Assignment4

Prashant Kubsad

6/4/2020

Question 9.1

As mentioned in the homework, lets use prcomp() to get the principal components for the given data. The summary of the response gives the proportion of variance explained by each principal component. The first 3 components explain for 72% of the variance in the model.

```
#cleaning environment and starting fresh.
rm(list = ls())
#setting seed for consistent results
set.seed(1)
#Read the text file into data.
data<-read.table("uscrime.txt", header=TRUE)
mu = colMeans(data[,-16])
pca<-prcomp(data[,-16],center = TRUE, scale. = TRUE)
cat("##### Summary of pca #####\n")
```

```
## ##### Summary of pca #####
```

```
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377 0.56729
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688 0.02145
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996 0.92142
##                            PC8     PC9    PC10    PC11    PC12    PC13   PC14
## Standard deviation     0.55444 0.48493 0.44708 0.41915 0.35804 0.26333 0.2418
## Proportion of Variance 0.02049 0.01568 0.01333 0.01171 0.00855 0.00462 0.0039
## Cumulative Proportion  0.94191 0.95759 0.97091 0.98263 0.99117 0.99579 0.9997
##                           PC15
## Standard deviation     0.06793
## Proportion of Variance 0.00031
## Cumulative Proportion  1.00000
```

To decide how many principal components to select, we have 2 options. One is a kaiser method which says any pca which has eigenvalues greater than or equal to 1 can be considered. To get the eigen values we just have to square the standard deviations that are given in the summary of prcomp. We see the first 4 components have eigen values greater than 1.

```
eigenValues<-pca$sdev^2
cat("#### eigen values for each component #####\n")
```

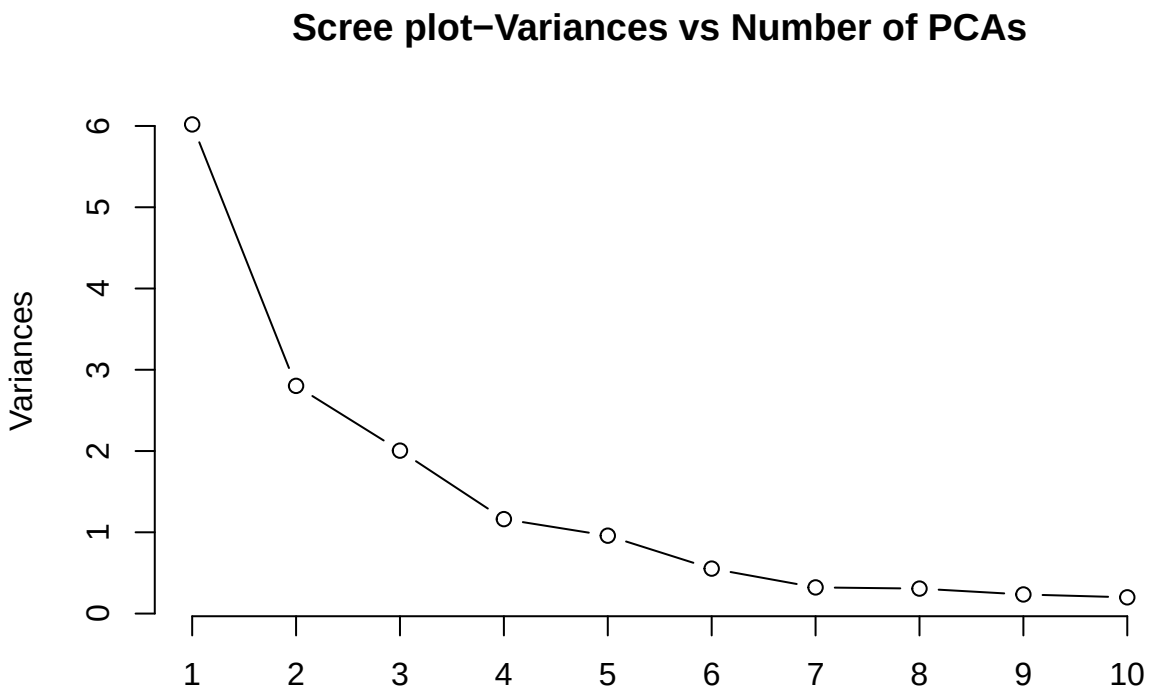
```
## #### eigen values for each component #####
```

```
eigenValues
```

```
##  [1] 6.018952657 2.801847026 2.004944334 1.162207801 0.958298972 0.553193900
##  [7] 0.321818687 0.307401270 0.235155292 0.199880931 0.175685403 0.128190107
## [13] 0.069341691 0.058467765 0.004614165
```

Second option to decide the number of principal components, we can use the scree plot which is elbow plot of variances (number of standard deviations) v/s the number of principal components. The diminishing returns is spotted around pcs = 5 or 6, meaning at n=5 the variance is within 1 std deviation . Combining both the approaches, lets go with n=5 as our number of principal components.

```
plot(pca,type="line",main="Scree plot-Variances vs Number of PCAs")
```



As explained in the lecture video, these coefficients help us to rotate the initial data in the new dimension. Inorder to reconstuct the original values with the selected principal components, we can use the equation PCA reconstruction=PC scores X (Eigenvectors+Mean). In the prcomps response, x denotes the scores and rotation denotes the eigen vectors.As mentioned in the homework, we will have to unscale the data back to the correct scale. We can run the linear regression on the reconstructed data and we see the results as *Multiple R-squared: 0.6452, Adjusted R-squared: 0.6019.* Compare this to the model we did in previous homework, we had got Multiple R-squared: 0.5031, Adjusted R-squared: 0.4424 this shows better Rsquared value. But to truly decide which model is better we have to run AIC/BIC analysis.

source for understanding recostruction of pca to original data: Credit/Source: https://stats.stackexchange.com/questions/229092/how-to-reverse-pca-and-reconstruct-original-variables-from-several-principal-com

```
cat("#### Reconstructing original values with PCAs. ####\n")
```

```
## #### Reconstructing original values with PCAs. ####
```

```
#number of pca's selected above.
nComp=5;
#Reconstruction of data with 5 pcas
```

```
Xhat = pca$x[,1:nComp] %*% t(pca$rotation[,1:nComp])

#reconstructed data
cat("##### reconstructed data ####\n")
```

```
## ##### reconstructed data ####
```

```
head(Xhat)
```

```
##                M         So         Ed        Po1        Po2         LF
## [1,]  0.98024142  1.7351119 -1.6845015 -0.8467191 -0.8514997 -1.5111137
## [2,]  0.09941375 -0.3789699  0.5620040  0.4247953  0.3940054  0.8829083
## [3,]  1.01933956  1.3766507 -1.3663099 -1.3830434 -1.3685722 -1.0151416
## [4,] -0.70569187 -0.5469824  0.8355325  2.4049190  2.3563403  0.5077229
## [5,] -0.28405218 -0.5944835  1.0531894  0.5028510  0.5327591  0.9771664
## [6,] -1.32917463 -0.7098543  1.0163442  1.0971249  1.1681262 -0.1652900
##             M.F         Pop          NW          U1          U2     Wealth
## [1,] -0.7176707 -0.07613024  1.53484229  0.42853552  0.82349952 -1.4415206
## [2,]  1.0922516 -0.08750438 -0.09947484  0.25691071  0.02635084  0.3689406
## [3,] -0.6105698 -0.68701523  1.02810645 -0.06245636 -0.03750334 -1.5590266
## [4,]  0.3397716  1.77973685  0.16459656  0.13042335  0.85410208  1.6832239
## [5,]  0.4625738 -0.34191588 -0.58586435 -0.89427756 -1.13385459  0.6936179
## [6,] -0.6829028  0.37070959 -0.93978540 -0.63991670 -0.43280788  1.2947149
##            Ineq        Prob        Time
## [1,]  1.5003515  1.38601786 -0.11906426
## [2,] -0.1865734 -0.42571934 -0.07491802
## [3,]  1.3714193  1.44400452 -0.38631637
## [4,] -1.0063667 -1.51138103  0.98745679
## [5,] -0.7767639 -0.05497804 -0.65237295
## [6,] -1.4509081 -0.03785214 -0.67838409
```

```
Xhat = scale(Xhat, center = -mu, scale = FALSE)
cat("### unscaled reconstructed data ####\n")
```

```
## ### unscaled reconstructed data ####
```

```
head(Xhat,5)
```

```
##             M          So        Ed       Po1       Po2         LF      M.F
## [1,] 14.83769  2.07553748  8.879328  7.653281  7.171905 -0.9499222 97.58446
## [2,] 13.95686 -0.03854435 11.125834  8.924795  8.417410  1.4440998 99.39438
## [3,] 14.87679  1.71707621  9.197520  7.116957  6.654832 -0.4539501 97.69156
## [4,] 13.15175 -0.20655682 11.399362 10.904919 10.379745  1.0689144 98.64190
## [5,] 13.57339 -0.25405796 11.617019  9.002851  8.556163  1.5383579 98.76470
##           Pop        NW          U1       U2   Wealth     Ineq         Prob
## [1,] 36.54089 11.647608  0.52400360 4.221372 5252.388 20.90035  1.433109241
## [2,] 36.52952 10.013291  0.35237879 3.424223 5254.199 19.21343 -0.378627953
## [3,] 35.93001 11.140872  0.03301173 3.360369 5252.271 20.77142  1.491095903
## [4,] 38.39676 10.277363  0.22589143 4.251974 5255.513 18.39363 -1.464289644
## [5,] 36.27511  9.526902 -0.79880948 2.264018 5254.523 18.62324 -0.007886658
##          Time
## [1,] 26.47886
## [2,] 26.52300
## [3,] 26.21160
## [4,] 27.58538
## [5,] 25.94555
```

```r
#adding the result column to the reconstructed data
xhatData<-cbind(Xhat,data[,16])

#running regression model on the reconstructed data
linearRegModelXhat<-lm(formula =V16~.,data = data.frame(xhatData))
cat("### summary of regression model on reconstructed data ####\n")
```

```
## ### summary of regression model on reconstructed data ####
```

```r
summary(linearRegModelXhat)
```

```
##
## Call:
## lm(formula = V16 ~ ., data = data.frame(xhatData))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -420.79 -185.01   12.21  146.24  447.86
##
## Coefficients: (10 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6043.55    1404.41  -4.303 0.000102 ***
## M              34.41     162.52   0.212 0.833379
## So            186.93     214.57   0.871 0.388718
## Ed            219.00     175.77   1.246 0.219874
## Po1          4587.82    2025.05   2.266 0.028828 *
## Po2         -4349.99    2108.53  -2.063 0.045480 *
## LF                NA         NA      NA       NA
## M.F               NA         NA      NA       NA
## Pop               NA         NA      NA       NA
## NW                NA         NA      NA       NA
## U1                NA         NA      NA       NA
## U2                NA         NA      NA       NA
## Wealth            NA         NA      NA       NA
## Ineq              NA         NA      NA       NA
## Prob              NA         NA      NA       NA
## Time              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 244 on 41 degrees of freedom
## Multiple R-squared:  0.6452, Adjusted R-squared:  0.6019
## F-statistic: 14.91 on 5 and 41 DF,  p-value: 2.446e-08
```

