Question 1

P1 Tree Model -- Step 1: Load Data

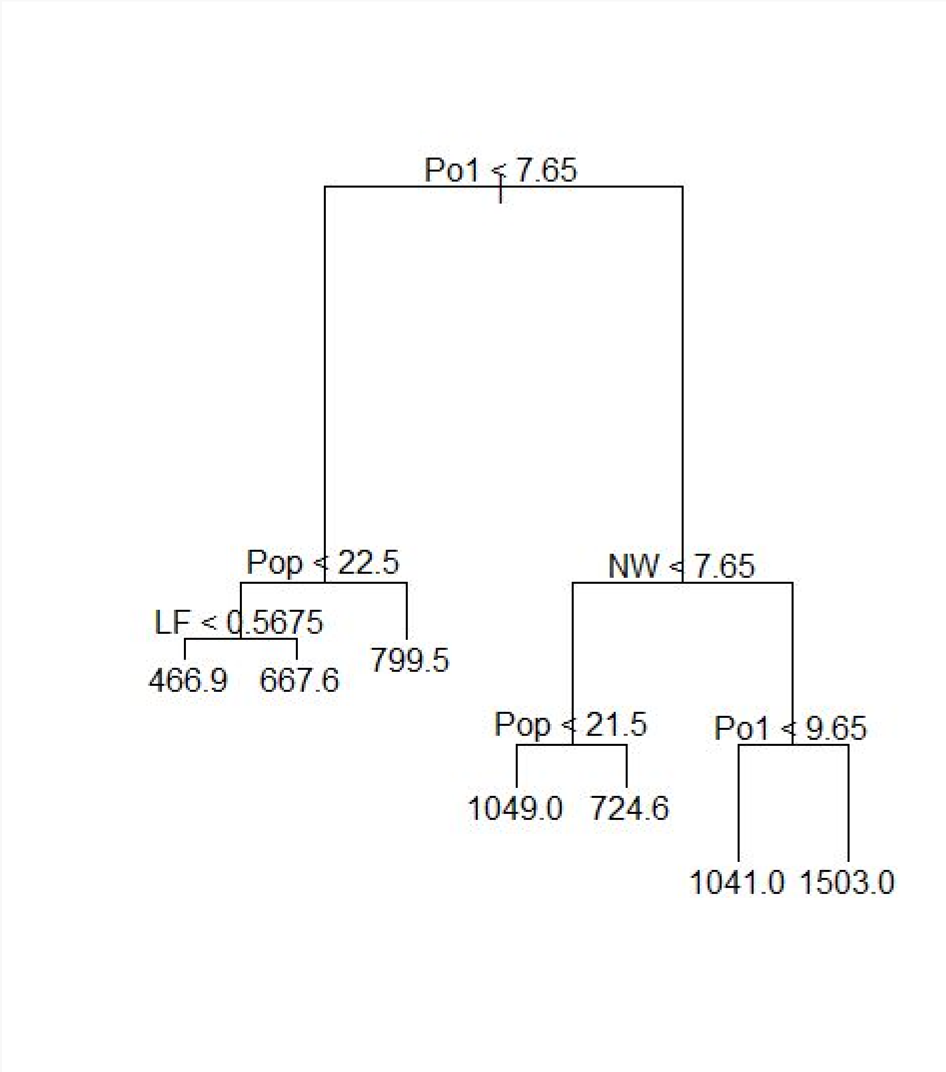
Setup the environment to load the tree and randomForest libs and load uscrime.txt into data_df:

```
# Clear the environment
rm(list = ls())

# Comment in set.seed(33) to repeat results
set.seed(33)

# Load tree lib
require(tree)
require(randomForest)

# Load crime data into a data frame
data_df <- read.table("uscrime.txt", header=TRUE)
```

P1 Tree Model -- Step 2: Train Tree Model

Using the tree function, I trained the tree_model, visualized it, and calculated the R^2:

```
# Train tree model
tree_model <- tree(Crime ~., data_df)
summary(tree_model)

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

# Visualize tree model
plot(tree_model)
text(tree_model)
```

