Homework #3: Binary Logistic Regression Models

Douglas Barley, Ethan Haley, Isabel Magnus, John Mazon, Vinayak Kamath, Arushi Arora

11/1/2021

DATA 621 – Business Analytics and Data Mining

Overview

Homework #3 Assignment Requirements

In this homework assignment, we will explore, analyze and model a data set containing information on crime for various neighborhoods of a major city. Each record has a response variable indicating whether or not the crime rate is above the median crime rate (1) or not (0).

Our objective is to build a binary logistic regression model on the training data set to predict whether the neighborhood will be at risk for high crime levels. We will provide classifications and probabilities for the evaluation data set using our binary logistic regression model. We can only use the variables given to us, as well as variables we derive from those. Below is a short description of the variables provided.

- **zn**: proportion of residential land zoned for large lots
- **indus**: proportion of non-retail business acres
- **chas**: binary indicator of whether the area borders the river
- **nox**: a measurement of nitric oxide in the area
- **rm**: average number of rooms per dwelling
- **age**: proportion of owner-occupied units built pre-1940
- **dis**: weighted distance to 5 employment centers
- **rad**: index of accessibility to radial highways
- **tax**: property tax rate
- **ptratio**: pupil/teacher ratio
- **lstat**: % of the population with "lower status"
- **medv**: median value of owner-occupied homes
- **target**: binary indicator of whether the crime rate is above the median

```
crime_eval_df <- read.csv("https://raw.githubusercontent.com/johnm1990/DATA621/main/hw3/crime-evaluatio
crime_train_df <- read.csv("https://raw.githubusercontent.com/johnm1990/DATA621/main/hw3/crime-training-
```

Load the datasets

1. Data Exploration

We are working with two datasets derived from the Boston dataset: a training dataset with 466 observations of 13 variables and an evaluation dataset with 40 observations of 12 variables. The training data includes a variable for "target", the target variable that the evaluation data set will be used to predict. We begin by exploring the data in order to best understand potential relationships between the independent and dependent variables, which variables are realistic, which variables are normally distributed or require some type of transformation to meet regression assumptions, and which variables have missing observations. First, we examine each variable's summary statistics: their minimum and maximum values, the median and mean, and the first and third quartile values.

```
###summary statistics
summary(crime_train_df)
```

```
##        zn             indus             chas              nox        
##  Min.   :  0.00   Min.   : 0.460   Min.   :0.00000   Min.   :0.3890  
##  1st Qu.:  0.00   1st Qu.: 5.145   1st Qu.:0.00000   1st Qu.:0.4480  
##  Median :  0.00   Median : 9.690   Median :0.00000   Median :0.5380  
##  Mean   : 11.58   Mean   :11.105   Mean   :0.07082   Mean   :0.5543  
##  3rd Qu.: 16.25   3rd Qu.:18.100   3rd Qu.:0.00000   3rd Qu.:0.6240  
##  Max.   :100.00   Max.   :27.740   Max.   :1.00000   Max.   :0.8710  
##        rm             age              dis              rad       
##  Min.   :3.863   Min.   :  2.90   Min.   : 1.130   Min.   : 1.00  
##  1st Qu.:5.887   1st Qu.: 43.88   1st Qu.: 2.101   1st Qu.: 4.00  
##  Median :6.210   Median : 77.15   Median : 3.191   Median : 5.00  
##  Mean   :6.291   Mean   : 68.37   Mean   : 3.796   Mean   : 9.53  
##  3rd Qu.:6.630   3rd Qu.: 94.10   3rd Qu.: 5.215   3rd Qu.:24.00  
##  Max.   :8.780   Max.   :100.00   Max.   :12.127   Max.   :24.00  
##       tax           ptratio         lstat             medv      
##  Min.   :187.0   Min.   :12.6   Min.   : 1.730   Min.   : 5.00  
##  1st Qu.:281.0   1st Qu.:16.9   1st Qu.: 7.043   1st Qu.:17.02  
##  Median :334.5   Median :18.9   Median :11.350   Median :21.20  
##  Mean   :409.5   Mean   :18.4   Mean   :12.631   Mean   :22.59  
##  3rd Qu.:666.0   3rd Qu.:20.2   3rd Qu.:16.930   3rd Qu.:25.00  
##  Max.   :711.0   Max.   :22.0   Max.   :37.970   Max.   :50.00  
##      target      
##  Min.   :0.0000  
##  1st Qu.:0.0000  
##  Median :0.0000  
##  Mean   :0.4914  
##  3rd Qu.:1.0000  
##  Max.   :1.0000  
```

None of the variables have null values, but it's possible that null and unknown values are coded in as zeros. Of the variables with 0 as their minimum observation, both **chas** and **target** are binary coded. The first predictor, **zn**, has a median of 0 and a max of 100. 339 **zn** observations are 0. If we had contextual experience with this area's land zoning, we would be able to gauge whether that's realistic or not, but regardless we'll need to deal with unusual distributions like **zn** in the following section, in order to include them in a linear model. When looking at maximum values, **rm**, **age**, and **rad** are interesting to call out. **rm** has a maximum value of 8.8, meaning that some suburbs have an average of 8.8 rooms per dwelling. **age**, with a max of 100, shows us that at least one suburb's owner-occupied units were all built prior to 1940. **rad** is an index of accessibility to radial highways. In the training set, we see the index spans between 1 and 24, which we can assume are discrete, ordinal values because of the nature of the index (a larger index value indicates greater access). Examining the count of values in "rad", 17 observations have a rank of 1, while 121 observations have a rank of 24.

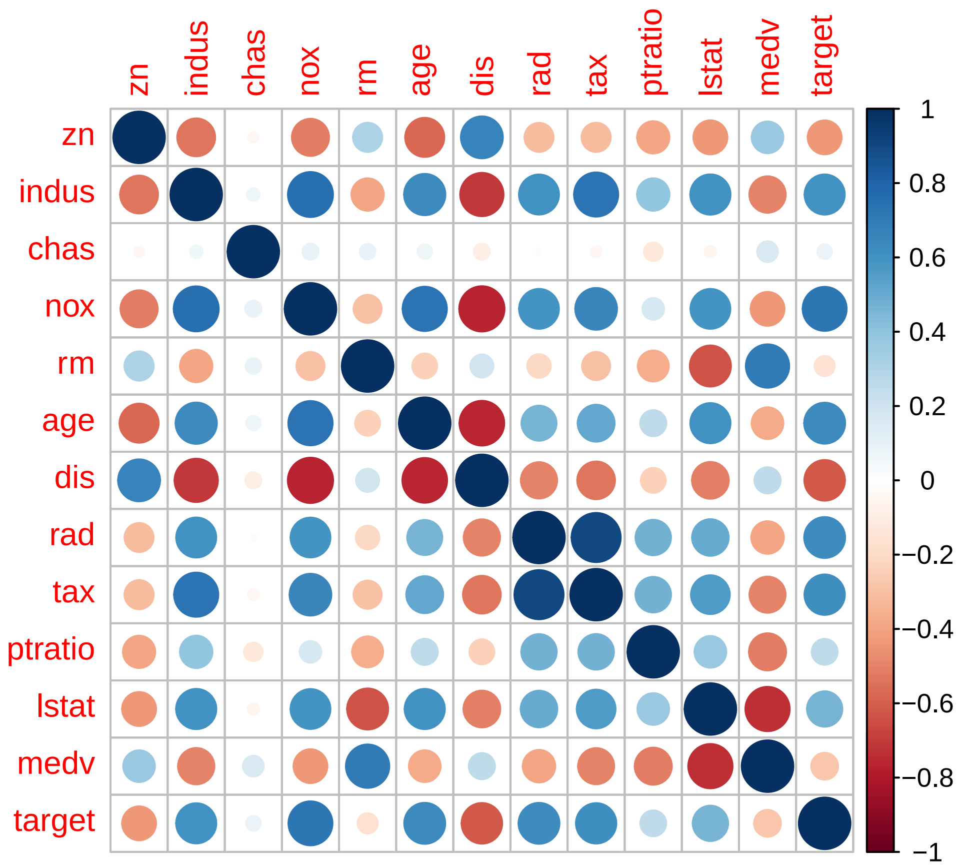
We'll look at how to deal with odd distributions of predictors in the next section, but for now we'll continue with some summary EDA statistics, to get a better overview of things. For instance, how are the variables correlated?

```r
##correlation matrix
crime_train_df.rcorr = rcorr(as.matrix(crime_train_df))
crime_train_df.rcorr
```

```
##            zn indus  chas   nox    rm   age   dis   rad   tax ptratio lstat
## zn       1.00 -0.54 -0.04 -0.52  0.32 -0.57  0.66 -0.32 -0.32   -0.39 -0.43
## indus   -0.54  1.00  0.06  0.76 -0.39  0.64 -0.70  0.60  0.73    0.39  0.61
## chas    -0.04  0.06  1.00  0.10  0.09  0.08 -0.10 -0.02 -0.05   -0.13 -0.05
## nox     -0.52  0.76  0.10  1.00 -0.30  0.74 -0.77  0.60  0.65    0.18  0.60
## rm       0.32 -0.39  0.09 -0.30  1.00 -0.23  0.20 -0.21 -0.30   -0.36 -0.63
## age     -0.57  0.64  0.08  0.74 -0.23  1.00 -0.75  0.46  0.51    0.26  0.61
## dis      0.66 -0.70 -0.10 -0.77  0.20 -0.75  1.00 -0.49 -0.53   -0.23 -0.51
## rad     -0.32  0.60 -0.02  0.60 -0.21  0.46 -0.49  1.00  0.91    0.47  0.50
## tax     -0.32  0.73 -0.05  0.65 -0.30  0.51 -0.53  0.91  1.00    0.47  0.56
## ptratio -0.39  0.39 -0.13  0.18 -0.36  0.26 -0.23  0.47  0.47    1.00  0.38
## lstat   -0.43  0.61 -0.05  0.60 -0.63  0.61 -0.51  0.50  0.56    0.38  1.00
## medv     0.38 -0.50  0.16 -0.43  0.71 -0.38  0.26 -0.40 -0.49   -0.52 -0.74
## target  -0.43  0.60  0.08  0.73 -0.15  0.63 -0.62  0.63  0.61    0.25  0.47
##          medv target
## zn       0.38  -0.43
## indus   -0.50   0.60
## chas     0.16   0.08
## nox     -0.43   0.73
## rm       0.71  -0.15
## age     -0.38   0.63
## dis      0.26  -0.62
## rad     -0.40   0.63
## tax     -0.49   0.61
## ptratio -0.52   0.25
## lstat   -0.74   0.47
## medv     1.00  -0.27
## target  -0.27   1.00
##
## n= 466
##
##
## P
##         zn     indus  chas   nox    rm     age    dis    rad    tax    ptratio
## zn             0.0000 0.3870 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
## indus   0.0000        0.1874 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
## chas    0.3870 0.1874        0.0355 0.0509 0.0890 0.0372 0.7321 0.3138 0.0054
## nox     0.0000 0.0000 0.0355        0.0000 0.0000 0.0000 0.0000 0.0000 0.0001
## rm      0.0000 0.0000 0.0509 0.0000        0.0000 0.0000 0.0000 0.0000 0.0000
## age     0.0000 0.0000 0.0890 0.0000 0.0000        0.0000 0.0000 0.0000 0.0000
## dis     0.0000 0.0000 0.0372 0.0000 0.0000 0.0000        0.0000 0.0000 0.0000
## rad     0.0000 0.0000 0.7321 0.0000 0.0000 0.0000 0.0000        0.0000 0.0000
## tax     0.0000 0.0000 0.3138 0.0000 0.0000 0.0000 0.0000 0.0000        0.0000
## ptratio 0.0000 0.0000 0.0054 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000
## lstat   0.0000 0.0000 0.2679 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
## medv    0.0000 0.0000 0.0005 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
## target  0.0000 0.0000 0.0843 0.0000 0.0010 0.0000 0.0000 0.0000 0.0000 0.0000
##         lstat  medv   target
## zn      0.0000 0.0000 0.0000
## indus   0.0000 0.0000 0.0000
## chas    0.2679 0.0005 0.0843
## nox     0.0000 0.0000 0.0000
## rm      0.0000 0.0000 0.0010
## age     0.0000 0.0000 0.0000
## dis     0.0000 0.0000 0.0000
## rad     0.0000 0.0000 0.0000
## tax     0.0000 0.0000 0.0000
## ptratio 0.0000 0.0000 0.0000
## lstat          0.0000 0.0000
## medv    0.0000        0.0000
## target  0.0000 0.0000
```

