HW7

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 1. (a) a regression tree model, and 2. (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).

```
install.packages("rpart")
install.packages("rpart.plot")
install.packages("tree")
library(rpart)
library(rpart.plot)
library(tree)
library(lattice)
library(caret)
```

STEP1: Analyze data with simple tree model

```
## Loading required package: ggplot2
```

```
#loading crime data
crime_data <- read.table("uscrime.txt",header=TRUE)
#building a tree model with our crime data

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

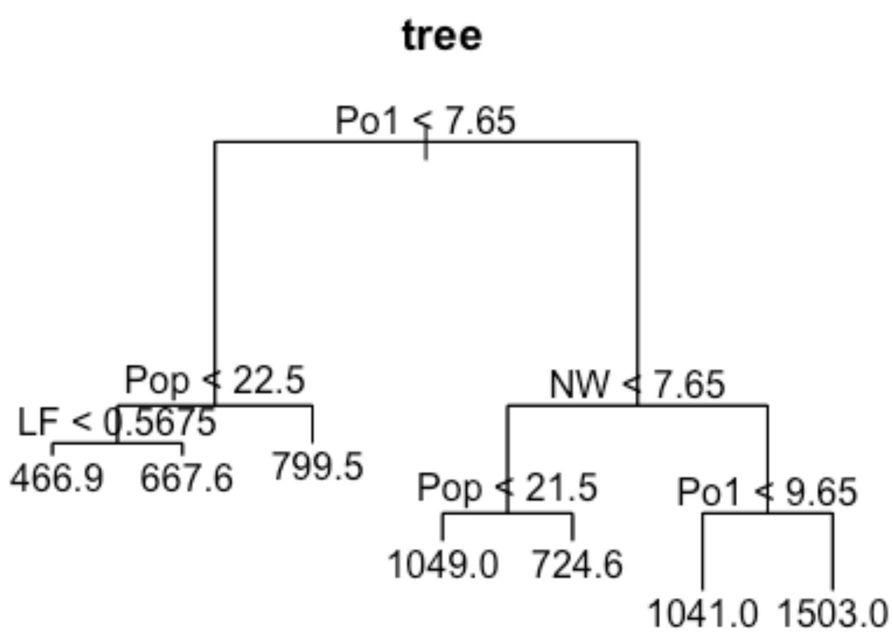
```
##
## Regression tree:
## tree(formula = Crime ~ ., data = crime_data)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean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

```
#this model suggests 4 variables used in tree(Pop 1, Pop, LF, NW) with number of terminal nodes is 7 and mean variance is 47390
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
```

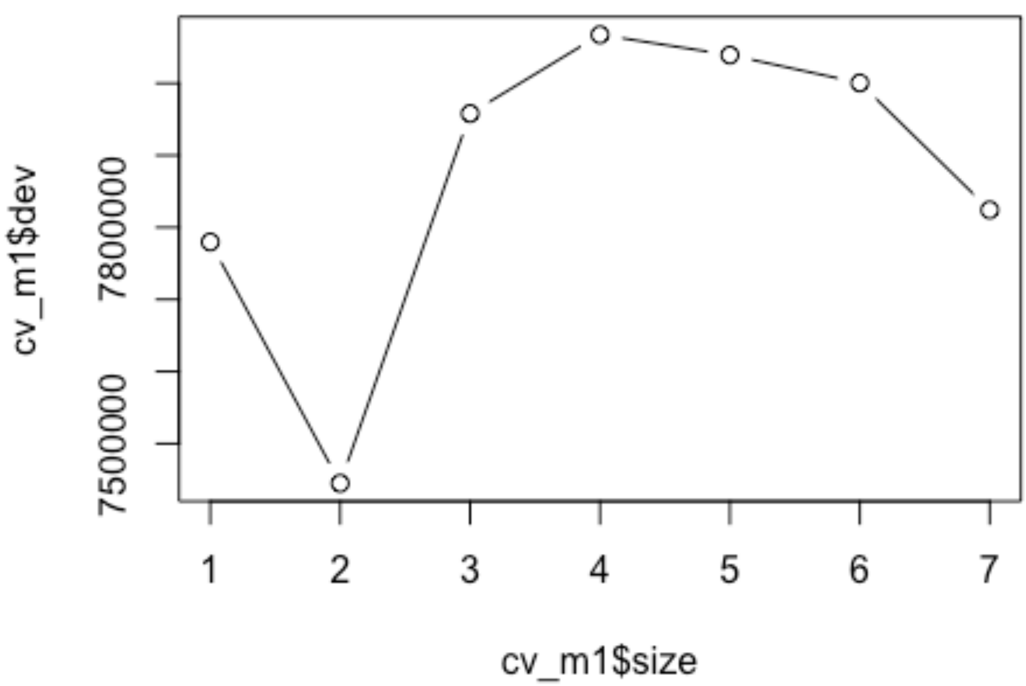
```
plot(model)
text(model)
title("tree")
```




```
#Pruning tree to 4 nodes
model2<-prune.tree(model,best = 4)
summary(model2)
```

```
##
## Regression tree:
## snip.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

#With 4 nodes, residual mean variance has increased,which means model 2 fits
less better than our original model with 7 leaves
#performing cross validation and checking deviation
cv_m1<-cv.tree(model)
prune.tree(model)$size

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

prune.tree(model)$dev

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

#plotting size and dev to visualize which model has the biggest dev
plot(cv_m1$size,cv_m1$dev,type="b")
```

