
## V6A62       -6.510e-01  3.559e-01  -1.829 0.067334 .
## V6A63       -1.013e+00  5.524e-01  -1.834 0.066685 .
## V6A64       -1.415e+00  6.152e-01  -2.300 0.021435 *
## V6A65       -8.464e-01  2.946e-01  -2.873 0.004065 **
## V8           3.419e-01  1.022e-01   3.345 0.000822 ***
## V10A102     -9.934e-02  4.787e-01  -0.208 0.835604
## V10A103     -1.531e+00  5.020e-01  -3.051 0.002284 **
## V14A142     -1.973e-01  4.906e-01  -0.402 0.687609
## V14A143     -9.244e-01  2.734e-01  -3.381 0.000723 ***
## V15A152     -7.385e-01  2.675e-01  -2.761 0.005769 **
## V15A153     -6.956e-01  3.802e-01  -1.830 0.067286 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 848.32  on 699  degrees of freedom
## Residual deviance: 631.89  on 670  degrees of freedom
## AIC: 691.89
##
## Number of Fisher Scoring iterations: 5
```

```r
#Address the categorical variables
train$V1A13[train$V1 == "A13"] <- 1
train$V1A13[train$V1 != "A13"] <- 0

train$V1A14[train$V1 == "A14"] <- 1
train$V1A14[train$V1 != "A14"] <- 0

train$V3A34[train$V3 == "A34"] <- 1
train$V3A34[train$V3 != "A34"] <- 0

train$V4A41[train$V4 == "A41"] <- 1
train$V4A41[train$V4 != "A41"] <- 0

train$V4A410[train$V4 == "A410"] <- 1
train$V4A410[train$V4 != "A410"] <- 0
```

```r
train$V4A42[train$V4 == "A42"] <- 1
train$V4A42[train$V4 != "A42"] <- 0

train$V4A43[train$V4 == "A43"] <- 1
train$V4A43[train$V4 != "A43"] <- 0

train$V4A49[train$V4 == "A49"] <- 1
train$V4A49[train$V4 != "A49"] <- 0

train$V6A64[train$V6 == "A64"] <- 1
train$V6A64[train$V6 != "A64"] <- 0

train$V6A65[train$V6 == "A65"] <- 1
train$V6A65[train$V6 != "A65"] <- 0

train$V10A103[train$V10 == "A103"] <- 1
train$V10A103[train$V10 != "A103"] <- 0

train$V14A143[train$V14 == "A143"] <- 1
train$V14A143[train$V14 != "A143"] <- 0

train$V15A152[train$V15 == "A152"] <- 1
train$V15A152[train$V15 != "A152"] <- 0

lreg <- glm(V21~ V1A13+V1A14+V2+V3A34+V4A41+V4A410+V4A42+V4A43+V4A49+V5+V6A64
+V6A65+V8+V10A103+V14A143+V15A152, family=binomial(link = "logit"), data = tr
ain)
summary(lreg)

##
## Call:
## glm(formula = V21 ~ V1A13 + V1A14 + V2 + V3A34 + V4A41 + V4A410 +
##     V4A42 + V4A43 + V4A49 + V5 + V6A64 + V6A65 + V8 + V10A103 +
##     V14A143 + V15A152, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2563  -0.7406  -0.3981   0.8149   2.7237
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.529e-01  4.543e-01  -0.997 0.318862
## V1A13       -9.802e-01  4.169e-01  -2.351 0.018702 *
## V1A14       -1.637e+00  2.389e-01  -6.852 7.30e-12 ***
## V2           3.102e-02  1.061e-02   2.925 0.003447 **
## V3A34       -6.940e-01  2.422e-01  -2.865 0.004172 **
## V4A41       -1.715e+00  4.285e-01  -4.002 6.28e-05 ***
## V4A410      -2.311e+00  9.305e-01  -2.484 0.012995 *
## V4A42       -6.579e-01  2.783e-01  -2.364 0.018077 *
## V4A43       -9.249e-01  2.594e-01  -3.566 0.000363 ***
```

