All explorations used crime data from the file uscrime.txt (http://www.statsci.org/data/general/uscrime.txt, description at http://www.statsci.org/data/general/uscrime.html)

Table of Contents

Dealing With Outliers **1**

Predicting Crime Rate With Linear Regression **9**

Principal Component Analysis and Linear Regression to Improve Crime Rate Modeling **13**

Predicting Crime Rate With Regression Trees and Random Forests **18**

Regression Trees 18

Random Forests 22

Regularization Using Stepwise Regression, LASSO, and Elastic Net **24**

Stepwise Regression 24

LASSO 28

Elastic Net 31

Dealing With Outliers in R

```
install.packages("outliers")
```

- Downloading the outliers library

```
library(outliers)
```

- loading the library

```
uscrime <- read.table("uscrime.txt", stringsAsFactors = FALSE,
header = TRUE)
```

- reading in the uscrime data set

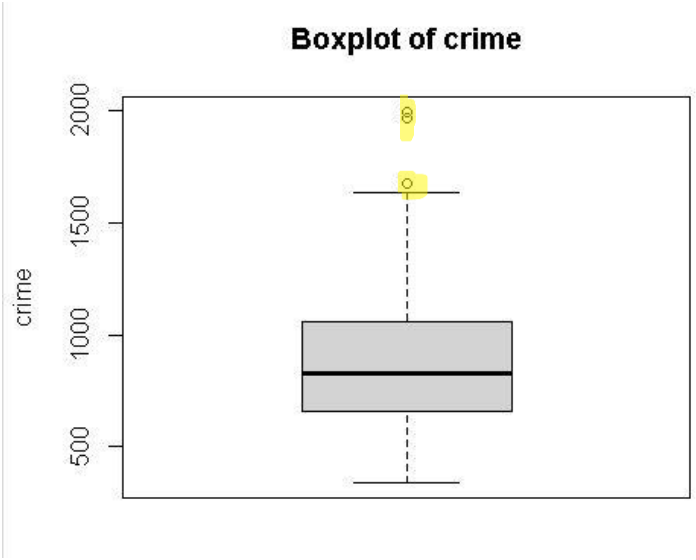
```
View(uscrime)
```

- Viewing the data and noticing that we have a wide range of values in the Crime column ranging from 342 to 1993.

```
crime <- uscrime[,"Crime"]
```

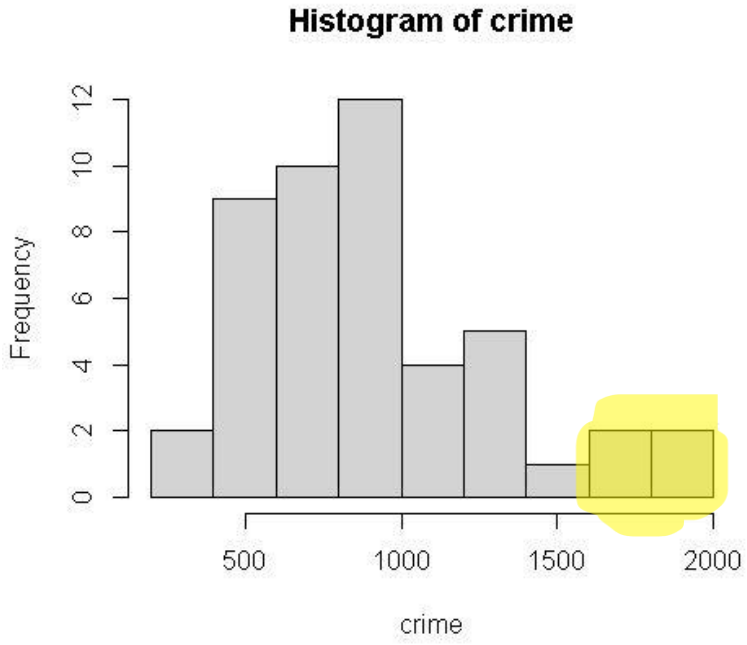
- Creating a data set with just the "Crime" column.

```
boxplot(crime, main = "Boxplot of crime", ylab = "crime")
```



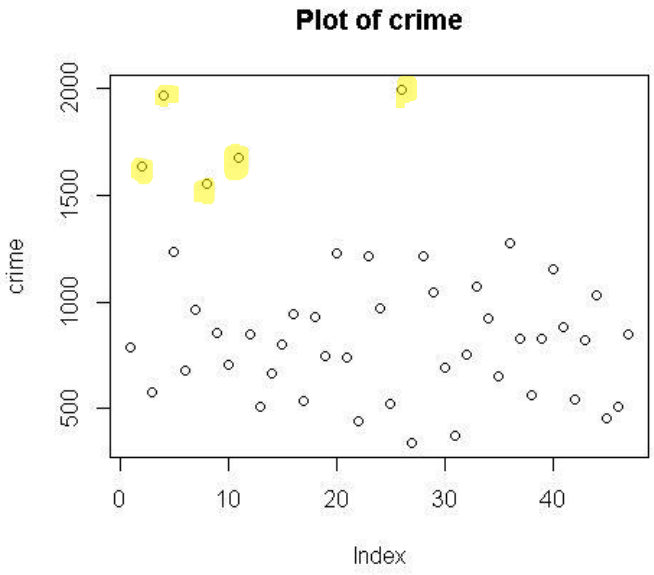
- The highlighted data points are our first indicator of outliers in the data set.

```
hist(crime)
```



- Here is a histogram which shows that the data is skewed to the right meaning the larger values may have some outliers and are causing the data to be more spread out.
- This makes the data less representative of the majority of data points which are on the lower end. We should investigate these values and see if they belong in the data set or if we can remove them.

```
plot(crime, main = "Plot of crime")
```



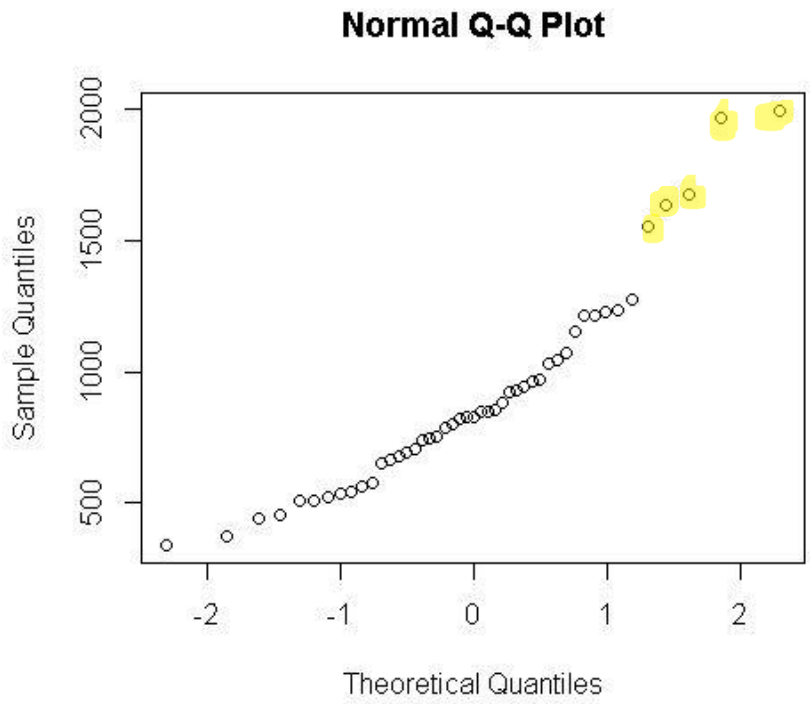
- This is another way of viewing potential outliers. We see that most of the data points are clustered around the lower crime values.

```
shapiro.test(crime)
```

```
        Shapiro-Wilk normality test

data:  crime
W = 0.91273, p-value = 0.001882
```

- The Shapiro-Wilk test is a test to see if a data set follows a normal distribution.
- **W** statistic is on a scale of 0 to 1 with 1 indicating a normal distribution and 0 indicating a non-normal distribution
- **P-value** of <.05 indicates that we reject the null hypothesis and that we do not have a normal distribution.

```
qqnorm(crime)
```



- A QQ plot compares the quantiles of the sample data to the quantiles of expected distribution to show if we have a normal distribution. In a perfect world, we should be able to draw a line through the data points.
- This is showing us that without the outliers, we do have normalish data since we will be able to draw a line through the data.

```
grubbs.test(crime, type = 11)
```

```
        Grubbs test for two opposite outliers

data:  crime
G = 4.26877, U = 0.78103, p-value = 1
alternative hypothesis: 342 and 1993 are outliers
```

