Homework 4

6/10/2020

Question 9.1

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few 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 Question 8.2. 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. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

For reference, my model in 8.2 used 8 predictors (M, Ed, Po1, M.F, U1, U2, Ineq and Prob) and had an adj. r squared value of 0.7444. I did not split the data into a test and training set so the model was definitely overfit. It predicted that the city had a crime rate of 1,038. To make the comparison fair, I won't split my data into a test and training set for this question.

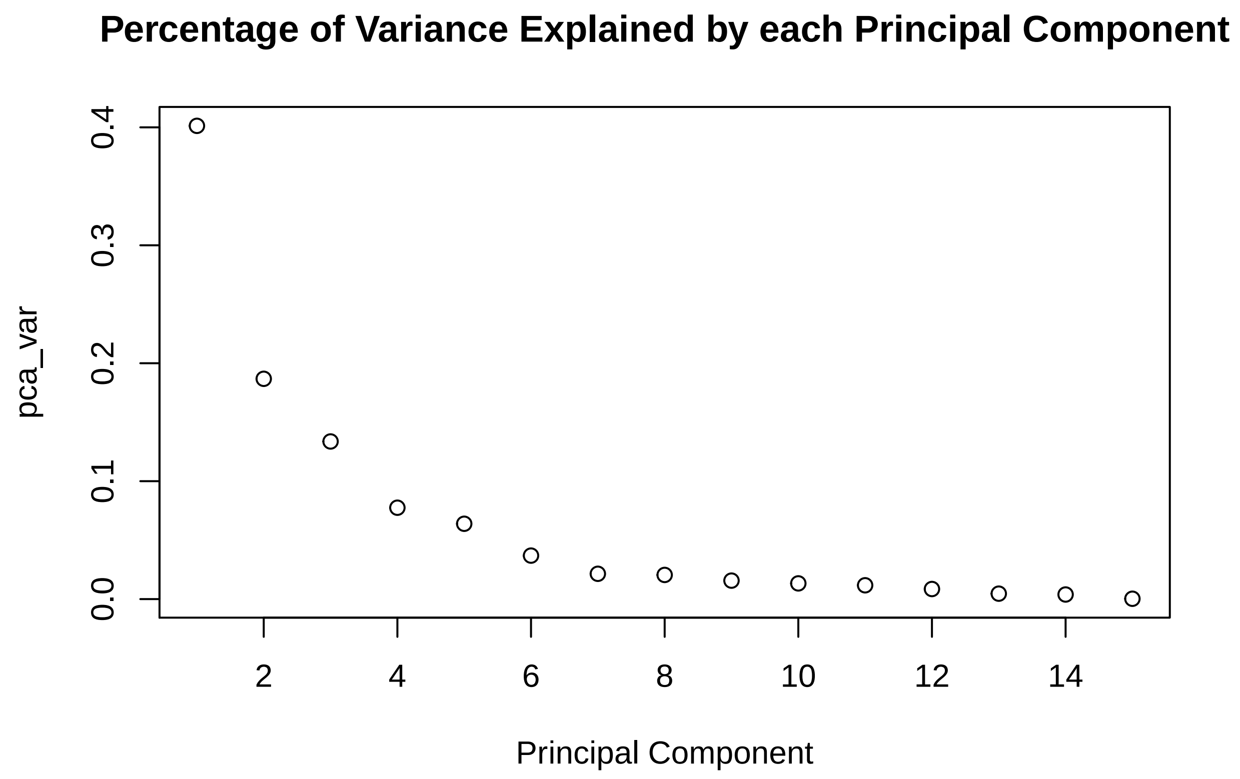
```
uscrime <- read.delim('uscrime.txt')
new_city <- data.frame(M = 14.0, So = 0, Ed = 10.0, Po1 = 12.0, Po2 = 15.5, LF = 0.640, M.F = 94.0, Pop
```

Applying PCA and plotting each component's variance:

