# 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).

In [262]:
```
# Clear environment

rm(list = ls())
```

In [280]:
```
##########LIBRARY##########
library(tidyr) # to use tibble function
library(DAAG)
library(tree) #to create regression tree using tree package
#library("ggplot2")
#install.packages("devtools")
#install.packages("corrplot")
#library("devtools")
#library("corrplot") # to use Correlation plot
#library(plyr) #to use arrange function
```

In [265]:
```
##########INGEST FILE##########

#crime<- read.table("C:\\Preethi\\R\\USCrimes.txt",header=TRUE,stringsAsFactors = FALSE,sep="\t")
#head(data,10)
#colClasses = c("numeric", "numeric", "numeric")
#colClasses
data = read.table("C:\\Preethi\\R\\USCrimes.txt",header=TRUE,stringsAsFactors = FALSE,sep="\t")
#%>% as_tibble()
head(data,2)
```

A data.frame: 2 × 16

| | M | So | Ed | Po1 | Po2 | LF | M.F | Pop | NW | U1 | U2 | Wealth | Ineq | Prob | Time | Crime |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <int> |
| **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 | 26.2011 | 791 |
| **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 | 25.2999 | 1635 |

# Question 10.1 a # Regression Tree

## Approach taken

1.fit a regression tree.Study the predictors and splits used in the regression tree will show that

- only 4 variables Po1,Pop, LF and NW are used for Tree split
- logic/value used to split is present in $frame parameter of the model.
- Deviance = 47390
- Number of terminal nodes =7 2.plot to visualize the tree structure with the splits and predictors used for the split
- PRUNING:
  -Limit the Terminal nodes to 4 and visualize the Deviance ( quality-of-fit statistic) vs. the number of terminal nodes of the tree. -When I changed the terminal nodes count, the best visualiation on deviance is obtained with 4 and hence retained the same.
- Interpret the model -Obtain the Sum of square error and residual for predicted vs. actual values along with the model' R sqaured value(72%) -for multiple values of Terminal nodes (from 1-7), obtain the residual error to understand the model with lowest errors. Finally use cross validation to compare the residual errors of the cross validated model #TAKEAWAYS # These errors are larger with cross validated model # indicating that the model does not do well with Cross validation; # it's just way overfit with original data

In [266]:
```
# Fit a regression tree function to the crime data
tree.data <- tree(Crime~., data = data)
summary(tree.data)
#Only 4 variables Po1,Pop, LF and NW are used in regression tress
#Deviance = 47390
#Number of terminal nodes =7
```

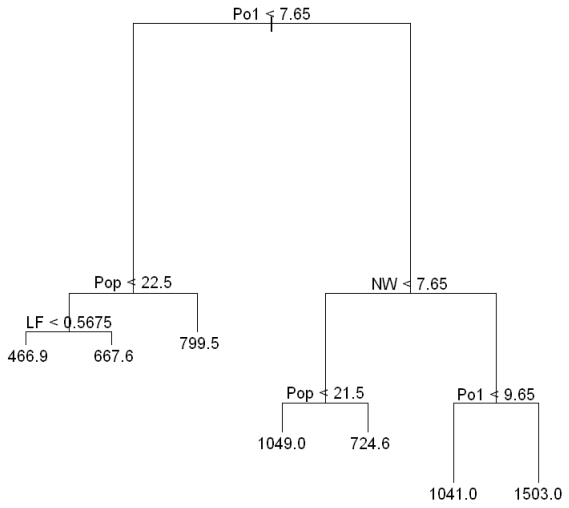
```
Regression tree:
tree(formula = Crime ~ ., data = data)
Variables actually used in tree construction:
[1] "Po1" "Pop" "LF"  "NW"
Number of terminal nodes:  7
Residual mean deviance:  47390 = 1896000 / 40
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-573.900  -98.300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

In [17]:
```
# More information about the way the tree was split
#Only 4 variables Po1,Pop, LF and NW have split amount (Tree splits are based on these variables)
tree.data$frame
```

A data.frame: 13 × 5

| | var | n | dev | yval | splits |
|---|---|---|---|---|---|
| | <fct> | <dbl> | <dbl> | <dbl> | <chr[,2]> |
| **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 | , |

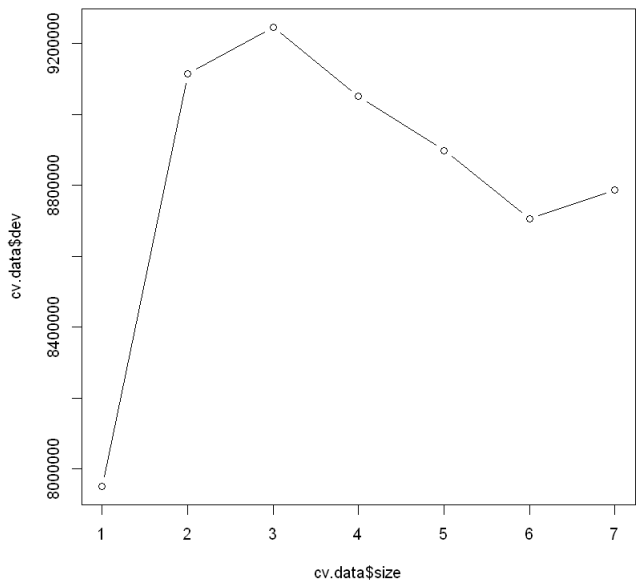
In [15]:
```
# Plot the regression tree