- Grubbs test is a test to detect outliers that are significantly different from the rest of the data.
- **Type = 11** is a test for two outliers on opposite tails.
- P-value of 1is telling us to accept the null hypothesis and reject the alternative hypothesis of 342 and 1993 being outliers.

```
grubbs.test(crime, type = 10)
```

```
        Grubbs test for one outlier

data:  crime
G = 2.81287, U = 0.82426, p-value = 0.07887
alternative hypothesis: highest value 1993 is an outlier
```

- Re-running the test with a different type.
- **Type = 10** means a test for one outlier (side is detected automatically and can be reversed by opposite parameter).
- **P-value** of 0.07887 means that we can fairly confidently assume that the 1993 data point is an outlier.

```
grubbs.test(crime, type = 10, opposite = TRUE)
```

```
        Grubbs test for one outlier

data:  crime
G = 1.45589, U = 0.95292, p-value = 1
alternative hypothesis: lowest value 342 is an outlier
```

- Re-running the test with type of 10 to see if we have any outlier that are lower values.
- P-value of 1 indicates that we do not have any.

```
crime2 <- uscrime[-26,"Crime"]
```

- Removing the 1993 data point from our data set

```
grubbs.test(crime2, type = 10)
```

```
        Grubbs test for one outlier

data:  crime2
G = 3.06343, U = 0.78682, p-value = 0.02848
alternative hypothesis: highest value 1969 is an outlier
```

- Re-running the Grubbs test without the 1993 outlier data point
- We now get a p-value of 0.02848 on the 1969 data point which means there is a very high likelihood that the 1969 data point did not occur due to random variation and it should also be removed from the data set.

```
crime3 <- uscrime[(-26,-4),"Crime"]
```

- Removing the 1993 and 1969 data points.

```
grubbs.test(crime3, type = 10)
```

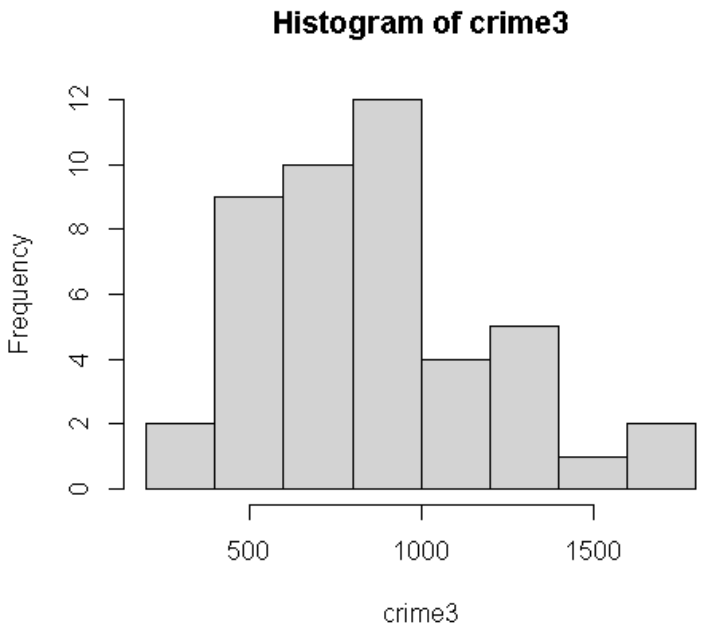
```
        Grubbs test for one outlier

data:  crime3
G = 2.56457, U = 0.84712, p-value = 0.1781
alternative hypothesis: highest value 1674 is an outlier
```

- Re-running the test without the 1993 and 1969 data points to see if we have any more outliers.
- We now ran into the statistically insignificant p-value and this is where we will stop.
- Let's now take a look at the graphs again and see how they've changed with the new crime3 data set.

```
boxplot(crime3, main = "Boxplot for crime3", ylab = "crime")
```

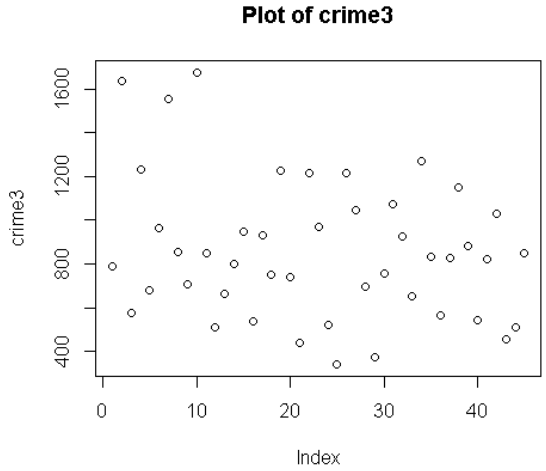
- We now only have a couple values that are just beyond the 75th percentile which is better.

```
hist(crime3)
```



- We got rid of the right skew and the data now has more of a normal distribution.

```
plot(crime3, main = "Plot of crime3")
```



- We no longer see the outlier values and the data points are all clustered together.

```
shapiro.test(crime3)
```

```
        Shapiro-Wilk normality test

data:  crime3
W = 0.95119, p-value = 0.05634
```

- The W statistic went from 0.91273 to 0.95119 gives us numerical evidence that our new crime3 data set has more of a normal distribution.

Predicting Crime Rate With Linear Regression in R

```
rm(list = ls())
```

- clearing the environment

```
uscrime <- read.table("uscrime.txt", stringsAsFactors = FALSE,
header = TRUE)
```

- reading in the uscrime dataset.

```
lm_uscrime <- lm(Crime~., data = uscrime)
```

- **lm()** is the function used to fit a linear model.
- **Crime~.:** the Crime variable is being compared to all of the other variables
- **data = uscrime**: look for all these variables in the uscrime data set.

```
lm(uscrime)
```

```
Call:
lm(formula = uscrime)

Coefficients:
(Intercept)           So           Ed          Po1
 14.5519610    0.3018453   -0.3600756   -0.2041257
        Po2           LF          M.F          Pop
  0.0270975   -3.6377376    0.1101936   -0.0011843
         NW           U1           U2       Wealth
  0.0332034    1.2434625   -0.5410293   -0.0004765
       Ineq         Prob         Time        Crime
 -0.1306275    3.0853787    0.0287017    0.0014245
```

- This shares the strength of relationship between crime and all of the variables in our data set.

```
summary(lm_uscrime)
```

```
Call:
lm(formula = Crime ~ ., data = uscrime)

Residuals:
    Min      1Q  Median      3Q     Max
-395.74  -98.09   -6.69  112.99  512.67

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.984e+03  1.628e+03  -3.675 0.000893 ***
M            8.783e+01  4.171e+01   2.106 0.043443 *
So          -3.803e+00  1.488e+02  -0.026 0.979765
Ed           1.883e+02  6.209e+01   3.033 0.004861 **
Po1          1.928e+02  1.061e+02   1.817 0.078892 .
Po2         -1.094e+02  1.175e+02  -0.931 0.358830
LF          -6.638e+02  1.470e+03  -0.452 0.654654
M.F          1.741e+01  2.035e+01   0.855 0.398995
Pop         -7.330e-01  1.290e+00  -0.568 0.573845
NW           4.204e+00  6.481e+00   0.649 0.521279
U1          -5.827e+03  4.210e+03  -1.384 0.176238
U2           1.678e+02  8.234e+01   2.038 0.050161 .
Wealth       9.617e-02  1.037e-01   0.928 0.360754
Ineq         7.067e+01  2.272e+01   3.111 0.003983 **
Prob        -4.855e+03  2.272e+03  -2.137 0.040627 *
Time        -3.479e+00  7.165e+00  -0.486 0.630708
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 209.1 on 31 degrees of freedom
Multiple R-squared:  0.8031,    Adjusted R-squared:  0.7078
F-statistic: 8.429 on 15 and 31 DF,  p-value: 3.539e-07
```