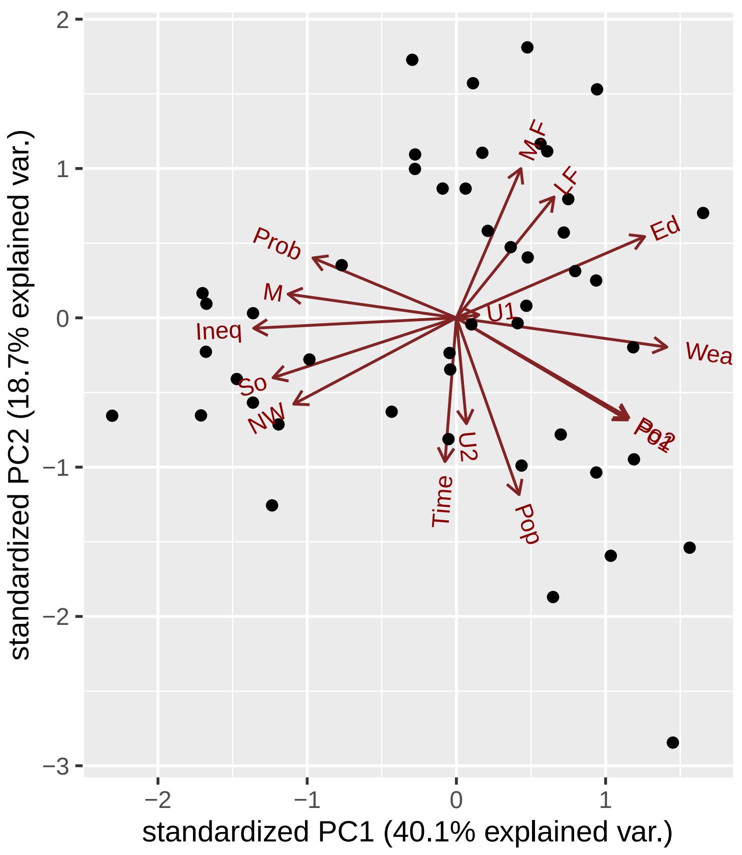
```
uscrime_pca <- prcomp(uscrime[,1:15], center = TRUE, scale. = TRUE )
summary(uscrime_pca)
```

```
## 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
```

```
pca_var = uscrime_pca$sdev^2 / sum(uscrime_pca$sdev^2)
plot(pca_var, xlab = "Principal Component",
     main = "Percentage of Variance Explained by each Principal Component")
```



```
ggbiplot(uscrime_pca)
```



I will use the first 8 principal components so that 95% of variance is retained. I calculated k in Excel using the method outlined in this video, @ 7:50: https://www.coursera.org/lecture/machine-learning/choosing-the-number-of-principal-components-S1bq1

```r
features_pca <- uscrime_pca$x
pca_data_model <- cbind(uscrime_pca$x[,1:8], uscrime[,16])
colnames(pca_data_model)[9]<-"Crime"
pca_model <- lm(Crime ~., data = as.data.frame(pca_data_model))
summary(pca_model)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = as.data.frame(pca_data_model))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -480.36 -130.30   31.42  132.73  395.46
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      34.57  26.184  < 2e-16 ***
## PC1            65.22      14.24   4.579 4.90e-05 ***
## PC2           -70.08      20.87  -3.357   0.0018 **
## PC3            25.19      24.68   1.021   0.3137
## PC4            69.45      32.41   2.143   0.0386 *
## PC5          -229.04      35.69  -6.417 1.53e-07 ***
## PC6           -60.21      46.98  -1.282   0.2077
## PC7           117.26      61.59   1.904   0.0645 .
## PC8            28.72      63.02   0.456   0.6512
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 237 on 38 degrees of freedom
## Multiple R-squared:  0.6899, Adjusted R-squared:  0.6246
## F-statistic: 10.57 on 8 and 38 DF,  p-value: 1.182e-07
```

Using principal components 1 to 8, the adj. r squared value is 0.6058 (less than the adj. r squared value of my model from Question 8.2 of 0.7444.)

Before I can predict crimes for the new city using this PCA model, I have to unscale the data.

```
b0 = pca_model$coefficients[1]
b_vals <- pca_model$coefficients[2:9]
a <-  uscrime_pca$rotation[,1:8] %*% b_vals


mu <- sapply((uscrime[1:15]), mean)
std_dev <-sapply(uscrime[,1:15],sd)
a_unscaled = a/std_dev
b0_unscaled = b0 - sum(a*mu/std_dev)

a_unscaled
```

```
##                  [,1]
## M       4.258605e+01
## So      1.533872e+02
## Ed     -1.056306e+01
## Po1     4.503214e+01
## Po2     4.674975e+01
## LF      8.546813e+02
## M.F     4.386624e+01
## Pop     5.142890e-01
## NW      6.255558e+00
## U1     -1.349584e+03
## U2      3.270216e+01
## Wealth  3.140421e-02
## Ineq    1.296823e+01
## Prob   -5.279172e+03
## Time   -1.424626e+00
```

```
b0_unscaled
```

```
## (Intercept)
##   -5369.778
```

Now that we have our unscaled coefficients and intercept, I can predict the crimes in the new city:

```
crimes_estimate = as.matrix(new_city)%*%a_unscaled + b0_unscaled
crimes_estimate
```

```
##            [,1]
## [1,] 1190.455
```

Using PCA, the predicted number of crimes for the new city is 1,190 (higher than my previous estimate of 1,038). The formula for regression using 8 principal components is:

Crime = -5369.778 + 42.58605M + 153.4So - 10.6Ed + 45.0Po1 + 46.7Po2 + 854.7LF + 43.9M.F + 0.51Pop + 6.3NW - 1349.6U1 + 32.7U2 + 0.0314Wealth + 13Ineq - 5279.2Prob - 1.4Time