Rebuilding new pca analysis using 5 as the new rank. I have plotted the biplot for this model. We can see the corelation among the predictors in the data. For example we see PO1 and PO2 are overlayed on top of each other. This suggests that the values of po1 and po2 are almost identical and are related in the same direction. If po1 increases, po2 also increases. In the biplot, row 19 almost falls ont he po1/po2 line. If we inspect the 19th row in our data, the po1 and po2 values are exactly the same.

Similarly we see lines M and Wealth are in opposite direction. This means that these 2 fields are inversly co related. When one increases the other decreases.Lets take row 6 of the data which on the wealth line, it shows higher wealth value but lower M value. We can confirm the same from the data where we see for 6th row, M is 12.1 (second lowest value) and Wealth is 6890 (highest value).

```
#recreating the new pca with 5 pca
pcaFinal<-prcomp(data[,-16],center = TRUE, scale. = TRUE, rank. = 5)
summary(pcaFinal)
```

```
## Importance of first k=5 (out of 15) components:
##                            PC1    PC2    PC3     PC4     PC5
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308
```

```
#biplot(pcaFinal)
```

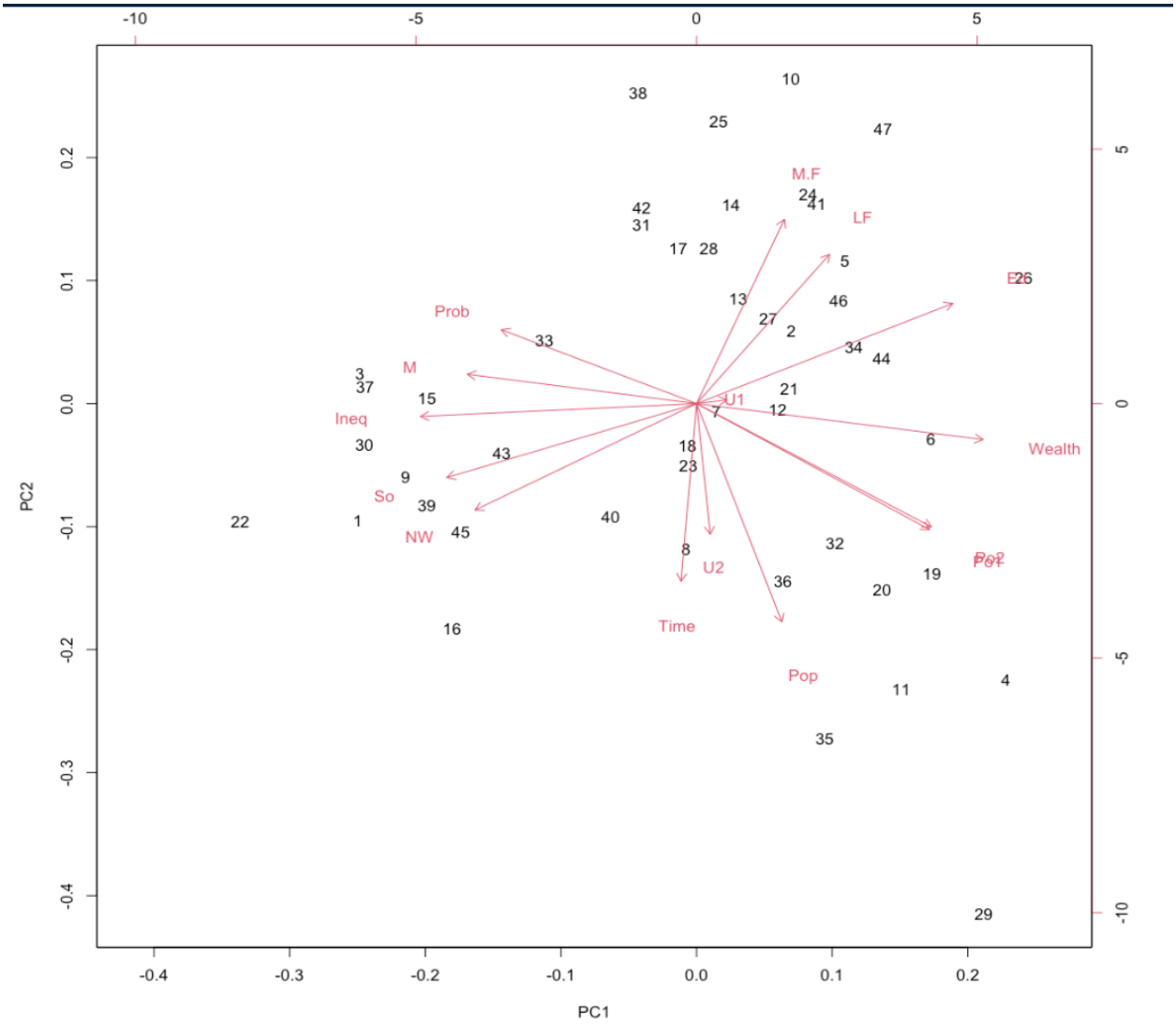


Figure 1: biplot.png

We can also visualise the co realtion among all the factors using cpairs chart. Darker color pink indicates strong co-relation between the factors while lighter colors yellow and green show weak or no co relation between the factors. Ex. Cells intersecting P01 and P02 circled in blue show co relation. Similary intersecting

cells for M and wealth (circled in red)

```
library(gclus)
```

```
## Loading required package: cluster
```

```
my.abs<-abs(cor(data))
my.colors<-dmat.color(my.abs)
my.ord<-order.single(cor(data))
#cpairs(data,my.ord,panel.colors = my.colors)
```

