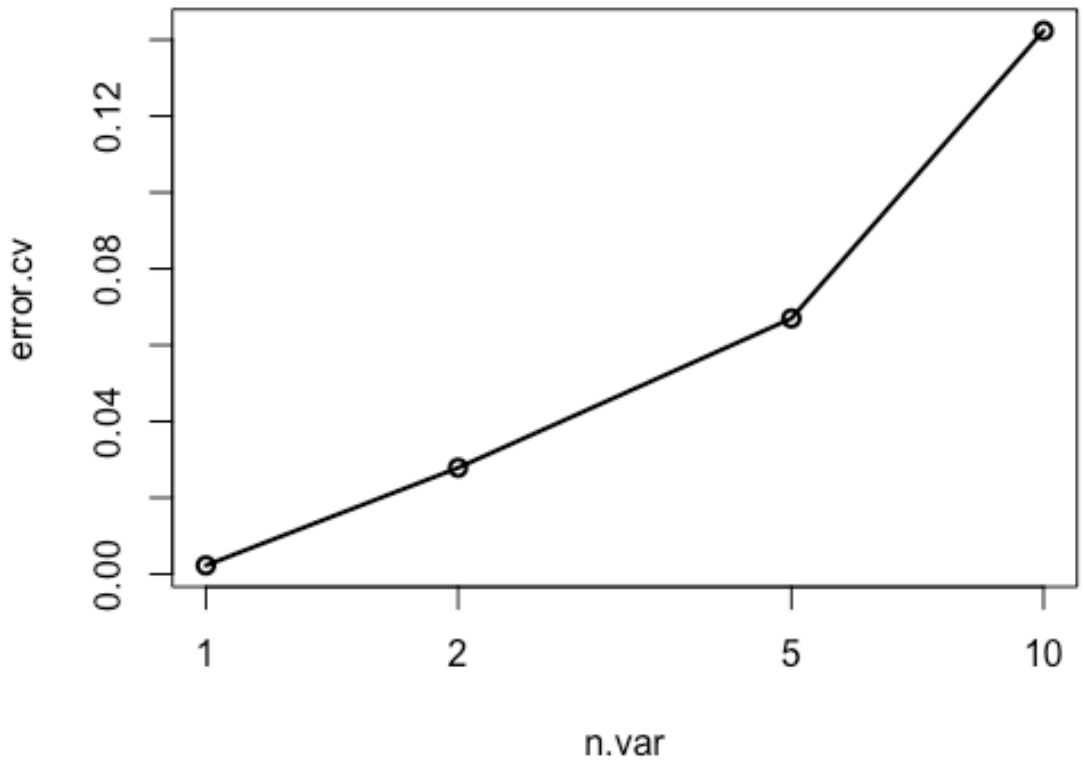
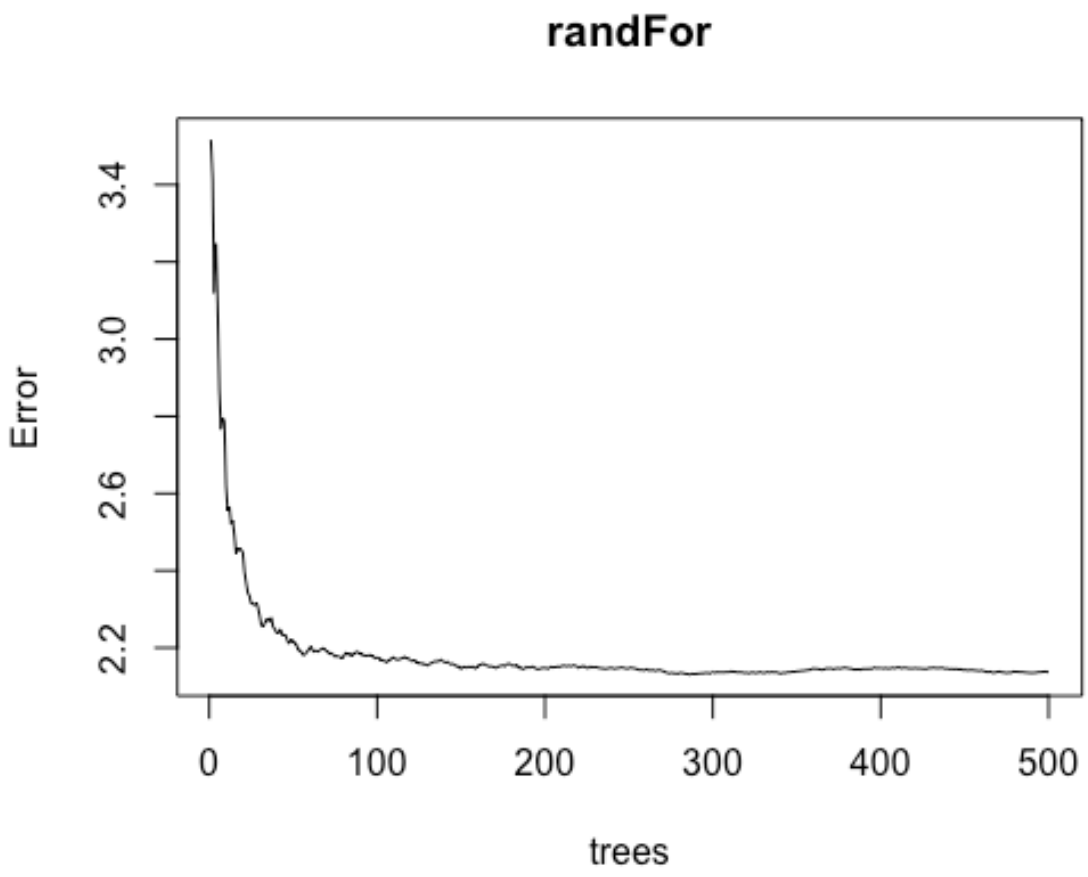
randomForest(formula = dataClear$armed ~ ., data = dataClear, importance = TRUE, ntree = 500)
Type of random forest: regression
Number of trees: 500
No. of variables tried at each split: 3