```
## V4A49       -8.619e-01  3.516e-01  -2.452 0.014225 *
## V5           1.295e-04  4.787e-05   2.706 0.006813 **
## V6A64       -1.294e+00  6.061e-01  -2.135 0.032766 *
## V6A65       -7.435e-01  2.841e-01  -2.617 0.008870 **
## V8           3.301e-01  9.876e-02   3.343 0.000829 ***
## V10A103     -1.325e+00  4.862e-01  -2.724 0.006445 **
## V14A143     -8.612e-01  2.350e-01  -3.665 0.000247 ***
## V15A152     -4.660e-01  2.161e-01  -2.157 0.031012 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 848.32  on 699  degrees of freedom
## Residual deviance: 649.02  on 683  degrees of freedom
## AIC: 683.02
##
## Number of Fisher Scoring iterations: 5

#Now we can validate
validate$V1A13[validate$V1 == "A13"] <- 1
validate$V1A13[validate$V1 != "A13"] <- 0

validate$V1A14[validate$V1 == "A14"] <- 1
validate$V1A14[validate$V1 != "A14"] <- 0

validate$V3A34[validate$V3 == "A34"] <- 1
validate$V3A34[validate$V3 != "A34"] <- 0

validate$V4A41[validate$V4 == "A41"] <- 1
validate$V4A41[validate$V4 != "A41"] <- 0

validate$V4A410[validate$V4 == "A410"] <- 1
validate$V4A410[validate$V4 != "A410"] <- 0

validate$V4A42[validate$V4 == "A42"] <- 1
validate$V4A42[validate$V4 != "A42"] <- 0

validate$V4A43[validate$V4 == "A43"] <- 1
validate$V4A43[validate$V4 != "A43"] <- 0

validate$V4A49[validate$V4 == "A49"] <- 1
validate$V4A49[validate$V4 != "A49"] <- 0

validate$V6A64[validate$V6 == "A64"] <- 1
validate$V6A64[validate$V6 != "A64"] <- 0

validate$V6A65[validate$V6 == "A65"] <- 1
validate$V6A65[validate$V6 != "A65"] <- 0
```

```r
validate$V10A103[validate$V10 == "A103"] <- 1
validate$V10A103[validate$V10 != "A103"] <- 0

validate$V14A143[validate$V14 == "A143"] <- 1
validate$V14A143[validate$V14 != "A143"] <- 0

validate$V15A152[validate$V15 == "A152"] <- 1
validate$V15A152[validate$V15 != "A152"] <- 0

#Now we can test the model
pred <- predict(lreg, validate, type = "response")
pred
```

```
##           2           3          14          15          16          21
## 0.553568685 0.056969561 0.436923371 0.498319732 0.383700733 0.096699687
##          24          28          29          30          34          37
## 0.069632580 0.043206655 0.075218133 0.749356493 0.032501921 0.456897734
##          38          44          47          49          52          53
## 0.337810868 0.159692958 0.167080308 0.075645738 0.174449540 0.056496259
##          65          66          76          79          82          83
## 0.134181850 0.209188508 0.137978581 0.089705674 0.176915514 0.112988732
##          85          87         104         108         113         124
## 0.248895835 0.252509580 0.306271594 0.509923161 0.739923467 0.221618020
##         130         131         133         135         138         144
## 0.341155087 0.597440225 0.149689577 0.376101109 0.286394923 0.288958462
##         150         153         156         161         162         163
## 0.014122647 0.379795360 0.315748809 0.050758173 0.109582947 0.187351775
##         167         168         170         171         172         174
## 0.398297612 0.134001193 0.264992250 0.644484926 0.092546870 0.092990399
##         180         184         187         190         193         197
## 0.394589322 0.029076080 0.362686698 0.433369239 0.505821621 0.012865722
##         201         203         208         210         211         216
## 0.083968239 0.143350305 0.355082964 0.007541972 0.008396107 0.070399330
##         222         232         236         237         240         243
## 0.657427778 0.156246202 0.612393582 0.469502691 0.080018424 0.511795979
##         244         246         247         248         250         251
## 0.030581897 0.114003063 0.037333114 0.394891351 0.093583991 0.075487171
##         252         253         255         257         258         259
## 0.081342578 0.571073991 0.121412272 0.067685268 0.433448364 0.020998082
##         260         265         272         276         279         281
## 0.038263399 0.065956883 0.059624180 0.032834058 0.049215887 0.008481598
##         282         287         295         297         300         301
## 0.101754145 0.483755995 0.257865621 0.016288288 0.081015384 0.056607705
##         310         313         315         316         318         325
## 0.579679650 0.178312060 0.054368299 0.771390606 0.176052635 0.109242185
##         326         327         329         332         336         340
## 0.189141819 0.036893771 0.338830592 0.120233084 0.152990037 0.352009085
##         343         346         352         354         355         357
## 0.282058375 0.044759577 0.153884966 0.563841898 0.226938856 0.008369745
##         358         360         364         366         374         375
```