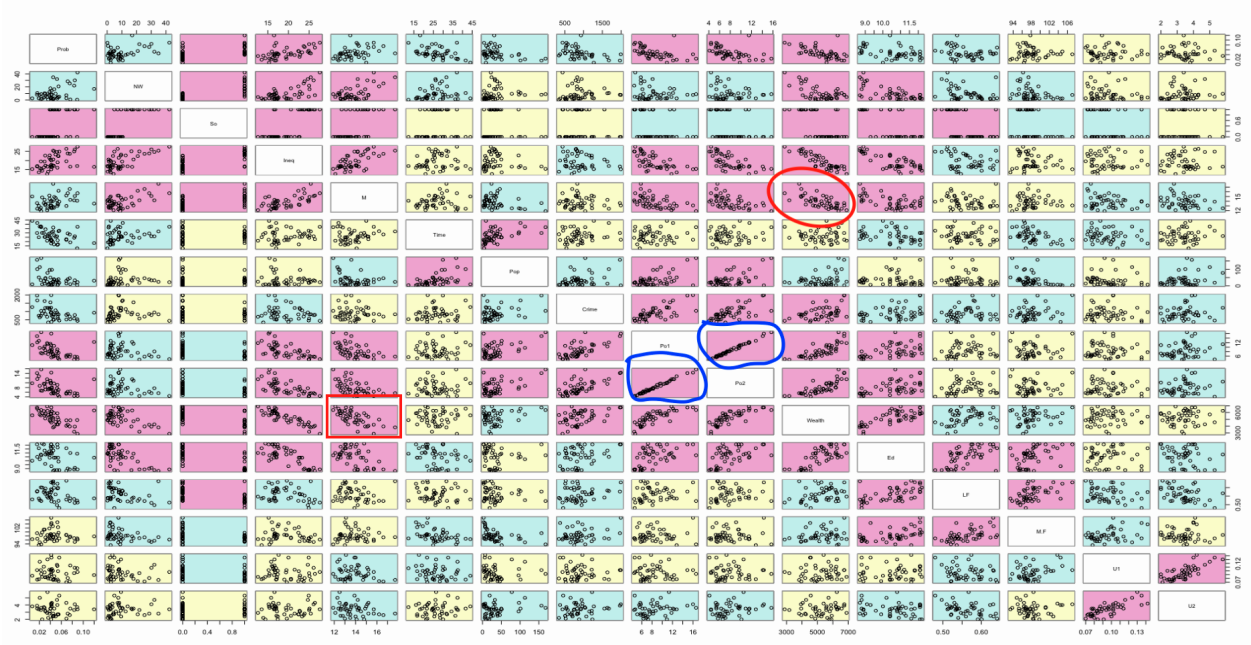


Figure 2: cpairs.png

Lets use the predict funciton to predict the value for the new city. I am building the linear regression model on the scores of the first 5 pca against the crimes column. So we have to translate the new city data also using the pcas and then predict the data.

```
#new city data
newCity <- 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)
#translate the new city data using the pcas
translatedNewCity <- data.frame(predict(pcaFinal, newCity))

#adding the crimes column against each vector
combinedData <- cbind(pcaFinal$x[,1:5],data[,16])

#running regression model
linearRegModel<-lm(formula = V6~.,data = data.frame(combinedData))
linearRegModel
```

```
##
## Call:
```

```
## lm(formula = V6 ~ ., data = data.frame(combinedData))
##
## Coefficients:
## (Intercept)          PC1          PC2          PC3          PC4          PC5
##      905.09        65.22       -70.08        25.19        69.45      -229.04
```

```
prediction<-predict(linearRegModel,translatedNewCity)
cat("#### predicted value: ####\n")
```

```
## #### predicted value: ####
```

```
cat(prediction)
```

```
## 1388.926
```

I have followed the approach given by TA in piazza to get coefficients interms of original data. The result what we see here, R-Squared = 0.6451941 and Adjusted R-Squared = 0.601925 is exactly same as in page 4(linear regression with xhat values).

Credit/Thanks: https://piazza.com/class/ka28g4qhcw67bo?cid=493 https://d1b10bmlvqabco.cloudfront.net/paste/jc6a65pgbyVx/ec3b8702a968d3404740f62577a73a1e0b23f675addff6d74515b414d3e05bf9/PCA_2.pdf

```
#get the intercept of from the linear regression model
intercept<-linearRegModel$coefficients[1]

#get the coefficients of each pca
slopes<-linearRegModel$coefficients[2:6]
intercept
```

```
## (Intercept)
##    905.0851
```

```
slopes
```

```
##        PC1        PC2        PC3        PC4        PC5
##   65.21593  -70.08312   25.19408   69.44603 -229.04282
```

```
#rotate the slopes using the eigen vector got in the pca mdoel
#these values are still scaled
coeffScaledOriginal <- pca$rotation[,1:5] %*% slopes
t(coeffScaledOriginal)
```

```
##             M       So       Ed      Po1      Po2      LF      M.F      Pop
## [1,] 60.79435 37.84824 19.94776 117.3449 111.4508 76.2549 108.1266 58.88024
##            NW       U1       U2   Wealth    Ineq      Prob     Time
## [1,] 98.07179 2.866783 32.34551 35.93336 22.1037 -34.64026 27.20502
```

```
#calculate mu and sigma for the data set.
sigma=sapply(data[,1:15],sd)
mu=sapply(data[,1:15],mean)

#unscaling the coefficients
coeffUnscaledOriginal<-coeffScaledOriginal/sigma
t(coeffUnscaledOriginal)
```

```
##             M       So      Ed      Po1      Po2       LF      M.F      Pop
## [1,] 48.37374 79.01922 17.8312 39.48484 39.85892 1886.946 36.69366 1.546583
##            NW       U1       U2     Wealth     Ineq      Prob     Time
## [1,] 9.537384 159.0115 38.29933 0.03724014 5.540321 -1523.521 3.838779
```

```
#unscaling the intercept to original parameters.
interceptUnscaledOriginal<-intercept-sum(coeffScaledOriginal*mu/sigma)
t(interceptUnscaledOriginal)
```

```
##      (Intercept)
## [1,]   -5933.837
```

```
#estimating using the equation in lecture Y=aX+b.
#%*% does matrix multiplication
estimates<-as.matrix(data[,1:15])  %*% coeffUnscaledOriginal + interceptUnscaledOriginal

#calculate sum of squared errors
residualSumSquare = sum((estimates - data[,16])^2)
totalSumOfSquares = sum((data[,16] - mean(data[,16]))^2)
R2 <- (1 - residualSumSquare/totalSumOfSquares)
R2
```

```
## [1] 0.6451941
```

```
adjustedR2 <- R2 - (1-R2)*5/(nrow(data)-5-1)
adjustedR2
```

```
## [1] 0.601925
```

Question 10.1

I am using rpart function in R to create tree for the given data. As per the lecture, any leaf with less than 5% of the data is not a good split. So am passing in minbucket =3 which will satisfy this condition.Also, it was mentioned in the lecture that a good way to perform tree regression is to do a cross validation. rpart function has built in support for cv with xval parameter. If we see the result of the model we see the variables used in the tree construction and cost parameter for each split. This cp value is the amount by which splitting that node improved the relative error. Lets take step 4 and 5 in the below output. We see when we split from 4 nodes to 5 nodes, the relative error at the 5th node is 0.1935465. If we subtract this with the relative error of previous step divided by number of nodes, we get the cp for step 4. * (0.24894-0.19354650)/1 =0.05539395*. I am calculating the mean absolute error(MAE) and Rsquared of the tree model against our data to get how good the model is. Higher the R2 is better the model for the given data.

```
#cleaning environment and starting fresh.
rm(list = ls())
library(tree)
library(rpart)
library(rpart.plot)
library(randomForest)
```