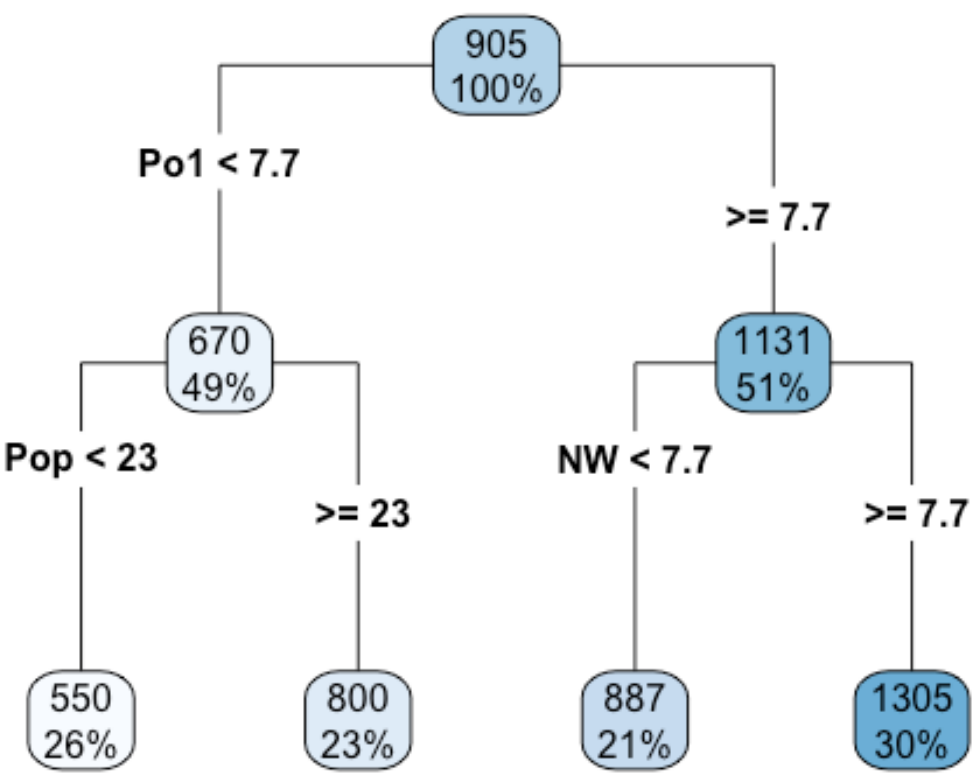



```
#this indicates it is best to use number of terminal nodes =7, since it has t
he smallest deviation
```

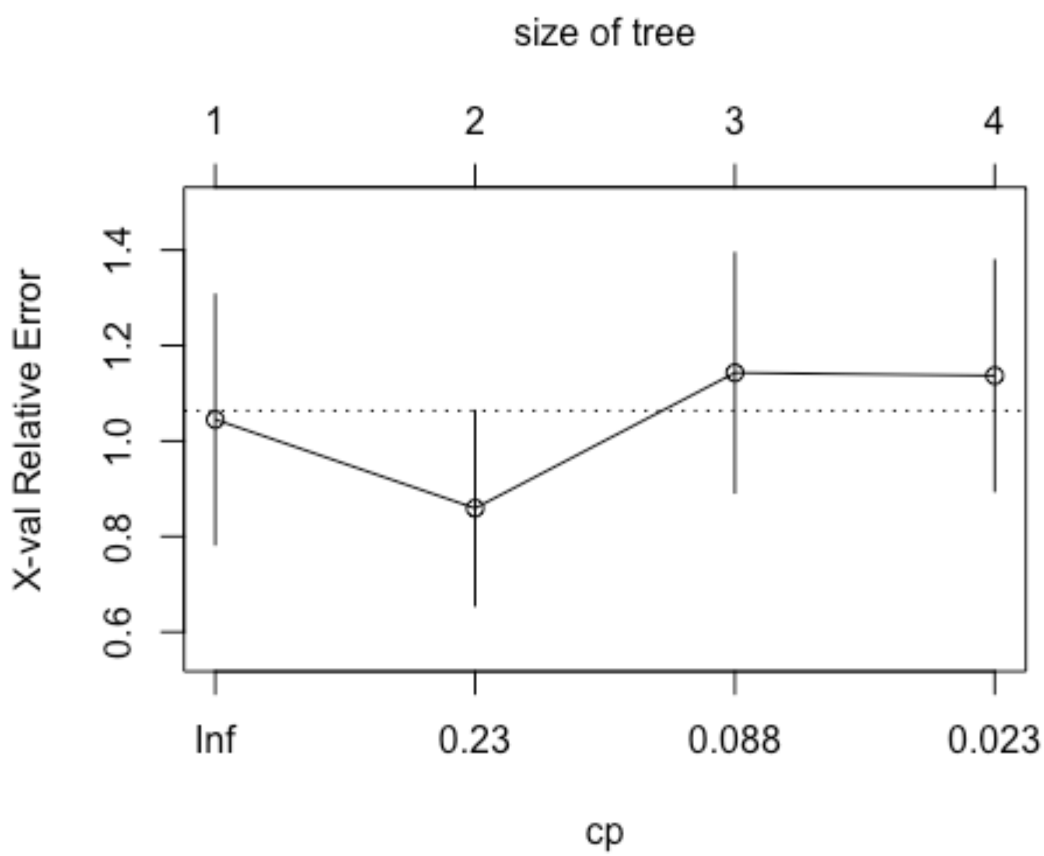
```r
#R^2= 1-SSEresidual/SSEtotal
ytree<-predict(model)
SStot<-sum((crime_data$Crime-mean(crime_data$Crime))^2)
res<-crime_data$Crime-ytree
SSres_model<-sum(res^2)
R_squared<-1-SSres_model/SStot
R_squared

## [1] 0.7244962

#R^2 seems  very high, overfitting may be a problem here.
#Key takeaways based on this regression model :  Po1 is the most important  p
redictor, then followed by NW and PoP.
```

