R Notebook

Code

Question 1

Using the same crime data set as in Homework 3 Question 4, apply Principal Component Analysis and then create a regression model using the first 4 principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Homework 3 Question 4. You can use the R function prcomp for PCA. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function.)

Hide

```
library(knitr)
library(ggplot2)
library(plotly)
library(RColorBrewer)
library(DAAG)
library(gridExtra)
```

Hide

```
df<-read.table('uscrime.txt',header=TRUE)
```

Running PCA

Running PCA is simple as a single line. We need to make sure we scale the data to make sure that when PCA determines the most variance in a certain direction, all the data is scaled to the same dimension or else data with large values will probably show the largest variance.

Hide

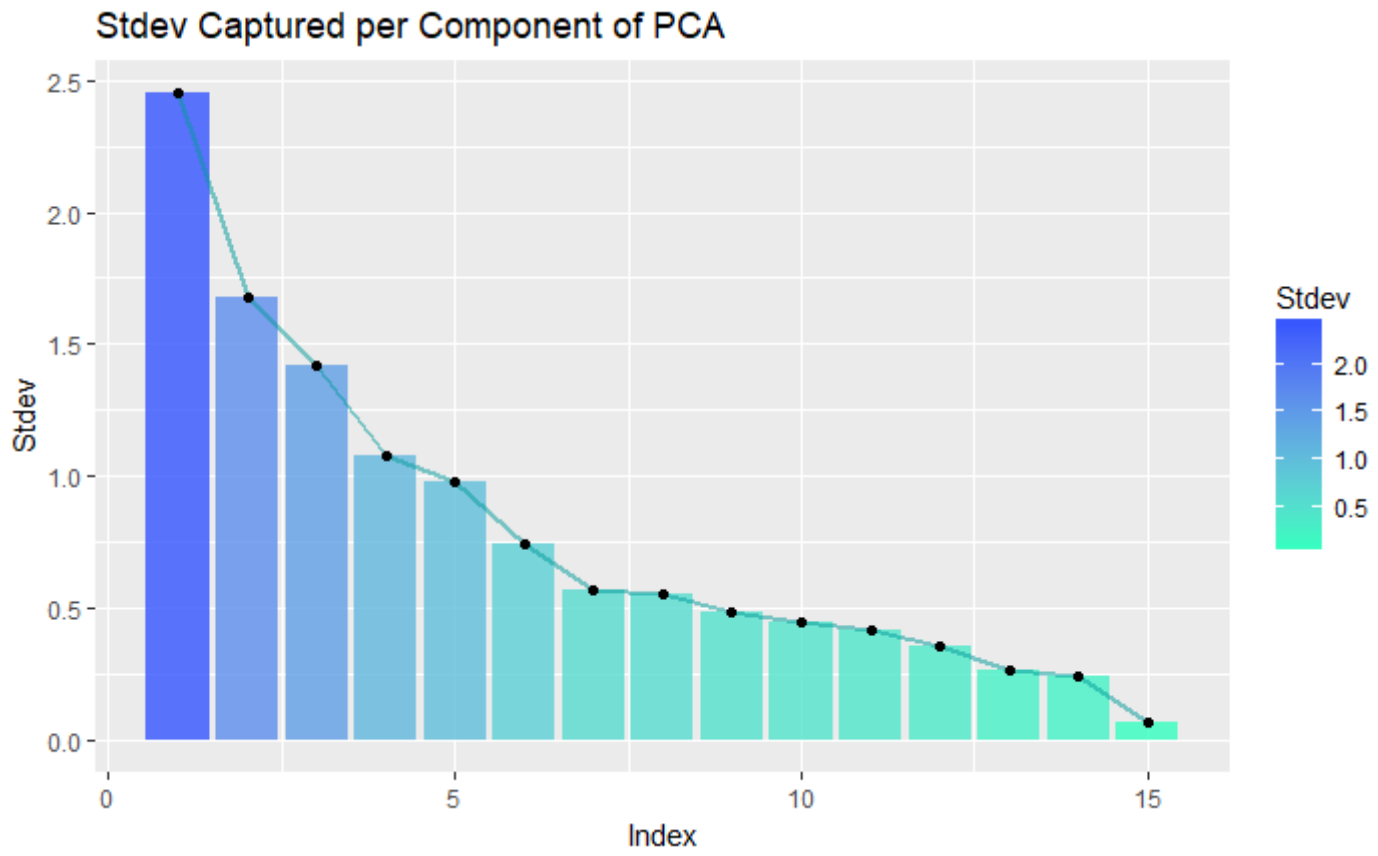
```
prdf <- prcomp(df[,1:15],scale=TRUE)
```

PCA variance

To see how much of the variance PCA captures per individual

Hide

```
pcaStdv <- data.frame(prdf$sdev)
pcaStdv$idx <- as.numeric(row.names(pcaStdv))
colnames(pcaStdv) <- c("Stdev","Index")
#can also use:
#screeplot(prdf, type="lines",col="blue")
#But mine's prettier =)
ggplot(pcaStdv,aes(x=Index,y=Stdev,fill=Stdev))+geom_bar(stat="identity",alpha=0.8)+geom_line(co
lor='#009999',size=1,alpha=0.5)+geom_point()+scale_fill_gradient(low="#33FFBF",high="#3354FF")+l
abs(title="Stdev Captured per Component of PCA")
```



From the above data, we do see an elbow joint at 4 PCA components but also another one at 7 PCA components. The homework asked us to use 4 but I am curious to see if using 7 will provide any additional improvements.

Running lm on PCA

I decided to run lm with 5-fold cross validation on PCA with 4 components, 7 components, and all 15 components. I compared this to our previous lm model which we ran for all of the variables none PCAed and the variables we selected to be the best. The quality of the model was assessed using the R^2 value of each model.

Hide

```
pcaFour <- data.frame(cbind(prdf$x[,1:4],df$Crime))
pcaSeven <- data.frame(cbind(prdf$x[,1:7],df$Crime))
pcaFifteen <- data.frame(cbind(prdf$x[,1:15],df$Crime))
modelFour <- lm(V5~.,data=pcaFour)
modelSeven <- lm(V8~.,data=pcaSeven)
modelFifteen <- lm(V16~.,data=pcaFifteen)
modelSelect <- lm(Crime~Ineq+Ed+Prob+M+U2+Po1+Crime,data=df)
```

Hide

```
modelAll <- lm(Crime~.,data=df)
cvFour <- cv.lm(pcaFour,modelFour,m=5,plotit=FALSE)
cvSeven <- cv.lm(pcaSeven,modelSeven,m=5,plotit=FALSE)
cvFifteen <- cv.lm(pcaFifteen,modelFifteen,m=5,plotit=FALSE)
cvSelect <- cv.lm(df,modelSelect,m=5,plotit=FALSE)
```