plot(tree.data)
text(tree.data)
```




# Determine if pruning the tree will improve performance through cross-validation

In [ ]:
```
#How?
# by looking at the deviance of trees with different number of terminal nodes.
# Deviance is a quality-of-fit statistic.
# The x-axis represents the number of terminal nodes.
```

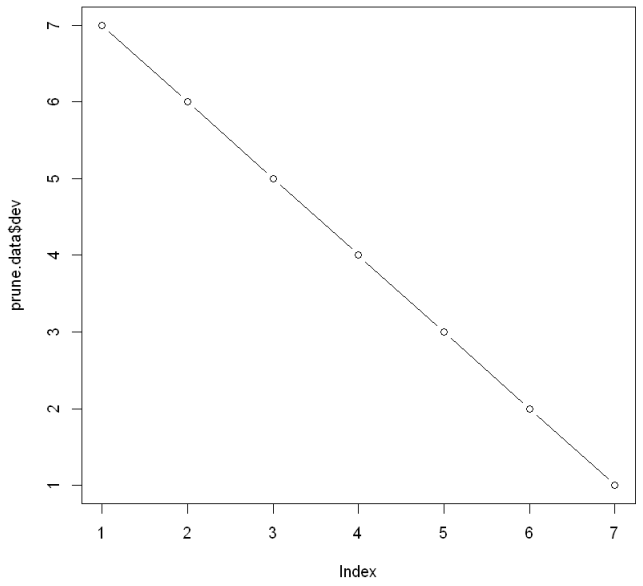
In [271]:
```
#x-axis is # of terminal nodes(from tree viz above,it is 7)
cv.data <- cv.tree(tree.data)
plot(cv.data$size, cv.data$dev, type = "b")
```



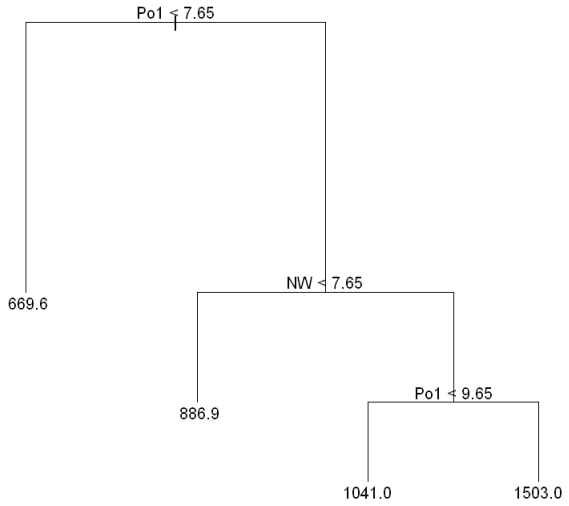

In [19]:
```
# This plot suggests that we get the best fit using all of the terminal nodes in the
# tree that we already plotted.
# prune the tree by limiting the number of terminal nodes of the tree if this
# is desired. This is just one example of how you can prune a regression tree.
```

In [276]:
```
termnodes <- 4 # Change this parameter to determine the leaf nodes
prune.data <- prune.tree(tree.data, best = termnodes)
```

In [277]:
```
#Plot the pruned tree' deviation
plot(cv.data$size, prune.data$dev, type = "b")
```



In [26]:
```
# Plot the pruned tree
plot(prune.data)
text(prune.data)
```

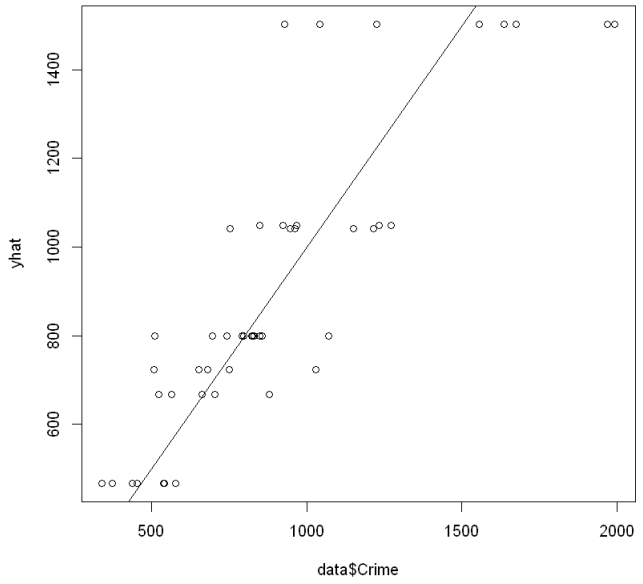



# Analysis#1 (Total variation in the dependent variable)

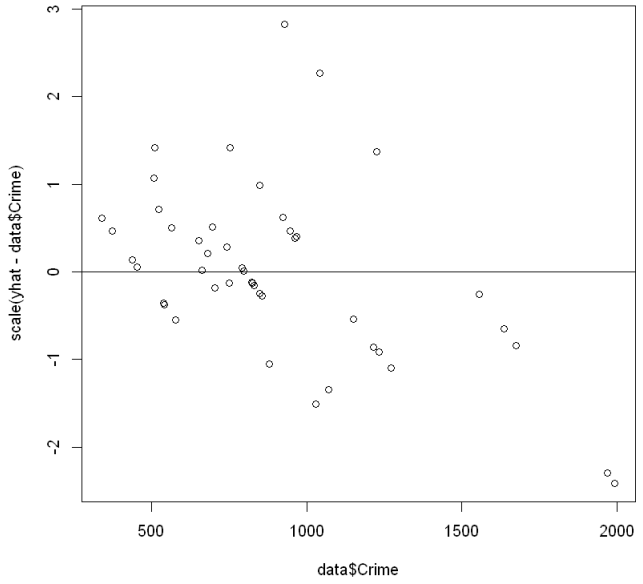
In [29]:
```
#Decision: use unpruned regression model?
# Calculate SSres of the unpruned regression model.

yhat <- predict(tree.data)
SSres <- sum((yhat-data$Crime)^2)
SSres
```

1895721.65941558

In [30]:
```
# Plot of actual vs. predicted crime values

plot(data$Crime, yhat)
abline(0,1)
```



In [31]:
```
# Plot the residuals

plot(data$Crime, scale(yhat - data$Crime))
abline(0,0)
```




In [35]:
```
prune.tree(tree.data)$size
prune.tree(tree.data)$dev
#below shows all terminal nodes and the residual errors with each of the nodes
```

7 · 6 · 5 · 4 · 3 · 2 · 1

1895721.65941558 · 2013256.60227273 · 2276669.50227273 · 2632631.25326087 · 3364043.3068323 · 4383405.97826087 · 6880927.65957447

In [36]:
```
# Now, compare with the sum of squared errors in cross-validation:

cv.data <- cv.tree(tree.data) # cross-validation
cv.data$size
cv.data$dev
```

7 · 6 · 5 · 4 · 3 · 2 · 1

8919913.73415682 · 8896137.01209577 · 8853786.97647446 · 8916545.24474757 · 8127298.23326752 · 7539144.63193484 · 8565824.837243

In [ ]:
```
#RESULTS
# These errors are larger
# indicating that the model does not do well with Cross validation;
# it's just way overfit with original data
```

# Analysis#2 (R2 value)

In [32]:
```
#Quality of the Regression model
#Calculate SStot and R-squared of this model on the training data

SStot <- sum((data$Crime - mean(data$Crime))^2)
R2 <- 1 - SSres/SStot
R2
#R2 is 0.72 showing the model fit is good
```

0.724496208475934

# Question 10.1 # Random Forest ##Approach Taken

1) Use randomforest libary to create a model. I have setup number of splits as 4 based on regression trees optimal split

# mtry:Number of variables available for splitting at each tree node.

It generates a forest with 500 trees and mean square error of 82187 2) Fit the model and compare the fit with the actual data points. 3) calculate the sum of square residual and plot the residuals 4) Random trees R2 value is 0.43 vs. 0.72 Regression trees 5) Also , the Residual error is far less than the regression tree (as noted in the lectures)

In [43]:
```
#LIBRARIES
#install.packages("randomForest")
library(randomForest)
```

In [45]:
```
numpred <- 4
rf.data <- randomForest(Crime~., data = data, mtry = numpred, importance = TRUE)
rf.data
```

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

          Mean of squared residuals: 82187.74
                    % Var explained: 43.86
```

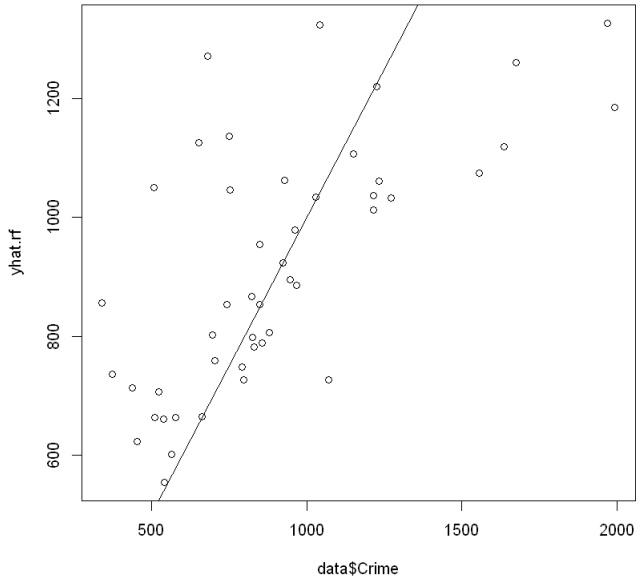
In [ ]:
```
#Above shows 500 trees and mean square error is 82187
```