STEP 2: Analyze model with Rpart instead of tree

```r
#Using Rpart instead of tree,
#building a rpart model using ANOVA method since we are doing statistical tes
t with regression
model=rpart(Crime~.,crime_data,method="anova")
#using type 4 in rpart.plot to draw separate split labels for left and right
directions, including nodes
rpart.plot(model,type=4,fallen.leaves=TRUE)
```



```
printcp(model)

## 
## Regression tree:
## rpart(formula = Crime ~ ., data = crime_data, method = "anova")
## 
## Variables actually used in tree construction:
## [1] NW  Po1 Pop
## 
## Root node error: 6880928/47 = 146403
## 
## n= 47 
## 
##         CP nsplit rel error  xerror    xstd
## 1 0.362963      0   1.00000 1.04497 0.26171
## 2 0.148143      1   0.63704 0.85918 0.20414
## 3 0.051732      2   0.48889 1.14289 0.25103
## 4 0.010000      3   0.43716 1.13684 0.24190

plotcp(model)
```




```
summary(model)
```

```
## Call:
## rpart(formula = Crime ~ ., data = crime_data, method = "anova")
##   n= 47
##
##           CP nsplit rel error    xerror      xstd
## 1 0.36296293      0 1.0000000 1.0449720 0.2617139
## 2 0.14814320      1 0.6370371 0.8591828 0.2041409
## 3 0.05173165      2 0.4888939 1.1428916 0.2510256
## 4 0.01000000      3 0.4371622 1.1368382 0.2418953
##
## Variable importance
##    Po1    Po2 Wealth   Ineq   Prob      M     NW    Pop   Time     Ed
LF
##     17     17     11     11     10     10      9      5      4      4
1
##     So
##      1
##
## Node number 1: 47 observations,    complexity param=0.3629629
##   mean=905.0851, MSE=146402.7
##   left son=2 (23 obs) right son=3 (24 obs)
##   Primary splits:
##       Po1    < 7.65      to the left,  improve=0.3629629, (0 missing)
##       Po2    < 7.2       to the left,  improve=0.3629629, (0 missing)
##       Prob   < 0.0418485 to the right, improve=0.3217700, (0 missing)
##       NW     < 7.65      to the left,  improve=0.2356621, (0 missing)
##       Wealth < 6240      to the left,  improve=0.2002403, (0 missing)
##   Surrogate splits:
##       Po2    < 7.2       to the left,  agree=1.000, adj=1.000, (0 split)
##       Wealth < 5330      to the left,  agree=0.830, adj=0.652, (0 split)
##       Prob   < 0.043598  to the right, agree=0.809, adj=0.609, (0 split)
##       M      < 13.25     to the right, agree=0.745, adj=0.478, (0 split)
##       Ineq   < 17.15     to the right, agree=0.745, adj=0.478, (0 split)
##
## Node number 2: 23 observations,    complexity param=0.05173165
##   mean=669.6087, MSE=33880.15
##   left son=4 (12 obs) right son=5 (11 obs)
##   Primary splits:
##       Pop < 22.5      to the left,  improve=0.4568043, (0 missing)
##       M   < 14.5      to the left,  improve=0.3931567, (0 missing)
##       NW  < 5.4       to the left,  improve=0.3184074, (0 missing)
##       Po1 < 5.75      to the left,  improve=0.2310098, (0 missing)
##       U1  < 0.093     to the right, improve=0.2119062, (0 missing)
##   Surrogate splits:
##       NW   < 5.4       to the left,  agree=0.826, adj=0.636, (0 split)
##       M    < 14.5      to the left,  agree=0.783, adj=0.545, (0 split)
##       Time < 22.30055  to the left,  agree=0.783, adj=0.545, (0 split)
##       So   < 0.5       to the left,  agree=0.739, adj=0.455, (0 split)
##       Ed   < 10.85     to the right, agree=0.739, adj=0.455, (0 split)
##
```

```
## Node number 3: 24 observations,    complexity param=0.1481432
##   mean=1130.75, MSE=150173.4
##   left son=6 (10 obs) right son=7 (14 obs)
##   Primary splits:
##       NW   < 7.65      to the left,  improve=0.2828293, (0 missing)
##       M    < 13.05     to the left,  improve=0.2714159, (0 missing)
##       Time < 21.9001   to the left,  improve=0.2060170, (0 missing)
##       M.F  < 99.2      to the left,  improve=0.1703438, (0 missing)
##       Po1  < 10.75     to the left,  improve=0.1659433, (0 missing)
##   Surrogate splits:
##       Ed   < 11.45     to the right, agree=0.750, adj=0.4, (0 split)
##       Ineq < 16.25     to the left,  agree=0.750, adj=0.4, (0 split)
##       Time < 21.9001   to the left,  agree=0.750, adj=0.4, (0 split)
##       Pop  < 30        to the left,  agree=0.708, adj=0.3, (0 split)
##       LF   < 0.5885    to the right, agree=0.667, adj=0.2, (0 split)
##
## Node number 4: 12 observations
##   mean=550.5, MSE=20317.58
##
## Node number 5: 11 observations
##   mean=799.5455, MSE=16315.52
##
## Node number 6: 10 observations
##   mean=886.9, MSE=55757.49
##
## Node number 7: 14 observations
##   mean=1304.929, MSE=144801.8
```

```
#From this model summary, when the size of the tree is 4, the x-val relative
error is the lowest (0.83). It seems like the rpart function finds the optima
l size for the tree so there is no need for pruning.
#Calculating Rsquared
ytree<-predict(model)
SStot<-sum((crime_data$Crime-mean(crime_data$Crime))^2)
res<-crime_data$Crime-ytree
SSres_model<-sum(res^2)
R_squared<-1-SSres_model/SStot
R_squared
```

