Homework-7

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).

Regression Tree

```
rm(list = ls())
library(DAAG)
```

```
## Loading required package: lattice
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.4
```

```
set.seed(42069)

data = read.table("C:/Users/Admin/Desktop/MM/Homework 7/uscrime.txt",
                  stringsAsFactors = FALSE,
                  header = TRUE)
head(data)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

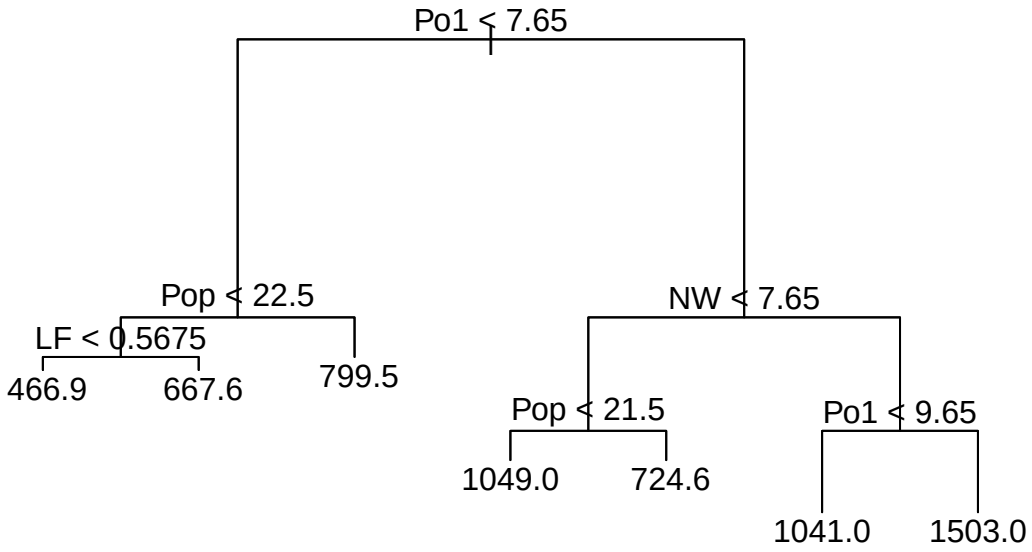
```
model = tree(Crime~., data = data)
summary(model)
```

```
##
## 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
```

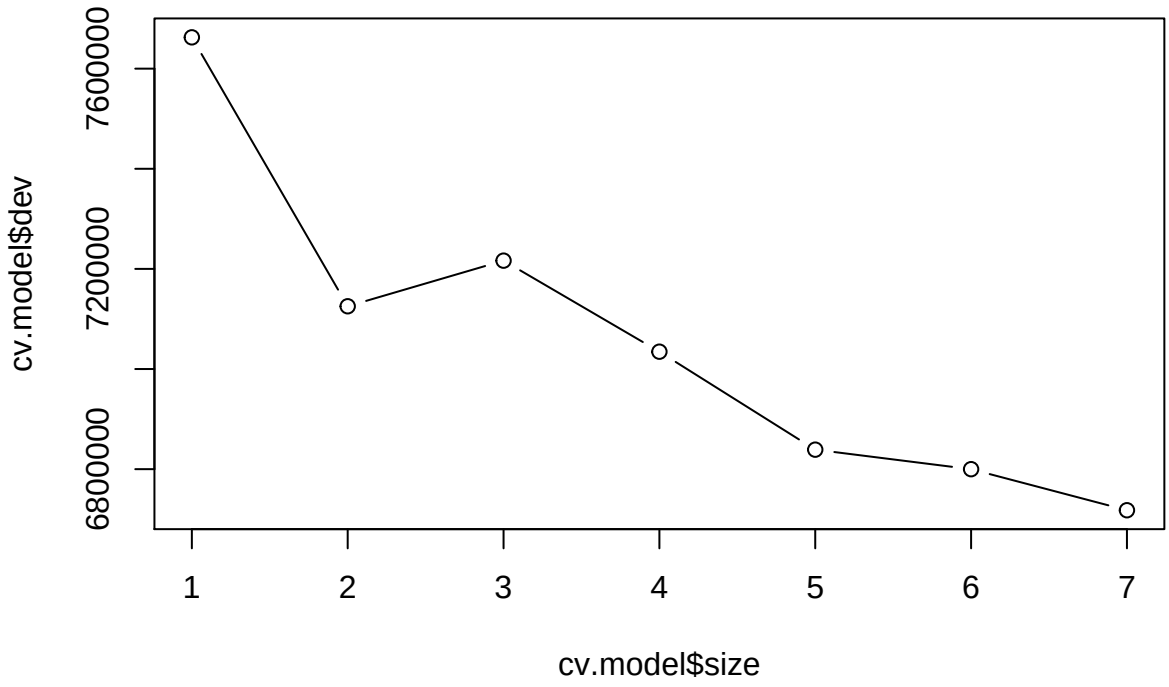
```
# tree splits
model$frame
```

