HW7

Question 10.1

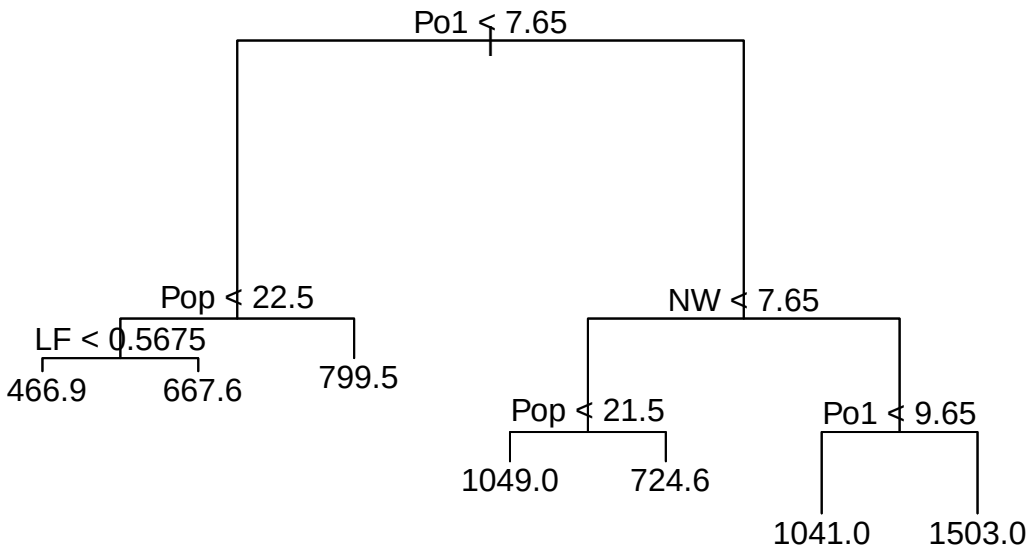
Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model.
In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

```
#Import the data and set seed
library(tree)
set.seed(1)
data<- read.table('uscrime.txt',header=T)

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

```
##
## 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
```

```
#Visualize the regression tree
plot(model)
text(model)
```




```
# Calculate the R^2 and adjusted R^2 of this tree model
sst<-sum((data$Crime-mean(data$Crime))^2)
sse_tree<- sum(((as.data.frame(predict(model))[,1]-data$Crime)^2))
r2<-1-sse_tree/sst
r2_ad<-1-(1-r2)*(nrow(data)-1)/(nrow(data)-15-1)
r2
```

```
## [1] 0.7244962
```

```
#R squared is 0.786 which we suspects the model is overfitting the training data
r2_ad
```

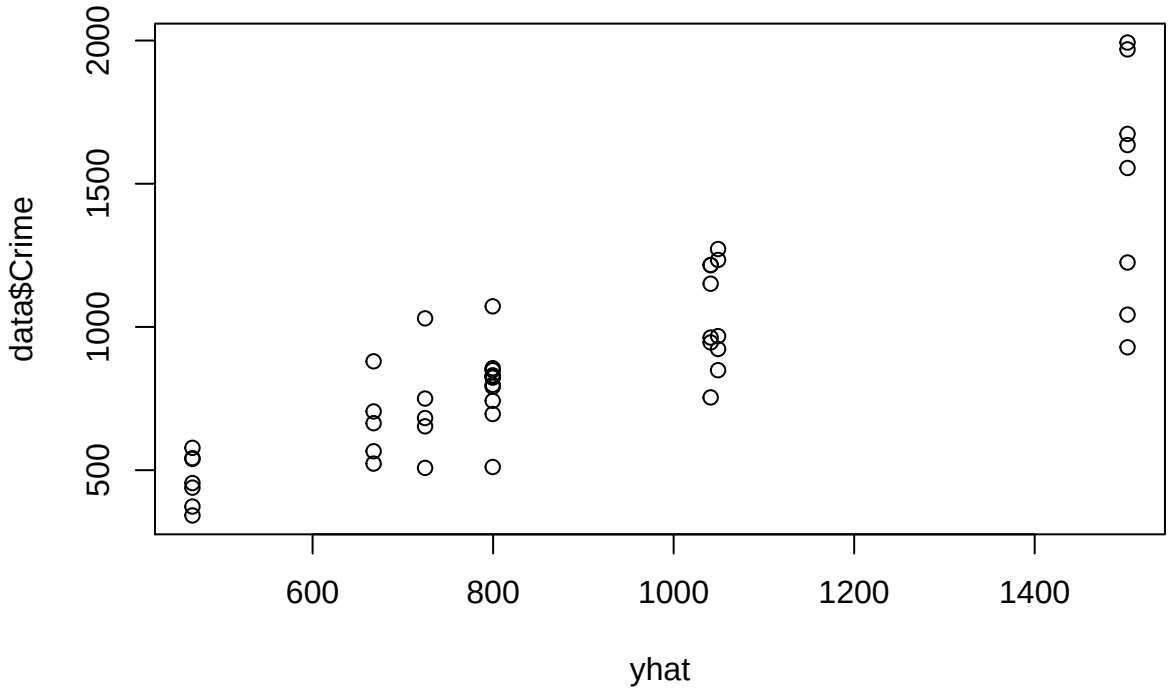
```
## [1] 0.5911879
```