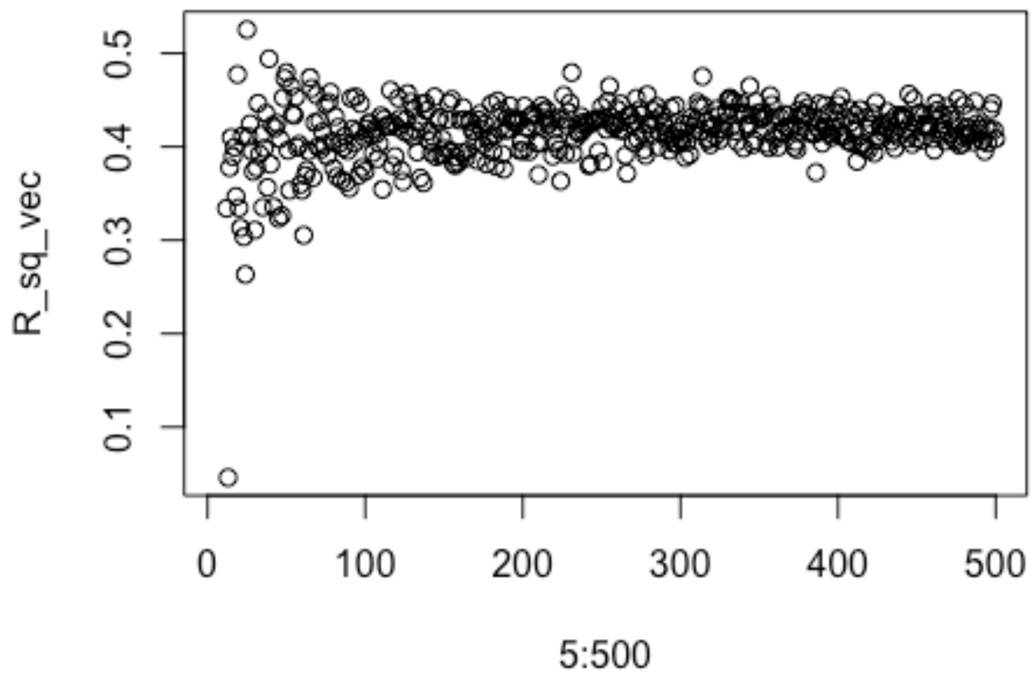
```
## [1] 0.5628378
```

```
#this value is lower than the previous Rsquared value, so this model seems to
be better fitted. Also rpart has cross validation included, so this Rsquared
value is cross-validated with 10 folds as default.

#Using this rpart tree to make a prediction on the crime rate from previous h
omeworks
test_data_frame<-data.frame(M = 14.0,So = 0, Ed = 10.0, Po1 = 12.0, Po2 = 15.
5,LF = 0.640, M.F = 94.0, Pop = 150, NW = 1.1, U1 = 0.120, U2 = 3.6, Wealth =
3200, Ineq = 20.1, Prob = 0.040,Time = 39.0)
```

```
pred<-predict(model,test_data_frame)
pred

##     1
## 886.9

#Our prediction is 887, which corresponds to the 3rd leaf on the right.
```

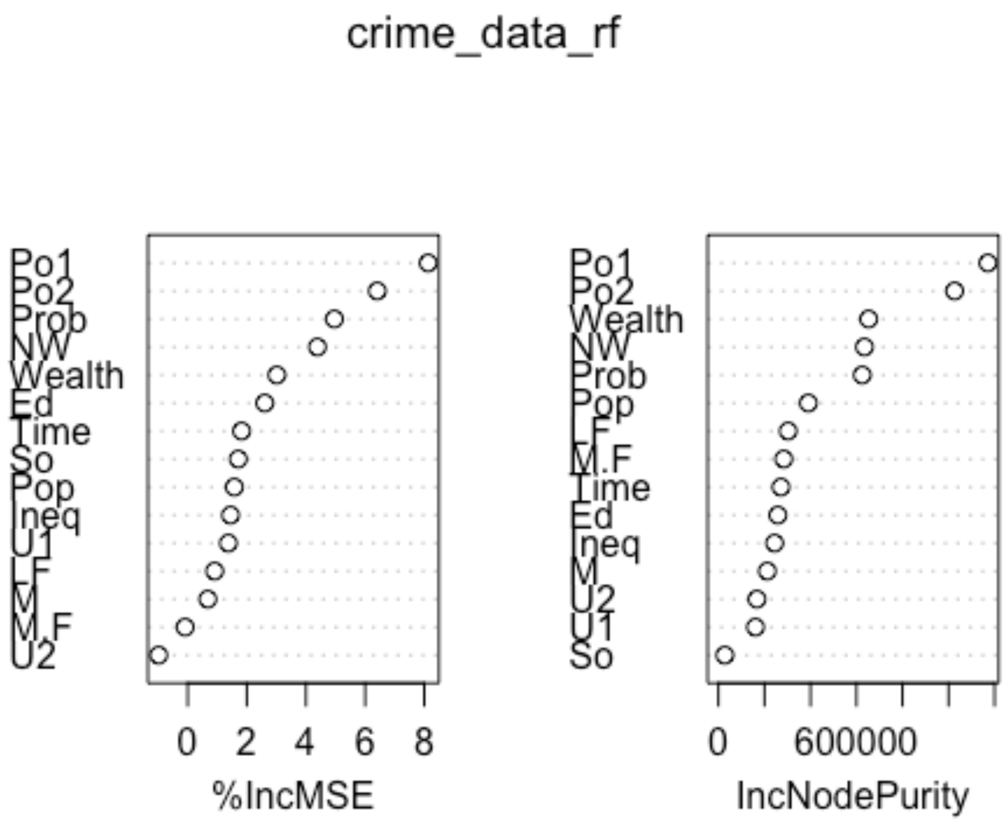
STEP 3: Analyze data using random forests

```
install.packages("randomForest")
library(randomForest)

## randomForest 4.7-1

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

set.seed(4565)
gp=runif(nrow(crime_data))
crime_data=crime_data[order(gp),]

#number of randomly chosen predictors
num_preds<-round(1+log(ncol(crime_data)-1))
R_sq_vec<-c()
#Find optimal ntree by running a for loop from 20 to 500 to find optimal R^2
for(i in 5:500){
  crime_data_rf<-randomForest(Crime~.,crime_data,mtry=num_preds,importance=TR
UE,ntree=i)
  y_rf<-predict(crime_data_rf)
  SStot<-sum((crime_data$Crime-mean(crime_data$Crime))^2)
  res<-crime_data$Crime-y_rf
  SSres_model<-sum(res^2)
  R_sq<-1-SSres_model/SStot
  R_sq_vec=c(R_sq_vec,R_sq)
}
plot(5:500,R_sq_vec)
```



