DATA 621 - Homework3 - Crime Rate Prediction

Soumya Ghosh

March 28, 2020

Libraries

```
library(kableExtra)
library(tidyverse)
library(ggplot2)
library(dplyr)
library(MASS)
library(corrplot)
library(RColorBrewer)
library(GGally)
library(ggResidpanel)
library(psych)
library(mice)
library(reshape2)
library(cowplot)
library(car)
library(caTools)
library(VIM)
library(broom)
library(pROC)
library(caret)
library(geoR)
library(moments)
library(glmulti)
```

Introduction

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 training data comprises 466 observations and 13 variables.

Objective

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. 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 data set:

Data Load

Loaded Training and Evalutaion data sets into respective data frames.

```
train_df <- read.csv("https://raw.githubusercontent.com/soumya2g/CUNYDataMiningHomeWork/master/HomeWork_

eval_df <- read.csv("https://raw.githubusercontent.com/soumya2g/CUNYDataMiningHomeWork/master/HomeWork_3
```

Training Data

Sample snapshot of training data frame -

```
head(train_df, 20) %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed",
```

zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	lstat	medv	target
0	19.58	0	0.605	7.929	96.2	2.0459	5	403	14.7	3.70	50.0	1
0	19.58	1	0.871	5.403	100.0	1.3216	5	403	14.7	26.82	13.4	1
0	18.10	0	0.740	6.485	100.0	1.9784	24	666	20.2	18.85	15.4	1
30	4.93	0	0.428	6.393	7.8	7.0355	6	300	16.6	5.19	23.7	0
0	2.46	0	0.488	7.155	92.2	2.7006	3	193	17.8	4.82	37.9	0
0	8.56	0	0.520	6.781	71.3	2.8561	5	384	20.9	7.67	26.5	0
0	18.10	0	0.693	5.453	100.0	1.4896	24	666	20.2	30.59	5.0	1
0	18.10	0	0.693	4.519	100.0	1.6582	24	666	20.2	36.98	7.0	1
0	5.19	0	0.515	6.316	38.1	6.4584	5	224	20.2	5.68	22.2	0
80	3.64	0	0.392	5.876	19.1	9.2203	1	315	16.4	9.25	20.9	0
22	5.86	0	0.431	6.438	8.9	7.3967	7	330	19.1	3.59	24.8	0
0	12.83	0	0.437	6.286	45.0	4.5026	5	398	18.7	8.94	21.4	0
0	18.10	0	0.532	7.061	77.0	3.4106	24	666	20.2	7.01	25.0	1
22	5.86	0	0.431	8.259	8.4	8.9067	7	330	19.1	3.54	42.8	1
0	2.46	0	0.488	6.153	68.8	3.2797	3	193	17.8	13.15	29.6	0
0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	5.21	28.7	0
100	1.32	0	0.411	6.816	40.5	8.3248	5	256	15.1	3.95	31.6	0
20	3.97	0	0.647	5.560	62.8	1.9865	5	264	13.0	10.45	22.8	1
0	18.10	0	0.679	5.896	95.4	1.9096	24	666	20.2	24.39	8.3	1
0	18.10	0	0.671	6.545	99.1	1.5192	24	666	20.2	21.08	10.9	1

```
str(train_df)
```

```
## 'data.frame':    466 obs. of  13 variables:
##  $ zn     : num  0 0 0 30 0 0 0 0 0 80 ...
##  $ indus  : num  19.58 19.58 18.1 4.93 2.46 ...
##  $ chas   : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.605 0.871 0.74 0.428 0.488 0.52 0.693 0.693 0.515 0.392 ...
##  $ rm     : num  7.93 5.4 6.49 6.39 7.16 ...
##  $ age    : num  96.2 100 100 7.8 92.2 71.3 100 100 38.1 19.1 ...
##  $ dis    : num  2.05 1.32 1.98 7.04 2.7 ...
##  $ rad    : int  5 5 24 6 3 5 24 24 5 1 ...
##  $ tax    : int  403 403 666 300 193 384 666 666 224 315 ...
##  $ ptratio: num  14.7 14.7 20.2 16.6 17.8 20.9 20.2 20.2 20.2 16.4 ...
##  $ lstat  : num  3.7 26.82 18.85 5.19 4.82 ...
##  $ medv   : num  50 13.4 15.4 23.7 37.9 26.5 5 7 22.2 20.9 ...
##  $ target : int  1 1 1 0 0 0 1 1 0 0 ...
```

PART I: Data Exploration

We wanted to start off data exploration process with high level descriptive statistical summary and missing/exception value analysis.

Descriptive Statistical Summary

Basic statistical summary of all features and dependant variable (TARGET_WINS).

```
stat_summary <- function(df){
  df %>%
    summary() %>%
    kable() %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%
    scroll_box(width="100%",height="400px")
}
stat_summary(train_df)
```

	zn	indus	chas	nox	rm	age	dis
	Min. : 0.00	Min. : 0.460	Min. :0.00000	Min. :0.3890	Min. :3.863	Min. : 2.90	Min. : 1.130
	1st Qu.: 0.00	1st Qu.: 5.145	1st Qu.:0.00000	1st Qu.:0.4480	1st Qu.:5.887	1st Qu.: 43.88	1st Qu.: 2.1(
	Median : 0.00	Median : 9.690	Median :0.00000	Median :0.5380	Median :6.210	Median : 77.15	Median : 3.1
	Mean : 11.58	Mean :11.105	Mean :0.07082	Mean :0.5543	Mean :6.291	Mean : 68.37	Mean : 3.79(
	3rd Qu.: 16.25	3rd Qu.:18.100	3rd Qu.:0.00000	3rd Qu.:0.6240	3rd Qu.:6.630	3rd Qu.: 94.10	3rd Qu.: 5.2
	Max. :100.00	Max. :27.740	Max. :1.00000	Max. :0.8710	Max. :8.780	Max. :100.00	Max. :12.127

We also used describe() function of 'psych' package to summarize additional statistical measurements like Standard Deviation, Skewness, Kurtois, Standard Error etc.

```
stat_desc <- function(df){
df %>%
    describe() %>%
    kable() %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%  scroll_box
}

stat_desc(train_df)
```

	vars	n	mean	sd	median	trimmed	mad	min	max	ra
zn	1	466	11.5772532	23.3646511	0.00000	5.3542781	0.0000000	0.0000	100.0000	100.(
indus	2	466	11.1050215	6.8458549	9.69000	10.9082353	9.3403800	0.4600	27.7400	27.2
chas	3	466	0.0708155	0.2567920	0.00000	0.0000000	0.0000000	0.0000	1.0000	1.(
nox	4	466	0.5543105	0.1166667	0.53800	0.5442684	0.1334340	0.3890	0.8710	0.4
rm	5	466	6.2906738	0.7048513	6.21000	6.2570615	0.5166861	3.8630	8.7800	4.9
age	6	466	68.3675966	28.3213784	77.15000	70.9553476	30.0226500	2.9000	100.0000	97.1
dis	7	466	3.7956929	2.1069496	3.19095	3.5443647	1.9144814	1.1296	12.1265	10.9
rad	8	466	9.5300429	8.6859272	5.00000	8.6978610	1.4826000	1.0000	24.0000	23.(
tax	9	466	409.5021459	167.9000887	334.50000	401.5080214	104.5233000	187.0000	711.0000	524.(
ptratio	10	466	18.3984979	2.1968447	18.90000	18.5970588	1.9273800	12.6000	22.0000	9.4
lstat	11	466	12.6314592	7.1018907	11.35000	11.8809626	7.0720020	1.7300	37.9700	36.2
medv	12	466	22.5892704	9.2396814	21.20000	21.6304813	6.0045300	5.0000	50.0000	45.(
target	13	466	0.4914163	0.5004636	0.00000	0.4893048	0.0000000	0.0000	1.0000	1.(

Missing Value Analysis

Below are the features that has NA values. It is clear from the table below, TEAM_BATTING_HBP(>90%) and TEAM_BASERUN_CS (>33%) have significant NA Values -

```
## Counts of missing data per feature
train_na_df <- data.frame(apply(train_df, 2, function(x) length(which(is.na(x)))))
train_na_df1 <- data.frame(apply(train_df, 2,function(x) {sum(is.na(x)) / length(x) * 100}))

train_na_df <- cbind(Feature = rownames(train_na_df), train_na_df, train_na_df1)
colnames(train_na_df) <- c('Feature Name','No. of NA Recocrds','Percentage of NA Records')
rownames(train_na_df) <- NULL


train_na_df%>% filter(`No. of NA Recocrds` != 0) %>% arrange(desc(`No. of NA Recocrds`)) %>% kable() %>%
```