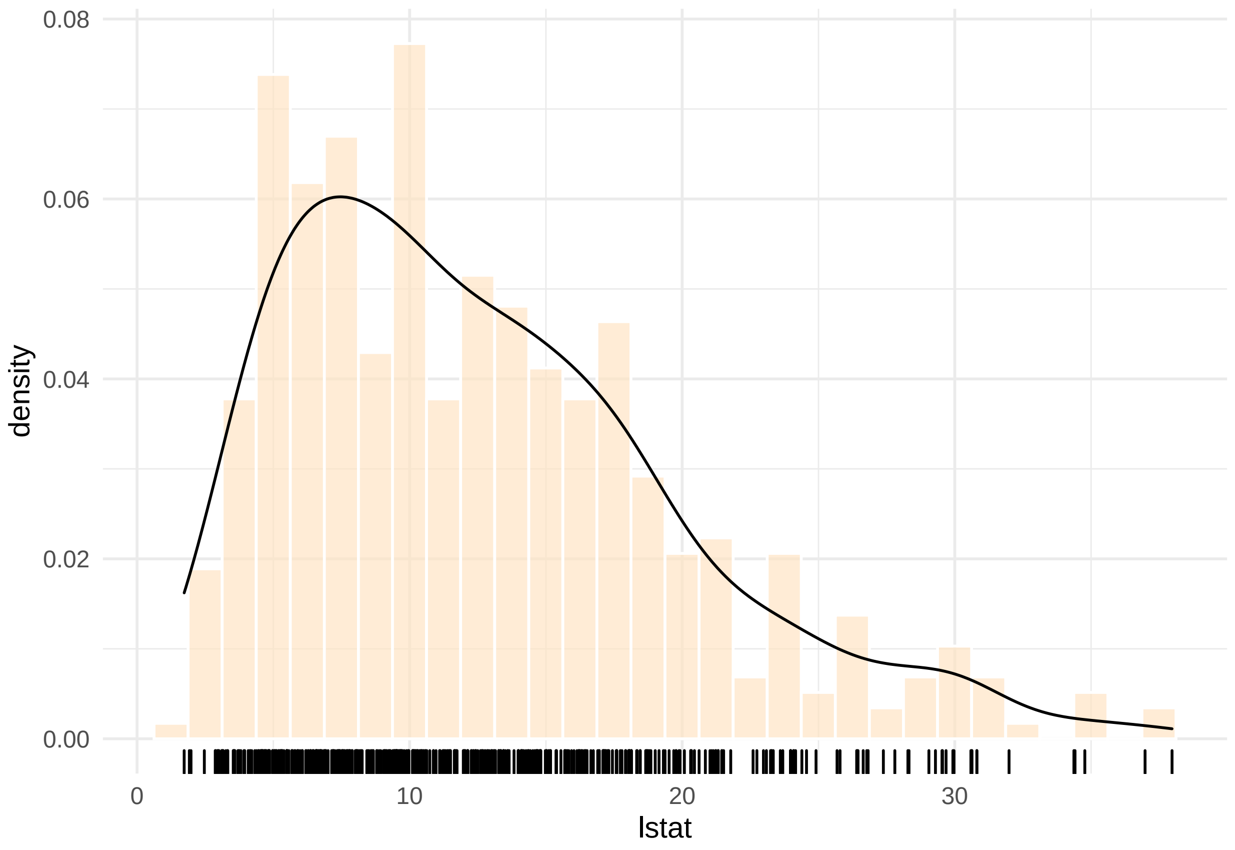
The p-values for **chas** are high, and its correlations are weak.
Below is the same correlation matrix, in a graphical form that's easier to grasp:

```
crime_train_df.cor = cor(crime_train_df)
corrplot(crime_train_df.cor)
```



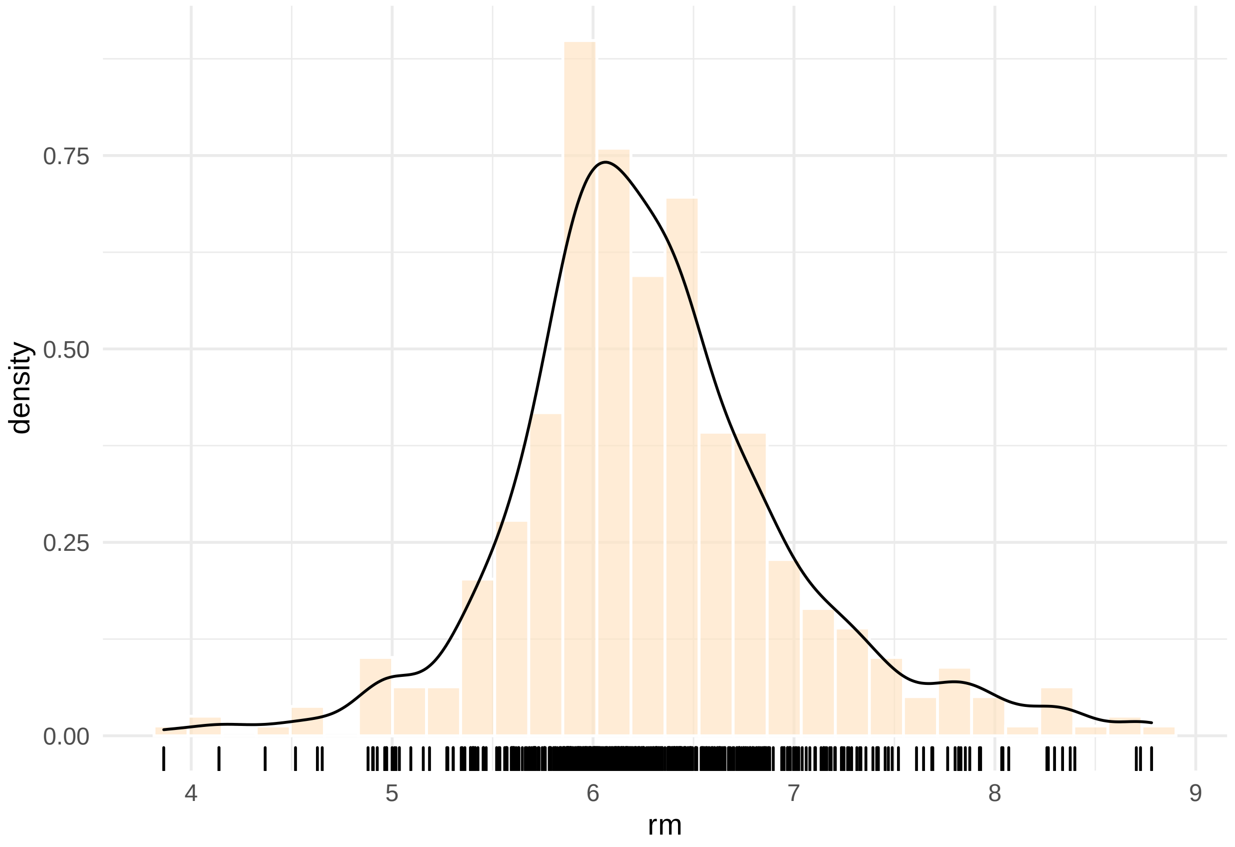
We need to explore the variable distributions in order to prepare the data for modeling. **lstat**, a socioeconomic indicator, is a percentage, meaning it's already scaled and might work best in its original form. Here's its distribution:

```
ggplot(data = crime_train_df, mapping = aes(lstat)) +
  geom_histogram(aes(y=..density..),fill="bisque",color="white",alpha=0.7,bins=30) +
  geom_density() + geom_rug() + labs(x='lstat') + theme_minimal()
```

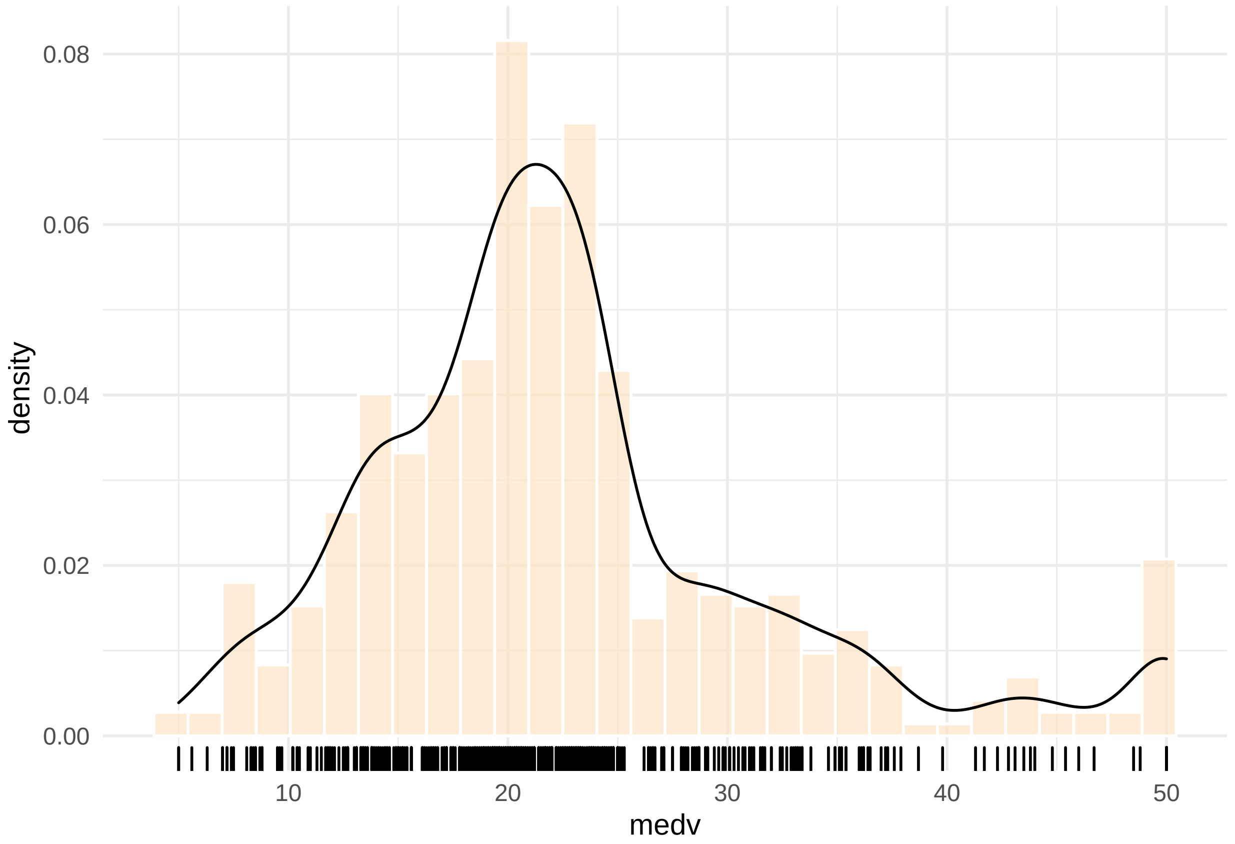


Both **medv** and **rm** are also relatively normally distributed and might be left alone for modeling. The other variables violate regression assumptions and should perform better when transformed.

```
ggplot(data = crime_train_df, mapping = aes(rm)) +
  geom_histogram(aes(y=..density..),fill="bisque",color="white",alpha=0.7,bins=30) +
  geom_density() + geom_rug() + labs(x='rm') + theme_minimal()
```



```
ggplot(data = crime_train_df, mapping = aes(medv)) +
  geom_histogram(aes(y=..density..),fill="bisque",color="white",alpha=0.7,bins=30) +
  geom_density() + geom_rug() + labs(x='medv') + theme_minimal()
```



The unusally high bar at the right of this **medv** distribution may warrant further attention.

Another part of our exploration involves examining correlation between variables because multicollinearity can pose challenges for regression models. The **dis** variable, a weighted mean of distances to five Boston employment centers, is strongly negatively correlated with **indus**, **age**, and the target variable (-0.618). **medv** and **lstat** are also negatively correlated, meaning that as the median value of owner-occupied homes increases, the percentage of the population at a lower socioeconomic status decreases. In the other direction, **nox**, the nitric oxides concentration, has strong positive correlations with **indus**, **age**, and **target** (0.726). **lstat** also has strong positive correlations with **indus** and **age**, while the average number of rooms per dwelling increases in suburbs with higher median values of owner-occupied homes.

Since **dis** and **nox** have relatively high correlations with the target, as well as somewhat normal summary statistics, they will likely have a role in our models.

```
cor_distarget <- cor.test(crime_train_df$dis, as.numeric(crime_train_df$target))
cor_distarget
```

```
##
##  Pearson's product-moment correlation
##
## data:  crime_train_df$dis and as.numeric(crime_train_df$target)
## t = -16.963, df = 464, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.6717579 -0.5592666
## sample estimates:
##        cor
## -0.6186731
```

```
cor_noxtarget <- cor.test(crime_train_df$nox, as.numeric(crime_train_df$target))
cor_noxtarget
```

```
##
##  Pearson's product-moment correlation
##
## data:  crime_train_df$nox and as.numeric(crime_train_df$target)
## t = 22.748, df = 464, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6801291 0.7663936
## sample estimates:
##       cor
## 0.7261062
```

Multicollinearity between **tax** and **rad**, as well as **lstat** and **medv**:

```
cor_taxrad<- cor.test(crime_train_df$tax, crime_train_df$rad)
cor_taxrad
```

```
##
##  Pearson's product-moment correlation
##
## data:  crime_train_df$tax and crime_train_df$rad
## t = 46.239, df = 464, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8888115 0.9214292
## sample estimates:
##       cor
## 0.9064632
```

lstat and **medv** are another pair of predictors with high correlation, this time negative:

```
cor_lstatMedv<- cor.test(crime_train_df$lstat, crime_train_df$medv)
cor_lstatMedv
```

```
##
##  Pearson's product-moment correlation
##
## data:  crime_train_df$lstat and crime_train_df$medv
## t = -23.405, df = 464, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.7748482 -0.6911599
## sample estimates:
##        cor
## -0.7358008
```

Now that we have a better sense of the data we're working with, we can begin to prepare and transform the data to maximize regression performance.