```
#Adjusted R squared is less than r2 since we reduced the increase attributing to multiple predictors
sse_tree
```

```
## [1] 1895722
```

The manually calculated sse is the same as the total deviance indicated in the model as expected. Going forward, we could take the Total deviance as sse when calculating the R^2

```
#Check the predict value of the tree model
yhat<-predict(model)
plot(yhat,data$Crime)
```




```
#Examining training and cv deviance for different tree sizes
prune.tree(model)$size
```

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

```
prune.tree(model)$dev
```

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

```
cv.tree(model)$dev
```

```
## [1] 7870297 7978845 8281875 8304665 8913771 7767232 8249636
```

#From the below output, it indicates that the deviance gets excessively larger when using cross validation. It means that our model is fitting to its training data better and there might be some overfittings. As the branches decrease, the deviance increases since our data groups together.

```
#Manually prune the tree with different sizes
tree_prune6<-prune.tree(model,best=6) # pruning 1 branches and 6 branches remaining
summary(tree_prune6)  #Residual deviance: 2010000
```

```
##
## Regression tree:
## snip.tree(tree = model, nodes = 4L)
## Variables actually used in tree construction:
```

```
## [1] "Po1" "Pop" "NW"
## Number of terminal nodes:  6
## Residual mean deviance:  49100 = 2013000 / 41
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -99.520   -1.545    0.000  122.800  490.100
```

```
tree_prune5<-prune.tree(tree_prune6,best=5)# pruning 1 more branches and 5 branches remaining
summary(tree_prune5)  #Residual deviance:2280000
```

```
##
## Regression tree:
## snip.tree(tree = tree_prune6, nodes = 6L)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "NW"
## Number of terminal nodes:  5
## Residual mean deviance:  54210 = 2277000 / 42
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -573.9  -107.5    15.5     0.0   122.8   490.1
```

```
tree_prune4<-prune.tree(tree_prune5,best=4)# pruning 1 more branches and 4 branches remaining
summary(tree_prune4)  #Residual deviance: 2630000
```

```
##
## Regression tree:
## snip.tree(tree = tree_prune5, nodes = 2L)
## Variables actually used in tree construction:
## [1] "Po1" "NW"
## Number of terminal nodes:  4
## Residual mean deviance:  61220 = 2633000 / 43
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -573.90 -152.60   35.39    0.00  158.90  490.10
```

```
tree_prune3<-prune.tree(tree_prune4,best=3)# pruning 1 more branches and 3 branches remaining
summary(tree_prune3)  #Residual deviance: 3360000
```