Feature Name	No. of NA Recocrds	Percentage of NA Records

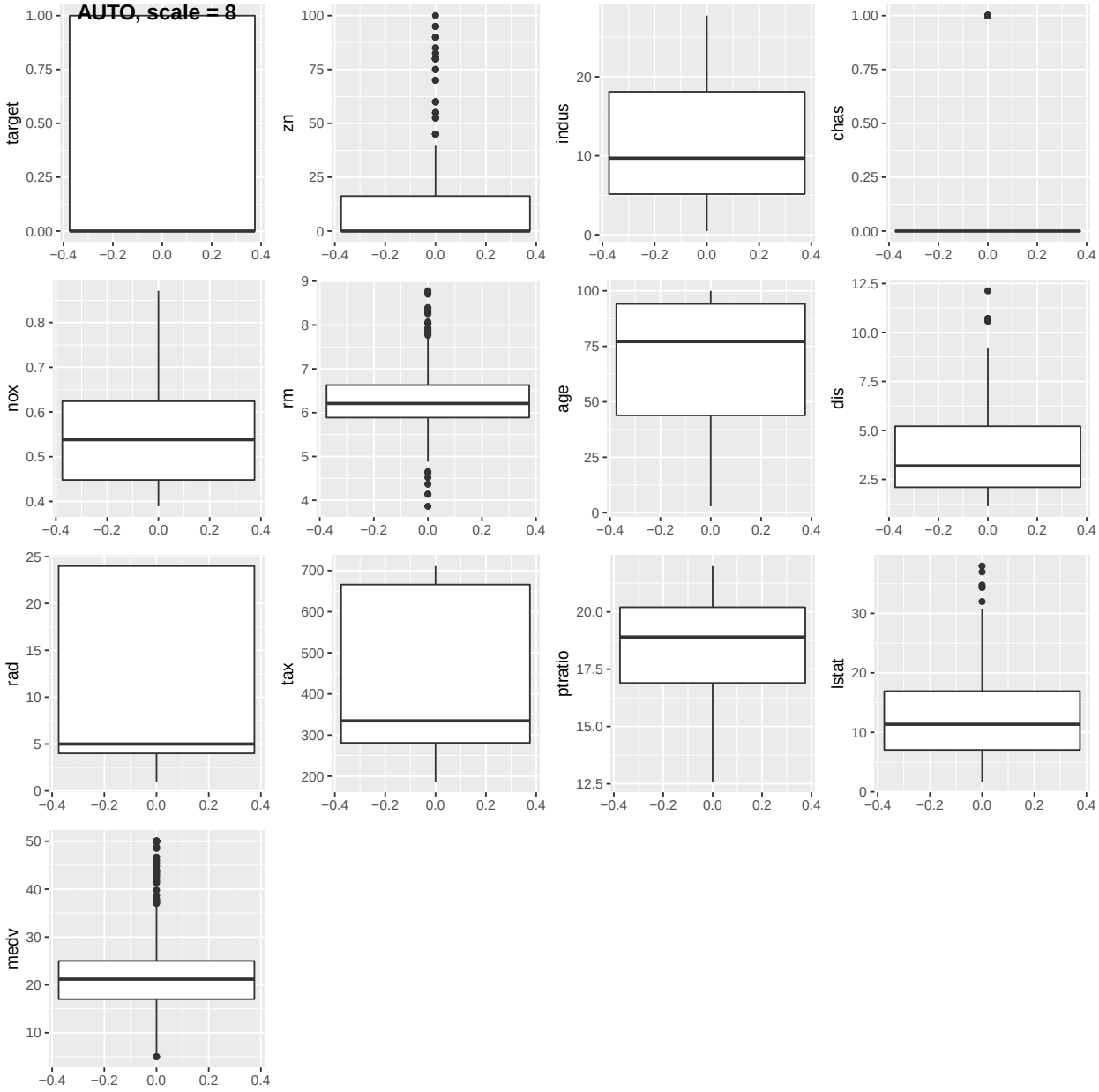
Based on the above empty table, there are no missing records. So we have a coplete data set to work with.

Descriptive Statistical Plots

Box Plots

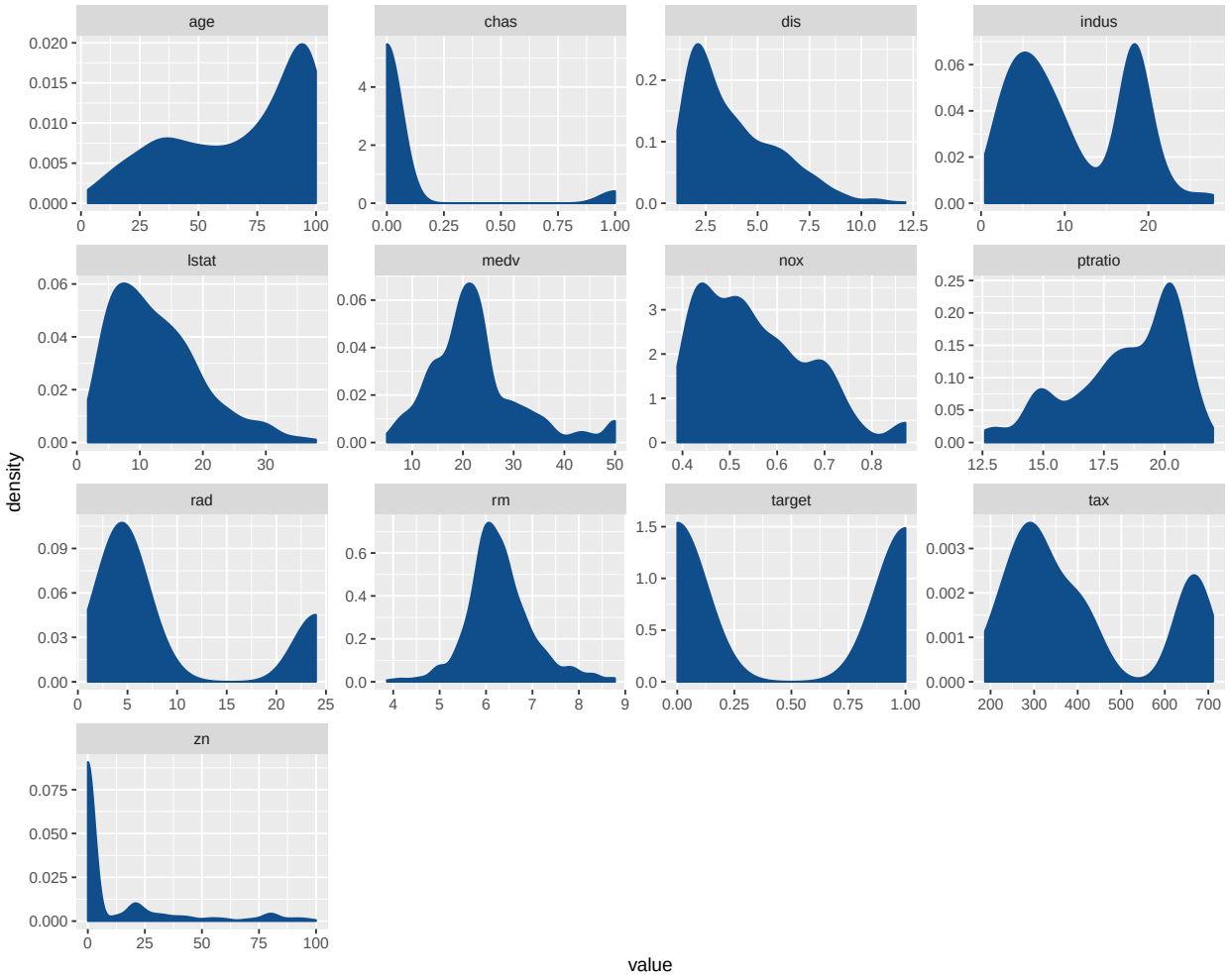
```
## Box plots:
gb1 <- ggplot(data = train_df, aes(y = target)) + geom_boxplot()
gb2 <- ggplot(data = train_df, aes(y = zn)) + geom_boxplot()
gb3 <- ggplot(data = train_df, aes(y = indus)) + geom_boxplot()
gb4 <- ggplot(data = train_df, aes(y = chas)) + geom_boxplot()
gb5 <- ggplot(data = train_df, aes(y = nox)) + geom_boxplot()
gb6 <- ggplot(data = train_df, aes(y = rm)) + geom_boxplot()
gb7 <- ggplot(data = train_df, aes(y = age)) + geom_boxplot()
gb8 <- ggplot(data = train_df, aes(y = dis)) + geom_boxplot()
gb9 <- ggplot(data = train_df, aes(y = rad)) + geom_boxplot()
gb10 <- ggplot(data = train_df, aes(y = tax)) + geom_boxplot()
gb11 <- ggplot(data = train_df, aes(y = ptratio)) + geom_boxplot()
gb12 <- ggplot(data = train_df, aes(y = lstat)) + geom_boxplot()
gb13 <- ggplot(data = train_df, aes(y = medv)) + geom_boxplot()


plot_grid(gb1, gb2, gb3, gb4, gb5, gb6, gb7, gb8, gb9, gb10,
          gb11, gb12, gb13, labels = "AUTO, scale = 8")
```



Density Plots

```
train_df %>%
  gather(variable, value, target:zn) %>%
  ggplot(., aes(value)) +
  geom_density(fill = "dodgerblue4", color="dodgerblue4") +
  facet_wrap(~variable, scales ="free", ncol = 4) +
  labs(x = element_blank(), y = element_blank())
```



Observations Summary

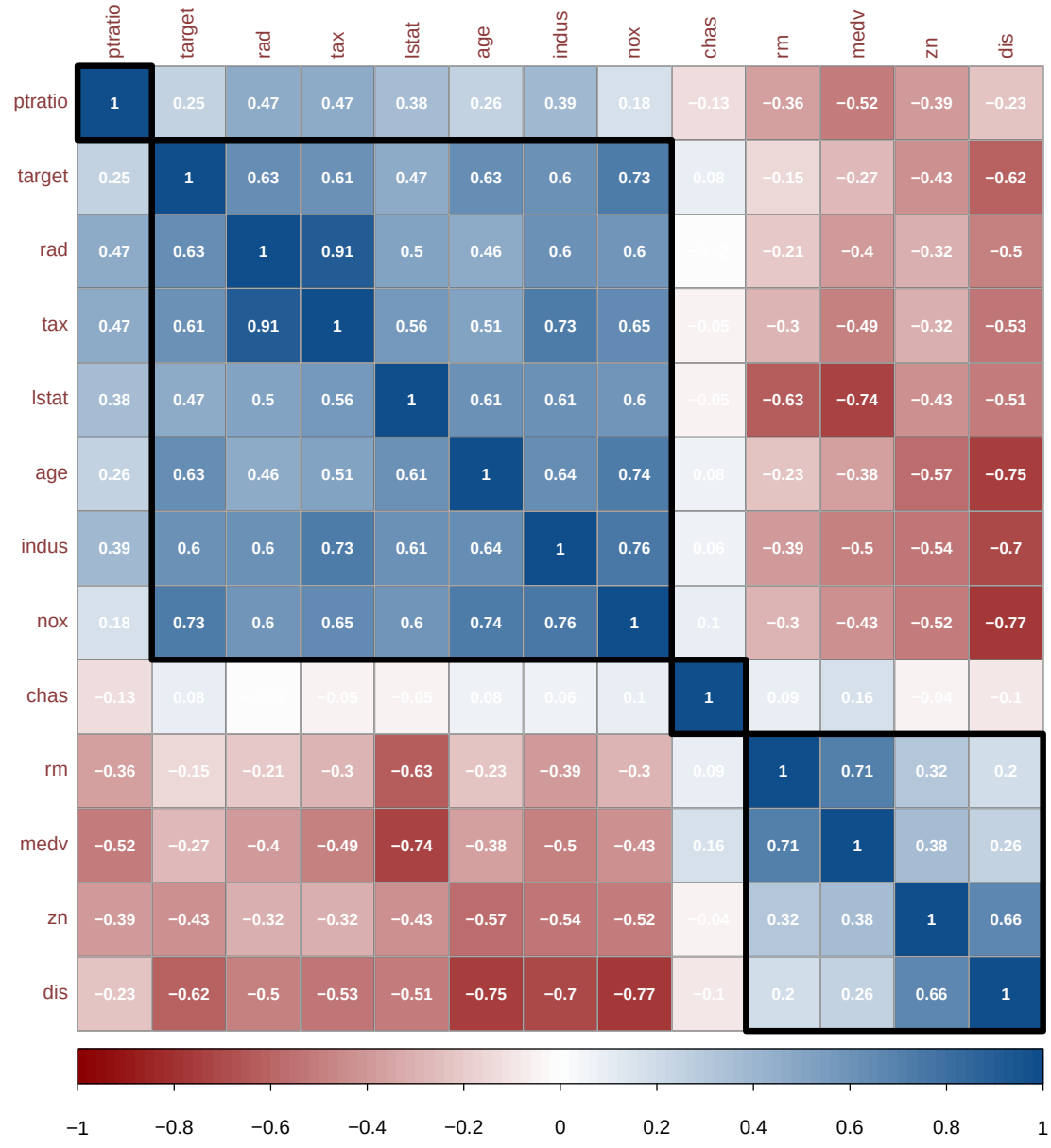
- Our dependant variable **target** is a binary variable as expected. There another bunary dummy variable **chas**
- Four of the predictors are expressed as proportions, with values ranging from 0 to 100:
- zn
- indus
- age
- lstat
- **indus, rad** and **tax** features are showing bi-modal distribution
- Some of the features like **dis, lstat, nox and zn** are right skewed
- **age** and **ptratio** are left skewed
- The variable rad is described as an index value of accessibility to radial highways. We assume this variable is an ordinal data type