Question 10.1

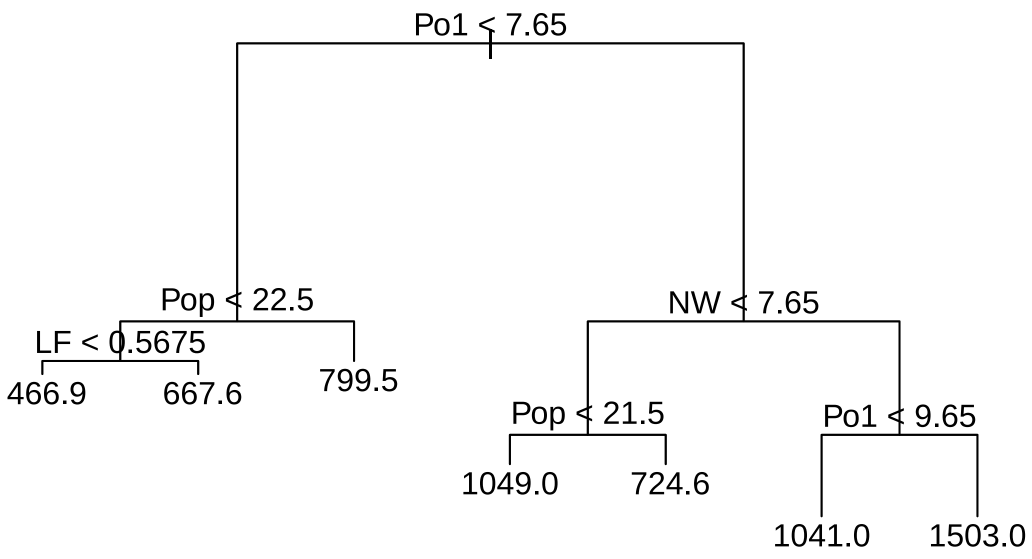
Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package,and the randomForest package. For each model, describe one or two qualitative takeaways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

Using the tree package to generate the tree:

```
set.seed(8010)
tree_model <- tree(Crime~ ., data = uscrime)
summary(tree_model)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = uscrime)
## 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
```

```
plot(tree_model)
text(tree_model)
```



Tree has 7 terminal nodes and only uses 4 out of 15 predictors to split the data. It's interesting to note that Po1 is used to split the data at the top and at the bottom right of the tree.

Let's do some pruning using cross validation to determine what the deviance is. Ideally, we want to use the number of nodes that give us the lowest amount of deviance:

```r
prune.tree(tree_model)$size
```

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

```r
prune.tree(tree_model)$dev
```

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

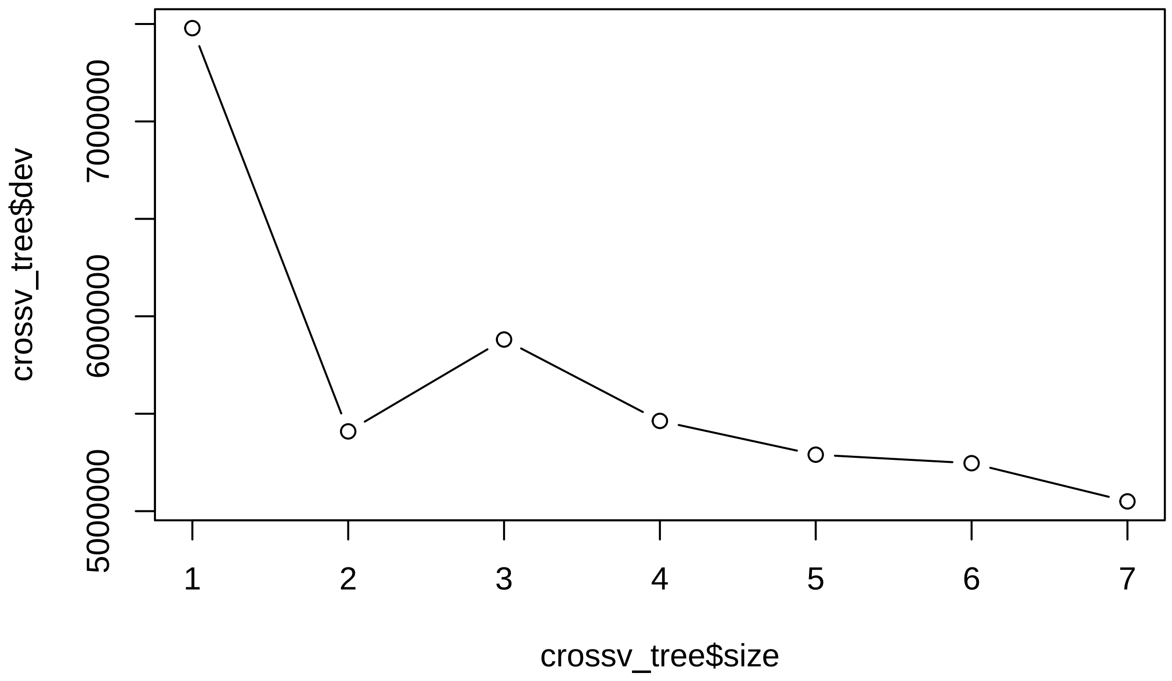
```r
set.seed(3)
crossv_tree <- cv.tree(tree_model, FUN = prune.tree)
crossv_tree$size
```

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

```r
crossv_tree$dev
```

```
## [1] 5050130 5245991 5289744 5463080 5880685 5409221 7478665
```

```r
plot(crossv_tree$size, crossv_tree$dev, type = "b")
```



In cross validation, the deviance is significantly higher than the pruned tree at all tree sizes. This is an indication that overfitting exists in the model.

