Homework 3 (Group 5)

Binary Logistic Regression

Maria A Ginorio

3/30/2022

Contents

Overview

In this homework assignment, you 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).

Your 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. You will provide classifications and probabilities for the evaluation data set using your binary logistic regression model. You can only use the variables given to you (or variables that you derive from the variables provided). Below is a short description of the variables of interest in the data set:

Dataset

- zn: proportion of residential land zoned for large lots (over 25000 square feet) (predictor variable)
- indus: proportion of non-retail business acres per suburb (predictor variable)
- chas: a dummy var. for whether the suburb borders the Charles River (1) or not (0) (predictor variable)
- nox: nitrogen oxides concentration (parts per 10 million) (predictor variable)
- rm: average number of rooms per dwelling (predictor variable)
- age: proportion of owner-occupied units built prior to 1940 (predictor variable)
- dis: weighted mean of distances to five Boston employment centers (predictor variable)
- rad: index of accessibility to radial highways (predictor variable)
- tax: full-value property-tax rate per $10,000 (predictor variable)
- ptratio: pupil-teacher ratio by town (predictor variable)
- black: $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town (predictor variable)
- lstat: lower status of the population (percent) (predictor variable)
- medv: median value of owner-occupied homes in $1000s (predictor variable)
- target: whether the crime rate is above the median crime rate (1) or not (0) (response variable)

1. Data Exploration

Objective

- Understand the variables provided
- Build a binary logistic regression model on the training data
- Predict the whether the neighborhood will be at risk for high crime.
- Provide classifications and probabilities for the evaluation data set using logistic regression.

Data Overview

Lets first look at the raw data values by using the skim package

Table 1: Data summary

Name	crime_train
Number of rows	466
Number of columns	13
Column type frequency:	
numeric	13
Group variables	None

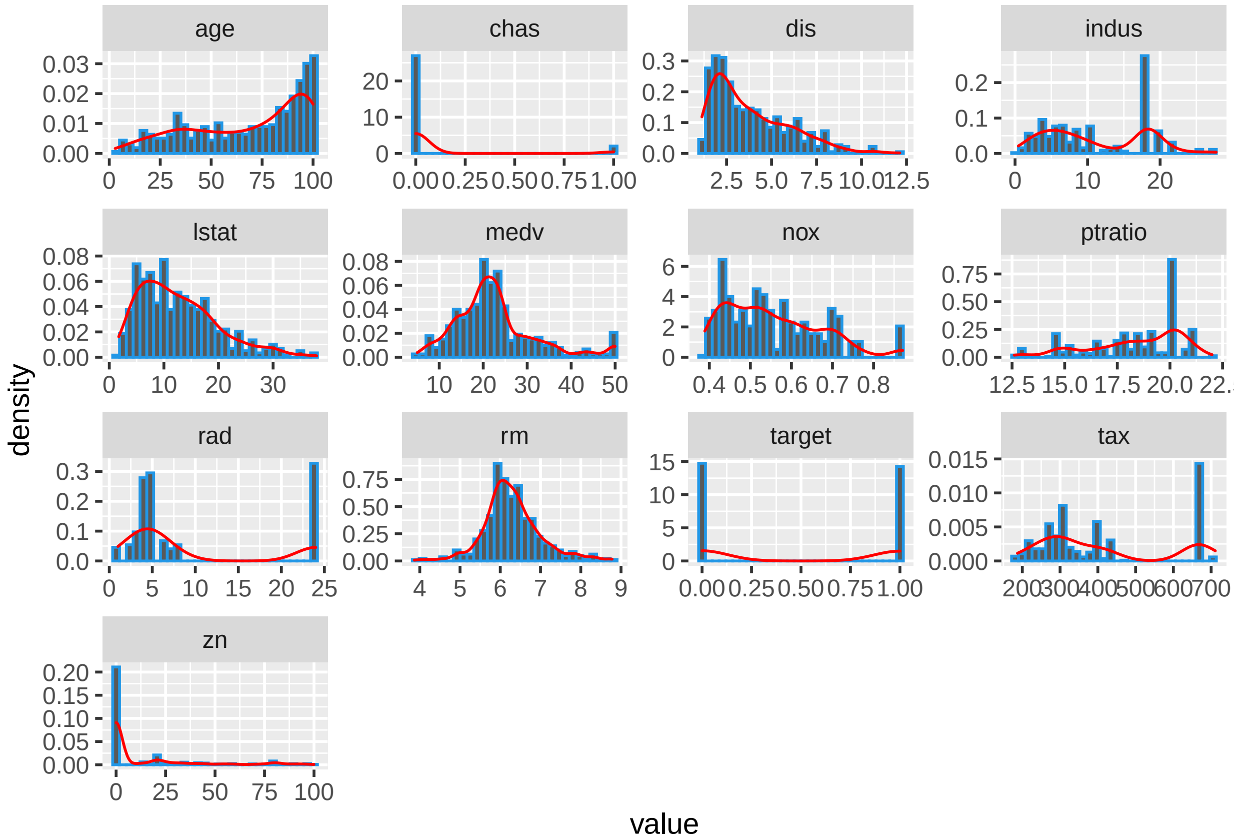
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
zn	0	1	11.58	23.36	0.00	0.00	0.00	16.25	100.00	
indus	0	1	11.11	6.85	0.46	5.15	9.69	18.10	27.74	
chas	0	1	0.07	0.26	0.00	0.00	0.00	0.00	1.00	
nox	0	1	0.55	0.12	0.39	0.45	0.54	0.62	0.87	
rm	0	1	6.29	0.70	3.86	5.89	6.21	6.63	8.78	
age	0	1	68.37	28.32	2.90	43.88	77.15	94.10	100.00	
dis	0	1	3.80	2.11	1.13	2.10	3.19	5.21	12.13	
rad	0	1	9.53	8.69	1.00	4.00	5.00	24.00	24.00	
tax	0	1	409.50	167.90	187.00	281.00	334.50	666.00	711.00	
ptratio	0	1	18.40	2.20	12.60	16.90	18.90	20.20	22.00	
lstat	0	1	12.63	7.10	1.73	7.04	11.35	16.93	37.97	
medv	0	1	22.59	9.24	5.00	17.02	21.20	25.00	50.00	
target	0	1	0.49	0.50	0.00	0.00	0.00	1.00	1.00	

From the description seen by the skim package we can observe we have two variables that should be transformed into factors since they have (1) or (0) values. `chas & target.`

Distributions

We will first explore the data looking for issues or challenges (i.e. missing data, outliers, possible coding errors, multicollinearlity, etc). Once we have a handle on the data, we will apply any necessary cleaning steps. Once we have a reasonable dataset to work with, we will build and evaluate three different Logistic models that predict seasonal wins.



The distribution of our variables can also alert us of of unusual patterns, in this case we have observed the prevalence of kurtosis for certain variables like:, `nox,` `Istat`, `rad,` `zn` are skewed to the right. In addition, `ptratio` and `age` are left skewed.

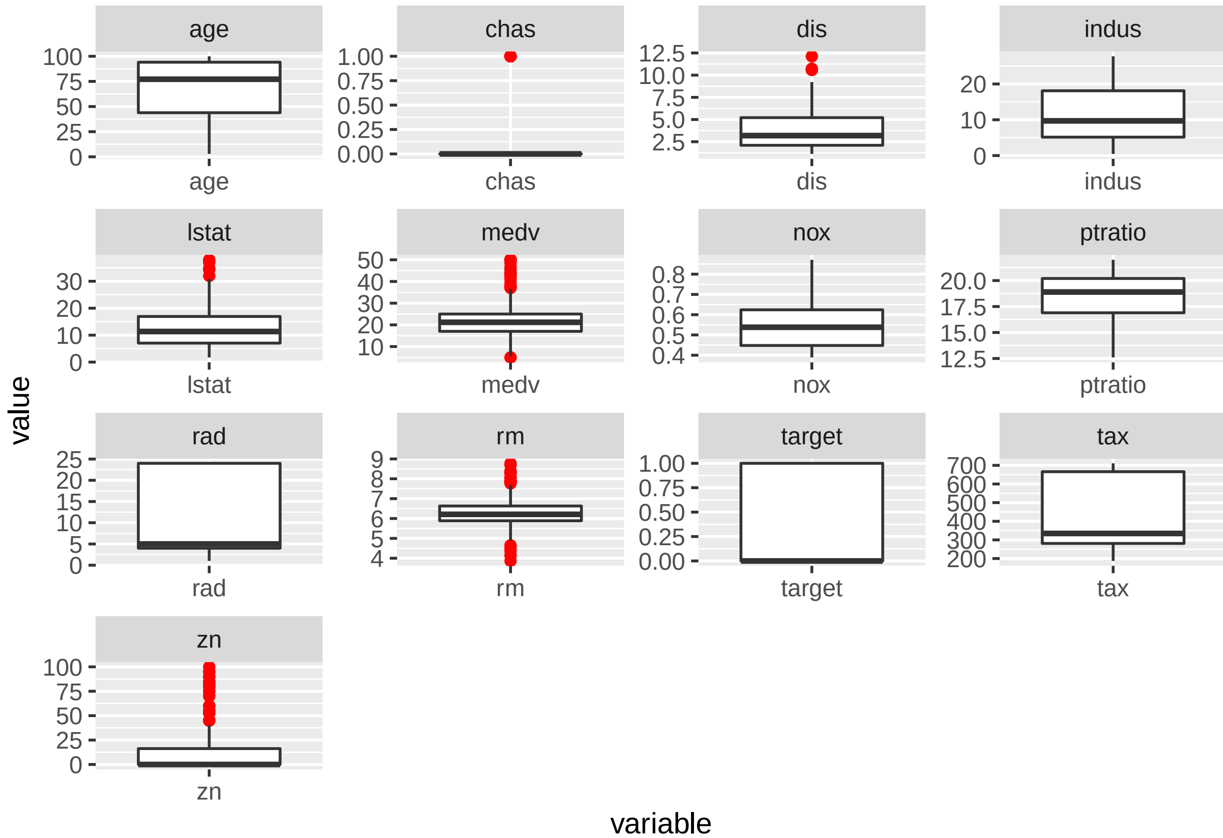
After creating independent histograms for each variable we have found 2 variables that appear to be bi-modal. We notice that the graphs of this variables have two distinct humps or peaks with a valley separating them. We could attribute this observations to possibly different groups. We find that `rad` and `tax` are bi-modal.

