Analysis of Boston Housing Data

Group Leader: Taha Bin Amer

Group Member: Pranav Vinod

University of Massachusetts Dartmouth

CIS 490 - Machine Learning

Supervisor: Professor (Julia) Hua Fang

02/23/2022

Contents

Abstract

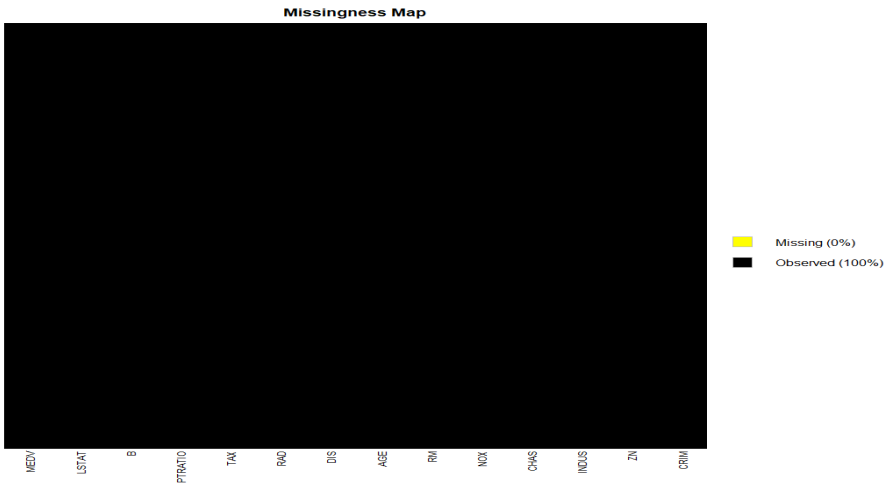
The aim of this project is to analyze the Boston housing dataset and apply different regression techniques to find out the most significant variables in predicting the median value of owner-occupied homes. We will be employing exploratory data analysis to pre-process the data such as finding any missing values and predicting any ambiguities. We will be employing multiple linear regression, ridge regression and lasso regression on the dataset. Lastly, we will compare and contrast the outcomes from these three regression techniques and discuss the results.

Description of the Dataset

This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston, Massachusetts from the 1970 census. The dataset is relatively small in size with 506 entries. There are 14 attributes in total namely, CRIM (per capita crime rate by town), ZN (proportion of residential land zoned for lots over 25,000 sq.ft.), INDUS (proportion of non-retail business acres per town), CHAS (Charles River dummy variable 1 if tract bounds river; 0 otherwise), NOX (nitric oxides concentration parts per 10 million), RM (average number of rooms per dwelling), AGE (proportion of owner-occupied units built prior to 1940), DIS (weighted distances to five Boston employment centers), RAD (index of accessibility to radial highways), TAX (full-value property-tax rate per $10,000), PTRATIO (pupil-teacher ratio by town), B (proportion of blacks by town), LSTAT (% lower status of the population), MEDV (median value of owner-occupied homes in $1000's). MEDV will be our target variable.