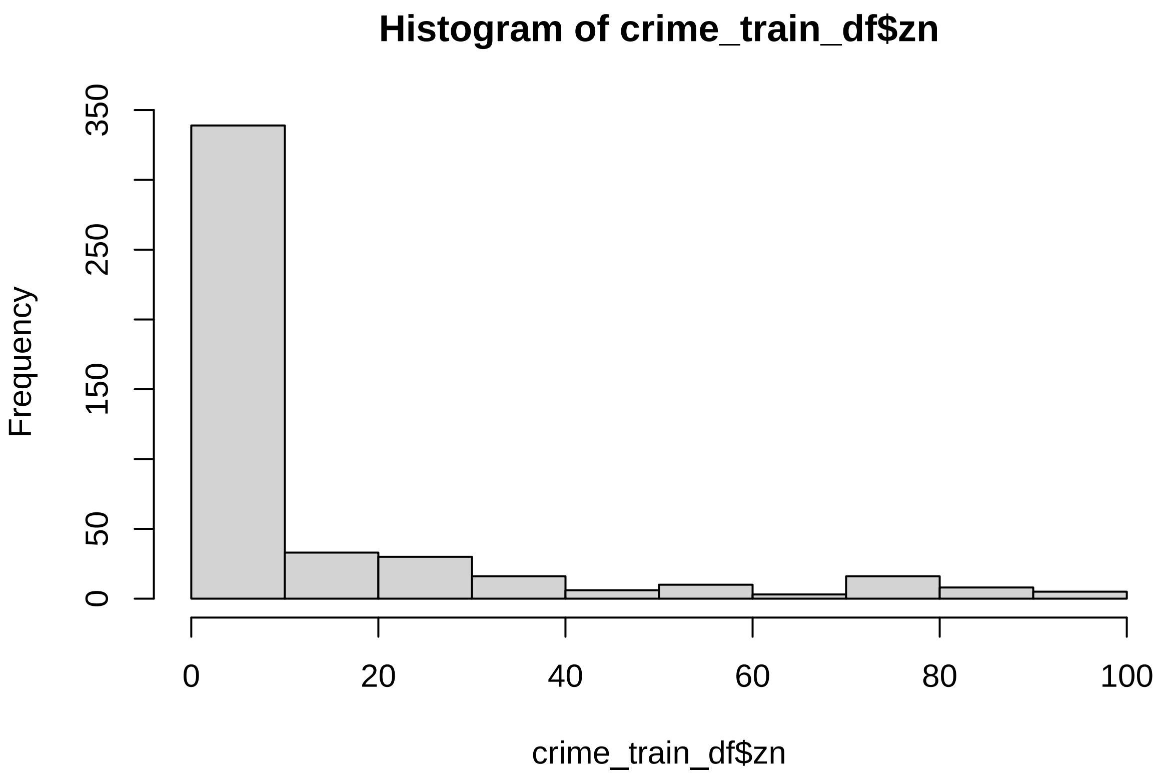
2. Data Preparation

As we prepare to model the data, we'll take two approaches: leaving the data alone with no transformations and transforming certain variables based on their specific distributions. In this section, we'll review the transformations we've experimented with, although you'll see in the subsequent section that we do not necessarily utilize each of the variables in its transformed state.

```
# transform target to a factor, for modeling
crime_train_df$target <-  as.factor(crime_train_df$target)
```

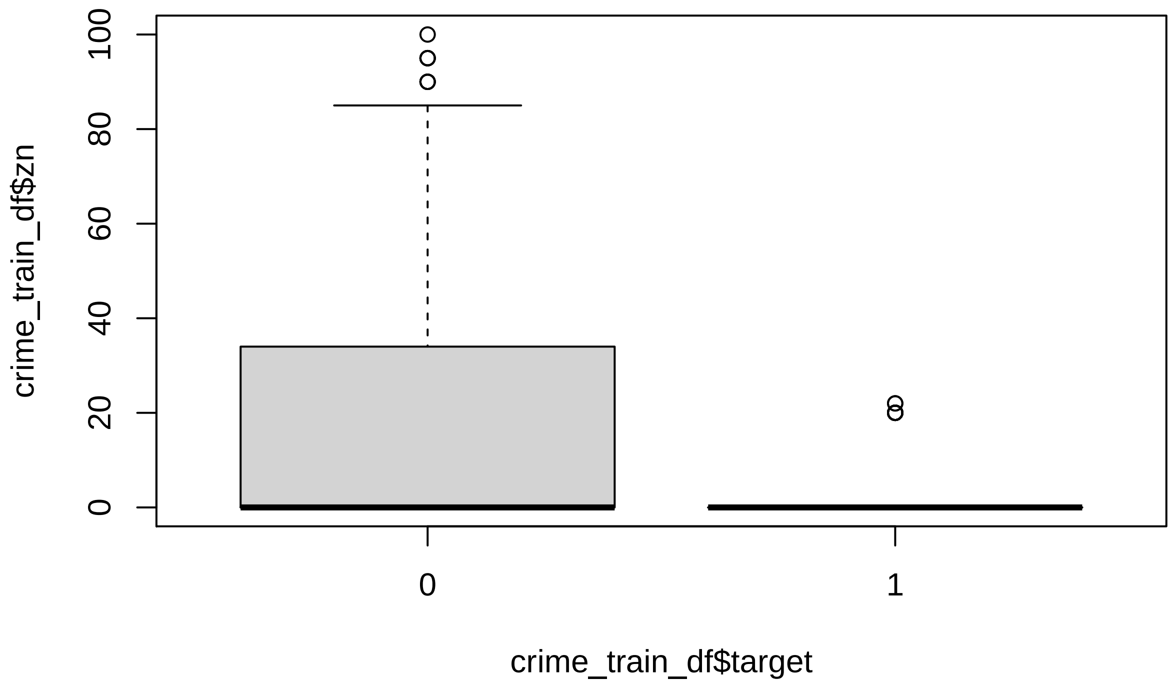
We'll go variable by variable, inspecting what looked strange during EDA, and attempting to make use of or transform the irregularities. First up is **zn**, which had 0 median and 100 max, as noted earlier.

```
hist(crime_train_df$zn)
```



How do those values break down based on the response?

```
boxplot(crime_train_df$zn ~ crime_train_df$target)
```



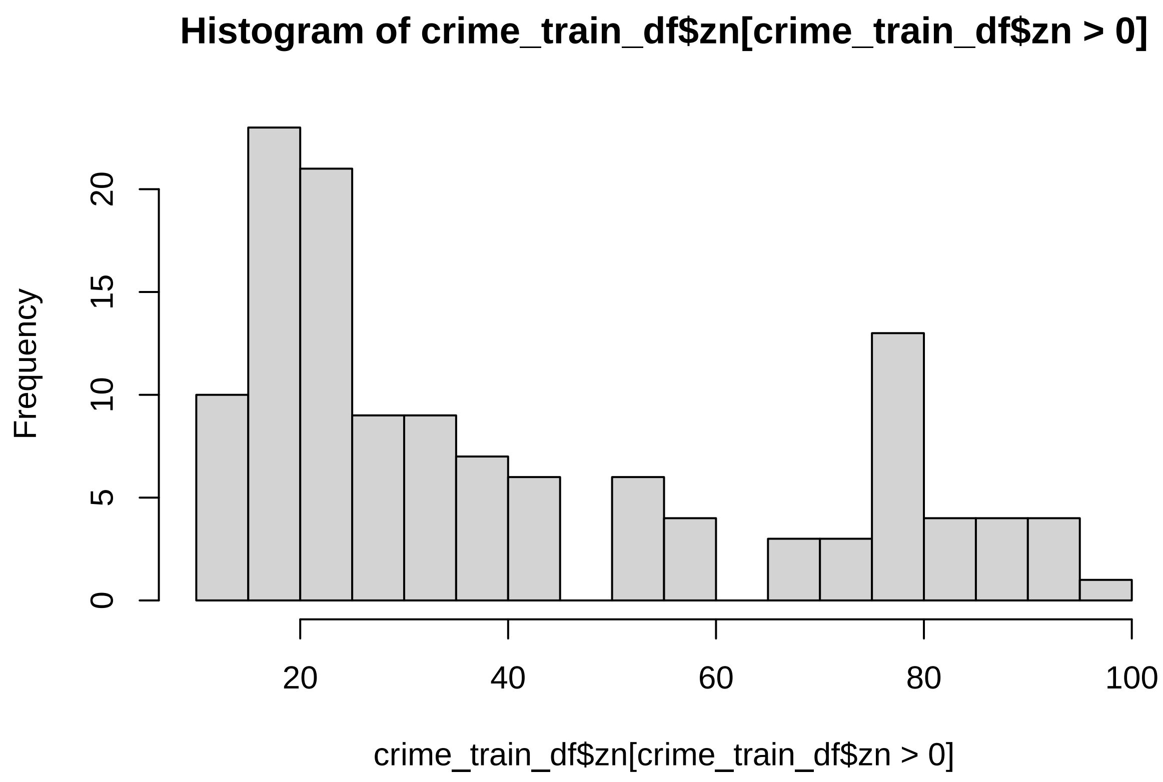
We notice that with just 2 exceptions, if the predictor is above a very tiny value, it is always in the lower crime towns (target==0). We know most of those low predictor values are 0, since that was the median, but let's get a finer-grained look than the histogram provided.

```
# Which values exist, below 10?
table(crime_train_df$zn[crime_train_df$zn<10])
```

```
##
##   0
## 339
```

All those small **zn** values are zero. What is the distribution of non-zero values?

```
hist(crime_train_df$zn[crime_train_df$zn>0], breaks=20)
```



Even after removing the zeros, the distribution appears to be bimodal, perhaps. Is there a threshold between the two modes, where the response variable becomes uniform?

```
table(crime_train_df$target[crime_train_df$zn > 22])
```

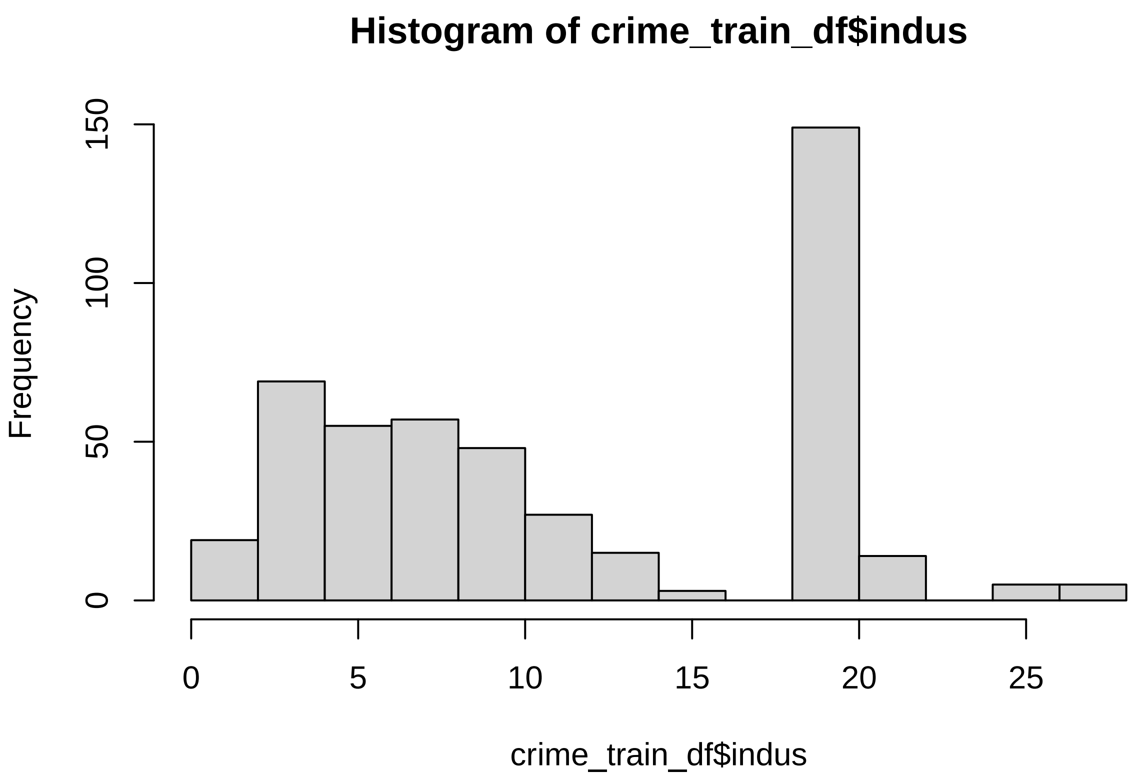
```
##
##  0  1
## 81  0
```

Every town with high proportions of land zoned for large residential lots has lower crime. So we'll create a dummy variable to indicate that, and hopefully that will allow the **zn** coefficient to fit to the remaining values better.

```
crime_train_df['zn_hi'] = crime_train_df$zn > 22
```

Next we look at **indus** similarly, starting with its distribution

```
hist(crime_train_df$indus)
```



The problem with a lot of these distributions is they don't arise from some random process. They are possibly the result of local zoning and legislative measures that create artificial clumps of data points, like the one around 20 in the **indus** chart above. Is that just one value?

```
table(crime_train_df$indus[crime_train_df$indus > 18])
```

```
##
##  18.1 19.58 21.89 25.65 27.74
##   121    28    14     5     5
```

Why would 121 different towns have exactly the same proportion of non-retail business acres? Let's see if the responses are uniform:

```
table(crime_train_df$target[(crime_train_df$indus>18) & (crime_train_df$indus<20)])
```

```
##
##   0   1
##   0 149
```

```
table(crime_train_df$target[(crime_train_df$indus>20)])
```

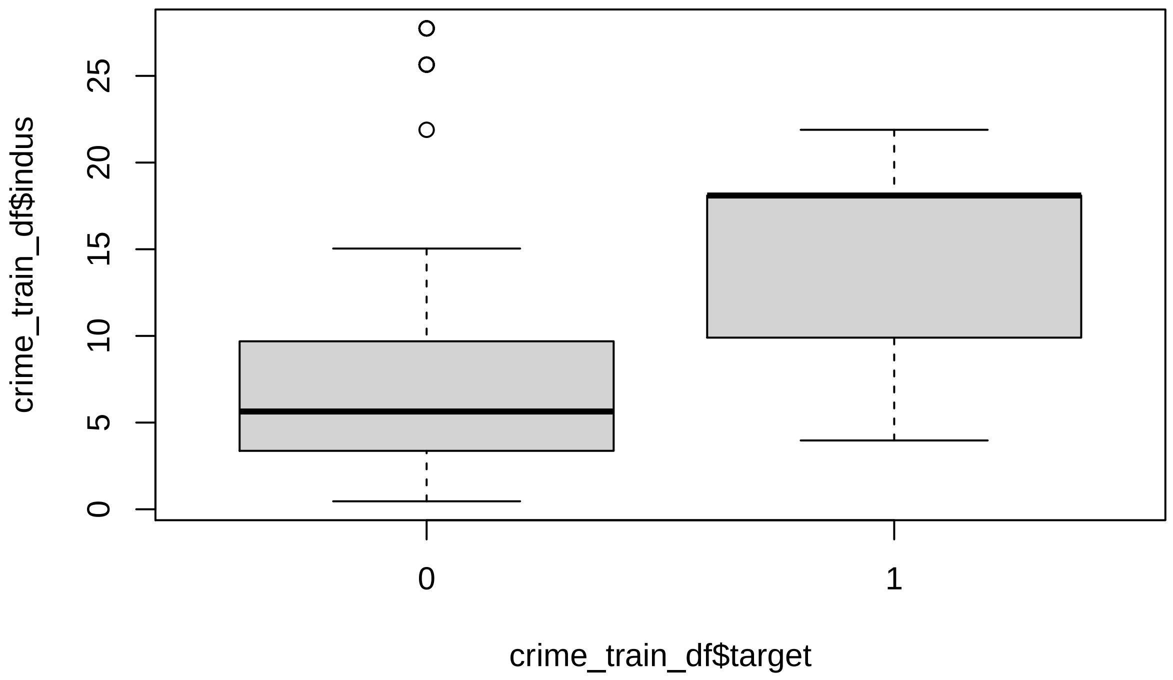
```
##
##  0  1
## 11 13
```

All 149 towns between 18 and 20 on the **indus** scale have higher crime, so we'll again make a dummy indicator for that fact, hoping to free up the coefficient to focus on the less bizarre values.

```
#Make a dummy indicator for 18<indus<20
crime_train_df['indus19'] = 1 * (crime_train_df$indus > 18 & crime_train_df$indus < 20)
```

How do the rest of the values break down along response lines?