```
## 0.209792479 0.643563681 0.053986251 0.045380207 0.684893130 0.881032296
##         378         382         387         388         393         395
## 0.035505682 0.550028403 0.069507961 0.637210156 0.851321767 0.028820779
##         397         402         406         408         409         410
## 0.431587231 0.250701747 0.203228153 0.313733652 0.100867382 0.162493848
##         411         415         420         422         424         427
## 0.404886081 0.430535751 0.375263611 0.092559558 0.130506109 0.076486218
##         428         431         433         434         440         441
## 0.013362698 0.034004847 0.285618627 0.144430355 0.135822283 0.185455928
##         442         443         446         447         452         456
## 0.232253003 0.166746649 0.067407156 0.710091835 0.043193226 0.061847411
##         458         476         484         489         495         500
## 0.226626432 0.336237789 0.022127114 0.187503068 0.408427542 0.080434523
##         502         504         505         512         517         518
## 0.368144390 0.421849485 0.712460058 0.092685110 0.189796051 0.167655688
##         519         520         528         529         534         535
## 0.389905195 0.023761184 0.017015467 0.621643743 0.108448740 0.067998179
##         540         546         548         549         550         553
## 0.218645955 0.715791716 0.175892691 0.482530102 0.056993158 0.213518800
##         557         574         578         579         586         587
## 0.515939685 0.320461536 0.053676564 0.581407083 0.510458820 0.330486378
##         589         590         592         593         595         596
## 0.563774877 0.185674752 0.565129989 0.073387047 0.150248588 0.429954660
##         606         607         611         616         619         624
## 0.701519908 0.065908960 0.502179145 0.769540031 0.516154972 0.490377452
##         628         629         634         635         638         641
## 0.341981125 0.164992996 0.082216384 0.532543633 0.551666700 0.548846145
##         642         646         658         662         664         669
## 0.543693115 0.252545218 0.477427309 0.328666917 0.516462418 0.495490141
##         675         676         681         683         688         690
## 0.247658311 0.150390279 0.062904297 0.181969104 0.816674300 0.130235156
##         696         697         701         704         705         707
## 0.025117060 0.100303830 0.145509326 0.688828692 0.460989530 0.824773754
##         708         709         710         713         720         722
## 0.526326667 0.279079547 0.183743757 0.027063567 0.443640653 0.454085507
##         726         730         732         736         743         747
## 0.024971944 0.017289196 0.359668844 0.723433283 0.062749862 0.451812583
##         748         750         752         755         760         762
## 0.437235997 0.026252798 0.317532252 0.087027601 0.333652796 0.374756398
##         765         768         769         776         781         784
## 0.151034070 0.012216196 0.183167414 0.546136503 0.098461332 0.707278792
##         795         798         799         801         805         806
## 0.241061769 0.042819073 0.118967933 0.202192607 0.404346869 0.768128133
##         811         814         816         821         828         834
## 0.216554766 0.749235287 0.806216968 0.128827433 0.159870746 0.200169351
##         835         838         842         846         856         862
## 0.094645465 0.044088623 0.096164049 0.205083652 0.238547283 0.107300098
##         865         868         870         874         879         881
## 0.183829784 0.079925499 0.451478521 0.094396621 0.534881154 0.084378888
##         883         884         885         894         895         898
```