Outliers

In addition to histogram graph of our variable we thought it was pertinent to take a look at our variables using a boxplot. It will help us quickly visualize the distribution of the values in the dataset and see where the five number summary values are located.

In addition, we will be able to create a clear picture of the median values and the spreads across all the distributions. One of the most important observation we will obtain from this graph however, is outlier detection.

Find outliers in red below:



Indication of outliers is present in variables `chas`, `dis, lstat, medv, rm, zn`

A key is whether an outlier represents a contaminated observation or a rare case.

Are these data points unusual or different in some way from the rest of the data? We will have to consider removing this and refit the data if we consider they could be affecting our results.

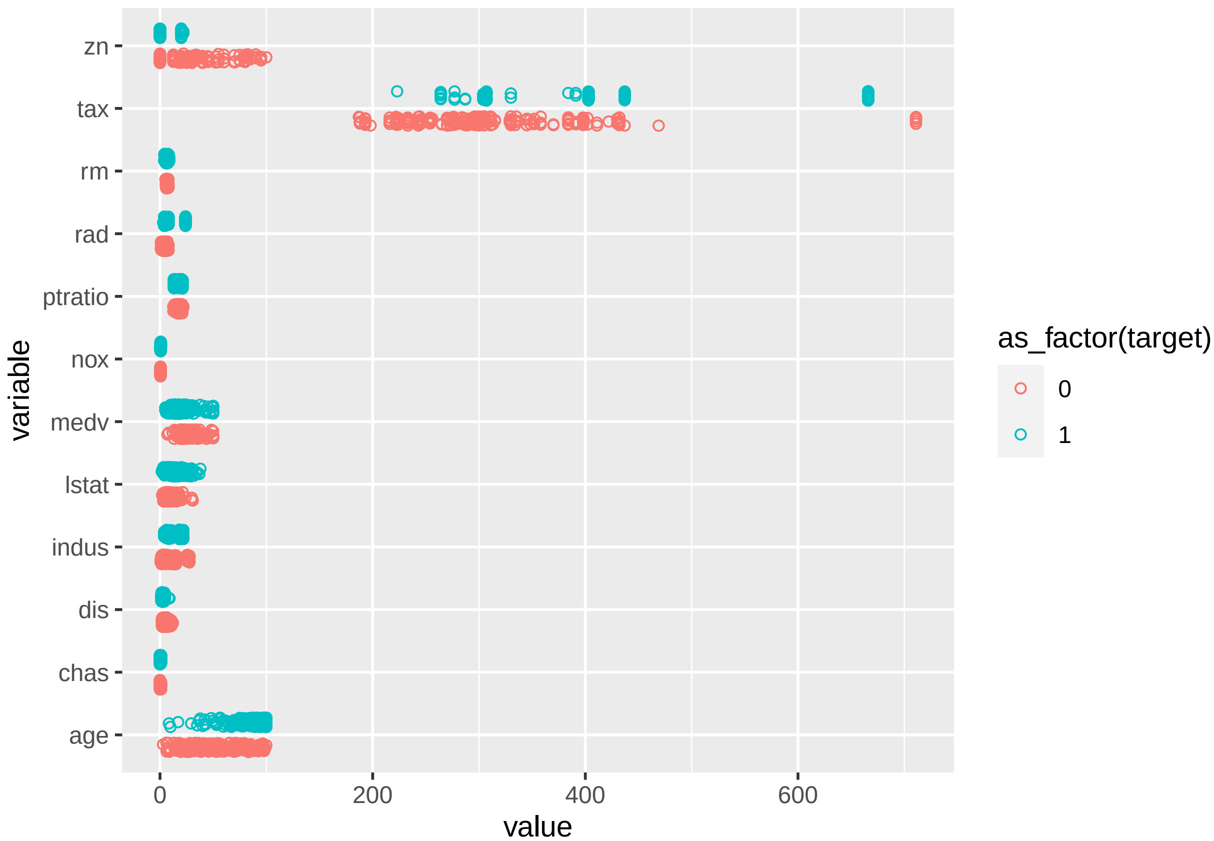
One of the first steps in any type of analysis is to take a closer look at the observations that have high leverage since they could have a large impact on the results of a given model.

Relationships

We want use scatter plots in each variable versus the target variable to get an idea of the relationship between them.

The plots indicate interesting relationship between the `target` variable however some of them start showing signs of relationship and groups.

Some of the predictors variables are skewed and not normally distributed, in addition we have outliers and bimodality.

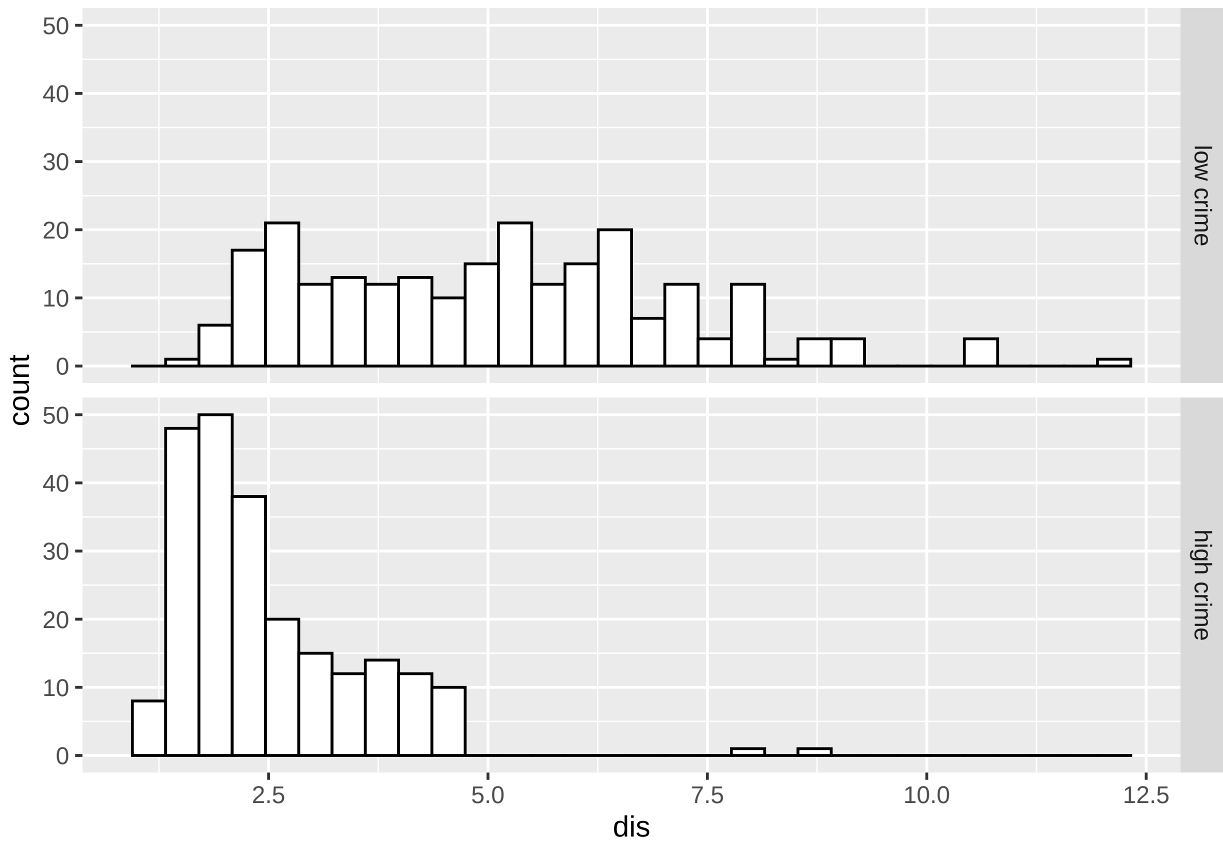


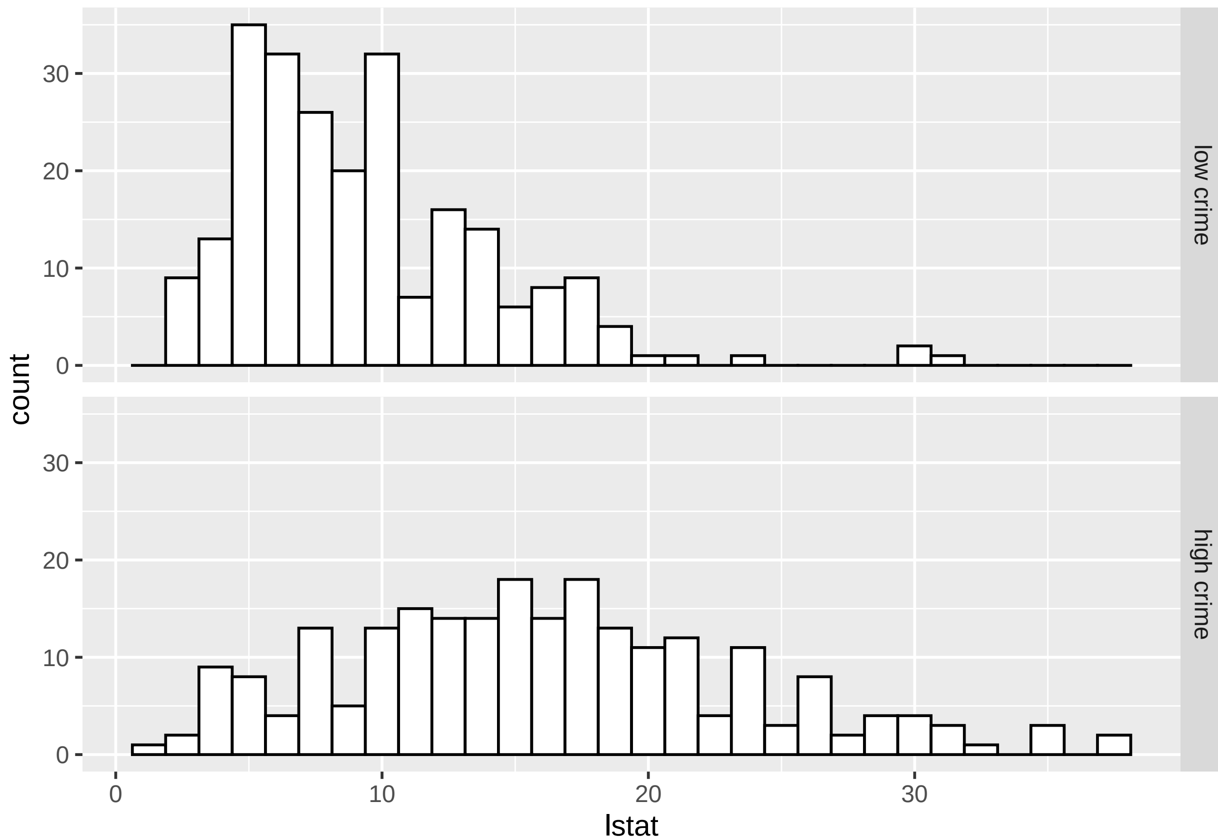
We take some of the variables to be analyzed separately against the target