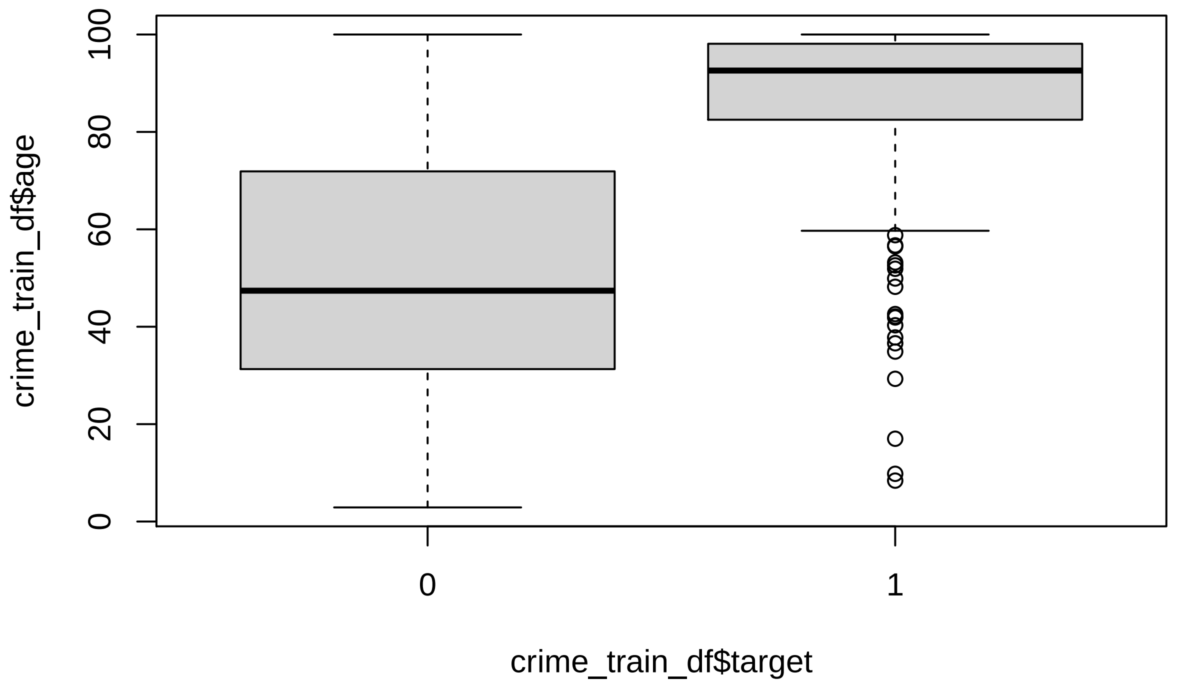
```
boxplot(crime_train_df$indus ~ crime_train_df$target)
```



Other than that clump of 18-20 (1/3 of the data) at the top of the right-side IQR, the split is fairly normal, with higher industry correlating with higher crime.

Moving on now to **age**, which had skewed summary statistics.

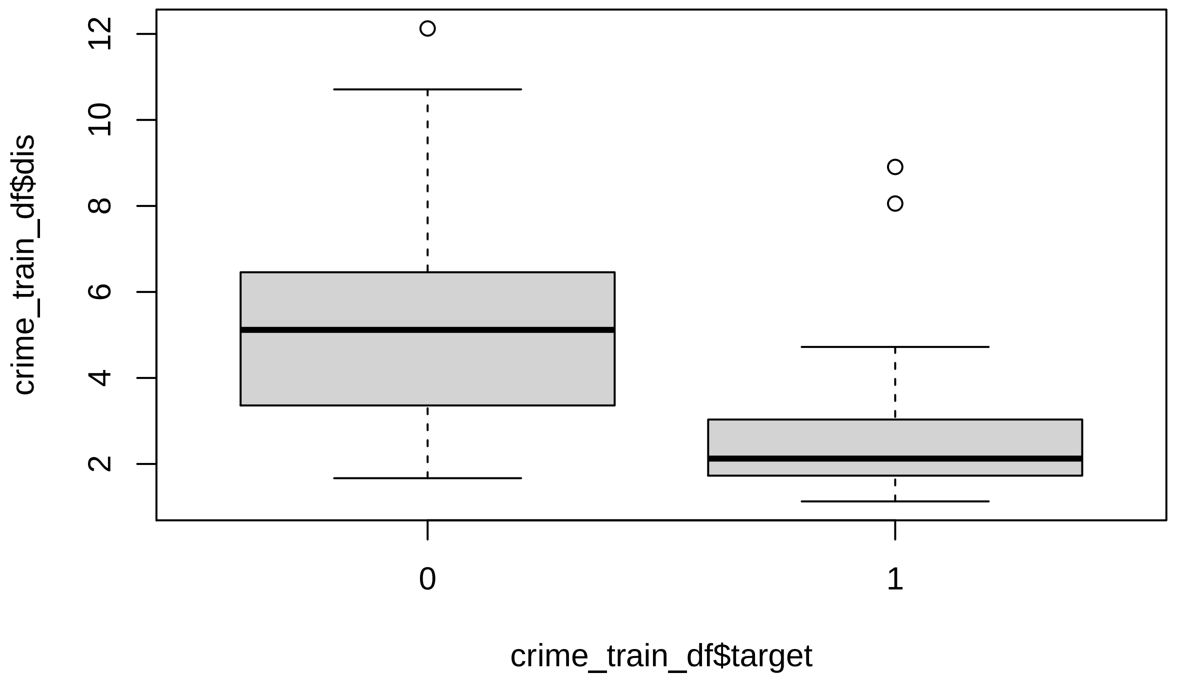
```
boxplot(crime_train_df$age ~ crime_train_df$target)
```



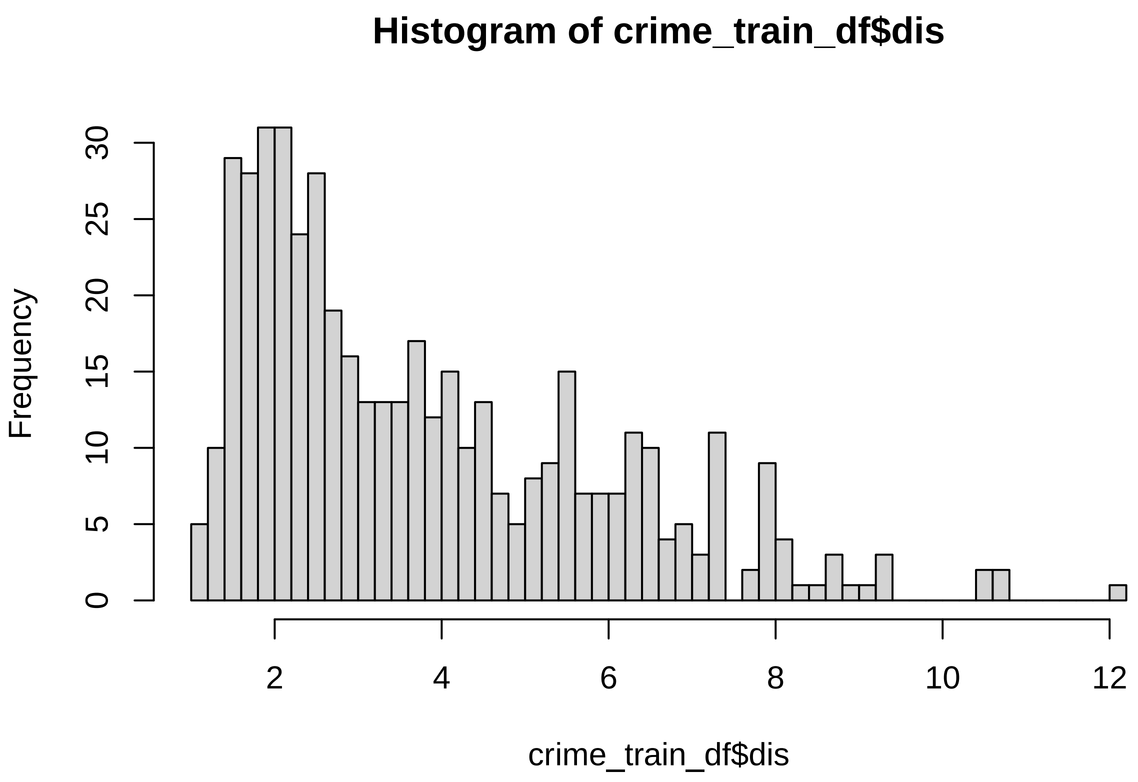
The lower crime areas are normally distributed for age values, but skewed left for higher crime areas. It's possible the model will be able to differentiate responses based on the different distributions, untransformed.

Next up is **dis**, which was right-skewed in the summary, and correlated negatively with crime.

```
boxplot(crime_train_df$dis ~ crime_train_df$target)
```



```
hist(crime_train_df$dis, breaks=50)
```



Although that predictor isn't perfectly normally distributed, by this dataset's standards, it's not far off. The variance of the values for the lower crime areas looked smaller than that of the higher crime areas.

```
paste('std.dev for higher crime areas: ', sd(crime_train_df$dis[crime_train_df$target==1]))
```

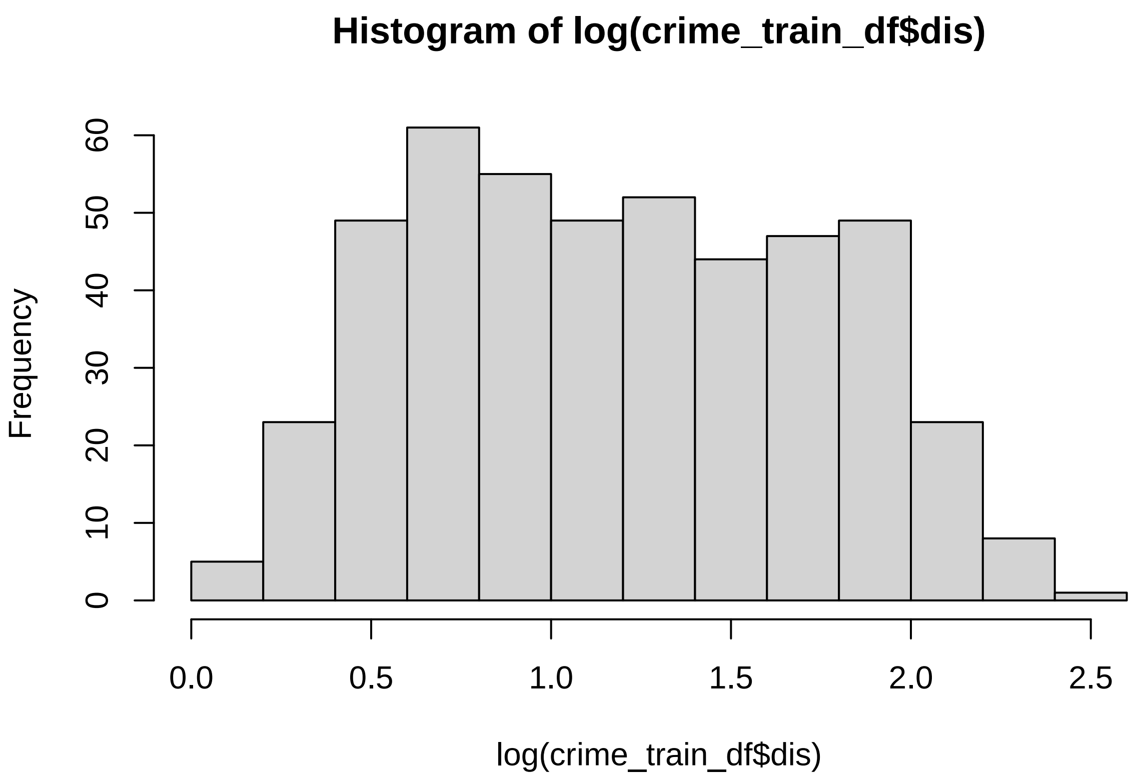
```
## [1] "std.dev for higher crime areas:  1.07899862600537"
```

```
paste('std.dev for lower crime areas: ', sd(crime_train_df$dis[crime_train_df$target==0]))
```

```
## [1] "std.dev for lower crime areas:  2.06739789850443"
```