# Random Forest Model' quality

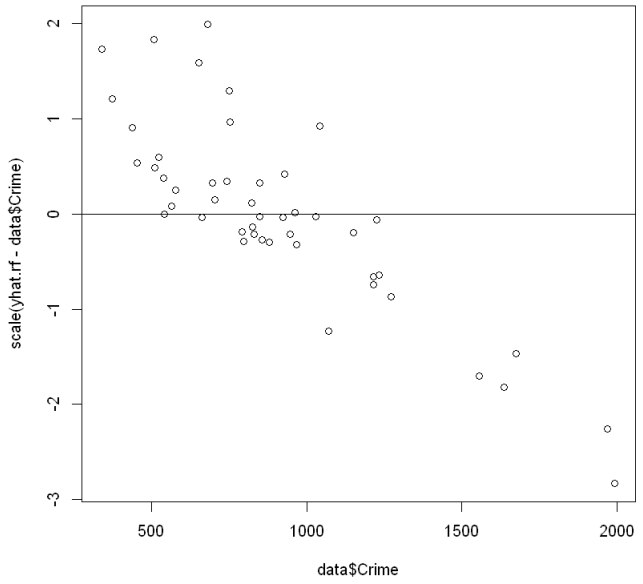
In [47]:
```
# Calculate SSres of the random forest model

yhat.rf <- predict(rf.data)
SSres <- sum((yhat.rf-data$Crime)^2)
SSres
```

3862823.55033399

In [49]:
```
# Plot of actual vs. predicted crime values

plot(data$Crime, yhat.rf)
abline(0,1)
```



In [50]:
```
# Plot residuals

plot(data$Crime, scale(yhat.rf - data$Crime))
abline(0,0)
```




# Analysis and Intepreting the Random forest trees

In [59]:
```
# Calculate SStot and R-squared of this model

SStot <- sum((data$Crime - mean(data$Crime))^2)
R2 <- 1 - SSres/SStot
R2
#Model quality of random is 0.43 vs. 0.72 Regression trees
```

0.438618782024389

In [60]:
```
# leave-one-out cross-validation on Random forest

SSE <- 0

for (i in 1:nrow(data)) {
  rd <- randomForest(Crime~., data = data[-i,], mtry = numpred, importance = TRUE)
  SSE = SSE + (predict(rd,newdata=data[i,]) - data[i,16])^2
}
1 - SSE/SStot
#Residual error(The error is much better than the regression tree)
```

**1:** 0.43553737284783

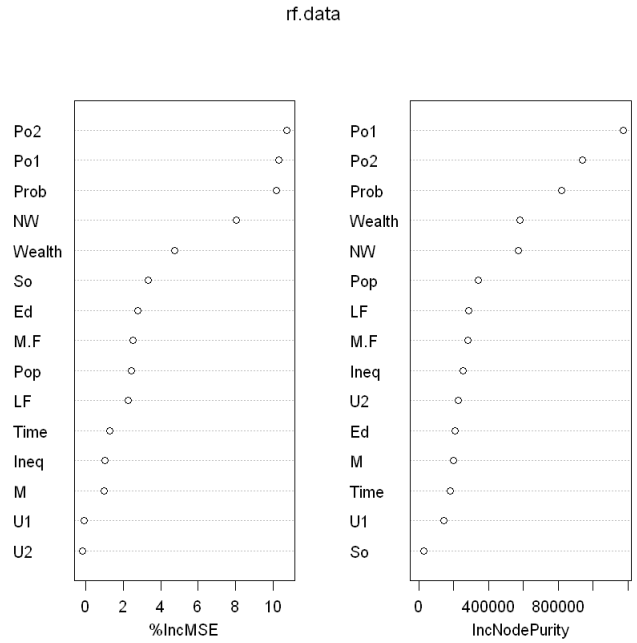
In [55]:
```
#This model seems to take the best of the process just like in the lectures
```

In [57]:
```
importance(rf.data)
```

A matrix: 15 × 2 of type dbl

| | %IncMSE | IncNodePurity |
|---|---|---|
| **M** | 1.00434273 | 199324.76 |
| **So** | 3.32835988 | 30718.92 |
| **Ed** | 2.77855877 | 208227.61 |
| **Po1** | 10.29965085 | 1174840.68 |
| **Po2** | 10.72770654 | 941376.07 |
| **LF** | 2.27372307 | 287574.89 |
| **M.F** | 2.51930528 | 282323.77 |
| **Pop** | 2.43114058 | 340008.58 |
| **NW** | 8.03090795 | 569033.69 |
| **U1** | -0.09193546 | 141520.89 |
| **U2** | -0.17281743 | 226691.08 |
| **Wealth** | 4.72864245 | 579328.33 |
| **Ineq** | 1.03493447 | 253017.29 |
| **Prob** | 10.16129757 | 817676.11 |
| **Time** | 1.28598153 | 182623.84 |

In [58]:
```
# Plots of these importance measures

varImpPlot(rf.data)
```




# 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.

# Question 10.3(a)

## Input:

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german (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 (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29) ), ##Ask: 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. ##Ideas: 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.

# Approach for Logistic Regression

## PART#1

1)Tranformation: Since binomial family of glm recognises 0 and 1 as the classfication values, I converted the 1s and 2s to 0 and 1 for response variable(y) 2) Using validation split the data into training and validation set 3)Using glm and logit (logistic regression), fit a model. AIC is 713 for this model using all predictors Input Transformation: 4) 2nd iteration=>fit the model only using categorical predictors.

# AIC is 734(high) vs.701 using all predictorsfit the model only using categorical predictors.

5) 3rd iteration=>fit the model only using SIGNIFICANT categorical predictors.

# This model has a lower AIC of 98(when run for first time), the second time shows 701 (when re-run) vs. 734 with all predictors shows the model' fit

6)In addition to binary conversion of response, the predictors used in the 3rd iteration are converted to 0 and 1 classification value as not all values of the categorical predictors are significant 7) Predict the model and created a confusion matrix vs. the validation set. 8)Accuracy of the model is 74% (# Model's accuracy is (170 + 43) / (170 + 53 + 43 + 34) = 74%.) and specificity if 84% 9)Using proc library, plotted the ROC curve with area under the curve(AUC) = 69%. This means that whenever a sample is chosen from the response group and another sample is chosen from the non-response group, then the model will correctly classify both the samples 68% of the times!