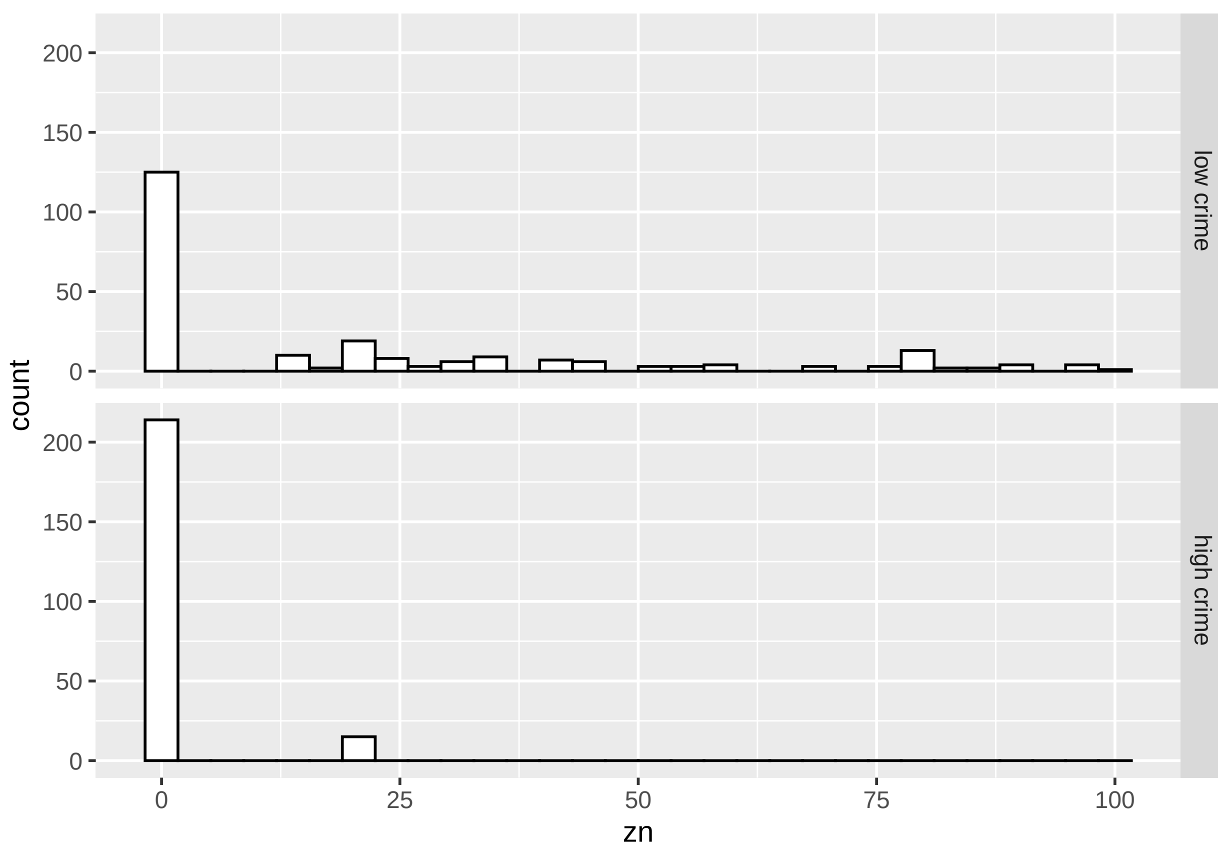
Skeweness

Histogram of predictors by factors of target

dis - weighted mean of distances to five Boston employment centers

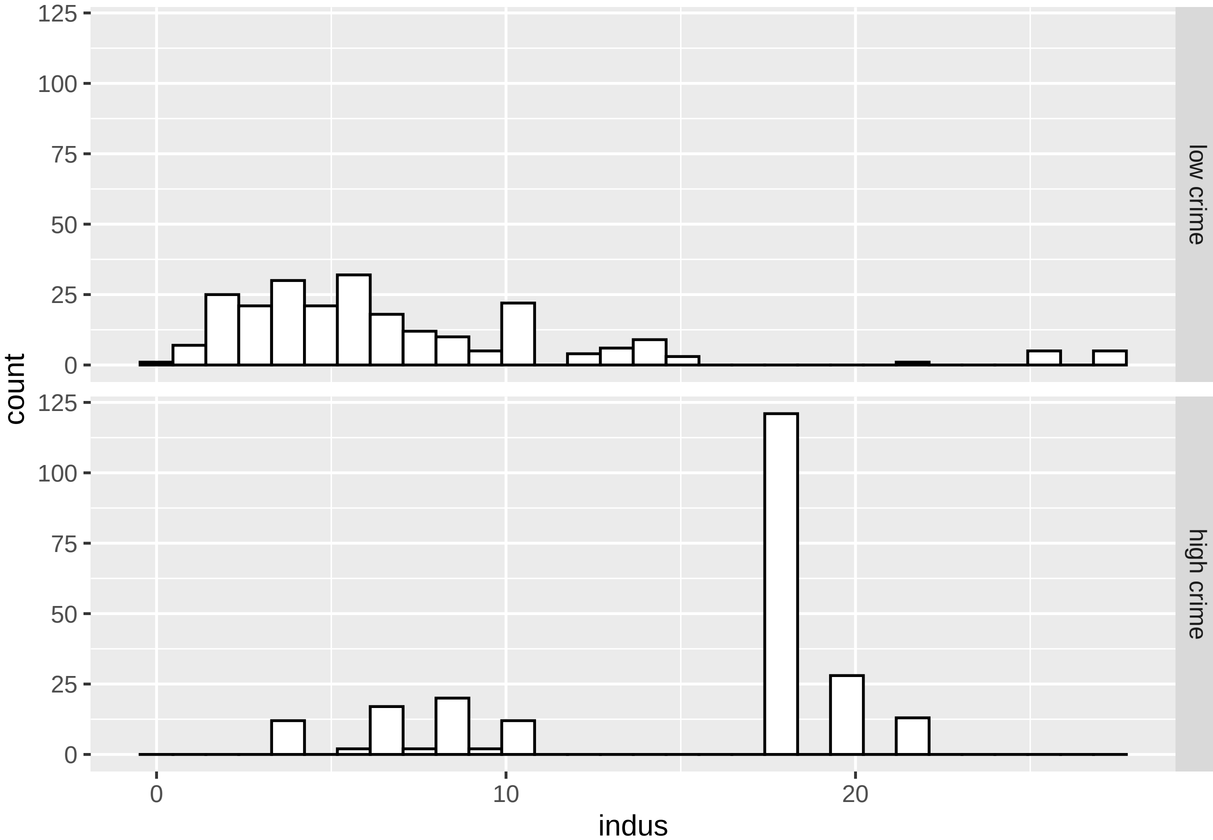






Compared to other predictors we can observe that `zn`has a lot of zero values.

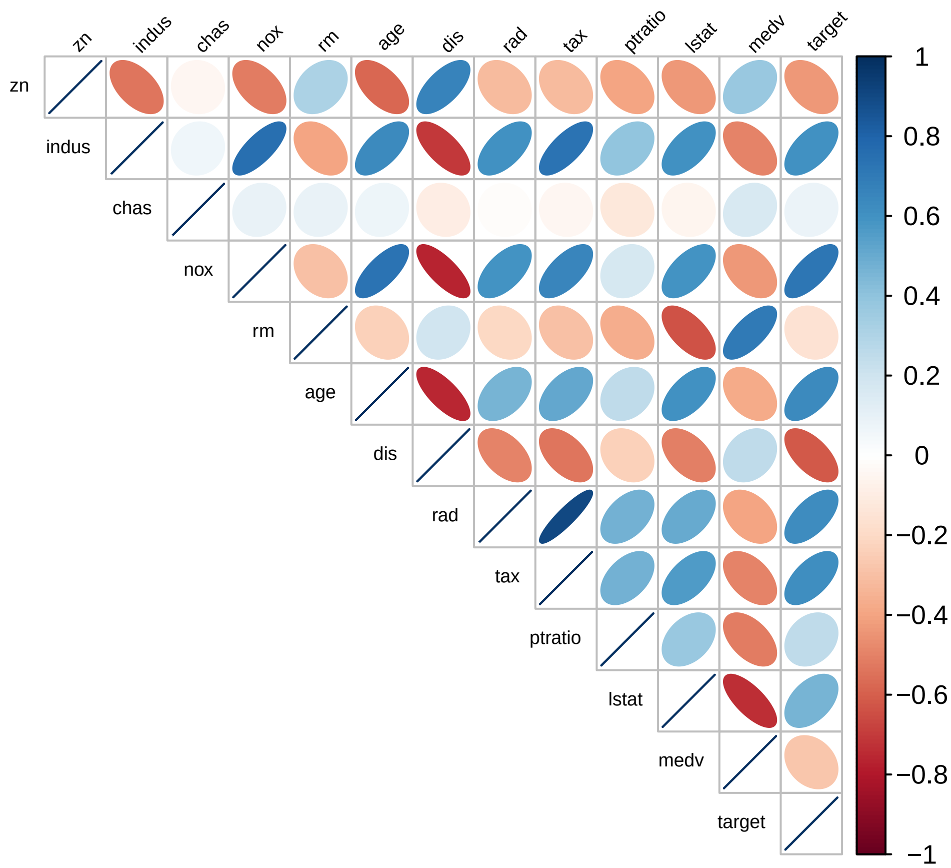
If there is a over dispersion of zeroes, where the peak of the curve is highest at zero we might have to consider a negative binomial. If zn is a significant predictor it could be an option to create negative binomial model using PSCL Package.