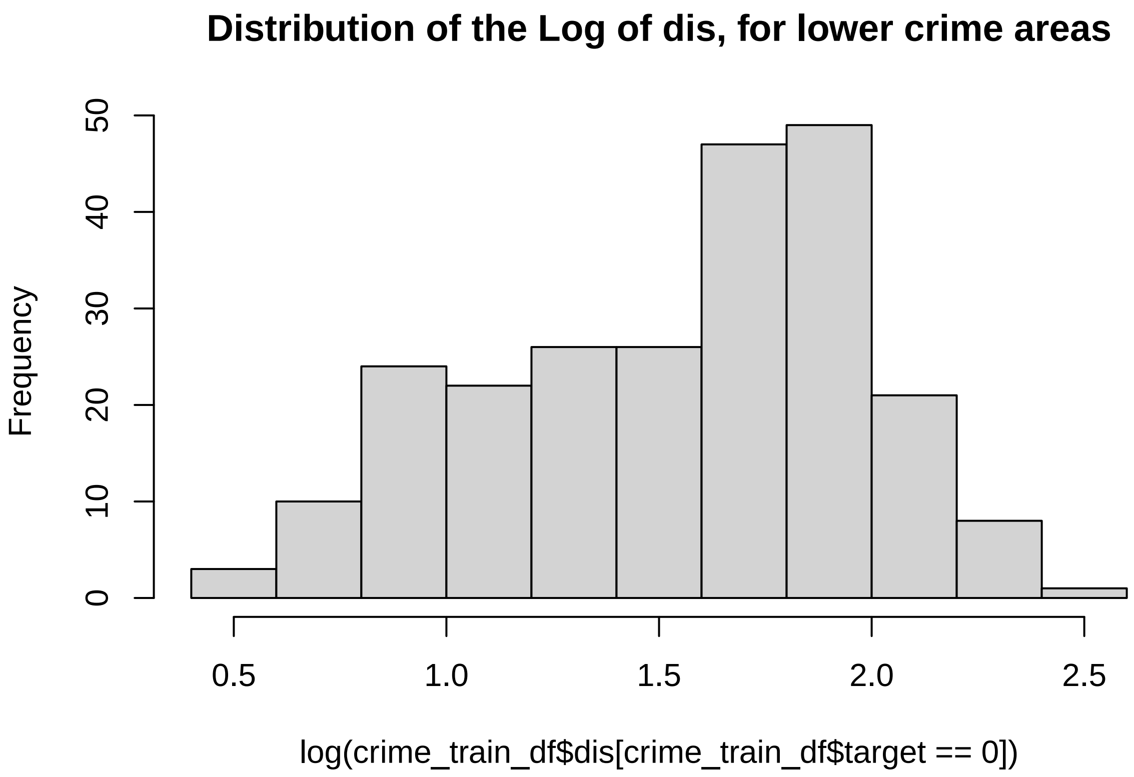
For the skew, let's check the distribution of the logarithm:

```
hist(log(crime_train_df$dis))
```

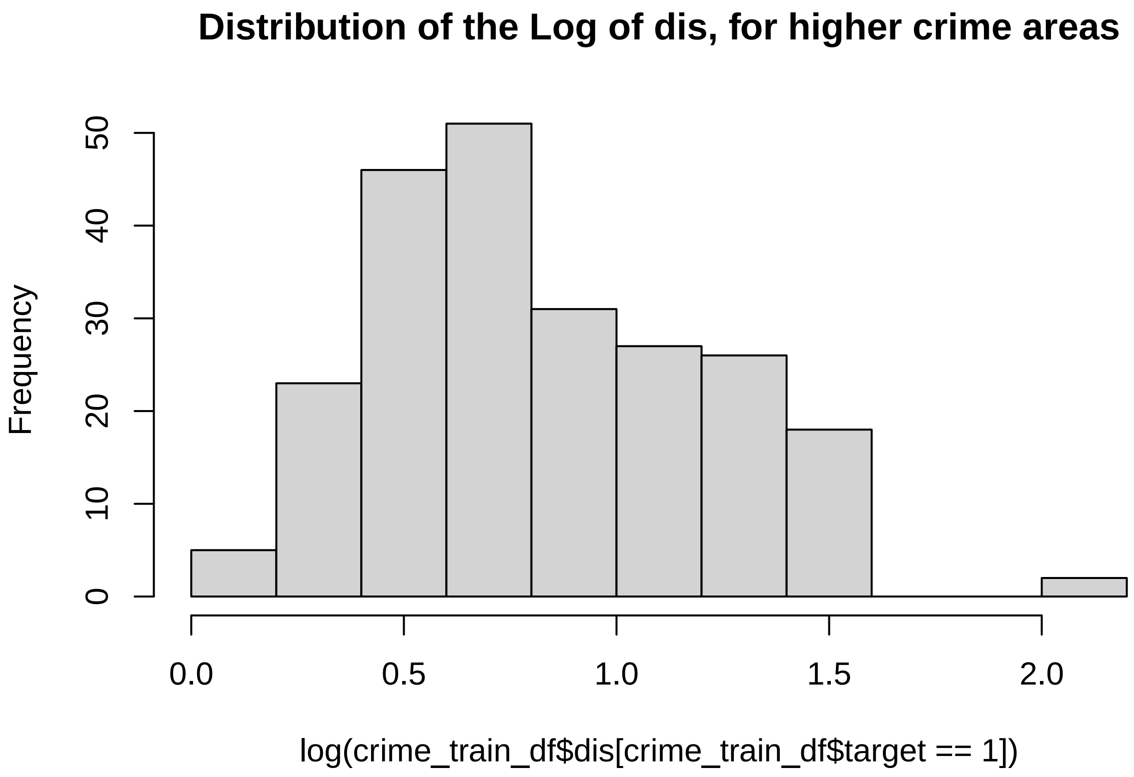


And if we condition that log distribution on the responses, how do the two look?

```
hist(log(crime_train_df$dis[crime_train_df$target==0]), main = 'Distribution of the Log of dis, for lowe
```



```r
hist(log(crime_train_df$dis[crime_train_df$target==1]), main = 'Distribution of the Log of dis, for hig
```

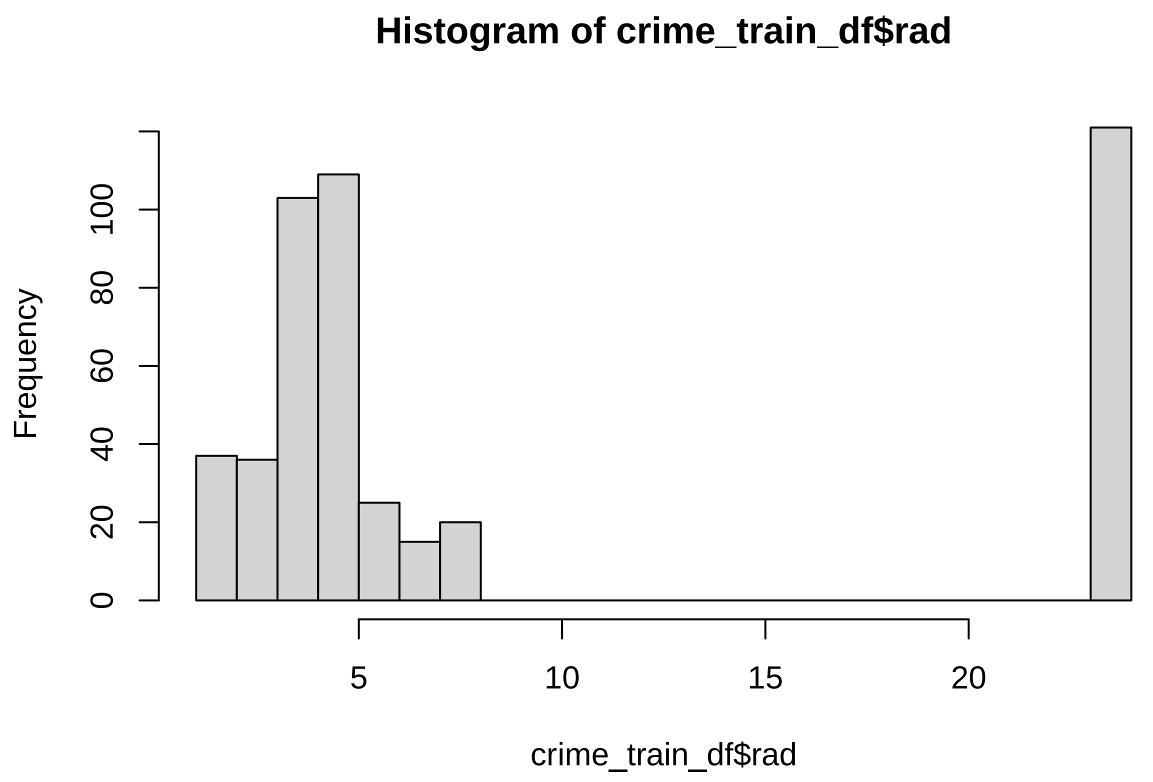


Taking the log normalizes the predictor conditioned on the response, so log_dis should be useful.

```r
crime_train_df['log_dis'] = log(crime_train_df$dis)
```

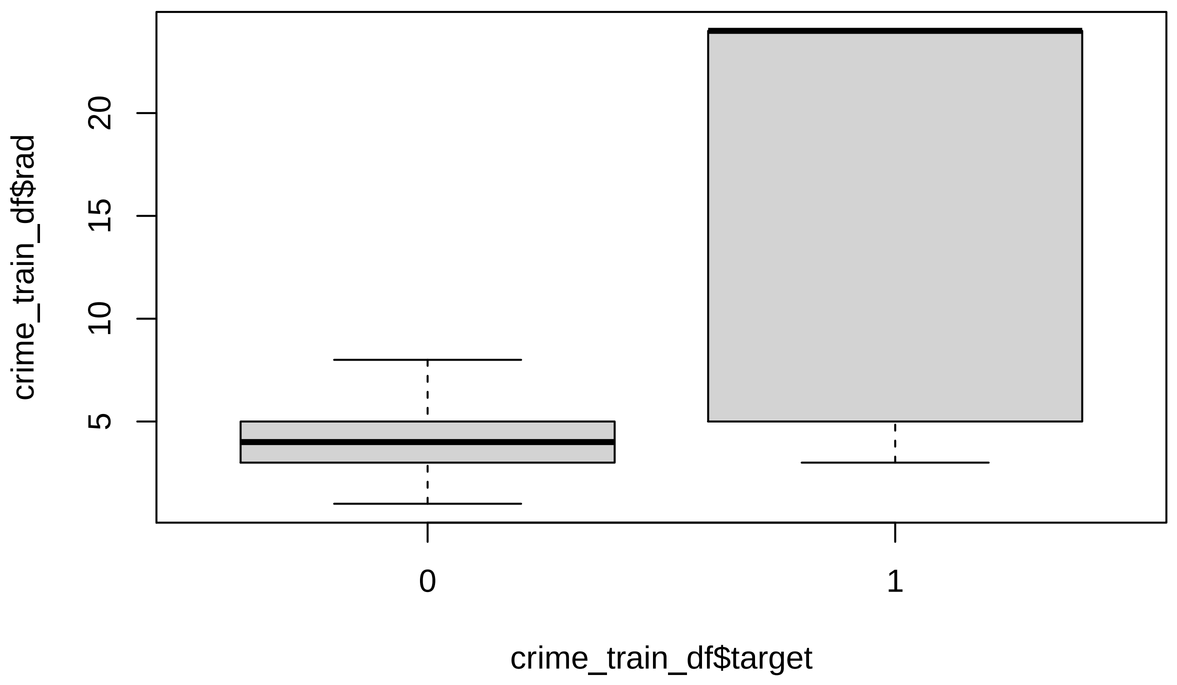
The next variable is **rad**, which was positively correlated with crime but oddly distributed.

```r
hist(crime_train_df$rad, breaks=20)
```



See how the responses split:

```
boxplot(crime_train_df$rad ~ crime_train_df$target)
```



Are the responses uniform based on predictor range, since there are two distinct ranges?

```
table(crime_train_df$target[crime_train_df$rad>15])
```

```
##
##     0   1
##     0 121
```

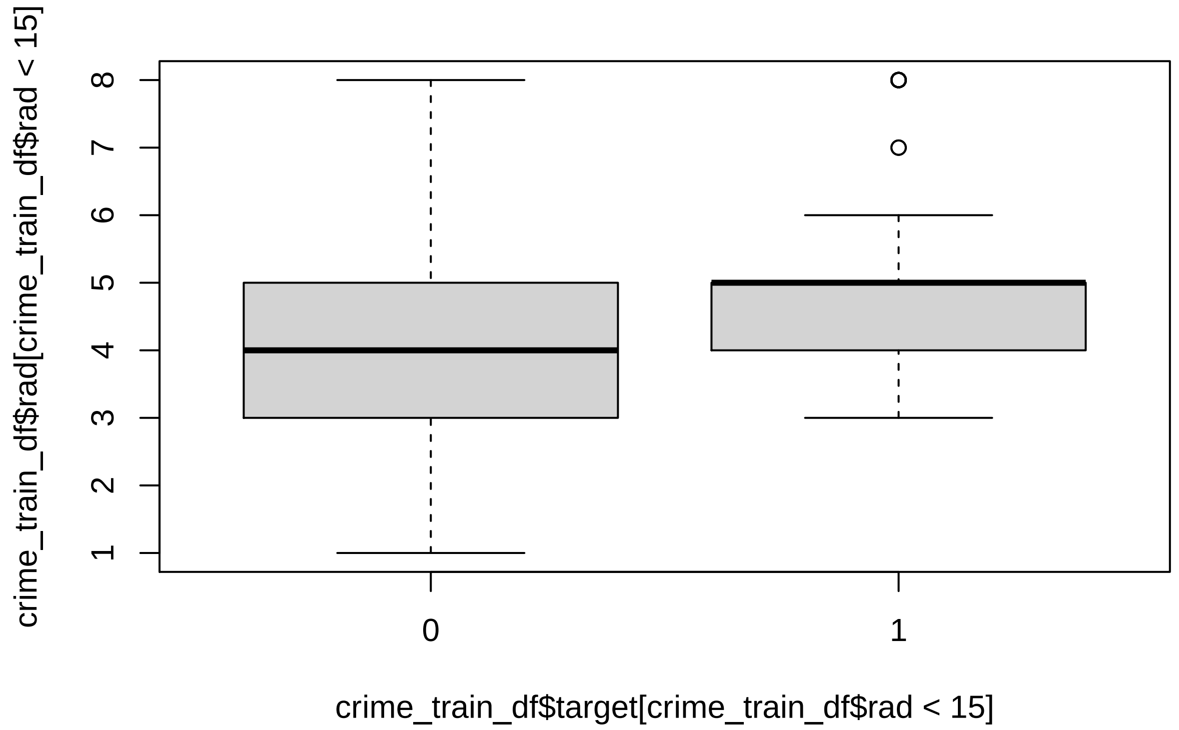
Uniformly high crime areas, for the high **rad** values. But are those just the same 121 towns that we already flagged with the **indus19** dummy variable we created earlier?

```
mean(crime_train_df$indus19[crime_train_df$rad>15])
```

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

Yes, that's redundant information. What about at the lower end of the **rad** range?

```
boxplot(crime_train_df$rad[crime_train_df$rad<15] ~ crime_train_df$target[crime_train_df$rad<15])
```



Is there a threshold below which every **rad** value corresponds to a 0 in **indus19**? Because if so, we can let the other dummy be responsible for those **rad** values and flag everything in between.

```
mean(crime_train_df$indus19[crime_train_df$rad<5])
```

