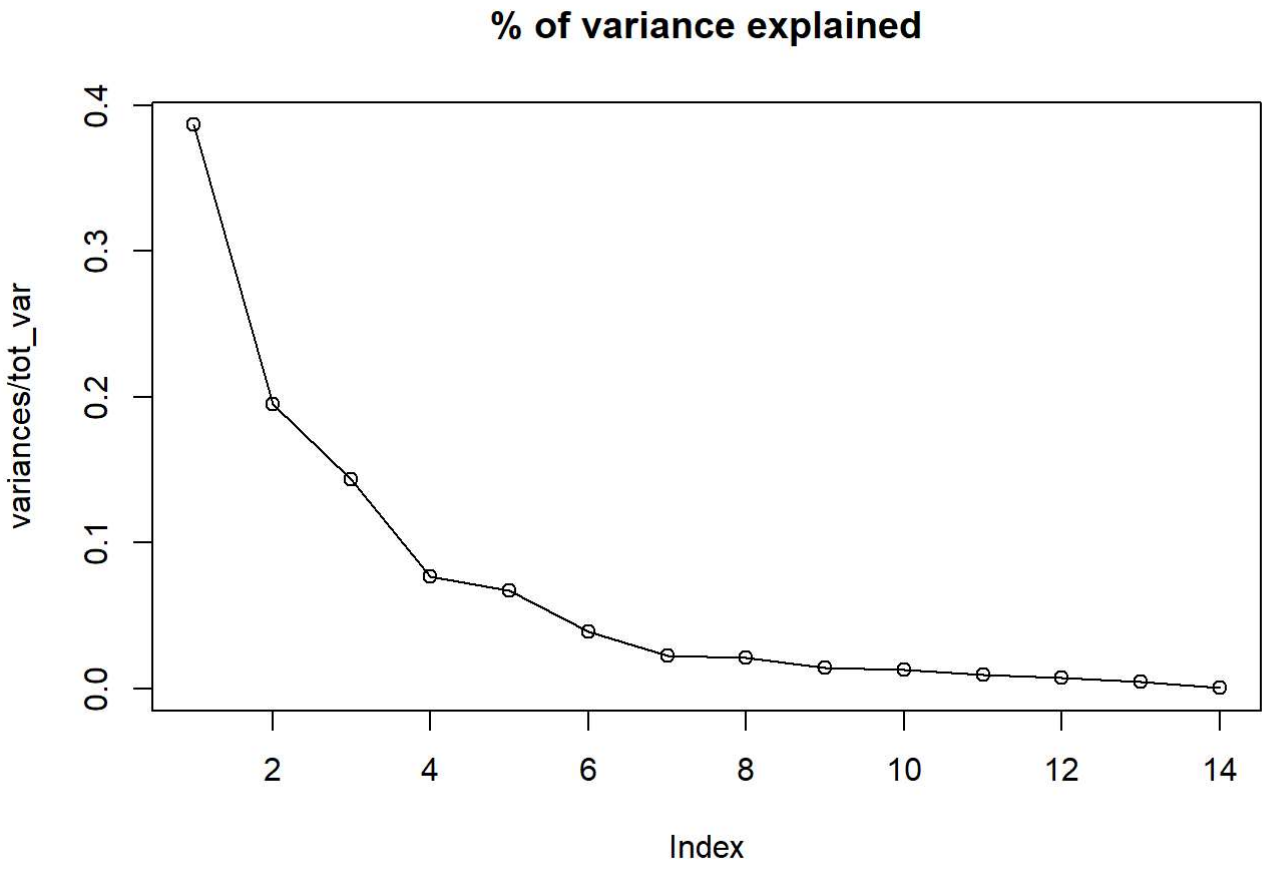
Homework week 4

2022-06-11

Question 9.1
Answer:

First, load the data and perform PCA. We need to exclude the response variable (column 16), the binary variable (column 2) and scale the data before implementing PCA.
Then we display the % of variance explained by each component in a graph.

```
rm(list = ls())
set.seed(19)
#read data
data <- read.table("C:\\Data\\week 4 data-summer\\data 9.1\\uscrime.txt",
                   stringsAsFactors = FALSE, header = TRUE)
#performing PCA, excluding unnecessary variables
crime_pca <- prcomp(data[, c(-2, -16)], scale = TRUE, center = TRUE)

# dsiplay the percentage of variance explained by each principle component
variances <- (crime_pca$sdev)^2
tot_var <- sum((crime_pca$sdev)^2)
plot(variances/tot_var, type = "o", main = "% of variance explained")
```