- This gives us an in-depth view of our data.
- **Residuals** are the difference between the observed and predicted values. It shows how well the model fits the data. Ideally it should be more or less symmetrical with the median close to 0. In our case here, it looks fairly symmetrical but is a little weighted towards larger values. This could potentially mean we have some outliers in the data.
- **Estimate** is the estimated coefficient for each predictor variable
- **Std. error** is the variability of the estimated coefficient. A smaller value means the estimated coefficient is more precise.
- **t-value** is (Estimate / Std. error), a larger value gives us more evidence that we can reject the null hypothesis.
- **Pr(>|t|):** this is the p-value for the t-value. Anything less than 0.05 is a statistically significant predictor in explaining the variation in crime. This helps us to narrow down the model to the best predictors.
- **Multiple R-squared**, our model accounts for 80.31% of the variability in the data. The statistic ranges from 0 to 1 with higher values indicating a better fit of the model. 0.7 or greater indicates a strong relationship so our value of 0.8031 is more than acceptable.

```
test_point <- data.frame(M = 14.0,So = 0, Ed = 10.0, Po1 = 12.0,
Po2 = 15.5, LF = 0.640, M.F = 94.0, Pop = 150, NW = 1.1, U1 =
0.120, U2 = 3.6, Wealth = 3200, Ineq = 20.1, Prob = 0.040, Time
= 39.0)
```

- Creating a data frame to predict the crime rate in a city with the specific attributes listed in the question.

```
pred_model <- predict(lm_uscrime, test_point)
```

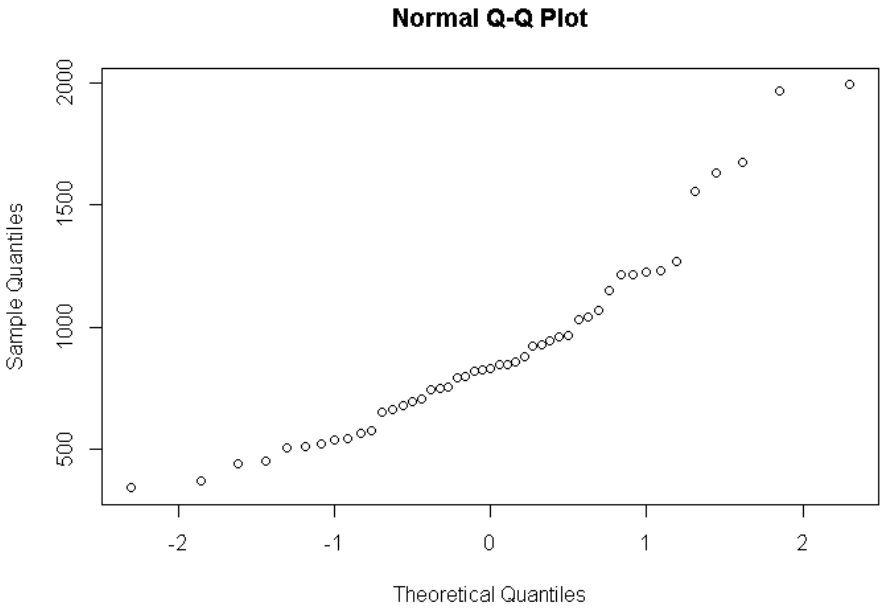
- We use the **predict()** function to make a prediction based on our previously trained model **lm_uscrime** and applying it to the new data points in **test_point.** This will give us a prediction for the crime rate with the specified city attributes.

```
pred_model
```

```
       1
155.4349
```

- Here is our predicted crime value.

```
qqnorm(uscrime$Crime)
```



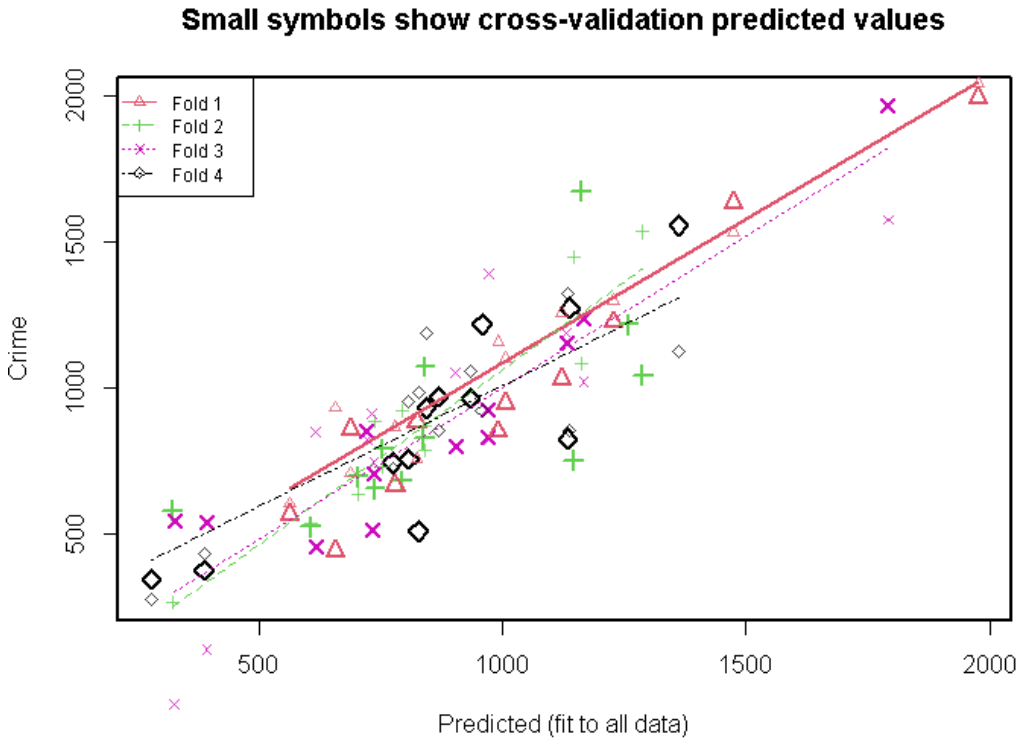
- Here is a Q-Q plot which shows the distribution of our data set to the theoretical distribution. The data point fall roughly in a straight line which tells that our data follows a normal distribution.
- Also, notice the crime values which means we have extremely low crime in our city.

```
install.packages("DAAG")
library(DAAG)
```

- loading the Data Analytics And Graphics library.

```
lm_uscrime_cv <- cv.lm(uscrime, lm_uscrime, m=4)
```

- We are performing cross validation on the linear regression model to show how well the model generalizes to unseen data.
- **cv.lm()** is a function to evaluate a linear model using cross validation.
- **uscrime** is the data set we are using to perform the cross validation
- **lm_uscrime** is the linear regression model we trained on the uscrime data set.
- **m=4** is the number of folds in the k-fold cross validation. The model will be trained and tested 4 times.



- Each shape in this graph shows a different fold.
- These data values create a fairly straight line which again shows us that we have a good quality of fit for our model.

Using Principal Component Analysis and Linear Regression to Improve Crime Rate Modeling in R

```
rm(list = ls())
```

- clearing environment

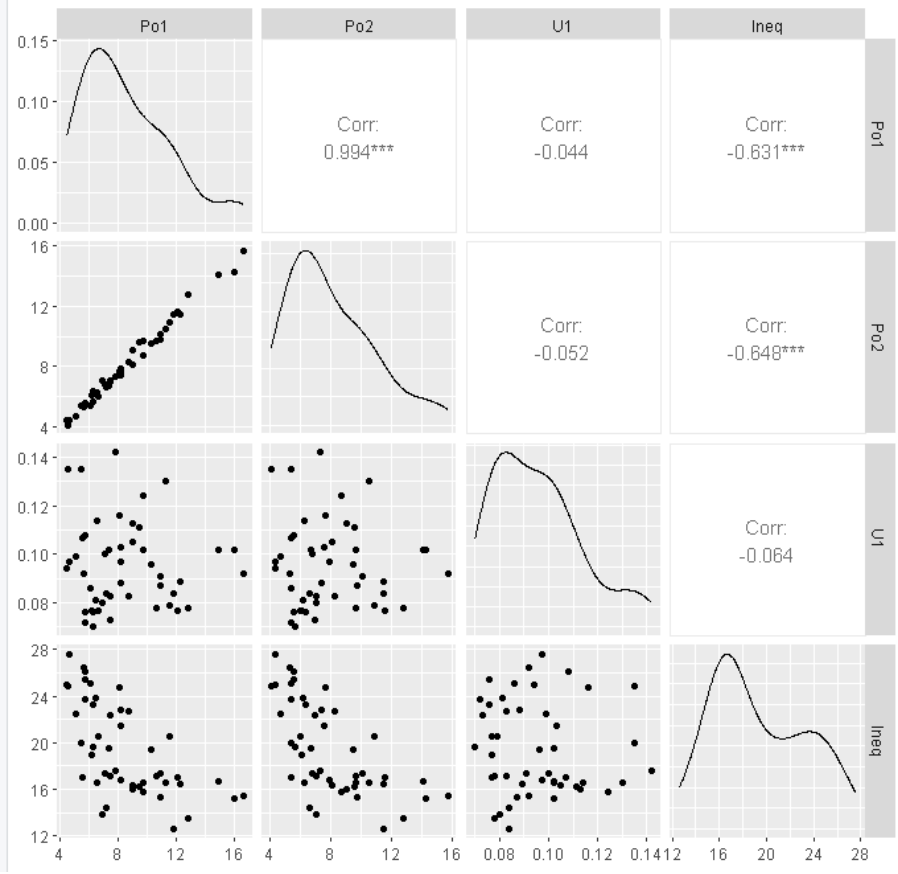
```
uscrime <- read.table("uscrime.txt", stringsAsFactors = FALSE,
header = TRUE)
```

- reading in uscrime data set

```
install.packages("GGally")
library(GGally)
```

- installing and running the "GGally" package which is a good package for correlation visualizations.

```
ggpairs(uscrime, columns = c("Po1", "Po2", "U1", "Ineq"))
```



- **ggpairs():** creates a matrix of scatterplots to visuals the variables in our dataset
- Visually examining the correlation in our data. We see that Po1 & Po2 are heavily correlated. However, just because they are correlated doesn't mean that they caused this relationship. There are numerous other outside factors that could have caused this to happen.