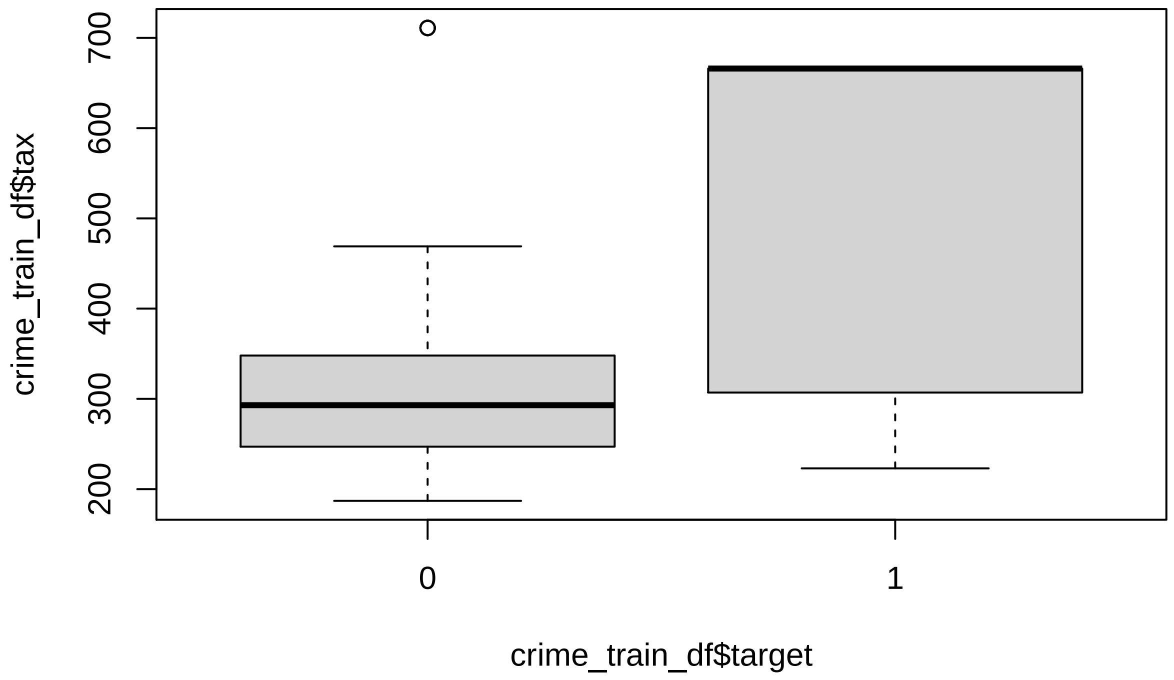
```
## [1] 0
```

Now we know that the only **rad** values that aren't redundant with **indus19** are those between 5 and 15, or 5 and 8, more specifically.

```
crime_train_df['rad5to8'] = 5 < crime_train_df$rad & crime_train_df$rad < 8
```

How about **tax**, which correlated positively with crime?

```
boxplot(crime_train_df$tax ~ crime_train_df$target)
```



```
sd(crime_train_df$tax[crime_train_df$target==1])
```

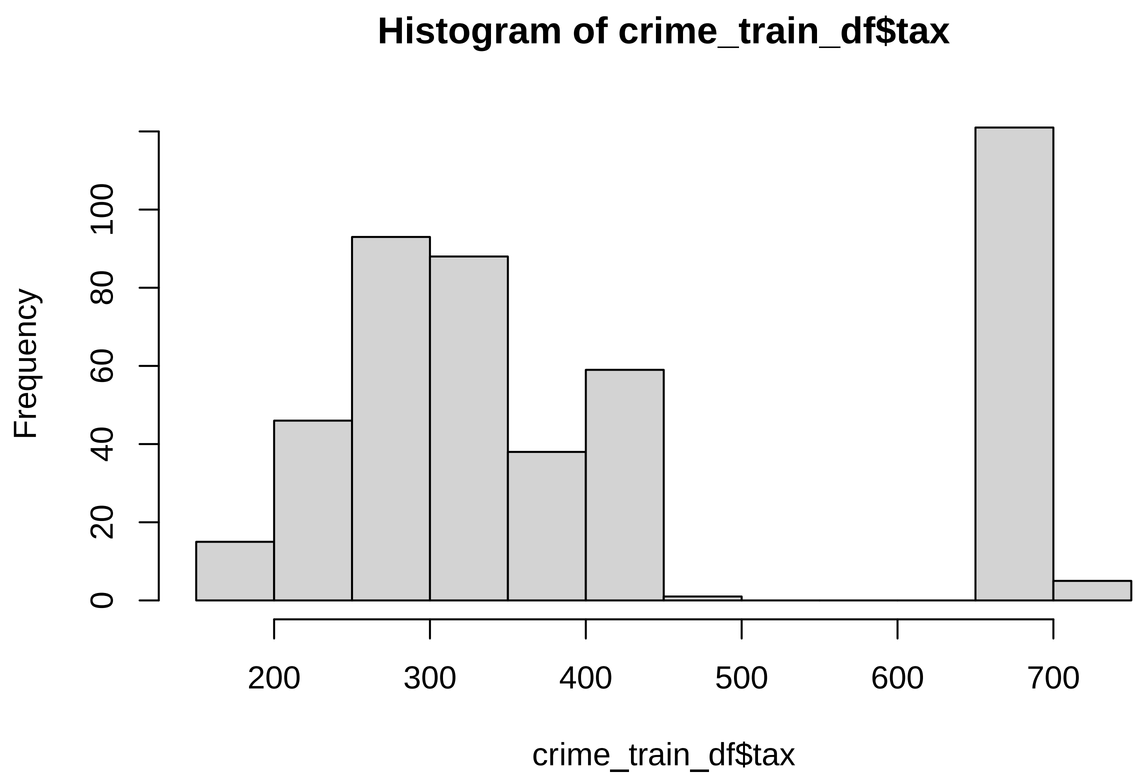
```
## [1] 166.6934
```

```
sd(crime_train_df$tax[crime_train_df$target==0])
```

```
## [1] 89.19775
```

Very different variances.

```
hist(crime_train_df$tax)
```



That looks like those same 121 towns, near the right of that chart. what are the exact values involved here?

```
table(crime_train_df$tax[crime_train_df$tax>600])
```

```
##
## 666 711
## 121   5
```

And are the 666 values the same towns as before?

```
table(crime_train_df$target[crime_train_df$tax == 666])
```

```
##
##   0   1
##   0 121
```

```
sum(crime_train_df$tax==666 & crime_train_df$indus19)
```

```
## [1] 121
```

Yes, the same towns again. Strange they chose 666 as the tax value per $10K, but people get angry about taxes. Or perhaps this is a proxy value for missing data. Whatever the reason, we'll make a flag for those.

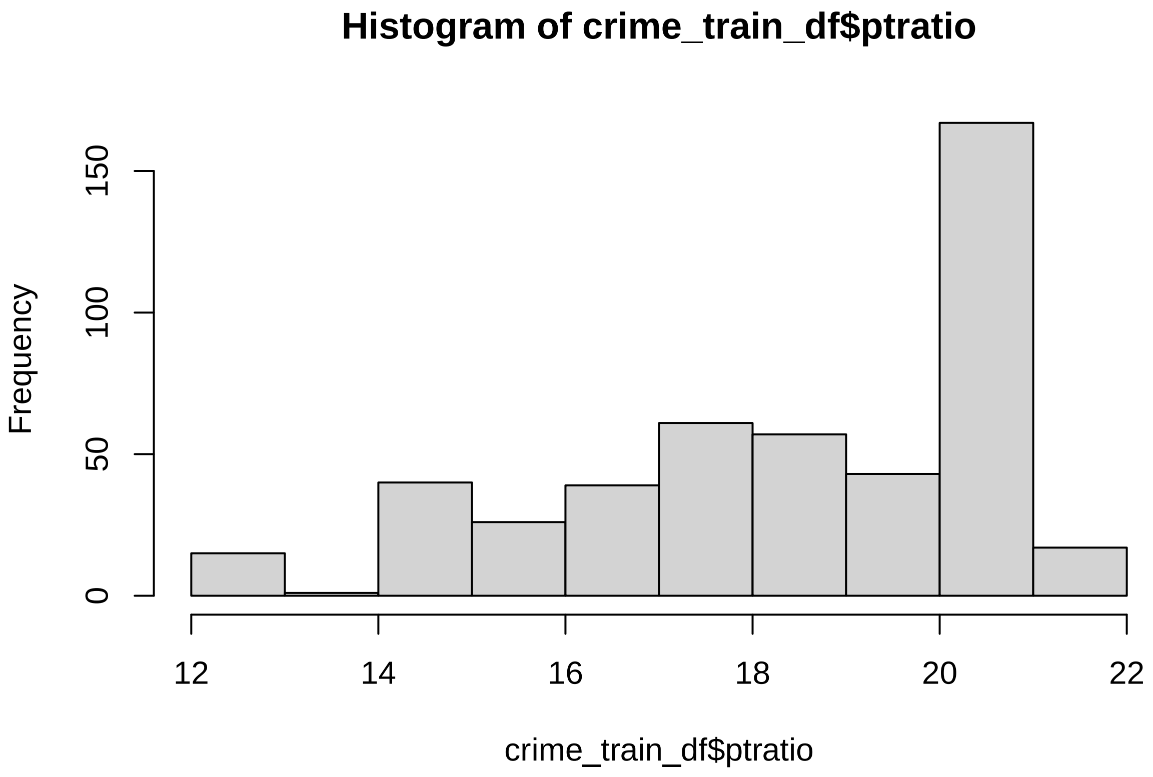
```
crime_train_df['tax_666'] = crime_train_df$tax==666
```

We'll test the logarithmic transform with **tax** as well.

```
crime_train_df['log_tax'] = log(crime_train_df$tax)
```

ptratio is next:

```
hist(crime_train_df$ptratio)
```



That clump of values around 21 should be inspected.

```
table(crime_train_df$ptratio[crime_train_df$ptratio>19])
```

```
##
## 19.1 19.2 19.6 19.7 20.1 20.2 20.9   21 21.1 21.2   22
##   14   17    6    6    5  128   11   23    1   14    2
```

Check the responses at 20.2

```
table(crime_train_df$target[crime_train_df$ptratio==20.2])
```

```
##
##   0   1
##   7 121
```

```
sum(crime_train_df$ptratio==20.2 & crime_train_df$indus19==1)
```

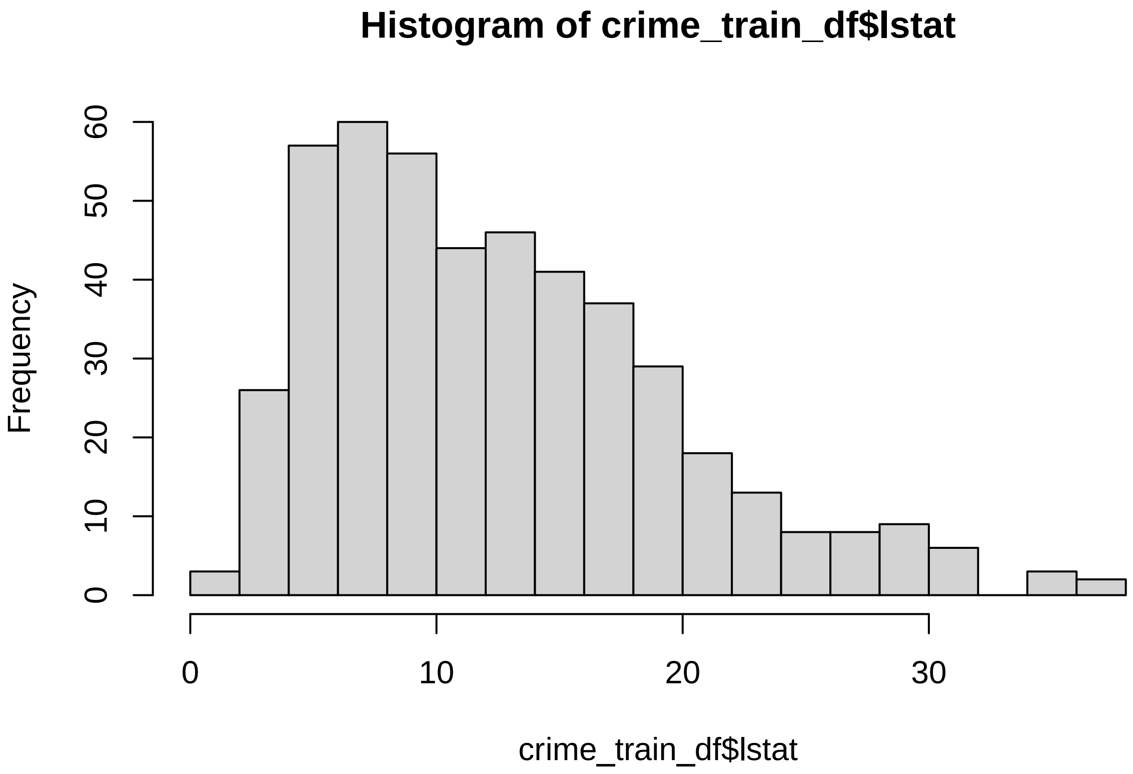
```
## [1] 121
```

Now it's harder, because those same 121 towns have a **ptratio** that's shared by 7 other towns in the other half of the crime split. The best we can do is to flag that value here, let **indus19** handle the high crime towns, and hope that the other predictors will correctly classify the 7 lower crime towns at that value. We add the logarithmic transform here as well.

```
crime_train_df['pt_peak'] = crime_train_df$ptratio == 20.2
crime_train_df['log_ptrat'] = log(crime_train_df$ptratio)
```

lstat:

```
hist(crime_train_df$lstat, breaks=20)
```



We'll try the logarithmic transform here again, to handle skew.

```
crime_train_df['log_lstat'] = log(crime_train_df$lstat)
```

Here's the full list of our variables at this point:

```
names(crime_train_df)
```

```
##  [1] "zn"        "indus"     "chas"      "nox"       "rm"        "age"
##  [7] "dis"       "rad"       "tax"       "ptratio"   "lstat"     "medv"
## [13] "target"    "zn_hi"     "indus19"   "log_dis"   "rad5to8"   "tax_666"
## [19] "log_tax"   "pt_peak"   "log_ptrat" "log_lstat"
```

3. Build the Models

We'll set aside 10% of the data as a validation check on our models. Although the proper way to do this would have been to split it off before transforming the data, especially when we used specific thresholds to create dummy variables, the fact is that the transforms would still be the same, as we didn't learn any new information from the validation data. The 121 towns that showed up as a bloc in several variables would instead have been 109 towns or so, but it wouldn't have changed any decisions for transformations or threshold points.

```r
set.seed(123)
split <- caret::createDataPartition(crime_train_df$target, p=0.90, list=FALSE)
train <- crime_train_df[split, ]
validation <- crime_train_df[ -split, ]
```

Model 1: Create a full model from the original, untransformed predictors

For our first model, we chose to include all the variables as is, without transformation to get a first look at our data and how the variables predict our outcome of interest while controlling for other predictors. We modeled our outcome as having 'Higher Crime Rate' based on the median crime rate without any kind of transformations.