Indus - proportion of non retail business acres per suburb.

We can observe the skeweness of `indus` as well and look at the distribution by factors of target we can see high crime in 125 observations of indus.

Multicolinearity



We can see that some variables are highly correlated with one another, such as `rad` & `tax`.

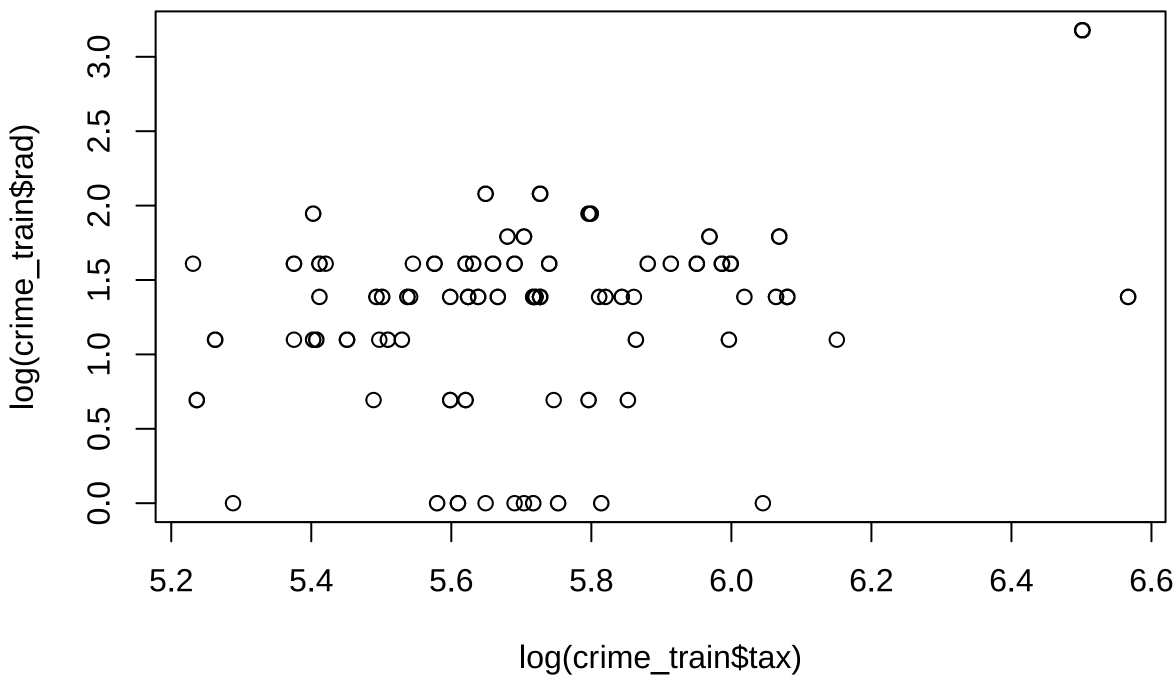
When we start considering features for our models, we'll need to account for the correlations between features and avoid including pairs with strong correlations.

Many features are also inherently associated, for example, as the distance to employment centers increase, we would expect a decrease in non-retail businesses acres, `nox` concentration and owner occupied units metrics. As median value owner-occupied homes increase, we would expect to see decreases in `lstat` - lower status of the population.

The target variable has linear correlation with `indus, nox, age, rad, tax and lstat`

Correlation

Earlier we discovered the correlation between tax and rad. We want to understand this relationship better by plotting them.



The plot of correlation between rad and tax shoes 90% of the relationship is made by the influential points. This predictors are not really correlated.

```
##
##  Pearson's product-moment correlation
##
## data:  crime_train$tax and crime_train$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
```

Regarding the strength of the relationship: The **more extreme** the correlation coefficient (the closer to -1 or 1), the **stronger the relationship**. This also means that a **correlation close to 0** indicates that the two variables are **independent**, that is, as one variable increases, there is no tendency in the other variable to either decrease or increase.

The *p*-value of the correlation test between these 2 variables is 2.2e-16. At the 5% significance level, we do not reject the null hypothesis of no correlation. We therefore conclude that we do not reject the hypothesis that there is no linear relationship between the 2 variables.

This test proves that even if the correlation coefficient is different from 0 (the correlation is 0.9 in the sample), it is actually not significantly different from 0 in the population.

The larger the sample size and the more extreme the correlation (closer to -1 or 1), the more likely the null hypothesis of no correlation will be rejected. With a small sample size, it is thus possible to obtain a *relatively* large correlation in the sample (based on the correlation coefficient), but still find a correlation not significantly different from 0 in the population (based on the correlation test). For this reason, it is recommended to always perform a correlation test before interpreting a correlation coefficient to avoid flawed conclusions.

2. Data Preparation

Missing Data

To prepare our data we have already determined that we do not have any missing data in our dataset.

See below:

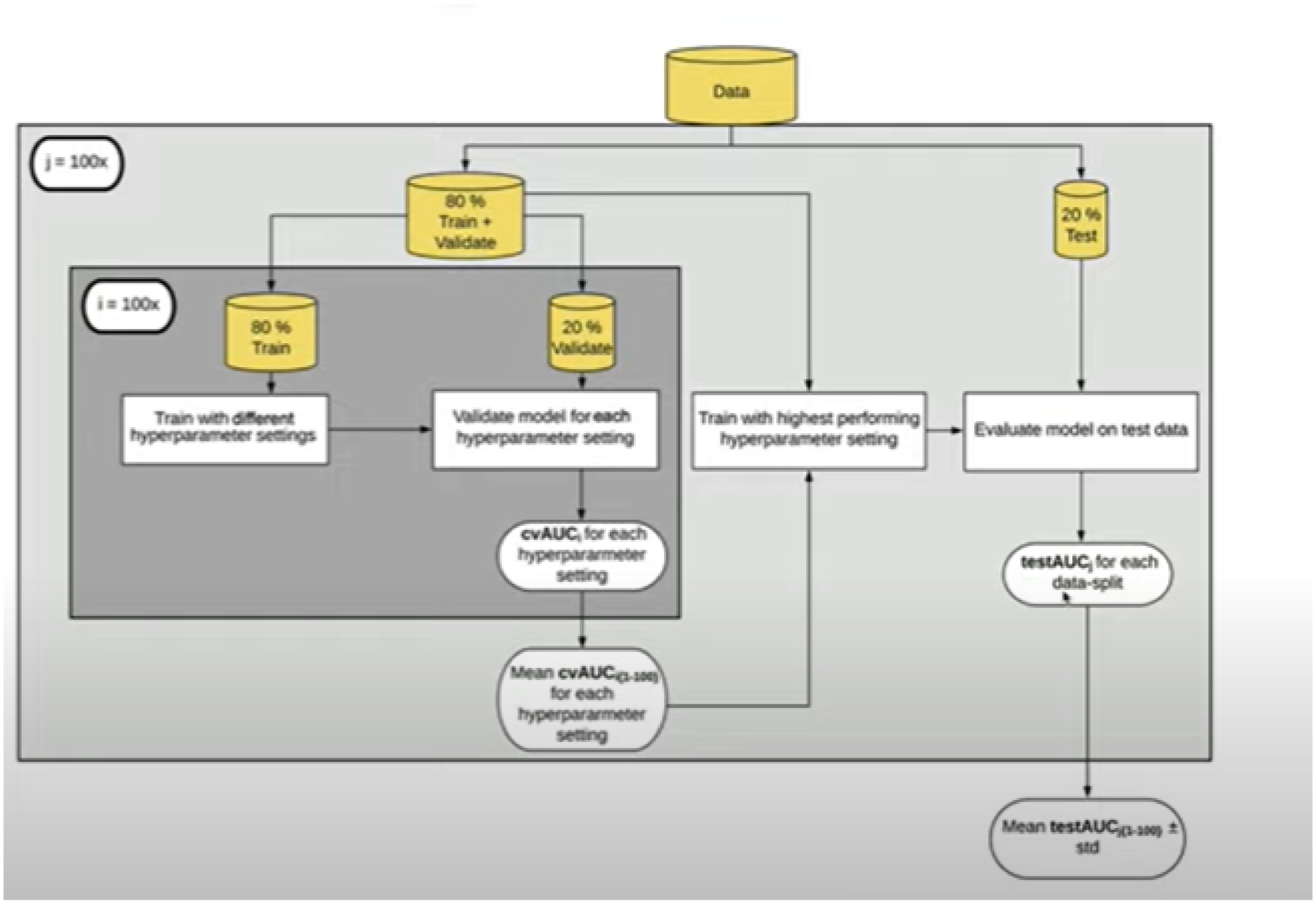
variable	total	isna	num.isna	pct
age	466	FALSE	466	100
chas	466	FALSE	466	100
dis	466	FALSE	466	100
indus	466	FALSE	466	100
lstat	466	FALSE	466	100
medv	466	FALSE	466	100
nox	466	FALSE	466	100
ptratio	466	FALSE	466	100
rad	466	FALSE	466	100
rm	466	FALSE	466	100
target	466	FALSE	466	100
tax	466	FALSE	466	100
zn	466	FALSE	466	100

Correlation

In order to determine the best predictor for our model we need to detect which are the predictor variables with low correlation value. We use the corrr package to determine all the variables with values <0.10. This will allow us to only manipulate the variables that have significance to our model.

```
##       term target
## 1       zn   -.47
## 2    indus    .62
## 3     chas    .08
## 4      nox    .75
## 5       rm   -.18
## 6      age    .65
## 7      dis   -.66
## 8      rad    .58
## 9      tax    .60
## 10 ptratio    .36
## 11   lstat    .48
## 12    medv   -.40
```