Hide

```
cvAll <- cv.lm(df,modelAll,m=5,plotit=FALSE)
```

Hide

```
SStot <- sum((df$Crime - mean(df$Crime))^2)
SSres_Four <- attr(cvFour,"ms")*nrow(df)
SSres_Seven <- attr(cvSeven,"ms")*nrow(df)
SSres_Fifteen <- attr(cvFifteen,"ms")*nrow(df)
SSres_Select <- attr(cvSelect,"ms")*nrow(df)
SSres_All <- attr(cvAll,"ms")*nrow(df)
R2_Four <- 1-SSres_Four/SStot
R2_Seven <- 1-SSres_Seven/SStot
R2_Fifteen <- 1-SSres_Fifteen/SStot
R2_Select <- 1-SSres_Select/SStot
R2_All <- 1-SSres_All/SStot
print(paste0("R^2 value of model Four is: ",R2_Four))
```

```
[1] "R^2 value of model Four is: 0.105671184448938"
```

Hide

```
print(paste0("R^2 value of model Seven is: ",R2_Seven))
```

```
[1] "R^2 value of model Seven is: 0.456232483740589"
```

Hide

```
print(paste0("R^2 value of model Fifteen is: ",R2_Fifteen))
```

```
[1] "R^2 value of model Fifteen is: 0.413363818818907"
```

Hide

```
print(paste0("R^2 value of model Select is: ",R2_Select))
```

```
[1] "R^2 value of model Select is: 0.638459146624993"
```

Hide

```
print(paste0("R^2 value of model All is: ",R2_All))
```

```
[1] "R^2 value of model All is: 0.413363818818906"
```

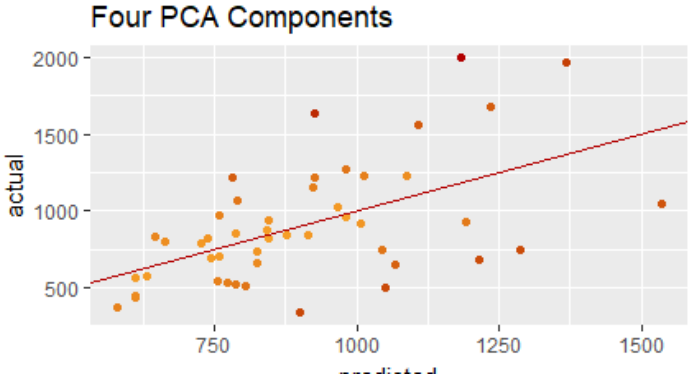
Using the selected values from our last homework assignment still gave the best scores, however, if we had to run PCA, picking 7 components rather than 4 gave a significantly better performance.

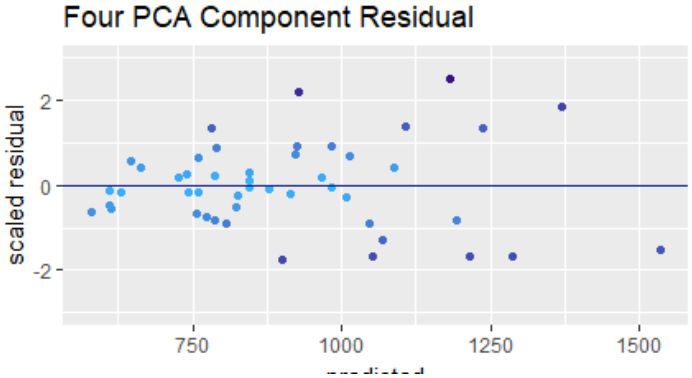
Plotting Fitted Value Against Actual Value

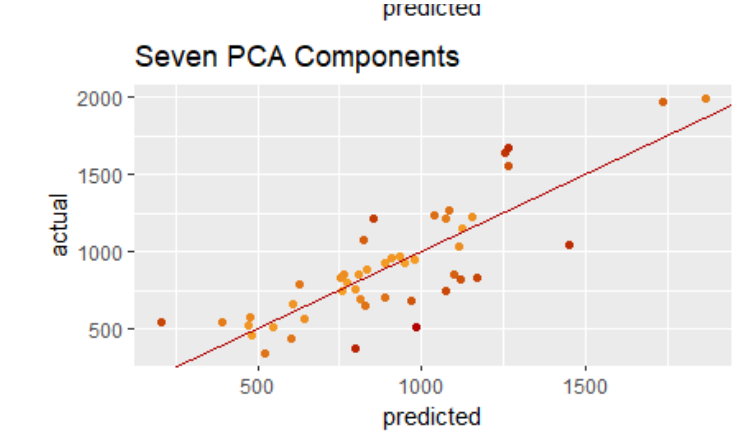
Now we can plot the fitted value against the actual value to see how well they lined up. I've also included a scaled residual plot to do some additional visual comparisons to see the spread comparison.