```
##
## Regression tree:
## snip.tree(tree = tree_prune4, nodes = 7L)
## Variables actually used in tree construction:
## [1] "Po1" "NW"
## Number of terminal nodes:  3
## Residual mean deviance:  76460 = 3364000 / 44
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -550.9  -181.8   -37.9     0.0   158.9   688.1
```

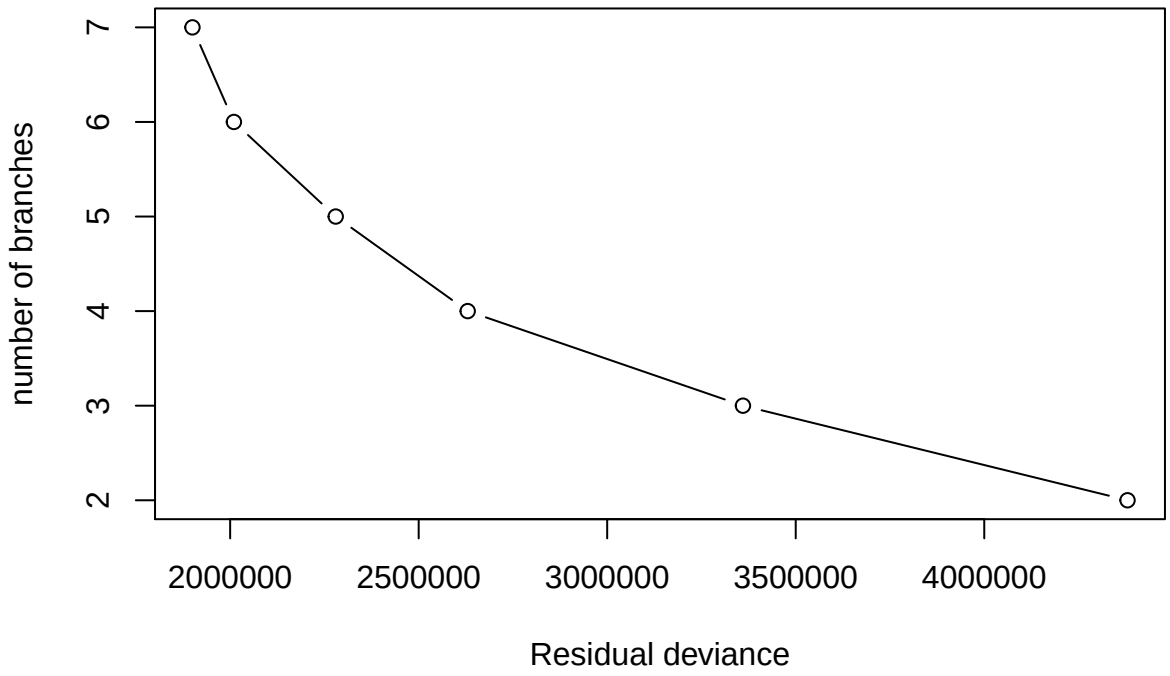
```
tree_prune2<-prune.tree(tree_prune3,best=2)# pruning 1 more branches and 2 branches remaining
summary(tree_prune2)  #Residual deviance: 4380000
```

```
##
```

```
## Regression tree:
## snip.tree(tree = tree_prune3, nodes = 3L)
## Variables actually used in tree construction:
## [1] "Po1"
## Number of terminal nodes:  2
## Residual mean deviance:  97410 = 4383000 / 45
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -622.800 -193.200   -5.609    0.000  147.300  862.200
```

Plot the change between number of branches and residual deviance, our goal is to maintain a low level residual deviance while pruning useless branch.

```
#Plot the change between number of branches and residual deviance,
plot(c(1900000,2010000,2280000,2630000,3360000,4380000),c(7,6,5,4,3,2),ylab='number of branches',xlab='Residual
```