```
##       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
```

```
# Visualizing regression tree
plot(model)
text(model)
```

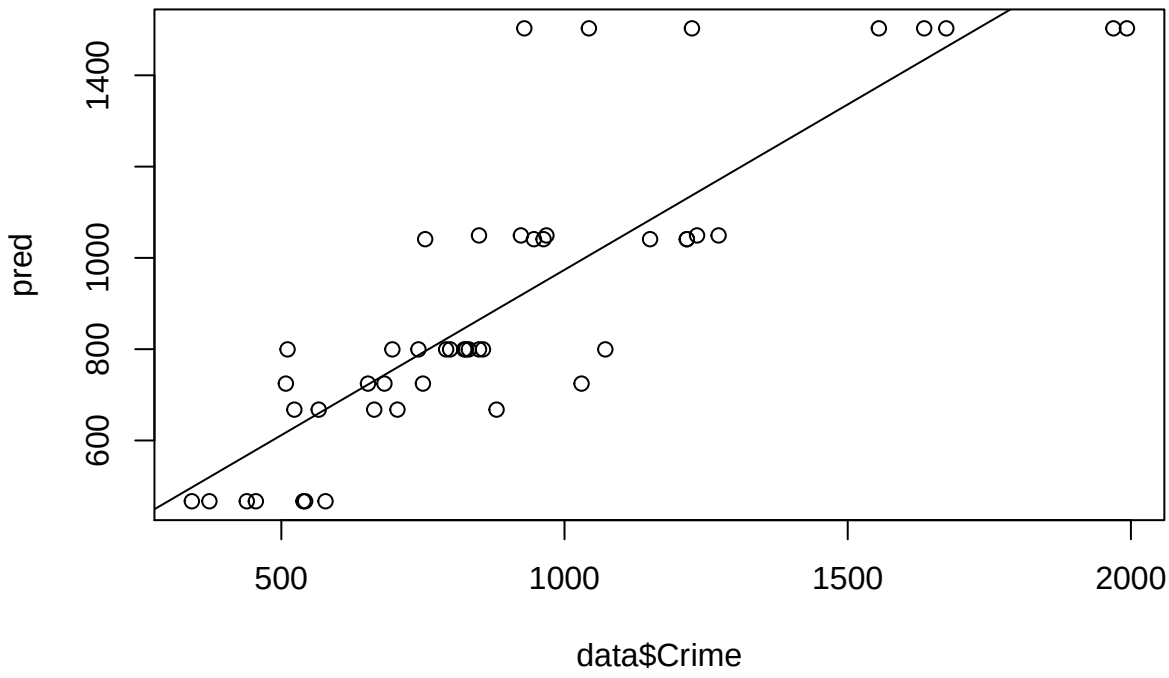



```
# validating 7-leaf node model using cross-validation
cv.model = cv.tree(model)
plot(cv.model$size, cv.model$dev, type = 'b')
```




```
# having 7 nodes yield least deviation in testing sets, no pruning required
pred = predict(model, data)

# visualizing predictions to actual
plot(data$Crime, pred)
abline(lm(pred~data$Crime))
```



Random Forest

```
rm(list = ls())
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.4
```

```
## randomForest 4.6-14
```

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