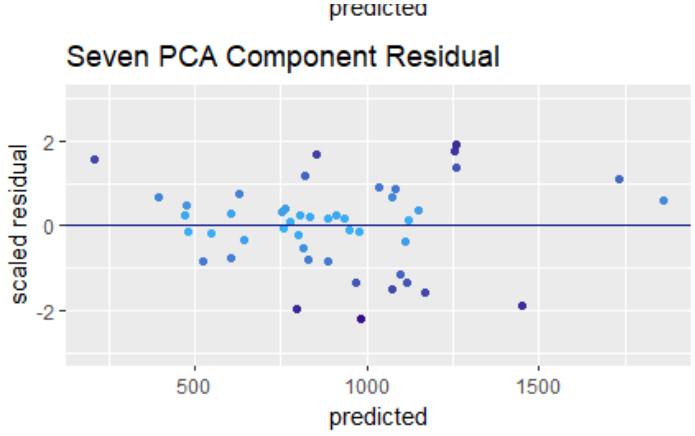
Hide

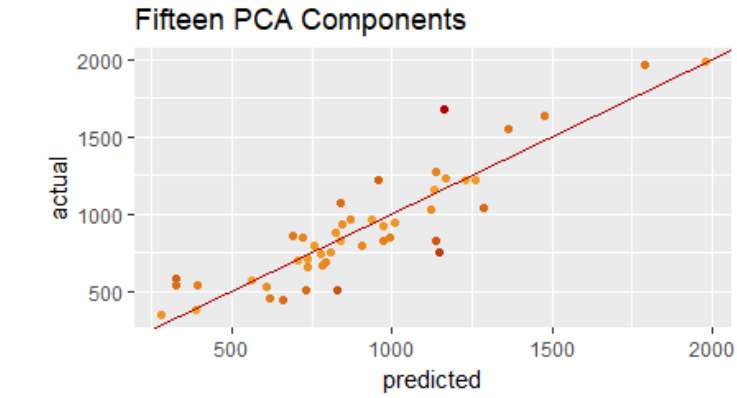
```
FourPlot    <- qplot(modelFour$fitted.values, df$Crime, colour=scale(modelFour$residuals))
+   geom_abline(slope=1,colour="#B20000")   +   labs(title="Four PCA
Components",x="predicted",y="actual") +
scale_colour_gradient2(low="#B20000",mid="#F8A329",high="#B20000",midpoint = 0,guide=FALSE)
SevenPlot   <- qplot(modelSeven$fitted.values, df$Crime, colour=scale(modelSeven$residuals))
+   geom_abline(slope=1,colour="#B20000")   +   labs(title="Seven PCA
Components",x="predicted",y="actual") +
scale_colour_gradient2(low="#B20000",mid="#F8A329",high="#B20000",midpoint = 0,guide=FALSE)
FifteenPlot <- qplot(modelFifteen$fitted.values, df$Crime, colour=scale(modelFifteen$residuals))
    +   geom_abline(slope=1,colour="#B20000")   +   labs(title="Fifteen PCA Components",x="predi
cted",y="actual") + scale_colour_gradient2(low="#B20000",mid="#F8A329",high="#B20000",midpoint =
 0,guide=FALSE)
SelectPlot  <- qplot(modelSelect$fitted.values, df$Crime, colour=scale(modelSelect$residuals))
+   geom_abline(slope=1,colour="#B20000")   +   labs(title="Select Features",x="predicted",y="ac
tual") + scale_colour_gradient2(low="#B20000",mid="#F8A329",high="#B20000",midpoint = 0,guide=FA
LSE)
AllPlot     <- qplot(modelAll$fitted.values, df$Crime, colour=scale(modelAll$residuals))
+   geom_abline(slope=1,colour="#B20000")   +   labs(title="All Features",x="predicted",y="actua
l") + scale_colour_gradient2(low="#B20000",mid="#F8A329",high="#B20000",midpoint =
0,guide=FALSE)
FourResid       <- qplot(modelFour$fitted.values, scale(modelFour$residuals), colour=scale(model
Four$residuals),ylim=c(-3,3)) + geom_hline(yintercept = 0,colour='#111C89') + labs(title="Four P
CA Component Residual",x="predicted",y="scaled residual") +
scale_colour_gradient2(low="#361189",mid="#33B9FF",high="#361189",midpoint = 0,guide=FALSE)
SevenResid      <- qplot(modelSeven$fitted.values, scale(modelSeven$residuals), colour=scale(mod
elSeven$residuals),ylim=c(-3,3))+geom_hline(yintercept = 0,colour='#111C89')+labs(title="Seven P
CA Component Residual",x="predicted",y="scaled residual") +
scale_colour_gradient2(low="#361189",mid="#33B9FF",high="#361189",midpoint = 0,guide=FALSE)
FifteenResid    <- qplot(modelFifteen$fitted.values, scale(modelFifteen$residuals),
colour=scale(modelFifteen$residuals),ylim=c(-3,3))+geom_hline(yintercept = 0,colour='#111C89')+l
abs(title="Fifteen PCA Component Residual",x="predicted",y="scaled residual") + scale_colour_gra
dient2(low="#361189",mid="#33B9FF",high="#361189",midpoint = 0,guide=FALSE)
SelectResid     <- qplot(modelSelect$fitted.values, scale(modelSelect$residuals), colour=scale(m
odelSelect$residuals),ylim=c(-3,3))+geom_hline(yintercept = 0,colour='#111C89')+labs(title="Sele
ct Features Residual",x="predicted",y="scaled residual") +
scale_colour_gradient2(low="#361189",mid="#33B9FF",high="#361189",midpoint = 0,guide=FALSE)
AllResid        <- qplot(modelAll$fitted.values, scale(modelAll$residuals), colour=scale(modelAl
l$residuals),ylim=c(-3,3))+geom_hline(yintercept = 0,colour='#111C89')+labs(title="All
Features",x="predicted",y="scaled residual") + scale_colour_gradient2(low="#361189",mid="#33B9F
F",high="#361189",midpoint = 0,guide=FALSE)
grid.arrange(FourPlot, FourResid, SevenPlot, SevenResid, FifteenPlot, FifteenResid, SelectPlot,
SelectResid, AllPlot, AllResid, ncol=2)
```