```
cv.tree(tree_prune6)$dev
```

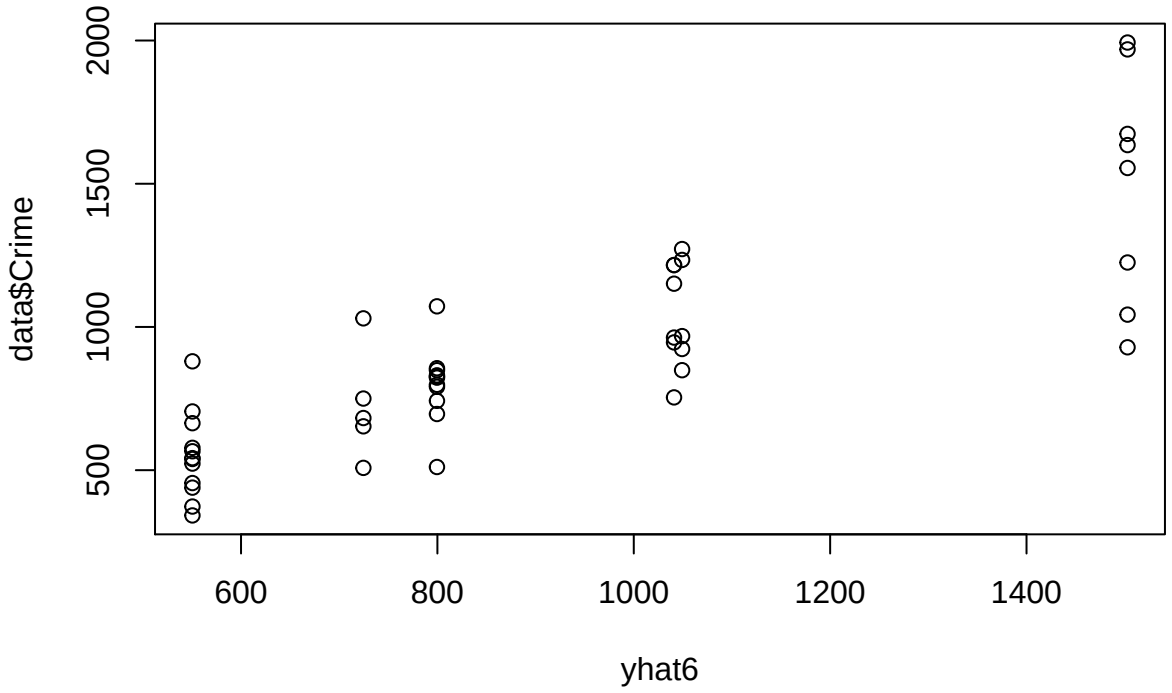
```
## [1] 8114496 8369027 8554913 8458678 8396658 9298822
```

```
#CV.tree output: 6306818 6010672 6068094 5967726 5977136 7653001.
#Comparing with the output from previous cv.tree output, dev decreases, which is great.
```

From the graph below, we can tell the residual deviance is gradually increasing with lower number of branch. When the number of branches equals to 6, it seems that this point both maintains a low level deviance and prunes a branch to avoid overfitting.

In conclusion,i decide to go with 6 branches.

```r
#Choose one branch to build linear regression
yhat6<-predict(tree_prune6)
plot(yhat6,data$Crime)
```



From the plot, the far right branch didnt predict well since there was a larger variance comparing with other branches. So investigating this branch and tuning the parameters to make it better fit is necessary.

As we picked far right branch(11th in row number), data points in row number 2,4,8,11,18,20,26,29 will be selected to form a linear regression.

```r
data_form<-rbind(data[2,],data[4,],data[8,],data[11,],data[18,],data[20,],data[26,],data[29,])
form_lm<-lm(Crime~.,as.data.frame(data_form))
summary(form_lm)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = as.data.frame(data_form))
##
## Residuals:
## ALL 8 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (8 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  28048.5         NA      NA       NA
## M              487.5         NA      NA       NA
## So             874.7         NA      NA       NA
## Ed             916.5         NA      NA       NA
## Po1           3535.8         NA      NA       NA
```

```
## Po2          -3769.8         NA      NA       NA
## LF           52544.0         NA      NA       NA
## M.F           -740.9         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
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 7 and 0 DF,  p-value: NA
```

The summary shows that since we don't have enough data, the number of predictors is larger than the size of data. So the no residual degrees of freedom left to form a linear regression. In this case, we need to cut few predictors and make them less than or equal to 8, since we only have 8 data points.