Based on these results, having 7 terminal nodes results in the lowest deviation. We can use our original tree model to compare against the random forest.

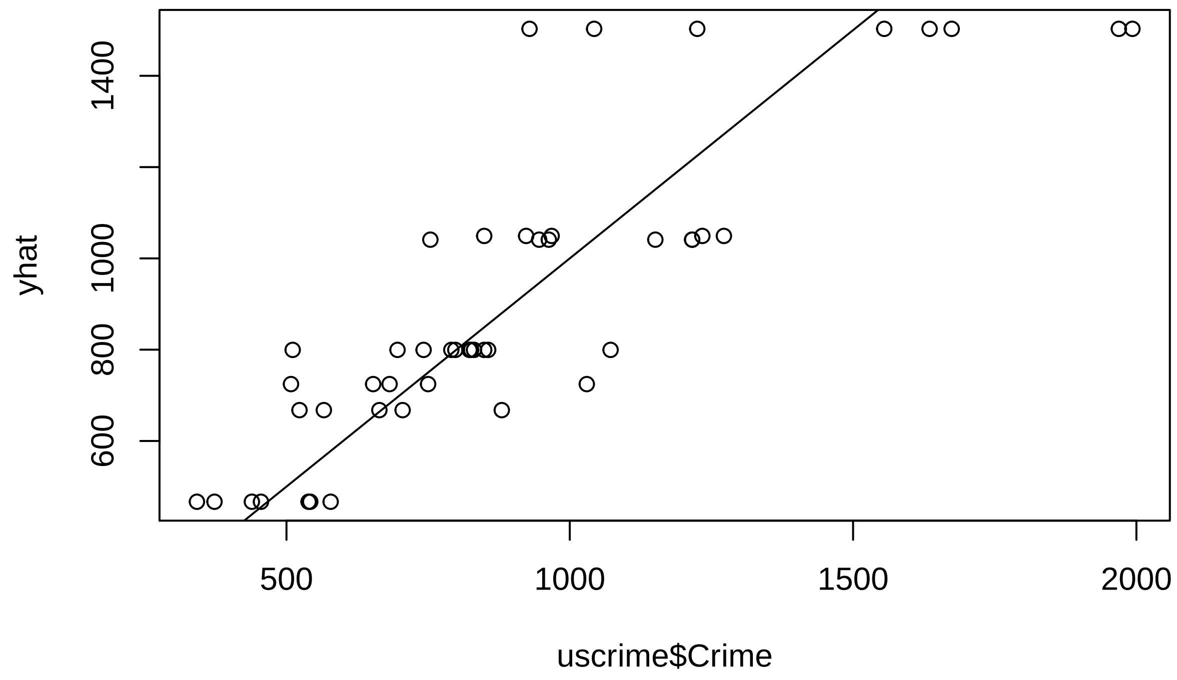
Calculating r squared for the tree:

```r
yhat=predict(tree_model)
y = uscrime[,16]
ss_res = sum((yhat - y)^2)
ss_total <- sum((y- mean(y))^2)
R2 = 1 - ss_res/ss_total
R2
```

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

Plotting actuals vs predicted:

```r
plot(uscrime$Crime, yhat)
abline(0,1)
```



Random Forest

We'll use 1+log(15) = 4 predictors for the number of randomly sampled variables at each split.

```
set.seed(4624)
rand_forest <- randomForest(Crime ~ . , data = uscrime, mtry = 4, importance = TRUE, ntree = 500)
rand_forest
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime, mtry = 4, importance = TRUE,      ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 82966.53
##                     % Var explained: 43.33
```

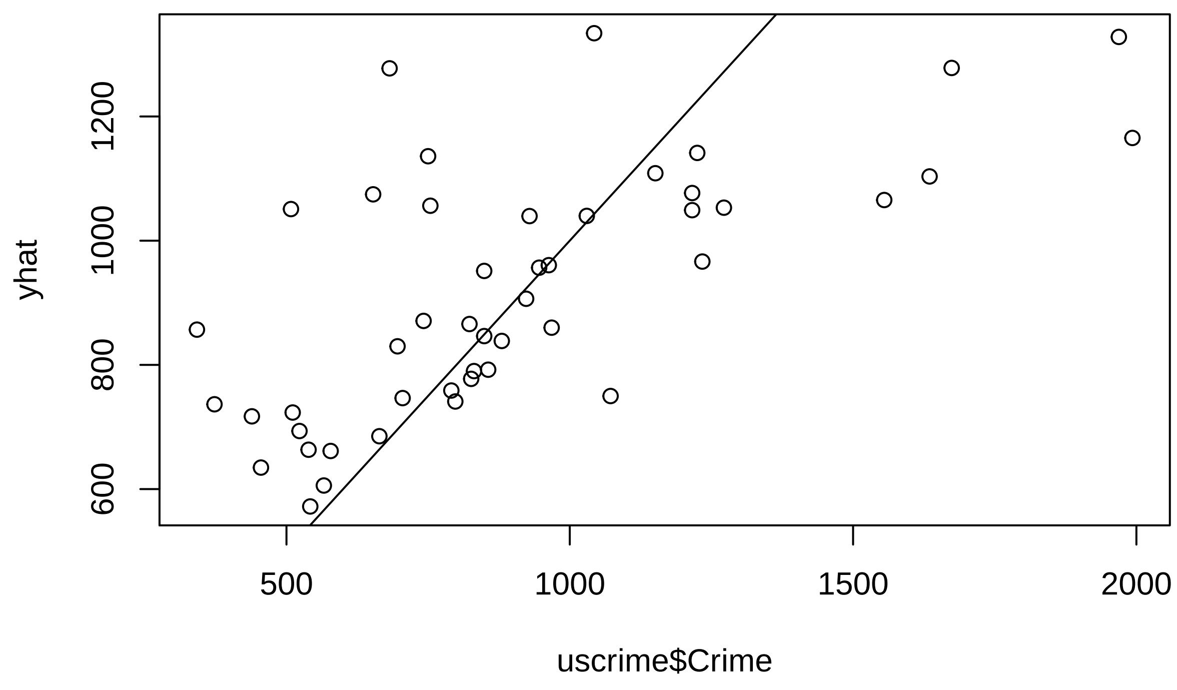
Calculating R squared for the random forest:

```
yhat <- predict(rand_forest)
y = uscrime[,16]
ss_res = sum((yhat - y)^2)
ss_total <- sum((y- mean(y))^2)
R2 = 1 - ss_res/ss_total
R2
```

```
## [1] 0.4332992
```