Throughout this process, we'll use AIC as an important metric to judge the fit of our models. Akaike Information Criterion (AIC) is a way of scoring a model based on its log-likelihood and complexity. AIC is a method for scoring and selecting a model. It is named for the developer of the method, Hirotugu Akaike, and may be shown to have a basis in information theory and frequentist-based inference.The AIC statistic is defined for logistic regression as follows AIC = -2/N * LL + 2 * k/N where N is the number of examples in the training dataset, LL is the log-likelihood of the model on the training dataset, and k is the number of parameters in the model. We prefer the model with the lowest AIC in general, but the AIC statistic penalizes complex models less than other measures such as BIC, meaning that it may put more emphasis on model performance on the training dataset, and, in turn, select more complex models.

```r
mod_1  <- glm(target ~ . - zn_hi - indus19 - log_dis - rad5to8 - tax_666 - log_tax -
                pt_peak - log_ptrat - log_lstat, data = train, family = 'binomial')
summary(mod_1)
```

```
##
## Call:
## glm(formula = target ~ . - zn_hi - indus19 - log_dis - rad5to8 -
##     tax_666 - log_tax - pt_peak - log_ptrat - log_lstat, family = "binomial",
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8052  -0.1683  -0.0025   0.0039   3.3402
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -39.225894   6.637568  -5.910 3.43e-09 ***
## zn           -0.059078   0.034500  -1.712 0.086823 .
## indus        -0.064667   0.047974  -1.348 0.177675
## chas          0.684938   0.788876   0.868 0.385260
## nox          46.468599   7.872057   5.903 3.57e-09 ***
## rm           -0.584050   0.748460  -0.780 0.435193
## age           0.030825   0.014177   2.174 0.029682 *
## dis           0.705181   0.235540   2.994 0.002754 **
## rad           0.642895   0.167471   3.839 0.000124 ***
## tax          -0.006111   0.003030  -2.017 0.043737 *
## ptratio       0.412020   0.130085   3.167 0.001539 **
## lstat         0.055347   0.056136   0.986 0.324160
## medv          0.178747   0.070322   2.542 0.011027 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 583.51  on 420  degrees of freedom
## Residual deviance: 179.07  on 408  degrees of freedom
## AIC: 205.07
##
## Number of Fisher Scoring iterations: 9
```

Based on the adjusted model, odds of high crime rate decrease with every unit increase in proportion of residential land zoned for large lots (Zn), while controlling for other independent variables. The proportion of non-retail business (Indus) and suburbs bordering the Charles River did not predict High Crime Rate in our adjusted model, p-value > .05. The odds of high crime rate increased exorbitantly with every unit increase in NOx (p<.05) and decreased with every unit increase in rooms/dwelling (Rm) but this relationship was not significant. Concurring with the histogram plots and other descriptive statistics, odds of high crime rate increase with unit increase in proportion of owner-occupied units built prior to 1940 (age). Higher crime rate was also significantly predicted by avg distances to five Boston employment centers (Dis) and index of accessibility to radial highways (Rad). Odds of high crime rate decreased with unit increase in full-value property-tax rate per $10,000 (Tax) but increased with higher pupil-teacher ratio by town (Ptratio). Percentage of lower status of the population was not predictive of Higher Crime Rate at p=.32. However, the odds of high crime rate increased with unit increase in median value of owner-occupied homes (in $1000s) significantly at p < .02.

The AIC value for this model was 205.07. We will compare this value with our next set of models to make a final decision about the model we use to make predictions on our evaluation dataset.

How well does Model 1 predict the validation split?

```
# generating the predictors
mod_1.pred = predict(mod_1, newdata = validation, type = 'response')
mod_1.pred[mod_1.pred >= 0.5] <- 1
mod_1.pred[mod_1.pred < 0.5] <- 0
mod_1.pred = as.factor(mod_1.pred)
#  generating the confusion matrix
mod_1.confusion.matrix <- confusionMatrix(mod_1.pred, validation$target, mode = "everything")
mod_1.confusion.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23  1
##          1  0 21
##
##                Accuracy : 0.9778
##                  95% CI : (0.8823, 0.9994)
##     No Information Rate : 0.5111
##     P-Value [Acc > NIR] : 3.366e-12
##
```

```
##                   Kappa : 0.9555
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9545
##          Pos Pred Value : 0.9583
##          Neg Pred Value : 1.0000
##               Precision : 0.9583
##                  Recall : 1.0000
##                      F1 : 0.9787
##              Prevalence : 0.5111
##          Detection Rate : 0.5111
##    Detection Prevalence : 0.5333
##       Balanced Accuracy : 0.9773
##
##        'Positive' Class : 0
##
```

There was just 1 false negative as far as errors on the 45 randomly held out towns. Thus the high F-1 score of .9787, and perfect recall, and a specificity of .9545.

Model 1, part 2: Create a model by backing out less significant predictors from Model 1

We next adapt the model using the AIC in a stepwise algorithm. The mode or direction of the stepwise search that will pass would be "backward". The stepAIC() function performs backward model selection by starting from a "maximal" model, which is then trimmed down. We can see below that the AIC has gone down and this suggests that the maximal or full model can be improved by simply discarding few of the attributes.

```r
mod_2 <- mod_1 %>% stepAIC(direction = "backward", trace = FALSE)
summary(mod_2)
```

```
##
## Call:
## glm(formula = target ~ zn + nox + age + dis + rad + tax + ptratio +
##     lstat + medv, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8816  -0.1756  -0.0025   0.0038   3.2731
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -37.559476   6.131018  -6.126 9.00e-10 ***
## zn           -0.063689   0.032940  -1.934  0.05317 .
## nox          40.964995   6.718795   6.097 1.08e-09 ***
## age           0.024696   0.011479   2.151  0.03144 *
## dis           0.633063   0.220894   2.866  0.00416 **
## rad           0.711231   0.156279   4.551 5.34e-06 ***
## tax          -0.007915   0.002708  -2.922  0.00347 **
## ptratio       0.348429   0.115424   3.019  0.00254 **
## lstat         0.071901   0.049216   1.461  0.14403
## medv          0.134875   0.042800   3.151  0.00163 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 583.51  on 420  degrees of freedom
## Residual deviance: 181.98  on 411  degrees of freedom
## AIC: 201.98
##
## Number of Fisher Scoring iterations: 9
```

The AIC is now slightly better, reduced from 205 to 202, using fewer predictors for a more parsimonious model.

How well does the slimmer version of Model 1 predict the same validation split?

```
# generating the predictors
mod_2.pred = predict(mod_2, newdata = validation, type = 'response')
mod_2.pred[mod_2.pred >= 0.5] <- 1
mod_2.pred[mod_2.pred < 0.5] <- 0
mod_2.pred = as.factor(mod_2.pred)
#  generating the confusion matrix
mod_2.confusion.matrix <- confusionMatrix(mod_2.pred, validation$target, mode = "everything")
mod_2.confusion.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23  1
##          1  0 21
##
##                Accuracy : 0.9778
##                  95% CI : (0.8823, 0.9994)
##     No Information Rate : 0.5111
##     P-Value [Acc > NIR] : 3.366e-12
##
##                   Kappa : 0.9555
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9545
##          Pos Pred Value : 0.9583
##          Neg Pred Value : 1.0000
##               Precision : 0.9583
##                  Recall : 1.0000
##                      F1 : 0.9787
##              Prevalence : 0.5111
##          Detection Rate : 0.5111
##    Detection Prevalence : 0.5333
##       Balanced Accuracy : 0.9773
##
```

```
##        'Positive' Class : 0
##
```

This slimmer version of Model 1 produces the same score on the 10% validation set.

Rather than forcing a decision on which of those models is preferable right now, we'll see how the variable transformations affect the fits.

Model 2: Add the logarithmic transformations as predictors

```
mod_6  <- glm(target ~ . - zn_hi - indus19 - rad5to8 - tax_666 -
                pt_peak, data = train, family = 'binomial')
summary(mod_6)
```

```
##
## Call:
## glm(formula = target ~ . - zn_hi - indus19 - rad5to8 - tax_666 -
##     pt_peak, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0253  -0.1170   0.0000   0.0518   4.0607
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -60.99142   90.66587  -0.673 0.501135
## zn            -0.02261    0.04361  -0.518 0.604226
## indus          0.31665    0.09964   3.178 0.001482 **
## chas          -0.61389    0.99397  -0.618 0.536827
## nox           34.74055    9.85078   3.527 0.000421 ***
## rm            -1.12371    0.96730  -1.162 0.245358
## age            0.02282    0.01542   1.479 0.139094
## dis           -2.25930    0.96916  -2.331 0.019743 *
## rad            1.31632    0.27993   4.702 2.57e-06 ***
## tax           -0.18464    0.04711  -3.919 8.89e-05 ***
## ptratio        7.61257    2.36590   3.218 0.001293 **
## lstat          0.43173    0.16633   2.596 0.009442 **
## medv           0.18530    0.10338   1.792 0.073056 .
## log_dis       10.53648    4.03093   2.614 0.008951 **
## log_tax       54.77392   13.58838   4.031 5.56e-05 ***
## log_ptrat   -123.81176   41.59933  -2.976 0.002918 **
## log_lstat     -5.19274    2.23200  -2.326 0.019992 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 583.51  on 420  degrees of freedom
## Residual deviance: 127.69  on 404  degrees of freedom
## AIC: 161.69
##
## Number of Fisher Scoring iterations: 9
```

The AIC has decreased significantly, from 202 to 162, by adding the 4 logarithmic transforms, all of which have significant p-values for the coefficients. Let's remove unimportant predictors like **chas**, the way we did with Model 1, and see if the AIC drops by a similar (small) amount. We'll also remove the intercept, since it has a large coefficient, yet an insignificant p-value, so the model is using it to explain a different variable.

```
mod_6  <- glm(target ~ . - zn_hi - indus19 - rad5to8 - tax_666 -
                pt_peak - 1, data = train, family = 'binomial') # removing intercept
mod_7 <- mod_6 %>% stepAIC(direction = "backward", trace = FALSE)
summary(mod_7)
```

```
##
## Call:
## glm(formula = target ~ indus + nox + dis + rad + tax + ptratio +
##     lstat + log_dis + log_tax + log_ptrat + log_lstat - 1, family = "binomial",
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0609  -0.1599  -0.0001   0.0453   3.8484
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## indus        0.30805    0.08786   3.506 0.000454 ***
## nox         31.40155    7.65065   4.104 4.05e-05 ***
## dis         -2.74103    0.83232  -3.293 0.000990 ***
## rad          1.29419    0.22707   5.700 1.20e-08 ***
## tax         -0.17639    0.03631  -4.858 1.18e-06 ***
## ptratio      8.83619    1.63674   5.399 6.71e-08 ***
## lstat        0.45479    0.15101   3.012 0.002598 **
## log_dis     11.50460    3.60920   3.188 0.001435 **
## log_tax     52.29727   10.65508   4.908 9.19e-07 ***
## log_ptrat -148.07127   27.78421  -5.329 9.86e-08 ***
## log_lstat   -5.40203    1.83116  -2.950 0.003177 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 583.63  on 421  degrees of freedom
## Residual deviance: 132.82  on 410  degrees of freedom
## AIC: 154.82
##
## Number of Fisher Scoring iterations: 9
```

That results in a lower AIC (155 vs 162). All the logarithmic terms have signs that offset their untransformed equivalents, which makes sense, since each pair has correlating terms, that are nevertheless important to the model's fit, as shown by the better fit. Let's see how the predictions do on the validation set:

```
# generating the predictors
mod_7.pred = predict(mod_7, newdata = validation, type = 'response')
mod_7.pred[mod_7.pred >= 0.5] <- 1
mod_7.pred[mod_7.pred < 0.5] <- 0
mod_7.pred = as.factor(mod_7.pred)
```

```r
#  generating the confusion matrix
mod_7.confusion.matrix <- confusionMatrix(mod_7.pred, validation$target, mode = "everything")
mod_7.confusion.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 22  1
##          1  1 21
##
##                Accuracy : 0.9556
##                  95% CI : (0.8485, 0.9946)
##     No Information Rate : 0.5111
##     P-Value [Acc > NIR] : 7.258e-11
##
##                   Kappa : 0.9111
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9565
##             Specificity : 0.9545
##          Pos Pred Value : 0.9565
##          Neg Pred Value : 0.9545
##               Precision : 0.9565
##                  Recall : 0.9565
##                      F1 : 0.9565
##              Prevalence : 0.5111
##          Detection Rate : 0.4889
##    Detection Prevalence : 0.5111
##       Balanced Accuracy : 0.9555
##
##        'Positive' Class : 0
##
```

Interestigly, despite the much better AIC score for Model 2, compared to model 1, there is now one false positive in addition to the one false negative. The two main possibilities are that this model has overfit, with the addition of the log terms, or that it just happened to make one more error than an already accurate score, on this 10% of the data.

Next we will add in the dummy variables we created.