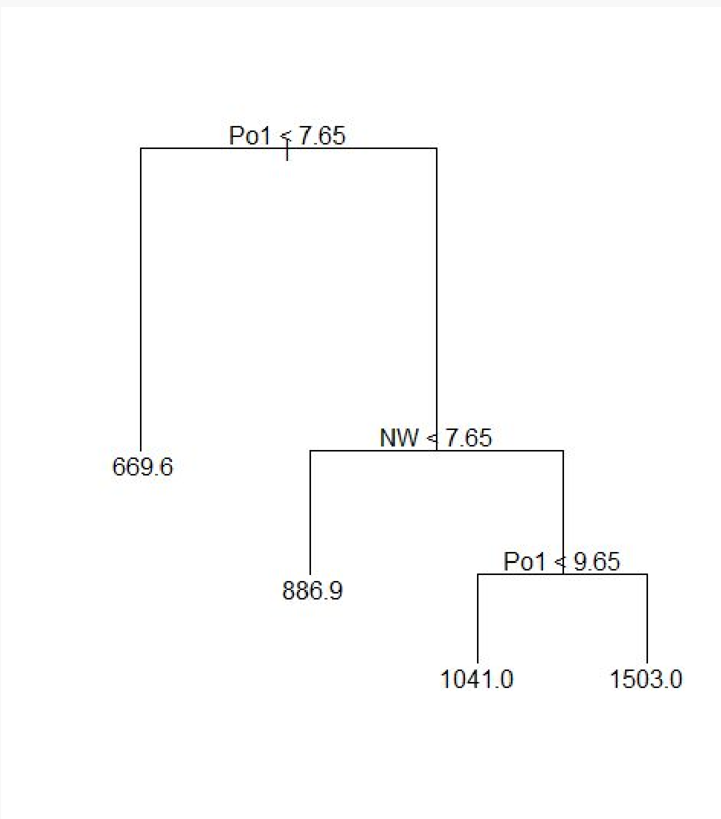



```r
# Function to calculate R^2
ComputeR2 <- function(yhat_df, data_df) {
  SSres <- sum((yhat_df - data_df$Crime)^2)
  SStot <- sum((data_df$Crime - mean(data_df$Crime))^2)
  R2 <- 1 - SSres/SStot
  return(R2)
}

# Calculate R^2
tree_yhat <- predict(tree_model)
tree_r2 <- ComputeR2(tree_yhat, data_df)
Tree_r2  # 0.7244962
```

P1 Tree Model -- Step 4: Prune the Tree

Due to the limited dataset size, the tree should be pruned to a smaller size to allow enough data at each terminal leaf:

```
# Manually prune tree
tree_model_pruned <- prune.tree(tree_model,best = 4)
summary(tree_model_pruned)

Regression tree:
snip.tree(tree = tree_model, nodes = c(6L, 2L))
Variables actually used in tree construction:
[1] "Po1" "NW"
Number of terminal nodes:  4
Residual mean deviance:  61220 = 2633000 / 43
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-573.90 -152.60   35.39    0.00  158.90  490.10

# Visualize pruned tree
plot(tree_model_pruned)
text(tree_model_pruned)
```



```
# Calc R^2
pruned_yhat <- predict(tree_model_pruned)
pruned_r2 <- ComputeR2(pruned_yhat, data_df)
Pruned_r2 # 0.6174017
```

__P2 Forest Model -- Step 1: Train Forest Model__

First I created a variable to set my factor set to 1 + log(n) and then trained a forest model using randomForest():

```
# Use 1+log(n) standard to pick number of factors in each set
factor_set <- round(1 + log(ncol(data_df)))

# Train forest model
forest_model <- randomForest(Crime ~., data_df, mtry = factor_set,
importance = TRUE, ntree = 500)
Forest_model

Call:
 randomForest(formula = Crime ~ ., data = data_df, mtry = factor_set,
importance = TRUE, ntree = 500)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 83777.12
                    % Var explained: 42.78
```

__P2 Forest Model -- Step 2: Assess Forest Model__

I then calculated R^2 and visualized variable importance for the forest model:

```
# Use R2 function to calculate
forest_model_yhat <- predict(forest_model)
ComputeR2(forest_model_yhat, data_df)
# 0.421717

# Visualize variable importance
importance(forest_model)
varImpPlot(forest_model)

          %IncMSE IncNodePurity
M       2.0028307     226264.31
```

```
So       2.5749953      29100.98
Ed       3.0818775     302023.75
Po1     11.5680884    1198828.12
Po2     11.7805096    1015636.06
LF       5.0612871     283691.24
M.F      1.2939713     257229.86
Pop      0.6063912     368227.18
NW       8.3066286     472483.85
U1       0.7838364     131504.61
U2       2.2275446     196529.76
Wealth   4.0709227     611259.13
Ineq     1.2877040     231915.88
Prob     7.7684823     734271.60
Time     1.8736891     195756.99
```