PART II: Data Preparation

Correlation Plot

```
corrMatrix <- round(cor(train_df),4)

corrMatrix %>% corrplot(., method = "color", outline = T, addgrid.col = "darkgray", order="hclust", addr
```



Based on the Correlation plot above, there is a high degree of collinearity amogst independent variables like **rad, tax, indus, dis, nox** etc.

Handling multicollinearity

indus and nox: correlation value of 0.76. This result makes sense, as we expect areas with dense industry concentration to have higher environmental pollutants such as NO2.

dis and nox: correlation value of -0.77. This result is consistent with our intuition: we expect areas close to employment centers to have higher concentrations of environmental pollutants, and areas farther away to have lower concentrations.

rad and tax: These two variables are strongly correlated (91%). Access to radial highways and tax rates appear are strongly correlated values. We are particularly concerned about the multicollinearity effects of these two variables. The fact that these two variables are nearly perfectly correlated indicates that one of them can legitimately be removed from the data set, and we chose **rad** to be removed from our model for further analysis.

```
train_df <- train_df %>% dplyr::select(-c(rad))
eval_df <- eval_df %>% dplyr::select(-c(rad))
```

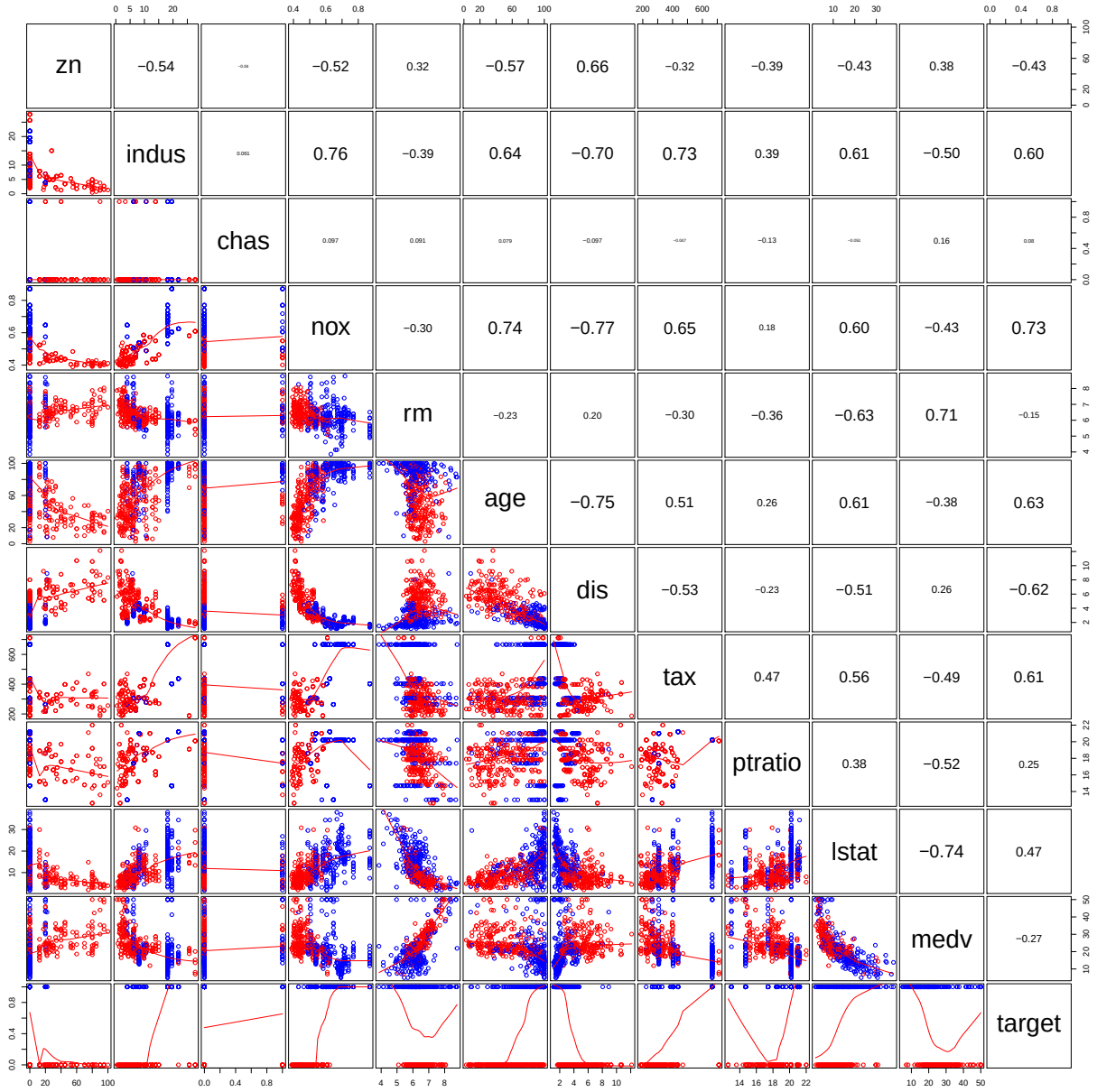
Pair-wise Relationships

Let's look at scatter plot/correlation matrix to get a better feel for the relationships between pairs of variables. In the figure below, we plotted the high crime areas in blue and the low crime areas in green. We also included a loess curve in the scatter plots to get a better feel for the relationship between variables.

```
cols = c('red','blue')
train_df$target_mod <- factor(ifelse(train_df$target == 1 , 'y', 'n'))

panel.cor <- function(x,y, digits = 2, cex.cor, ...) {
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- cor(x,y)
  txt <- format(c(r, 0.123456789), digits = digits)[1]
  text(0.5,0.5, txt, cex =sqrt(abs(r*6)))

}
pairs(train_df[,1:12],  col= cols[train_df$target_mod], gap=0.3, lower.panel = panel.smooth, upper.panel
```



Per the above par plot, there are some complex, non-linear LOESS curve shapes for some of the predictor variables against the response variable -

- **rm:** The LOESS curve initially indicates a negative relationship between the number of rooms and the crime rate category. However, when the number of rooms exceeds approximately 7, the relationship becomes positive. This strange LOESS curve shape is most likely the result of the model fitting to sparse data, as there are only about 30 observations with an average room count in excess of 7.
- **ptratio:** Normally we would expect crime rates to be higher in areas with high pupil-teacher ratios. However, the LOESS curve initially indicates an increasing propensity for high crime rates with increases to the this ptratio. Because this variable is left skewed with a high density of ratios clustered around 20, we believe this unusual curve shape is due to the LOESS model fitting to sparse data at low ratio values.
- **medv:** We expect high median home values to be associated with higher median values. This pattern appears to hold in our LOESS curve for median values below approximately $30k. The pattern then

reverses for values above $30k. Once again, we believe this pattern reversal is due to sparse data. The variable medv is right skewed, with relatively few data observations where the median value exceeds 30k

Feature Engineering

In binary logistic regression, it is desirable to have predictor variables that are normally distributed, whenever possible. The data in the crime dataset presents some factors that would lead us to have to perform some "Transformations" on the data. These transformation include adding categorical variables, log of variables, and adding power transformations etc.

age: This variable is left skewed. The level of skewness is as below -

```
skewness(train_df$age)
```

```
## [1] -0.5795721
```

We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$age)
```

```
## Fitted parameters:
##       lambda         beta      sigmasq
##     1.317655   205.697942 10492.780979
##
## Convergence code returned by optim: 0
```

We apply the suggested power transformation of 1.3 and store in a new variable, `age_new`.

dis: This variable is right skewed. The level of skewness is as below -

```
skewness(train_df$dis)
```

```
## [1] 1.002117
```

We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$dis)
```

```
## Fitted parameters:
##     lambda       beta    sigmasq
## -0.1467279  1.0719066  0.2051234
##
## Convergence code returned by optim: 0
```

Given that the value of the lambda parameter is fairly close to 0, we will use the log transformation and save to results to a new variable, `dis_new`.

indus: This variable has a bimodal distribution with values clustered around two ranges. Basic power transformation will not result in approximate normal distribution. Also, based on the pairwise plot, high crime rates are primarily concentrated in high industry areas. Hence, we decided to add a categorical variable indus_high_ind with 1 and 0 with a cut-off point of 14 which is approximately in the middle of two mode centers.