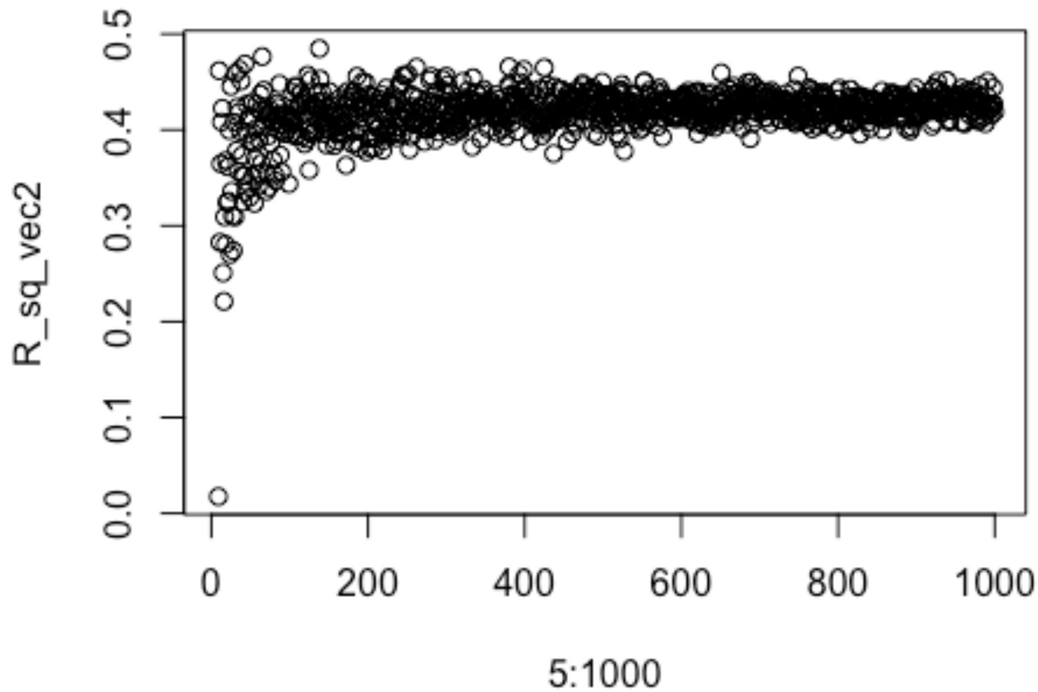

```
#I want to find a right balance for the number of trees (not too large, nor t
oo small), so I will pick the number of trees at 200, since the R^2 start eve
ning out from that number
crime_data_rf<-randomForest(Crime~.,crime_data,mtry=num_preds,importance=TRUE
,ntree=200)
y_rf<-predict(crime_data_rf)
  SStot<-sum((crime_data$Crime-mean(crime_data$Crime))^2)
  res<-crime_data$Crime-y_rf
  SSres_model<-sum(res^2)
  R_sq<-1-SSres_model/SStot
  R_sq

## [1] 0.4283483

  #Cross validation on random forest model
 install.packages("rfUtilities")
  library(rfUtilities)
  #performing a 20-fold data validation with proprotion data withhold = 0.1
  crime_data_rf_cv<-rf.crossValidation(crime_data_rf,crime_data,p=0.1,n=20)

## running: regression cross-validation with 20 iterations
```

```
  SSE_cv<-crime_data_rf_cv$fit.mse*nrow(crime_data)
  R_sq_cv<-1-SSE_cv/SStot
  R_sq_cv

## [1] 0.4253123

  #cross validation R^2 and model R^2 are pretty close, this is a good sign o
ur model is not overfitted.
  varImpPlot(crime_data_rf)
```




```
  #our most important variables are: Po1, Po2, Wealth,Prob, NW, Pop
  opt_crime_data<-crime_data[c(4,5,8,9,12,14,16)]
  rf_optimized<-randomForest(Crime~.,opt_crime_data,mtry=num_preds,importance
=TRUE,ntree=200)
  y_rf_opt<-predict(rf_optimized)
  SStot<-sum((opt_crime_data$Crime-mean(opt_crime_data$Crime))^2)
  res<-opt_crime_data$Crime-y_rf_opt
  SSres_model<-sum(res^2)
  R_sq_opt<-1-SSres_model/SStot
  R_sq_opt

## [1] 0.269389
```

```
  crime_data_rf_cv2<-rf.crossValidation(rf_optimized,opt_crime_data,p=0.1,n=2
0)

## running: regression cross-validation with 20 iterations

  SSE_cv2<-crime_data_rf_cv2$fit.mse*nrow(opt_crime_data)
  R_sq_cv2<-1-SSE_cv2/SStot
  R_sq_cv2

## [1] 0.2581139

  #Optimized Rsquared cross validation and model are similar, which means our
data is definitely not overfitted. This makes sense because one of the advant
age of a random forest is that it avoids overfitting.

#Increasing the  number of trees to 1000.
  R_sq_vec2<-c()
  for(i in 5:1000){
  crime_data_rf<-randomForest(Crime~.,crime_data,mtry=num_preds,importance=TR
UE,ntree=i)
  y_rf2<-predict(crime_data_rf)
  SStot2<-sum((crime_data$Crime-mean(crime_data$Crime))^2)
  res2<-crime_data$Crime-y_rf2
  SSres_model2<-sum(res2^2)
  R_sq2<-1-SSres_model2/SStot2
  R_sq_vec2=c(R_sq_vec2,R_sq2)
}
plot(5:1000,R_sq_vec2)
```




```
#With 1000 trees, our R^2 value is constant no matter the number of trees, th
is shows that having a large number of trees mean more computational cost(thi
s took more time to loop) and after a certain number of trees, the improvemen
t is negligible.
```