Mean of squared residuals: 2.128844
% Var explained: -8.31
>
RFCV (Random Forest Cross-Valdidation for feature selection)
# This function shows the cross-validated prediction performance of models with sequentially reduced
# number of predictors (ranked by variable importance) via a nested cross-validation procedure.
result <- rfcv(dataClear, dataClear$armed, cv.fold=3)
with(result, plot(n.var, error.cv, log="x", type="o", lwd=2))

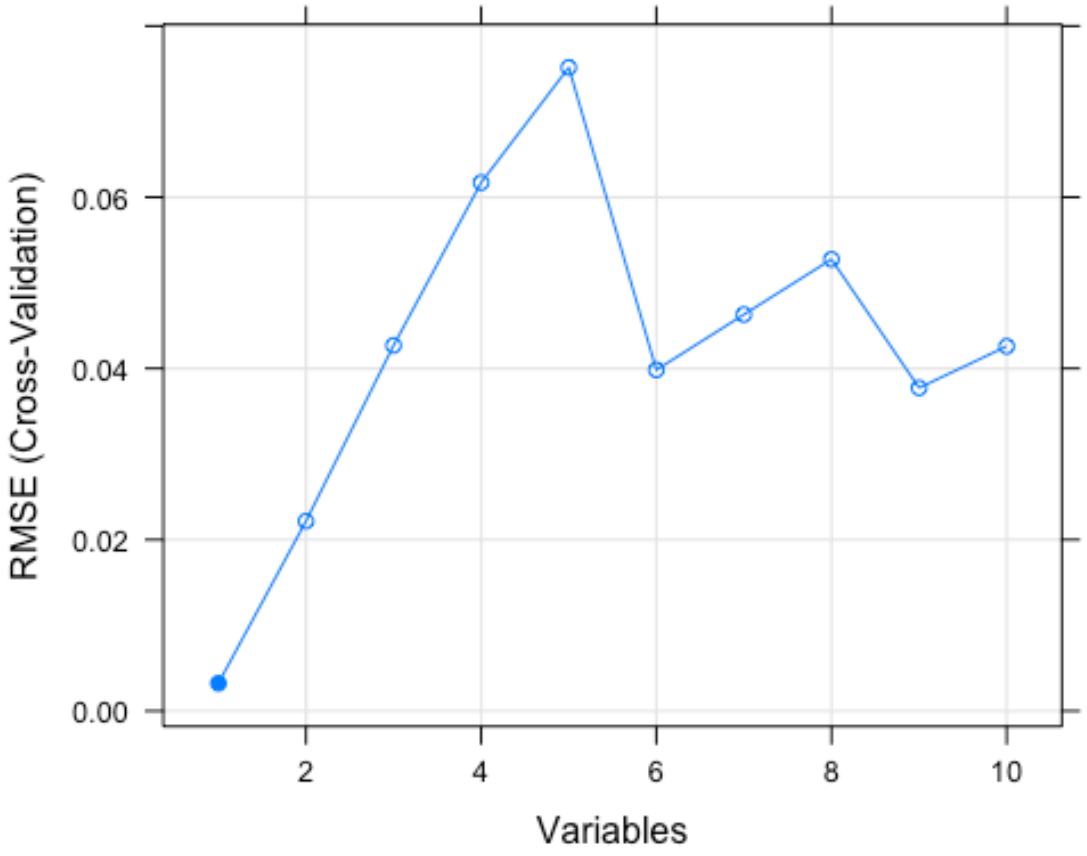


We can use now feature selection algorithms to find which one is more valuable for us?
randFor = randomForest::randomForest(dataClear$armed~. , data = dataClear)
randFor
plot(randFor)

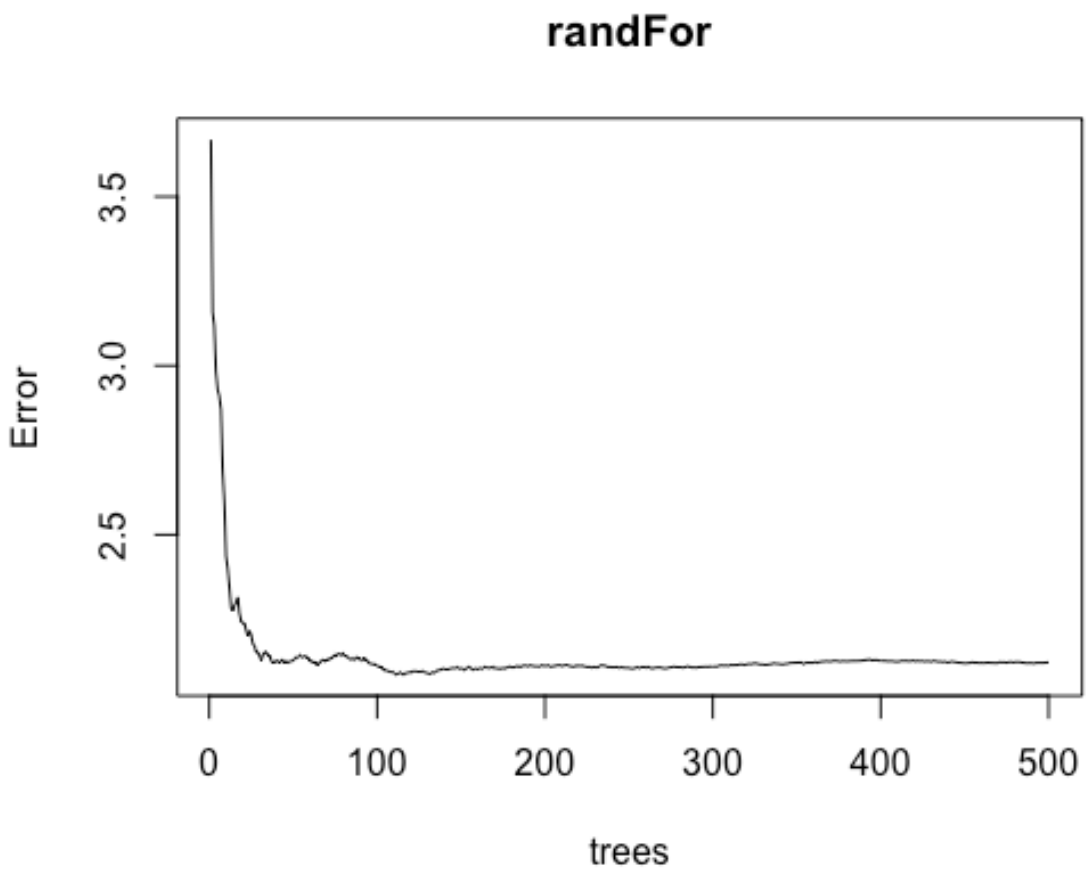



```
# ====================== RECURSIVE FOREST ELIMINATION TEST (RFE)
# Recursive feature elimination (RFE) is a feature selection method that fits a model
and
# removes the weakest feature (or features) until the specified number of features is
reached.
# Features are ranked by the model's coef_ or feature_importances_ attributes, and
by recursively
# eliminating a small number of features per loop, RFE attempts to eliminate
dependencies and collinearity that may exist in the model.
# ensure the results are repeatable
set.seed(7)
#needed for random forest selection function
library(mlbench)
library(caret)
# define the control using a random forest selection function
control <- rfeControl(functions=rfFuncs, method="cv", number=10)
# run the RFE algorithm
 x <- matrix(0,14,7)
 for(i in 1:nrow(x))
   { for(j in 1:ncol(x))
     {x[i,j] = 1}}
results <- rfe(remCorData[,1:10], remCorData[,10], sizes=c(1:10), rfeControl=control)
# summarize the results
```