From the line chart above we can see that the first 6 principle components explain most the variances in the data set. Therefore, we will use the first 6 principle components in our model.

```
    # Load the first 6 principle components and the response variable (Crime)
    # into one data frame
    crime_pca_df <- cbind(crime_pca$x[,1:6], data["Crime"])
    head(crime_pca_df)
```

```
##         PC1         PC2         PC3         PC4        PC5        PC6 Crime
## 1 -3.893446 -1.29197714 -1.10138991  0.85371781 -0.2582188 -0.2220763   791
## 2  0.971018  0.69084709 -0.05783388 -0.36390197 -1.1311282  0.6335173  1635
## 3 -3.946974  0.08584861 -0.36356482  0.58382631  0.3690864 -0.7120597   578
## 4  3.951699 -2.29488126  0.22720214 -0.04741697 -1.7050714 -0.6698441  1969
## 5  1.647039  1.44338543  1.25954528  0.64890563 -0.1162943  0.4327958  1234
## 6  2.861367 -0.11765453  0.51823342  1.44813377  1.0732058  0.6906828   682
```

Next, we will perform a linear regression using the data frame generated in last step.

```
    model <- lm(Crime~PC1+PC2+PC3+PC4+PC5+PC6, data = crime_pca_df)
    summary(model)
```

```
##
## Call:
## lm(formula = Crime ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6, data = crime_pca_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -399.15 -166.78   15.28  150.91  452.53
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  905.085     36.058  25.101  < 2e-16 ***
## PC1           76.750     15.668   4.898 1.64e-05 ***
## PC2          -57.648     22.072  -2.612   0.0126 *
## PC3           24.313     25.744   0.944   0.3506
## PC4            3.786     35.157   0.108   0.9148
## PC5         -235.831     37.674  -6.260 2.04e-07 ***
## PC6           64.174     49.221   1.304   0.1998
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 247.2 on 40 degrees of freedom
## Multiple R-squared:  0.6448, Adjusted R-squared:  0.5915
## F-statistic:  12.1 on 6 and 40 DF,  p-value: 1.036e-07
```

Then we need to find the coefficients for each original variable using the results above from both PCA and linear regression model.

```
    #Find coefficients for orginal variables(scaled) by multiplying
    #coeffients of principle components with eighenvectors
    beta_vec <- model$coefficients[2:7]
    beta_intercept <- model$coefficients[1]

    alpha_vec <- c((crime_pca$rotation[,1:6] %*% beta_vec)[,1])

    #display coefficients for original factors, without removing effect of scaling
    as.data.frame(alpha_vec)
```

```
##         alpha_vec
## M       95.400235
## Ed      13.470475
## Po1    124.092698
## Po2    119.580606
## LF      43.199820
## M.F    107.629381
## Pop     28.786137
## NW     103.806024
## U1       2.756535
## U2      29.061629
## Wealth  38.505494
## Ineq     8.489242
## Prob   -43.150501
## Time    35.340376
```

Considering that the data has been scaled before applying PCA to it, we need to execute following steps to remove the effect of scaling:

```
    #Calculate the means and standard deviations of the variables in the orginal
    #data, excluding response variable (Crime) and binary variable (So)
    means <- colMeans(data[,c(-2,-16)])
    stdev <- apply(data[,c(-2,-16)],2,sd)

    #remove effect of scaling
    origin_vec <- alpha_vec/stdev
    as.data.frame(origin_vec)
```

```
##           origin_vec
## M       7.590946e+01
## Ed      1.204119e+01
## Po1     4.175538e+01
## Po2     4.276644e+01
## LF      1.068990e+03
## M.F     3.652494e+01
## Pop     7.561134e-01
## NW      1.009503e+01
## U1      1.528963e+02
## U2      3.441099e+01
## Wealth  3.990581e-02
## Ineq    2.127840e+00
## Prob   -1.897812e+03
## Time    4.986722e+00
```

```
    origin_intercept <- beta_intercept - sum(alpha_vec * means / stdev)

    #display coefficients for original factors, removing effect of scaling
    as.data.frame(origin_intercept)
```

```
##             origin_intercept
## (Intercept)        -5717.968
```

The results above shows the coefficients and intercepts for the original factors. Next, we are going to using them to estimate the crime rate in each row of the original data set (without binary variable, column 2)
Then we are going to compare our estimated result with real results, and calculate adjusted R squared.

```
    input <- as.matrix(data[, c(-2,-16)])
    estimate <- input %*% origin_vec + origin_intercept

    RSS <- sum((data[,16]-estimate)^2)
    TSS <- sum((data[,16]-mean(data[,16]))^2)

    R2 <- 1 - RSS/TSS
    R2
```

```
## [1] 0.644773
```

```
    Adj_R2 = 1- (1-R2)*nrow(data)/(nrow(data)-6-1)
    Adj_R2
```

```
## [1] 0.5826083
```

As the last step of this question, we are going to make a prediction with the data of the new city.

```
    input <- as.matrix(data.frame("M" = 14.0,

                                  "Ed" = 10.0,
                                  "Po1" = 12.0,
                                  "Po2" = 15.5,
                                  "LF" = 0.64,
                                  "M.F" = 94.0,
                                  "Pop" = 150,
                                  "NW" = 1.1,
                                  "U1" = 0.12,
                                  "U2" = 3.6,
                                  "Wealth" = 3200,
                                   "Ineq" = 20.1,
                                   "Prob" = 0.04,
                                  "Time" = 39.0))
    prediction <- input %*% origin_vec + origin_intercept
    prediction
```

```
##          [,1]
## [1,] 1302.405
```

Question 10.1
Answer:

First, we read the data and run a regression tree model by calling tree() function. Since later we are going to use cross validation to prune the tree model, we need to split the data into 2 parts, 70% for training and cross validation, the rest 30% for testing.
We are going to show the nodes, and branching criteria in the graph below.