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

- **prcomp():** R function that performs Principal Component Analysis (PCA) transforming our original variables into new uncorrelated variables.
- **scale. = TRUE:** this standardizes each variable in the data centering their means around 0.

```
summary(PCA)
```

```
Importance of components:
                          PC1    PC2    PC3     PC4
Standard deviation     2.4534 1.6739 1.4160 1.07806
Proportion of Variance 0.4013 0.1868 0.1337 0.07748
Cumulative Proportion  0.4013 0.5880 0.7217 0.79920
                           PC5     PC6     PC7     PC8
Standard deviation     0.97893 0.74377 0.56729 0.55444
Proportion of Variance 0.06389 0.03688 0.02145 0.02049
Cumulative Proportion  0.86308 0.89996 0.92142 0.94191
                           PC9    PC10    PC11    PC12
Standard deviation     0.48493 0.44708 0.41915 0.35804
Proportion of Variance 0.01568 0.01333 0.01171 0.00855
Cumulative Proportion  0.95759 0.97091 0.98263 0.99117
                          PC13   PC14    PC15
Standard deviation     0.26333 0.2418 0.06793
Proportion of Variance 0.00462 0.0039 0.00031
Cumulative Proportion  0.99579 0.9997 1.00000
```

- **Standard deviation** tells us how much variance each component captures
- **Proportions of variance** tells us what proportion of the total variance is explained by the principal component.
- **Cumulative Proportion** tells us how much variance is explained by the variables up to and including the current one.

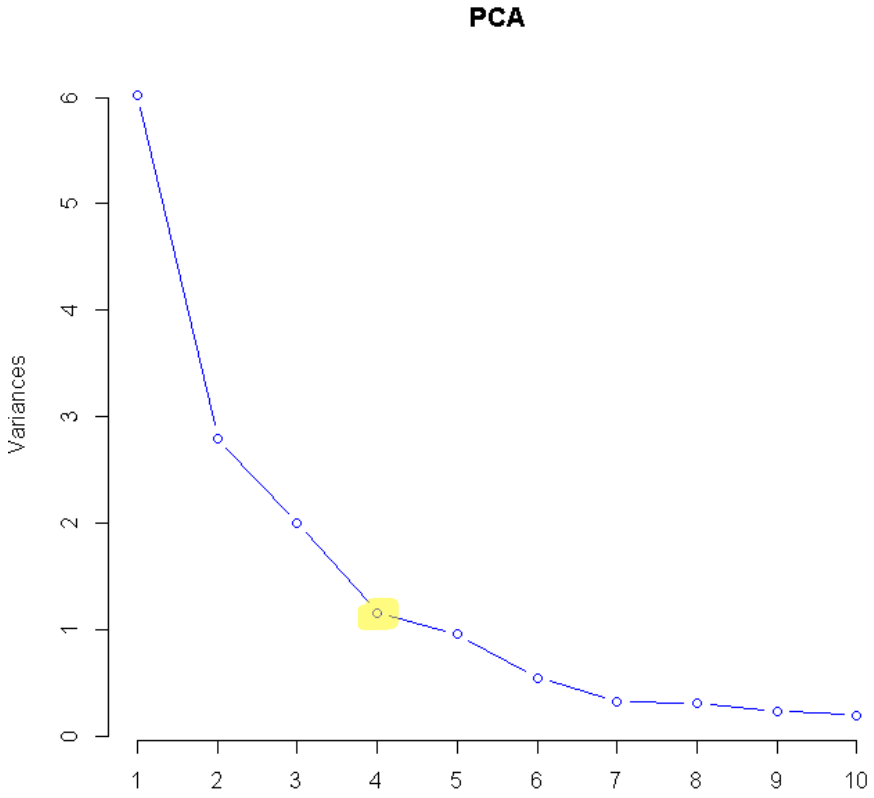
```
PCA$rotation
```

- gives us a rotation matrix or matrix of eigenvectors.

```
                PC1         PC2           PC3         PC4
M      -0.30371194  0.06280357  0.1724199946 -0.02035537
So     -0.33088129 -0.15837219  0.0155433104  0.29247181
Ed      0.33962148  0.21461152  0.0677396249  0.07974375
Po1     0.30863412 -0.26981761  0.0506458161  0.33325059
Po2     0.31099285 -0.26396300  0.0530651173  0.35192809
LF      0.17617757  0.31943042  0.2715301768 -0.14326529
M.F     0.11638221  0.39434428 -0.2031621598  0.01048029
Pop     0.11307836 -0.46723456  0.0770210971 -0.03210513
NW     -0.29358647 -0.22801119  0.0788156621  0.23925971
U1      0.04050137  0.00807439 -0.6590290980 -0.18279096
U2      0.01812228 -0.27971336 -0.5785006293 -0.06889312
Wealth  0.37970331 -0.07718862  0.0100647664  0.11781752
Ineq   -0.36579778 -0.02752240 -0.0002944563 -0.08066612
Prob   -0.25888661  0.15831708 -0.1176726436  0.49303389
Time   -0.02062867 -0.38014836  0.2235664632 -0.54059002
                PC5          PC6         PC7          PC8
M      -0.35832737 -0.449132706 -0.15707378 -0.55367691
So     -0.12061130 -0.100500743  0.19649727  0.22734157
Ed     -0.02442839 -0.008571367 -0.23943629 -0.14644678
Po1    -0.23527680 -0.095776709  0.08011735  0.04613156
Po2    -0.20473383 -0.119524780  0.09518288  0.03168720
LF     -0.39407588  0.504234275 -0.15931612  0.25513777
M.F    -0.57877443 -0.074501901  0.15548197 -0.05507254
Pop    -0.08317034  0.547098563  0.09046187 -0.59078221
NW     -0.36079387  0.051219538 -0.31154195  0.20432828
```

- We can see our new data and corresponding coefficients.

```
screeplot(PCA, type="lines",col="blue")
```



- **screeplot()** gives us a graph that shows the variance explained as we add more variables. In our data, we see that we get less and less explanation of variance as we add more variables. This can help us decide on the number of principal components. 4 looks like a good number where the explanation of variance starts to level out.

```
PC <- PCA$x[,1:4]
```

- Grabbing the first 4 principal components

```
PC
```

```
              PC1         PC2         PC3         PC4
 [1,] -4.1992835 -1.09383120 -1.11907395  0.67178115
 [2,]  1.1726630  0.67701360 -0.05244634 -0.08350709
 [3,] -4.1737248  0.27677501 -0.37107658  0.37793995
 [4,]  3.8349617 -2.57690596  0.22793998  0.38262331
 [5,]  1.8392999  1.33098564  1.27882805  0.71814305
 [6,]  2.9072336 -0.33054213  0.53288181  1.22140635
 [7,]  0.2457752 -0.07362562 -0.90742064  1.13685873
 [8,] -0.1301330 -1.35985577  0.59753132  1.44045387
 [9,] -3.6103169 -0.68621008  1.28372246  0.55171150
[10,]  1.1672376  3.03207033  0.37984502 -0.28887026
[11,]  2.5384879 -2.66771358  1.54424656 -0.87671210
[12,]  1.0065920 -0.06044849  1.18861346 -1.31261964
[13,]  0.5161143  0.97485189  1.83351610 -1.59117618
[14,]  0.4265556  1.85044812  1.02893477 -0.07789173
```