This is lower than the r squared value generated by the tree. This is a good indication that the random forest has taken care of the overfitting.

```r
plot(uscrime$Crime, yhat)
abline(0,1)
```



Let's look at the importance of each predictor in the random forest model:

```r
importance(rand_forest)
```

```
##             %IncMSE IncNodePurity
## M       4.209762638     154810.43
## So      1.859295426      21602.23
## Ed      4.611293467     367491.19
## Po1    11.101468690    1116949.58
## Po2    10.529959411    1106287.55
## LF     -0.006871027     278537.38
## M.F     1.219614469     291058.82
## Pop     1.306014984     334533.73
## NW      7.634567608     588221.42
## U1      2.177481963     166920.70
## U2      1.461603361     184877.83
## Wealth  3.309282222     616347.78
## Ineq    1.715679075     246245.16
## Prob    8.337082224     719195.83
## Time    4.004826238     240533.61
```

Based on the %IncMSE, it looks like Po1 is the most important predictor, closely followed by Po2. It's interesting that Po2 didn't show up as node in the tree diagram. The %IncMSE says that when Po1 is randomly chosen instead of using its actual value, the MSE increases by 11.1%. Similarily, the IncNodePurity for Po1 and Po2 are the highest among the predictors. This suggests that by spliting on these, we improve the similiarity of the data points in each leaf more than if we split on other predictors.

Question 10.2

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

I work for a loyalty program that gives members points for spending at our participating retailers. Members can get even more points by paying with a co-branded credit card we have with a bank. I would use a logistic regression model to predict the probability of a member acquiring the credit card.

Predictors I would use:

- Age
- Household Income
- Spend at retailers in the last 12 months
- Number of points earned in the last 12 months

Question 10.3.1

Using the GermanCredit data set germancredit.txt, 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.

For this question, I won't show all the output as it makes the homework quite long.

```
german_credit <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/ge
# summary(german_credit)
```

Converting the response column V21 to binary 1 = Good and 0 = Bad. In original data set, 1 = Good and 2 = Bad.

```
german_credit$V21[german_credit$V21==1] = 1
german_credit$V21[german_credit$V21==2] = 0
```

Splitting dataset into test and training sets:

```
set.seed(8010)
train_set = createDataPartition(y = german_credit$V21, p = 0.7, list = FALSE)
train = german_credit[train_set, ]
test = german_credit[-train_set,]
```

Let's try creating the regression model using all predictors:

```
log_reg_model <- glm(V21 ~ ., data = train, family = binomial(link = 'logit'))
# summary(log_reg_model)
```

Here, I'm recreating the model with only significant variables:

```
log_reg_model2 <- glm(V21 ~ V1+V2+V3+V4+V5+V6+V7+V8+V9+V14+V15+V19 , data = train, family = binomial(li
# summary(log_reg_model2)
```