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 could use a logistic model at my job to determine whether a truck needs service or not. The following factors could be used to determine the prediction:

Number of total miles on the truck

Average number of safety events categorized as “severe braking”

Years of life of the truck

Average idle time

Average number of dealership visits per year.

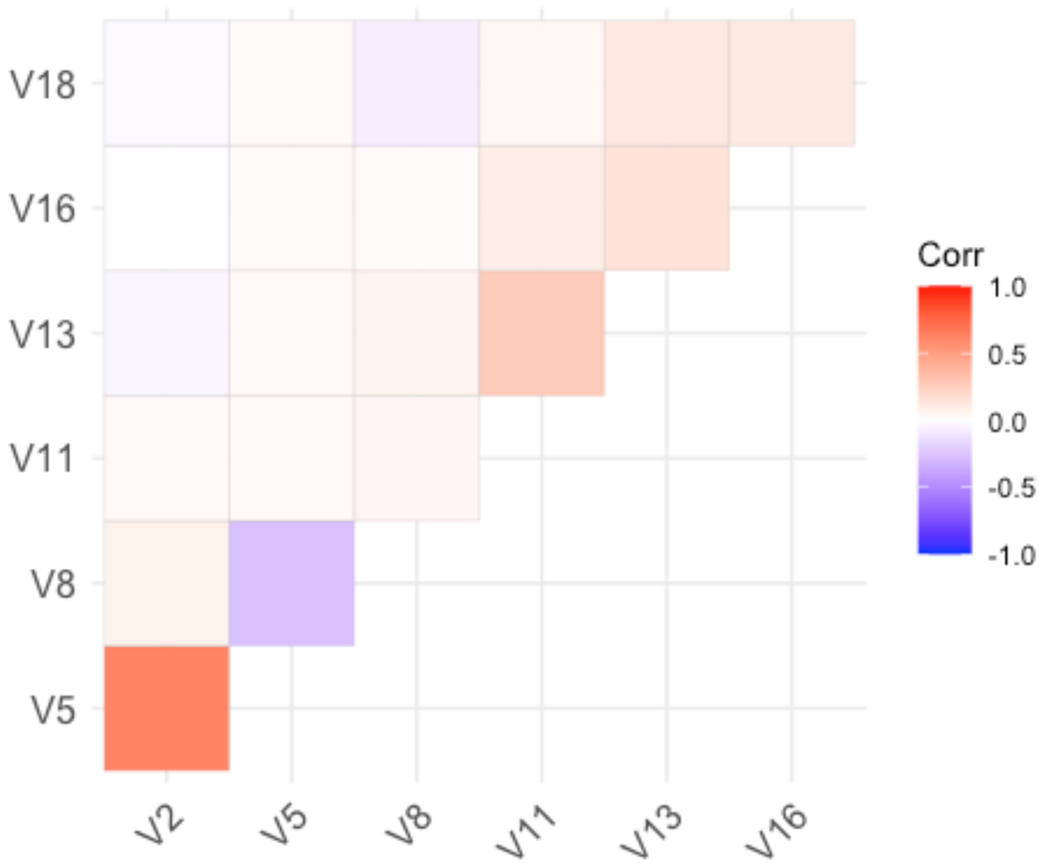
Question 10.3

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.

```
install.packages("ggcorrplot")
install.packages("ggplot2")
install.packages("boot")
library(ggplot2)
library(boot)

##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
##
##     melanoma

library(ggcorrplot)
library(pROC)

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

##
## Attaching package: 'pROC'

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

set.seed(4565)
german_credit=read.csv(file="german.txt",header=F,sep = " ")
gp=runif(nrow(german_credit))
german_credit=german_credit[order(gp),]
summary(german_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
```

```
#1=good, 2=bad
#converting these to 1 and 0 respectively
german_credit$V21=ifelse(german_credit$V21==2,0,german_credit$V21)
#Correlation plot to check for collinearity between factors
pred_factor=(names(Filter(is.factor,german_credit[,1:20])))
factor_col<-which(colnames(german_credit)%in%pred_factor)
pred_int=(names(Filter(is.numeric,german_credit[,1:20])))
corr_mat=round(cor(german_credit[,pred_int]),2)
ggcorrplot(corr_mat,type="upper")
```




```
#graph shows V5~V2 and V5~V8 have a stronger corr with each other.
#Split data in train/test set using 80/20 rule
train_number=sample(1:1000,800)
train_data_german<-german_credit[train_number,]
test_data_german<-german_credit[-train_number,]
#Basic logistic regression model
log_r_m1<-glm(V21~.,data=train_data_german,family = binomial(link="logit"))
summary(log_r_m1)

##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train_dat
a_german)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9479  -0.6403   0.3259   0.6772   2.2033
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.432e-01  1.313e+00   0.261 0.793784
## V1A12        3.582e-01  2.476e-01   1.447 0.147939
## V1A13        1.155e+00  4.423e-01   2.611 0.009018 **
```

```
## V1A14         1.875e+00  2.724e-01   6.886 5.74e-12 ***
## V2           -2.856e-02  1.058e-02  -2.700 0.006931 **
## V3A31        -1.965e-01  6.339e-01  -0.310 0.756549
## V3A32         2.883e-01  4.950e-01   0.582 0.560248
## V3A33         4.292e-01  5.336e-01   0.804 0.421211
## V3A34         1.272e+00  5.033e-01   2.527 0.011492 *
## V4A41         1.413e+00  4.178e-01   3.383 0.000716 ***
## V4A410        1.667e+00  8.863e-01   1.881 0.059981 .
## V4A42         5.950e-01  3.012e-01   1.975 0.048238 *
## V4A43         7.301e-01  2.895e-01   2.522 0.011660 *
## V4A44         2.566e-01  8.617e-01   0.298 0.765859
## V4A45        -1.595e-01  6.495e-01  -0.246 0.805996
## V4A46         4.402e-01  4.687e-01   0.939 0.347555
## V4A48         2.018e+00  1.320e+00   1.529 0.126142
## V4A49         4.057e-01  3.901e-01   1.040 0.298319
## V5           -1.527e-04  5.141e-05  -2.971 0.002968 **
## V6A62         8.549e-01  3.399e-01   2.516 0.011883 *
## V6A63         3.952e-01  4.411e-01   0.896 0.370356
## V6A64         1.394e+00  5.649e-01   2.467 0.013618 *
## V6A65         1.260e+00  3.125e-01   4.033 5.50e-05 ***
## V7A72         2.470e-01  4.881e-01   0.506 0.612807
## V7A73         2.774e-01  4.647e-01   0.597 0.550586
## V7A74         7.709e-01  5.107e-01   1.510 0.131163
## V7A75         4.359e-01  4.749e-01   0.918 0.358685
## V8           -4.700e-01  1.061e-01  -4.428 9.51e-06 ***
## V9A92        -1.174e-01  4.633e-01  -0.253 0.799956
## V9A93         5.804e-01  4.564e-01   1.272 0.203432
## V9A94        -1.010e-01  5.308e-01  -0.190 0.849168
## V10A102      -3.319e-01  4.399e-01  -0.755 0.450448
## V10A103       8.422e-01  4.974e-01   1.693 0.090437 .