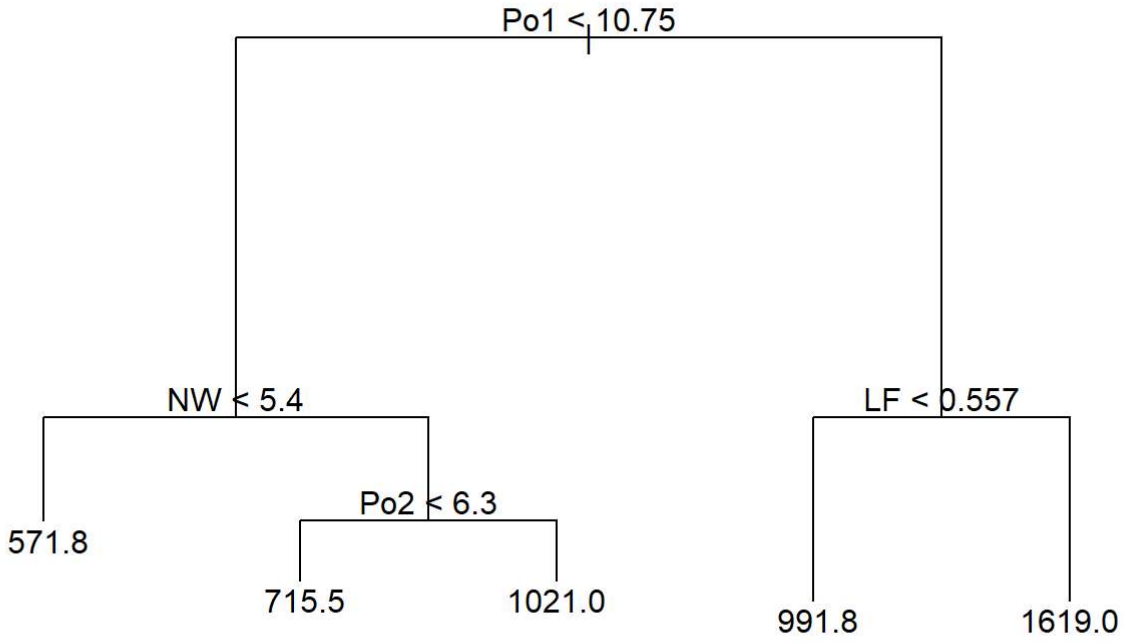
```
    library(tree)
    rm(list = ls())
    set.seed(19)
    #read data
    data <- read.table("C:\\Data\\week 4 data-summer\\data 9.1\\uscrime.txt",
                       stringsAsFactors = FALSE, header = TRUE)
    #spliting data into training(cross validation) and testing
    training_index <- sample(1:nrow(data), size = round(nrow(data)*0.7))
    training <- data[training_index,]
    testing <- data[-training_index,]

    #run the regression tree model with training dataset
    tree <- tree(Crime~., data = training)
    tree
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 33 5601000  929.1
##    2) Po1 < 10.75 23 1539000  765.5
##      4) NW < 5.4 9  318300  571.8 *
##      5) NW > 5.4 14  666100  890.0
##       10) Po2 < 6.3 6  140900  715.5 *
##       11) Po2 > 6.3 8  205400 1021.0 *
##    3) Po1 > 10.75 10 2031000 1305.0
##      6) LF < 0.557 5  478200  991.8 *
##      7) LF > 0.557 5  568900 1619.0 *
```

```
    summary(tree)
```