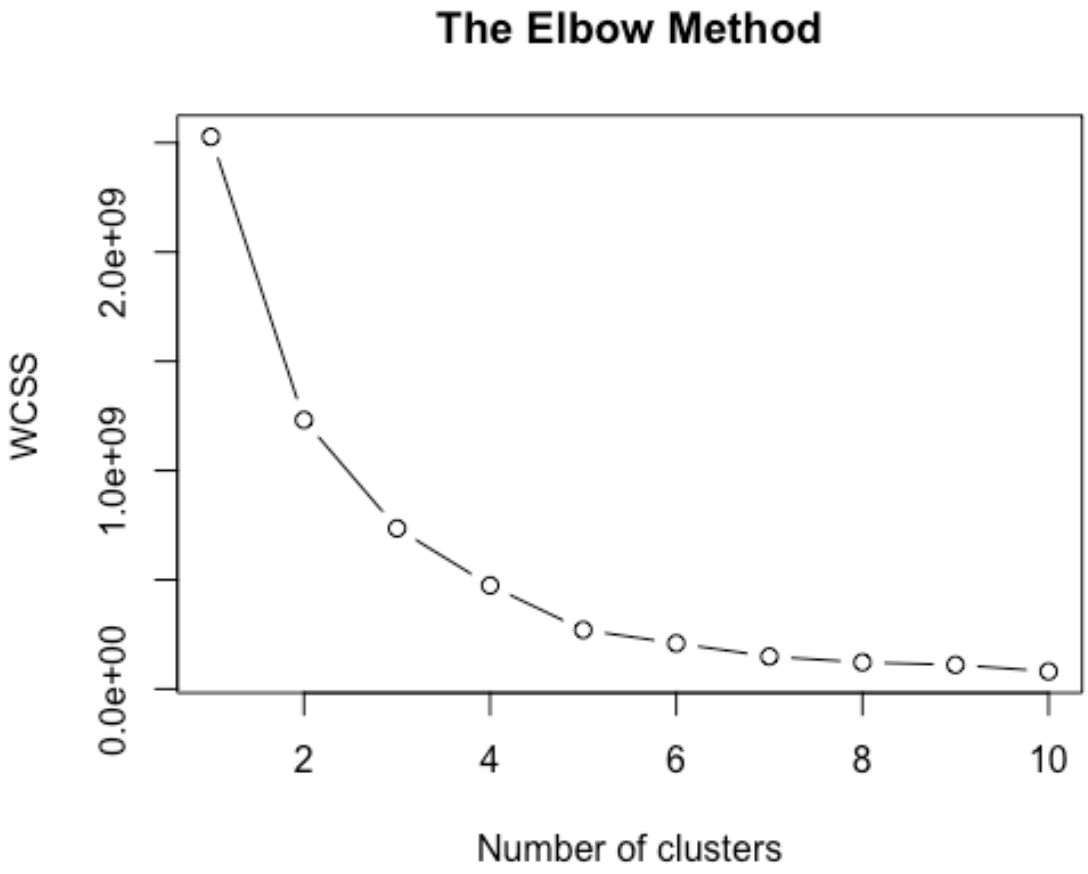
```
print(results)

Recursive feature selection

Outer resampling method: Cross-Validated (10 fold)

Resampling performance over subset size:

 Variables     RMSE Rsquared      MAE   RMSESD RsquaredSD     MAESD Selected
         1 0.003235   0.9996 0.001294 0.001971  0.0003408 0.0004409        *
         2 0.022173   0.9810 0.011309 0.011863  0.0167292 0.0033374
         3 0.042695   0.9516 0.027469 0.009386  0.0175396 0.0047792
         4 0.061711   0.9102 0.042359 0.010763  0.0236641 0.0066563
         5 0.075157   0.8763 0.052741 0.008989  0.0260524 0.0053195
         6 0.039834   0.9563 0.024631 0.008693  0.0183093 0.0035673
         7 0.046304   0.9435 0.030223 0.009011  0.0222125 0.0030353
         8 0.052754   0.9315 0.034931 0.009453  0.0240535 0.0029406
         9 0.037736   0.9615 0.023407 0.009203  0.0176110 0.0028329
        10 0.042593   0.9533 0.027091 0.008554  0.0180734 0.0027163

The top 1 variables (out of 1):
   college

predictors(results)
[1] "college"
> plot(results, type=c("g", "o"))
```




```
randFor = randomForest::randomForest(remCorData$armed~. , data = remCorData)
randFor
plot(randFor)
```