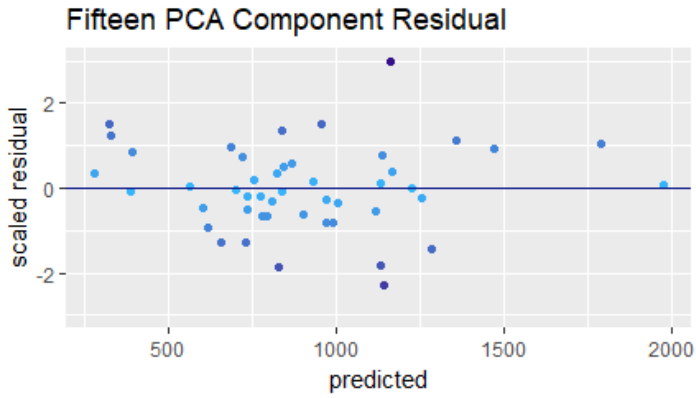


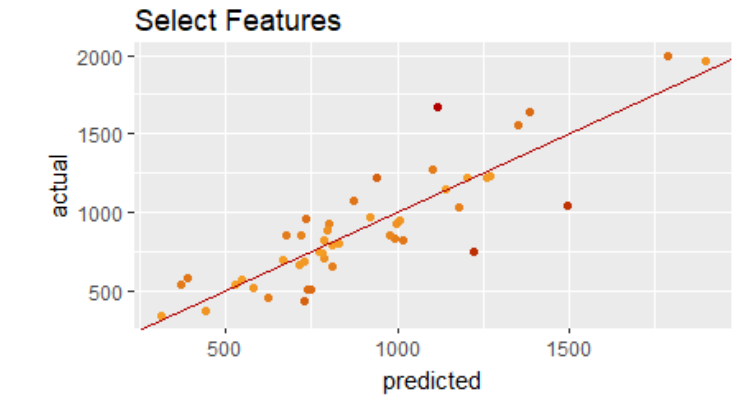


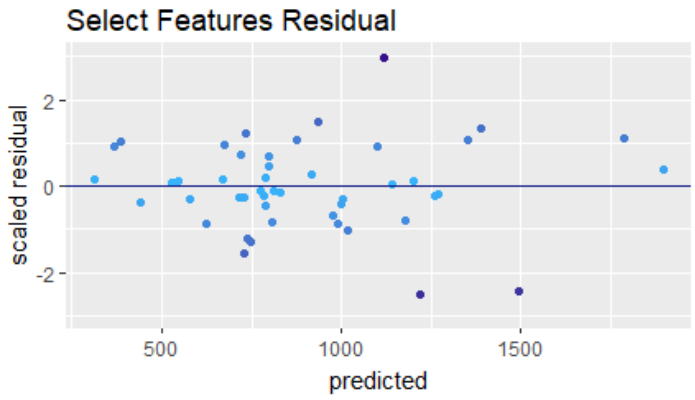


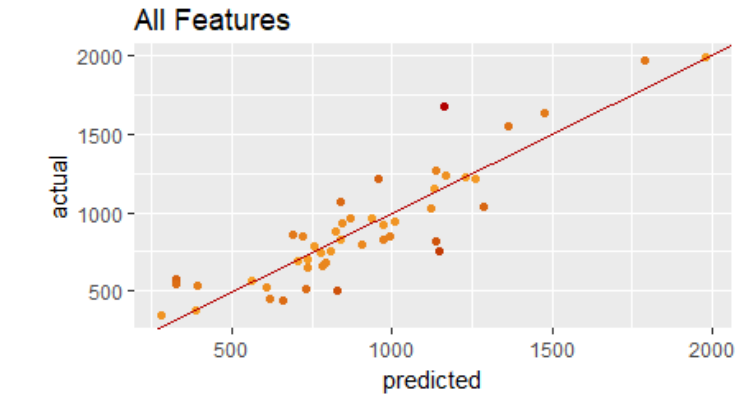


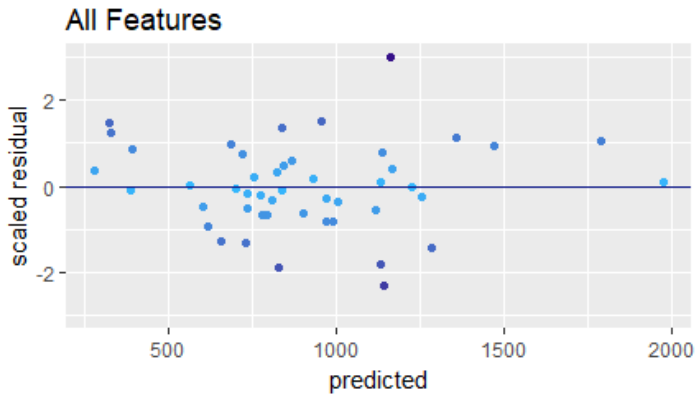












PCA coefficient to data coefficient

To get the original coefficients, you would need to multiply the coefficients of the PCA model by the eigenvectors generated from PCA, this was made easy using the $rotation dataframe of pca.

Hide

```
beta0 <- modelFour$coefficients[1]
betas <- modelFour$coefficients[2:5]
alphas <- prdf$rotation[,1:4] %*% betas
```

Hide

```
colnames(alphas) <- "coefficients"
kable(alphas, align='l')
```

	coefficients
M	-21.28
So	10.22
Ed	14.35
Po1	63.46
Po2	64.56
LF	-14.01
M.F	-24.44
Pop	39.83
NW	15.44
U1	-27.22
U2	1.43
Wealth	38.61
Ineq	-27.54
Prob	3.30
Time	-6.61

Question 2

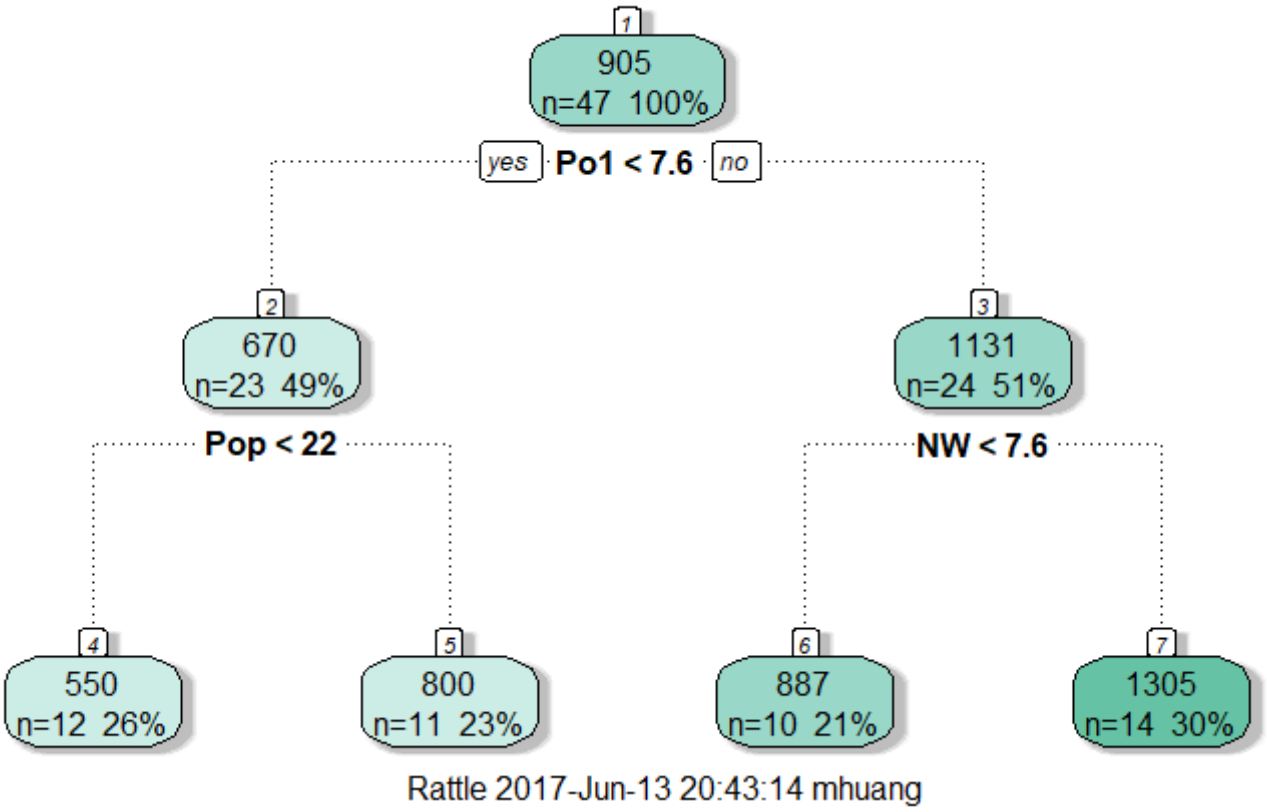
Using the same crime data set as in Homework 3 Question 4, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

Running and viewing trees generated from rpart library

Because there are two different packages that can run the decision trees, I wanted to do some quick tests on both to see how well they perform. First, I tested the rpart library and it provided 4 leaf nodes.