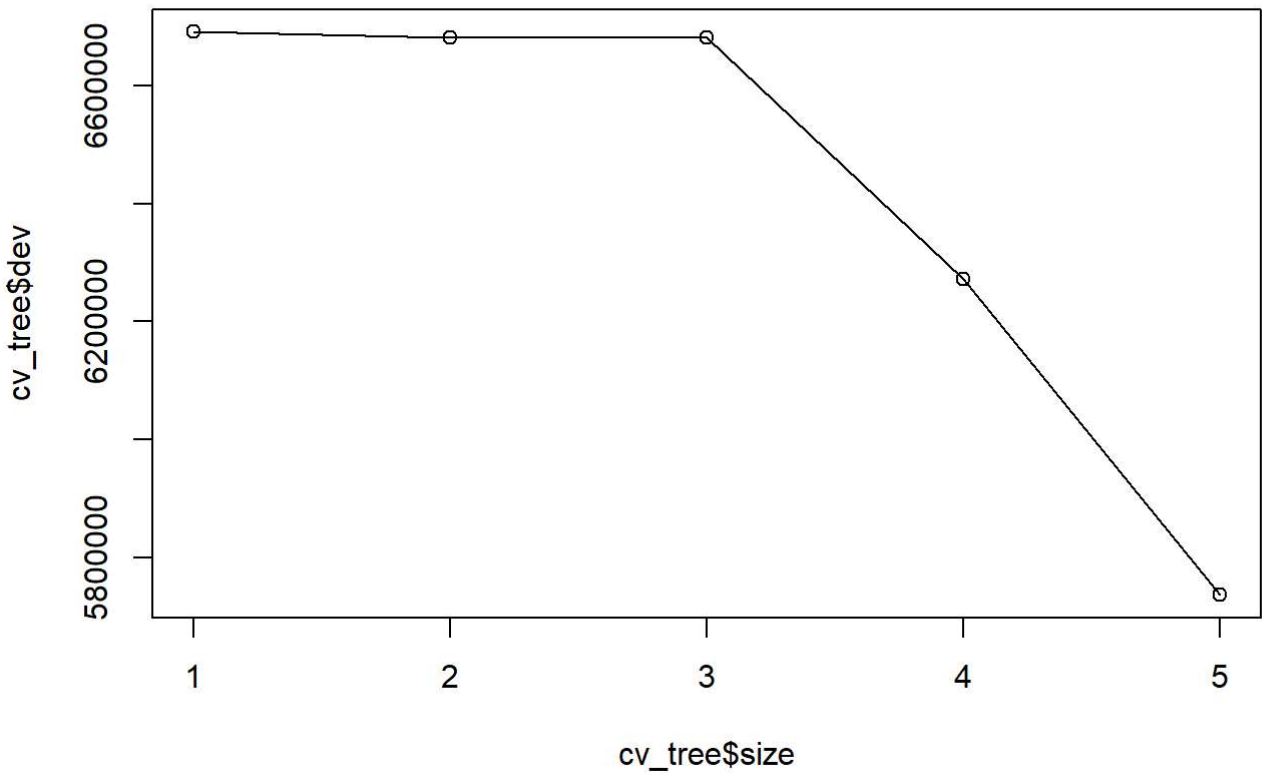
```
##
## Regression tree:
## tree(formula = Crime ~ ., data = training)
## Variables actually used in tree construction:
## [1] "Po1" "NW"  "Po2" "LF"
## Number of terminal nodes:  5
## Residual mean deviance:  61130 = 1712000 / 28
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -394.00 -171.90  -29.78    0.00  133.20  563.20
```

```
    plot(tree)
    text(tree, pretty = 0)
```



Next, we need to prune the tree and choose the best number of nodes. Here we use cv.tree() to perform the pruning and cross validation process at the same time. We will use deviance to measure the quality of each model.
We will do a 3-fold cross validation and show the deviance of the model under each number of nodes in a line chart.

```
set.seed(10)
#prune the tree & cross validation
cv_tree <- cv.tree(tree,K=3)
plot(cv_tree$size, cv_tree$dev, type = "o")
```



From the chart above, we can see that the model has lowest deviation when there are 5 nodes in total. Therefore, we will choose have 5 nodes in our tree. We will show this model by calling prune.tree() function with parameter best = 5.
The structure of the tree will be displayed in a graph.

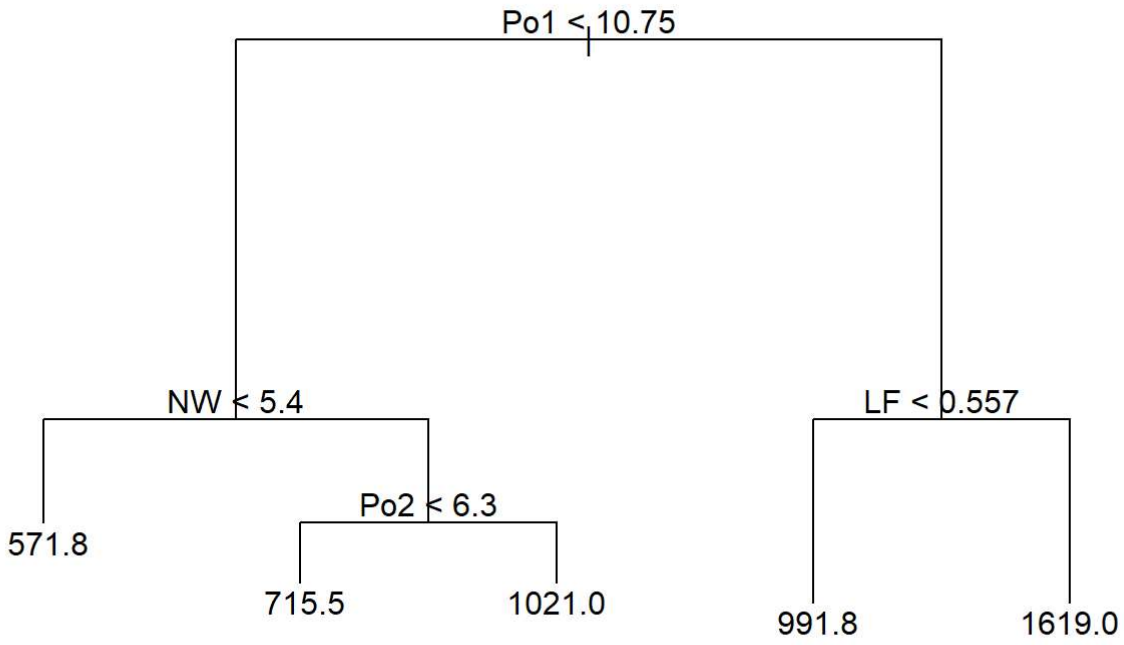
```
tree_pruned <- prune.tree(tree, best = 5)
tree_pruned
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 33 5601000  929.1
##    2) Po1 < 10.75 23 1539000  765.5
##      4) NW < 5.4 9  318300  571.8 *
##      5) NW > 5.4 14  666100  890.0
##       10) Po2 < 6.3 6  140900  715.5 *
##       11) Po2 > 6.3 8  205400 1021.0 *
##    3) Po1 > 10.75 10 2031000 1305.0
##      6) LF < 0.557 5  478200  991.8 *
##      7) LF > 0.557 5  568900 1619.0 *
```

```
summary(tree_pruned)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = training)
## Variables actually used in tree construction:
## [1] "Po1" "NW"  "Po2" "LF"
## Number of terminal nodes:  5
## Residual mean deviance:  61130 = 1712000 / 28
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -394.00 -171.90  -29.78    0.00  133.20  563.20
```

```
    plot(tree_pruned)
    text(tree_pruned)
```




```
    # Calculate R squared
    prediction <- predict(tree_pruned, testing[,1:15])
    RSS <- sum((testing[,16] - prediction)^2)
    TSS <- sum((testing[,16] - mean(testing[,16]))^2)
    R2 <- 1- RSS/TSS
    R2
```

```
## [1] 0.04584516
```