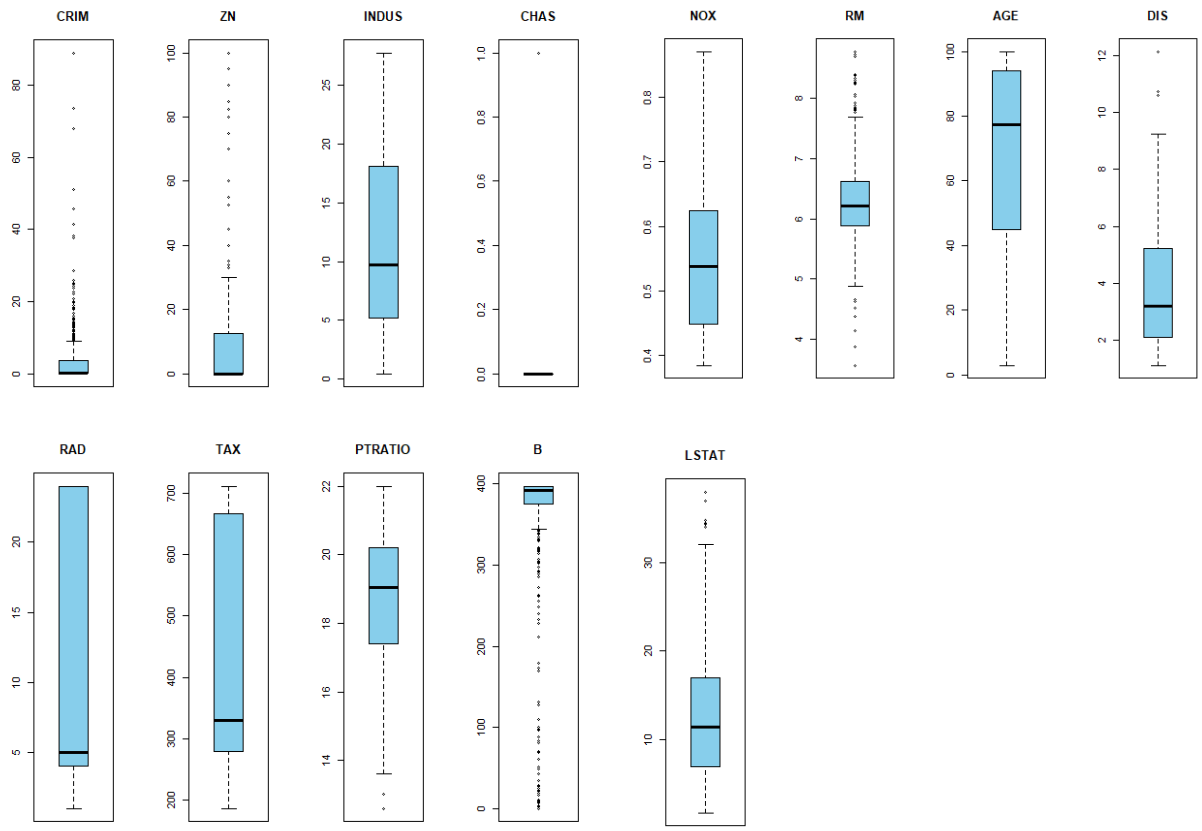
Exploratory Data Analysis

```
# Missing values in data
missmap(data, col = c('yellow', 'black'), y.at = 1, y.labels = '', legend = TRUE)
```



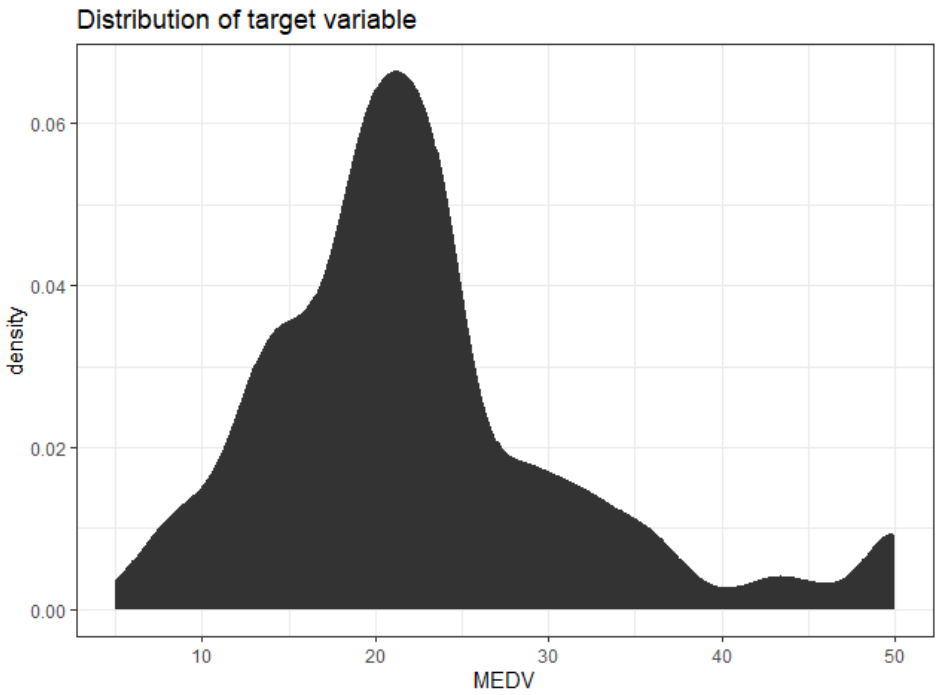
The above plot clearly shows that the data is free from NA's.

```
# Distribution of attributes
boxplot(data$CRIM, main='CRIM',col='Sky Blue')
boxplot(data$ZN, main='ZN',col='Sky Blue')
boxplot(data$INDUS, main='INDUS',col='Sky Blue')
boxplot(data$CHAS, main='CHAS',col='Sky Blue')
boxplot(data$NOX, main='NOX',col='Sky Blue')
boxplot(data$RM, main='RM',col='Sky Blue')
boxplot(data$AGE, main='AGE',col='Sky Blue')
boxplot(data$DIS, main='DIS',col='Sky Blue')
boxplot(data$RAD, main='RAD',col='Sky Blue')
boxplot(data$TAX, main='TAX',col='Sky Blue')
boxplot(data$PTRATIO, main='PTRATIO',col='Sky Blue')
boxplot(data$B, main='B',col='Sky Blue')
boxplot(data$LSTAT, main='LSTAT',col='Sky Blue')
```