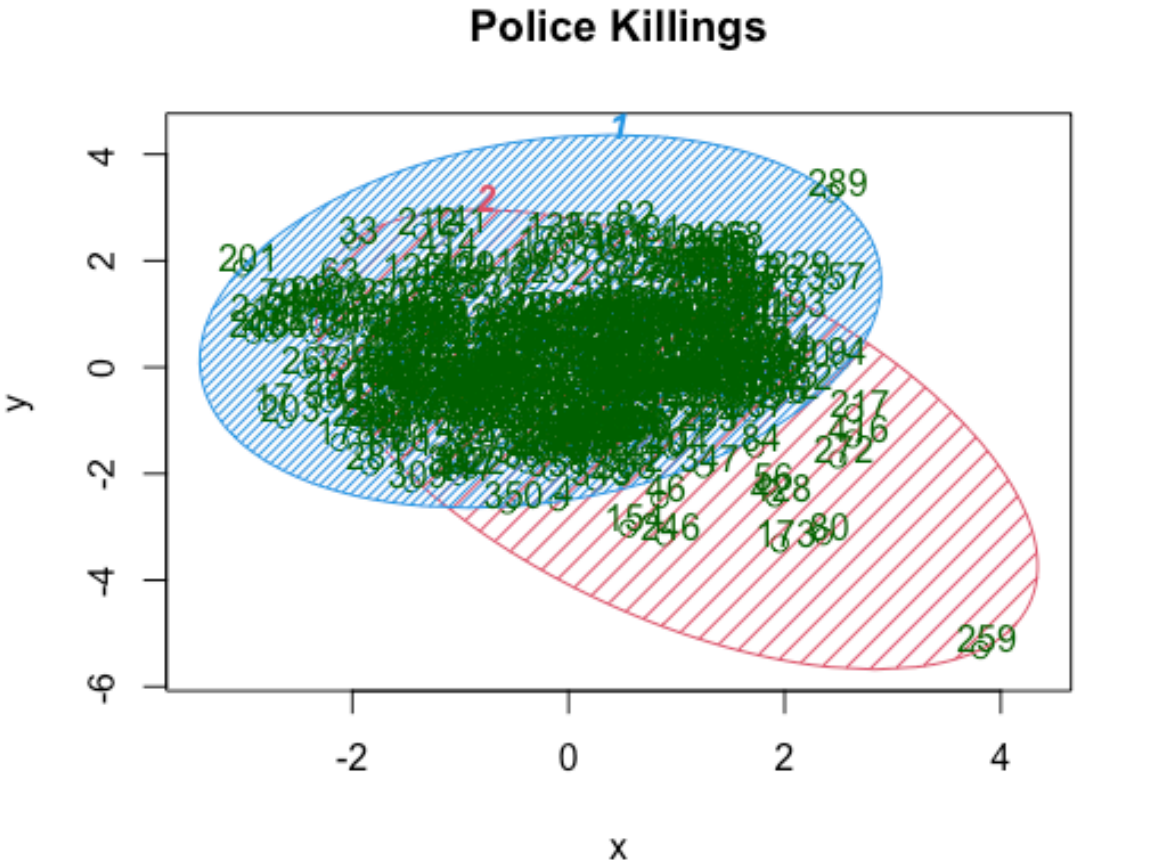



```
for (i in 1:10) v[i] = sum(kmeans(remCorData, i)$withinss)
plot(1:10,
     v,
     type = 'b',
     main = paste('The Elbow Method'),
     xlab = 'Number of clusters',
     ylab = 'WCSS')
```




```
# Above is for plotting Elbow Method, shows us the number of clusters
# Elbow method is used to find the optimum number of clusters, We can say that 6-8 is optimum number of cluster from plot.
# However we will use 2 as k because we are looking for Armed or not Armed for now;

# Applying K-Means to the dataset, we chose clusters as 2 by using elbow method above;
set.seed(123)
kmeans = kmeans(x = remCorData, centers = 2)
y_kmeans = kmeans$cluster
# For Plotting the clusters;
library(cluster)
clusplot(dataAllNum,
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste('Police Killings'),
         xlab = 'x',
```