```
set.seed(42069)

data = read.table("C:/Users/Admin/Desktop/MM/Homework 7/uscrime.txt",
                  stringsAsFactors = FALSE,
                  header = TRUE)
head(data)
```

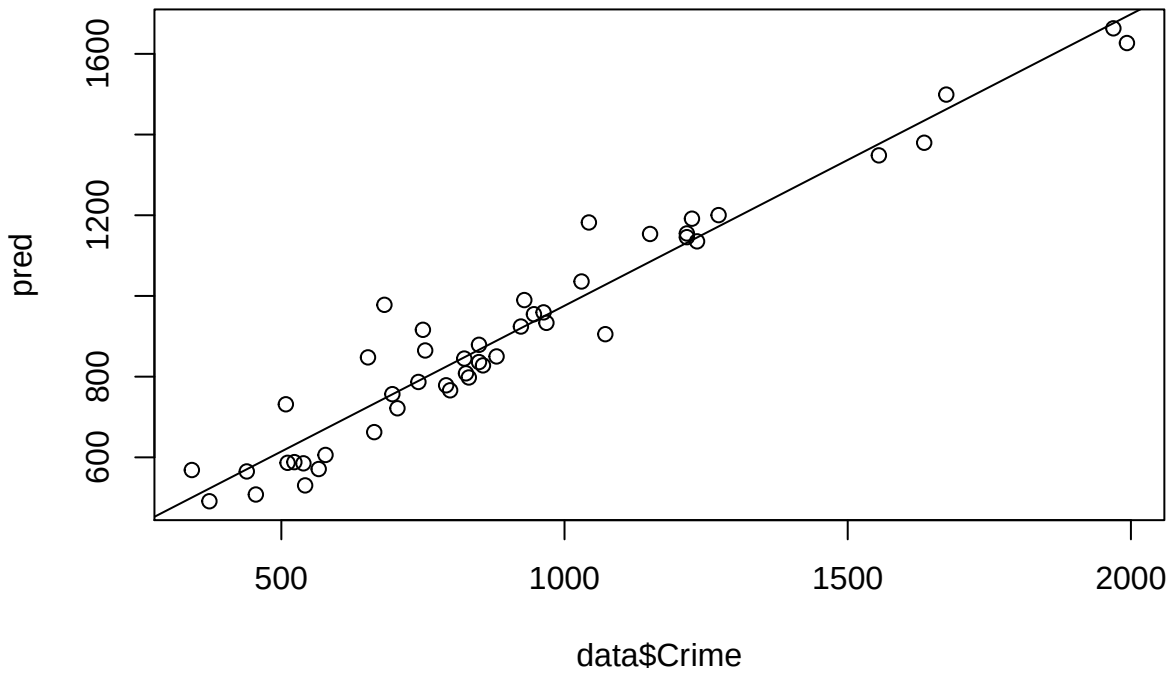
```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
```

```
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

```
model = randomForest(Crime~.,
                     data = data,
                     mtry = floor(ncol(data)/3), # default of p no. of cols/3
                     importance = TRUE)
model
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = data, mtry = floor(ncol(data)/3),      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 86824.09
##                     % Var explained: 40.7
```

```
# visualizing predictions
pred = predict(model, data)
plot(data$Crime, pred)
abline(lm(pred~data$Crime))
```




```
# r-squared
SSR = sum((pred - data$Crime)^2)
SST = sum((data$Crime - mean(data$Crime))^2)
r = 1 - SSR/SST # r=0.88

# variable importance in model
importance(model)
```

```
##            %IncMSE IncNodePurity
## M       1.4902488      212086.7
## So      1.8516926       27802.1
## Ed      4.0600039      204226.5
## Po1    11.7674869     1205283.3
## Po2    10.7247361     1018167.8
## LF      2.5024530      247216.3
## M.F     2.2873609      284966.9
## Pop     0.5198144      333572.2
## NW      9.6441763      514503.8
## U1     -0.7824041      141601.2
## U2      1.1771464      128395.6
## Wealth  4.6392670      725256.1
## Ineq    1.4053776      209179.9
## Prob    7.2634930      840715.3
## Time    0.2853256      194683.3
```

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.

In the credit cards industry, credit loans are one of the most lucrative businesses for the banks. However, loan defaults can turn such opportunities to risks if not dealt with properly. Using a customer's data ranging from:

1. Credit history,
2. Spending patterns,
3. Networth,
4. and even Family size,

the bank is able build an indicative and holistic profile of a customer on whether he/she will have a high chance of defaulting or not. In this case, a logistic regression for binary outcomes will be appropriate be it using a soft or hard classifier, the sensitivity rate to a default could also be adjusted depending on the bank's risk appetite.