In [228]:
```
# Set the seed to produce reproducible results as random sampling is done in
#the next step

set.seed(1)
rm(list = ls())
```

In [279]:
```
#LIbRARY
library(pROC) #ROC graph
```

In [281]:
```
##########INGEST FILE##########

data = read.table("C:\\Preethi\\R\\german.data",header=TRUE,stringsAsFactors = FALSE,sep=" ") %>% as_tibble()
head(data,3)
```

A tibble: 3 × 21

| A11 | X6 | A34 | A43 | X1169 | A65 | A75 | X4 | A93 | A101 | ... | A121 | X67 | A143 | A152 | X2 | A173 | X1 | A192 | A201 | X1.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | ... | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <int> | <chr> | <chr> | <int> |
| A12 | 48 | A32 | A43 | 5951 | A61 | A73 | 2 | A92 | A101 | ... | A121 | 22 | A143 | A152 | 1 | A173 | 1 | A191 | A201 | 2 |
| A14 | 12 | A34 | A46 | 2096 | A61 | A74 | 2 | A93 | A101 | ... | A121 | 49 | A143 | A152 | 1 | A172 | 2 | A191 | A201 | 1 |
| A11 | 42 | A32 | A42 | 7882 | A61 | A74 | 2 | A93 | A103 | ... | A122 | 45 | A143 | A153 | 1 | A173 | 2 | A191 | A201 | 1 |

In [282]:
```
# Since binomial family of glm recognises 0 and 1 as the classfication values,
# convert 1s and 2s to 0s and 1s for the response variable

data$X1.1[data$X1.1==1]<- 0
data$X1.1[data$X1.1==2]<- 1
#data$X1.1==2
```

In [283]:
```
# Divide the data into 70% training and 30% test/validation data

m <- nrow(data)
#sample(1:m, size = round(m*0.7), replace = FALSE)
trn <- sample(1:m, size = round(m*0.7), replace = FALSE)
d.learn <- data[trn,]
d.valid <- data[-trn,]
head(d.learn,2)
head(d.valid,2)
```

A tibble: 2 × 21

| A11 | X6 | A34 | A43 | X1169 | A65 | A75 | X4 | A93 | A101 | ... | A121 | X67 | A143 | A152 | X2 | A173 | X1 | A192 | A201 | X1.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | ... | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <int> | <chr> | <chr> | <dbl> |
| A14 | 24 | A34 | A46 | 1927 | A65 | A73 | 3 | A92 | A101 | ... | A123 | 33 | A143 | A152 | 2 | A173 | 1 | A192 | A201 | 0 |
| A14 | 24 | A32 | A41 | 9277 | A65 | A73 | 2 | A91 | A101 | ... | A124 | 48 | A143 | A153 | 1 | A173 | 1 | A192 | A201 | 0 |

A tibble: 2 × 21

| A11 | X6 | A34 | A43 | X1169 | A65 | A75 | X4 | A93 | A101 | ... | A121 | X67 | A143 | A152 | X2 | A173 | X1 | A192 | A201 | X1.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | ... | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <int> | <chr> | <chr> | <dbl> |
| A14 | 12 | A34 | A46 | 2096 | A61 | A74 | 2 | A93 | A101 | ... | A121 | 49 | A143 | A152 | 1 | A172 | 2 | A191 | A201 | 0 |
| A11 | 42 | A32 | A42 | 7882 | A61 | A74 | 2 | A93 | A103 | ... | A122 | 45 | A143 | A153 | 1 | A173 | 2 | A191 | A201 | 0 |

In [235]:
```
head(d.learn,2)
```

A tibble: 2 × 21

| A11 | X6 | A34 | A43 | X1169 | A65 | A75 | X4 | A93 | A101 | ... | A121 | X67 | A143 | A152 | X2 | A173 | X1 | A192 | A201 | X1.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | ... | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <int> | <chr> | <chr> | <dbl> |
| A14 | 12 | A32 | A43 | 886 | A65 | A73 | 4 | A92 | A101 | ... | A123 | 21 | A143 | A152 | 1 | A173 | 1 | A191 | A201 | 0 |
| A14 | 18 | A32 | A43 | 1453 | A61 | A72 | 3 | A92 | A101 | ... | A121 | 26 | A143 | A152 | 1 | A173 | 1 | A191 | A201 | 0 |

In [284]:
```
# Develop the logistic regression model
# 1st iteration: Use all the available variables

reg = glm(X1.1 ~.,family=binomial(link = "logit"),data=d.learn)
summary(reg)
#AIC is 701 for this model using all predictors
```

```
Call:
glm(formula = X1.1 ~ ., family = binomial(link = "logit"), data = d.learn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4552  -0.6884  -0.3540   0.6043   2.5817

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.036e-01  1.344e+00  -0.449 0.653269
A11A12      -3.439e-01  2.745e-01  -1.253 0.210360
A11A13      -9.660e-01  4.731e-01  -2.042 0.041154 *
A11A14      -1.722e+00  2.870e-01  -5.999 1.98e-09 ***
X6           2.277e-02  1.139e-02   1.999 0.045632 *
A34A31       2.201e-02  6.894e-01   0.032 0.974528
A34A32      -7.375e-01  5.441e-01  -1.356 0.175229
A34A33      -1.143e+00  5.978e-01  -1.913 0.055768 .
A34A34      -1.990e+00  5.600e-01  -3.554 0.000380 ***
A43A41      -1.639e+00  4.431e-01  -3.700 0.000216 ***
A43A410     -1.786e+01  5.754e+02  -0.031 0.975240
A43A42      -7.737e-01  3.164e-01  -2.446 0.014460 *
A43A43      -1.122e+00  3.121e-01  -3.594 0.000326 ***
A43A44      -1.484e+00  1.021e+00  -1.454 0.146049
A43A45      -9.324e-01  6.418e-01  -1.453 0.146283
A43A46       1.570e-01  4.994e-01   0.314 0.753300
A43A48      -1.850e+00  1.228e+00  -1.506 0.132105
A43A49      -6.140e-01  4.041e-01  -1.519 0.128671
X1169        1.854e-04  5.593e-05   3.314 0.000918 ***
A65A62      -3.806e-01  3.764e-01  -1.011 0.312004
A65A63       1.390e-01  4.849e-01   0.287 0.774423
A65A64      -1.048e+00  5.970e-01  -1.755 0.079218 .
A65A65      -1.022e+00  3.195e-01  -3.198 0.001383 **
A75A72      -6.937e-03  5.078e-01  -0.014 0.989101
A75A73      -5.607e-01  4.930e-01  -1.137 0.255459
A75A74      -9.946e-01  5.335e-01  -1.864 0.062306 .
A75A75      -7.558e-01  4.997e-01  -1.513 0.130404
X4           3.220e-01  1.091e-01   2.952 0.003160 **
A93A92       1.665e-01  4.966e-01   0.335 0.737444
A93A93      -3.960e-01  4.863e-01  -0.814 0.415464
A93A94       8.984e-03  5.842e-01   0.015 0.987731
A101A102     4.990e-01  5.082e-01   0.982 0.326169
A101A103    -8.940e-01  4.903e-01  -1.823 0.068241 .
X4.1         9.388e-02  1.033e-01   0.909 0.363404
A121A122     1.930e-01  3.075e-01   0.628 0.530215
A121A123     1.201e-01  2.899e-01   0.414 0.678581
A121A124     6.877e-01  5.164e-01   1.332 0.182950
X67         -6.252e-03  1.084e-02  -0.577 0.564252
A143A142     2.889e-01  5.176e-01   0.558 0.576658
A143A143    -5.152e-01  3.174e-01  -1.623 0.104615
A152A152    -6.927e-02  2.972e-01  -0.233 0.815687
A152A153    -5.819e-01  6.046e-01  -0.962 0.335841
X2           5.239e-01  2.450e-01   2.139 0.032473 *
A173A172     3.685e-01  8.084e-01   0.456 0.648471
A173A173     5.959e-01  7.732e-01   0.771 0.440895
```

```
A173A174     2.926e-01  7.938e-01   0.369 0.712403
X1           1.734e-01  3.063e-01   0.566 0.571398
A192A192    -5.592e-01  2.528e-01  -2.213 0.026930 *
A201A202    -1.616e+00  7.981e-01  -2.025 0.042908 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 831.35  on 698  degrees of freedom
Residual deviance: 603.33  on 650  degrees of freedom
AIC: 701.33

Number of Fisher Scoring iterations: 14
```