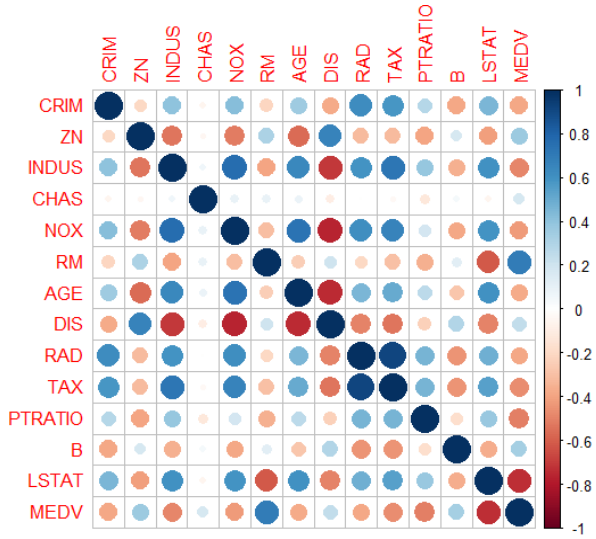
From the above boxplots it can be seen that the attributes CRIM, ZN, RM and B have a lot of outliers.

```
# Distribution of target variable
data %>% ggplot(aes(MEDV)) + stat_density() + theme_bw() + ggtitle("Distribution
of target variable")
```



The above visualization reveals that peak densities of MEDV are in between 15 and 30.

```
# Correlation plot to quantify the relation
corr_matrix <- cor(data)
corrplot(corr_matrix, method = "circle")
```



From the above correlation plot we can see that:

- our target variable MEDV decreases with an increase in CRIM, INDUS, NOX, AGE, RAD, TAX, PTRATIO, and LSTAT.
- MEDV increases with an increase in ZN, and RM.
- CHAS, DIS, and B have little to no effect on our target variable.

Multiple Linear Regression

The Loss function or Residual Sum of Squares (RSS) or Sum of Squared Residue (SSR) or Sum of Squared Loss (SSL) is the function that determines the difference between the observed value of the outcome and the predicted value of the outcome. It is a measure of the cost of predicting the outcome using a model, in this case, the linear model.

Coefficients of the regression line are found such that the loss function is minimized.

Let $\widehat{y} = \widehat{\beta_0} + \widehat{\beta_1} * \widehat{x} + \epsilon$ be the predicted outcome,

Where $\widehat{\beta_1}$ and $\widehat{\beta_0}$ are the slope and intercept respectively and ϵ is the error term.