From the charts and tables in previous steps, we can have following findings:
1. “Po1” is probably the most important factor in the regression tree model. It is the node at the top of the tree. When Po1 < 10.75, the overall estimation of crime rate is much lower comparing to when Po1>=10.75.
2. Comparing to the deviation we see in cross validation (5736093), the deviation of the model shown in the last step (1712000) is much lower. That means our model may have a issue of overfitting to training data. The R squared value shown in the testing result also implies this issue. It is as low as only 4.58%.

In the next part of this questions. We are go to run a random forest model using the same training / testing data. To select the best model, we are going to change the values of mytry and nodesize parameters. We are going to select the combination of these two parameters that generates the best R squared value.
In the following code, we are going to try all the combinations of mytry between 1 to14, and nodesize between 2 and 10.

```
    library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

```
    set.seed(19)

    mtry_vec <- c()
    nodesize_vec <- c()
    R2_vec <- c()

    #try all possible combinations of mytry and nodesize
    for (m in 1:14){
      for (n in 2:10){
        rf <- randomForest(Crime~., data = training, importance = TRUE, mtry = m,
                           nodeszie = n )

        #calculate R squared value
        RSS <- sum((training[,16] - rf$predicted)^2)
        TSS <- sum((training[,16] - mean(training[,16]))^2)
        R2 <- 1- RSS/TSS

        mtry_vec <- append(mtry_vec,m)
        nodesize_vec <- append(nodesize_vec, n)
        R2_vec = append(R2_vec, R2)
      }
    }

    training_result_df <- data.frame("mytry" = mtry_vec,
                             "node size" = nodesize_vec,
                             "R2" = R2_vec)
    #display part of the training results
    head(training_result_df)
```

```
##   mytry node.size        R2
## 1     1         2 0.3027683
## 2     1         3 0.3569474
## 3     1         4 0.3645463
## 4     1         5 0.3255773
## 5     1         6 0.3370738
## 6     1         7 0.3669648
```

After reviewing all the results, we will find that the combination mytry=3 and nodesize=6 generates the best R squared value. (41.05%) using training data.
Therefore, we will choose mytry = 3 and nodesize = 6, and display the model in more details, running with testing data.

```
    set.seed(19)
    rf <- randomForest(Crime~., data = testing, importance = TRUE, mtry = 3,
                           nodeszie = 6 )
    rf
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = testing, importance = TRUE,      mtry = 3, nodeszie
= 6)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 76002.87
##                     % Var explained: 12.47
```

```
    rf$importance
```

```
##            %IncMSE IncNodePurity
## M        732.10650      38708.27
## So        23.87658       5132.08
## Ed       363.16270      37132.30
## Po1    10483.79150     197348.74
## Po2    13726.27332     157622.28
## LF      -657.93964      29230.19
## M.F    -1144.77825      39535.88
## Pop    -1156.57353      39786.06
## NW      -381.29790      47953.62
## U1       472.13250      35077.22
## U2      2054.21812      57467.56
## Wealth  7482.76063      83363.60
## Ineq   -1950.20277      37966.50
## Prob    5483.90492      95448.39
## Time     477.94403      52282.65
```

```
#Calculate R squared using testing data
RSS <- sum((testing[,16] - rf$predicted)^2)
TSS <- sum((testing[,16] - mean(testing[,16]))^2)
R2 <- 1- RSS/TSS
R2
```