In [285]:
```
# 2nd iteration: Use all the variables found significant in
# the 1st iteration.
# For categorical variables, if any value is significant,
# we'll keep all values in the model.

reg2 = glm(X1.1 ~ A11+A34+A43+A65+A75+A93+A101+A121+A143+A152+A173+A192,family=binomial(link = "logit"),data=d.learn)
summary(reg2)
#AIC is 734(high) vs.701 using al predictors
```

```
Call:
glm(formula = X1.1 ~ A11 + A34 + A43 + A65 + A75 + A93 + A101 +
    A121 + A143 + A152 + A173 + A192, family = binomial(link = "logit"),
    data = d.learn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0605  -0.7468  -0.4310   0.7383   2.5258

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.7787     1.0370   1.715 0.086304 .
A11A12       -0.3092     0.2567  -1.205 0.228353
A11A13       -1.2994     0.4495  -2.891 0.003843 **
A11A14       -1.6608     0.2703  -6.144 8.06e-10 ***
A34A31       -0.4046     0.6426  -0.630 0.528894
A34A32       -1.1712     0.4948  -2.367 0.017922 *
A34A33       -1.1464     0.5671  -2.022 0.043215 *
A34A34       -1.9976     0.5258  -3.799 0.000145 ***
A43A41       -1.2852     0.4129  -3.113 0.001855 **
A43A410     -16.1155   665.6749  -0.024 0.980686
A43A42       -0.6501     0.2947  -2.206 0.027355 *
A43A43       -0.8773     0.2906  -3.019 0.002540 **
A43A44       -1.2549     0.9590  -1.309 0.190678
A43A45       -0.6199     0.6339  -0.978 0.328135
A43A46        0.2988     0.4666   0.640 0.521971
A43A48       -1.7380     1.2223  -1.422 0.155043
A43A49       -0.3385     0.3785  -0.894 0.371158
A65A62       -0.3330     0.3573  -0.932 0.351299
A65A63       -0.1480     0.4570  -0.324 0.746085
A65A64       -1.0011     0.5709  -1.754 0.079483 .
A65A65       -0.7361     0.2988  -2.463 0.013772 *
A75A72        0.1590     0.4833   0.329 0.742206
A75A73       -0.3145     0.4708  -0.668 0.504142
A75A74       -0.5531     0.5025  -1.101 0.271029
A75A75       -0.4755     0.4755  -1.000 0.317318
A93A92        0.1953     0.4654   0.420 0.674825
A93A93       -0.1786     0.4516  -0.395 0.692498
A93A94       -0.0509     0.5515  -0.092 0.926464
A101A102      0.5811     0.4780   1.216 0.224100
A101A103     -0.8215     0.4718  -1.741 0.081676 .
A121A122      0.3016     0.2935   1.028 0.304114
A121A123      0.4345     0.2763   1.572 0.115919
A121A124      0.9364     0.4603   2.034 0.041919 *
A143A142      0.5688     0.5059   1.124 0.260865
A143A143     -0.3934     0.3029  -1.299 0.193981
A152A152     -0.1029     0.2710  -0.380 0.704200
A152A153     -0.1952     0.5334  -0.366 0.714417
A173A172      0.1508     0.7415   0.203 0.838828
A173A173      0.5272     0.7155   0.737 0.461275
A173A174      0.4938     0.7349   0.672 0.501628
A192A192     -0.2980     0.2284  -1.305 0.191944
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 831.35  on 698  degrees of freedom
Residual deviance: 652.87  on 658  degrees of freedom
AIC: 734.87

Number of Fisher Scoring iterations: 14
```

In [286]:
```
# 3rd iteration: Use all the variables found significant in
# the 1st iteration.
# For categorical variables, if any value is significant,
# we'll keep all values in the model.
#Of these A11,A34,A43,A65,A121,A143 are significant
reg2 = glm(X1.1 ~ A11+A34+A43+A65+A121+A143 ,family=binomial(link = "logit"),data=d.learn)
summary(reg)
# This model has a lower AIC of 98 (704 second time) vs. 753 with all predictors shows the model' fit
```

```
Call:
glm(formula = X1.1 ~ ., family = binomial(link = "logit"), data = d.learn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4552  -0.6884  -0.3540   0.6043   2.5817

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.036e-01  1.344e+00  -0.449 0.653269
A11A12      -3.439e-01  2.745e-01  -1.253 0.210360
A11A13      -9.660e-01  4.731e-01  -2.042 0.041154 *
A11A14      -1.722e+00  2.870e-01  -5.999 1.98e-09 ***
X6           2.277e-02  1.139e-02   1.999 0.045632 *
A34A31       2.201e-02  6.894e-01   0.032 0.974528
A34A32      -7.375e-01  5.441e-01  -1.356 0.175229
A34A33      -1.143e+00  5.978e-01  -1.913 0.055768 .
A34A34      -1.990e+00  5.600e-01  -3.554 0.000380 ***
A43A41      -1.639e+00  4.431e-01  -3.700 0.000216 ***
A43A410     -1.786e+01  5.754e+02  -0.031 0.975240
A43A42      -7.737e-01  3.164e-01  -2.446 0.014460 *
A43A43      -1.122e+00  3.121e-01  -3.594 0.000326 ***
A43A44      -1.484e+00  1.021e+00  -1.454 0.146049
A43A45      -9.324e-01  6.418e-01  -1.453 0.146283
A43A46       1.570e-01  4.994e-01   0.314 0.753300
A43A48      -1.850e+00  1.228e+00  -1.506 0.132105
A43A49      -6.140e-01  4.041e-01  -1.519 0.128671
X1169        1.854e-04  5.593e-05   3.314 0.000918 ***
A65A62      -3.806e-01  3.764e-01  -1.011 0.312004
A65A63       1.390e-01  4.849e-01   0.287 0.774423
A65A64      -1.048e+00  5.970e-01  -1.755 0.079218 .
A65A65      -1.022e+00  3.195e-01  -3.198 0.001383 **
A75A72      -6.937e-03  5.078e-01  -0.014 0.989101
A75A73      -5.607e-01  4.930e-01  -1.137 0.255459
A75A74      -9.946e-01  5.335e-01  -1.864 0.062306 .
A75A75      -7.558e-01  4.997e-01  -1.513 0.130404
X4           3.220e-01  1.091e-01   2.952 0.003160 **
A93A92       1.665e-01  4.966e-01   0.335 0.737444
A93A93      -3.960e-01  4.863e-01  -0.814 0.415464
A93A94       8.984e-03  5.842e-01   0.015 0.987731
A101A102     4.990e-01  5.082e-01   0.982 0.326169
A101A103    -8.940e-01  4.903e-01  -1.823 0.068241 .
X4.1         9.388e-02  1.033e-01   0.909 0.363404
A121A122     1.930e-01  3.075e-01   0.628 0.530215
A121A123     1.201e-01  2.899e-01   0.414 0.678581
A121A124     6.877e-01  5.164e-01   1.332 0.182950
X67         -6.252e-03  1.084e-02  -0.577 0.564252
A143A142     2.889e-01  5.176e-01   0.558 0.576658
A143A143    -5.152e-01  3.174e-01  -1.623 0.104615
A152A152    -6.927e-02  2.972e-01  -0.233 0.815687
A152A153    -5.819e-01  6.046e-01  -0.962 0.335841
X2           5.239e-01  2.450e-01   2.139 0.032473 *
A173A172     3.685e-01  8.084e-01   0.456 0.648471
A173A173     5.959e-01  7.732e-01   0.771 0.440895
```

```
A173A174     2.926e-01  7.938e-01   0.369 0.712403
X1           1.734e-01  3.063e-01   0.566 0.571398
A192A192    -5.592e-01  2.528e-01  -2.213 0.026930 *
A201A202    -1.616e+00  7.981e-01  -2.025 0.042908 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 831.35  on 698  degrees of freedom
Residual deviance: 603.33  on 650  degrees of freedom
AIC: 701.33

Number of Fisher Scoring iterations: 14
```