<u>Step 4: Interpret Results</u>

The tree_model_pruned is limited in its predictive ability because it assigns one of four predictions; limiting its variability to new data. Whereas, the forest_model has a lot more variability in its predicted values but is harder to explain. The tree_model_pruned does have a higher R^2 but that's likely due to overfitting on the small uscrime dataset.

```
# Use models to predict
predict(tree_model_pruned, data_df[,1:15])
predict(forest_model, data_df[,1:15])
```

```
> predict(tree_model_pruned, data_df[,1:15])
        1         2         3         4         5         6         7         8         9        10        11        12        13        14        15        16 
 669.6087 1502.8750  669.6087 1502.8750  886.9000  886.9000 1041.0000 1502.8750  669.6087  669.6087 1502.8750  669.6087  669.6087  669.6087  669.6087 1041.0000 
       17        18        19        20        21        22        23        24        25        26        27        28        29        30        31        32 
 669.6087 1502.8750  886.9000 1502.8750  669.6087  669.6087 1041.0000  886.9000  669.6087 1502.8750  669.6087 1041.0000 1502.8750  669.6087  669.6087 1041.0000 
       33        34        35        36        37        38        39        40        41        42        43        44        45        46        47 
 669.6087  886.9000  886.9000  886.9000  669.6087  669.6087  669.6087 1041.0000  669.6087  669.6087  669.6087  886.9000  669.6087  886.9000  886.9000 
> predict(forest_model, data_df[,1:15])
        1         2         3         4         5         6         7         8         9        10        11        12        13        14        15        16 
 783.7498 1384.1008  597.4483 1664.2102 1150.0543  943.2849  948.0244 1329.0192  826.6916  719.7465 1534.9990  841.8490  582.8568  670.5281  770.4288  958.9704 
       17        18        19        20        21        22        23        24        25        26        27        28        29        30        31        32 
 596.1851  986.2564  889.1561 1194.8476  765.3563  559.6587 1149.3142  947.6447  591.4752 1636.3687  580.2940 1162.6364 1183.8141  757.2752  534.6020  889.0966 
       33        34        35        36        37        38        39        40        41        42        43        44        45        46        47 
 929.0435  939.9612  860.4663 1204.9387  805.1989  581.4471  815.8237 1151.5587  850.5561  548.8967  837.1231 1039.4939  526.9124  768.5093  918.2002 
```

Question 2
A situation where a logistic regression model could be useful in my personal life is whether I will enjoy seeing a movie in the theater or not. Potential predictors for this model include: weeks since release, box office receipts, Rotten Tomatoes score, IMDB score, and genre.

Question 3.1
Step 1: Load Data
First, I cleared the environment, set the seed, and loaded germancredit.txt into data_df. Once loaded, the response (V21) needed to be updated to 0 & 1 instead of 1 & 2. I also split the dataset into a training and testing set:

```
# Clear the environment
rm(list = ls())

# Comment in set.seed(33) to repeat results
set.seed(33)

# Load crime data into a data frame
data_df <- read.table("germancredit.txt", header=FALSE)

# Change Response (V21) to 1 (good) or 0 (bad)
data_df$V21[data_df$V21 == 1] <- 1
data_df$V21[data_df$V21 == 2] <- 0

# Create training and test datasets
pct70 <- sample(1:nrow(data_df), size = round(0.7*(nrow(data_df))))
train_df <- data_df[pct70,]
test_df <- data_df[-pct70,]
```