- Now we only have PC1 through PC4.
- Now, we are going to build a linear regression model with these 4 principal components.

```
uscrimePC <- cbind(PC, uscrime[,16])
```

- bind the crime column back on so we have something to predict.

```
modelPCA <- lm(V5~., data = as.data.frame(uscrimePC))
```

- training our PCA component model basically in the same way we would train a regular model.

```
summary(modelPCA)
```

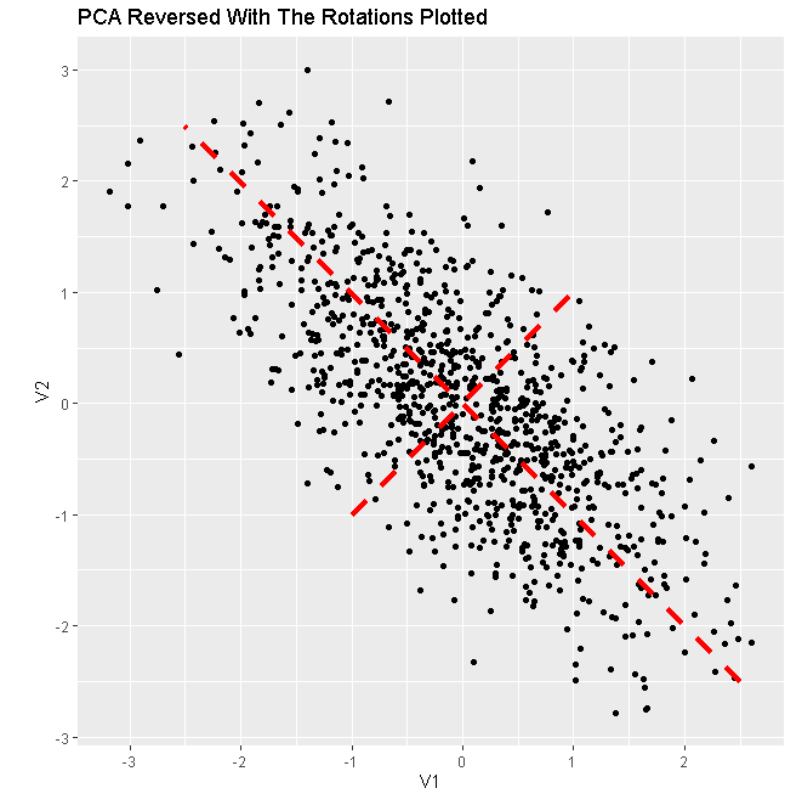
```
Call:
lm(formula = V5 ~ ., data = as.data.frame(uscrimePC))

Residuals:
    Min      1Q  Median      3Q     Max
-557.76 -210.91  -29.08  197.26  810.35

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   905.09      49.07  18.443  < 2e-16 ***
PC1            65.22      20.22   3.225  0.00244 **
PC2           -70.08      29.63  -2.365  0.02273 *
PC3            25.19      35.03   0.719  0.47602
PC4            69.45      46.01   1.509  0.13872
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 336.4 on 42 degrees of freedom
Multiple R-squared:  0.3091,    Adjusted R-squared:  0.2433
F-statistic: 4.698 on 4 and 42 DF,  p-value: 0.003178
```

- Our data seems to be pretty well distributed based on the min, 1Q, median, 3Q, and max. However the difference between min and max shows us that the data being pulled a bit by that max value.
- The p-values are large because our goal with our principal components is to maximize the explanation of variance.

```
reversed_df <- as.data.frame(PCA$x %*% t(PCA$rotation))
```

- This code is using a matrix multiplication to convert back to our original data.
- PCA$x is the transformed data in terms of the principal components
- %*% is the matrix multiplier
- t(PCA$rotation) is the transpose of the rotation matrix where each row corresponds with a principal component now. This is so the PCA$x can correspond with each principal component.

```
reversed_plot <- ggplot(reversed_df, aes(x= V1, y= V2)) +
geom_point() + coord_fixed() + ggtitle(“PCA Reversed With The
Rotations Plotted”)

reversed_plot + geom_segment(x = 2.5, y = -2.5, xend = -2.5,
yend = 2.5, colour = "red", linetype="dashed", size = 1.5) +
geom_segment(x = -1, y = -1, xend = 1, yend = 1, colour = "red",
linetype="dashed", size = 1.5)
```



- Here is the resulting graph showing the quality of fit.

Predicting Crime Rate Using Regression Trees and Random Forests in R

Regression Tree Model

```
rm(list = ls())
```

- Clearing environment

```
uscrime <- read.table("uscrime.txt", stringsAsFactors = FALSE, header = TRUE)
```

- Read in uscrime dataset

```
install.packages("tree")
```

- This is my first time using this package, so this code is to install it.

```
library(tree)
```

- Reading in the package

```
uscrime_tree <- tree(Crime~., data = uscrime)
```

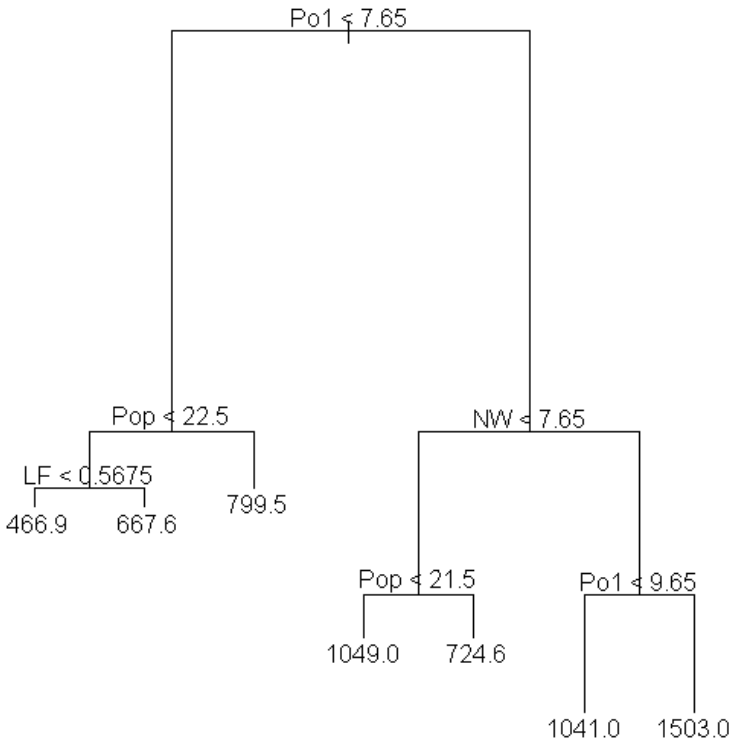
- This code is to create our first decision tree.
- **tree():** is the function we use to create a decision tree.
- **Crime~.** is the dependent or target variable that we are attempting to predict.
    - The **~.** indicates that we are modeling Crime against all the other variables in the data set.

```
summary(uscrime_tree)
```

```
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.
-573.900  -98.300   -1.545    0.000  110.600
    Max.
 490.100
```

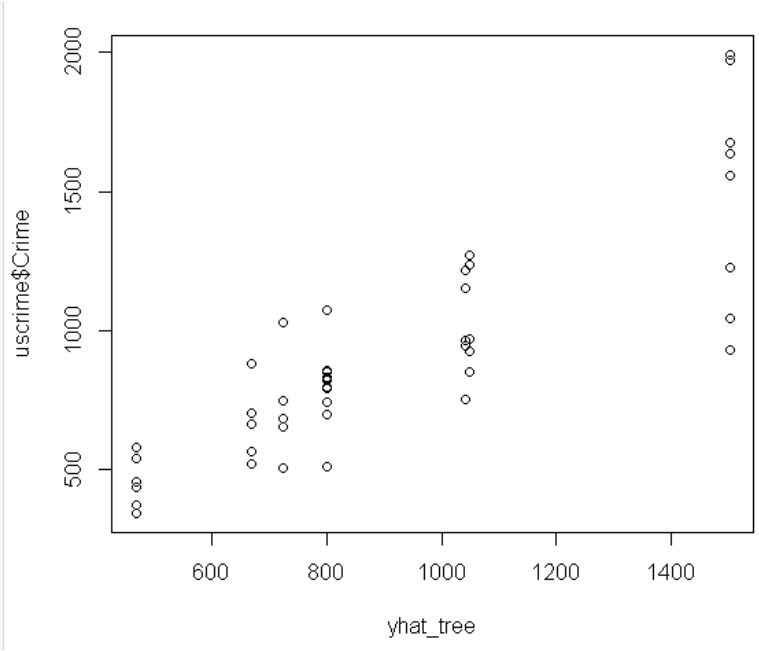
- The tree determined that **"Po1", "Pop", "LF", and "NW"** were the most relevant predictors in explaining the variation in Crime.
- **Number of terminal nodes: 7** , there are 7 leaves or 7 distinct groups of the data where the tree has been divided based on the predictor values.
- **Residual mean deviance** is a goodness of fit metric and how well the tree fits the data.
- **Distribution of residuals** shows that the data is very balanced with the median being close to 0 and min being somewhat close to max and 1st Qu being somewhat close to 3rd Qu.

```
plot(uscrime_tree)
text(uscrime_tree)
```



- Here is our regression tree and it shows what our summary was telling us. We have 7 leaves or 7 distinct groups. Each group has a number which is a prediction of the crime for that group. However, we are only using the averages to make these predictions. We can do better.

```
yhat_tree <- predict(uscrime_tree)
plot(yhat_tree,uscrime$Crime)
```



- We called the predict function to compare the predicted values to the actual values.

```
prune.tree(uscrime_tree)$size
prune.tree(uscrime_tree)$dev
```

- We are using these functions to make the tree smaller and lower the error of our tree.