## V11           2.708e-02  1.009e-01   0.268 0.788433
## V12A122      -3.278e-01  2.945e-01  -1.113 0.265622
## V12A123      -2.806e-01  2.698e-01  -1.040 0.298224
## V12A124      -1.234e+00  4.950e-01  -2.492 0.012690 *
## V13           2.272e-02  1.067e-02   2.128 0.033319 *
## V14A142       2.719e-01  4.607e-01   0.590 0.555059
## V14A143       8.716e-01  2.820e-01   3.091 0.001997 **
## V15A152       4.211e-01  2.778e-01   1.516 0.129539
## V15A153       7.202e-01  5.514e-01   1.306 0.191474
## V16          -1.276e-01  2.236e-01  -0.571 0.568229
## V17A172      -9.181e-01  8.106e-01  -1.133 0.257377
## V17A173      -8.908e-01  7.798e-01  -1.142 0.253344
## V17A174      -3.729e-01  7.813e-01  -0.477 0.633149
## V18          -3.454e-01  2.930e-01  -1.179 0.238445
## V19A192       3.227e-01  2.315e-01   1.394 0.163248
## V20A202       1.393e+00  7.434e-01   1.873 0.061035 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 972.25  on 799  degrees of freedom
## Residual deviance: 685.55  on 751  degrees of freedom
## AIC: 783.55
##
## Number of Fisher Scoring iterations: 5
```

```
#To improve model, only variables with significance at 10% are chosen (V1, v2
, v3,v4,v5,v6,v8,v10,v12,v20) (with p<0.05)
log_r_m2<-glm(V21~.,data=train_data_german[,c(1,2,3,4,5,6,8,10,12,20,21)],fam
ily =  binomial(link="logit"))
summary(log_r_m2)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train_dat
a_german[,
##     c(1, 2, 3, 4, 5, 6, 8, 10, 12, 20, 21)])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5464  -0.7111   0.3946   0.7028   2.3823
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.479e-01  6.151e-01   1.216 0.224008
## V1A12        3.012e-01  2.317e-01   1.300 0.193644
## V1A13        1.125e+00  4.164e-01   2.702 0.006900 **
## V1A14        1.819e+00  2.564e-01   7.095 1.29e-12 ***
## V2          -2.882e-02  9.925e-03  -2.904 0.003686 **
## V3A31       -2.027e-01  5.740e-01  -0.353 0.723997
## V3A32        6.000e-01  4.513e-01   1.330 0.183629
## V3A33        7.090e-01  5.113e-01   1.387 0.165584
## V3A34        1.577e+00  4.773e-01   3.304 0.000953 ***
## V4A41        1.376e+00  3.936e-01   3.496 0.000473 ***
## V4A410       1.635e+00  8.065e-01   2.027 0.042679 *
## V4A42        3.453e-01  2.808e-01   1.230 0.218792
## V4A43        6.209e-01  2.702e-01   2.298 0.021553 *
## V4A44       -6.544e-02  8.280e-01  -0.079 0.937006
## V4A45        1.481e-01  6.552e-01   0.226 0.821228
## V4A46        1.589e-01  4.467e-01   0.356 0.722053
## V4A48        1.688e+00  1.203e+00   1.402 0.160816
## V4A49        4.097e-01  3.664e-01   1.118 0.263517
## V5          -9.487e-05  4.739e-05  -2.002 0.045314 *
## V6A62        5.996e-01  3.231e-01   1.856 0.063499 .
## V6A63        4.255e-01  4.232e-01   1.005 0.314755
## V6A64        1.002e+00  5.158e-01   1.943 0.052066 .
## V6A65        1.169e+00  2.922e-01   4.000 6.32e-05 ***
## V8          -3.314e-01  9.849e-02  -3.365 0.000765 ***
## V10A102     -4.463e-01  4.345e-01  -1.027 0.304315
```

```
## V10A103      5.977e-01  4.574e-01   1.307 0.191319
## V12A122     -1.789e-01  2.767e-01  -0.646 0.518040
## V12A123     -1.676e-01  2.529e-01  -0.663 0.507483
## V12A124     -4.826e-01  3.254e-01  -1.483 0.138041
## V20A202      1.058e+00  6.966e-01   1.519 0.128747
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 972.25  on 799  degrees of freedom
## Residual deviance: 730.84  on 770  degrees of freedom
## AIC: 790.84
##
## Number of Fisher Scoring iterations: 5

abs(BIC(log_r_m1))

## [1] 1013.096

abs(BIC(log_r_m2))

## [1] 931.3772

#our optimized model has the lowest BIC, so it looks like it best fits the da
ta

#Using ROC and AUC to determine which threshold is the best, and what model i
s the best fit
AUC_vec<-c()
#We are deciding which threshold value is the best between 0 and 1, by increm
ent of 0.1
vec<-seq(0,1,by =0.1)
for (k in vec){
  y_model<-ifelse(predict(log_r_m2,test_data_german[,c(1,2,3,4,5,6,8,10,12,20
,21)],type="response")<=k,0,1)
  A=roc(test_data_german$V21,y_model)
  AUC_value=A$auc #area under the curve
  AUC_vec<-c(AUC_vec,AUC_value)
}

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases
```