It looks like all variables are significant but not all categorical values within the variables are significant. Ie - V1A14 is significant but V1A12 is not. I want to separate the significant categorical values and convert them to binary.

```
train$V1A13[train$V1 == "A14"] <- 1
train$V1A13[train$V1 != "A14"] <- 0
train$V3A34[train$V3 == "A34"] <- 1
train$V3A34[train$V3 != "A34"] <- 0
train$V4A41[train$V4 == "A41"] <- 1
train$V4A41[train$V4 != "A41"] <- 0
train$V4A42[train$V4 == "A42"] <- 1
train$V4A42[train$V4 != "A42"] <- 0
train$V4A43[train$V4 == "A43"] <- 1
train$V4A43[train$V4 != "A43"] <- 0
train$V4A49[train$V4 == "A49"] <- 1
train$V4A49[train$V4 != "A49"] <- 0
train$V6A65[train$V6 == "A65"] <- 1
train$V6A65[train$V6 != "A65"] <- 0
train$V7A74[train$V7 == "A74"] <- 1
train$V7A74[train$V7 != "A74"] <- 0
train$V9A93[train$V9 == "A93"] <- 1
train$V9A93[train$V9 != "A93"] <- 0
train$V14A143[train$V14 == "A143"] <- 1
train$V14A143[train$V14 != "A143"] <- 0
train$V15A152[train$V15 == "A152"] <- 1
train$V15A152[train$V15 != "A152"] <- 0
train$V19A192[train$V15 == "A192"] <- 1
train$V19A192[train$V15 != "A192"] <- 0
```

```
log_reg_model3 <- glm(V21 ~ V1A13+V2+V3A34+V4A41+V4A42+V4A43+V4A49+V5+V6A65+V7A74+V8+V9A93+V14A143+V15A
# summary(log_reg_model3)
```

It looks like almost all of our variables are significant; I'll just remove VA4A42 and V7A74 to get our final model:

```
log_reg_model4 <- glm(V21 ~ V1A13+V2+V3A34+V4A41+V4A43+V4A49+V5+V6A65+V8+V9A93+V14A143+V15A152+V19A192
summary(log_reg_model4)
```

```
##
## Call:
## glm(formula = V21 ~ V1A13 + V2 + V3A34 + V4A41 + V4A43 + V4A49 +
##     V5 + V6A65 + V8 + V9A93 + V14A143 + V15A152 + V19A192, family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6707  -0.8007   0.4025   0.7411   1.8986
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.8155333  0.4391544   1.857 0.063303 .
## V1A13        1.3468858  0.2271271   5.930 3.03e-09 ***
## V2          -0.0404791  0.0108358  -3.736 0.000187 ***
## V3A34        1.0394823  0.2434155   4.270 1.95e-05 ***
## V4A41        1.5472169  0.4116080   3.759 0.000171 ***
## V4A43        1.0600963  0.2478124   4.278 1.89e-05 ***
## V4A49        0.8698966  0.3404466   2.555 0.010614 *
## V5          -0.0001054  0.0000501  -2.103 0.035458 *
## V6A65        0.9544379  0.2958152   3.226 0.001253 **
## V8          -0.3877889  0.1004630  -3.860 0.000113 ***
## V9A93        0.6005116  0.2016197   2.978 0.002897 **
## V14A143      0.7094036  0.2322690   3.054 0.002256 **
## V15A152      0.4420235  0.2143436   2.062 0.039187 *
## V19A192             NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 855.21  on 699  degrees of freedom
## Residual deviance: 653.62  on 687  degrees of freedom
## AIC: 679.62
##
## Number of Fisher Scoring iterations: 5
```

Now let's test this model on the test data. In a hidden code cell, I've converted the categorical variables in the test data set into binary variables.

Let's predict the responses and create a confusion matrix to determine accuracy of model using default of p = 0.5:

```
res <-predict(log_reg_model4, newdata = test[,-21], type="response")
yhat <- round(res) #Rounds up with p > 0.5
y <- test$V21
conf_table <- table(yhat, y)
conf_table
```

```
##     y
## yhat   0   1
##    0  33  29
##    1  57 181
```

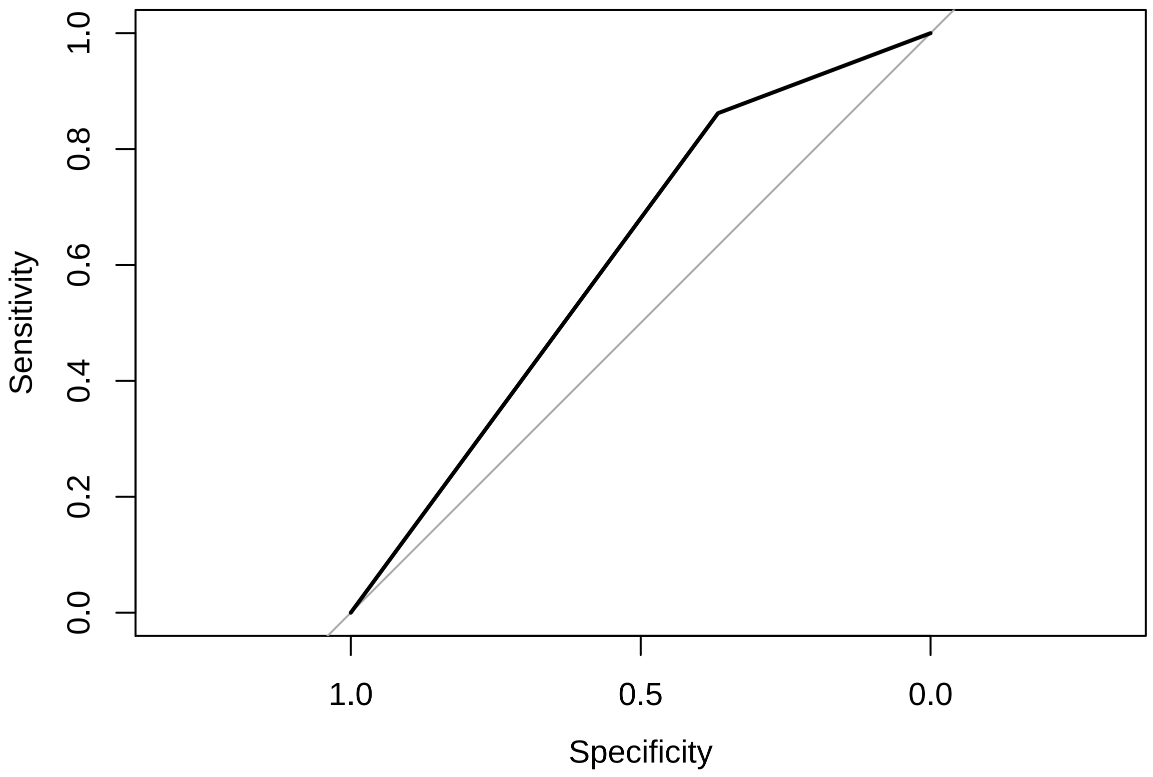
```
acc <-(conf_table[1,1] + conf_table[2,2]) / sum(conf_table)
acc
```

```
## [1] 0.7133333
```