Hide

```
library(rpart)
library(rattle)
library(RColorBrewer)
crimedf <- read.table('uscrime.txt',header=TRUE)
rpartAll <- rpart(Crime~.,data=crimedf)
fancyRpartPlot(rpartAll, palettes=c("BuGn"))
```

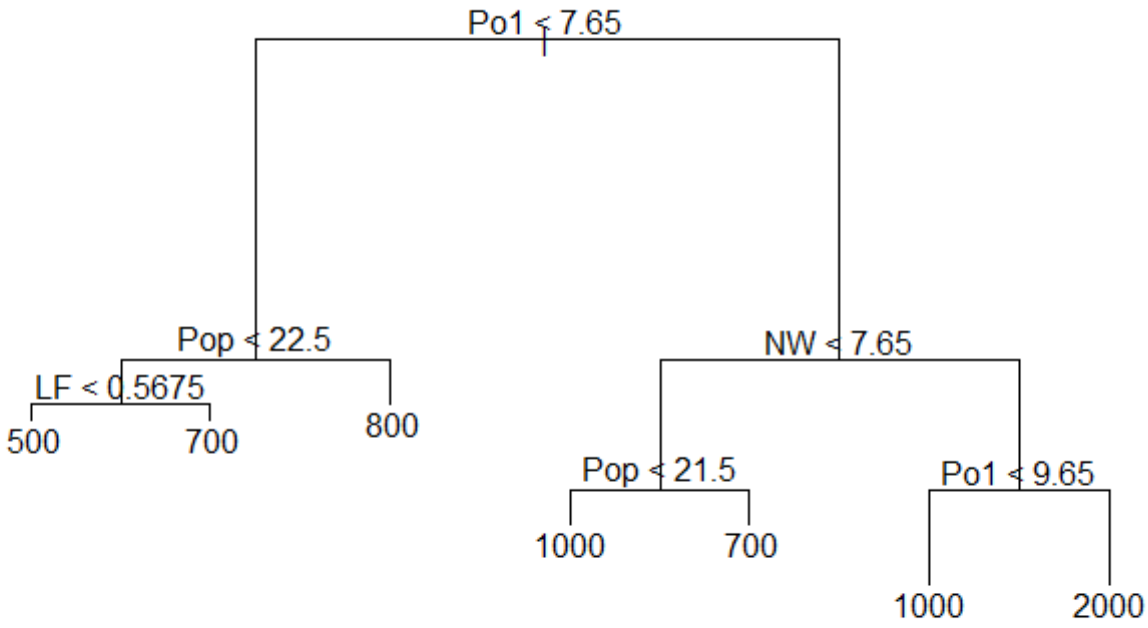


running and viewing trees generated from tree library

Next testing the tree library to see how it splits the data and it's a bit more specifc. It generated 7 leaf nodes which would definitely overfit our dataset.

Hide

```
library(tree)
treeAll <- tree(Crime~.,data=data.frame(crimedf))
plot(treeAll)
text(treeAll)
```



Non CV R^2

Doing a quick R^2 check even though I feel like the tree model would do better because it looks more overfit.

Hide

```
treeyhat <- predict(treeAll,crimedf)
treeSSres <- sum((treeyhat-crimedf$Crime)^2)
rpartyhat <- predict(rpartAll,crimedf)
rpartSSres <- sum((rpartyhat-crimedf$Crime)^2)
SStot <- sum((crimedf$Crime - mean(crimedf$Crime))^2)
treeR2=1-treeSSres/SStot
rpartR2=1-rpartSSres/SStot
print(paste0("tree library gave R^2 of ",treeR2))
```

```
[1] "tree library gave R^2 of 0.724496208475934"
```

Hide

```
print(paste0("rpart library gave R^2 of ",rpartR2))
```

```
[1] "rpart library gave R^2 of 0.562837788062114"
```

Yup, we definitely see a higher R^2 value because of this.

Prune data back

Because we really have barely any data, we really shouldn't have more than just 1 split. But we know that this would probably give a bad R^2 valued

Hide

```
treePrune <- prune.tree(treeAll, best=2)
prunyhat <- predict(treePrune,crimedf)
prunSSres <- sum((prunyhat - crimedf$Crime)^2)
pruneR2=1-prunSSres/SStot
print(paste0("pruned tree gave R^2 of ",pruneR2))
```

```
[1] "pruned tree gave R^2 of 0.362962932452636"
```

The ideal thing to do for this model is to actually run linear regression for each branch, then run cross validation on each of these model. This would give a better depiction of how well it works. However, I did run out of time for this project.

pruning tree library

I'm going to use the best parameter to prune this tree by designating how many final leaf nodes I want. I tested 2, 3, 4, and 5 leaf nodes. Below are the R^2 value for each.

Hide

```
treeSSres  <- 0
pruneSSres <- 0
pruneThreeSSres <- 0
pruneFourSSres <- 0
pruneFiveSSres <- 0
for (i in 1:nrow(crimedf))
{
  treeModel <- tree(Crime~.,data=crimedf[-i,])
  pruneModel <- prune.tree(treeModel, best=2)
  pruneThreeModel <- prune.tree(treeModel, best=3)
  pruneFourModel <- prune.tree(treeModel, best=4)
  pruneFiveModel <- prune.tree(treeModel, best=5)

  treeSSres <- treeSSres + (predict(treeModel,crimedf[i,]) - crimedf[i,16])^2
  pruneSSres <- pruneSSres + (predict(pruneModel,crimedf[i,]) - crimedf[i,16])^2
  pruneThreeSSres <- pruneThreeSSres + (predict(pruneThreeModel,crimedf[i,]) - crimedf[i,16])^2
  pruneFourSSres <- pruneFourSSres + (predict(pruneFourModel,crimedf[i,]) - crimedf[i,16])^2
  pruneFiveSSres <- pruneFiveSSres+ (predict(pruneFiveModel,crimedf[i,]) - crimedf[i,16])^2
}
SStot <- sum((crimedf$Crime - mean(crimedf$Crime))^2)
treeR2  <- 1-treeSSres/SStot
pruneR2 <- 1-pruneSSres/SStot
pruneThreeR2 <- 1-pruneThreeSSres/SStot
pruneFourR2 <- 1-pruneFourSSres/SStot
pruneFiveR2 <- 1-pruneFiveSSres/SStot
print(paste0("tree R2 value: ",treeR2))
```

```
[1] "tree R2 value: -0.295789680305296"
```

Hide

```
print(paste0("prune R2 value: ",pruneR2))
```

```
[1] "prune R2 value: 0.00508390713967466"
```

Hide

```
print(paste0("prune Three R2 value: ",pruneThreeR2))
```

```
[1] "prune Three R2 value: -0.350330820723268"
```

Hide

```
print(paste0("prune Four R2 value: ",pruneFourR2))
```

```
[1] "prune Four R2 value: -0.351472229212876"
```

Hide

```
print(paste0("prune Five R2 value: ",pruneFiveR2))
```

```
[1] "prune Five R2 value: -0.369245771002201"
```

With the cross validation, we can see that even though the tree model has a high R^2 value when trained and tested on the dataset, it would do poorly in a real world scenario. This is because these overfit the data significantly and it is not able to extrapolate well. When I pruned it down to only 2 leaf nodes, it performanced much better. When I pruned it down to 3, 4, and 5, it did worse which shows that we can overfit the model really quickly.

pruning rpart library

Now I'm going to test the prunning ability of the rpart library. This isn't as easy as designating the number of leaf nodes, but instead you designate this by a cp value which defines how complex the model will be. By using the cptable component of the model, we're able to get the cp value for each individual leaf split.