lstat: This variable is right skewed. The level of skewness is as below -

```
skewness(train_df$lstat)
```

```
## [1] 0.9085092
```

We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$lstat)
```

```
## Fitted parameters:
##    lambda      beta   sigmasq
## 0.2328042 3.2351269 1.0549878
##
## Convergence code returned by optim: 0
```

Based on this output, we create a new variable `lstat_new`, that applies a quarter root transformation to the original variable.

medv: This variable is right skewed. The level of skewness is as below -

```
skewness(train_df$medv)
```

```
## [1] 1.080167
```

We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$medv)
```

```
## Fitted parameters:
##    lambda      beta   sigmasq
## 0.2348612 4.4693904 0.6926584
##
## Convergence code returned by optim: 0
```

Based on this output, we create a new variable `medv_new`, that applies a quarter root transformation to the original variable.

nox: This variable is moderately right skewed. We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$nox)
```

```
## Fitted parameters:
##     lambda       beta    sigmasq
## -0.9494657 -0.8622020  0.1261795
##
## Convergence code returned by optim: 0
```

Based on the box-cox procedure output, We will create a new variable `nox_inv`, that is the reciprocal of the raw nox value. We then multiply the reciprocal by -1 to preserve the direction of the original relationship.

ptratio: This variable is moderately left skewed. The level of skewness is as below -

```
skewness(train_df$ptratio)
```

```
## [1] -0.7567025
```

We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$ptratio)
```

```
## Fitted parameters:
##       lambda         beta      sigmasq
## 4.144091e+00 4.574381e+04 3.608320e+08
##
## Convergence code returned by optim: 0
```

The suggested power transformation of 4 does not correct the the left skew, and its implementation also creates unusually large, transformed ptratio values(e.g. 200,000 and higher). Therefore, We will forgo the

power transformation for the sake of simplicity.

rm: This variable is moderately right skewed. The level of skewness is as below -

```
skewness(train_df$rm)
```

```
## [1] 0.4808673
```

We perform the box-cox procedure to determine an appropriate transformation:

```
boxcoxfit(train_df$rm)
```

```
## Fitted parameters:
##     lambda       beta    sigmasq
## 0.20380031 2.22393623 0.02626356
##
## Convergence code returned by optim: 0
```

Based on this output, we create a new variable rm_new‘, that applies a quarter root transformation to the original variable.

tax: The variable tax also has a bi-modal shape, with values densely clustered around 300 and 700 with no values recording in the training data between between 470 and 665. Because power transformations have limited effectiveness in approximating a normal distribution, we'll create a new categorical variable, `tax_high_ind`, that assigns a value of 1 when the tax value is greater than or equal to 500, and 0 otherwise. The 500 cutoff reflects an approximate halfway point between the two modal centers.

```
feature_engineering <- function(df){

  df$age_new  <- df$age^1.3
  df$dis_new <- log(df$dis)
  df$indus_high_ind <- ifelse(df$indus <= 14, 0, 1)
  df$lstat_new <- df$lstat^0.25
  df$medv_new <- df$medv^0.25
  df$nox_inv <- -1/df$nox
  df$rm_new <- df$rm^0.25
  df$tax_high_ind <- ifelse(df$tax <= 500, 0, 1)

  return(df)
}

train_df <- feature_engineering(train_df)
eval_df <- feature_engineering(eval_df)
```

Data Preparation Results

After making feature removal and additional feature engineering steps, below is the summary of all available features -

```
stat_desc(train_df)
```

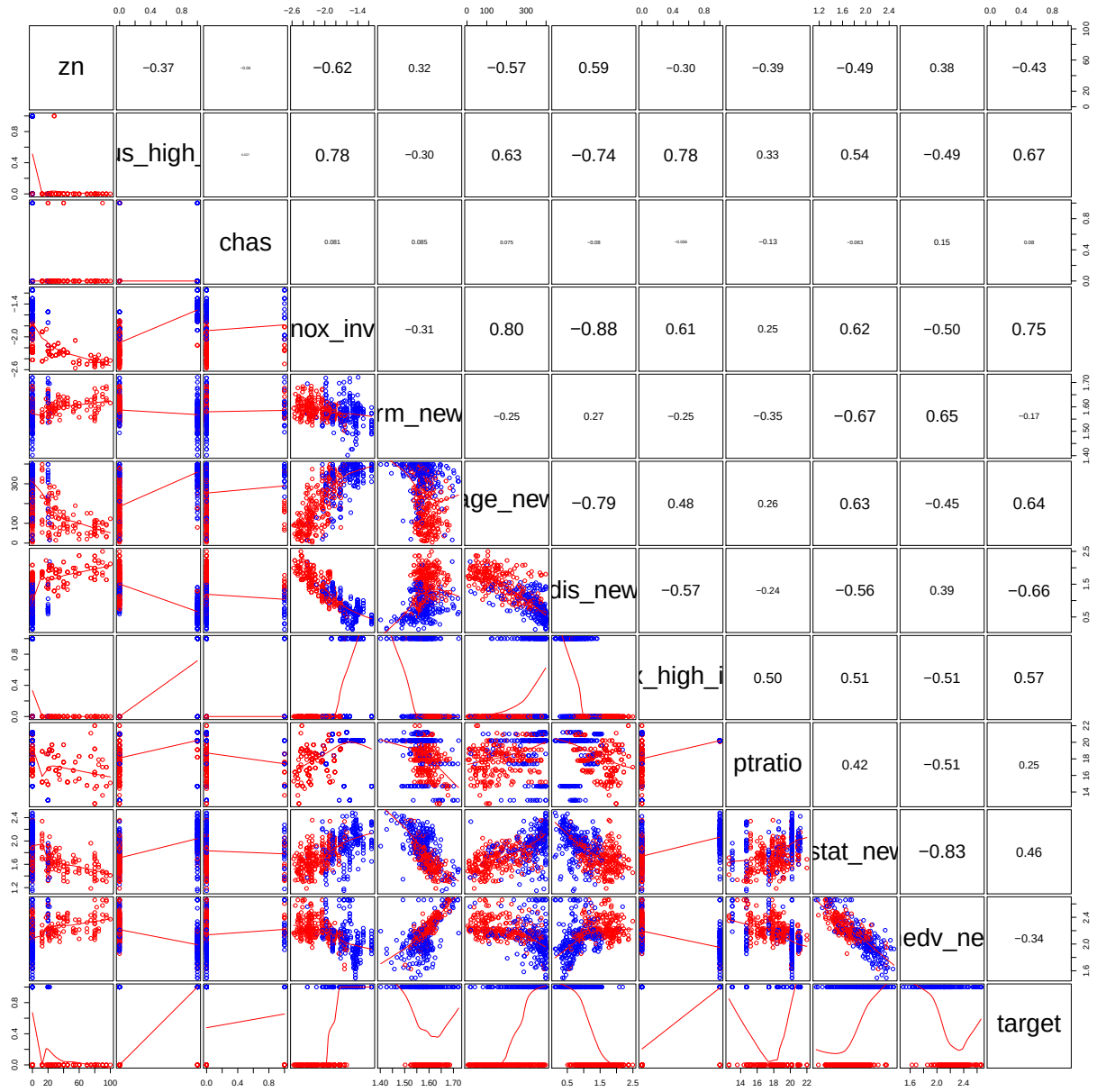
	vars	n	mean	sd	median	trimmed	mad	min	
zn	1	466	11.5772532	23.3646511	0.000000	5.3542781	0.0000000	0.0000000	10
indus	2	466	11.1050215	6.8458549	9.690000	10.9082353	9.3403800	0.4600000	2
chas	3	466	0.0708155	0.2567920	0.000000	0.0000000	0.0000000	0.0000000	
nox	4	466	0.5543105	0.1166667	0.538000	0.5442684	0.1334340	0.3890000	
rm	5	466	6.2906738	0.7048513	6.210000	6.2570615	0.5166861	3.8630000	
age	6	466	68.3675966	28.3213784	77.150000	70.9553476	30.0226500	2.9000000	10
dis	7	466	3.7956929	2.1069496	3.190950	3.5443647	1.9144814	1.1296000	1
tax	8	466	409.5021459	167.9000887	334.500000	401.5080214	104.5233000	187.0000000	71
ptratio	9	466	18.3984979	2.1968447	18.900000	18.5970588	1.9273800	12.6000000	2
lstat	10	466	12.6314592	7.1018907	11.350000	11.8809626	7.0720020	1.7300000	3
medv	11	466	22.5892704	9.2396814	21.200000	21.6304813	6.0045300	5.0000000	5
target	12	466	0.4914163	0.5004636	0.000000	0.4893048	0.0000000	0.0000000	
target_mod*	13	466	1.4914163	0.5004636	1.000000	1.4893048	0.0000000	1.0000000	
age_new	14	466	251.8570242	124.0342831	284.143045	260.3295138	147.1891999	3.9913278	39
dis_new	15	466	1.1875848	0.5412299	1.160315	1.1782986	0.6732974	0.1218636	
indus_high_ind	16	466	0.3776824	0.4853286	0.000000	0.3475936	0.0000000	0.0000000	
lstat_new	17	466	1.8287387	0.2677688	1.835476	1.8277552	0.2935691	1.1468630	
medv_new	18	466	2.1474072	0.2180936	2.145774	2.1452873	0.1620255	1.4953488	
nox_inv	19	466	-1.8792908	0.3667673	-1.858736	-1.8812132	0.4642629	-2.5706941	-
rm_new	20	466	1.5818618	0.0441441	1.578603	1.5810231	0.0327835	1.4019456	
tax_high_ind	21	466	0.2703863	0.4446367	0.000000	0.2139037	0.0000000	0.0000000	

```
str(train_df)
```

```
## 'data.frame':    466 obs. of  21 variables:
##  $ zn             : num  0 0 0 30 0 0 0 0 0 80 ...
##  $ indus          : num  19.58 19.58 18.1 4.93 2.46 ...
##  $ chas           : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ nox            : num  0.605 0.871 0.74 0.428 0.488 0.52 0.693 0.693 0.515 0.392 ...
##  $ rm             : num  7.93 5.4 6.49 6.39 7.16 ...
##  $ age            : num  96.2 100 100 7.8 92.2 71.3 100 100 38.1 19.1 ...
##  $ dis            : num  2.05 1.32 1.98 7.04 2.7 ...
##  $ tax            : int  403 403 666 300 193 384 666 666 224 315 ...
##  $ ptratio        : num  14.7 14.7 20.2 16.6 17.8 20.9 20.2 20.2 20.2 16.4 ...
##  $ lstat          : num  3.7 26.82 18.85 5.19 4.82 ...
##  $ medv           : num  50 13.4 15.4 23.7 37.9 26.5 5 7 22.2 20.9 ...
##  $ target         : int  1 1 1 0 0 0 1 1 0 0 ...
##  $ target_mod     : Factor w/ 2 levels "n","y": 2 2 2 1 1 1 2 2 1 1 ...
##  $ age_new        : num  378.6 398.1 398.1 14.4 358.2 ...
##  $ dis_new        : num  0.716 0.279 0.682 1.951 0.993 ...
##  $ indus_high_ind: num  1 1 1 0 0 0 1 1 0 0 ...
##  $ lstat_new      : num  1.39 2.28 2.08 1.51 1.48 ...
##  $ medv_new       : num  2.66 1.91 1.98 2.21 2.48 ...
##  $ nox_inv        : num  -1.65 -1.15 -1.35 -2.34 -2.05 ...
##  $ rm_new         : num  1.68 1.52 1.6 1.59 1.64 ...
##  $ tax_high_ind   : num  0 0 1 0 0 0 1 1 0 0 ...
```

```
cols = c('red','blue')

pairs(train_df[,c(1, 16, 3, 19:20, 14:15, 21, 9, 17:18, 12)],  col= cols[train_df$target_mod], gap=0.3,
```