```
## 0.335397884 0.046781719 0.455717951 0.209115208 0.023006874 0.011055260
##         901         907         909         911         916         919
## 0.273937631 0.326405896 0.020425417 0.209599295 0.726193588 0.257730009
##         924         925         927         935         936         937
## 0.576381911 0.672178603 0.342741377 0.510757152 0.658349188 0.089554972
##         939         941         942         944         947         950
## 0.935969805 0.064717556 0.128333851 0.041575689 0.628754512 0.078148910
##         951         953         956         957         958         961
## 0.210471125 0.318420118 0.118942558 0.177007424 0.090298475 0.029377798
##         963         968         974         975         976         977
## 0.095880666 0.317273837 0.875562769 0.081837806 0.143401920 0.043705507
##         978         989         990         991         995        1000
## 0.232872176 0.472153586 0.248223312 0.033574263 0.103604189 0.229769893

#Threshold is 0.5 here
rounded_pred <- as.integer(pred > 0.5)

#Create confusion matrix
tab <- table(rounded_pred, validate$V21)
tab

##
## rounded_pred   0   1
##            0 186  57
##            1  20  37

#Calculate accuracy
accuracy <- (tab[1,1] + tab[2,2])/sum(tab)
accuracy

## [1] 0.7433333

#Create ROC curve
curve <- roc(validate$V21, rounded_pred)

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

## Setting direction: controls < cases

accuracy

## [1] 0.7433333

curve

##
## Call:
## roc.default(response = validate$V21, predictor = rounded_pred)
##
## Data: rounded_pred in 206 controls (validate$V21 0) < 94 cases (validate$V
21 1).
## Area under the curve: 0.6483
```

```
plot(curve, main="Receiver Operating Characteristic (ROC) Curve")
```

Question 10.3.2

```
#We can iterate through different threshold values to find the best ones
loss <- c()

for(x in 1:100) {
  rounded_pred <- as.integer(pred > (x/100))
  tmatrix <- as.matrix(table(rounded_pred, validate$V21))
  if(nrow(tmatrix)>1){
    c1 <- tmatrix[2,1]
  }else{
    c1 <- 0
  }
  if(ncol(tmatrix) > 1){
    c2 <- tmatrix[1,2]
  }else{
    c2 <- 0
  }
  #Perform cost of loss calculations
  loss <- c(loss, c2*5 + c1)
}

plot(c(1:100)/100,loss,xlab = "Threshold Values",ylab = "Loss Values",main =
"Loss vs. Threshold")

loss

##   [1] 202 194 182 173 169 160 149 147 154 160 158 154 156 156 157 173 180
174
##  [19] 176 176 180 182 184 181 183 190 195 199 207 207 206 219 222 234 243
246
##  [37] 250 258 262 265 262 262 272 290 289 296 300 298 302 305 314 332 337
347
##  [55] 356 365 374 377 382 381 381 386 396 395 405 409 409 414 412 412 416
431
##  [73] 435 434 444 444 448 453 453 453 452 451 456 456 456 455 455 460 465
465
##  [91] 465 465 465 470 470 470 470 470 470 470

which.min(loss)

## [1] 8

rounded_pred <- as.integer(pred > (which.min(loss)/100))
tab <- table(rounded_pred, validate$V21)
accuracy <- (tab[1,1] + tab[2,2])/sum(tab)
curve <- roc(validate$V21, rounded_pred)
```

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

## Setting direction: controls < cases

accuracy

## [1] 0.5366667

curve

##
## Call:
## roc.default(response = validate$V21, predictor = rounded_pred)
##
## Data: rounded_pred in 206 controls (validate$V21 0) < 94 cases (validate$V
21 1).
## Area under the curve: 0.6568
```