57 or 63% of applicants that are classified as bad risk would be considered good risk under this model. This is costly in terms of approving loans to bad risk applicants.

Let's plot the ROC curve:

```
roc_curve <- roc(test$V21, yhat, auc = TRUE)
plot(roc_curve)
```



```
roc_curve$auc
```

```
## Area under the curve: 0.6143
```

With an AUC of 61%, that means my model will classify applicants correctly 61% of the time. Based on the confusion matrix above, it looks like my model is pretty good at classifying the good applicants but not so good at classifying the bad applicants.

I want to try and change the threshold to see if I can improve the classification.

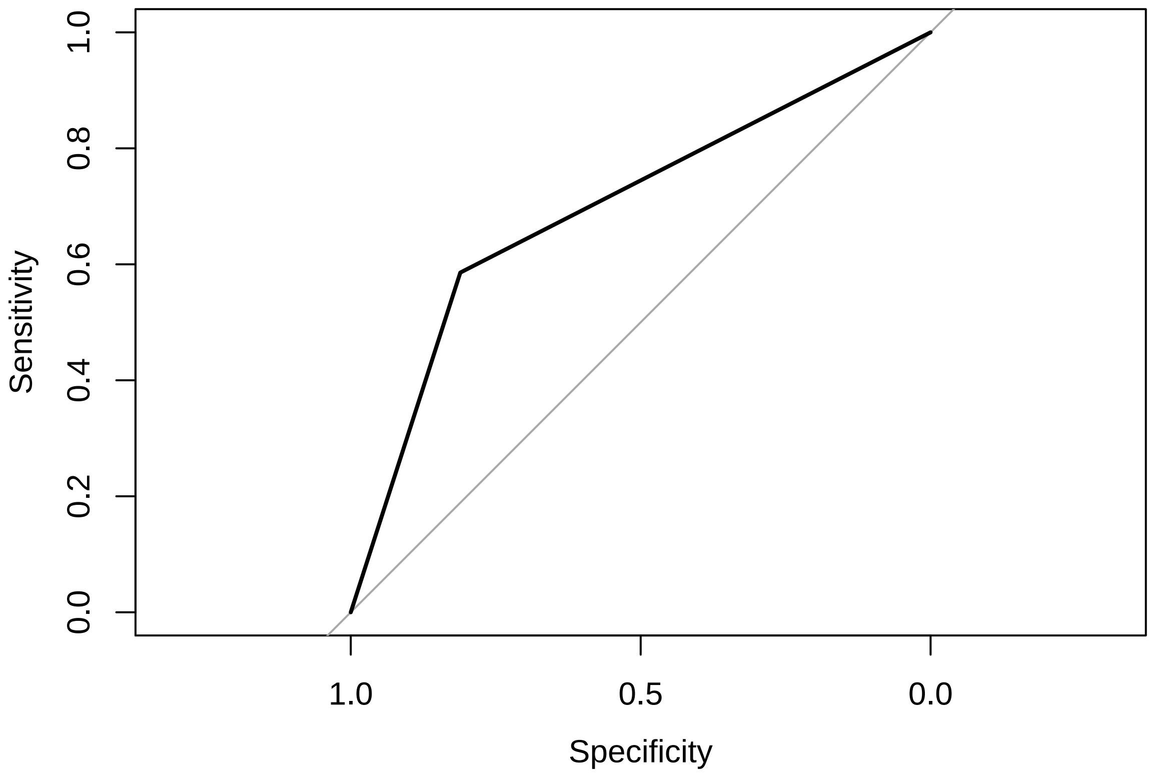
```
vals = c(0.6,0.7,0.8,0.9)
for (i in vals) {
  yhat <- as.integer(res > i)
  conf_table <- table(yhat,y)
  acc <-(conf_table[1,1] + conf_table[2,2]) / sum(conf_table)
  fp <- conf_table[2,1]/sum(conf_table[, 1])
  print(paste0("p = ", i))
  print(paste0("Accuracy = ", acc))
  print(paste0("False Positives = ", fp))
  }
```

```
## [1] "p = 0.6"
## [1] "Accuracy = 0.71"
## [1] "False Positives = 0.477777777777778"
## [1] "p = 0.7"
## [1] "Accuracy = 0.693333333333333"
## [1] "False Positives = 0.311111111111111"
## [1] "p = 0.8"
## [1] "Accuracy = 0.653333333333333"
## [1] "False Positives = 0.188888888888889"
## [1] "p = 0.9"
## [1] "Accuracy = 0.523333333333333"
## [1] "False Positives = 0.0777777777777778"
```