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

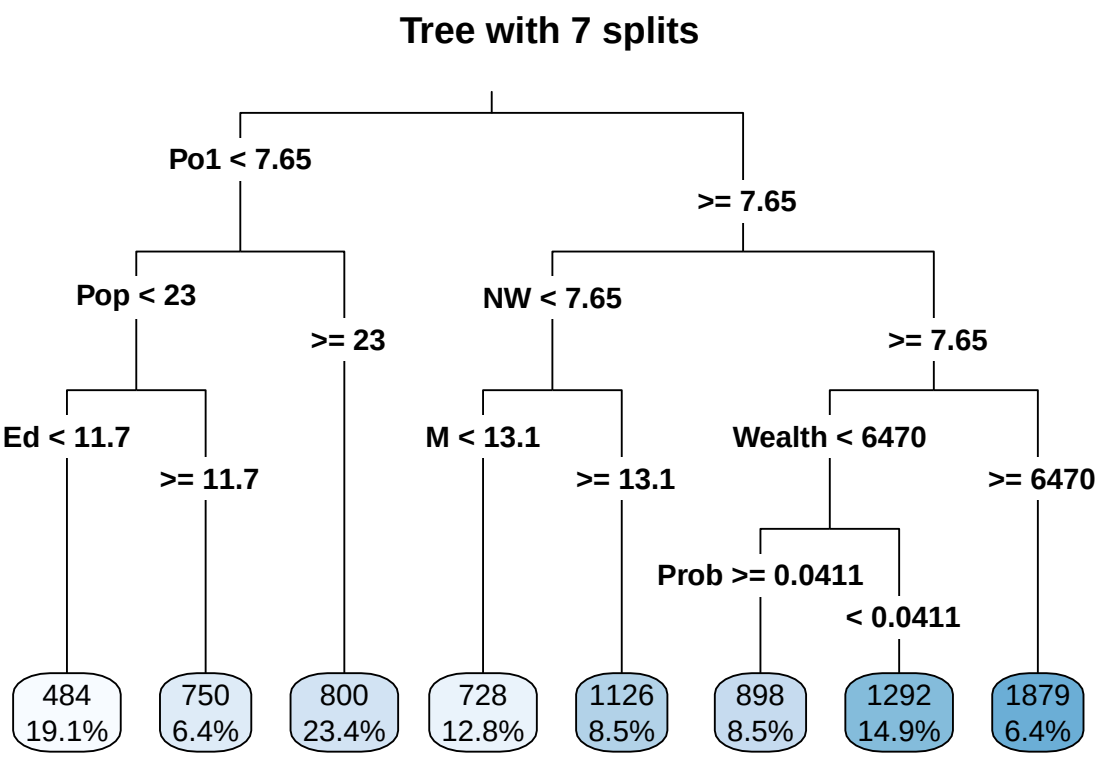
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
#setting seed for consistent results
set.seed(1)
#Read the text file into data.
data<-read.table("uscrime.txt", header=TRUE)
#creating the tree model withe given data.
rp<-rpart(formula = Crime ~ M+So+Ed+Po1+Po2+LF+M.F+Pop+NW+U1+U2+Wealth+Ineq+Prob+Time,
            data=data,
            control=rpart.control(xval=20,
                                  minbucket=3))
```

```
#printing the cp table for the model
rp$cptable
```

```
##           CP nsplit rel error    xerror      xstd
## 1 0.36296293      0 1.0000000 1.0456692 0.2603120
## 2 0.16540024      1 0.6370371 0.8421735 0.2072608
## 3 0.05730143      3 0.3062366 1.2560787 0.3169366
## 4 0.05538867      4 0.2489352 1.1853968 0.3119015
## 5 0.05173165      5 0.1935465 1.1905342 0.3116813
## 6 0.02305931      6 0.1418148 1.0910926 0.3082772
## 7 0.01000000      7 0.1187555 1.1319815 0.3258800
```

```
#plotting the tree
rpart.plot(rp,type=3,digits=3,fallen.leaves = TRUE, main="Tree with 7 splits")
```




```
#running prediction using this tree against all the data in the input
predictedValues<-predict(rp,data)
maeFunction<-function(actual,predicted)  {mean(abs(actual-predicted))}
mae<-maeFunction(data$Crime,predictedValues)

#Rsquaed calculatoins
residualSumSquare <- sum((predictedValues - data$Crime)^2)
totalSumOfSquares <- sum((data$Crime - mean(data$Crime))^2)
R2 <- (1 -(residualSumSquare/totalSumOfSquares))

cat("mean absolute error for tree with 7 splits is :",mae)
```

```
## mean absolute error for tree with 7 splits is : 104.9587
```

```
cat("\nRsquared  for tree with 7 splits is :",R2)
```

```
##
```

```
## Rsquared  for tree with 7 splits is : 0.8812445
```

As mentioned in the lecture, lets prune the table for few values of cp and see if the model's mean absolute error decreases further. I am trying it for few of the cps listed in the above cp table. We see that model with 7 splits gives the best mean absolute error and Rsquared value. Also, "Po1" seems to be a significant factor as all the trees are using this field as the first split as Po1 is less than 7.65 or not.

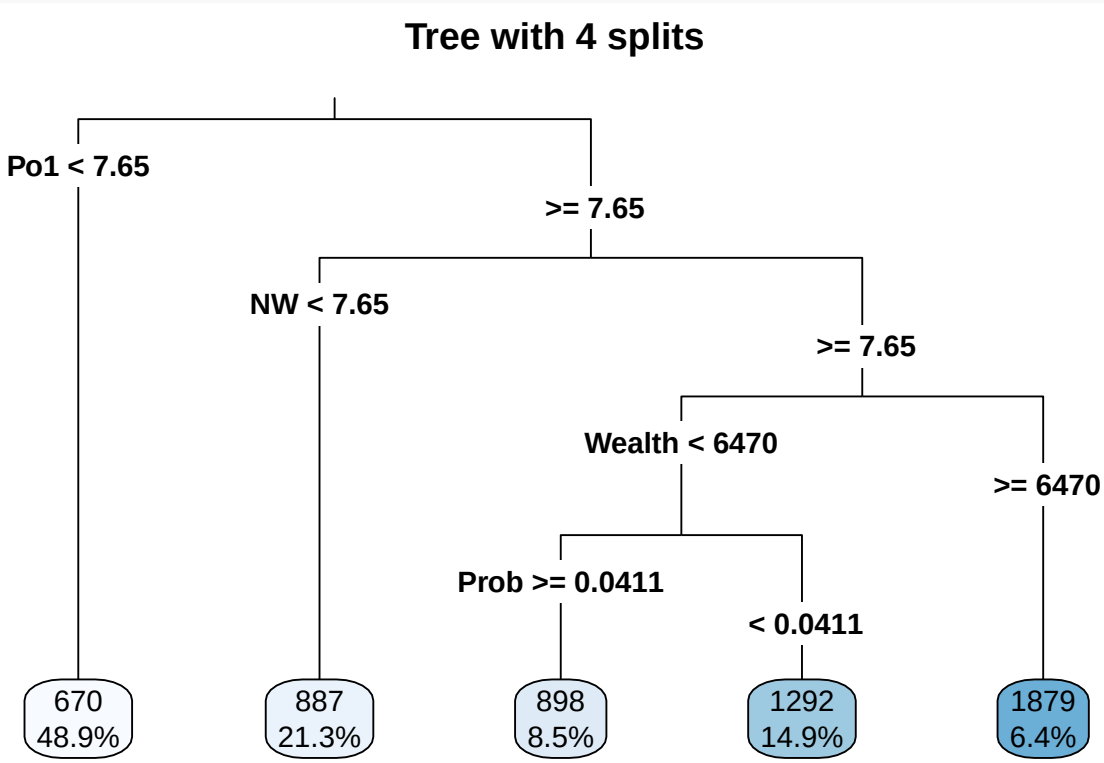
```
prunedModel1<-prune.rpart(rp,cp=0.05538867)
cat("***** CP table for pruned model for upto 4 splits *****\n")
```

```
## ***** CP table for pruned model for upto 4 splits *****
```

```
prunedModel1$cptable
```

```
##           CP nsplit rel error    xerror      xstd
## 1 0.36296293      0 1.0000000 1.0456692 0.2603120
## 2 0.16540024      1 0.6370371 0.8421735 0.2072608
## 3 0.05730143      3 0.3062366 1.2560787 0.3169366
## 4 0.05538867      4 0.2489352 1.1853968 0.3119015
```

```
newPredications1<-predict(prunedModel1,data)
newmae1<-maeFunction(data$Crime,newPredications1)
rpart.plot(prunedModel1,type=3,digits=3,fallen.leaves = TRUE,main="Tree with 4 splits")
```




```
residualSumSquare <- sum((newPredications1 - data$Crime)^2)
totalSumOfSquares <- sum((data$Crime - mean(data$Crime))^2)
R21 <- (1 -(residualSumSquare/totalSumOfSquares))