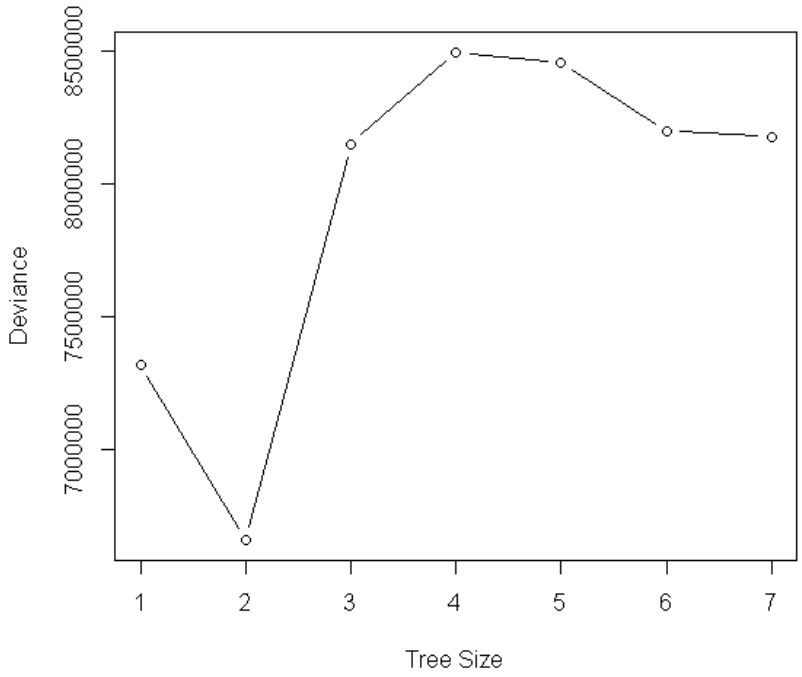
```
set.seed(916)
```

- setting seed for reproducability

```
cv_tree <- cv.tree(uscrime_tree)
```

- This performs a cross validation of deviances across different tree sizes. This can help us to select the best tree size.

```
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "Tree Size",
ylab = "Deviance")
```

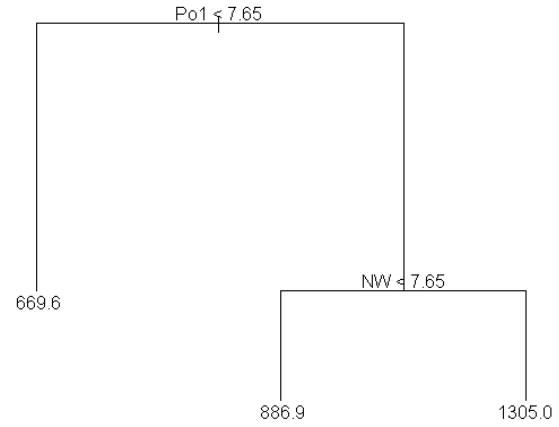


- After plotting the tree size vs deviance, our tree size with the lowest error is 2. However, this value may underfit the data and picking a value like 7 will overfit the data. It is best to look for the elbow point which in our case is 3.

```
uscrime_tree_prune <- prune.tree(uscrime_tree, best = 3)
```

- Pruning the tree to have the 2 best predictors

```
plot(uscrime_tree_prune)
text(uscrime_tree_prune)
```



- Here is our best regression tree model

Random Forests

```
rm(list = ls())
```

- Clearing the environment

```
uscrime <- read.table("uscrime.txt", stringsAsFactors = FALSE,
header = TRUE)
```

- Reading back in our uscrime data

```
install.packages("randomForest")
```

- Downloading the random forest package.

```
library(randomForest)
```

- Reading in the random forest package

```
set.seed(916)
```

- Setting seed for reproducability

```
num_pred <- 4
```

- This will be the number of variables randomly sampled as candidates at each split.

```
uscrime_rf <- randomForest(Crime~., data = uscrime, mtry =
num_pred, importance = TRUE, ntree = 500)
```

- randomForest(): we are making a random forest which builds multiple decision trees and averages their predictions to make a final prediction.
- Crime~.: same as before, we are taking Crime as a target variable and setting it against all other variables
- mtry = num_pred: mtry is what we set in the previous code. It specifies how many predictors should be randomly selected for each split in each tree.
- importance = TRUE: the function will return how much each variable contributes to reducing the prediction error.
- ntree = 500: we will create 500 decision trees

```
uscrime_rf
```

```
Call:
 randomForest(formula = Crime ~ ., data = uscrim
e, mtry = num_pred,      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: 85087.35
                    % Var explained: 41.88
```

- Our random forest has a deviance of 85087 and explains 41.88% of the variance.

```
importance(uscrime_rf)
```

```
           %IncMSE IncNodePurity
M       1.54681882     199647.99
So      2.49245171      30888.86
Ed      3.28835742     231603.72
Po1    11.81482540    1114567.09
Po2    10.12051153    1198329.84
LF      3.62911533     275827.14
M.F     0.09867849     289663.84
Pop     1.74220677     337725.34
NW      8.96023331     480229.68
U1      2.23705423     154275.85
U2      3.36429327     227188.09
Wealth  4.73923679     604762.32
Ineq    1.94617687     247079.83
Prob    8.62039012     700084.20
Time    2.08321245     226457.50
```

- **%IncMSE** or Percentage Increase in Mean Squared Error shows how important the variable is. The greater the value, the more important it is.
- **IncNodePurity** or Increase in Node Purity shows how well the nodes separate the data into distinct classes. A high value means that it has high predictive power and vise versa.

Regularization Using Stepwise Regression, LASSO, and Elastic Net in R

Stepwise Regression

```
rm(list = ls())
```

- Clearing the environment

```
uscrime <- read.table("uscrime.txt", stringsAsFactors = FALSE, header = TRUE)
```

- Reading in the uscrime dataset

```
model_back <- lm(Crime~., data = uscrime)
```

- Creating a linear model using Crime as the dependent variable and all other columns as the independent variables.

```
step(model_back, direction = "backward")
```

- **step()** is a stepwise regression which is a method that automatically selects the most relevant predictors. It starts with a full model then looks for additions or removals to the model at each step.
- **model_back**: feeding in our linear model that we just created
- **direction = "backward":** backwards stepwise regression means that we will start with the full model and remove the least significant variables one at a time based on their p-value or AIC until we get to a simpler model.

```
Start:  AIC=514.65
Crime ~ M + So + Ed + Po1 + Po2 + LF + M.F + Pop + NW + U1 +
    U2 + Wealth + Ineq + Prob + Time

          Df Sum of Sq     RSS    AIC
- So       1        29 1354974 512.65
- LF       1      8917 1363862 512.96
- Time     1     10304 1365250 513.00
- Pop      1     14122 1369068 513.14
- NW       1     18395 1373341 513.28
- M.F      1     31967 1386913 513.74
- Wealth   1     37613 1392558 513.94
- Po2      1     37919 1392865 513.95
<none>                1354946 514.65
- U1       1     83722 1438668 515.47
- Po1      1    144306 1499252 517.41
- U2       1    181536 1536482 518.56
- M        1    193770 1548716 518.93
- Prob     1    199538 1554484 519.11
- Ed       1    402117 1757063 524.86
- Ineq     1    423031 1777977 525.42
```