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.**
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.**

Part 1

```
rm(list=ls())
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.0.4
```

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

```
##
## Attaching package: 'pROC'
```

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

```
set.seed(42069)
data = read.table("C:/Users/Admin/Desktop/MM/Homework 7/germancredit.txt",
                  sep = " ")
head(data)
```

```
##    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   1
## 2 A191 A201   2
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   2
## 6 A192 A201   1
```

```r
# V21 is the default indicator
data$V21[data$V21==1] = 0
data$V21[data$V21==2] = 1

# train:valid split 70:30
train.idx = sample(nrow(data), size=nrow(data)*0.7)
train = data[train.idx,]
test = data[-train.idx,]

# logistic model
model = glm(V21~.,
          family = binomial(link = "logit"),
          data = train)
summary(model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3888  -0.6690  -0.3438   0.6330   2.9274
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.388e-01  1.371e+00   0.466 0.641331
## V1A12       -4.817e-01  2.709e-01  -1.778 0.075355 .
## V1A13       -1.096e+00  4.885e-01  -2.243 0.024873 *
## V1A14       -1.651e+00  2.839e-01  -5.814 6.09e-09 ***
## V2           3.369e-02  1.132e-02   2.976 0.002920 **
## V3A31       -1.152e-01  7.233e-01  -0.159 0.873446
## V3A32       -9.573e-01  6.127e-01  -1.563 0.118148
## V3A33       -1.162e+00  6.465e-01  -1.798 0.072256 .
## V3A34       -1.658e+00  6.122e-01  -2.709 0.006750 **
## V4A41       -1.585e+00  4.346e-01  -3.646 0.000266 ***
## V4A410      -9.817e-01  9.712e-01  -1.011 0.312061
## V4A42       -8.194e-01  3.193e-01  -2.566 0.010280 *
## V4A43       -9.588e-01  3.086e-01  -3.107 0.001892 **
## V4A44       -6.252e-01  9.130e-01  -0.685 0.493471
## V4A45        1.415e-01  7.391e-01   0.191 0.848181
## V4A46        3.091e-01  5.021e-01   0.616 0.538147
## V4A48       -1.236e+00  1.270e+00  -0.973 0.330378
## V4A49       -7.958e-01  4.027e-01  -1.976 0.048159 *
## V5           1.745e-04  5.549e-05   3.145 0.001660 **
## V6A62       -2.870e-01  3.417e-01  -0.840 0.401075
## V6A63       -5.309e-01  4.753e-01  -1.117 0.263984
## V6A64       -1.201e+00  6.109e-01  -1.965 0.049401 *
## V6A65       -1.225e+00  3.401e-01  -3.602 0.000315 ***
## V7A72        1.409e-01  5.190e-01   0.272 0.785973
## V7A73        1.142e-01  4.983e-01   0.229 0.818750
## V7A74       -6.143e-01  5.400e-01  -1.138 0.255268
## V7A75       -2.297e-01  4.977e-01  -0.462 0.644373
## V8           4.776e-01  1.115e-01   4.282 1.85e-05 ***
```

```
## V9A92       -3.354e-01  4.812e-01  -0.697 0.485782
## V9A93       -7.600e-01  4.690e-01  -1.620 0.105125
## V9A94       -2.806e-01  5.604e-01  -0.501 0.616510
## V10A102      3.807e-01  5.141e-01   0.741 0.458963
## V10A103     -7.513e-01  4.949e-01  -1.518 0.129016
## V11         -5.787e-03  1.089e-01  -0.053 0.957635
## V12A122      3.790e-01  3.078e-01   1.231 0.218188
## V12A123      1.363e-01  2.946e-01   0.463 0.643541
## V12A124      1.684e-01  5.287e-01   0.319 0.750091
## V13         -9.539e-03  1.149e-02  -0.830 0.406443
## V14A142     -4.602e-01  5.260e-01  -0.875 0.381597
## V14A143     -8.039e-01  2.965e-01  -2.711 0.006705 **
## V15A152     -6.492e-01  2.899e-01  -2.240 0.025112 *
## V15A153     -5.962e-01  5.899e-01  -1.011 0.312180
## V16          3.193e-01  2.427e-01   1.316 0.188216
## V17A172     -6.641e-02  7.957e-01  -0.083 0.933485
## V17A173     -2.602e-02  7.599e-01  -0.034 0.972691
## V17A174      2.242e-01  7.692e-01   0.291 0.770712
## V18          2.007e-01  3.010e-01   0.667 0.504898
## V19A192     -4.407e-01  2.554e-01  -1.726 0.084386 .
## V20A202     -2.267e+00  9.946e-01  -2.280 0.022621 *
## ---
## 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: 601.14  on 651  degrees of freedom
## AIC: 699.14
##
## Number of Fisher Scoring iterations: 6
```

```
# model with significant predictors
model = glm(V21~V1+V2+V3+V4+V5+V6+V7+V8+V14+V15+V20,
          family = binomial(link = "logit"),
          data=train)
summary(model)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V14 +
##     V15 + V20, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2710  -0.6844  -0.3713   0.6714   2.9987
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.699e-01  8.411e-01   0.796 0.425774
## V1A12       -4.903e-01  2.627e-01  -1.866 0.061995 .
## V1A13       -1.128e+00  4.716e-01  -2.392 0.016755 *
## V1A14       -1.665e+00  2.784e-01  -5.982 2.20e-09 ***
## V2           3.315e-02  1.089e-02   3.045 0.002324 **
```

```