prunedModel2<-prune.rpart(rp,cp=0.16540024)
```

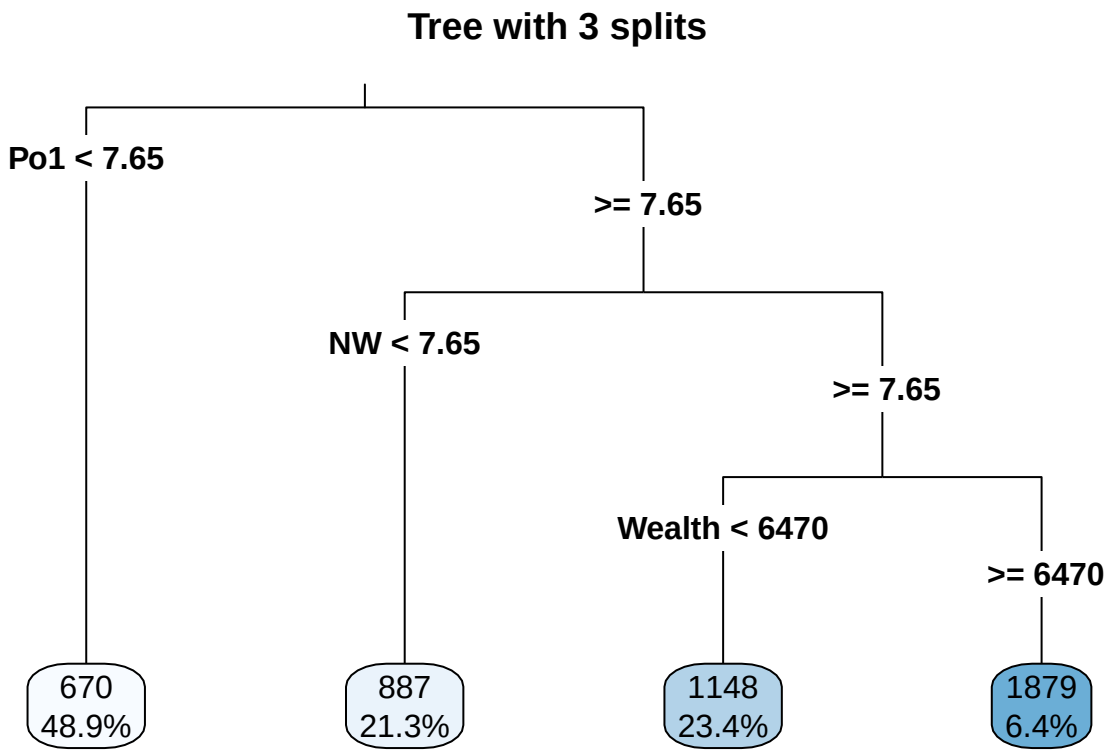
```
cat("***** CP table for pruned model for upto 3 splits *****\n")
```

```
## ***** CP table for pruned model for upto 3 splits *****
```

```
prunedModel2$cptable
```

```
##          CP nsplit rel error    xerror      xstd
## 1 0.3629629      0 1.0000000 1.0456692 0.2603120
## 2 0.1654002      1 0.6370371 0.8421735 0.2072608
## 3 0.1654002      3 0.3062366 1.2560787 0.3169366
```

```
newPredications2<-predict(prunedModel2,data)
newmae2<-maeFunction(data$Crime,newPredications2)
rpart.plot(prunedModel2,type=3,digits=3,fallen.leaves = TRUE,main="Tree with 3 splits")
```




```
residualSumSquare <- sum((newPredications2 - data$Crime)^2)
totalSumOfSquares <- sum((data$Crime - mean(data$Crime))^2)
R22 <- (1 -(residualSumSquare/totalSumOfSquares))


cat("\n Mean absolute error for split=7: ",mae)
```

```
##
##  Mean absolute error for split=7:  104.9587
```

```
cat("\n Mean absolute error for split=4: ",newmae1)
```

```
##
##  Mean absolute error for split=4:  160.9242
```

```
cat("\n Mean absolute error for split=3: ",newmae2)
```

```
##
```

```
##  Mean absolute error for split=3:  176.0909
```

```
cat("\nRsquared  for tree with 7 splits is :",R2)
```

```
##
## Rsquared  for tree with 7 splits is : 0.8812445
```

```
cat("\nRsquared  for tree with 4 splits is :",R21)
```

```
##
## Rsquared  for tree with 4 splits is : 0.7510648
```

```
cat("\nRsquared  for tree with 3 splits is :",R22)
```

```
##
## Rsquared  for tree with 3 splits is : 0.6937634
```

```
#tree<-tree(formula = Crime ~ M+So+Ed+Po1+Po2+LF+M.F+Pop+NW+U1+U2+Wealth+Ineq+Prob+Time, data)
#plot(tree)
#text(tree)
```

Exploring Tree function from tree package: I have used the tree function from tree package to generate the tree and it generates similar results as above and we can prune and optimise the parameters as done above.

```
#cleaning environment and starting fresh.
rm(list = ls())
library(randomForest)
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

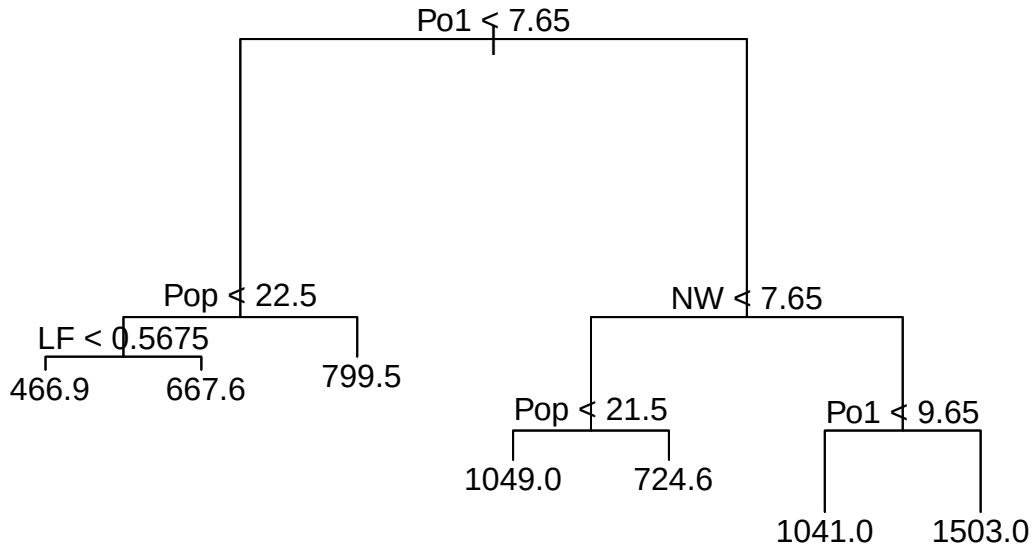
```
#setting seed for consistent results
set.seed(123)
#Read the text file into data.
data<-read.table("uscrime.txt", header=TRUE)
treeModel <- tree(Crime ~ ., data = data)
summary(treeModel)
```

```
##
## 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
```

```
cat("*****Tree created using tree() of Tree package******\n");
```

```
## *****Tree created using tree() of Tree package******
```

```
plot(treeModel, main="Tree created using tree() of Tree package")
text(treeModel)
```




```
treePredictions<-predict(treeModel,data)
residualSumSquare <- sum((treePredictions - data$Crime)^2)
totalSumOfSquares <- sum((data$Crime - mean(data$Crime))^2)
RTree <- (1 -(residualSumSquare/totalSumOfSquares))
cat("\n Rsquared from tree function: ",RTree)
```

```
##
##  Rsquared from tree function:  0.7244962
```

Random Forest

Lets run the random forest on our data without any parameters and take a look at the output. It uses default tree size of 500 trees with default mtry of 5. This means 500 trees were generated with various combinations of upto 5 variables split. When we run a predict, all the 500 trees are executed and averaged to get the predicted value. If we run the same model for prediction, we can use R-squared as factor to determine how our model is doing. As illustrated below, we can see the rsquared of the default model is around 0.900455. This value is much higher than the single tree. This observation aligns with what we learnt in the lecture.Since random forest generates many trees and then aggregates the results, the effect of randomness is spread across mutliple trees. We could have split the data into training and test sets but since the given data had only 47 rows, I opted to run it on the whole data.

```
#cleaning environment and starting fresh.
rm(list = ls())
library(randomForest)
library(ggplot2)
#setting seed for consistent results
set.seed(123)
#Read the text file into data.
data<-read.table("uscrime.txt", header=TRUE)
#creating random forest model with default values
rf <- randomForest(Crime~., data=data,importance=TRUE)
#running prediction on the input data using the model
predict <- predict(rf, newdata=data[,-16])
#calculating the R-squared
```

```
residualSumSquare <- sum((predict - data$Crime)^2)
totalSumOfSquares <- sum((data$Crime - mean(data$Crime))^2)
R2 <- (1 -(residualSumSquare/totalSumOfSquares))
cat("Rsqaured of the default model:", R2)
```

```
## Rsqaured of the default model: 0.900455
```