Preprocess



Lets take a look at our dataset now.

```
## # A tibble: 2 x 3
##   target count  prop
##    <dbl> <int> <dbl>
## 1      0   237 0.509
## 2      1   229 0.491
```

Table 4: Data summary

Crime Rate above median	Target var codes	Percent Frequency
Yes	1	51%
No	0	49%

Name	crime_train
Number of rows	466
Number of columns	13
Column type frequency:	
numeric	13

Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
zn	0	1	11.58	23.36	0.00	0.00	0.00	16.25	100.00
indus	0	1	11.11	6.85	0.46	5.15	9.69	18.10	27.74
chas	0	1	0.07	0.26	0.00	0.00	0.00	0.00	1.00
nox	0	1	0.55	0.12	0.39	0.45	0.54	0.62	0.87
rm	0	1	6.29	0.70	3.86	5.89	6.21	6.63	8.78
age	0	1	68.37	28.32	2.90	43.88	77.15	94.10	100.00
dis	0	1	3.80	2.11	1.13	2.10	3.19	5.21	12.13
rad	0	1	9.53	8.69	1.00	4.00	5.00	24.00	24.00
tax	0	1	409.50	167.90	187.00	281.00	334.50	666.00	711.00
ptratio	0	1	18.40	2.20	12.60	16.90	18.90	20.20	22.00
lstat	0	1	12.63	7.10	1.73	7.04	11.35	16.93	37.97
medv	0	1	22.59	9.24	5.00	17.02	21.20	25.00	50.00
target	0	1	0.49	0.50	0.00	0.00	0.00	1.00	1.00

Partition

The first thing we will do is to divide our training data into two parts: train set and test set. Our partition will be 70%, 30%

Caret provides us the `CreateDataPartition()` function for this, which will allow us to partition based on the proportion from the response variable.

Table 7: Data summary

Name	trainSet
Number of rows	327
Number of columns	13
Column type frequency:	
numeric	13
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
zn	0	1	11.52	23.04	0.00	0.00	0.00	20.00	100.00
indus	0	1	11.30	6.83	0.46	5.19	9.90	18.10	27.74
chas	0	1	0.06	0.23	0.00	0.00	0.00	0.00	1.00
nox	0	1	0.56	0.12	0.39	0.45	0.54	0.63	0.87
rm	0	1	6.29	0.67	4.37	5.88	6.21	6.63	8.72
age	0	1	69.36	28.41	2.90	45.75	78.90	94.80	100.00
dis	0	1	3.79	2.14	1.17	2.11	3.10	5.22	12.13
rad	0	1	9.75	8.76	1.00	4.00	5.00	24.00	24.00

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
tax	0	1	415.65	169.05	187.00	284.00	358.00	666.00	711.00
ptratio	0	1	18.35	2.25	12.60	16.60	18.70	20.20	22.00
lstat	0	1	12.86	7.02	1.73	7.38	11.66	17.09	34.77
medv	0	1	22.48	9.32	5.00	16.60	21.10	24.90	50.00
target	0	1	0.50	0.50	0.00	0.00	0.00	1.00	1.00

Hot Encoding

Next we will use what is known as "one hot encoding" to transform the dummy variables. In our case we only have 1 - `chas`

The output will be a matrix of the predictors, which omits the response variable.

Table 9: Data summary

Name	trainSet_X
Number of rows	327
Number of columns	12
Column type frequency:	
numeric	12
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
zn	0	1	11.52	23.04	0.00	0.00	0.00	20.00	100.00
indus	0	1	11.30	6.83	0.46	5.19	9.90	18.10	27.74
chas	0	1	0.06	0.23	0.00	0.00	0.00	0.00	1.00
nox	0	1	0.56	0.12	0.39	0.45	0.54	0.63	0.87
rm	0	1	6.29	0.67	4.37	5.88	6.21	6.63	8.72
age	0	1	69.36	28.41	2.90	45.75	78.90	94.80	100.00
dis	0	1	3.79	2.14	1.17	2.11	3.10	5.22	12.13
rad	0	1	9.75	8.76	1.00	4.00	5.00	24.00	24.00
tax	0	1	415.65	169.05	187.00	284.00	358.00	666.00	711.00
ptratio	0	1	18.35	2.25	12.60	16.60	18.70	20.20	22.00
lstat	0	1	12.86	7.02	1.73	7.38	11.66	17.09	34.77
medv	0	1	22.48	9.32	5.00	16.60	21.10	24.90	50.00

Normalization

Typically we **normalize** data when performing some type of analysis in which we have multiple variables that are measured on different scales and we want each of the variables to have the same range.

This prevents one variable from being overly influential (in our case tax and rad), especially if it's measured in different units (i.e. if one variable is measured in inches and another is measured in yards).

Table 11: Data summary

Name	trainSet_X
Number of rows	327
Number of columns	12
Column type frequency:	
numeric	12
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
zn	0	1	0.12	0.23	0	0.00	0.00	0.20	1
indus	0	1	0.40	0.25	0	0.17	0.35	0.65	1
chas	0	1	0.06	0.23	0	0.00	0.00	0.00	1
nox	0	1	0.35	0.24	0	0.12	0.31	0.50	1
rm	0	1	0.44	0.15	0	0.35	0.42	0.52	1
age	0	1	0.68	0.29	0	0.44	0.78	0.95	1
dis	0	1	0.24	0.20	0	0.09	0.18	0.37	1
rad	0	1	0.38	0.38	0	0.13	0.17	1.00	1
tax	0	1	0.44	0.32	0	0.19	0.33	0.91	1
ptratio	0	1	0.61	0.24	0	0.43	0.65	0.81	1
lstat	0	1	0.34	0.21	0	0.17	0.30	0.47	1
medv	0	1	0.39	0.21	0	0.26	0.36	0.44	1

Now we will make the final training set by adding this to our original response variable (target)

Our last step will be to transform the test set as well. We will use the same procedures.

- One hot encoding - using the `dummyModel`
- Normalization using `rangeModel`object

Feature Elimination

Nest thing that we need to consider is low information features. If uniformative, useless features are included in the dataset, this will almost always lead to a decrease in the model's performance.

In this case we will use the Recursive Feature Elimination. The function to implement this is the `rfe()`

Recursive Feature Elimination works by building many models of a type of machine learning method on the training set, and iteratively re-calculating the most important variables.

We will use the random forest approach to rank the features.

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables   RMSE Rsquared     MAE  RMSESD RsquaredSD   MAESD Selected
##          1 0.1917   0.8376 0.07239 0.07623     0.1080 0.03302
##          2 0.1786   0.8586 0.08194 0.07540     0.1099 0.03318
##          3 0.1700   0.8673 0.07112 0.07901     0.1127 0.03422        *
##          4 0.1709   0.8650 0.07069 0.08084     0.1159 0.03502
##          5 0.1747   0.8633 0.07385 0.07461     0.1082 0.03385
##          7 0.1722   0.8669 0.08168 0.07264     0.1087 0.03383
##          8 0.1725   0.8685 0.09045 0.07191     0.1117 0.03422
##          9 0.1758   0.8668 0.09255 0.06975     0.1090 0.03366
##         10 0.1791   0.8634 0.09772 0.06806     0.1091 0.03336
##         12 0.1786   0.8640 0.09560 0.06739     0.1066 0.03262
##
## The top 3 variables (out of 3):
##    nox, rad, indus
```

Top 5 Variables to Use
nox
rad
indus
tax
dis

3. Building Models

- Dependent Variable: Whether or not the crime rate is above median crime rate
- Independent variables: all predictors described earlier.

Top 5 Variables to Use
nox
rad
indus
tax
dis

Model # 1

We will begin our first model using all the predictors to see the level of significance of each one of them. This model will include original tax and all the variables in the dataset

Logit Model

```
##
## Call:
## glm(formula = trainSet$target ~ trainSet$nox + trainSet$indus +
##     trainSet$rad + trainSet$tax + trainSet$dis, family = binomial(link = "logit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.72520  -0.32264  -0.04691   0.01152   2.55355
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -7.6074     1.4509  -5.243 1.58e-07 ***
## trainSet$nox    18.2624     3.6618   4.987 6.12e-07 ***
## trainSet$indus   0.4585     1.5864   0.289 0.772576
## trainSet$rad    12.1080     3.4047   3.556 0.000376 ***
## trainSet$tax    -4.2341     1.6232  -2.608 0.009095 **
## trainSet$dis     2.2632     1.9165   1.181 0.237652
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 453.29  on 326  degrees of freedom
## Residual deviance: 159.16  on 321  degrees of freedom
## AIC: 171.16
##
## Number of Fisher Scoring iterations: 8
```

Crime Rate Above median	Logit Coefficients	Std. Error
indus (prop non-retail business acres/suburb)	0.4585	1.5864
nox (nitrogen oxide concentration)	18.2624	3.6618

Crime Rate Above median	Logit Coefficients	Std. Error
dis	2.2632	1.9165
(wmu dist to empl centers)		
rad (access radial highways)	12.1080	3.4047
tax (full value prop tax)	-4.2341	1.6232

- (*) Indicates significance at the 5% level
- Coefficient Interpretation: higher taxes are **less likely** to have crime rate above median. Higher proportion of indus, nox rad, dis are **more likely** to have crime rate above median.

Model #2