In [241]:
```
# But for the categorical variables, not all values are significant.
# So, we can create a binary variable for eachsignificant factor
#A11+A34+A43+A65+A121+A143

d.learn$A11A13[d.learn$X1.1 == "A11"] <- 1
d.learn$A11A13[d.learn$X1.1 == "A11"] <- 0

d.learn$A11A14[d.learn$X1.1 == "A14"] <- 1
d.learn$A11A14[d.learn$X1.1 == "A14"] <- 0

d.learn$A34A34[d.learn$X1.1 == "A34"] <- 1
d.learn$A34A34[d.learn$X1.1 == "A34"] <- 0

d.learn$A43A41[d.learn$X1.1 == "A43"] <- 1
d.learn$A43A41[d.learn$X1.1 == "A43"] <- 0

d.learn$A121A41[d.learn$X1.1 == "A121"] <- 1
d.learn$A121A41[d.learn$X1.1 == "A121"] <- 0

d.learn$A65A143[d.learn$X1.1 == "A143"] <- 1
d.learn$A65A143[d.learn$X1.1 == "A143"] <- 0
```

In [240]:
```
#Iterative do the intialization to avoid error with initialization of the previous step
head(d.learn,2)
```

A tibble: 2 × 27

| A11 | X6 | A34 | A43 | X1169 | A65 | A75 | X4 | A93 | A101 | ... | X1 | A192 | A201 | X1.1 | A11A13 | A11A14 | A34A34 | A43A41 | A121A41 | A65A143 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | <int> | <chr> | <chr> | ... | <int> | <chr> | <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| A14 | 12 | A32 | A43 | 886 | A65 | A73 | 4 | A92 | A101 | ... | 1 | A191 | A201 | 0 | NA | NA | NA | NA | NA | NA |
| A14 | 18 | A32 | A43 | 1453 | A61 | A72 | 3 | A92 | A101 | ... | 1 | A191 | A201 | 0 | NA | NA | NA | NA | NA | NA |

In [290]:
```
#predict mode
y_hat<-predict(reg,d.valid,type = "response")
y_hat
```