ylab = 'y')



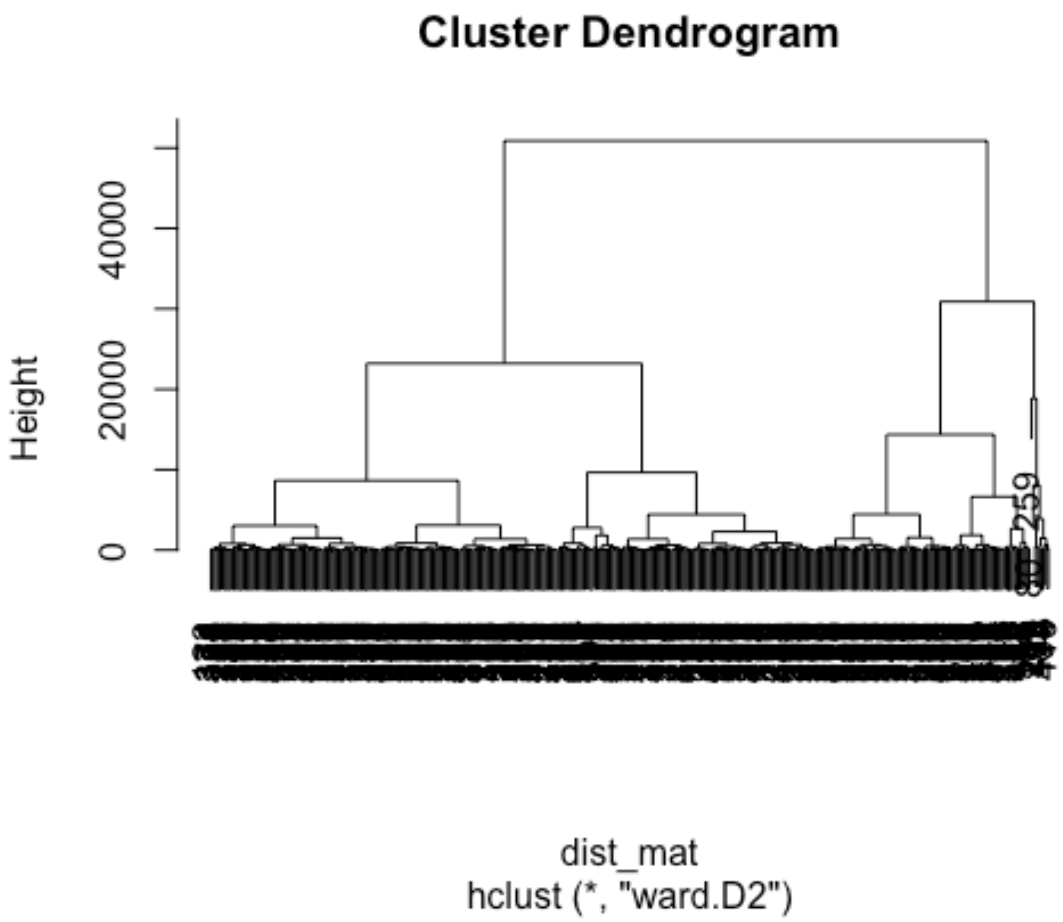
These two components explain 28.53 % of the point variability

We have very bad results with k-means, let's go on with Agglomeration Method;