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

values_thresholds=vec
index=which.max(AUC_vec) #max AUC value
optimal_t<-values_thresholds[index] #optimal threshold
Max_AUC<-AUC_vec[index]
optimal_t

## [1] 0.6

Max_AUC

## [1] 0.6871741
```

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.

```
#Start with 0.6 based on the results from our AUC/ROC analysis
#if predicted response is <=.6, mark as "bad" credit or 0, else "good credit risk.
#start with our base logistic data
pred_log_r_m1<-ifelse(predict(log_r_m1,test_data_german,type="response")<=.6,
```

```
0,1)
table(pred_log_r_m1,test_data_german[,21])

##
## pred_log_r_m1   0   1
##             0  35  23
##             1  28 114

sum(train_data_german$V21==0)

## [1] 237

#It is believed false positives are 5X as costly as false negatives, they are
weighed 5:1
cost_fp<-5
cost_fn<-1
train_data_score1<-table(pred_log_r_m1,test_data_german[,21])[2,1]*cost_fp+ta
ble(pred_log_r_m1,test_data_german[,21])[1,2]*cost_fn
paste0("The baseline train data score is : ",train_data_score1)

## [1] "The baseline train data score is : 163"

#then find score with our optimized logistic data.
pred_log_r_m2<-ifelse(predict(log_r_m2,test_data_german[,c(1,2,3,4,5,6,8,10,1
2,20,21)],type="response")<=.6,0,1)
table(pred_log_r_m2,test_data_german[,21])

##
## pred_log_r_m2   0   1
##             0  36  27
##             1  27 110

train_data_score2<-table(pred_log_r_m2,test_data_german[,21])[2,1]*cost_fp+ta
ble(pred_log_r_m2,test_data_german[,21])[1,2]*cost_fn
paste0("The new baseline train data score is ",train_data_score2) #one less f
alse positive, but 4 more false negatives, and we have a lower score by 1 poi
nt

## [1] "The new baseline train data score is 162"

# I previously used .6 as the cutoff, however, this may affect our model badl
y, and given the new information about cost of false positives and false nega
tives, we would like to avoid all false positive ideally. However, if too man
y creditors are rejected, there are less chances for a bank to make money, so
there needs to be a balance.Creating a loop to find a new optimal threshold.

train_data_costs<-c()
vec2<-seq(0.1,0.9,by =0.1)
for (k in vec2){
  pred_log_r<-ifelse(predict(log_r_m2,test_data_german[,c(1,2,3,4,5,6,8,10,12
,20,21)],type="response")<=k,0,1)
 train_data_cost<-table(pred_log_r,test_data_german[,21])[2,1]*cost_fp+(table
```

```r
(pred_log_r,test_data_german[,21]))[1,2]*cost_fn
  train_data_costs<-c(train_data_costs,train_data_cost)
}
train_data_costs

## [1] 310 295 269 233 177 162 146 139 128

opt_cost_index<-which.min(train_data_costs)#minimum index for cost
opt_cost_index

## [1] 9

opt_threshold<-vec2[opt_cost_index] #matching this to the correct threshold.
opt_threshold

## [1] 0.9

#new optimal threshold is 0.9, predicting with 0.9 now.

pred_log_r_m3<-ifelse(predict(log_r_m2,test_data_german[,c(1,2,3,4,5,6,8,10,1
2,20,21)],type="response")<=.9,0,1)
table(pred_log_r_m3,test_data_german[,21])

##
## pred_log_r_m3  0  1
##             0 55 88
##             1  8 49

#Score
train_data_score3<-table(pred_log_r_m3,test_data_german[,21])[2,1]*cost_fp+ta
ble(pred_log_r_m3,test_data_german[,21])[1,2]*cost_fn
paste0("The new baseline train data score is ",train_data_score3)

## [1] "The new baseline train data score is 128"

AUC_vec[opt_cost_index]

## [1] 0.6576874

 #We have reduced the number of false positives by 29% (from 27 to 8), howeve
r, the number of false negatives has increased by almost 3.25x (from 27 to 88
). In this case we are now classifying good credits as bad. banks with differ
ent risk threshold may have different preferences for false positive/false ne
gative tradeoffs. Associated AUC for this threshold is 0.644. Score has decre
ased from 167 to 128
```