**1:** 0.0262541102961707 **2:** 0.335919191182075 **3:** 0.791915453614011 **4:** 0.134042920309172 **5:** 0.905349430697022 **6:** 0.110149866428091 **7:** 0.936895564169456 **8:** 0.0683581459915182 **9:** 0.0032065669899146 **10:** 0.123786616898688 **11:** 0.806732914611602 **12:** 0.290396059962923 **13:** 0.626174955647017 **14:** 0.489945782590698 **15:** 0.0315861821654858 **16:** 0.116111158168528 **17:** 0.304299915528803 **18:** 0.087136239684185 **19:** 0.056904030268832 **20:** 0.00850736822666429 **21:** 0.659238468390413 **22:** 0.0112474824223386 **23:** 0.443135284782538 **24:** 5.89009292225748e-09 **25:** 0.444062336182445 **26:** 0.733276322302777 **27:** 0.134015938422631 **28:** 0.0989783179031707 **29:** 0.0176673586471939 **30:** 0.434546618714602 **31:** 0.0516779413793954 **32:** 0.0198175555485388 **33:** 0.0310930829000574 **34:** 0.140505890675594 **35:** 0.125433113853058 **36:** 0.19563415272682 **37:** 3.56628482627457e-08 **38:** 0.131655606936058 **39:** 0.00978273979738719 **40:** 0.0667245471174334 **41:** 0.0978730323378211 **42:** 0.335722887299927 **43:** 0.636453286161674 **44:** 0.0439298391664398 **45:** 0.0276321435851717 **46:** 0.396222310506517 **47:** 0.00921906615638505 **48:** 0.0115285828457731 **49:** 0.0468856402567384 **50:** 0.650136705253986 **51:** 0.32135520346486 **52:** 0.301746593385296 **53:** 0.108414828542197 **54:** 0.221090125424419 **55:** 0.137315303374562 **56:** 0.246155598854879 **57:** 0.114842448299077 **58:** 0.218861811954387 **59:** 0.667197416968552 **60:** 0.0386315758718067 **61:** 0.156634277482666 **62:** 0.103216688324964 **63:** 0.201241421310928 **64:** 0.796260868816571 **65:** 0.584633093764016 **66:** 0.154322064995787 **67:** 0.778238184808198 **68:** 0.34445341235501 **69:** 0.066844947964976 **70:** 0.599148333114331 **71:** 0.789873815547454 **72:** 0.0436095183101236 **73:** 0.106068915924513 **74:** 0.043618639083514 **75:** 0.620561057816034 **76:** 0.087024915653913 **77:** 0.253007114652182 **78:** 0.0195171994745678 **79:** 0.808426267515294 **80:** 0.0428923839089808 **81:** 0.880295459846557 **82:** 0.522842213189857 **83:** 0.778464447289437 **84:** 0.0176624487053829 **85:** 0.0318048941115579 **86:** 0.610702472217885 **87:** 0.281779161759604 **88:** 0.38692973564985 **89:** 2.42151313159352e-08 **90:** 0.405035746768609 **91:** 0.118217275942332 **92:** 0.409435718530245 **93:** 0.709331035525078 **94:** 0.287247456336581 **95:** 0.934307612498884 **96:** 0.0559218521048343 **97:** 0.191613683541948 **98:** 0.669922987407228 **99:** 0.392961166684749 **100:** 0.00282666790172517 **101:** 0.109404789629453 **102:** 0.618599109657203 **103:** 0.446871902186724 **104:** 0.0707724129086167 **105:** 0.0183789261748995 **106:** 0.643174516355674 **107:** 0.181804037997709 **108:** 0.0729383457031842 **109:** 1.21453469821237e-06 **110:** 0.122077257808149 **111:** 0.686263012274433 **112:** 0.0442669712866269 **113:** 0.375005694354245 **114:** 0.0191985735193171 **115:** 0.178596552485721 **116:** 0.188280123128712 **117:** 0.02788954218689 **118:** 0.197575278727132 **119:** 0.11234038267415 **120:** 0.111674323769866 **121:** 0.367118512232061 **122:** 1.66974926695743e-07 **123:** 0.31763451506631 **124:** 0.0408299217105441 **125:** 0.550614664004172 **126:** 0.278052794537795 **127:** 0.84552309937729 **128:** 0.0630436554494535 **129:** 0.0322453377585592 **130:** 0.580694417702317 **131:** 0.0798588120019712 **132:** 0.389534246381164 **133:** 0.224284066256021 **134:** 0.329480006913146 **135:** 0.834292059293327 **136:** 0.0132073429133512 **137:** 0.133303239977064 **138:** 0.116512474487956 **139:** 0.0592810878343377 **140:** 0.0172950526983148 **141:** 0.825576929746072 **142:** 0.0612471375270428 **143:** 0.0125997299995142 **144:** 0.0349712580735897 **145:** 0.385702671921424 **146:** 0.108793855833266 **147:** 0.643508678682285 **148:** 0.669965108133365 **149:** 0.0510372894252668 **150:** 0.417422653494713 **151:** 0.232012195517213 **152:** 0.128201506086098 **153:** 0.110403158675029 **154:** 0.0138583973999629 **155:** 0.889806443431729 **156:** 0.726217834963811 **157:** 0.191776654714851 **158:** 0.430739941590273 **159:** 0.0574916377930104 **160:** 0.0457123319951057 **161:** 0.0662402537532521 **162:** 0.536006359995032 **163:** 0.176037734852602 **164:** 2.0028841999341e-08 **165:** 0.0702017271638281 **166:** 0.108447594842391 **167:** 0.405651561959908 **168:** 0.871322504071041 **169:** 0.178953619061188 **170:** 0.614649505248505 **171:** 0.59570937678776 **172:** 0.0852753903784437 **173:** 0.592084950733945 **174:** 0.156525388705636 **175:** 0.273027284784856 **176:** 0.027539564900633 **177:** 0.110103353196841 **178:** 0.622341218478283 **179:** 0.513253173732628 **180:** 0.626339879204505 **181:** 0.29581640376903 **182:** 0.729105852018251 **183:** 0.0144896533813076 **184:** 0.154560756084853 **185:** 0.110566206133116 **186:** 2.57594604097518e-08 **187:** 0.534267671234725 **188:** 0.483199110754618 **189:** 0.0466475357679148 **190:** 0.0389478737932743 **191:** 0.14933745942891 **192:** 0.316779705620714 **193:** 0.16568843837771 **194:** 0.498133227971185 **195:** 0.0426823650898274 **196:** 0.229531003826723 **197:** 0.245721121516952 **198:** 0.655057841095266 **199:** 0.128900645225803 **200:** 0.0300124510382124 **201:** 0.841990242914613 **202:** 0.14883663740095 **203:** 0.938854107459873 **204:** 0.583517344310879 **205:** 0.254981021014969 **206:** 0.384210252814526 **207:** 0.793476886068114 **208:** 0.943729718153004 **209:** 0.0284183419884234 **210:** 0.862425341201509 **211:** 0.666878833231457 **212:** 0.555568156720251 **213:** 0.0233060195363274 **214:** 0.593041040381449 **215:** 0.0415290992084238 **216:** 0.0157223114851342 **217:** 0.0348384983410256 **218:** 0.191204548440923 **219:** 0.0288211195665811 **220:** 0.0611239375931927 **221:** 0.307559972723811 **222:** 0.581343605577129 **223:** 0.0151969496625188 **224:** 0.73952393661164 **225:** 0.0742421122610703 **226:** 0.0106306705929144 **227:** 0.299891375372186 **228:** 0.188544404103885 **229:** 0.171755646261708 **230:** 0.0487958855160275 **231:** 0.415714342219413 **232:** 0.0686264726383295 **233:** 0.0258868642376737 **234:** 0.0275146912724957 **235:** 0.185571009948282 **236:** 0.19630399099828 **237:** 0.555082466556529 **238:** 0.771754017777665 **239:** 0.223447653597471 **240:** 0.105284875786713 **241:** 0.0958811875532551 **242:** 0.208000926492694 **243:** 0.378501054141287 **244:** 0.0641735766630168 **245:** 0.350371527350234 **246:** 0.181523706349016 **247:** 0.0296996457305181 **248:** 0.820814288097475 **249:** 0.00863424948836101 **250:** 0.021934485089544 **251:** 0.0820518650519645 **252:** 0.175657992793455 **253:** 0.543453062647147 **254:** 0.244936962876417 **255:** 0.0924763856458179 **256:** 0.433036954640309 **257:** 0.0664062624590226 **258:** 0.845509395502849 **259:** 0.64973131937257 **260:** 0.0232281840541003 **261:** 0.152343594666689 **262:** 0.563376406717324 **263:** 0.108034246636096 **264:** 0.0237674836101379 **265:** 0.104747616222874 **266:** 0.073655623423472 **267:** 0.179570502545221 **268:** 0.341668702049912 **269:** 0.0127551708326717 **270:** 0.66230838403741 **271:** 1.96257058218435e-07 **272:** 0.40119055175856 **273:** 0.33351714398911 **274:** 0.916826228122979 **275:** 0.550814828577136 **276:** 0.371717046720488 **277:**

0.699664968761511 **278:** 0.295684649281502 **279:** 0.911446029136818 **280:** 0.00946928513824437 **281:** 0.0291280023347744 **282:** 0.972726841424771 **283:** 0.152136882923718 **284:** 0.0894148622391138 **285:** 0.360088318305072 **286:** 0.0667033057333654 **287:** 0.105749652572269 **288:** 0.077638852856699 **289:** 0.635538801585363 **290:** 0.220126925094037 **291:** 0.304436517722608 **292:** 0.940544336504893 **293:** 0.0437552645266986 **294:** 0.0701479588071494 **295:** 0.35799603184621 **296:** 0.207559346340878 **297:** 0.0237761537175112 **298:** 0.438722573485778 **299:** 0.0749303640452978 **300:** 0.122643356102131

In [244]:
```
#confusion matrix of predicted vs observed on the validation data set
# This will help to calculate the accuracy and specificity

y_hat_round <- as.integer(y_hat > 0.5)

t <- table(y_hat_round,d.valid$X1.1)
t
#CONFUSION MATRIX
```

```
y_hat_round   0   1
          0 170  43
          1  34  53
```

In [246]:
```
# Model's accuracy is (170 + 43) / (170 + 53 + 43 + 34) = 74%.
Accuracy <- (t[1,1] + t[2,2]) / sum(t)
Accuracy
```

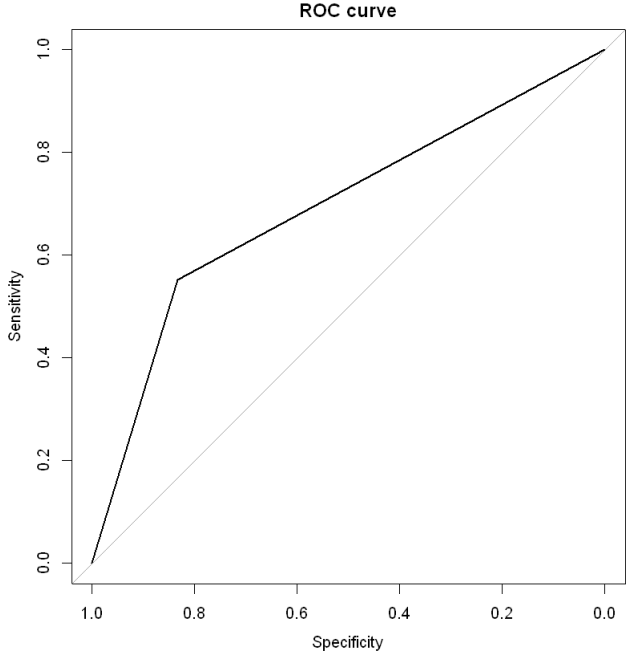
0.743333333333333

In [247]:
```
specificity <- (t[1,1])/(t[1,1]+t[2,1])
specificity
```

0.833333333333333

In [248]:
```
library(pROC)

# Develop ROC curve to determine the quality of fit

r<-roc(d.valid$X1.1,y_hat_round)

# Plot the ROC curve

plot(r,main="ROC curve")
r
```