Based on above plot, below feature pairs are showing high degree of colinearity -

- `nox_inv` and `age_new`: 0.80
- `nox_inv` and `dis_new`: -0.88
- `lstat_new` and `medv_new`: -0.83

Multicollinearity can assessed by computing a score called the Variance Inflation Factor (or VIF), which measures how much the variance of a regression coefficient is inflated due to multicollinearity in the model. So let's review VIFs -

```
lm_model <- lm(target ~ . ,data=train_df[,c(1, 16, 3, 19:20, 14:15, 21, 9, 17:18, 12)])
vif(lm_model)
```

```
##              zn indus_high_ind            chas        nox_inv          rm_new
```

```
##       2.154960       4.531029       1.067921       7.022698       2.203387
##        age_new        dis_new   tax_high_ind        ptratio      lstat_new
##       3.685974       5.444450       3.325173       1.871051       5.199120
##       medv_new
##       4.155697
```

Due to VIF scores above 5, `nox_inv`, dis_new and `lstat_new` may not be used together in a model.

```
train_df <- train_df %>% dplyr::select(-c(target_mod))
```

PART III: Building Models

Test train approach

We have divided the traning dataset into training and test sets using a 80/20 split. We will build our models on the training set and evaluate it on the test set.

```
set.seed(123)
split <- sample.split(train_df$target, SplitRatio = 0.8)
training_set <- subset(train_df, split == TRUE)
test_set <- subset(train_df, split == FALSE)
```

Below is the list of complete list of variables before we embark on Model building process -

```
str(training_set)
```

```
## 'data.frame':    373 obs. of  20 variables:
##  $ zn            : num  0 0 0 0 0 80 22 0 0 22 ...
##  $ indus         : num  19.58 18.1 2.46 8.56 5.19 ...
##  $ chas          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox           : num  0.605 0.74 0.488 0.52 0.515 0.392 0.431 0.437 0.532 0.431 ...
##  $ rm            : num  7.93 6.49 7.16 6.78 6.32 ...
##  $ age           : num  96.2 100 92.2 71.3 38.1 19.1 8.9 45 77 8.4 ...
##  $ dis           : num  2.05 1.98 2.7 2.86 6.46 ...
##  $ tax           : int  403 666 193 384 224 315 330 398 666 330 ...
##  $ ptratio       : num  14.7 20.2 17.8 20.9 20.2 16.4 19.1 18.7 20.2 19.1 ...
##  $ lstat         : num  3.7 18.85 4.82 7.67 5.68 ...
##  $ medv          : num  50 15.4 37.9 26.5 22.2 20.9 24.8 21.4 25 42.8 ...
##  $ target        : int  1 1 0 0 0 0 0 0 1 1 ...
##  $ age_new       : num  379 398 358 256 114 ...
##  $ dis_new       : num  0.716 0.682 0.993 1.049 1.865 ...
##  $ indus_high_ind: num  1 1 0 0 0 0 0 0 1 0 ...
##  $ lstat_new     : num  1.39 2.08 1.48 1.66 1.54 ...
##  $ medv_new      : num  2.66 1.98 2.48 2.27 2.17 ...
##  $ nox_inv       : num  -1.65 -1.35 -2.05 -1.92 -1.94 ...
##  $ rm_new        : num  1.68 1.6 1.64 1.61 1.59 ...
##  $ tax_high_ind  : num  0 1 0 0 0 0 0 0 1 0 ...
```

We will build four different models to see which one yields the best performance.

Model 1:

We will start off with a model with all the original variables excluding any derived features -

```r
baseline_logit_model <- glm(target~. - age_new - dis_new - indus_high_ind - lstat_new -medv_new - nox_i

summary(baseline_logit_model)
```

```
##
## Call:
## glm(formula = target ~ . - age_new - dis_new - indus_high_ind -
##     lstat_new - medv_new - nox_inv - rm_new - tax_high_ind, family = "binomial",
##     data = training_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2759  -0.2826  -0.0073   0.1604   3.4518
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -39.144899   6.514643  -6.009 1.87e-09 ***
## zn           -0.077304   0.035255  -2.193 0.028328 *
## indus        -0.143998   0.059105  -2.436 0.014838 *
## chas          2.386050   0.876089   2.724 0.006459 **
## nox          46.693161   7.930718   5.888 3.92e-09 ***
## rm           -0.173873   0.623150  -0.279 0.780228
## age           0.029778   0.013226   2.252 0.024353 *
## dis           0.874731   0.230132   3.801 0.000144 ***
## tax           0.007596   0.002334   3.255 0.001135 **
## ptratio       0.273426   0.119378   2.290 0.021997 *
## lstat        -0.006013   0.054019  -0.111 0.911365
## medv          0.154569   0.058928   2.623 0.008715 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 516.96  on 372  degrees of freedom
## Residual deviance: 186.58  on 361  degrees of freedom
## AIC: 210.58
##
## Number of Fisher Scoring iterations: 8
```

Model Statistics

```r
test_set$baseline_logit_model <- ifelse(predict.glm(baseline_logit_model, test_set,"response") >= 0.5,1
cm_model1 <- confusionMatrix(factor(test_set$baseline_logit_model), factor(test_set$target),"1")

cm_model1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 42  5
##          1  5 41
##
##                Accuracy : 0.8925
```

```
##                  95% CI : (0.8111, 0.9472)
##     No Information Rate : 0.5054
##     P-Value [Acc > NIR] : 2.021e-15
##
##                   Kappa : 0.7849
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8913
##             Specificity : 0.8936
##          Pos Pred Value : 0.8913
##          Neg Pred Value : 0.8936
##              Prevalence : 0.4946
##          Detection Rate : 0.4409
##    Detection Prevalence : 0.4946
##       Balanced Accuracy : 0.8925
##
##        'Positive' Class : 1
##
```

VIF Results

Multicollinearity can assessed by computing a score called the Variance Inflation Factor (or VIF), which measures how much the variance of a regression coefficient is inflated due to multicollinearity in the model.

```
vif(baseline_logit_model)
```

```
##       zn    indus     chas      nox       rm      age      dis      tax
## 1.756565 3.719601 1.316733 4.529997 4.593193 2.481361 4.186295 2.707443
##  ptratio    lstat     medv
## 1.744104 3.005254 6.165711
```

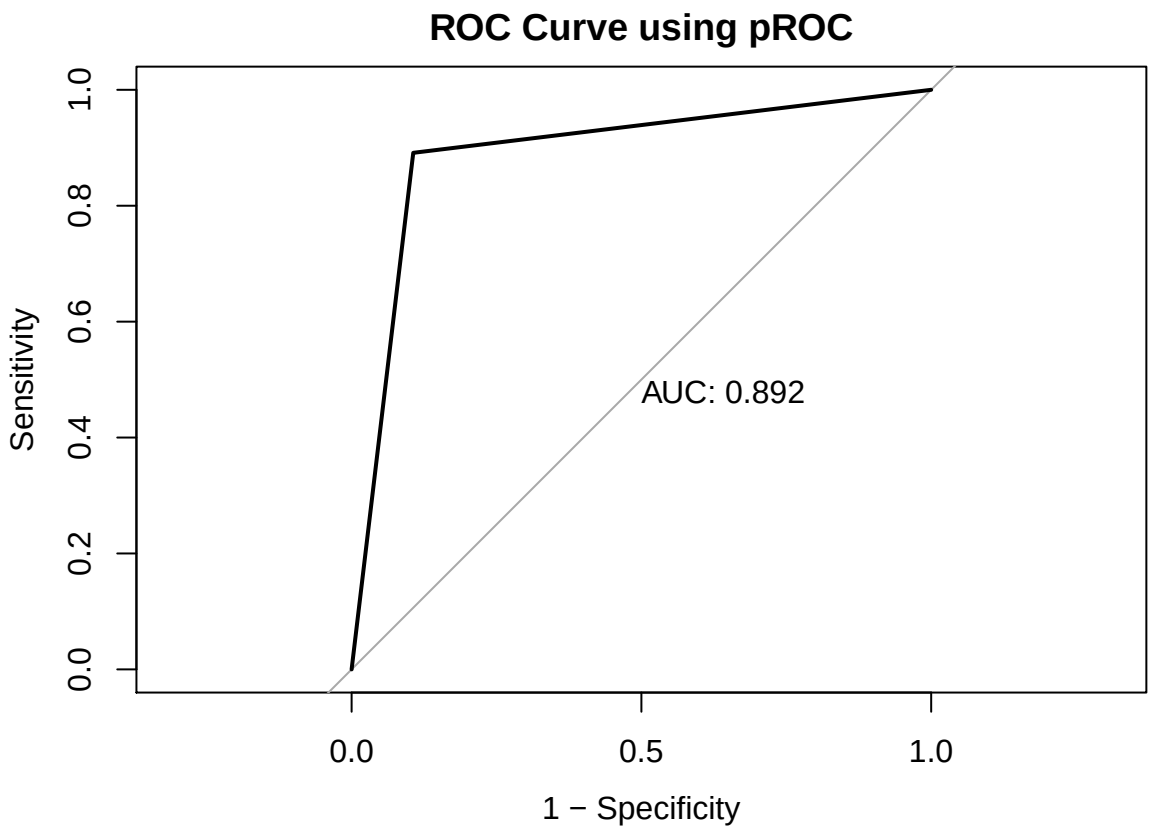
AUC Curve

```
rocCurve <- roc(response = test_set$target,
                predictor = test_set$baseline_logit_model)
```