```
##
## Call:
## glm(formula = trainSet$target ~ trainSet$nox + trainSet$indus +
##     trainSet$rad + trainSet$tax + trainSet$dis, family = binomial(link = "probit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.65917  -0.31411  -0.01082   0.00018   2.59866
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -4.4201     0.7793  -5.672 1.41e-08 ***
## trainSet$nox    10.3377     1.9009   5.438 5.38e-08 ***
## trainSet$indus   0.5180     0.8897   0.582 0.560374
## trainSet$rad     7.0039     2.0227   3.463 0.000535 ***
## trainSet$tax    -2.5443     0.9470  -2.687 0.007213 **
## trainSet$dis     1.4488     1.0731   1.350 0.176971
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 453.29  on 326  degrees of freedom
## Residual deviance: 158.27  on 321  degrees of freedom
## AIC: 170.27
##
## Number of Fisher Scoring iterations: 10
```

Crime Rate Above median	Probit Coefficients	Std. Error
nox	10.3377	0.7793
dis	1.4488	1.0731
rad	7.0039	2.0227
tax	-2.5443	0.9470
indus	0.5180	0.8897

- (*) Indicates significance at the 5% level
- Coefficient Interpretation: Higher proportion in nox, dis, rad are **more likely** to have crime rates above median. Meanwhile higher taxes are **less likely** to have crime rate above median.

Model #3

Caret GBM Model

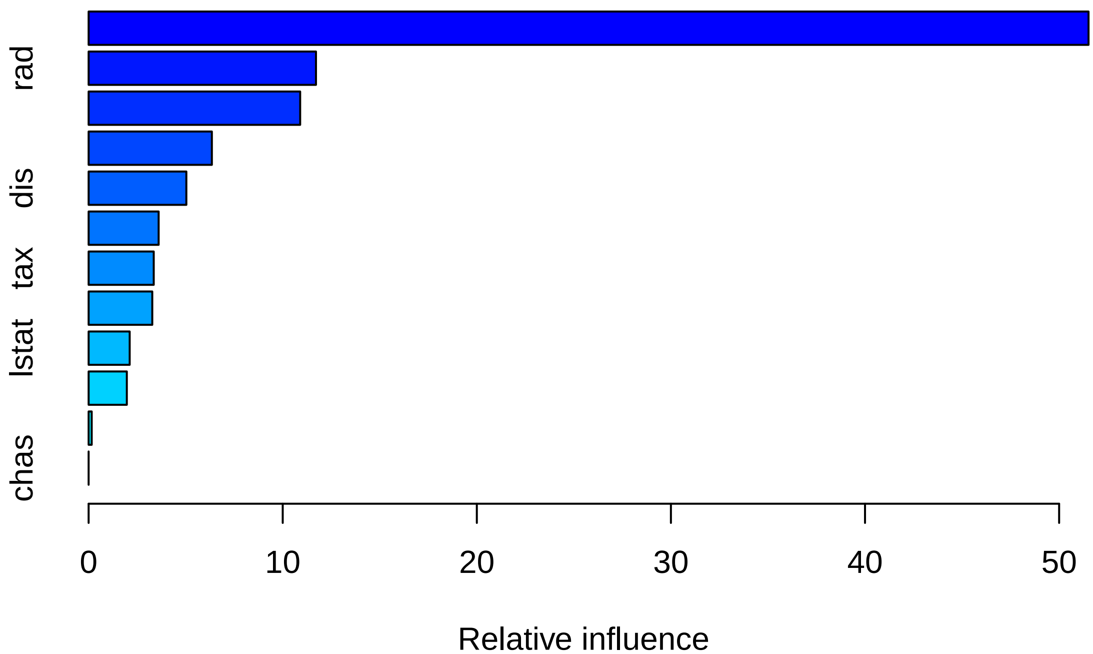
I will use the Caret package to train the GBM model, as this is the package that best supports the odds plot statistics.

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2789             nan     0.1000    0.0532
##      2        1.1844             nan     0.1000    0.0415
##      3        1.1079             nan     0.1000    0.0358
##      4        1.0466             nan     0.1000    0.0299
##      5        0.9932             nan     0.1000    0.0208
##      6        0.9346             nan     0.1000    0.0204
##      7        0.8852             nan     0.1000    0.0163
##      8        0.8411             nan     0.1000    0.0181
##      9        0.8067             nan     0.1000    0.0166
##     10        0.7753             nan     0.1000    0.0129
##     20        0.5791             nan     0.1000    0.0041
##     40        0.4427             nan     0.1000    0.0006
##     60        0.3699             nan     0.1000    0.0003
##     80        0.3193             nan     0.1000   -0.0014
##    100        0.2885             nan     0.1000   -0.0014
##    120        0.2642             nan     0.1000   -0.0002
##    140        0.2356             nan     0.1000   -0.0020
##    150        0.2273             nan     0.1000   -0.0012
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2635             nan     0.1000    0.0555
##      2        1.1618             nan     0.1000    0.0501
##      3        1.0751             nan     0.1000    0.0399
##      4        1.0021             nan     0.1000    0.0284
##      5        0.9391             nan     0.1000    0.0302
##      6        0.8973             nan     0.1000    0.0145
##      7        0.8506             nan     0.1000    0.0209
##      8        0.8044             nan     0.1000    0.0203
##      9        0.7643             nan     0.1000    0.0166
##     10        0.7292             nan     0.1000    0.0164
##     20        0.4992             nan     0.1000    0.0114
##     40        0.3093             nan     0.1000    0.0014
##     60        0.2260             nan     0.1000   -0.0023
##     80        0.1815             nan     0.1000   -0.0018
##    100        0.1463             nan     0.1000   -0.0013
##    120        0.1190             nan     0.1000   -0.0025
##    140        0.0997             nan     0.1000   -0.0018
##    150        0.0908             nan     0.1000   -0.0014
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2579             nan     0.1000    0.0580
##      2        1.1510             nan     0.1000    0.0456
##      3        1.0472             nan     0.1000    0.0399
##      4        0.9683             nan     0.1000    0.0320
##      5        0.8970             nan     0.1000    0.0338
##      6        0.8392             nan     0.1000    0.0206
##      7        0.7793             nan     0.1000    0.0283
##      8        0.7388             nan     0.1000    0.0172
##      9        0.7053             nan     0.1000    0.0114
```

```
##     10        0.6663             nan     0.1000    0.0168
##     20        0.4141             nan     0.1000    0.0059
##     40        0.2202             nan     0.1000   -0.0006
##     60        0.1386             nan     0.1000   -0.0001
##     80        0.1034             nan     0.1000   -0.0002
##    100        0.0761             nan     0.1000   -0.0005
##    120        0.0530             nan     0.1000   -0.0012
##    140        0.0415             nan     0.1000   -0.0009
##    150        0.0358             nan     0.1000   -0.0006
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2904             nan     0.1000    0.0504
##      2        1.2070             nan     0.1000    0.0394
##      3        1.1435             nan     0.1000    0.0278
##      4        1.0710             nan     0.1000    0.0311
##      5        1.0164             nan     0.1000    0.0252
##      6        0.9683             nan     0.1000    0.0179
##      7        0.9262             nan     0.1000    0.0194
##      8        0.8936             nan     0.1000    0.0138
##      9        0.8565             nan     0.1000    0.0161
##     10        0.8292             nan     0.1000    0.0129
##     20        0.6334             nan     0.1000    0.0022
##     40        0.4849             nan     0.1000    0.0019
##     60        0.4134             nan     0.1000   -0.0029
##     80        0.3493             nan     0.1000   -0.0013
##    100        0.3054             nan     0.1000    0.0009
##    120        0.2799             nan     0.1000   -0.0018
##    140        0.2580             nan     0.1000   -0.0019
##    150        0.2451             nan     0.1000   -0.0001
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2673             nan     0.1000    0.0539
##      2        1.1696             nan     0.1000    0.0379
##      3        1.0937             nan     0.1000    0.0325
##      4        1.0280             nan     0.1000    0.0271
##      5        0.9683             nan     0.1000    0.0271
##      6        0.9226             nan     0.1000    0.0190
##      7        0.8777             nan     0.1000    0.0197
##      8        0.8290             nan     0.1000    0.0207
##      9        0.7835             nan     0.1000    0.0187
##     10        0.7441             nan     0.1000    0.0183
##     20        0.5367             nan     0.1000    0.0015
##     40        0.3437             nan     0.1000    0.0015
##     60        0.2625             nan     0.1000   -0.0012
##     80        0.2082             nan     0.1000   -0.0019
##    100        0.1658             nan     0.1000   -0.0012
##    120        0.1356             nan     0.1000   -0.0005
##    140        0.1118             nan     0.1000   -0.0007
##    150        0.1022             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2627             nan     0.1000    0.0608
##      2        1.1573             nan     0.1000    0.0437
##      3        1.0699             nan     0.1000    0.0421
##      4        0.9942             nan     0.1000    0.0307
##      5        0.9287             nan     0.1000    0.0232
```

```
##      6        0.8672             nan     0.1000    0.0277
##      7        0.8102             nan     0.1000    0.0259
##      8        0.7668             nan     0.1000    0.0180
##      9        0.7241             nan     0.1000    0.0166
##     10        0.6904             nan     0.1000    0.0117
##     20        0.4561             nan     0.1000    0.0016
##     40        0.2752             nan     0.1000   -0.0017
##     60        0.1968             nan     0.1000   -0.0032
##     80        0.1446             nan     0.1000   -0.0014
##    100        0.1055             nan     0.1000   -0.0006
##    120        0.0789             nan     0.1000   -0.0005
##    140        0.0595             nan     0.1000   -0.0006
##    150        0.0521             nan     0.1000    0.0002
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2756             nan     0.1000    0.0501
##      2        1.1996             nan     0.1000    0.0278
##      3        1.1191             nan     0.1000    0.0418
##      4        1.0489             nan     0.1000    0.0301
##      5        0.9951             nan     0.1000    0.0248
##      6        0.9513             nan     0.1000    0.0161
##      7        0.9016             nan     0.1000    0.0220
##      8        0.8565             nan     0.1000    0.0211
##      9        0.8189             nan     0.1000    0.0159
##     10        0.7871             nan     0.1000    0.0137
##     20        0.5797             nan     0.1000    0.0038
##     40        0.4230             nan     0.1000   -0.0001
##     60        0.3485             nan     0.1000   -0.0020
##     80        0.2974             nan     0.1000   -0.0003
##    100        0.2572             nan     0.1000   -0.0013
##    120        0.2217             nan     0.1000   -0.0018
##    140        0.1999             nan     0.1000   -0.0000
##    150        0.1915             nan     0.1000    0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2610             nan     0.1000    0.0596
##      2        1.1690             nan     0.1000    0.0469
##      3        1.0817             nan     0.1000    0.0436
##      4        1.0088             nan     0.1000    0.0362
##      5        0.9516             nan     0.1000    0.0250
##      6        0.8960             nan     0.1000    0.0242
##      7        0.8412             nan     0.1000    0.0231
##      8        0.7927             nan     0.1000    0.0231
##      9        0.7497             nan     0.1000    0.0200
##     10        0.7114             nan     0.1000    0.0164
##     20        0.4919             nan     0.1000    0.0107
##     40        0.2983             nan     0.1000    0.0007
##     60        0.2180             nan     0.1000   -0.0020
##     80        0.1676             nan     0.1000   -0.0007
##    100        0.1374             nan     0.1000   -0.0012
##    120        0.1080             nan     0.1000   -0.0008
##    140        0.0880             nan     0.1000   -0.0005
##    150        0.0796             nan     0.1000   -0.0008
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2652             nan     0.1000    0.0534
```

```
##      2        1.1635             nan     0.1000    0.0480
##      3        1.0750             nan     0.1000    0.0404
##      4        0.9922             nan     0.1000    0.0299
##      5        0.9192             nan     0.1000    0.0243
##      6        0.8624             nan     0.1000    0.0278
##      7        0.8117             nan     0.1000    0.0174
##      8        0.7648             nan     0.1000    0.0189
##      9        0.7166             nan     0.1000    0.0197
##     10        0.6718             nan     0.1000    0.0169
##     20        0.4210             nan     0.1000    0.0033
##     40        0.2239             nan     0.1000    0.0011
##     60        0.1423             nan     0.1000   -0.0016
##     80        0.1040             nan     0.1000   -0.0015
##    100        0.0731             nan     0.1000   -0.0009
##    120        0.0572             nan     0.1000   -0.0006
##    140        0.0441             nan     0.1000   -0.0001
##    150        0.0376             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2487             nan     0.1000    0.0654
##      2        1.1462             nan     0.1000    0.0493
##      3        1.0571             nan     0.1000    0.0411
##      4        0.9854             nan     0.1000    0.0353
##      5        0.9175             nan     0.1000    0.0329
##      6        0.8479             nan     0.1000    0.0332
##      7        0.7944             nan     0.1000    0.0249
##      8        0.7483             nan     0.1000    0.0218
##      9        0.7051             nan     0.1000    0.0168
##     10        0.6732             nan     0.1000    0.0121
##     20        0.4247             nan     0.1000    0.0051
##     40        0.2604             nan     0.1000   -0.0003
##     60        0.1830             nan     0.1000    0.0005
##     80        0.1416             nan     0.1000   -0.0010
##    100        0.1168             nan     0.1000   -0.0015
##    120        0.0890             nan     0.1000   -0.0002
##    140        0.0717             nan     0.1000   -0.0004
##    150        0.0647             nan     0.1000   -0.0005
```




```
##             var    rel.inf
## nox         nox 51.5156428
## rad         rad 11.7135394
## indus     indus 10.8992739
## ptratio ptratio  6.3493003
## dis         dis  5.0357683
## age         age  3.6077953
## tax         tax  3.3549456
## rm           rm  3.2797074
## lstat     lstat  2.1152329
## medv       medv  1.9689997
## zn           zn  0.1597944
## chas       chas  0.0000000
```

```
## Stochastic Gradient Boosting
##
## 327 samples
##  12 predictor
##   2 classes: 'above', 'below'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 218, 218, 218
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.9616162  0.8641975  0.9030303
##   1                  100      0.9732884  0.9197531  0.9454545
```

```
##   1                  150      0.9735129  0.9382716  0.9272727
##   2                   50      0.9753086  0.9444444  0.9454545
##   2                  100      0.9728395  0.9444444  0.9515152
##   2                  150      0.9716049  0.9444444  0.9515152
##   3                   50      0.9755331  0.9320988  0.9636364
##   3                  100      0.9749719  0.9320988  0.9515152
##   3                  150      0.9762065  0.9382716  0.9515152
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

Model # 4

GBM model used above utlilizes boosted trees for classification. Now we will use GLMnet from caret package. GLMnet uses Regression & classification. We will choose to use regression