## V3A31       -3.236e-01  6.826e-01  -0.474 0.635501
## V3A32       -1.099e+00  5.759e-01  -1.909 0.056275 .
## V3A33       -1.155e+00  6.375e-01  -1.812 0.070009 .
## V3A34       -1.587e+00  5.997e-01  -2.646 0.008142 **
## V4A41       -1.605e+00  4.169e-01  -3.849 0.000118 ***
## V4A410      -1.179e+00  8.763e-01  -1.346 0.178335
## V4A42       -7.163e-01  3.058e-01  -2.342 0.019158 *
## V4A43       -9.704e-01  2.969e-01  -3.268 0.001081 **
## V4A44       -6.159e-01  8.943e-01  -0.689 0.491051
## V4A45        2.689e-01  7.305e-01   0.368 0.712804
## V4A46        2.866e-01  4.939e-01   0.580 0.561745
## V4A48       -7.197e-01  1.225e+00  -0.588 0.556805
## V4A49       -7.611e-01  3.923e-01  -1.940 0.052376 .
## V5           1.629e-04  5.145e-05   3.165 0.001549 **
## V6A62       -1.596e-01  3.325e-01  -0.480 0.631160
## V6A63       -5.476e-01  4.724e-01  -1.159 0.246398
## V6A64       -1.161e+00  5.838e-01  -1.989 0.046681 *
## V6A65       -1.175e+00  3.276e-01  -3.587 0.000335 ***
## V7A72        2.577e-01  4.481e-01   0.575 0.565253
## V7A73        8.974e-02  4.185e-01   0.214 0.830189
## V7A74       -6.657e-01  4.669e-01  -1.426 0.153947
## V7A75       -3.284e-01  4.361e-01  -0.753 0.451454
## V8           4.496e-01  1.080e-01   4.162 3.15e-05 ***
## V14A142     -4.947e-01  5.108e-01  -0.969 0.332758
## V14A143     -7.687e-01  2.872e-01  -2.677 0.007430 **
## V15A152     -7.565e-01  2.678e-01  -2.825 0.004729 **
## V15A153     -7.842e-01  3.989e-01  -1.966 0.049330 *
## V20A202     -2.317e+00  9.310e-01  -2.489 0.012800 *
## ---
## 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: 616.33  on 667  degrees of freedom
## AIC: 682.33
##
## Number of Fisher Scoring iterations: 5
```

```r
# validating model on test data
pred = predict(model, test, type = "response")
pred # values between 0-1
```

```
##           1           2           3           4           5           6
## 0.061333608 0.533443799 0.054765983 0.552518737 0.728920172 0.293375842
##           7          10          20          21          22          24
## 0.068742007 0.671032476 0.059333276 0.171710819 0.250261356 0.083285243
##          27          28          34          35          38          43
## 0.147700236 0.144401525 0.038665721 0.416686051 0.278403188 0.590062309
##          46          51          57          58          59          60
## 0.188471533 0.403374825 0.096231303 0.256584525 0.241160852 0.865532847
##          62          64          65          66          69          76
## 0.054160120 0.943858655 0.227816507 0.211534688 0.572134561 0.123674629
##          77          80          82          86          90          93
```

```
## 0.762983602 0.404750177 0.063018326 0.086844537 0.754843212 0.029698133
##          97          98         100         101         105         106
## 0.041579476 0.388472054 0.221071405 0.373255640 0.024361380 0.405393044
##         110         125         134         135         137         138
## 0.050274263 0.351289147 0.312739030 0.294106901 0.040898461 0.174767852
##         143         149         150         155         156         157
## 0.491219857 0.586182187 0.019553458 0.598236420 0.420948109 0.042214594
##         158         159         171         175         179         180
## 0.688535394 0.441584234 0.883217424 0.664902311 0.120923894 0.454376399
##         181         182         184         185         187         188
## 0.273418530 0.480025070 0.063791205 0.511739691 0.304033709 0.207632945
##         193         196         197         201         202         203
## 0.553494520 0.177662597 0.016067854 0.079600352 0.627509370 0.116563906
##         207         208         212         214         217         218
## 0.066553378 0.143580558 0.033036100 0.245194641 0.718635242 0.225543350
##         221         222         223         228         229         230
## 0.477292970 0.762001242 0.194561128 0.610675073 0.051239284 0.807627025
##         236         241         242         243         244         247
## 0.559672625 0.513463607 0.030808770 0.705588535 0.055233091 0.043380625
##         250         261         264         265         267         269
## 0.139680480 0.266136716 0.159242279 0.007594750 0.086791089 0.097559317
##         277         280         281         284         288         296
## 0.043070990 0.268898589 0.017112444 0.022228624 0.464514905 0.681219148
##         298         299         302         304         308         309
## 0.006052264 0.131075994 0.799813102 0.308126138 0.374646176 0.221669124
##         310         315         318         325         327         328
## 0.632522923 0.004102693 0.091742951 0.189182099 0.034594209 0.068843890
##         329         335         337         338         346         347
## 0.340444541 0.919759510 0.184626607 0.402319291 0.079074137 0.218887873
##         350         