Step 2: Build Model with All Factors
I started off by building the model using all factors. This helped to determine which factors were insignificant and could be removed:

```
# Build model with all factors to determine significant factors
model_all <- glm(V21 ~., family = binomial(link="logit"), train_df)
summary(model_all)
Call:
glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train_df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6666  -0.6807   0.3539   0.6914   2.3340
```

```
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.238e+00  1.322e+00  -0.937 0.348800
V1A12        3.811e-01  2.608e-01   1.461 0.143946
V1A13        9.413e-01  4.369e-01   2.155 0.031189 *
V1A14        1.799e+00  2.880e-01   6.248 4.15e-10 ***
V2          -2.844e-02  1.075e-02  -2.645 0.008178 **
V3A31        6.488e-02  6.742e-01   0.096 0.923331
V3A32        9.223e-01  5.151e-01   1.791 0.073351 .
V3A33        1.409e+00  5.737e-01   2.455 0.014082 *
V3A34        1.672e+00  5.269e-01   3.173 0.001508 **
V4A41        1.729e+00  4.716e-01   3.666 0.000246 ***
V4A410       1.244e+00  8.354e-01   1.489 0.136548
V4A42        7.339e-01  3.141e-01   2.337 0.019451 *
V4A43        1.133e+00  3.031e-01   3.738 0.000185 ***
V4A44        5.839e-01  8.025e-01   0.728 0.466866
V4A45        9.406e-01  7.074e-01   1.330 0.183606
V4A46       -7.489e-02  4.803e-01  -0.156 0.876102
V4A48        2.142e+00  1.298e+00   1.650 0.098878 .
V4A49        9.885e-01  4.028e-01   2.454 0.014128 *
V5          -1.248e-04  5.418e-05  -2.304 0.021221 *
V6A62        5.615e-01  3.540e-01   1.586 0.112743
V6A63        1.135e+00  5.757e-01   1.971 0.048695 *
V6A64        1.122e+00  6.142e-01   1.827 0.067680 .
V6A65        1.053e+00  3.263e-01   3.226 0.001256 **
V7A72        2.824e-01  5.257e-01   0.537 0.591063
V7A73        1.380e-01  5.093e-01   0.271 0.786451
V7A74        9.270e-01  5.561e-01   1.667 0.095528 .
V7A75        2.524e-01  5.091e-01   0.496 0.620012
V8          -4.225e-01  1.089e-01  -3.878 0.000105 ***
V9A92        7.668e-01  4.866e-01   1.576 0.115064
V9A93        1.125e+00  4.761e-01   2.363 0.018109 *
V9A94        7.472e-01  5.660e-01   1.320 0.186832
V10A102     -1.883e-02  4.945e-01  -0.038 0.969619
V10A103      9.826e-01  5.303e-01   1.853 0.063872 .
V11         -8.698e-02  1.060e-01  -0.820 0.412021
V12A122     -4.707e-01  3.081e-01  -1.527 0.126654
V12A123     -3.021e-01  2.860e-01  -1.056 0.290809
V12A124     -8.060e-01  5.169e-01  -1.559 0.118977
V13          1.909e-02  1.114e-02   1.714 0.086466 .
V14A142      2.648e-01  5.122e-01   0.517 0.605189
V14A143      3.556e-01  2.861e-01   1.243 0.213886
```

```
V15A152      6.519e-01  2.876e-01   2.267 0.023406 *
V15A153      1.105e+00  5.904e-01   1.872 0.061223 .
V16         -1.937e-01  2.371e-01  -0.817 0.413772
V17A172     -4.302e-01  8.056e-01  -0.534 0.593374
V17A173     -2.375e-01  7.761e-01  -0.306 0.759625
V17A174     -3.778e-01  7.785e-01  -0.485 0.627526
V18         -2.013e-01  3.085e-01  -0.653 0.514023
V19A192      3.302e-01  2.499e-01   1.321 0.186416
V20A202      1.678e+00  7.646e-01   2.194 0.028215 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 861.88  on 699  degrees of freedom
Residual deviance: 617.04  on 651  degrees of freedom
AIC: 715.04

Number of Fisher Scoring iterations: 5
```