```
## Warning in train.default(trainSet[, 2:13], trainSet[, "target"], method =
## "glmnet", : You are trying to do regression and your outcome only has two
## possible values Are you trying to do classification? If so, use a 2 level factor
## as your outcome column.
```

```
##             Length Class      Mode
## a0           78    -none-     numeric
## beta        936    dgCMatrix  S4
## df           78    -none-     numeric
## dim           2    -none-     numeric
## lambda       78    -none-     numeric
## dev.ratio    78    -none-     numeric
## nulldev       1    -none-     numeric
## npasses       1    -none-     numeric
## jerr          1    -none-     numeric
## offset        1    -none-     logical
## call          5    -none-     call
## nobs          1    -none-     numeric
## lambdaOpt     1    -none-     numeric
## xNames       12    -none-     character
## problemType   1    -none-     character
## tuneValue     2    data.frame list
## obsLevels     1    -none-     logical
## param         0    -none-     list
```

4. Select Models

Model	AIC	AUC	Accuracy
Logit	171.16	-	0.8838
Probit	170.27	-	0.8807
GBM (classification)	-	0.9892	0.9712230
GLMnet (regression)	-	0.9515	

Predicted Probabilities

Logit and Probit

```
## Warning: package 'stargazer' was built under R version 4.1.2

##
## Please cite as:

##  Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.

##  R package version 5.2.3. https://CRAN.R-project.org/package=stargazer

##
## ==================================================
## Statistic         N  Mean  St. Dev.   Min    Max
## --------------------------------------------------
## target           327 0.495  0.501      0      1
## targethat_logit  327 0.495  0.415    0.001  1.000
## targethat_probit 327 0.496  0.415   0.00002 1.000
## --------------------------------------------------

##   target targethat_logit targethat_probit
## 1      1      0.80836716       0.80534779
## 2      1      0.99998844       1.00000000
## 3      1      0.99999944       1.00000000
## 4      0      0.04213553       0.04013261
## 5      0      0.07571503       0.06992085
```

Logit

```
##   [1] 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0
##  [38] 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1
##  [75] 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1
## [112] 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1
## [149] 1 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 1 1 0 0
## [186] 0 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0
## [223] 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0
## [260] 1 0 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0
## [297] 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1
## Levels: 0 1
```

```
##   [1] 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0
##  [38] 1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1
##  [75] 0 1 1 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1
## [112] 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1
## [149] 1 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 1 0
## [186] 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 0 1 0
## [223] 1 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0
## [260] 1 0 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0
## [297] 1 1 1 1 1 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1
## Levels: 0 1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 153  26
##          1  12 136
##
##                Accuracy : 0.8838
##                  95% CI : (0.844, 0.9164)
##     No Information Rate : 0.5046
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7674
##
##  Mcnemar's Test P-Value : 0.03496
##
##             Sensitivity : 0.8395
##             Specificity : 0.9273
##          Pos Pred Value : 0.9189
##          Neg Pred Value : 0.8547
##              Prevalence : 0.4954
##          Detection Rate : 0.4159
##    Detection Prevalence : 0.4526
##       Balanced Accuracy : 0.8834
##
##        'Positive' Class : 1
##
```

Probit

```
##   [1] 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0
##  [38] 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1
##  [75] 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1
## [112] 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1
## [149] 1 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 1 1 0 0
## [186] 0 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0
## [223] 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0
## [260] 1 0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0
## [297] 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 1
## Levels: 0 1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 153  27
##          1  12 135
##
##                Accuracy : 0.8807
##                  95% CI : (0.8406, 0.9138)
##     No Information Rate : 0.5046
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7612
##
##  Mcnemar's Test P-Value : 0.02497
##
##             Sensitivity : 0.8333
##             Specificity : 0.9273
##          Pos Pred Value : 0.9184
##          Neg Pred Value : 0.8500
##              Prevalence : 0.4954
##          Detection Rate : 0.4128
##    Detection Prevalence : 0.4495
##       Balanced Accuracy : 0.8803
##
##        'Positive' Class : 1
##
```

Model_3

There are two types of evaluation we can do here, `raw` or `prob`. **Raw** gives you a class prediction, in our case `above` and `below`, while **prob** gives you the probability on how sure the model is about it's choice. I always use **prob**, as I like to be in control of the threshold and also like to use AUC score which requires probabilities, not classes. There are situations where having class values can come in handy, such as with multinomial models where you're predicting more than two values.

We now call the `predict` function and pass it our trained model and our testing data. Let's start by looking at class predictions and using the **caret** `postResample` function to get an accuracy score:

Get Predictions on testing Data

Class Predictions

```
## [1] below above above above above below
## Levels: above below
```

```
##  Accuracy     Kappa
## 0.9712230 0.9423715
```

Probabilities Predictions

```
##         above        below
## 1 0.082545030 0.9174549703
## 2 0.996415363 0.0035846374
## 3 0.999053023 0.0009469770
## 4 0.849784859 0.1502151413
## 5 0.999076732 0.0009232681
## 6 0.001632334 0.9983676660
```

```
##     RMSE Rsquared      MAE
##       NA 0.894008       NA
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls > cases
```

```
## Area under the curve: 0.9892
```

The **AUC** is telling us that our model has a **0.9892 AUC** score (remember that an **AUC** ranges between **0.5** and **1**, where **0.5** is random and **1** is perfect).

Model_4

Let's change gears and try this out on a regression model. Let's look at what modeling types **glmnet** supports and reset our outcome variable as we're going to be using the numerical outcome instead of the factor.

This is a less strong **AUC** score than our previous **gbm** model. Testing with different types of models does pay off (take it with a grain of salt as we didn't tune our models much).