Based on these results, I would choose p = 0.8 as my new threshold. My accuracy has decreased compared to when p = 0.5 but at least the % of bad applicants my model preducts has decreased to 18% (vs 61% when p = 0.5).

Plotting the ROC curve for p = 0.8:

```
yhat <- as.integer(res > 0.8)
roc_curve <- roc(test$V21, yhat, auc = TRUE)
plot(roc_curve)
```



```
roc_curve$auc
```

```
## Area under the curve: 0.6984
```

AUC has increased with p = 0.8.

Question 10.2.2

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.

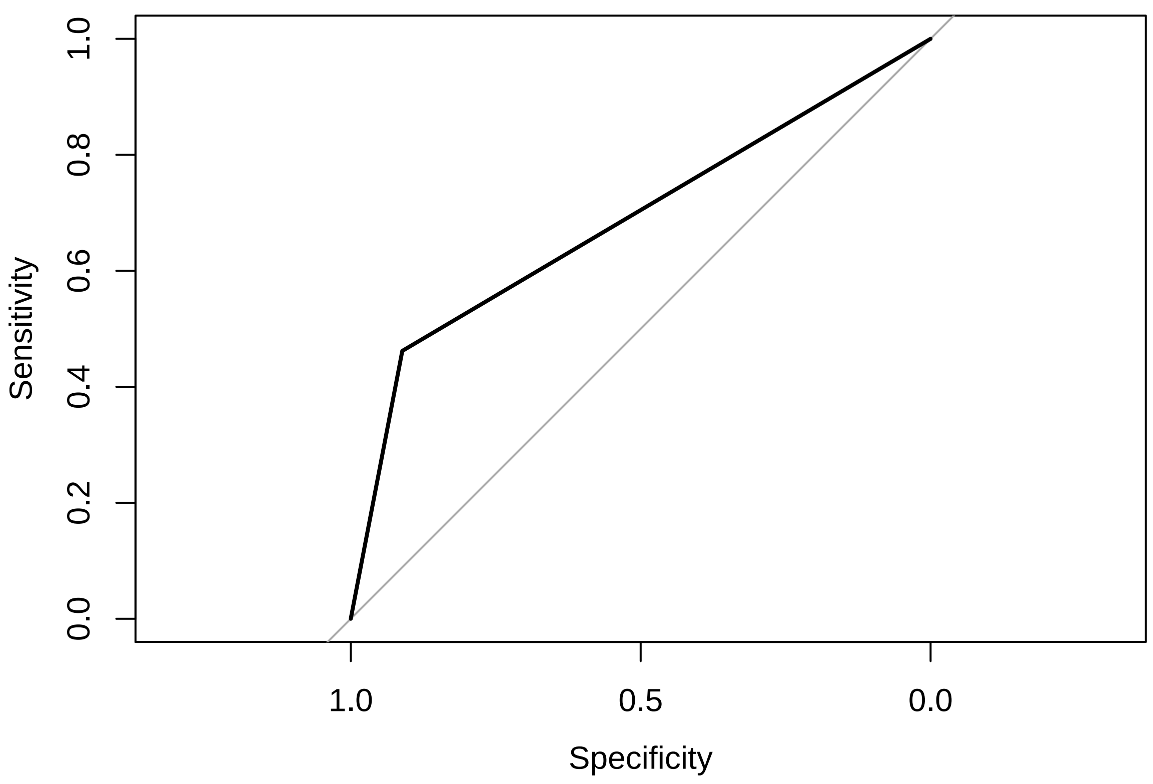
Similar what I did in part 1 of this question, I will loop through more granular thresholds and calculate cost to determine the best one that minimizes the cost. Cost = 5*False Positives + False Negatives

```
loss_results = c()

for (i in seq(from = 1, to = 100)) {
  n = 1
  yhat <- as.integer(res > i/100)
  conf_table <- confusionMatrix(data = factor(yhat),
                       reference = factor(y),
                       positive = '1')
  fn = conf_table$table[1,2]
  fp = conf_table$table[2,1]
  cost <- (5*fp)+(fn)
  loss_results <- c(loss_results, cost)

}

which.min(loss_results)
```

```
## [1] 86
```

Based on this result, my threshold to minimize cost should be p = 0.86.

```
yhat <- as.integer(res > 0.86)
roc_curve <- roc(test$V21, yhat, auc = TRUE)
plot(roc_curve)
```



```
roc_curve$auc
```

```
## Area under the curve: 0.6865
```