But we did not consider the parameters that can be passed to randomForest function. We can run a loop on multiple values of mtry, nodesize and ntree and determine which set of parameters give the best R-squared value. I have printed the R output with best R-squared value.

```
#creating list of K values from 1 to 50.
r2Frame <-data.frame(ntree=numeric(), nodeSize=numeric(), mtry=numeric(),R2=numeric())
mtryList = seq(1,10)
nodesizeList = seq(1,6)
ntreeList=c(300,400,500,600,700,800,900,1000,1200)
for(ntree in ntreeList){
  for(nodeSize in nodesizeList){
   for(mtry in mtryList){
     rfInLoop <- randomForest(Crime~., data=data,importance=TRUE,nodeSize=nodeSize,mtry=mtry,ntree=ntre
     predictInLoop <- predict(rfInLoop, newdata=data[,-16])
     residualSumSquare <- sum((predictInLoop - data$Crime)^2)
     totalSumOfSquares <- sum((data$Crime - mean(data$Crime))^2)
     R2 <- (1 -(residualSumSquare/totalSumOfSquares))
     r2Frame[nrow(r2Frame) + 1,] = c(ntree,nodeSize,mtry,R2)
   }
  }
}
result<-r2Frame[which.max(r2Frame$R2),]
cat("\n Best R2 is recorded for:")
```

```
##
##  Best R2 is recorded for:
```

```
result
```

```
##     ntree nodeSize mtry        R2
## 225   600        5    5 0.9053607
```

```
finalModel<-randomForest(Crime~., data=data,importance=TRUE,nodeSize=5,mtry=5,ntree=600)
finalModel
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = data, importance = TRUE,      nodeSize = 5, mtry = 5, ntre
##                Type of random forest: regression
##                      Number of trees: 600
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 85642.79
##                     % Var explained: 41.5
```

```
newCity <- 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)
finalPrediction<-predict(finalModel,newCity)
```

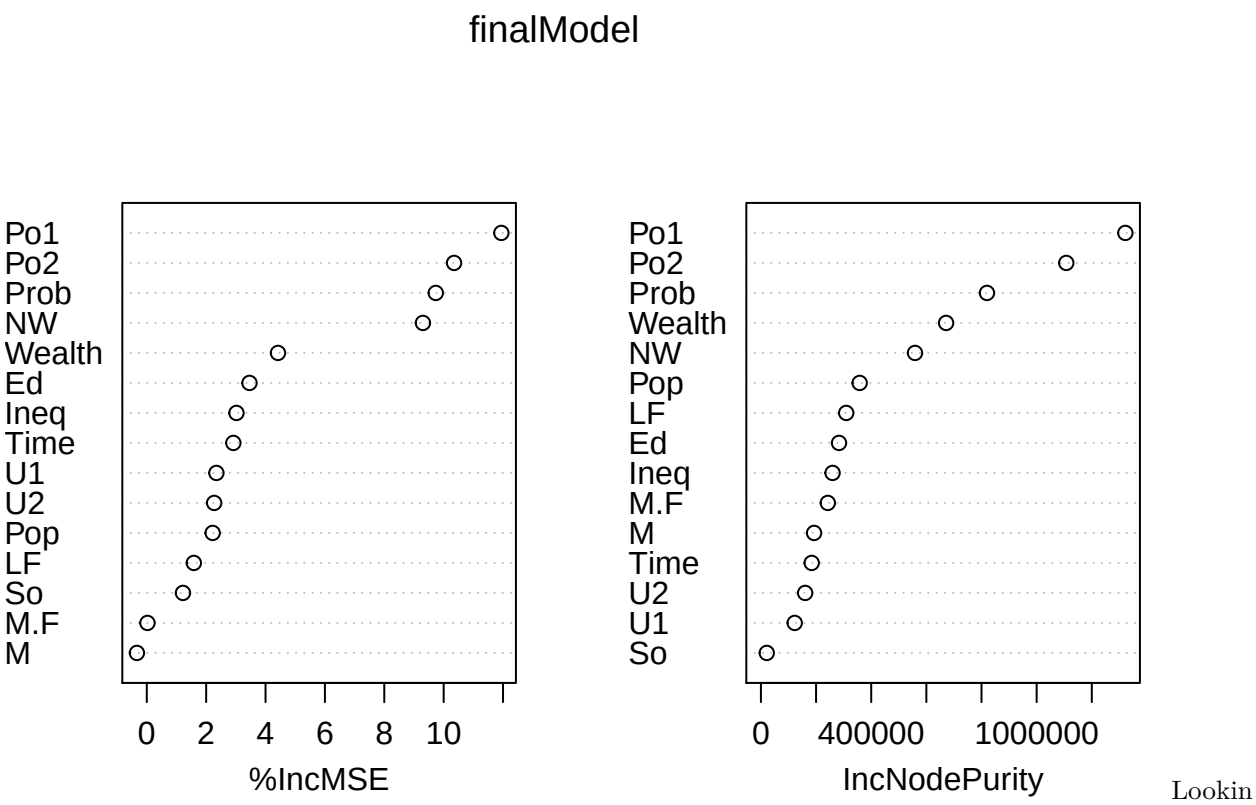
```
cat("prediction for the given input data in the previous problem: ",finalPrediction)
```

```
## prediction for the given input data in the previous problem:  1238.36
```

```
importance(finalModel)
```

```
##            %IncMSE IncNodePurity
## M       -0.33284216     192839.97
## So       1.21828979      21181.99
## Ed       3.45981149     283277.84
## Po1     11.94348233    1321686.36
## Po2     10.35138253    1107441.08
## LF       1.58474035     309412.58
## M.F      0.02194566     242596.89
## Pop      2.21875923     358160.58
## NW       9.30097596     558823.18
## U1       2.34496521     122527.51
## U2       2.26948479     160482.12
## Wealth   4.41904477     671673.91
## Ineq     3.02043365     259919.85
## Prob     9.73434877     819861.98
## Time     2.91573644     184213.51
```

```
varImpPlot(finalModel)
```



Lookin at the the %IncMSE plot we can see that Po1, Po2 and Prob are key factors for tree splits. These 3 factors account for nearly 35% of mean square error. On the similar lines if we look at Node Purity charity shows that more significant predictors achieve higher increases in node purities. Th number represents how much node impurity is reduced by splitting on that factor, incase of regression it is difference between RSS before and after the split

Question 10.2

I am working as software developer in one of the mortgage insurance companies. When we price insurance we measure the quality of the underlying mortgage loan and its probability of being repaid in with defaults. Logistic model can be used to predict the probability of a succesful repayment of the loan. This probability factor helps us determine the risk involved in insuring a loan. Few of the factors we consider for this regression are:

- Credit Score
- Number of borrowers on the loan application. Applications with 2 borrowers are more favourable compared to single borrower applications.
- DTI(Debt to Income ratio): Ratio of individual's monthly debt to his/her monthly gross income.Monthly debt includes payments like: credit card payments, auto loan payments, alimony, etc.,
- History of prior bankruptcy on the borrower. Factor of 0 or 1.
- LTV Ratio (Loan to Value): Ratio of selling price or appraised value to loan amount. (selling price or appraised value/loan amount applied for)
- Borrower Income: Annual income of all the borrowers combined.

Question 10.3.1

Explaination of logistic regression: The data in the attachment does not look factored and cannot be used as is.We have to convert the 21st column(response variable) in factors of 0 and 1. '0' is good credit and '1' is bad credit. I am using only one factor for the sake of understanding/explaining the logistic regression model and later run with all the factors. Lets run a logistic regression of V21 against V2(Credit type vs DurationInMonth). If we plot the CreditType vs Duration amount plot, we see the points are all on either 0 or 1. I generated sequence of Credit amounts to predict using the glm model. The predictions are nothing but the probabilities of a given record being good credit or a bad credit. If I plot these probablilites, we see it is S shaped curve between 0 and 1. If we convert the data to log scale by converting to log(p/1-p) the s curve gets transformed to a regression straight line. The co-efficients in the response of the model are the intercept and the slope parameters of this regression line. In the case below, we see that intercept = -1.66635138 and co-efficient of V2(slope) =0.03753769. you can see the regression line in the 2nd plot.

```
#cleaning environment and starting fresh.
rm(list = ls())
library(randomForest)
library(ggplot2)
library(caret)
```