Hide

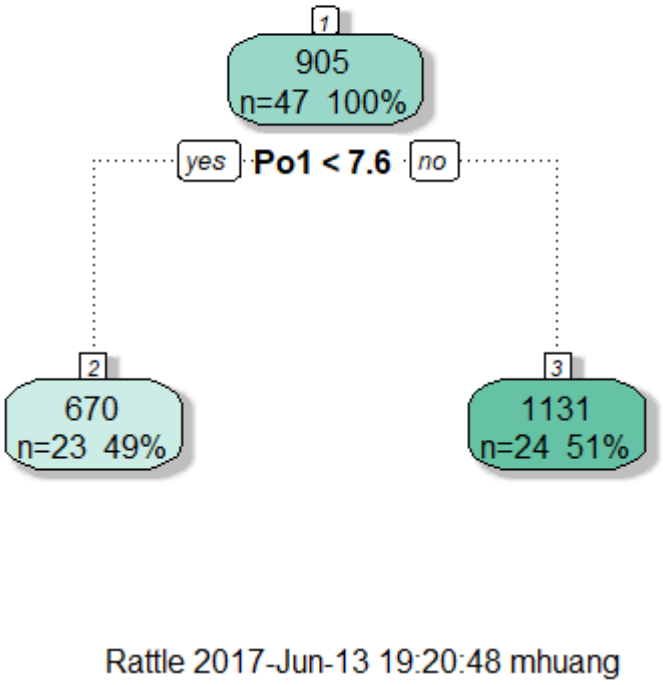
```
kable(rpartAll$cptable,align='l')
```

CP	nsplit	rel error	xerror	xstd
0.363	0	1.000	1.057	0.265
0.148	1	0.637	0.992	0.234
0.052	2	0.489	1.016	0.219
0.010	3	0.437	0.946	0.208

From the data above, we can see that cp 0.15 will probably give us a single split.

Hide

```
prunePartModel <- prune(rpartAll,cp=0.15)
fancyRpartPlot(prunePartModel, palettes=c("BuGn"))
```



So from this, we can test the R^2 value of every individual split.

Hide

```
rpartSSres <- 0
onePartSSres <- 0
twoPartSSres <- 0
threePartSSres <- 0
for (i in 1:nrow(crimedf))
{
  rpartModel <- rpart(Crime~.,data=crimedf[-i,])
  onePartModel <- prune(rpartModel,cp=0.15)
  twoPartModel <- prune(rpartModel,cp=0.1)
  threePartModel <- prune(rpartModel,cp=0.02)
  rpartSSres <- rpartSSres + (predict(rpartModel,crimedf[i,]) - crimedf[i,16])^2
  onePartSSres <- onePartSSres + (predict(onePartModel,crimedf[i,]) - crimedf[i,16])^2
  twoPartSSres <- twoPartSSres + (predict(twoPartModel,crimedf[i,]) - crimedf[i,16])^2
  threePartSSres <- threePartSSres + (predict(threePartModel,crimedf[i,]) - crimedf[i,16])^2
}
SStot <- sum((crimedf$Crime - mean(crimedf$Crime))^2)
rpartR2 <- 1-rpartSSres/SStot
onePartR2 <- 1-onePartSSres/SStot
twoPartR2 <- 1-twoPartSSres/SStot
threePartR2 <- 1-threePartSSres/SStot
print(paste0("rpart R2 value: ",rpartR2))
```

```
[1] "rpart R2 value: -0.0972918677678238"
```

Hide

```
print(paste0("onePart R2 value: ",onePartR2))
```

```
[1] "onePart R2 value: -0.172506458385694"
```

Hide

```
print(paste0("twoPart R2 value: ",twoPartR2))
```

```
[1] "twoPart R2 value: -0.163212301543434"
```

Hide

```
print(paste0("threePart R2 value: ",threePartR2))
```

```
[1] "threePart R2 value: -0.0972918677678238"
```

Variable importance

One thing that these models provide to you is the variable.importance column which provides you with a ranking of how good it is to split on a certain variable. In our case, every single model decided that Po1 and Po2 are the best components to split on.

Hide

```
rpartAll$variable.importance
```

```
    Po1     Po2  Wealth    Ineq    Prob       M      NW     Pop    Time      Ed      LF      So 
2497522 2497522 1628818 1602212 1520231 1388628 1245884  661771  601906  569546  203873  161801 
```

Randomforest

Now I will run the randomForest model to test its performance which actually performanced really well.

Hide

```
library(randomForest)
numpred <- 5
forestData <- randomForest(Crime~., data=crimedf, mtry=numpred,importance=TRUE)
forestyhat <-predict(forestData)
SSres <-sum((forestyhat-crimedf$Crime)^2)
SStot <- sum((crimedf$Crime - mean(crimedf$Crime))^2)
R2=1-SSres/SStot
print(paste0("R2 value: ",R2))
```

```
[1] "R2 value: 0.43143876108508"
```

Cross Validation on Random Forest

To run cross validation, I had to use the leave 1 out method where the model trained on the rest of the data set and tested on the single data point that I left out. The provided R value is quite close to the R value we calculated from the previous step, showing that randomForest is really good at not overfitting or underfitting.