```
form_lm2<-lm(Crime~+M+Ed+Po1+U2+Ineq+Prob,as.data.frame(data_form))
summary(form_lm2)
```

```
##
## Call:
## lm(formula = Crime ~ +M + Ed + Po1 + U2 + Ineq + Prob, data = as.data.frame(data_form))
##
## Residuals:
##       2       4       8      11      18      20      26      29
## -27.395  42.644   3.776   9.444  10.250   2.092 -28.046 -12.765
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3400.17    1048.70   3.242   0.1905
## M            -679.68     106.91  -6.357   0.0993 .
## Ed           1283.37     125.52  10.224   0.0621 .
## Po1          -293.64      37.81  -7.767   0.0815 .
## U2           -456.26      58.44  -7.807   0.0811 .
## Ineq          -96.93      25.54  -3.795   0.1640
## Prob          915.22    1218.46   0.751   0.5899
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 61.09 on 1 degrees of freedom
## Multiple R-squared:  0.9967, Adjusted R-squared:  0.9768
## F-statistic: 50.08 on 6 and 1 DF,  p-value: 0.1077
```

After adjusting the predictors for linear regression, we got the R2 adjusted which equals to 0.977. I suspect this number is too good to be true due to the limited size of data.

Random Forest Model

```
library(randomForest)
```

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

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

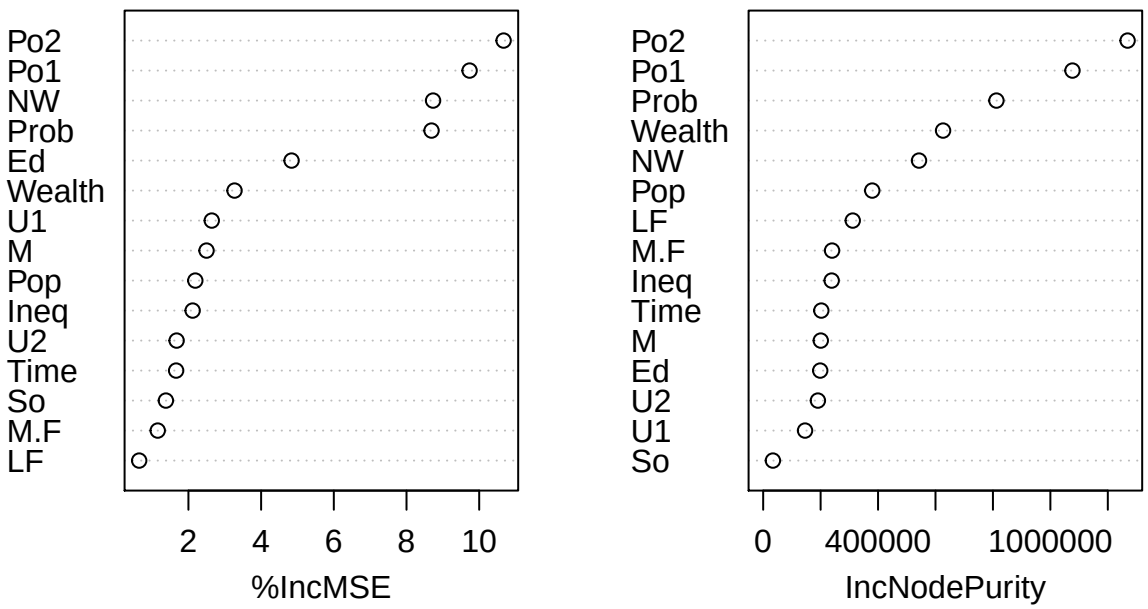
```
set.seed(42)
rm(list=ls())
uscrime<-read.table('uscrime.txt',header=T)
num_pred<- 4
uscrime_rf<-randomForest(Crime~.,data=uscrime,mtry=num_pred,importance=T,ntree=500)
```

Note:we can use the importance function to see what the most important factors for prediction are. %IncMSE is the amount that the MSE of predictions increases if the variable is randomly chosen instead of using its actual value. IncNodePurity measures how much splitting on it improves the similarity of the data points in each leaf.

```
#Check the importance of variables
importance(uscrime_rf)
```

```
##           %IncMSE IncNodePurity
## M       2.4984854     200566.40
## So      1.3802135      33881.59
## Ed      4.8378328     198601.72
## Po1     9.7354718    1076933.25
## Po2    10.6715396    1268930.03
## LF      0.6449124     311872.13
## M.F     1.1555044     239897.22
## Pop     2.1893155     379760.15
## NW      8.7310286     542658.76
## U1      2.6422460     145760.60
## U2      1.6754487     190587.49
## Wealth  3.2683848     626353.30
## Ineq    2.1162044     238557.90
## Prob    8.6884908     812217.29
## Time    1.6622726     202467.06
```

```
#Plot the importance
varImpPlot(uscrime_rf)
```



Higher the %IncMSE and IncNodePurity, higher the importance of the variable in the model, In the plot shown above, Po2 is the most important variable.

In the above model, we assume the mtry=4 is the optimal value guided by rule of thumb. We need to investigate if it prevails with different mtry values.