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

```
library(pROC)
```

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

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

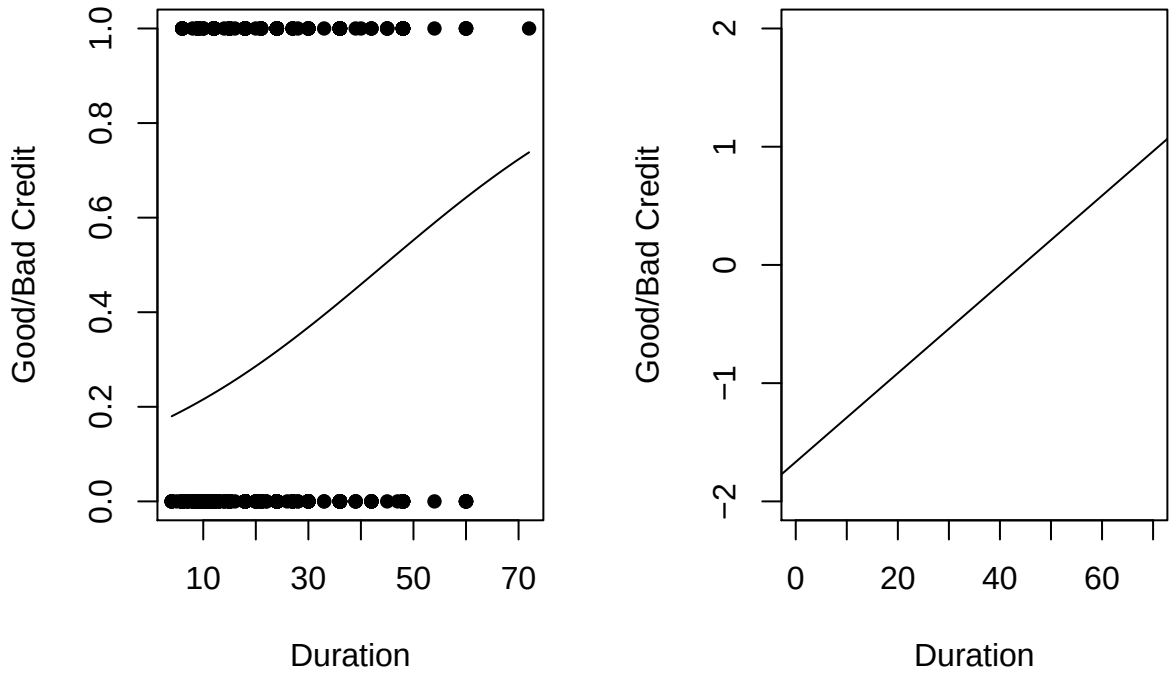
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
#setting seed for consistent results
set.seed(123)
#Read the text file into data.
data<-read.table("germancredit.txt", header=FALSE)

#factoring the response column, 0= good credit, 1= bad credit
data$V21[data$V21==1]<-0
```

```r
data$V21[data$V21==2]<-1

#creating sequence for duration
v2 <- seq(4,72, 1)

#creating test model of V21 vs V2, for the sake of understanding logistic regression
testModel<-glm(V21 ~ V2, data = data, family=binomial(link="logit"))
testModel$coefficients
```

```
## (Intercept)          V2
## -1.66635138  0.03753769
```

```r
#predictions of V21 using sample of V1 creating earlier
testPredictions<-predict(testModel, list(V2 = v2),type="response")

#setting to print 2 plots in same screen
par(mfrow=c(1,2))
#plotting V21 vs V2 with all dots on either 0 or 1
plot(data$V2, data$V21, pch = 16, xlab = "Duration", ylab = "Good/Bad Credit")
#plotting the prediction probablilities which will result in a s-shaped curve
lines(v2, testPredictions)
#plotting regression line.
plot(1, type="n", xlab="Duration", ylab="Good/Bad Credit", xlim=c(0,70), ylim=c(-2, 2))
abline(-1.66635138, 0.03753769)
```



I am splitting the data into randomly selected 70% training set and 30% into testing set. We will run the glm model on training set and run predictions on the testing set. Created confusion matrix using the predictions and actual data. Looking at the confusion matrix, the model has classified 178 correctly as good credit. 47 as good data but were actually bad credit. 26 as bad credit when they were actually good credit and 49 correctly as bad credit.

```r
head(data)
```

```
##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   0
## 2 A191 A201   1
## 3 A191 A201   0
## 4 A191 A201   0
## 5 A191 A201   1
## 6 A192 A201   0
```

```r
#randomly selecting 70% data for training and 30% for testing.
randomTrainingRows<-sample(1:nrow(data),round(0.70*nrow(data)))
trainingData<-data[randomTrainingRows,]
testingData<-data[-randomTrainingRows,]

# running logistic regression on training data
logitModel<-glm(V21 ~ ., data = trainingData, family=binomial(link="logit"))
summary(logitModel)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = trainingData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3765  -0.6592  -0.3506   0.6199   2.7732
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.131e+00  1.371e+00  -0.826 0.409079
## V1A12       -2.300e-01  2.707e-01  -0.850 0.395436
## V1A13       -7.906e-01  4.579e-01  -1.727 0.084217 .
## V1A14       -1.607e+00  2.887e-01  -5.565 2.63e-08 ***
## V2           3.316e-02  1.113e-02   2.979 0.002891 **
## V3A31        3.886e-01  7.048e-01   0.551 0.581384
## V3A32       -6.438e-01  5.712e-01  -1.127 0.259669
## V3A33       -6.974e-01  6.299e-01  -1.107 0.268225
## V3A34       -1.499e+00  5.748e-01  -2.607 0.009130 **
## V4A41       -1.871e+00  4.589e-01  -4.077 4.57e-05 ***
## V4A410      -1.448e+00  9.904e-01  -1.462 0.143799
## V4A42       -8.020e-01  3.284e-01  -2.442 0.014589 *
## V4A43       -8.683e-01  3.108e-01  -2.794 0.005209 **
## V4A44       -1.960e-01  9.672e-01  -0.203 0.839403
## V4A45        3.885e-01  7.031e-01   0.553 0.580580
## V4A46       -5.850e-01  5.131e-01  -1.140 0.254256
## V4A48       -1.452e+00  1.357e+00  -1.070 0.284801
## V4A49       -9.212e-01  4.239e-01  -2.173 0.029769 *
```

```
## V5           1.460e-04  5.318e-05   2.746 0.006033 **
## V6A62       -4.910e-01  3.662e-01  -1.341 0.180073
## V6A63       -9.304e-01  5.278e-01  -1.763 0.077903 .
## V6A64       -1.751e+00  6.703e-01  -2.612 0.008995 **
## V6A65       -1.204e+00  3.389e-01  -3.551 0.000384 ***
## V7A72        6.838e-01  5.403e-01   1.266 0.205652
## V7A73        3.936e-01  5.225e-01   0.753 0.451270
## V7A74       -4.145e-01  5.537e-01  -0.749 0.454059
## V7A75       -1.297e-01  5.235e-01  -0.248 0.804266
## V8           3.944e-01  1.090e-01   3.617 0.000298 ***
## V9A92        5.229e-02  4.744e-01   0.110 0.912228
## V9A93       -6.667e-01  4.672e-01  -1.427 0.153598
## V9A94       -1.792e-02  5.530e-01  -0.032 0.974145
## V10A102      5.693e-01  5.422e-01   1.050 0.293782
## V10A103     -9.706e-01  5.144e-01  -1.887 0.059191 .
## V11          2.448e-02  1.073e-01   0.228 0.819576
## V12A122      1.891e-01  3.193e-01   0.592 0.553703
## V12A123      4.270e-01  2.869e-01   1.488 0.136650
## V12A124      1.110e+00  6.484e-01   1.711 0.087001 .
## V13          1.887e-03  1.096e-02   0.172 0.863333
## V14A142     -3.944e-02  5.344e-01  -0.074 0.941172
## V14A143     -2.443e-01  3.002e-01  -0.814 0.415666
## V15A152     -5.696e-01  2.947e-01  -1.933 0.053292 .
## V15A153     -1.061e+00  6.918e-01  -1.534 0.125022
## V16          2.889e-01  2.544e-01   1.136 0.256084
## V17A172     -3.075e-01  8.077e-01  -0.381 0.703390
## V17A173     -2.913e-02  7.783e-01  -0.037 0.970149
## V17A174     -2.059e-01  7.808e-01  -0.264 0.791974
## V18          4.692e-01  3.096e-01   1.516 0.129584
## V19A192     -4.319e-01  2.519e-01  -1.714 0.086482 .
## V20A202     -1.136e+00  7.597e-01  -1.496 0.134732
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 844.80  on 699  degrees of freedom
## Residual deviance: 600.93  on 651  degrees of freedom
## AIC: 698.93
##
## Number of Fisher Scoring iterations: 5
```

```r
# running predictions on testing data
predictions <- predict(logitModel, newdata=testingData[,-21], type="response")
#rounding the predictions to get 0 and 1
preds<-round(predictions)
#dropping the names in the output
names(preds)<-NULL
#created confusion matrix
cm<-confusionMatrix(reference = as.factor(testingData$V21),data = as.factor(preds))
cat(" ### confusion matrix #### \n")
```

```
##  ### confusion matrix ####
```

```
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 178  47
##          1  26  49
##
##                Accuracy : 0.7567
##                  95% CI : (0.704, 0.8041)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.002227
##
##                   Kappa : 0.4065
##
##  Mcnemar's Test P-Value : 0.019241
##
##             Sensitivity : 0.8725
##             Specificity : 0.5104
##          Pos Pred Value : 0.7911
##          Neg Pred Value : 0.6533
##              Prevalence : 0.6800
##          Detection Rate : 0.5933
##    Detection Prevalence : 0.7500
##       Balanced Accuracy : 0.6915
##
##        'Positive' Class : 0
##
```

Using the p-value in the response of model, we can run one more regression using only the significant factors. There are other ways to determine the significant values like lasso method, stepwise conditional forward or backward methods. I am not doing those methods and going based on p-values which is approximate measure to select significant parameters. We see the accuracy of this new model went up from 0.75 to 0.77.