351         362         372         373         374
## 0.132224980 0.093426438 0.023987895 0.077822193 0.064217869 0.564299004
##         384         386         388         394         400         401
## 0.398360641 0.041970372 0.555101569 0.042652297 0.035277253 0.104577545
##         404         406         408         410         411         414
## 0.232732782 0.170740860 0.074629281 0.080578440 0.303788662 0.024162345
##         416         418         420         421         425         426
## 0.047583381 0.592284097 0.275832892 0.058681937 0.069234490 0.153896370
##         430         434         438         439         446         453
## 0.374875130 0.305804132 0.057671080 0.827802858 0.093774355 0.139554911
##         454         463         465         469         470         472
## 0.074184844 0.522732270 0.113030355 0.129666938 0.045668040 0.659244854
##         476         481         482         483         484         491
## 0.730289065 0.084411458 0.646205230 0.494700333 0.014534479 0.022000852
##         492         496         502         503         511         515
## 0.665118397 0.171743168 0.283309330 0.040687345 0.425440882 0.171368160
##         516         521         522         523         524         527
## 0.034594912 0.239842453 0.305786108 0.930207854 0.034309116 0.206568915
##         531         538         540         545         547         549
## 0.522077476 0.191028942 0.151280778 0.115539427 0.250762065 0.732510416
##         552         554         555         561         564         570
## 0.073182076 0.422321646 0.370060257 0.223619482 0.525698567 0.769466012
##         572         577         584         586         587         590
## 0.080860081 0.090559674 0.901875695 0.579908231 0.514354858 0.201904850
##         591         593         596         606         610         612
```

```
## 0.139511686 0.086950292 0.378257916 0.726236615 0.029902292 0.068146273
##         615         616         619         622         631         633
## 0.085631540 0.886439527 0.494884629 0.258365757 0.367152305 0.141650419
##         635         644         646         651         652         653
## 0.544348517 0.054773363 0.424057781 0.904279450 0.363041321 0.665094465
##         659         661         662         666         668         670
## 0.637427301 0.222845910 0.332483064 0.289889974 0.351347806 0.170103276
##         678         683         684         688         691         694
## 0.670835057 0.222459612 0.133245992 0.608625562 0.180421427 0.125335966
##         703         710         718         720         727         733
## 0.147963350 0.143920757 0.131447741 0.585094286 0.019357565 0.130835582
##         740         743         745         750         759         760
## 0.887948242 0.043751465 0.599043349 0.017976754 0.166772747 0.386558099
##         767         769         772         783         793         802
## 0.496662402 0.143790834 0.856902486 0.389632722 0.006294990 0.385045516
##         804         805         812         816         819         822
## 0.014399091 0.310236477 0.156336464 0.838692225 0.823623649 0.151138370
##         827         832         835         836         840         845
## 0.609885080 0.839258922 0.117106195 0.914949734 0.084158685 0.280187212
##         847         850         851         852         857         859
## 0.205593640 0.339001042 0.420845413 0.008577271 0.019748039 0.651564979
##         866         872         876         879         885         887
## 0.092832844 0.004730209 0.164435659 0.538445364 0.264693117 0.077271209
##         888         889         891         892         894         897
## 0.872858753 0.409265854 0.458466772 0.056634189 0.276780354 0.732657038
##         899         902         904         906         910         911
## 0.033933964 0.052399587 0.083348504 0.353993803 0.094337727 0.301684113
##         919         922         923         926         930         935
## 0.231267093 0.402858888 0.489697578 0.896351518 0.621153318 0.592328832
##         936         937         952         954         958         964
## 0.501604472 0.100199496 0.269873099 0.608911875 0.054900672 0.103128196
##         966         968         970         971         975         978
## 0.193373807 0.373590156 0.289723823 0.573546844 0.150037544 0.121892477
##         986         987         991         992         994         997
## 0.553998360 0.696111584 0.054134984 0.272353570 0.724256540 0.487908227
```

```
# confusion matrix
pred_round = round(pred)
cm = table(pred_round, test$V21)
cm
```

```
##
## pred_round   0   1
##          0 176  48
##          1  34  42
```

```
# Model's accuracy is (183 + 43) / (183 + 43 + 22 + 52) = 75%.
acc = (cm[1,1]+cm[2,2])/sum(cm)
acc
```