```
## [1] 0.1247075
```

Reviewing the table above and compare the random forest tree model with the regression tree mode, we can have following findings:
1. The random forest model has better quality of fit than regression tree model since it has a higher R squared value running on testing data (12.47%)
2. In the random forest model, factors “Po1” and “Po2” are two most important factors, since they have the highest value of “%IncMSE”

Question 10.2
Answer:

I am recently applying for a new credit card. I think a logistic model can be used to predict the probability of whether my application will be approved.
The predictors my include: my gross annual income, monthly housing rent payment, credit score, years of longest credit history, etc.
Question 10.3
Question 10.3.1
Answer:

First, we need to read the data. Since the last column (response variable) has values of 1 and 2, we need to convert them to 0 and 1.
To measure the quality of fit, we also need to split the data into training and testing. Here we will use 70% of data for training and the rest 30% for testing.

```
rm(list = ls())
set.seed(19)
#read data
data <- read.table("C:\\Data\\week 4 data-summer\\data 10.3\\germancredit.txt",
                   stringsAsFactors = FALSE, header = FALSE)

#convert values in last column to 0 and 1
data$V21[data$V21==1] <- 0
data$V21[data$V21==2] <- 1
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
```

```
    #split data into training and testing
    training_index <- sample(1:nrow(data), size = round(nrow(data)*0.7))
    training <- data[training_index,]
    testing <- data[-training_index,]
```

Next, we will run the logistic regression model using all the variables and see the result. We will test the model use testing data set and compare the prediction results and the real response factor in a confusion matrix.
Right now, we don't have a probably threshold for the prediction. Therefore we will just round the predicted probabilities to zero and one.

```
    model <- glm(V21~., data = training, family=binomial(link="logit"))
    summary(model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1933  -0.7031  -0.3556   0.7312   2.4779
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.859e-01  1.297e+00   0.143 0.886093
## V1A12       -5.424e-01  2.674e-01  -2.029 0.042465 *
## V1A13       -1.424e+00  4.435e-01  -3.210 0.001326 **
## V1A14       -1.845e+00  2.836e-01  -6.504 7.84e-11 ***
## V2           2.054e-02  1.132e-02   1.815 0.069525 .
## V3A31       -4.140e-01  6.402e-01  -0.647 0.517836
## V3A32       -3.680e-01  5.167e-01  -0.712 0.476258
## V3A33       -1.217e+00  5.858e-01  -2.077 0.037791 *
## V3A34       -1.733e+00  5.378e-01  -3.222 0.001271 **
## V4A41       -1.759e+00  4.531e-01  -3.882 0.000104 ***
## V4A410      -1.383e+00  1.179e+00  -1.173 0.240744
## V4A42       -7.525e-01  3.201e-01  -2.351 0.018736 *
## V4A43       -4.869e-01  3.001e-01  -1.623 0.104668
## V4A44       -5.411e-01  8.941e-01  -0.605 0.545061
## V4A45       -2.986e-01  7.163e-01  -0.417 0.676776
## V4A46        4.326e-01  4.913e-01   0.880 0.378660
## V4A48       -1.345e+00  1.216e+00  -1.106 0.268696
## V4A49       -5.121e-01  3.949e-01  -1.297 0.194731
## V5           1.852e-04  5.651e-05   3.276 0.001051 **
## V6A62       -3.102e-01  3.523e-01  -0.881 0.378503
## V6A63       -3.862e-01  4.641e-01  -0.832 0.405241
## V6A64       -9.202e-01  6.311e-01  -1.458 0.144799
## V6A65       -8.523e-01  3.293e-01  -2.588 0.009645 **
## V7A72        5.377e-01  5.183e-01   1.037 0.299572
## V7A73        3.398e-01  5.035e-01   0.675 0.499767
## V7A74       -7.879e-01  5.523e-01  -1.427 0.153716
## V7A75       -5.283e-02  5.055e-01  -0.105 0.916759
## V8           3.682e-01  1.069e-01   3.446 0.000569 ***
## V9A92       -6.305e-01  4.577e-01  -1.378 0.168341
## V9A93       -8.454e-01  4.480e-01  -1.887 0.059181 .
## V9A94       -5.725e-01  5.269e-01  -1.087 0.277166
## V10A102      1.680e-01  5.264e-01   0.319 0.749603
## V10A103     -9.469e-01  4.890e-01  -1.936 0.052819 .
## V11          1.023e-01  1.035e-01   0.989 0.322717
## V12A122     -2.861e-02  3.068e-01  -0.093 0.925695
## V12A123      7.580e-02  2.783e-01   0.272 0.785333
## V12A124      3.433e-01  5.225e-01   0.657 0.511088
## V13         -1.834e-02  1.113e-02  -1.647 0.099466 .
## V14A142      9.205e-02  4.883e-01   0.189 0.850468
## V14A143     -6.556e-01  2.903e-01  -2.258 0.023923 *
## V15A152     -5.805e-01  2.756e-01  -2.107 0.035139 *
## V15A153     -2.242e-01  5.784e-01  -0.388 0.698257
```