Hide

```
CVSSres <- 0
for (i in 1:nrow(crimedf))
{
  forestModel <- randomForest(Crime ~.,data=crimedf[-i,], mtry=numpred, importance = TRUE)
  CVSSres=CVSSres+(predict(forestModel, newdata=crimedf[i,]) - crimedf[i,16])^2
}
SStot <- sum((df$Crime - mean(df$Crime))^2)
R2=1-CVSSres/SStot
print(paste0("CV R2 value: ",R2))
```

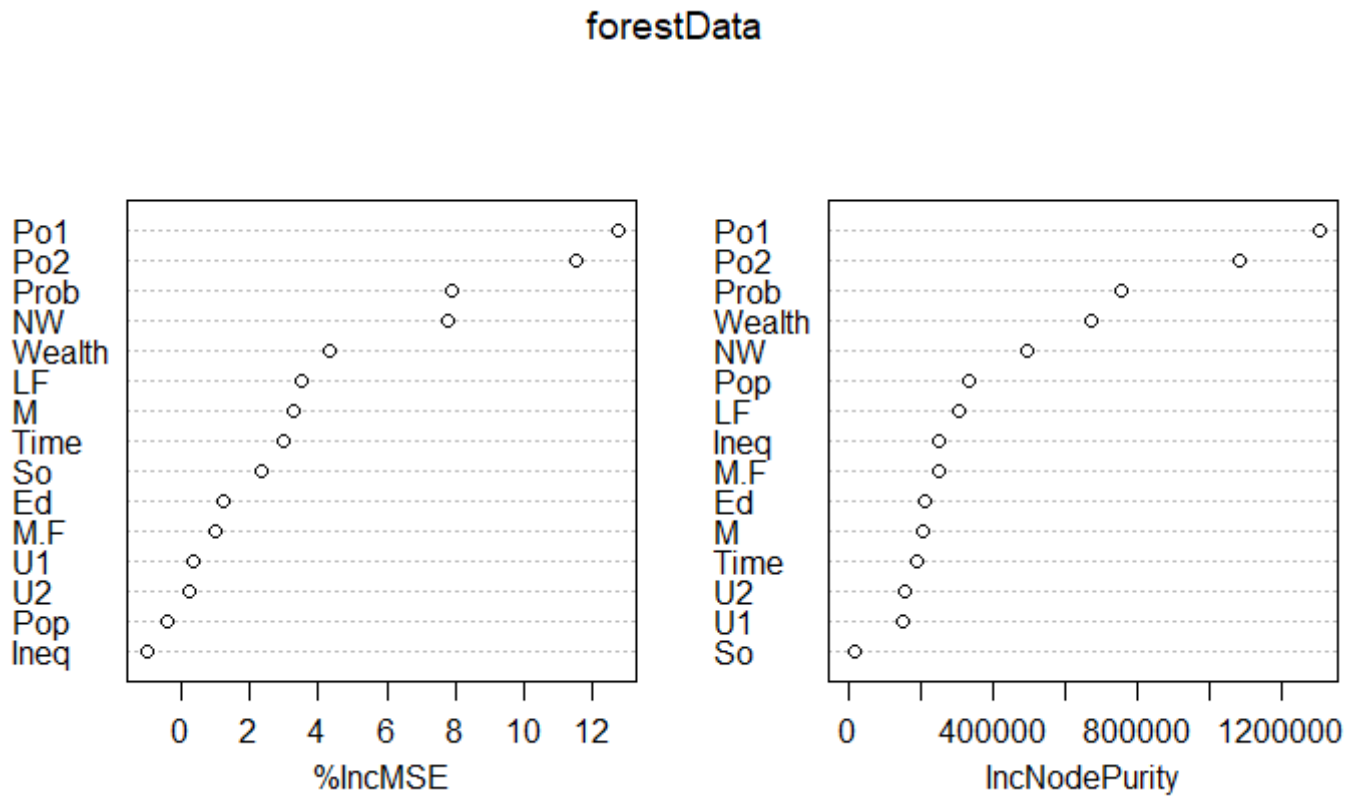
```
[1] "CV R2 value: 0.423052258052691"
```

There’s also additional metrics that can help identify components that are helpful in separating the data. Below are two of them which identifies Po1 and Po2 as major impact factors.

Hide

```
importance(forestData)
```

```
       %IncMSE IncNodePurity
M        3.298        206311
So       2.353         19315
Ed       1.202        214507
Po1     12.765       1306205
Po2     11.514       1082346
LF       3.481        306884
M.F      0.979        252441
Pop     -0.440        332450
NW       7.811        498001
U1       0.336        149666
U2       0.245        158145
Wealth   4.327        671945
Ineq    -1.026        252524
Prob     7.891        754485
Time     2.958        188670
```



Question 3

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.

Response

Logistic regression model would be useful in predicting the whether a patient will be admitted into the ED within the next year. Some of the factors that contribute to this include: * Whether they've been to the ED in the past year(risk of readmission) * Whether they have cardio vascular diseases * Whether they currently have a grouper of risky diagnoses * Whether they are currently on vasodilators * Whether they are overweight

Question 4 Part 1

Using the GermanCredit (http://archive.ics.uci.edu/ml/machine-learningdatabases/statlog/german/) data set (description (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.

Logistic Regression run on all data

I ran the logistic regression with all of the components, this can help identify features that I want to select but also as a basis to compare another model against.

Hide

```
oldCredDF <- read.table("germancredit.txt", sep=" ")
#convert people who are good(1) to 0 and bad(2) to 1
oldCredDF$V21[oldCredDF$V21==1] <- 0
oldCredDF$V21[oldCredDF$V21==2] <- 1
credSub <- sample(1:nrow(oldCredDF), size=round(0.8*(nrow(oldCredDF))))
credTrain <- oldCredDF[credSub,]
credVal <- oldCredDF[-credSub,]
model=glm(V21~., family=binomial(link="logit"), data=credTrain)
y_hat <- predict(model, credVal, type="response")
y_pred <- as.integer(y_hat >0.5)
table(y_pred, credVal$V21)
```

```
y_pred   0   1
     0 121  30
     1  21  28
```

Logistic Regression on Selected Data

I'm using summary to determine which variables to select by the p value.