```
rsq_rf<-vector(length=10)
for (i in 1:10){
  rf_mtry<-randomForest(Crime~.,data=uscrime,mtry=i,mtree=500,importance=T)
  rsq_rf[i]<-rf_mtry$rsq[length(rf_mtry$rsq)]
}
rsq_rf
```

```
##  [1] 0.3580472 0.4062374 0.4359149 0.4271885 0.3817764 0.3873319 0.3968311
##  [8] 0.3922331 0.3842341 0.3606838
```

when mtry=3, the model gives the highest rsq : 43.2%.

```
#Use cross validation to compare R2s
rf.cv<-rfcv(uscrime[,1:15],uscrime$Crime,step=0.8,cv.fold = 5)
rf.cv$error.cv
```

```
##        15        12        10         8         6         5         4         3
##  80513.03  83404.16  78190.68  82467.22  84896.08  92318.96  86582.69  93519.14
##         2         1
## 117281.02 133637.05
```

```
uscrime_rf
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime, mtry = num_pred,      importance = T, ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 83636.77
##                     % Var explained: 42.87
```

MSE(from rf.cv) = 83987 MSE(from random forest)=83637 Comparing these two values, there isn't much differences which indicates that random forest gets an relative accurate result and we dont need to build another cross validation model on it.

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.

Logistic regression model could be used to predict if the city lockdown is necessary due to covid-19. (The response- Yes:1, No: 0) Using logistic model could effectively define the correlation between city lockdown and different related variables.In order to build the model, we need the sample data from most of the cities all over the world that are hugely impacted by covid-19 and their responses(lockdown or not) In the regression model, variables such as the number of new test per 1 Million people, number of confirmed cases per 1 million people etc. Predictors including: 1. Number of new confirmed cases that related to community spread. 2. Number of hospitals per 1 million people 3. Number of confirmed cases per 1 million people 4. Number of super spreaders(Refers to someone who is capable of spreading the virus to at least 25 people) 5. Direct economic loss in 4 weeks after the lockdown

10.3.1

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.

```
rm(list=ls())
gecredit<-read.table('germancredit.txt',sep="")
head(gecredit)
```

```
##    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   1
## 2 A191 A201   2
## 3 A191 A201   1
```

```
## 4 A191 A201   1
## 5 A191 A201   2
## 6 A192 A201   1
```

```
#Convert the response variable in terms of 0 and 1
gecredit$V21[gecredit$V21==1]<-0
gecredit$V21[gecredit$V21==2]<-1
```

```
#Split the data into training (80%) and testing(20%) sets.
gecredit_train<-gecredit[1:800,]
gecredit_test<-gecredit[801:nrow(gecredit),]
```