Model 3: Add dummies, to fit a model with all our predictors

```r
mod_3  <- glm(target ~ . , data = train, family = 'binomial')
summary(mod_3)
```

```
##
## Call:
## glm(formula = target ~ ., family = "binomial", data = train)
##
```

```
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.11821  -0.00003   0.00000   0.00000   2.17431
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.558e+03  4.200e+02  -3.710 0.000207 ***
## zn           9.553e-01  2.483e-01   3.847 0.000119 ***
## indus        1.911e-01  2.271e-01   0.842 0.400035
## chas        -1.687e-01  1.447e+00  -0.117 0.907201
## nox          1.659e+02  4.033e+01   4.115 3.88e-05 ***
## rm          -5.077e+00  2.016e+00  -2.518 0.011789 *
## age          4.148e-02  2.495e-02   1.662 0.096421 .
## dis         -3.898e+00  2.213e+00  -1.762 0.078134 .
## rad          3.747e+00  1.008e+00   3.716 0.000203 ***
## tax         -5.894e-01  1.843e-01  -3.199 0.001381 **
## ptratio     -1.379e+01  5.206e+00  -2.649 0.008078 **
## lstat        7.066e-01  2.379e-01   2.970 0.002982 **
## medv         7.024e-01  2.236e-01   3.141 0.001682 **
## zn_hiTRUE   -7.759e+01  1.985e+03  -0.039 0.968814
## indus19      3.594e+01  6.235e+03   0.006 0.995401
## log_dis      1.452e+01  7.908e+00   1.836 0.066347 .
## rad5to8TRUE -1.107e+01  3.033e+00  -3.649 0.000263 ***
## tax_666TRUE  2.982e+00  1.985e+04   0.000 0.999880
## log_tax      1.690e+02  5.633e+01   3.001 0.002693 **
## pt_peakTRUE -1.941e+01  1.870e+04  -0.001 0.999172
## log_ptrat    3.215e+02  1.047e+02   3.072 0.002129 **
## log_lstat   -7.247e+00  3.045e+00  -2.380 0.017317 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 583.514  on 420  degrees of freedom
## Residual deviance:  71.971  on 399  degrees of freedom
## AIC: 115.97
##
## Number of Fisher Scoring iterations: 21
```

Before we even try to interpret this model, let's back out some unimportant predictors.

```r
mod_3  <- glm(target ~ . , data = train, family = 'binomial')
mod_4 <- mod_3 %>% stepAIC(direction = "backward", trace = FALSE)
summary(mod_4)
```

```
##
## Call:
## glm(formula = target ~ zn + nox + rm + age + dis + rad + tax +
##     ptratio + lstat + medv + zn_hi + indus19 + log_dis + rad5to8 +
##     log_tax + log_ptrat + log_lstat, family = "binomial", data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.19148  -0.00005   0.00000   0.00000   2.07352
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.604e+03  3.697e+02  -4.340 1.42e-05 ***
## zn           9.403e-01  2.148e-01   4.378 1.20e-05 ***
## nox          1.711e+02  3.774e+01   4.534 5.77e-06 ***
## rm          -4.999e+00  1.963e+00  -2.546 0.010894 *
## age          4.196e-02  2.359e-02   1.779 0.075300 .
## dis         -3.081e+00  1.464e+00  -2.105 0.035307 *
## rad          3.456e+00  9.410e-01   3.673 0.000240 ***
## tax         -5.548e-01  1.316e-01  -4.216 2.49e-05 ***
## ptratio     -1.647e+01  4.814e+00  -3.422 0.000621 ***
## lstat        7.394e-01  2.359e-01   3.134 0.001722 **
## medv         6.963e-01  2.184e-01   3.188 0.001434 **
## zn_hiTRUE   -7.767e+01  3.314e+03  -0.023 0.981299
## indus19      3.451e+01  2.588e+03   0.013 0.989359
## log_dis      1.143e+01  5.903e+00   1.935 0.052931 .
## rad5to8TRUE -1.170e+01  2.881e+00  -4.062 4.87e-05 ***
## log_tax      1.612e+02  3.811e+01   4.229 2.35e-05 ***
## log_ptrat    3.668e+02  9.921e+01   3.697 0.000218 ***
## log_lstat   -7.670e+00  3.035e+00  -2.527 0.011490 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 583.514  on 420  degrees of freedom
## Residual deviance:  73.737  on 403  degrees of freedom
## AIC: 109.74
##
## Number of Fisher Scoring iterations: 22
```

This model produced the same errors as Model 1, on the validation set, with just the one false negative. The AIC for Model 3 is 110, compared to 155 for Model 2. But the fact that there are dummy variables that completely predict the response with 100% accuracy for large subsets of the data, such as **indus19**, is causing problems for the linear model. We'll move onto the model selection phase to discuss how we'll make predictions for the evaluation data.

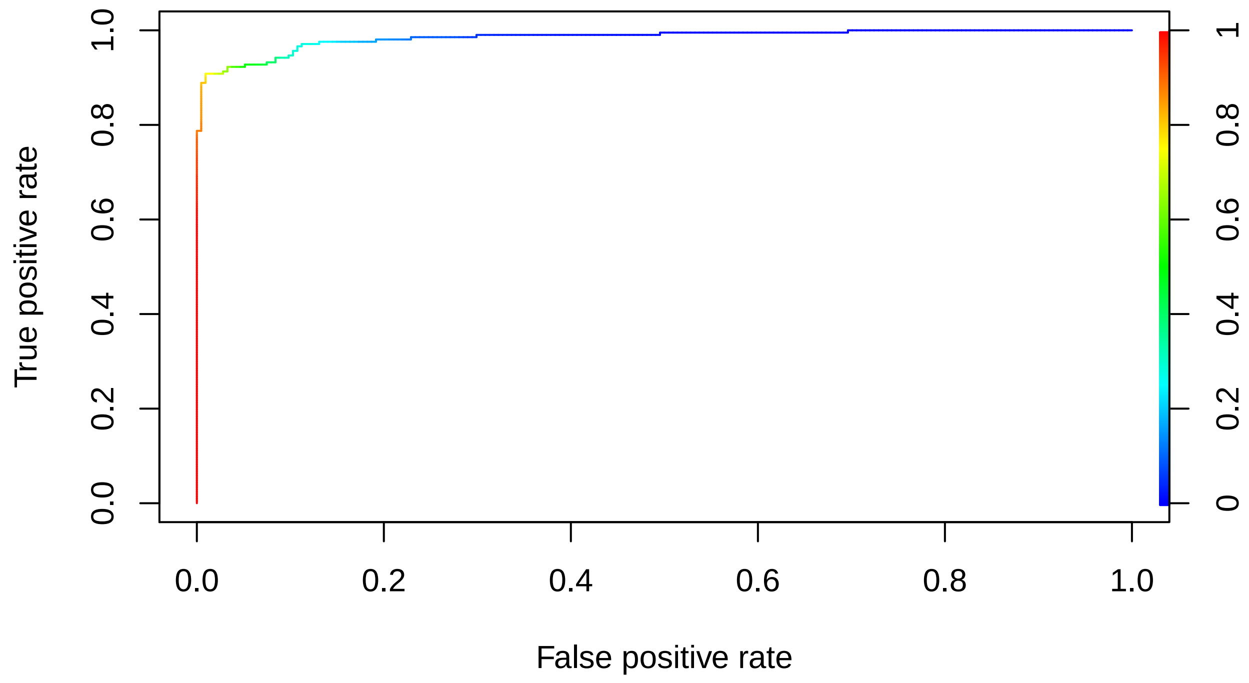
4. Select Models

Our best predictions on the validation data were from Models 1 and 3. Our best AIC score for an interpretable model is for Model 2, which uses the logarithmic transforms, has an AIC of 152, but makes 1 more error than the other models on the validation set. It's AUC is .9846, as shown here:

```
p <- predict(mod_7, type = "response")
roc_pred <- prediction(predictions = p,labels=mod_7$y)
auc.tmp <- performance(roc_pred,"auc"); auc <- as.numeric(auc.tmp@y.values)
auc
```

```
## [1] 0.9845591
```

```r
#plotting roc
roc_perf <- performance(roc_pred , "tpr" , "fpr")
plot(roc_perf,
     colorize = TRUE,
     #print.cutoffs.at= seq(0,1,0.05),
     text.adj=c(-0.2,1.7))
```

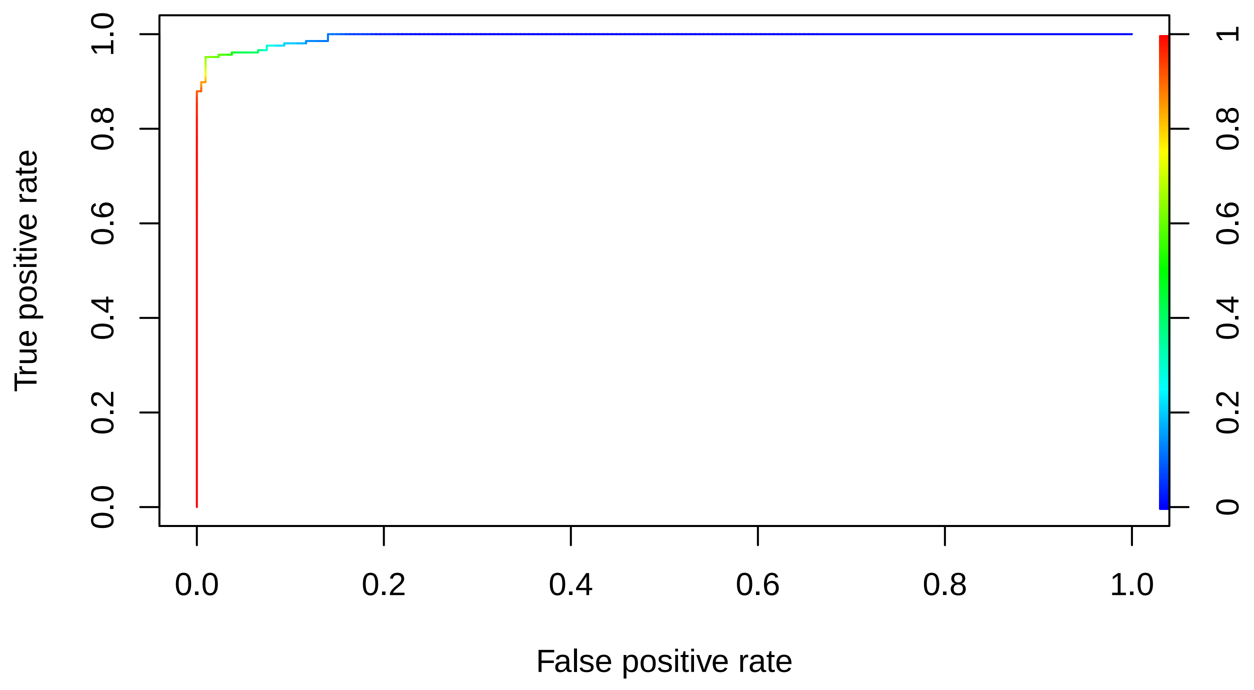


Model 3, meanwhile, uses nonlinear patterns in the data to essentially approximate a decision tree model, which is much better suited to this dataset. The glm model using its dummy variables makes perfect use of the information handed to it, but produces hard-to-understand results. Nevertheless, it makes good predictions, with a much lower AIC of 110, and an AUC score of .995, as shown here:

```r
p <- predict(mod_4, type = "response")
roc_pred <- prediction(predictions = p,labels=mod_4$y)
auc.tmp <- performance(roc_pred,"auc"); auc <- as.numeric(auc.tmp@y.values)
auc
```

```
## [1] 0.9950336
```

```r
#plotting roc
roc_perf <- performance(roc_pred , "tpr" , "fpr")
plot(roc_perf,
     colorize = TRUE,
     #print.cutoffs.at= seq(0,1,0.05),
     text.adj=c(-0.2,1.7))
```



All in all we see Model selection is the problem of choosing one from among a set of candidate models. It is common to choose a model that performs the best on a hold-out test dataset or to estimate model performance using a resampling technique, such as k-fold cross-validation. An alternative approach to model selection involves using probabilistic statistical measures that attempt to quantify both the model performance on the training dataset and the complexity of the model. One such example is AIC. The benefit of an information criterion statistic is that it does not require a hold-out test set, although a limitation is that it doesn't take the uncertainty of the model into account and may end-up leading to selecting a model that is too simple or too complex.

If our goal is to make the most accurate predictions possible, we will obviously choose Model 3. But instead, for a linear model course project, we'll use Model 2, since it's actually a linear model that makes sense, even if the dataset isn't appropriate for a linear model. Here are our evaluation predictions for Model 2:

```r
crime_eval_df['log_dis'] = log(crime_eval_df$dis)
crime_eval_df['log_tax'] = log(crime_eval_df$tax)
crime_eval_df['log_ptrat'] = log(crime_eval_df$ptratio)
crime_eval_df['log_lstat'] = log(crime_eval_df$lstat)
eval.pred = predict(mod_7, newdata = crime_eval_df, type = "response")

probs = eval.pred
preds = 1 * (eval.pred > .5)
evalOutput = data.frame('Probability'=probs, 'Predictions'= preds)

evalOutput
```

```
##     Probability Predictions
## 1  2.350293e-03           0
## 2  8.509902e-01           1
## 3  8.460342e-01           1
## 4  9.868598e-01           1
## 5  1.994312e-01           0
## 6  3.158239e-01           0
## 7  4.394515e-01           0
## 8  1.075639e-03           0
## 9  6.084014e-04           0
## 10 8.485025e-05           0
## 11 1.025941e-01           0
## 12 8.639141e-02           0
## 13 9.288431e-01           1
## 14 8.033777e-01           1
## 15 8.242191e-01           1
## 16 1.189475e-01           0
## 17 9.713287e-01           1
## 18 9.730168e-01           1
## 19 4.784282e-02           0
## 20 1.633514e-06           0
## 21 1.504951e-08           0
## 22 2.326209e-02           0
## 23 2.712710e-01           0
## 24 5.217162e-01           1
## 25 4.376334e-01           0
## 26 9.185217e-02           0
## 27 4.714218e-05           0
## 28 9.998573e-01           1
## 29 9.987911e-01           1
## 30 9.977558e-01           1
## 31 9.999993e-01           1
## 32 9.999866e-01           1
## 33 9.999866e-01           1
## 34 9.999777e-01           1
## 35 9.999916e-01           1
## 36 9.999855e-01           1
## 37 9.999825e-01           1
## 38 9.999325e-01           1
## 39 6.950442e-01           1
## 40 2.889035e-01           0
```