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

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

```
## Plot the ROC Curve:
plot(rocCurve, legacy.axes = TRUE, main = "ROC Curve using pROC",print.auc = TRUE)
```




```
# Model Stats
results <- tibble(model = "baseline_logit_model",predictors = 11,F1 = cm_model1$byClass[7],
                  deviance=baseline_logit_model$deviance,
                  r2 = 1 - baseline_logit_model$deviance/baseline_logit_model$null.deviance,
                  aic=baseline_logit_model$aic, auc = auc(rocCurve), AICc = aicc(baseline_logit_model))

model <- as.data.frame(glance(baseline_logit_model))

results <- cbind(results, model)
```

Model 2:

In the 2nd model, we have added all the variables including the derived features.

```
full_logit_model <- glm(target~. ,family="binomial",data=training_set)

summary(full_logit_model)
```

```
##
## Call:
## glm(formula = target ~ ., family = "binomial", data = training_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0476  -0.1904  -0.0061   0.0805   3.4507
##
```

```
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     5.453e+02  1.787e+02   3.051 0.002280 **
## zn             -4.620e-02  3.587e-02  -1.288 0.197825
## indus          -5.106e-01  1.109e-01  -4.605 4.13e-06 ***
## chas            2.789e+00  1.003e+00   2.780 0.005439 **
## nox             8.432e+01  5.322e+01   1.584 0.113081
## rm              3.349e+01  9.311e+00   3.597 0.000322 ***
## age            -3.548e-01  1.630e-01  -2.177 0.029512 *
## dis            -2.752e+00  1.010e+00  -2.724 0.006449 **
## tax             9.898e-03  4.783e-03   2.069 0.038518 *
## ptratio         4.591e-01  1.644e-01   2.792 0.005231 **
## lstat          -3.286e-01  2.259e-01  -1.454 0.145818
## medv           -1.640e-01  2.168e-01  -0.757 0.449258
## age_new         8.968e-02  3.735e-02   2.401 0.016366 *
## dis_new         1.567e+01  4.306e+00   3.639 0.000274 ***
## indus_high_ind  7.697e+00  1.775e+00   4.337 1.45e-05 ***
## lstat_new       7.809e+00  5.994e+00   1.303 0.192646
## medv_new        1.168e+01  9.023e+00   1.295 0.195401
## nox_inv        -7.018e+00  1.455e+01  -0.482 0.629571
## rm_new         -5.446e+02  1.485e+02  -3.667 0.000245 ***
## tax_high_ind   -1.793e+00  1.808e+00  -0.992 0.321409
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 516.96  on 372  degrees of freedom
## Residual deviance: 139.90  on 353  degrees of freedom
## AIC: 179.9
##
## Number of Fisher Scoring iterations: 8
```

Model Statistics

```
test_set$full_logit_model <- ifelse(predict.glm(full_logit_model, test_set,"response") >= 0.5,1,0)
cm_model2 <- confusionMatrix(factor(test_set$full_logit_model), factor(test_set$target),"1")

cm_model2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 43  6
##          1  4 40
##
##                Accuracy : 0.8925
##                  95% CI : (0.8111, 0.9472)
##     No Information Rate : 0.5054
##     P-Value [Acc > NIR] : 2.021e-15
##
##                   Kappa : 0.7848
##
```

```
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.8696
##             Specificity : 0.9149
##          Pos Pred Value : 0.9091
##          Neg Pred Value : 0.8776
##              Prevalence : 0.4946
##          Detection Rate : 0.4301
##    Detection Prevalence : 0.4731
##       Balanced Accuracy : 0.8922
##
##        'Positive' Class : 1
##
```

VIF Results

Multicollinearity can assessed by computing a score called the Variance Inflation Factor (or VIF), which measures how much the variance of a regression coefficient is inflated due to multicollinearity in the model.

```
vif(full_logit_model)
```

```
##             zn          indus           chas            nox             rm
##       1.610938       9.477768       1.504821     152.077801     811.155752
##            age            dis            tax        ptratio          lstat
##     324.363346      47.101697       8.075497       2.036233      43.445253
##           medv        age_new        dis_new indus_high_ind      lstat_new
##      69.324325     331.738615      64.150678       9.434319      45.841001
##       medv_new        nox_inv         rm_new   tax_high_ind
##      63.495066     139.102795     769.268700       6.423736
```

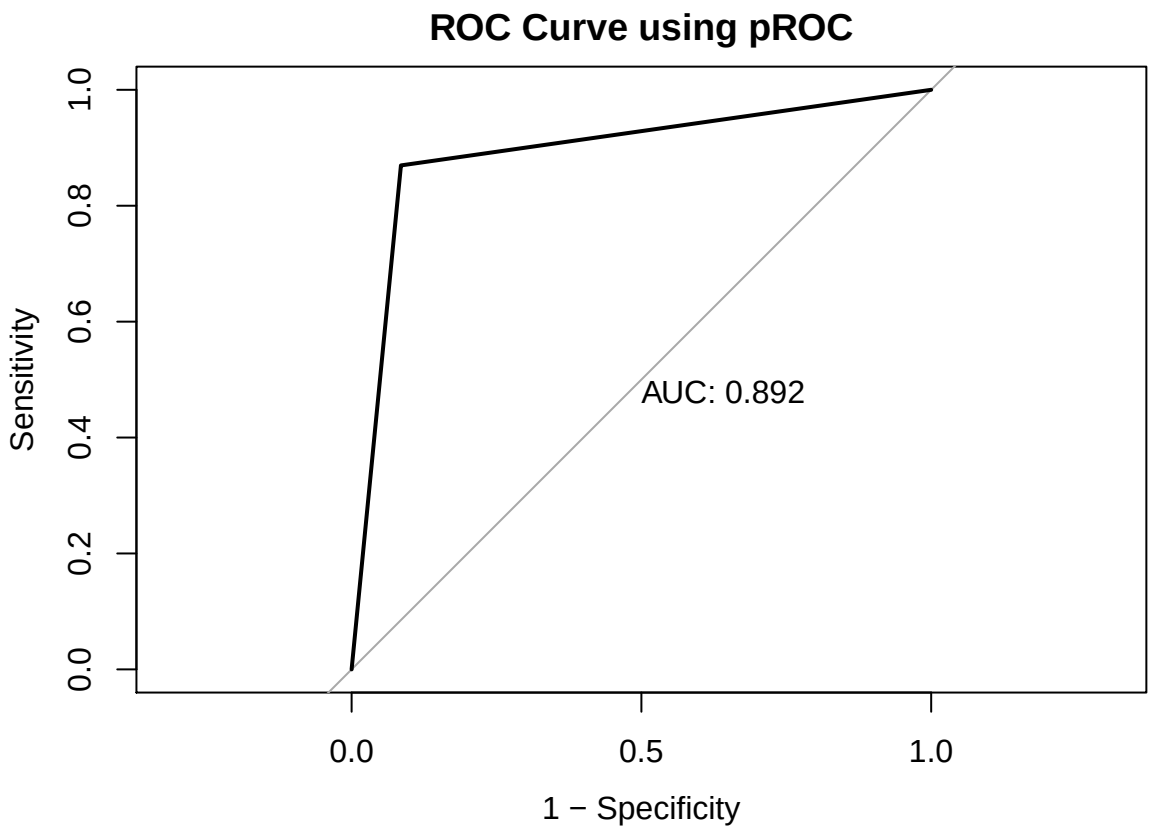
AUC Curve

```
rocCurve <- roc(response = test_set$target,
                predictor = test_set$full_logit_model)
```

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

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

```
## Plot the ROC Curve:
plot(rocCurve, legacy.axes = TRUE, main = "ROC Curve using pROC",print.auc = TRUE)
```




```
# Model Stats
results_2 <- tibble(model = "full_logit_model",predictors = 19,F1 = cm_model2$byClass[7],
                    deviance=full_logit_model$deviance,
                    r2 = 1 - full_logit_model$deviance/full_logit_model$null.deviance,
                    aic = full_logit_model$aic, auc = auc(rocCurve), AICc = aicc(full_logit_model))

model_2 <- as.data.frame(glance(full_logit_model))

results_2 <- cbind(results_2, model_2)

results <- rbind(results,results_2)
```

Model Summary

In this model F1 score and AUC is lower than the first model and also many variables are insignificant.