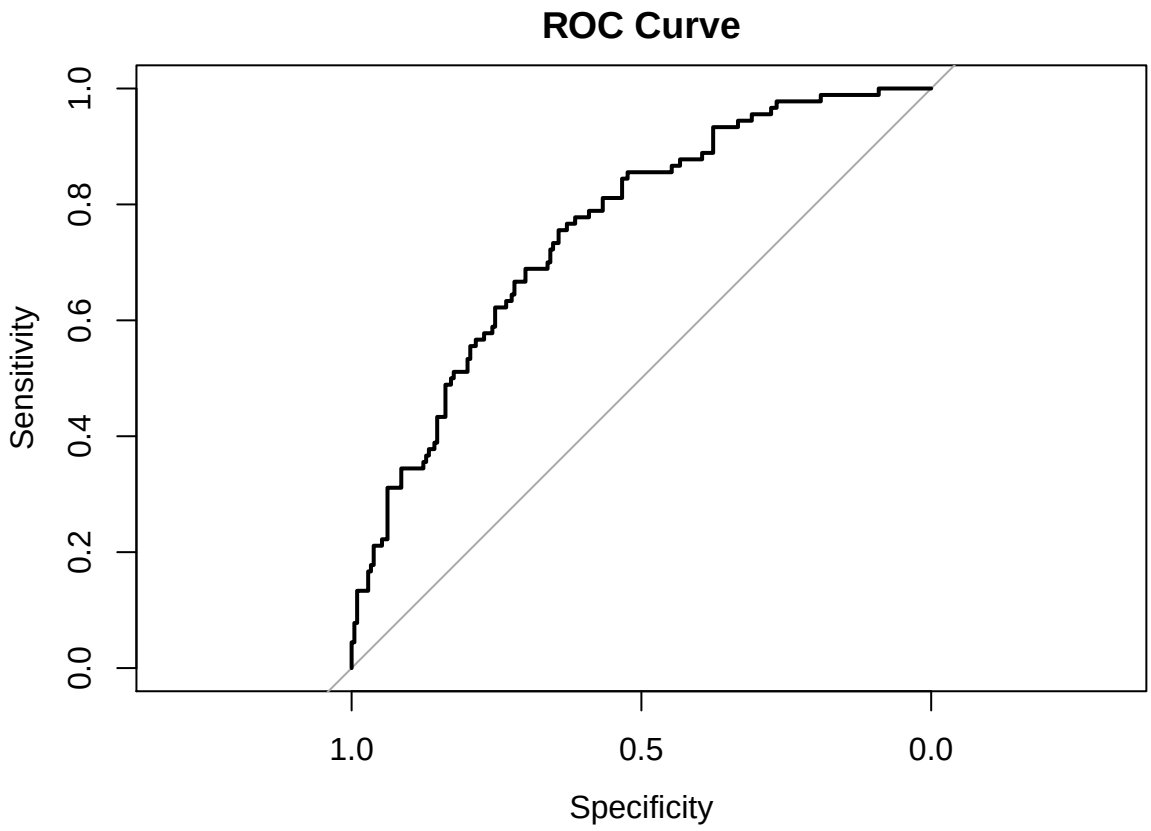
```
## [1] 0.7266667
```

```r
# ROC curve
rc = roc(test$V21, pred)
```