```
## V16          4.997e-01  2.354e-01   2.123 0.033782 *
## V17A172     -1.320e-01  7.689e-01  -0.172 0.863697
## V17A173     -5.099e-02  7.360e-01  -0.069 0.944769
## V17A174      6.705e-02  7.500e-01   0.089 0.928760
## V18          3.714e-01  3.082e-01   1.205 0.228106
## V19A192     -2.753e-01  2.428e-01  -1.134 0.256812
## V20A202     -1.406e+00  7.724e-01  -1.820 0.068728 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 877.62  on 699  degrees of freedom
## Residual deviance: 630.06  on 651  degrees of freedom
## AIC: 728.06
##
## Number of Fisher Scoring iterations: 5
```

```
    #test the model use testing data
    predicttion <- predict(model, newdata = testing, type = "response")
    table(testing[,21], round(predicttion))
```

```
##
##       0   1
##   0 195  29
##   1  44  32
```

From the results above, the model correctly predicted 227 data points out of 300. But we can see from the summary table of the model that many predictors do not have a significant coefficients.
Therefore, we will run the model again after excluding all the predictors with p value bigger than 0.01.
For each column in original data, we will only keep the most significant coefficient, since the coefficients of the same column may be correlated with each other. Here's the results:

```r
#Manually add significtant predictors to the training / testing data sets
add_coefficients <- function(dataset){

  dataset$V1A14[dataset$V1 == "A14"] <- 1
  dataset$V1A14[dataset$V1 != "A14"] <- 0

  dataset$V3A34[dataset$V3 == "A34"] <- 1
  dataset$V3A34[dataset$V3 != "A34"] <- 0

  dataset$V4A41[dataset$V4 == "A41"] <- 1
  dataset$V4A41[dataset$V4 != "A41"] <- 0

  dataset$V6A65[dataset$V6 == "A65"] <- 1
  dataset$V6A65[dataset$V6 != "A65"] <- 0

  return (dataset)
}

training_2 <- add_coefficients(training)
testing_2 <- add_coefficients(testing)

#Rerun the model after excluding insignificant coefficients
model_2 <- glm(V21~
               V1A14+V3A34+V4A41+V5+V6A65+V8,
             data = training_2, family=binomial(link="logit"))
summary(model_2)
```

```
##
## Call:
## glm(formula = V21 ~ V1A14 + V3A34 + V4A41 + V5 + V6A65 + V8,
##     family = binomial(link = "logit"), data = training_2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3023  -0.8456  -0.4748   1.0095   2.3581
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.691e+00  3.478e-01  -4.864 1.15e-06 ***
## V1A14       -1.516e+00  2.201e-01  -6.887 5.69e-12 ***
## V3A34       -8.850e-01  2.262e-01  -3.912 9.14e-05 ***
## V4A41       -1.073e+00  3.658e-01  -2.934 0.003348 **
## V5           2.259e-04  3.784e-05   5.970 2.38e-09 ***
## V6A65       -8.271e-01  2.900e-01  -2.852 0.004346 **
## V8           3.434e-01  9.014e-02   3.810 0.000139 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 877.62  on 699  degrees of freedom
## Residual deviance: 724.90  on 693  degrees of freedom
## AIC: 738.9
##
## Number of Fisher Scoring iterations: 5
```

```
    #test the model use testing data
    predicttion <- predict(model_2, newdata = testing_2, type = "response")
    table(testing_2[,21], round(predicttion))
```