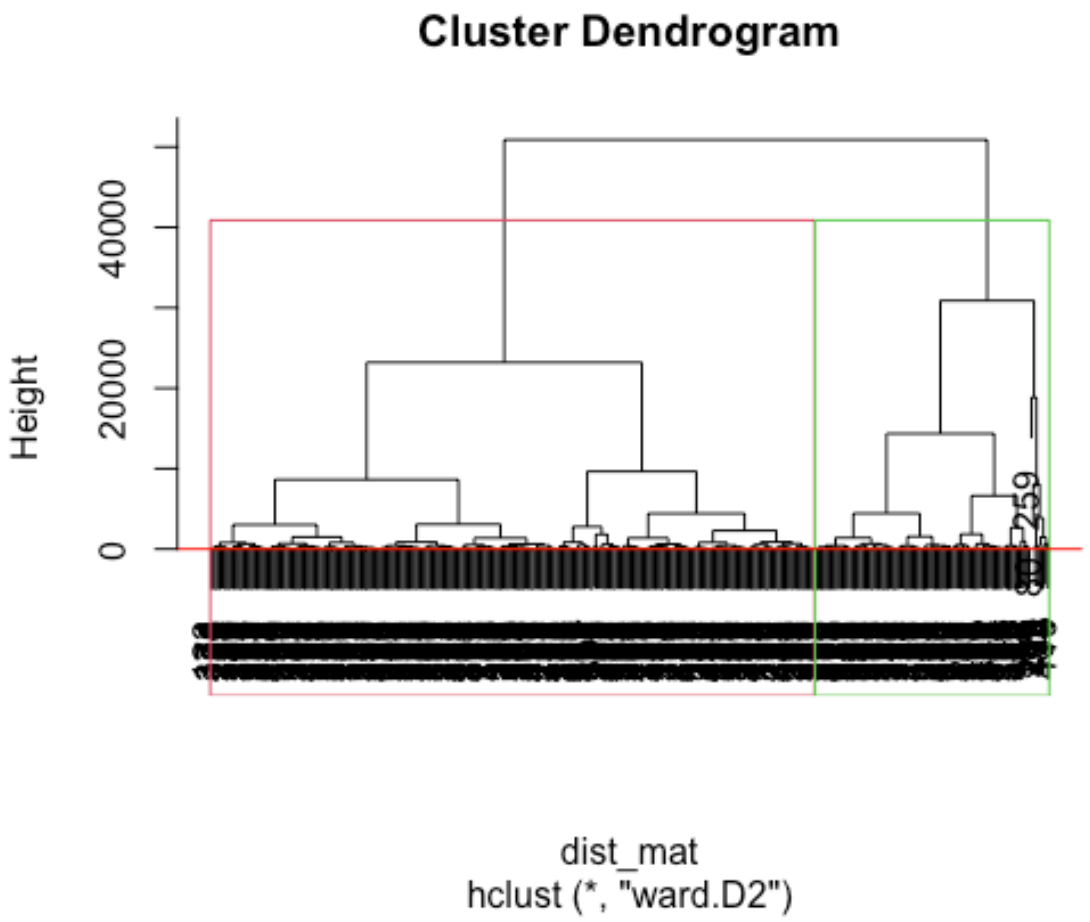
----------------> HIERARCHIAL CLUSTERING WARD'S METHOD
Here the Link means the agglomeration method to be used, we will use ward's method
There are different functions available in R for computing hierarchical clustering. The commonly used functions are:
hclust() and agnes() for agglomerative hierarchical clustering (HC)
hclust() is the built-in R function [in stats package] for computing hierarchical clustering.
The simplified format is:
hclust(d, method = "complete")
The dist() function is used to compute the Euclidean distance between points
METHOD ONE by USING Dissimilartiy Matrix and HCLUST
Dissimilarity matrix
Compute distances and hierarchical clustering
the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of
"ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC).

dist_mat <- dist(remCorData, method = 'euclidean')

```
hclust_avg <- hclust(dist_mat, method = 'ward.D2')
plot(hclust_avg)
```