```
#Create a logistic regression model
gecred_model<-glm(V21~.,family=binomial(link="logit"),data=gecredit_train)
summary(gecred_model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = gecredit_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7373  -0.6979  -0.3604   0.6663   2.5591
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.978e-01  1.249e+00   0.318 0.750139
## V1A12       -2.701e-01  2.437e-01  -1.108 0.267784
## V1A13       -9.306e-01  4.018e-01  -2.316 0.020567 *
## V1A14       -1.737e+00  2.686e-01  -6.465 1.02e-10 ***
## V2           2.893e-02  1.042e-02   2.777 0.005479 **
## V3A31        2.255e-01  6.289e-01   0.359 0.719936
## V3A32       -7.639e-01  4.753e-01  -1.607 0.108022
## V3A33       -9.172e-01  5.233e-01  -1.753 0.079627 .
## V3A34       -1.487e+00  4.907e-01  -3.031 0.002440 **
## V4A41       -1.832e+00  4.425e-01  -4.141 3.46e-05 ***
## V4A410      -1.413e+00  8.263e-01  -1.710 0.087326 .
## V4A42       -9.368e-01  2.990e-01  -3.134 0.001727 **
## V4A43       -9.044e-01  2.799e-01  -3.230 0.001236 **
## V4A44       -8.312e-01  8.946e-01  -0.929 0.352807
## V4A45       -3.222e-01  6.092e-01  -0.529 0.596843
## V4A46        1.688e-02  4.255e-01   0.040 0.968354
## V4A48       -2.213e+00  1.219e+00  -1.816 0.069365 .
## V4A49       -8.368e-01  3.850e-01  -2.173 0.029760 *
## V5           1.138e-04  5.166e-05   2.202 0.027682 *
## V6A62       -3.991e-01  3.182e-01  -1.254 0.209771
## V6A63       -4.615e-01  4.762e-01  -0.969 0.332404
## V6A64       -1.222e+00  5.473e-01  -2.232 0.025592 *
## V6A65       -7.093e-01  2.929e-01  -2.421 0.015462 *
## V7A72       -2.017e-01  4.948e-01  -0.408 0.683485
## V7A73       -3.028e-01  4.706e-01  -0.643 0.519975
## V7A74       -1.105e+00  5.113e-01  -2.162 0.030623 *
## V7A75       -4.092e-01  4.712e-01  -0.869 0.385102
## V8           3.602e-01  9.933e-02   3.626 0.000287 ***
## V9A92       -4.434e-01  4.300e-01  -1.031 0.302374
```

```
## V9A93       -1.230e+00  4.245e-01  -2.897 0.003769 **
## V9A94       -4.630e-01  5.119e-01  -0.905 0.365705
## V10A102      7.521e-01  4.771e-01   1.576 0.114917
## V10A103     -9.329e-01  4.830e-01  -1.931 0.053423 .
## V11          3.282e-03  9.850e-02   0.033 0.973420
## V12A122      4.101e-01  2.897e-01   1.415 0.156969
## V12A123      1.536e-01  2.649e-01   0.580 0.562115
## V12A124      7.122e-01  4.714e-01   1.511 0.130827
## V13         -1.868e-02  1.055e-02  -1.770 0.076682 .
## V14A142     -1.442e-02  4.733e-01  -0.030 0.975695
## V14A143     -4.354e-01  2.724e-01  -1.599 0.109919
## V15A152     -3.967e-01  2.739e-01  -1.448 0.147576
## V15A153     -5.576e-01  5.303e-01  -1.051 0.293071
## V16          3.297e-01  2.124e-01   1.552 0.120602
## V17A172      5.151e-01  7.807e-01   0.660 0.509351
## V17A173      5.655e-01  7.507e-01   0.753 0.451267
## V17A174      8.202e-01  7.597e-01   1.080 0.280307
## V18          5.065e-01  2.854e-01   1.775 0.075972 .
## V19A192     -3.739e-01  2.323e-01  -1.610 0.107489
## V20A202     -1.498e+00  8.079e-01  -1.854 0.063779 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 975.68  on 799  degrees of freedom
## Residual deviance: 705.07  on 751  degrees of freedom
## AIC: 803.07
##
## Number of Fisher Scoring iterations: 5
```

```
yhat<-predict(gecred_model,gecredit_test,type="response")
```

```
#Calculate the Pseudo R^2
ll.null<- gecred_model$null.deviance/-2
ll.proposed<-gecred_model$deviance/-2
(ll.null-ll.proposed) /ll.null
```

```
## [1] 0.2773544
```

After doing the math and we end up with a Pseudo R^2=0.277, This R^2 along with the AIC value(803.1) can be interpreted as the overall effect size.

Also, by calculating ROC below, it gives 0.67

10.3.2

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.