Hide

```
summary(model)
```

```
Call:
glm(formula = V21 ~ ., family = binomial(link = "logit"), data = credTrain)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-2.549  -0.672  -0.349   0.657   2.743

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.95e-01   1.42e+00    0.28  0.78136
V1A12       -5.77e-01   2.51e-01   -2.30  0.02131 *
V1A13       -1.13e+00   4.41e-01   -2.56  0.01041 *
V1A14       -1.72e+00   2.61e-01   -6.58  4.7e-11 ***
V2           1.68e-02   1.09e-02    1.54  0.12406
V3A31        6.92e-02   6.39e-01    0.11  0.91382
V3A32       -6.40e-01   5.18e-01   -1.24  0.21622
V3A33       -9.18e-01   5.54e-01   -1.66  0.09715 .
V3A34       -1.50e+00   5.23e-01   -2.87  0.00410 **
V4A41       -1.97e+00   4.31e-01   -4.58  4.7e-06 ***
V4A410      -1.08e+00   8.55e-01   -1.26  0.20634
V4A42       -8.57e-01   3.07e-01   -2.79  0.00532 **
V4A43       -8.40e-01   2.79e-01   -3.01  0.00258 **
V4A44       -5.13e-01   7.83e-01   -0.66  0.51204
V4A45       -2.06e-01   6.49e-01   -0.32  0.75063
V4A46       -2.85e-01   4.58e-01   -0.62  0.53403
V4A48       -2.08e+00   1.36e+00   -1.54  0.12422
V4A49       -7.70e-01   3.78e-01   -2.04  0.04135 *
V5           1.78e-04   5.09e-05    3.50  0.00047 ***
V6A62       -4.96e-01   3.21e-01   -1.55  0.12158
V6A63       -6.90e-01   4.69e-01   -1.47  0.14154
V6A64       -9.91e-01   5.63e-01   -1.76  0.07858 .
V6A65       -1.23e+00   3.06e-01   -4.03  5.5e-05 ***
V7A72       -2.08e-01   4.85e-01   -0.43  0.66714
V7A73       -4.16e-01   4.61e-01   -0.90  0.36672
V7A74       -1.10e+00   5.02e-01   -2.18  0.02889 *
V7A75       -6.08e-01   4.68e-01   -1.30  0.19415
V8           4.48e-01   1.04e-01    4.32  1.6e-05 ***
V9A92       -5.93e-01   4.50e-01   -1.32  0.18770
V9A93       -1.06e+00   4.42e-01   -2.39  0.01664 *
V9A94       -4.12e-01   5.22e-01   -0.79  0.42991
V10A102      3.54e-01   4.41e-01    0.80  0.42203
V10A103     -6.81e-01   4.85e-01   -1.40  0.16025
V11          2.22e-02   9.91e-02    0.22  0.82266
V12A122      6.14e-01   2.95e-01    2.08  0.03718 *
V12A123      5.16e-01   2.75e-01    1.87  0.06107 .
V12A124      1.27e+00   5.00e-01    2.55  0.01092 *
V13         -1.37e-02   1.06e-02   -1.29  0.19786
V14A142      6.49e-02   4.61e-01    0.14  0.88805
V14A143     -7.03e-01   2.76e-01   -2.55  0.01092 *
V15A152     -5.21e-01   2.66e-01   -1.95  0.05072 .
V15A153     -1.17e+00   5.59e-01   -2.09  0.03648 *
V16          2.41e-01   2.21e-01    1.09  0.27563
```

```
V17A172      7.87e-01   9.03e-01    0.87  0.38356
V17A173      8.46e-01   8.78e-01    0.96  0.33533
V17A174      6.12e-01   8.83e-01    0.69  0.48796
V18          2.35e-01   2.79e-01    0.84  0.40021
V19A192     -2.94e-01   2.26e-01   -1.30  0.19425
V20A202     -1.81e+00   7.69e-01   -2.36  0.01825 *
---
Signif. codes:  0 □***□ 0.001 □**□ 0.01 □*□ 0.05 □.□ 0.1 □ □ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 980.75  on 799  degrees of freedom
Residual deviance: 698.95  on 751  degrees of freedom
AIC: 797

Number of Fisher Scoring iterations: 5
```

First we have to do one hot encoding on the various factors of select features A1,A3,A4,etc.

Hide

```
credDF <- oldCredDF
credDF$V1A12[credDF$V1 == "A12"] <-1
credDF$V1A12[credDF$V1 != "A12"] <- 0
credDF$V1A13[credDF$V1 == "A13"] <-1
credDF$V1A13[credDF$V1 != "A13"] <- 0
credDF$V1A14[credDF$V1 == "A14"] <-1
credDF$V1A14[credDF$V1 != "A14"] <- 0
credDF$V3A34[credDF$V3 == "A34"] <-1
credDF$V3A34[credDF$V3 != "A34"] <- 0
credDF$V4A41[credDF$V4 == "A41"] <-1
credDF$V4A41[credDF$V4 != "A41"] <- 0
credDF$V4A42[credDF$V4 == "A42"] <-1
credDF$V4A42[credDF$V4 != "A42"] <- 0
credDF$V4A43[credDF$V4 == "A43"] <-1
credDF$V4A43[credDF$V4 != "A43"] <- 0
credDF$V4A49[credDF$V4 == "A49"] <-1
credDF$V4A49[credDF$V4 != "A49"] <- 0
credDF$V6A65[credDF$V6 == "A65"] <-1
credDF$V6A65[credDF$V6 != "A65"] <- 0
credDF$V7A74[credDF$V7 == "A74"] <-1
credDF$V7A74[credDF$V7 != "A74"] <- 0
credDF$V9A93[credDF$V9 == "A93"] <-1
credDF$V9A93[credDF$V9 != "A93"] <- 0
credDF$V12A122[credDF$V12 == "A122"] <-1
credDF$V12A122[credDF$V12 != "A122"] <- 0
credDF$V12A124[credDF$V12 == "A124"] <-1
credDF$V12A124[credDF$V12 != "A124"] <- 0
credDF$V14A143[credDF$V14 == "A143"] <-1
credDF$V14A143[credDF$V14 != "A143"] <- 0
credDF$V15A153[credDF$V15 == "A153"] <-1
credDF$V15A153[credDF$V15 != "A153"] <- 0
credDF$V20A202[credDF$V20 == "A202"] <-1
credDF$V20A202[credDF$V20 != "A202"] <- 0
```

Next, we'll split it into training and validation and run glm on the set of features that summary of the previous model determined to be the most optimal. We'll determine the cost of each model in part b.

Hide

```
selectSub <- sample(1:nrow(credDF), size=round(0.8*(nrow(credDF))))
selectTrain <- credDF[selectSub,]
selectVal <- credDF[-selectSub,]
selectModel=glm(V21~V1A12 + V1A13 + V1A14 + V3A34 + V4A41 + V4A42 + V4A43 + V4A49 + V5 + V6A65 +
 V7A74 + V8 + V9A93 + V12A122 + V12A124 + V14A143 + V15A153 + V20A202, family=binomial(link="log
it"), data=selectTrain)
selectY_hat <- predict(selectModel, selectVal, type="response")
selectY_pred <- as.integer(selectY_hat >0.5)
table(selectY_pred, selectVal$V21)
```

```
selectY_pred   0   1
           0 129  33
           1  12  26
```

Question 4 Part 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.

Hide

```
library(pROC)
crimeRoc <- roc(credVal$V21,y_pred)
```

Assessing cost

Now we can check our two models to see which one costs more. Since it costs 5 times for someone we predict as 1 to be 0 as someone we predict 0 to be 1, we can use the formula below to calculate this cost.

Hide

```
table <- as.matrix(table(y_pred,credVal$V21))
cost <- table[2,1] + 5*table[1,2]
print(table)
```

```
y_pred   0   1
     0 259  67
     1  24  50
```

Hide

```
print(cost)
```

```
[1] 359
```

Hide

```
selectTable <- as.matrix(table(selectY_pred, selectVal$V21))
selectCost <- selectTable[2,1] + 5*selectTable[1,2]
print(selectTable)
```

```
selectY_pred   0   1
           0 129  33
           1  12  26
```

Hide

```
print(selectCost)
```

```
[1] 177
```

From here, we can see that after choosing to use certain features, we cut down the cost by nearly 50%.