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

```
## Setting direction: controls < cases
```

```r
plot(rc, main = "ROC Curve")
```



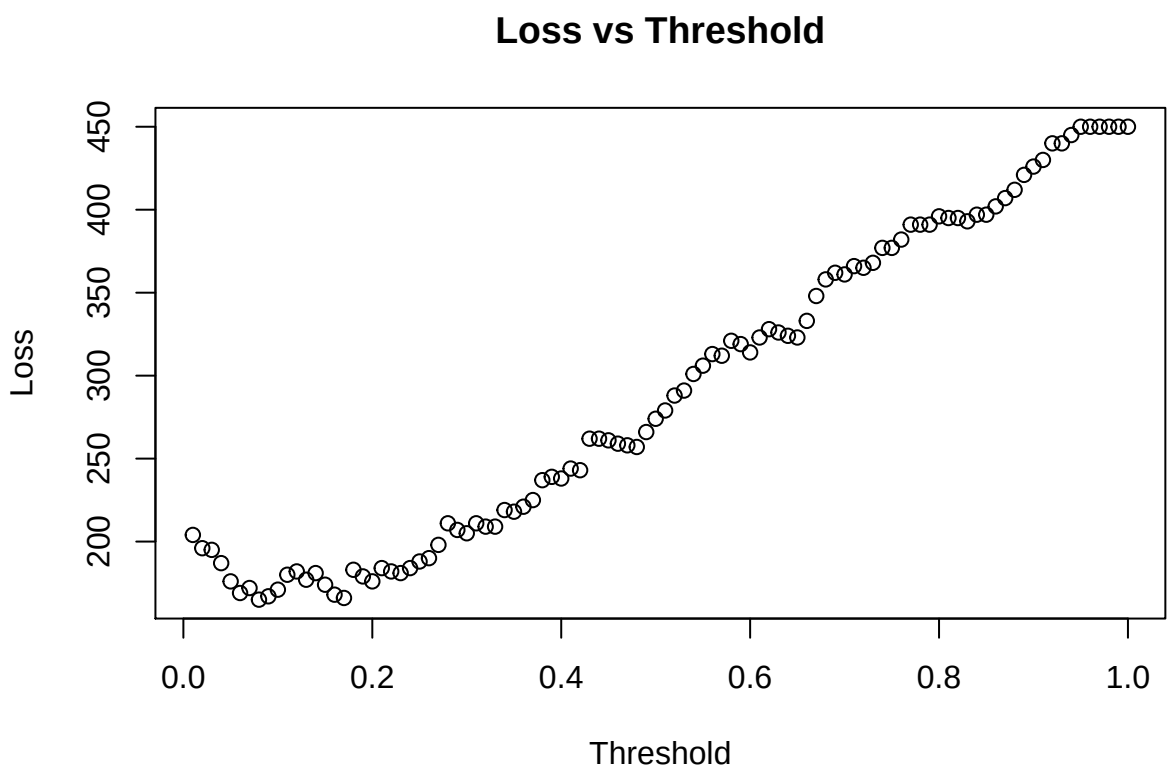

```r
rc # AUC
```

```
##
## Call:
## roc.default(response = test$V21, predictor = pred)
##
## Data: pred in 210 controls (test$V21 0) < 90 cases (test$V21 1).
## Area under the curve: 0.7548
```

Part 2

```r
# The loss of incorrectly classifying a "bad" customer is 5x
loss = c()
for(i in 1:100){
  pred_round = as.integer(pred > (i/100)) # rounding using as.integer
  cm = table(pred_round, test$V21)
  if (nrow(cm) > 1){false_pos = cm[2,1]} else {false_pos = 0}
  if (ncol(cm) > 1){false_neg = cm[1,2]*5} else {false_neg = 0}
  loss[i] = false_neg + false_pos
}

# visualizing threshold to loss
plot(c(1:100)/100,
     loss,
     xlab = "Threshold",
     ylab = "Loss",
     main = "Loss vs Threshold")
```




```r
min(loss) # lowest loss score
```

```
## [1] 165
```

```r
which.min(loss) # threshold with lowest loss
```

```
## [1] 8
```