Step 3: Train Refined Model

I used step() to assist with variable selection. After determining the optimal factor set, I retrained the model:

```
# Use step for variable selection
step_output <- step(model_all)
step_output

Call:  glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V15 +
    V20, family = binomial(link = "logit"), data = train_df)

Coefficients:
(Intercept)        V1A12        V1A13        V1A14           V2
V3A31        V3A32        V3A33        V3A34        V4A41       V4A410
V4A42        V4A43
 -1.3460003    0.4718098    1.0697297    1.8212661   -0.0279530
0.2423034    1.1363883    1.4692351    1.7854395    1.6885645    1.1265416
0.6139784    1.1482593
      V4A44        V4A45        V4A46        V4A48        V4A49
V5         V6A62        V6A63        V6A64        V6A65           V8
V9A92        V9A93
  0.5442421    0.8686075   -0.2715785    2.0768567    0.9240859
```

```
-0.0001192    0.3949389    1.0925827    1.1476477    1.0200351   -0.3802927
0.6369140    1.0034172
      V9A94      V15A152      V15A153      V20A202
  0.6844993    0.6639122    0.6466581    1.5887962

Degrees of Freedom: 699 Total (i.e. Null);  670 Residual
Null Deviance:        861.9
Residual Deviance: 638.3      AIC: 698.3

# Rebuild model with optimal step_output variable combination
model_refined <- glm(V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10
+ V14 + V15 + V20, family = binomial(link="logit"), data_df)
summary(model_refined)

Call:
glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 +
    V10 + V14 + V15 + V20, family = binomial(link = "logit"),
    data = data_df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6294  -0.7269   0.3939   0.6910   2.3155

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.145e+00  7.478e-01  -1.531 0.125670
V1A12        4.210e-01  2.137e-01   1.970 0.048833 *
V1A13        1.069e+00  3.621e-01   2.952 0.003156 **
V1A14        1.751e+00  2.294e-01   7.635 2.27e-14 ***
V2          -3.038e-02  9.004e-03  -3.374 0.000740 ***
V3A31        2.944e-02  5.270e-01   0.056 0.955457
V3A32        7.672e-01  4.111e-01   1.866 0.062001 .
V3A33        8.984e-01  4.672e-01   1.923 0.054515 .
V3A34        1.447e+00  4.337e-01   3.337 0.000847 ***
V4A41        1.635e+00  3.650e-01   4.478 7.53e-06 ***
V4A410       1.623e+00  7.572e-01   2.144 0.032052 *
V4A42        7.281e-01  2.530e-01   2.878 0.004000 **
V4A43        8.826e-01  2.433e-01   3.627 0.000286 ***
V4A44        5.943e-01  7.552e-01   0.787 0.431336
V4A45        1.757e-01  5.443e-01   0.323 0.746795
V4A46       -9.063e-02  3.903e-01  -0.232 0.816397
V4A48        1.978e+00  1.219e+00   1.623 0.104535
```

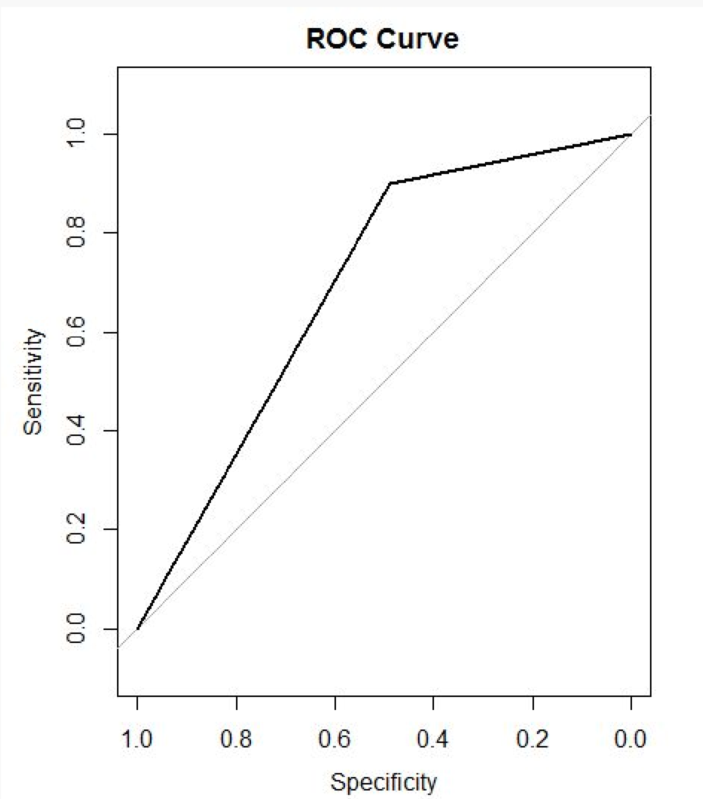
```
V4A49        7.809e-01  3.291e-01   2.373 0.017660 *
V5          -1.156e-04  4.149e-05  -2.786 0.005340 **
V6A62        2.340e-01  2.776e-01   0.843 0.399186
V6A63        4.235e-01  3.966e-01   1.068 0.285639
V6A64        1.318e+00  5.118e-01   2.576 0.010005 *
V6A65        9.527e-01  2.577e-01   3.698 0.000218 ***
V7A72       -2.114e-01  3.729e-01  -0.567 0.570717
V7A73       -4.642e-02  3.480e-01  -0.133 0.893873
V7A74        5.869e-01  3.905e-01   1.503 0.132778
V7A75        1.118e-01  3.625e-01   0.308 0.757737
V8          -3.120e-01  8.553e-02  -3.648 0.000265 ***
V9A92        2.101e-01  3.751e-01   0.560 0.575355
V9A93        6.906e-01  3.668e-01   1.883 0.059763 .
V9A94        3.265e-01  4.417e-01   0.739 0.459797
V10A102     -4.903e-01  4.086e-01  -1.200 0.230242
V10A103      9.846e-01  4.141e-01   2.378 0.017420 *
V14A142      1.597e-01  4.082e-01   0.391 0.695663
V14A143      6.801e-01  2.357e-01   2.886 0.003904 **
V15A152      4.929e-01  2.218e-01   2.222 0.026269 *
V15A153      3.644e-01  3.320e-01   1.098 0.272396
V20A202      1.377e+00  6.221e-01   2.214 0.026851 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1221.73  on 999  degrees of freedom
Residual deviance:  907.68  on 962  degrees of freedom
AIC: 983.68

Number of Fisher Scoring iterations: 5
```