Model 3:

In the 3rd model, we have kept the significant variables only from the previous model.

```
significant_logit_model <- glm(target ~ chas + nox + tax + ptratio + dis_new + indus_high_ind + rm_new,

summary(significant_logit_model)
```

```
##
## Call:
```

```
## glm(formula = target ~ chas + nox + tax + ptratio + dis_new +
##     indus_high_ind + rm_new, family = "binomial", data = training_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5398  -0.3616  -0.1202   0.2698   2.6743
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -53.187062  10.652443  -4.993 5.95e-07 ***
## chas             2.077610   0.746562   2.783  0.00539 **
## nox             41.082799   6.191022   6.636 3.23e-11 ***
## tax              0.003717   0.001940   1.916  0.05542 .
## ptratio          0.195953   0.095617   2.049  0.04043 *
## dis_new          2.286927   0.738955   3.095  0.00197 **
## indus_high_ind   0.241489   0.638261   0.378  0.70517
## rm_new          14.592401   4.896342   2.980  0.00288 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 516.96  on 372  degrees of freedom
## Residual deviance: 206.96  on 365  degrees of freedom
## AIC: 222.96
##
## Number of Fisher Scoring iterations: 7
```

Model Statistics

```
test_set$significant_logit_model <- ifelse(predict.glm(significant_logit_model, test_set,"response") >=
cm_model3 <- confusionMatrix(factor(test_set$significant_logit_model), factor(test_set$target),"1")

cm_model3
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 41  7
##          1  6 39
##
##                Accuracy : 0.8602
##                  95% CI : (0.7728, 0.9234)
##     No Information Rate : 0.5054
##     P-Value [Acc > NIR] : 6.388e-13
##
##                   Kappa : 0.7203
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8478
##             Specificity : 0.8723
##          Pos Pred Value : 0.8667
```

```
##          Neg Pred Value : 0.8542
##              Prevalence : 0.4946
##          Detection Rate : 0.4194
##    Detection Prevalence : 0.4839
##       Balanced Accuracy : 0.8601
##
##        'Positive' Class : 1
##
```

VIF Results

Multicollinearity can assessed by computing a score called the Variance Inflation Factor (or VIF), which measures how much the variance of a regression coefficient is inflated due to multicollinearity in the model.

```
vif(significant_logit_model)
```

```
##           chas            nox            tax        ptratio        dis_new
##       1.157036       3.773906       1.942803       1.282896       3.559120
## indus_high_ind         rm_new
##       2.166474       1.483406
```

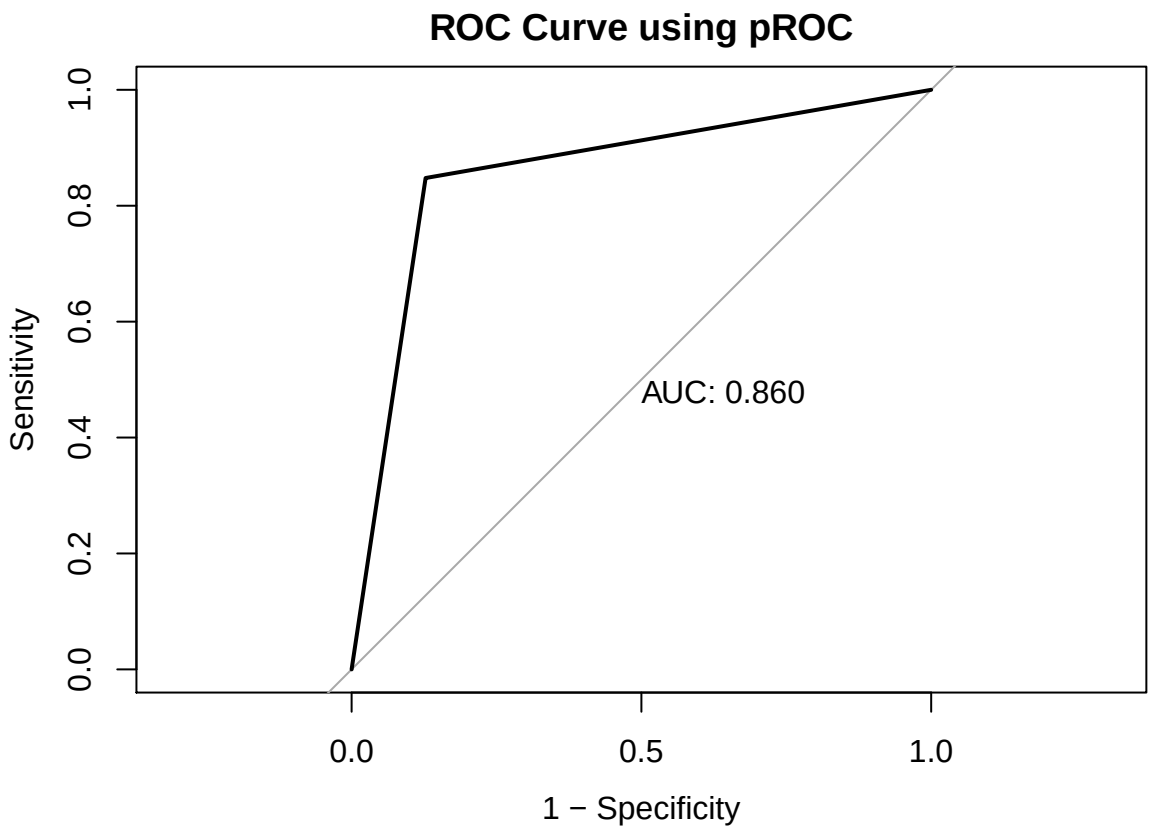
AUC Curve

```
rocCurve <- roc(response = test_set$target,
                predictor = test_set$significant_logit_model)
```

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

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

```
## Plot the ROC Curve:
plot(rocCurve, legacy.axes = TRUE, main = "ROC Curve using pROC",print.auc = TRUE)
```




```
# Model Stats
results_3 <- tibble(model = "significant_logit_model",predictors = 7,F1 = cm_model3$byClass[7],
                    deviance=significant_logit_model$deviance,
                    r2 = 1 - significant_logit_model$deviance/significant_logit_model$null.deviance,
                    aic = significant_logit_model$aic, auc = auc(rocCurve), AICc = aicc(significant_logit_

model_3 <- as.data.frame(glance(significant_logit_model))

results_3 <- cbind(results_3, model_3)

results <- rbind(results,results_3)
```

Model Summary

In this model AIC is higher than the other two models. But the F1 and AUC scores are lower than the first and second model.

Model 4: Automatic Selection Stepwise Model

For our last model, we'll fit a binary logistic regression using a stepwise regression procedure, with variable selection occurring in both forward and backward directions.

For simplicity, we'll only include first order terms, but we'll open up the pool of candidate variables to all variables in our data set-using transformed versions of our variables, where applicable. There is one exception though: we exclude rad_high due to its extremely high correlation with tax_high.

```
model.upper <- glm(target ~ zn + indus_high_ind + chas + nox_inv + rm_new + age_new + dis_new + tax_high
model.null = glm(target ~ 1,
                 data=training_set,
                 family = "binomial")
stepwise_model <- step(model.null, scope = list(upper=model.upper, lower = model.null),
                 trace = 0, direction = 'both')
summary(stepwise_model)
```

```
##
## Call:
## glm(formula = target ~ nox_inv + tax_high_ind + dis_new + medv_new +
##     age_new + ptratio + chas, family = "binomial", data = training_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8158  -0.2621  -0.0256   0.2758   2.8889
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.851370   4.762656  -0.599 0.549377
## nox_inv      12.670515   1.912865   6.624 3.50e-11 ***
## tax_high_ind  2.475549   0.738061   3.354 0.000796 ***
## dis_new       3.384582   0.838093   4.038 5.38e-05 ***
## medv_new      6.562114   1.624009   4.041 5.33e-05 ***
## age_new       0.007125   0.002482   2.871 0.004096 **
## ptratio       0.321442   0.113148   2.841 0.004499 **
## chas          1.705403   0.758426   2.249 0.024537 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 516.96  on 372  degrees of freedom
## Residual deviance: 194.22  on 365  degrees of freedom
## AIC: 210.22
##
## Number of Fisher Scoring iterations: 7
```

Model Statistics

```
test_set$stepwise_model <- ifelse(predict.glm(stepwise_model, test_set,"response") >= 0.5,1,0)
cm_model4 <- confusionMatrix(factor(test_set$stepwise_model), factor(test_set$target),"1")

cm_model4
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 42  5
##          1  5 41
##
##                Accuracy : 0.8925
##                  95% CI : (0.8111, 0.9472)
```

```
##     No Information Rate : 0.5054
##     P-Value [Acc > NIR] : 2.021e-15
##
##                   Kappa : 0.7849
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8913
##             Specificity : 0.8936
##          Pos Pred Value : 0.8913
##          Neg Pred Value : 0.8936
##              Prevalence : 0.4946
##          Detection Rate : 0.4409
##    Detection Prevalence : 0.4946
##       Balanced Accuracy : 0.8925
##
##        'Positive' Class : 1
##
```

VIF Results

Multicollinearity can assessed by computing a score called the Variance Inflation Factor (or VIF), which measures how much the variance of a regression coefficient is inflated due to multicollinearity in the model.

```
vif(stepwise_model)
```

```
##      nox_inv tax_high_ind      dis_new     medv_new      age_new
##     4.067633     1.598767     3.795100     2.938636     1.825200
##      ptratio         chas
##     1.688802     1.130040
```