```
##         1         2         3         4         5         6
## 0.3905949 1.0316777 1.0504249 0.4317653 1.0171748 0.1973471
```

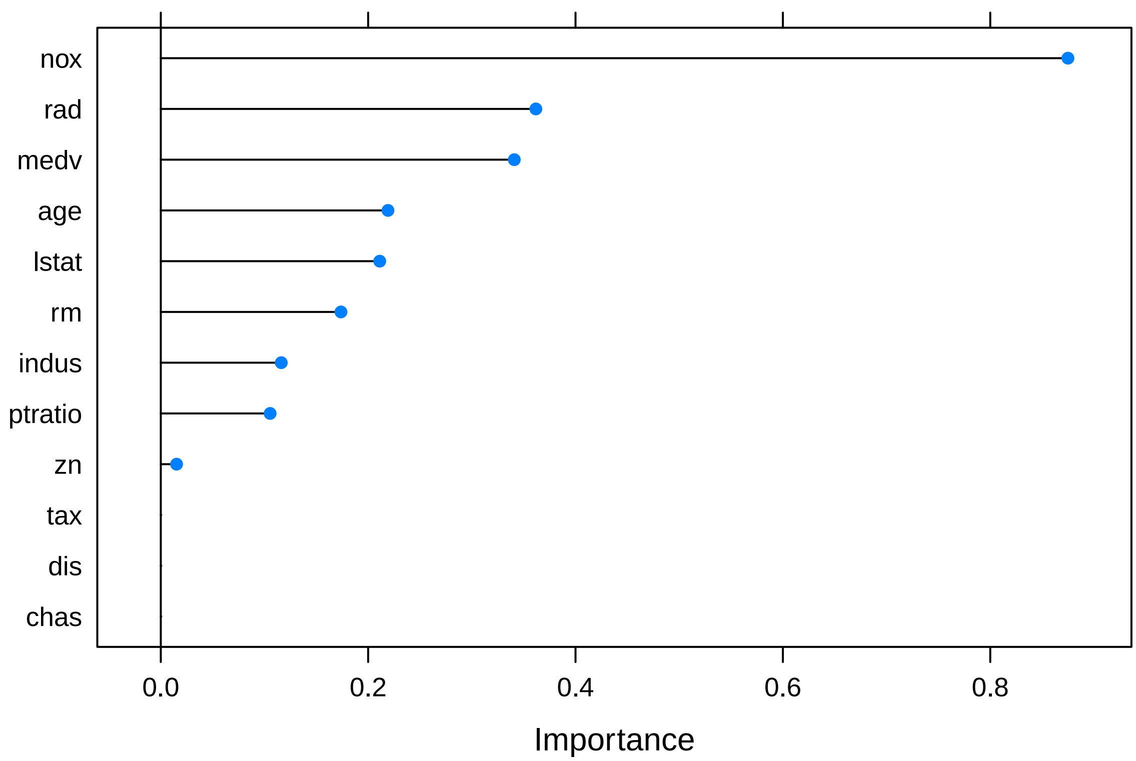
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## Area under the curve: 0.9515
```

You can also call the **caret** function `varImp` to figure out the variables that were important to the model. And this is one great feature of the **glmnet** model; it returns positive and negative variable importance unlike most models. This helps deepens your understanding about your variables, such that being `nox` leans the probabilities in being above crime median rate favor while `zn` is the least likely.

Surprisingly enough in the GLM model tax has influence. When we saw that it does have some significance in the other models.



Final Selection

References

Apendix

```
knitr::opts_chunk$set(echo = FALSE)

library(tidyverse)
library(skimr)
library(tinytex)
library(e1071)
library(ggthemes)
library(caret)
crime_train <- read_csv("https://raw.githubusercontent.com/mgino11/Business_Analytics/main/Projects/PROJEC

crime_eval <- read_csv("https://raw.githubusercontent.com/mgino11/Business_Analytics/main/Projects/PROJECT_

skim(crime_train)


crime_dist <- crime_train %>%
  pivot_longer(
    everything(),
    names_to = c("variable"),
    values_to = "value"
  )

ggplot(crime_dist, aes(value)) +
  geom_histogram(aes(x=value, y = ..density..),
                 colour = 4, bins = 30) +
  geom_density(aes(x=value), color = "red") +
  facet_wrap(~variable, scales = "free")

ggplot(crime_dist, aes(value, variable)) +
  geom_boxplot(outlier.color = "red") +
  facet_wrap(~variable, scales = "free", drop = FALSE)+
  coord_flip()
crime_scatter <- crime_train %>%
  pivot_longer(
    cols = -target,
    names_to = c("variable"),
    values_to = "value")
ggplot(crime_scatter, aes(x = value,
                          y = variable,
                          color = as_factor(target))) +
  geom_jitter(position = position_jitterdodge(dodge.width = 0.8,
                                              jitter.width = 0.3),
            shape=21  )
crime_skew <- crime_train
  #convert target to factor and new names
  crime_skew$target <- recode_factor(
    crime_skew$target, '0' = 'low crime','1' = 'high crime' )
```

```
ggplot(crime_skew, aes(x=dis)) +
  geom_histogram(fill = 'white', colour = 'black') +
  facet_grid(target ~ .)
ggplot(crime_skew, aes(x=lstat)) +
  geom_histogram(fill = 'white', colour = 'black') +
  facet_grid(target ~ .)
ggplot(crime_skew, aes(x=zn)) +
  geom_histogram(fill = 'white', colour = 'black') +
  facet_grid(target ~ .)
ggplot(crime_skew, aes(x=indus)) +
  geom_histogram(fill = 'white', colour = 'black') +
  facet_grid(target ~ .)

library(corrplot)

corrplot(corr = cor(crime_train,
                    use = 'pairwise.complete.obs'),
         method = "ellipse",
         type = "upper",
         order = "original",
         tl.col = "black",
         tl.srt = 45,
         tl.cex = 0.55)
plot(log(crime_train$tax), log(crime_train$rad))
cor.test(crime_train$tax, crime_train$rad, method = "pearson")
crime_na <- crime_train %>%
  pivot_longer(
    everything(),
    names_to = c("variable"),
    values_to = "value" ) %>%
  mutate(isna = is.na(value)) %>%
  group_by(variable) %>%
  mutate(total = n()) %>%
  group_by(variable,total,isna) %>%
  summarise(num.isna = n()) %>%
  mutate(pct = num.isna / total * 100)
knitr::kable(crime_na)

library(corrr)
crime_corr <- correlate(crime_train,
                        use = "pairwise.complete.obs",
                        method = "spearman")


crime_corr %>%
  focus(target) %>%
  fashion()
crime_train %>%
  group_by(target) %>%
  summarise(count = n() ) %>%
  mutate( prop = count / sum(count) )
skim_without_charts(crime_train)
set.seed(1188)
trainIndex <- createDataPartition(crime_train$target, p = 0.7, list = F)
trainSet <- crime_train[trainIndex,]
```

```
testSet <- crime_train[-trainIndex,]
skim_without_charts(trainSet)
dummyModel <- dummyVars(target ~ ., data = trainSet)
trainSet_X <- as.data.frame(predict(dummyModel, newdata = trainSet))
skim_without_charts(trainSet_X)
rangeModel <- preProcess(trainSet_X, method = "range")
trainSet_X <- predict(rangeModel, newdata = trainSet_X)
skim_without_charts(trainSet_X)
trainSet <- cbind(trainSet$target, trainSet_X)
names(trainSet)[1] <- "target"
testset_dummy <- predict(dummyModel, testSet)
testset_range <- predict(rangeModel, testset_dummy)
testset_range <- as.data.frame(testset_range)
testSet <- cbind(testSet$target, testset_range)
names(testSet) <- names(trainSet)


subset <- c(1:5, 7, 8, 9, 10, 12, 13)

set.seed(1188)

rfectrl <- rfeControl(functions = rfFuncs,
                      method = "cv",
                      verbose = F)

rfProfile <- rfe(x = trainSet[,2:13],
                 y = trainSet$target,
                 sizes = subset,
                 rfeControl = rfectrl)


rfProfile
logit<- glm(formula = trainSet$target ~ trainSet$nox +
      trainSet$indus +
      trainSet$rad +
      trainSet$tax +
      trainSet$dis,
    family = "binomial"
    (link = "logit"))
summary(logit)

probit <- glm(formula = trainSet$target ~ trainSet$nox +
      trainSet$indus +
      trainSet$rad +
      trainSet$tax +
      trainSet$dis,
    family = "binomial"
    (link = "probit"))
summary(probit)
trainSet$target2 <- as.factor(ifelse(trainSet$target==0, 'below', 'above'))
outcomeName <- 'target2'

testSet$target2 <- as.factor(ifelse(testSet$target==0, 'below', 'above'))


objControl <- trainControl(method = 'cv',
```

```r
                              number = 3,
                              returnResamp = 'none',
                              summaryFunction = twoClassSummary,
                              classProbs = T)
set.seed(1188)

gbmModel <- train(trainSet[,2:13], trainSet[,"target2"],
                  method = "gbm",
                  trControl = objControl,
                  metric = "ROC")

summary(gbmModel)
print(gbmModel)

objControl_reg <- objControl <- trainControl(method = 'cv',
                              number = 3,
                              returnResamp = 'none')
regModel <- train(trainSet[,2:13], trainSet[,"target"],
                  method = "glmnet",
                  metric = "RMSE")

summary(regModel)
predTrain <- trainSet %>%
  mutate(targethat_logit = fitted(logit)) %>%
  mutate(targethat_probit = fitted(probit))

library(stargazer)

predTrain %>%
  select(target, targethat_logit, targethat_probit) %>%
  stargazer(type = "text")
predTrain %>%
  select(target, targethat_logit, targethat_probit) %>%
  head(5)
(pred_logit <- (fitted(logit) > 0.5) %>% as.numeric %>% as.factor)
(actual <- trainSet$target %>% as.factor)
confusionMatrix(pred_logit,actual, positive = "1")
(pred_probit <- (fitted(probit) > 0.5) %>% as.numeric %>% as.factor)
confusionMatrix(pred_probit,actual, positive = "1")

pred_gbm <- predict(object = gbmModel, testSet[,2:13], type = 'raw')
head(pred_gbm)
postResample(pred = pred_gbm, obs = as.factor(testSet[,"target2"]))
predProb_gbm <- predict(object = gbmModel, testSet[,2:13], type = 'prob')
head(predProb_gbm)
postResample(pred = predProb_gbm, obs = testSet[,"target"])

library(pROC)
auc_gbm <- roc(ifelse(testSet[,"target2"]=="above",1,0), predProb_gbm[[2]])
print(auc_gbm$auc)
pred_glmnet <-predict(object = regModel, testSet[,2:13])
head(pred_glmnet)
library(pROC)
auc_glmnet <- roc(testSet[,"target"], pred_glmnet)
print(auc_glmnet$auc)
plot(varImp(regModel, scale = F))
```