- Our model starts with an AIC of 514.65

```
Step:  AIC=503.93
Crime ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq + Prob

       Df Sum of Sq     RSS    AIC
<none>              1453068 503.93
- M.F   1    103159 1556227 505.16
- U1    1    127044 1580112 505.87
- Prob  1    247978 1701046 509.34
- U2    1    255443 1708511 509.55
- M     1    296790 1749858 510.67
- Ed    1    445788 1898855 514.51
- Ineq  1    738244 2191312 521.24
- Po1   1   1672038 3125105 537.93
```

- It performed several steps and we ended with a simpler model with fewer variables and an AIC of 503.93.

```
step(model_back, direction = "backward", trace = 0)
```

- **trace = 0**: runs the same thing as the previous line of code, but without printing every step.

```
model_forward <- lm(Crime~1, data = uscrime)
```

- This time we are creating a linear model and starting with just the intercept denoted by the **~1**.

```
step(model_forward, scope = formula(lm(Crime~., data =
uscrime)), direction  = "forward")
```

- This code performs a forward stepwise regression where we iteratively add predictors and slowly improve the model by deciding to keep them using criteria like AIC as we saw in stepwise backwards regression.
- The scope argument tells the model to consider the linear model with every predictor variable in it and then begin applying it in our forward model.

```
Start:  AIC=561.02
Crime ~ 1

          Df Sum of Sq     RSS    AIC
+ Po1      1   3253302 3627626 532.94
+ Po2      1   3058626 3822302 535.39
+ Wealth   1   1340152 5540775 552.84
+ Prob     1   1257075 5623853 553.54
+ Pop      1    783660 6097267 557.34
+ Ed       1    717146 6163781 557.85
+ M.F      1    314867 6566061 560.82
<none>                6880928 561.02
+ LF       1    245446 6635482 561.32
+ Ineq     1    220530 6660397 561.49
+ U2       1    216354 6664573 561.52
+ Time     1    154545 6726383 561.96
+ So       1     56527 6824400 562.64
+ M        1     55084 6825844 562.65
+ U1       1     17533 6863395 562.90
+ NW       1      7312 6873615 562.97
```

- The forward model starts with a higher AIC at 561.02 since we started with just the intercept.

```
Step:  AIC=504.79
Crime ~ Po1 + Ineq + Ed + M + Prob + U2

          Df Sum of Sq     RSS    AIC
<none>                1611057 504.79
+ Wealth   1     59910 1551147 505.00
+ U1       1     54830 1556227 505.16
+ Pop      1     51320 1559737 505.26
+ M.F      1     30945 1580112 505.87
+ Po2      1     25017 1586040 506.05
+ So       1     17958 1593098 506.26
+ LF       1     13179 1597878 506.40
+ Time     1      7159 1603898 506.58
+ NW       1       359 1610698 506.78

Call:
lm(formula = Crime ~ Po1 + Ineq + Ed + M + Prob + U2, data = uscrime)

Coefficients:
(Intercept)          Po1         Ineq           Ed            M
   -5040.50       115.02        67.65       196.47       105.02
       Prob           U2
   -3801.84        89.37
```

- We finished with an AIC of 504.79 and with fewer variables than backwards regression.

```
model_both <- lm(Crime~., data = uscrime)
```

- Creating a linear model in both directions and all of the variables in the data set.

```
step(model_both, scope  = list(lower = formula(lm(Crime~1, data
= uscrime)), upper = formula(lm(Crime~., data  = uscrime))),
direction = "both")
```

- Here for the scope, we need to create an upper and lower bound. Lower being the simplest model and upper being the most complex model.

```
Start:  AIC=514.65
Crime ~ M + So + Ed + Po1 + Po2 + LF + M.F + Pop + NW + U1 +
    U2 + Wealth + Ineq + Prob + Time

          Df Sum of Sq     RSS    AIC
- So      1        29 1354974 512.65
- LF      1      8917 1363862 512.96
- Time    1     10304 1365250 513.00
- Pop     1     14122 1369068 513.14
- NW      1     18395 1373341 513.28
- M.F     1     31967 1386913 513.74
- Wealth  1     37613 1392558 513.94
- Po2     1     37919 1392865 513.95
<none>                1354946 514.65
- U1      1     83722 1438668 515.47
- Po1     1    144306 1499252 517.41
- U2      1    181536 1536482 518.56
- M       1    193770 1548716 518.93
- Prob    1    199538 1554484 519.11
- Ed      1    402117 1757063 524.86
- Ineq    1    423031 1777977 525.42
```

```
Step:  AIC=503.93
Crime ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq + Prob

         Df Sum of Sq     RSS    AIC
<none>                 1453068 503.93
+ Wealth  1     26493 1426575 505.07
- M.F     1    103159 1556227 505.16
+ Pop     1     16697 1436371 505.39
+ Po2     1     14148 1438919 505.47
+ So      1      9329 1443739 505.63
+ LF      1      4374 1448694 505.79
+ NW      1      3799 1449269 505.81
+ Time    1      2293 1450775 505.86
- U1      1    127044 1580112 505.87
- Prob    1    247978 1701046 509.34
- U2      1    255443 1708511 509.55
- M       1    296790 1749858 510.67
- Ed      1    445788 1898855 514.51
- Ineq    1    738244 2191312 521.24
- Po1     1   1672038 3125105 537.93

Call:
lm(formula = Crime ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq + Prob,
    data = uscrime)

Coefficients:
(Intercept)            M           Ed          Po1          M.F
   -6426.10        93.32       180.12       102.65        22.34
         U1           U2         Ineq         Prob
   -6086.63       187.35        61.33     -3796.03
```

- With the both model, it is interesting to note that we started and ended with the same AIC and variables as the backwards stepwise regression model. This is perhaps because both models had the same starting point.

LASSO

```
install.packages("glmnet")
```

```
library(glmnet)
```

- Installing and reading in the glmnet package which is used for regularized regression models (regression model where the coefficient estimates are constrained to 0). Specifically, this will be used for Lasso and Elastic Net.

```
set.seed(916)
```

- Setting seed for reproducibility. It is especially important because the cv.glmnet() function uses randomization to make the folds in cross validation.

```
model_lasso <- cv.glmnet(x = as.matrix(uscrime[,-16]), y =
as.matrix(uscrime[,16]), alpha = 1, nfolds = 8, nlambda = 20,
type.measure = "mse", family = "gaussian", standardize = TRUE)
```

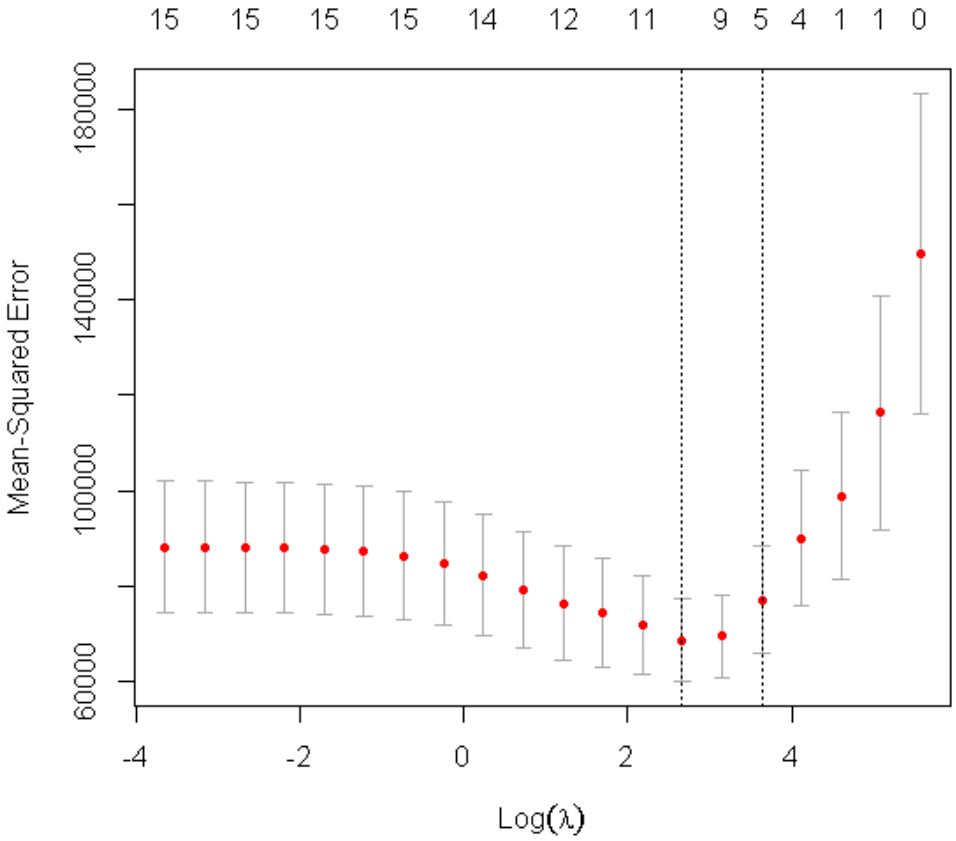
- We are creating a LASSO (Least Absolute Shrinkage and Selection Operator) model which helps select features and improve the accuracy and interpretation of our results.
- **cv.glmnet()** fits regularized regression models (specifically LASSO and Elastic Net) and performs cross validation to find the penalty parameter lambda.
- **x = as.matrix(uscrime[,-16])** : our response variable Crime.
- **alpha = 1** indicates to use LASSO while 0 indicates to use Ridge and everything inbetween is a spectrum of the two.
- **nfolds = 8** number of folds in the cross validation. The data will be split into 8 parts with 7 parts for training and 1 part for validation until all folds have been used for validation.
- **nlambda = 20** number of lambda values (strength of penalty) to be tested during cross validation.
- **type.measure = "mse"** Mean Squared Error will be used to test the model's performance.
- **family = "gaussian":** gaussian is the default family for linear regression in this function.
- **standardize = TRUE:** the predictor variables will be standardized ensuring that all variables are on the same scale.

```
model_lasso
```

```
Call:  cv.glmnet(x = as.matrix(uscrime[, -16]), y = as.matrix(uscrime[,
16]), type.measure = "mse", nfolds = 8, alpha = 1, nlambda = 20,      family
= "gaussian", standardize = TRUE)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min  14.35     7   68623  8667      10
1se  37.84     5   77106 11155       5
```