```
library(pROC)
```

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

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

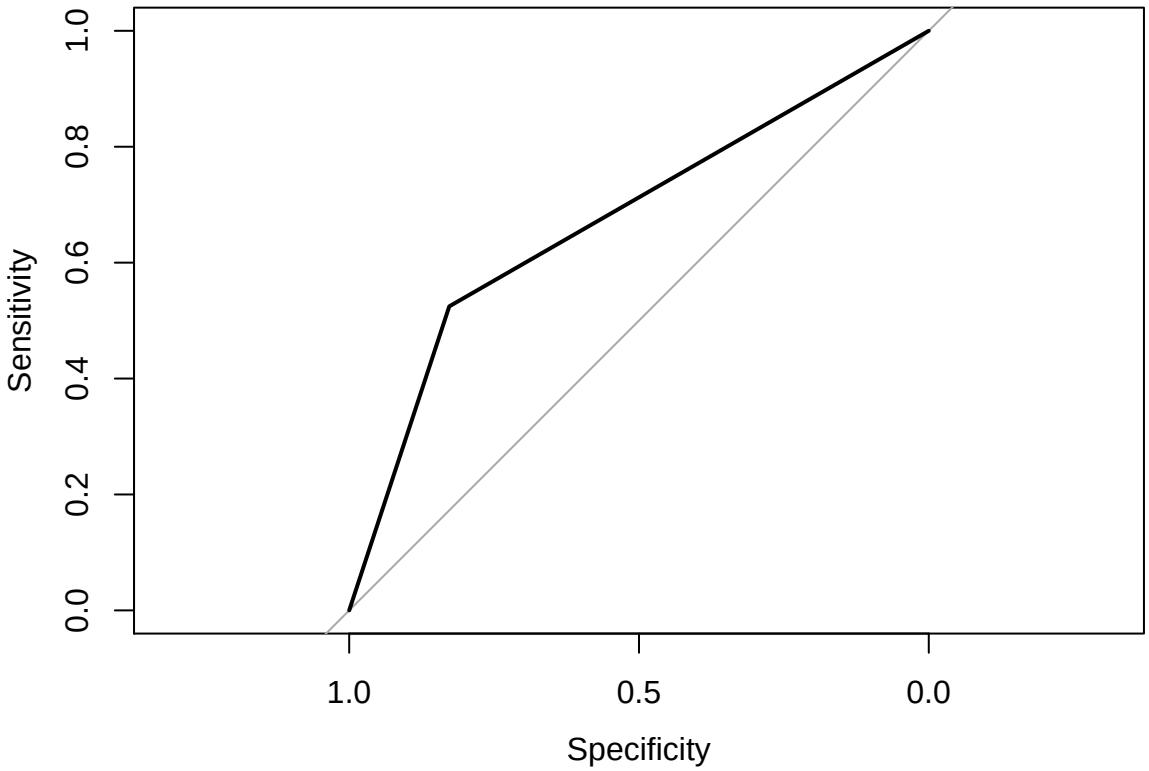
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
myroc<-roc(gecredit_test$V21,round(yhat))
```

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

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

```
plot(myroc)
```



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. we can interpret this assumption in this way: cost of FALSE POSITIVE = 5* cost of FALSE NEGATIVE Assume cost of FALSE NEGATIVE= 1 point and cost of FALSE POSITIVE=5 points

We create a penalty score(PS) based on each threshold's performance of FALSE POSITIVE and FALSE NEGATIVE.

Now, we try out different thresholds.

```
thresh<-0.5
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 115  29
##           1  24  32
```

```
#PS1
PS=24*5+29*1
PS
```

```
## [1] 149
```

```
thresh<-0.8
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 134  53
##           1   5   8
```

```
#PS2
PS=5*5+53*1
PS
```

```
## [1] 78
```

```
thresh<-0.9
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 137  58
##           1   2   3
```

```
#PS3
PS=2*5+58*1
PS
```

```
## [1] 68
```

```
thresh<-0.6
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 124  35
##           1  15  26
```

```r
#PS4
PS=15*5+35*1
PS
```

```
## [1] 110
```

```r
thresh<-0.7
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 132  43
##           1   7  18
```

```r
#PS5
PS=7*5+43*1
PS
```

```
## [1] 78
```

```r
thresh<-1
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 139  61
```

```r
#PS6
PS=0*5+61*1
PS
```

```
## [1] 61
```

```r
thresh<-0.95
yhat_thresh<-as.integer(yhat>thresh)
conf_matrix<-as.matrix(table(yhat_thresh,gecredit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 139  60
##           1   0   1
```

```r
#PS7
PS=0*5+60*1
PS
```

```
## [1] 60
```

In conclusion, comparing these 7 threshold, we acknowledged that we need to set the bar higher to avoid False Positive. The optimal threshold is 0.95.