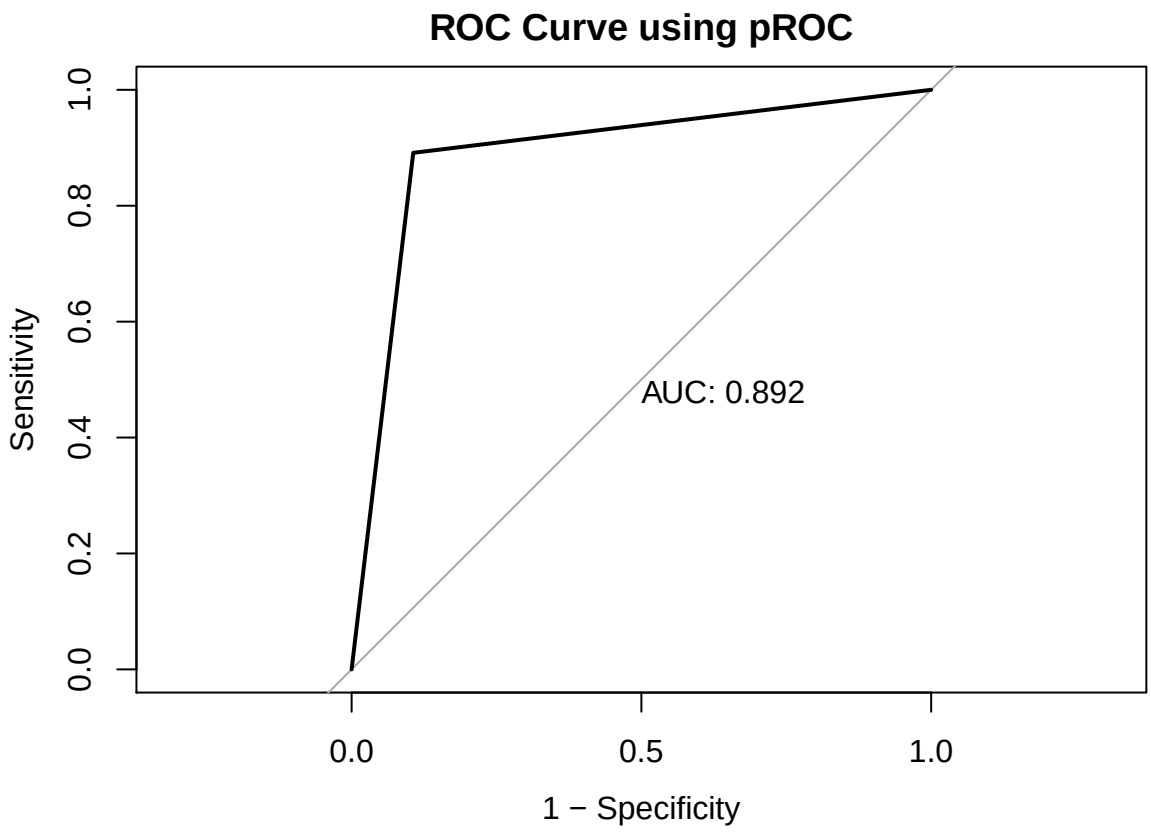
AUC Curve

```
rocCurve <- roc(response = test_set$target,
                predictor = test_set$stepwise_model)
```

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

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

```
## Plot the ROC Curve:
plot(rocCurve, legacy.axes = TRUE, main = "ROC Curve using pROC",print.auc = TRUE)
```




```
# Model Stats
results_4 <- tibble(model = "stepwise_model",predictors = 7,F1 = cm_model4$byClass[7],
                    deviance=stepwise_model$deviance,
                    r2 = 1 - stepwise_model$deviance/stepwise_model$null.deviance,
                    aic = stepwise_model$aic, auc = auc(rocCurve), AICc = aicc(stepwise_model))

model_4 <- as.data.frame(glance(stepwise_model))

results_4 <- cbind(results_4, model_4)

results <- rbind(results,results_4)
```

Model Summary

In this model F1 and AUC scores are identical to first model and higher than the other two models.

PART IV: Selecting Models

Compare Key statistics

The table below summarizes the model statistics for all four of our models. The models are listed from left to right in accordance with the order in which they were described in Part III.

The p-value does not indicate a statistically significant difference. Now let's compare all three model fits, using AIC, corrected AIC, BIC, and loglikehood values.

```
results %>% kable() %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%  scroll_box
```

model	predictors	F1	deviance	r2	aic	auc	AICc	null.devian
baseline_logit_model	11	0.8913043	186.5777	0.6390844	210.5777	0.8924607	211.4443	516.95
full_logit_model	19	0.8888889	139.8985	0.7293804	179.8985	0.8922294	182.2849	516.95
significant_logit_model	7	0.8571429	206.9577	0.5996613	222.9577	0.8600833	223.3533	516.95
stepwise_model	7	0.8913043	194.2181	0.6243048	210.2181	0.8924607	210.6137	516.95

Based on the above table, AIC, AICc and BIC provide a consistent interpretation of model fits:

Model 1 has the second highest AIC, AICc, and BIC values, which is indicative of poor relative fit. Model 2 has lower AIC and AICc but slightly higher BIC than those of Model 1. The interpretation is that Model 2 is superior to Model 1. Based on similar observation Model 2 is superior to Model 3 as well. Model 3 has higher AIC, AICc, and BIC, compared to Model 1,2 and 4. Model 4 has smallest BIC of all 4 models, smaller AIC and AICc compared to model 1 and model 3 which indicates a superior model fit.

Model Interpretation

Between the four models, we decide to select model 4 as the best model based on the above analysis.

```
model_coefficients <- cm_model4$byClass

model_coefficients %>% kable() %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%  scroll_box
```

	x
Sensitivity	0.8913043
Specificity	0.8936170
Pos Pred Value	0.8913043
Neg Pred Value	0.8936170
Precision	0.8913043
Recall	0.8913043
F1	0.8913043
Prevalence	0.4946237
Detection Rate	0.4408602
Detection Prevalence	0.4946237
Balanced Accuracy	0.8924607

Our model has relatively high values of sensitivity and specificity. We conclude that our model is fairly strong. Our last step is to judge our model against a test data set.

Model Prediction

```
eval_df$predict_target <- ifelse(predict.glm(stepwise_model, eval_df,"response") >= 0.5,1,0)


eval_df %>% kable() %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%  scroll_box
```

zn	indus	chas	nox	rm	age	dis	tax	ptratio	lstat	medv	age_new	dis_new	indus_high
0	7.07	0	0.469	7.185	61.1	4.9671	242	17.8	4.03	34.7	209.82323	1.6028362	
0	8.14	0	0.538	6.096	84.5	4.4619	307	21.0	10.26	18.2	319.82597	1.4955747	
0	8.14	0	0.538	6.495	94.4	4.4547	307	21.0	12.80	18.4	369.37168	1.4939597	
0	8.14	0	0.538	5.950	82.0	3.9900	307	21.0	27.71	13.2	307.57994	1.3837912	
0	5.96	0	0.499	5.850	41.5	3.9342	279	19.2	8.77	21.0	126.90026	1.3697076	
25	5.13	0	0.453	5.741	66.2	7.2254	284	19.7	13.15	18.7	232.87099	1.9776026	
25	5.13	0	0.453	5.966	93.4	6.8185	284	19.7	14.44	16.0	364.29309	1.9196395	
0	4.49	0	0.449	6.630	56.1	4.4377	247	18.5	6.53	26.6	187.78103	1.4901362	
0	4.49	0	0.449	6.121	56.8	3.7476	247	18.5	8.44	22.2	190.83272	1.3211156	
0	2.89	0	0.445	6.163	69.6	3.4952	276	18.0	11.34	21.4	248.53755	1.2513906	
0	25.65	0	0.581	5.856	97.0	1.9444	188	19.1	25.41	17.3	382.65135	0.6649534	
0	25.65	0	0.581	5.613	95.6	1.7572	188	19.1	27.26	15.7	375.48731	0.5637216	
0	21.89	0	0.624	5.637	94.7	1.9799	437	21.2	18.34	14.3	370.89841	0.6830463	
0	19.58	0	0.605	6.101	93.0	2.2834	403	14.7	9.81	25.0	362.26621	0.8256656	
0	19.58	0	0.605	5.880	97.3	2.3887	403	14.7	12.03	19.1	384.19056	0.8707493	
0	10.59	1	0.489	5.960	92.1	3.8771	277	18.6	17.27	21.7	357.71530	1.3550875	
0	6.20	0	0.504	6.552	21.4	3.3751	307	17.4	3.76	31.5	53.64607	1.2164250	
0	6.20	0	0.507	8.247	70.4	3.6519	307	17.4	3.95	48.3	252.25771	1.2952476	
22	5.86	0	0.431	6.957	6.8	8.9067	330	19.1	3.53	29.6	12.08542	2.1868038	
90	2.97	0	0.400	7.088	20.8	7.3073	285	15.3	7.85	32.2	51.69902	1.9888738	
80	1.76	0	0.385	6.230	31.5	9.0892	241	18.2	12.93	20.1	88.67552	2.2070869	
33	2.18	0	0.472	6.616	58.1	3.3700	222	18.4	8.93	28.4	196.53005	1.2149127	
0	9.90	0	0.544	6.122	52.8	2.6403	304	18.4	5.98	22.1	173.54979	0.9708925	
0	7.38	0	0.493	6.415	40.1	4.7211	287	19.6	6.12	25.0	121.36339	1.5520418	
0	7.38	0	0.493	6.312	28.9	5.4159	287	19.6	6.15	23.0	79.28065	1.6893391	
0	5.19	0	0.515	5.895	59.6	5.6150	224	20.2	10.56	18.5	203.15155	1.7254416	
80	2.01	0	0.435	6.635	29.7	8.3440	280	17.0	5.99	24.5	82.14543	2.1215427	
0	18.10	0	0.718	3.561	87.9	1.6132	666	20.2	7.12	27.5	336.65536	0.4782198	
0	18.10	1	0.631	7.016	97.5	1.2024	666	20.2	2.96	50.0	385.21749	0.1843196	
0	18.10	0	0.584	6.348	86.1	2.0527	666	20.2	17.64	14.5	327.72087	0.7191560	
0	18.10	0	0.740	5.935	87.9	1.8206	666	20.2	34.02	8.4	336.65536	0.5991661	
0	18.10	0	0.740	5.627	93.9	1.8172	666	20.2	22.88	12.8	366.83036	0.5972969	
0	18.10	0	0.740	5.818	92.4	1.8662	666	20.2	22.11	10.5	359.23080	0.6239043	
0	18.10	0	0.740	6.219	100.0	2.0048	666	20.2	16.59	18.4	398.10717	0.6955443	
0	18.10	0	0.740	5.854	96.6	1.8956	666	20.2	23.79	10.8	380.60130	0.6395354	
0	18.10	0	0.713	6.525	86.5	2.4358	666	20.2	18.13	14.1	329.70151	0.8902752	
0	18.10	0	0.713	6.376	88.4	2.5671	666	20.2	14.65	17.7	339.14697	0.9427769	
0	18.10	0	0.655	6.209	65.4	2.9634	666	20.2	13.22	21.4	229.21924	1.0863373	
0	9.69	0	0.585	5.794	70.6	2.8927	391	19.2	14.10	18.3	253.18974	1.0621903	
0	11.93	0	0.573	6.976	91.0	2.1675	273	21.0	5.64	23.9	352.17118	0.7735744	

Model output

```
# export to .csv for submission
write.csv(eval_df, file = "C:/CUNY/Semester4(Spring)/DATA 621/Assignments/Homework3/Output/crime_logisti
```

Our model prediction output can be found in the below GitHub location -

Model Output