```
#running logistic regressin using only significant factors
logitModel2<-glm(V21 ~V1+V2+V3+V4+V6+V8+V14, data = trainingData, family=binomial(link="logit"))
summary(logitModel2)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V6 + V8 + V14, family = binomial(link = "logit"),
##     data = trainingData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0525  -0.7569  -0.4336   0.7449   2.5534
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.063383   0.645524  -0.098 0.921783
## V1A12       -0.212140   0.246029  -0.862 0.388548
## V1A13       -0.786148   0.425383  -1.848 0.064589 .
## V1A14       -1.570932   0.271464  -5.787 7.17e-09 ***
```

```
## V2           0.044616   0.008256   5.404 6.51e-08 ***
## V3A31        0.039672   0.620929   0.064 0.949057
## V3A32       -0.891910   0.500030  -1.784 0.074470 .
## V3A33       -0.865972   0.580666  -1.491 0.135872
## V3A34       -1.730835   0.527656  -3.280 0.001037 **
## V4A41       -1.508023   0.414751  -3.636 0.000277 ***
## V4A410      -1.153364   0.836264  -1.379 0.167837
## V4A42       -0.639882   0.295757  -2.164 0.030500 *
## V4A43       -0.904829   0.278798  -3.245 0.001173 **
## V4A44       -0.074219   0.912034  -0.081 0.935142
## V4A45        0.641127   0.672092   0.954 0.340120
## V4A46       -0.112897   0.479331  -0.236 0.813796
## V4A48       -1.741381   1.232772  -1.413 0.157781
## V4A49       -0.799248   0.380940  -2.098 0.035897 *
## V6A62       -0.392743   0.331088  -1.186 0.235536
## V6A63       -0.912937   0.502669  -1.816 0.069343 .
## V6A64       -1.358390   0.607941  -2.234 0.025456 *
## V6A65       -1.062982   0.316330  -3.360 0.000778 ***
## V8           0.233577   0.091679   2.548 0.010842 *
## V14A142      0.012098   0.498008   0.024 0.980620
## V14A143     -0.031030   0.278951  -0.111 0.911428
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 844.80  on 699  degrees of freedom
## Residual deviance: 656.76  on 675  degrees of freedom
## AIC: 706.76
##
## Number of Fisher Scoring iterations: 5
```

```
predictions2 <- predict(logitModel2, newdata=testingData[,-21], type="response")
cm2<-confusionMatrix(reference = as.factor(testingData$V21>0.5),data = as.factor(predictions2>0.5))
cat(" ### confusion matrix 2#### \n")
```

```
##  ### confusion matrix 2####
```

```
cm2
```

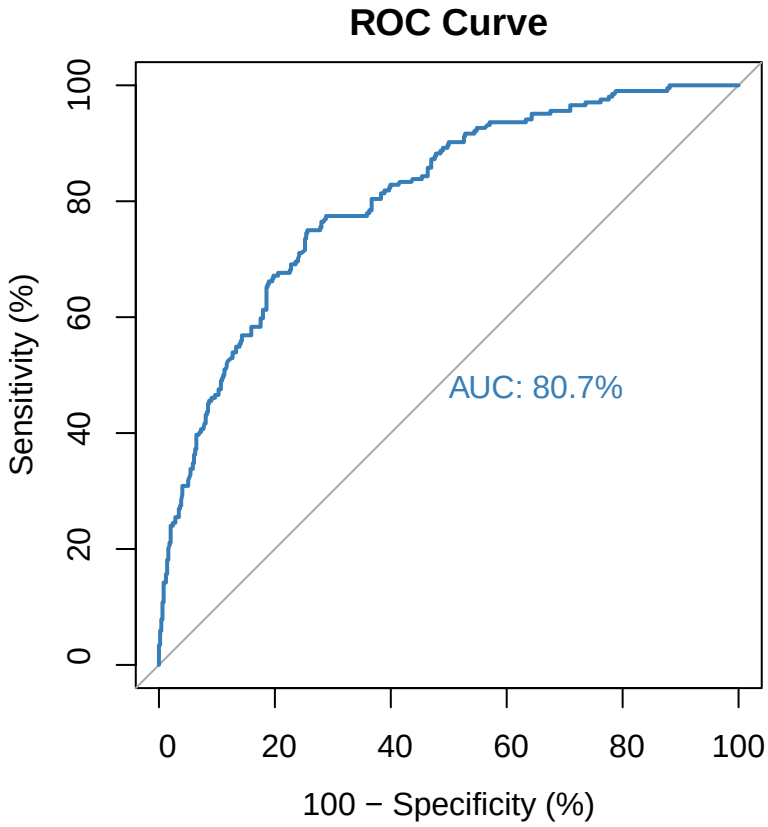
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   185   50
##      TRUE     19   46
##
##                Accuracy : 0.77
##                  95% CI : (0.7182, 0.8164)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.0003784
##
##                   Kappa : 0.4221
##
##  Mcnemar's Test P-Value : 0.0003043
##
```

```
##             Sensitivity : 0.9069
##             Specificity : 0.4792
##          Pos Pred Value : 0.7872
##          Neg Pred Value : 0.7077
##              Prevalence : 0.6800
##          Detection Rate : 0.6167
##    Detection Prevalence : 0.7833
##       Balanced Accuracy : 0.6930
##
##        'Positive' Class : FALSE
##
```

```
par(pty='s')
roc(trainingData$V21,logitModel2$fitted.values,plot=TRUE,
    legacy.axes=TRUE,percent=TRUE, col="#377eb8", print.auc=TRUE,main="ROC Curve" )
```

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

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




```
##
## Call:
## roc.default(response = trainingData$V21, predictor = logitModel2$fitted.values,     percent = TRUE, p
##
## Data: logitModel2$fitted.values in 496 controls (trainingData$V21 0) < 204 cases (trainingData$V21 1)
## Area under the curve: 80.71%
```

Question 10.3.2

Given the cost of incorrectly identifying a bad customer as good is 5 times worse than identifying a good customer bad, we can apply the cost analysis to the above model as:

cost= 185X0 + 50X5 + 19X1 +46X0 = 231.

To reduce this cost, we can alter the threshold of probablility we are ready to accept as good credit vs bad credit. I have run loop for different threshold values and captured the corresponding cost.

```
cost<-(5*39)+(1*23)
cost
```

```
## [1] 218
```

```
#Setting threshold
t=0.15

#copying predictions to new list.
predictionsTest<-predictions2
#Checking each prediction against the threshold.
predictionsTest[predictionsTest<t]<-0
predictionsTest[predictionsTest>=t]<-1
#Confusion Matrix with threshold
cm3<-confusionMatrix(reference = as.factor(testingData$V21),data = as.factor(predictionsTest))

cmTable<-as.matrix(cm3)
falseneg<-cmTable[1,2]
falsepos<-cmTable[2,1]
cost<-(5*falseneg) + (falsepos)
cat("cost with fixed threshold of 0.15: ",cost)
```

```
## cost with fixed threshold of 0.15:  181
```

```
#data frame to capture accuracy, cost for different values of threshold.
thresholdFrame <-data.frame(threshold=numeric(), cost=numeric(), accuracy=numeric())
thresholdList<-seq(0.09,0.5,by=0.01)
for(t in thresholdList){
  predictionsLoop<-predictions2
  predictionsLoop[predictionsLoop<t]<-0
  predictionsLoop[predictionsLoop>=t]<-1
  cmLoop<-confusionMatrix(reference = as.factor(testingData$V21),data = as.factor(predictionsLoop))
  cmTable<-as.matrix(cmLoop)
  fn<-cmTable[1,2]
  fp<-cmTable[2,1]
  cost<-(5*fn) + (fp)
  thresholdFrame[nrow(thresholdFrame) + 1,] = c(t,cost,cmLoop$overall['Accuracy'] )
}
result<-thresholdFrame[which.min(thresholdFrame$cost),]
cat("\n#### Lowest cost thresholds ####")
```

```
##
## #### Lowest cost thresholds ####
```

```
result
```

```
##   threshold cost  accuracy
## 2       0.1  171 0.5366667
```

```r
cat("threshold with lowest cost: ",result$threshold)
```

```
## threshold with lowest cost:  0.1
```

```r
barPlot <- barplot(height = thresholdFrame$cost, names=thresholdFrame$threshold, space=0.4, xaxt='n', x
main="Cost v/s threshold", col = "blue")
```