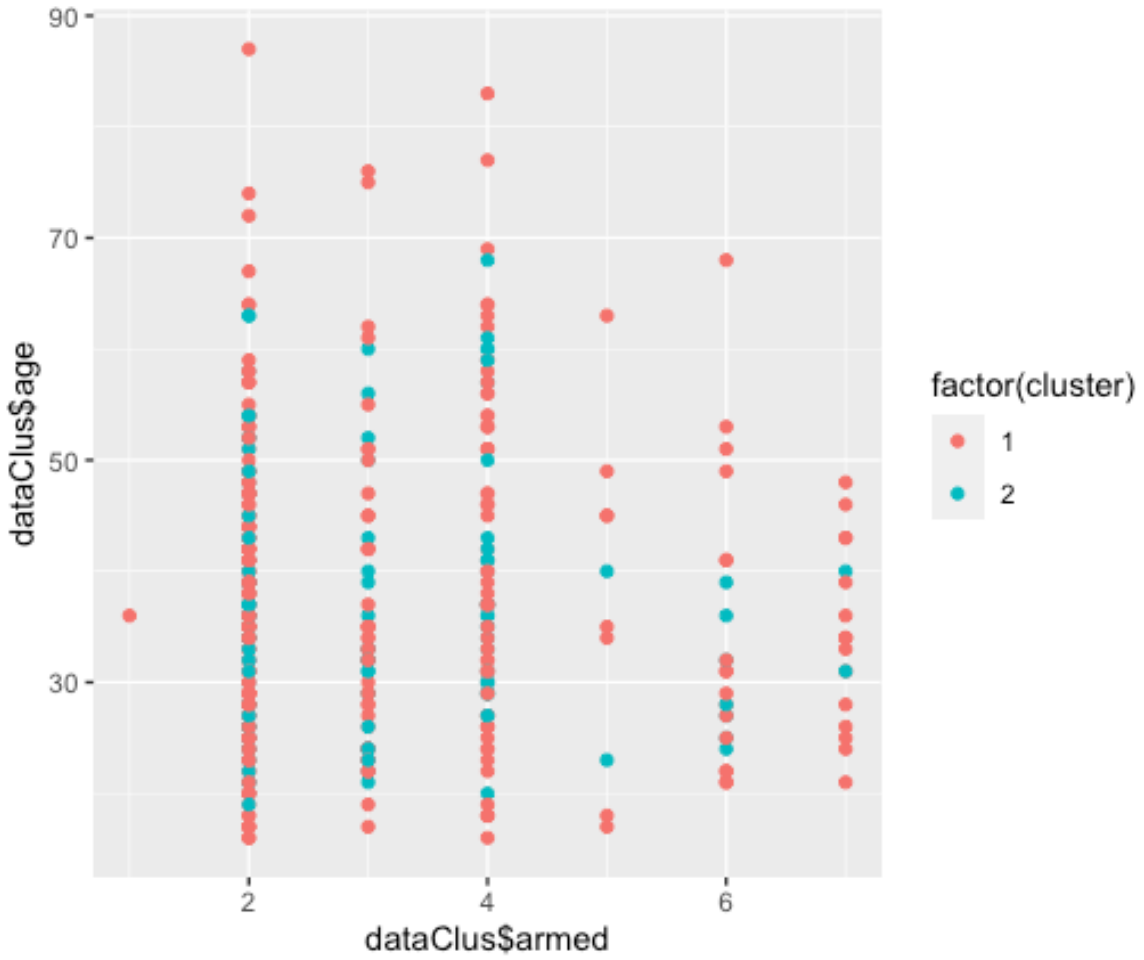



```
# We can cut the dendrogram in order to create the desired number of clusters.
# 2 means that -> ARMED or -> NOT-ARMED

plot(hclust_avg)
rect.hclust(hclust_avg , k = 2, border = 2:6)
abline(h = 2, col = 'red')
```



Now we will append the cluster results obtained back in the original dataframe under column name the cluster with mutate(),
from the dplyr package and count how many observations were assigned to each cluster with the count() function.

cluster n
1 1 317
2 2 123
****
ggplot(dataClus, aes(x=dataClus$armed, y = dataClus$age, color = factor(cluster)))
+ geom_point()




```
table(dataClus$cluster, dataClus$armed)
 1   2   3   4   5   6   7
  1   1 162  44  69   8  17  16
  2   0  58  22  32   2   7   2
```

The train dataset has 288 rows while the test dataset has 96 rows.

```
# ==============> DECISION TREES <==============
# We are ready to build model
install.packages("rpart.plot")
library(rpart)
library(rpart.plot)
# We use the class method because we predict a class. (Data -> TRAIN and Class
Attr -> Cluster)
fit <- rpart(train$cluster~ + train$armed + train$age + train$gender +
train$raceethnicity  + train$city + train$state
             + train$share_black + train$share_hispanic + train$college, data=dataClus,
method = 'class')
rpart.plot(fit, extra = 106)
```

predict_unseen <-predict(fit, test, type = 'class')
Warning message:
'newdata' had 110 rows but variables found have 330 rows