```
##
##       0   1
##   0 201  23
##   1  52  24
```

From the results above, even thought the prediction accuracy didn't improve (225 out of 300), there are no longer insignificant coefficients. That means we have reduced the likely hood of overfitting problem. Therefore, we are going to choose the second model.

Question 10.3.2

Answer:

In our prediction above, 0 stands for good and 1 stands for bad.

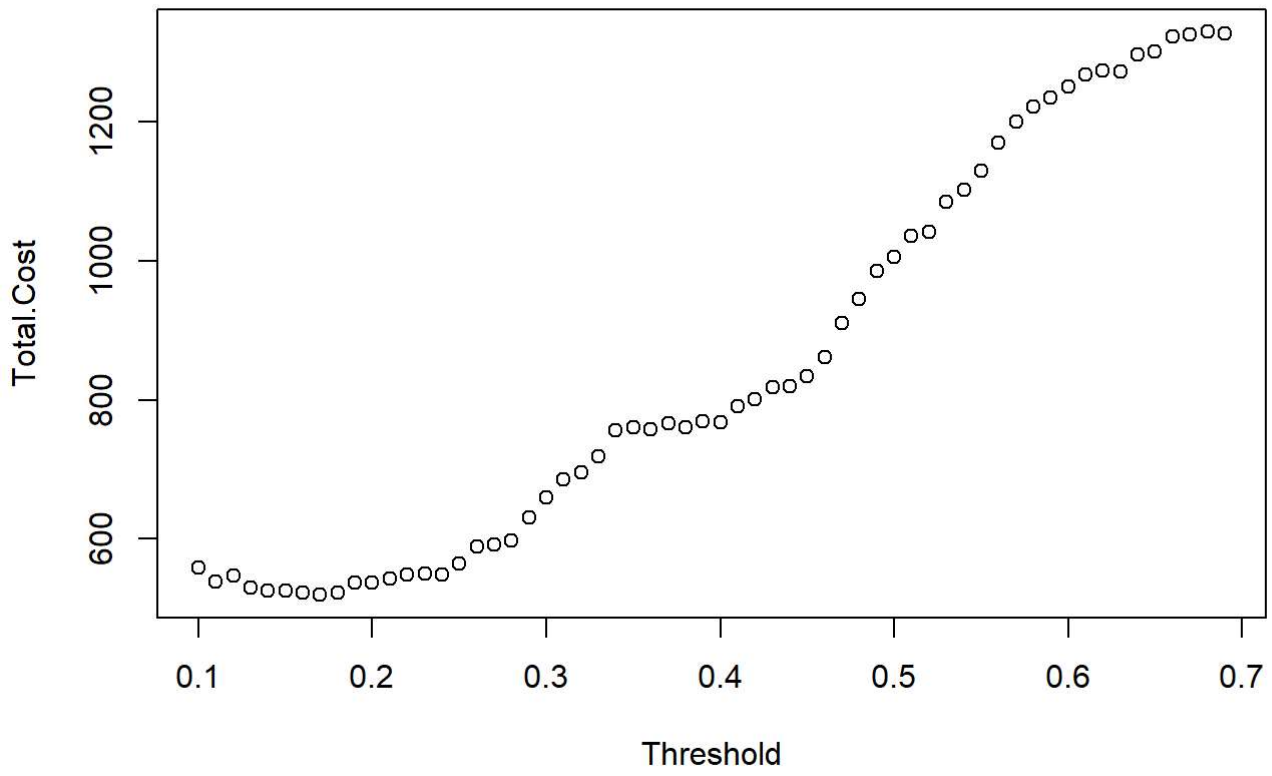
Assume that the cost of incorrectly identifying a bad customer as good is 5. and the cost of incorrectly classifying a good customer as bad is 1. If we do not change the probability threshold, the total cost in the last scenario equals 52 * 5 + 23 * 1 = 283

To find the best probability threshold and minimize the total cost, we will try every possible probability threshold

starting from 10% to 70%. If the probability prediction result for one customer is bigger or equal to the threshold, we will identify this customer as “bad” (1). Each time, we will raise the threshold by 1% and calculate the total cost. The probability threshold that minimize the total cost will be our answer.

Let’s run second model on the full data set to make the prediction.

```
#add necessary predictors to orginal dataset and run the model
data_2 <- add_coefficients(data)
pred_prob <- predict(model_2, newdata = data_2, type = "response")

#try different threshold
p = 0.1
threshold <- c()
cost <- c()
while (p<=0.7){
    prediction = pred_prob
    for (i in 1:length(prediction)){
      if (prediction[i] >= p){
        prediction[i] <- 1
      }
      else {prediction[i] <- 0}
    }

    con_mat <- table(data_2[,21], prediction)
    total_cost <- con_mat[1,2] * 1 + con_mat[2,1] * 5

    threshold <- append(threshold, p)
    cost <- append(cost, total_cost)
    p = p + 0.01
}

results <- data.frame("Threshold" = threshold, "Total Cost" = cost)
plot(results)
```




```
results
```

```
##    Threshold Total.Cost
## 1       0.10        558
## 2       0.11        538
## 3       0.12        547
## 4       0.13        529
## 5       0.14        525
## 6       0.15        525
## 7       0.16        522
## 8       0.17        519
## 9       0.18        522
## 10      0.19        537
## 11      0.20        537
## 12      0.21        542
## 13      0.22        548
## 14      0.23        550
## 15      0.24        548
## 16      0.25        564
## 17      0.26        589
## 18      0.27        592
## 19      0.28        597
## 20      0.29        631
## 21      0.30        659
## 22      0.31        685
## 23      0.32        696
## 24      0.33        719
## 25      0.34        756
## 26      0.35        760
## 27      0.36        757
## 28      0.37        766
## 29      0.38        760
## 30      0.39        769
## 31      0.40        767
## 32      0.41        791
## 33      0.42        801
## 34      0.43        818
## 35      0.44        820
## 36      0.45        834
## 37      0.46        862
## 38      0.47        911
## 39      0.48        945
## 40      0.49        986
## 41      0.50       1006
## 42      0.51       1036
## 43      0.52       1042
## 44      0.53       1085
## 45      0.54       1102
## 46      0.55       1130
## 47      0.56       1171
## 48      0.57       1201
## 49      0.58       1223
## 50      0.59       1236
## 51      0.60       1251
```

```
## 52      0.61       1269
## 53      0.62       1274
## 54      0.63       1273
## 55      0.64       1297
## 56      0.65       1301
## 57      0.66       1324
## 58      0.67       1326
## 59      0.68       1330
## 60      0.69       1328
```

From the results above, we can easily observe that the total cost will be minimized (equals 519) when the threshold is 17%. Therefore, we would choose the threshold to be 17%