- The best value of **lambda** is 14.35 that minimizes the mean squared error.
- 14.35 is in the 7th position or **index** of lambda values tested.
- the **measure or MSE** of 68623 is the average squared difference between the model's prediction and the actual values.
- the **SE or standard error** for this value of lambda is 8667. This represents the variability in the mean squared error across the different cross validation folds. A large SE suggest more variability and vise versa.
- **Nonzero** of 10 means that 10 predictor variables remained when all was said and done. These 10 variables are what contributed to our predictions.
- The **1se or 1 standard error** values will give use a model that is 1 standard error away. It will give use worse predictions but offers more simplicity and avoids overfitting. The predictor variables are slashed from 10 down to 5.

```
plot(model_lasso)
```



- The dotted lines represent the 2 lambdas that we were given. The one on the left represents our best performing model in terms of mean squared error. The one on the right is our more conservative and simpler model that can be used to avoid overfitting.

```
coef(model_lasso, s = model_lasso$lambda.min)
```

```
(Intercept) -4188.8210567
M              62.5173281
So             46.3065171
Ed             89.8715021
Po1           105.0118443
Po2             .
LF             80.0219314
M.F            15.7901041
Pop             .
NW              0.2279318
U1              .
U2             35.8812138
Wealth          .
Ineq           41.6921408
Prob        -3404.8060716
Time            .
```

- Here are the predictor values and corresponding coefficients for our optimal lambda min model.

```
coef(model_lasso, s = model_lasso$lambda.1se)
```

```
(Intercept) -1979.87971
M              29.90147
So              .
Ed              .
Po1            89.01607
Po2             .
LF              .
M.F            15.23799
Pop             .
NW              .
U1              .
U2              .
Wealth          .
Ineq           15.63956
Prob        -1855.20705
Time            .
```

- Here are the predictor values and corresponding coefficients for our simpler 1se model.

Elastic Net

```
model_elasticnet <- cv.glmnet(x = as.matrix(uscrime[,-16]), y = as.matrix(uscrime[,16]), alpha = 0.5, nfolds = 8, nlambda = 20, type.measure = "mse", family = "gaussian", standardize = TRUE)
```

- Now we will created an Elastic Net model which has an alpha value between 0 and 1. It is a combination of both LASSO and Ridge. We will run it for all values of alpha in increments of 0.1.

```
model_elasticnet
```

```
    Lambda Index Measure    SE Nonzero
min  10.89     9   68141 18487      14
1se  46.61     6   83739 21286      11
```

- alpha = 0.5

```
    Lambda Index Measure    SE Nonzero
min  33.53    10   71684 16954      13
1se 143.53     7   81012 18756      15
```

- alpha = 0.1

```
    Lambda Index Measure    SE Nonzero
min  27.22     9   68809 12109      13
1se  71.77     7   73210 12145      12
```

- alpha = 0.2

```
    Lambda Index Measure    SE Nonzero
min  29.47     8   73494 17548      13
1se 126.15     5   90011 16333       7
```

- alpha = 0.3

```
    Lambda Index Measure    SE Nonzero
min   5.16    11   71515 13037      13
1se  58.27     6   78702 13556      11
```

- alpha = 0.4

```
    Lambda Index Measure    SE Nonzero
min  5.588    10   67221 10646      12
1se 23.922     7   72416 15563      12
```

- alpha = 0.6

```
    Lambda Index Measure    SE Nonzero
min   4.79    10   83488 10916      12
1se  33.30     6   93191 13885      10
```

- alpha = 0.7

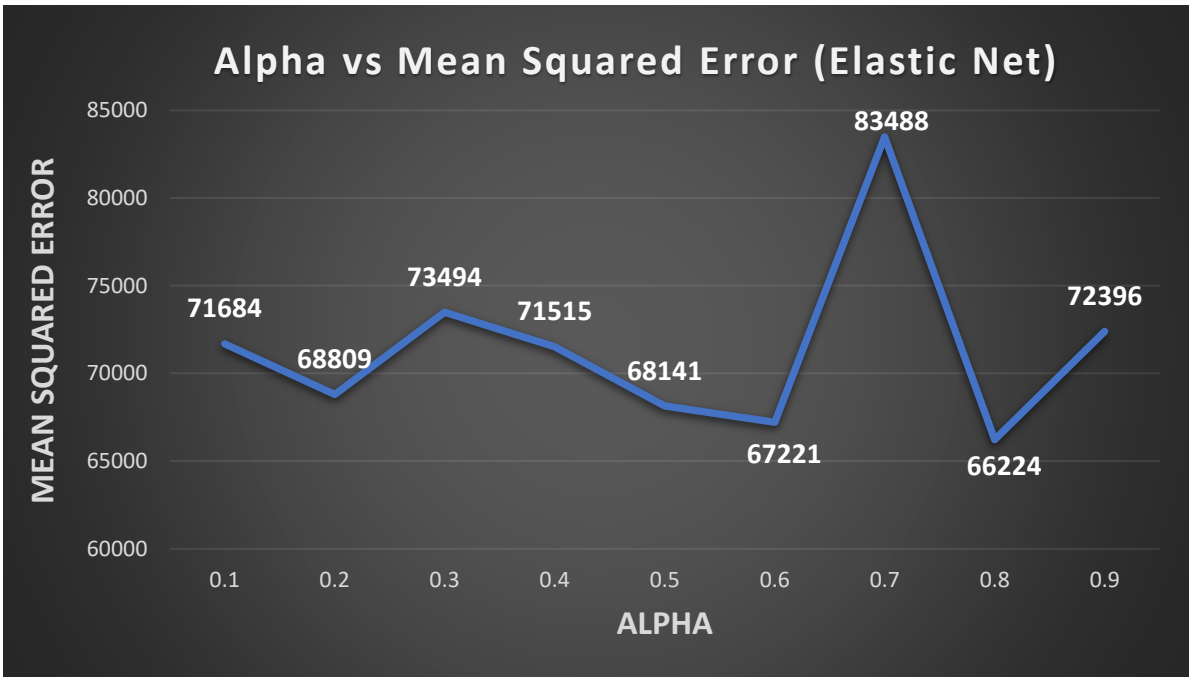
```
    Lambda Index Measure    SE Nonzero
min  6.805     9   66224 11929      12
1se 29.133     6   73467  9483      10
```

- **alpha = 0.8**

```
    Lambda Index Measure    SE Nonzero
min  9.822     8   72396 10066      11
1se 25.896     6   80950 17330       9
```

- alpha = 0.9

Alpha	Mean Squared Error
0.1	71684
0.2	68809
0.3	73494
0.4	71515
0.5	68141
0.6	67221
0.7	83488
0.8	66224
0.9	72396



- Our best Elastic Net model is at an alpha value of 0.8 and it has a mean squared error of 66224 which slightly outperformed the LASSO model.