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

Setting direction: controls < cases


Call:
roc.default(response = d.valid$X1.1, predictor = y_hat_round)

Data: y_hat_round in 204 controls (d.valid$X1.1 0) < 96 cases (d.valid$X1.1 1).
Area under the curve: 0.6927
```



In [251]:
```
# The area under the curve is 69%. This means that whenever a sample is chosen
# from the response group and another sample is chosen from the non-response
# group, then the model will correctly classify both the samples 68% of the times.


acc <- c()
auc <- c()

for (i in 1:9) {
  y_hat_round <- as.integer(y_hat > i/10)
  t <- table(y_hat_round,d.valid$X1.1)
  acc <- cbind(acc,(t[1,1] + t[2,2]) / sum(t))
  r<-roc(d.valid$X1.1,y_hat_round)
  auc <- cbind(auc,r$auc)
}

acc
auc
```

```
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
```

A matrix: 1 × 9 of type dbl

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0.57 | 0.6833333 | 0.7033333 | 0.7233333 | 0.7433333 | 0.7366667 | 0.73 | 0.72 | 0.6933333 |

A matrix: 1 × 9 of type dbl

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0.6700368 | 0.7175245 | 0.7074142 | 0.7000613 | 0.6927083 | 0.6492034 | 0.6112132 | 0.5707721 | 0.5235907 |

## ---------- Part 2 ----------

## The loss of incorrectly classfying a "bad" customer is 5 times the loss of incorrectly classifying a "good" customer.

## Calulating loss for the value of thresholds ranging from 0.01 to 1.

# Approach for Part#2

1) for the confusion matrix,Associated cost to False Positives and True negatives (as TP and FN are Zero cost) 2) per requirement, use True negatives are 5 times more costlier than False positives. For values 1-100, calculate the loss 3) identify the threshold with lowest loss 4) For the minimum threshold point, obtain the Accuracy and AUC

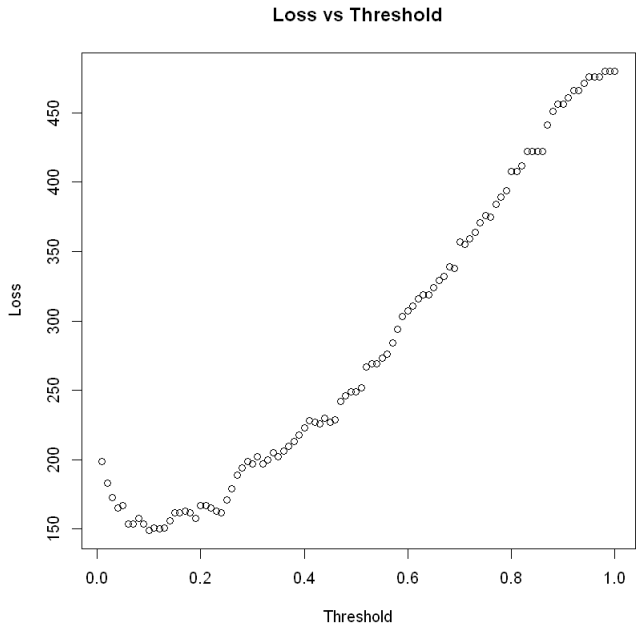
## Accuracy : 0.57

## Area under the curve: 0.67

In [296]:
```
loss <- c()
#Associated cost to False Positives and True negatives (as TP and FN are Zero cost)
for(i in 1:100)
{
  y_hat_round <- as.integer(y_hat > (i/100)) # calculate threshold predictions

  tm <-as.matrix(table(y_hat_round,d.valid$X1.1)) #confusion matrix

  if(nrow(tm)>1) { c1 <- tm[2,1] } else { c1 <- 0 }
  if(ncol(tm)>1) { c2 <- tm[1,2] } else { c2 <- 0 }
  loss <- c(loss, c2*5 + c1)
}
loss#loss for 100 values(1-100)
```

```
212 · 207 · 192 · 183 · 182 · 182 · 171 · 162 · 175 · 171 · 173 · 169 · 169 · 170 · 167 · 173 · 172 · 178 · 185 · 185 · 182 · 187 ·
200 · 205 · 208 · 206 · 206 · 216 · 220 · 222 · 218 · 216 · 220 · 223 · 227 · 237 · 241 · 250 · 270 · 268 · 282 · 280 · 280 · 288 ·
297 · 297 · 297 · 297 · 307 · 306 · 306 · 311 · 310 · 314 · 313 · 315 · 314 · 314 · 316 · 330 · 330 · 345 · 359 · 363 · 366 · 369 ·
370 · 370 · 375 · 380 · 385 · 385 · 395 · 405 · 405 · 405 · 405 · 414 · 419 · 428 · 438 · 438 · 448 · 453 · 468 · 468 · 473 · 478 ·
482 · 482 · 487 · 497 · 497 · 506 · 516 · 516 · 516 · 515 · 515 · 515
```

In [254]:
```
plot(c(1:100)/100,loss,xlab = "Threshold",ylab = "Loss",main = "Loss vs Threshold")
```




In [255]:
```
which.min(loss) # find the mininmuj loss
```

10

In [256]:
```
loss
```

199 · 183 · 173 · 165 · 167 · 154 · 154 · 158 · 154 · 149 · 151 · 150 · 151 · 156 · 162 · 162 · 163 · 162 · 158 · 167 · 167 · 165 ·
163 · 162 · 171 · 179 · 189 · 194 · 199 · 197 · 202 · 197 · 200 · 205 · 202 · 206 · 210 · 213 · 218 · 223 · 228 · 227 · 226 · 230 ·
227 · 229 · 242 · 246 · 249 · 249 · 252 · 267 · 269 · 269 · 273 · 276 · 284 · 294 · 303 · 307 · 311 · 316 · 319 · 319 · 324 · 329 ·
332 · 339 · 338 · 357 · 355 · 359 · 364 · 371 · 376 · 375 · 384 · 389 · 394 · 408 · 408 · 412 · 422 · 422 · 422 · 422 · 441 · 451 ·
456 · 456 · 461 · 466 · 466 · 471 · 476 · 476 · 476 · 480 · 480 · 480

In [259]:
```
# The threshold probability corresponding to minimum expected loss is 0.14.
# The range from 0.07-0.14 is all pretty good.


#Here's the accuracy and area-under-curve for the 0.14 threshold:

y_hat_round <- as.integer(y_hat > (which.min(loss)/100)) # find 0/1 predictions
t <- table(y_hat_round,d.valid$X1.1) # put in table form
acc <- (t[1,1] + t[2,2]) / sum(t) # calculate accuracy
r<-roc(d.valid$X1.1,y_hat_round) # calculate ROC curve
auc <- r$auc # get AUC

acc
auc

## [1] 0.57
## Area under the curve: 0.67
```

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

Setting direction: controls < cases
```

0.57

0.670036764705882

In [ ]:
```
########End of Week 7 Advanced Regression assignment###########
```