<u>Step 4: Assess the Model</u>

Using a confusion matrix and AUC to assess the model:

```
# Predict using the model
model_refined_yhat <- predict(model_refined, test_df, type = "response")
yhat_pred <- as.integer(model_refined_yhat > 0.5)

# Create confusion matrix
table(yhat_pred, test_df$V21)
```

```
yhat_pred   0   1
        0  42  21
        1  44 193

# Plot the ROC
require(pROC)
AUC <- roc(test_df$V21, yhat_pred)
AUC
Call:
roc.default(response = test_df$V21, predictor = yhat_pred)

Data: yhat_pred in 86 controls (test_df$V21 0) < 214 cases (test_df$V21 1).
Area under the curve: 0.6951

plot(AUC, main = "ROC Curve")
```



Question 3.2

I created a function to calculate predicted cost based on a threshold that I looped over. It was determined that the threshold should be set to 15%:

```
# Function to calculate the predicted cost
predCost <- function(threshold) {
```

```r
  yhat_pred <- as.integer(model_refined_yhat > threshold)
  table <- as.matrix(table(yhat_pred, test_df$V21))
  cost <- table[2,1] + 5*table[1,2]
  return(cost)
}

# Create placeholder for results
results <- as.matrix(vector("list", 100))
i <- seq(8, 99, by=1)

# Loop through all values from 0.0 to 1.0
for (i in 8:99) {
  threshold <- (i/100)
  results[[i]] <- predCost(threshold)
}

# Display results
results
       [,1]
  [1,] NULL
  [2,] NULL
  [3,] NULL
  [4,] NULL
  [5,] NULL
  [6,] NULL
  [7,] NULL
  [8,] 85
  [9,] 84
 [10,] 83
 [11,] 83
 [12,] 83
 [13,] 83
 [14,] 83
 [15,] 81
 [16,] 85
 [17,] 90
 [18,] 90
 [19,] 90
 [20,] 90
 [21,] 92
 [22,] 90
 [23,] 89
```

```
[24,] 89
[25,] 94
[26,] 92
[27,] 91
[28,] 97
[29,] 97
[30,] 96
[31,] 95
[32,] 100
[33,] 103
[34,] 112
[35,] 111
[36,] 113
[37,] 120
[38,] 120
[39,] 120
[40,] 125
[41,] 130
[42,] 135
[43,] 139
[44,] 137
[45,] 141
[46,] 140
[47,] 143
[48,] 141
[49,] 144
[50,] 149
[51,] 153
[52,] 157
[53,] 161
[54,] 166
[55,] 176
[56,] 180
[57,] 198
[58,] 203
[59,] 217
[60,] 220
[61,] 224
[62,] 238
[63,] 242
[64,] 251
[65,] 250
```

```
 [66,] 249
 [67,] 262
 [68,] 270
 [69,] 269
 [70,] 274
 [71,] 284
 [72,] 285
 [73,] 290
 [74,] 301
 [75,] 309
 [76,] 319
 [77,] 333
 [78,] 338
 [79,] 338
 [80,] 358
 [81,] 376
 [82,] 414
 [83,] 428
 [84,] 453
 [85,] 478
 [86,] 508
 [87,] 518
 [88,] 541
 [89,] 575
 [90,] 589
 [91,] 618
 [92,] 673
 [93,] 758
 [94,] 816
 [95,] 856
 [96,] 885
 [97,] 935
 [98,] 990
 [99,] 1045
[100,] NULL
```