And y be the observed outcome

$$\epsilon_i = y_i - \widehat{y_i}$$

is the residual (i.e., loss) in the prediction of outcome. Then,

$$\text{SSR/RSS/SSL} = \sum_{i=0}^{n} (y - \widehat{y_i})^2$$

Now, we fit our multiple linear regression model to find coefficients $\widehat{\beta_1}$ and $\widehat{\beta_0}$ such that they minimize the loss function.

To fit our multiple linear model, we carry out the following steps:

1. Dividing our data into training and testing sets.

```
#creating training and testing sets in the ratio of 80:20
set.seed(490)
Random.seed <- c("Mersenne-Twister", 1)

training.indices <- sample(1:nrow(data), 0.8 * nrow(data), replace = FALSE)
training_set <- data[training.indices,]
```

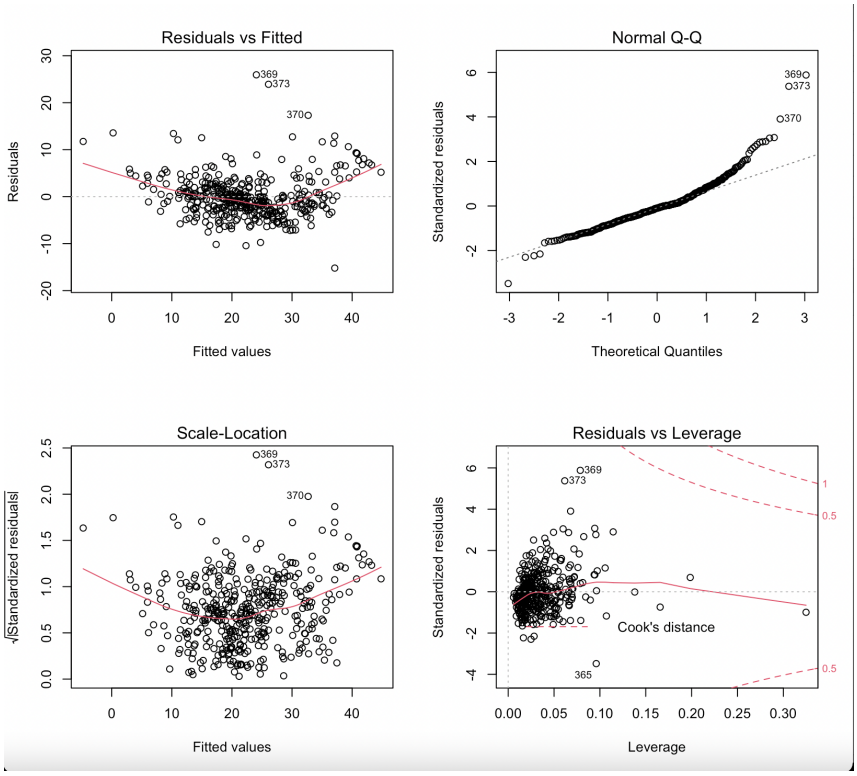
```
testing_set <- data[-training.indices,]
View(training_set)
View(testing_set)
```

2. Running the multiple linear regression on training set

```
multi.model <-
lm(MEDV~CRIM+ZN+INDUS+CHAS+NOX+RM+AGE+DIS+RAD+TAX+PTRATIO+B+LSTAT, data =
training_set)
summary(multi.model)
mse <- mean(residuals(multi.model)^2)
mse
rss <- sum(residuals(multi.model)^2)
rss
par(mfrow=c(2,2))
plot(multi.model)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-15.2075  -2.6613  -0.4224   1.8636  25.9366

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.294381   5.473518   6.814 3.62e-11 ***
CRIM         -0.102513   0.032762  -3.129 0.001886 **
ZN            0.042695   0.014710   2.902 0.003914 **
INDUS        -0.002297   0.065610  -0.035 0.972087
CHAS          2.951422   0.971235   3.039 0.002535 **
NOX         -15.539470   4.122880  -3.769 0.000189 ***
RM            3.539043   0.434237   8.150 5.00e-15 ***
AGE           0.006415   0.014609   0.439 0.660826
DIS          -1.286633   0.212956  -6.042 3.56e-09 ***
RAD           0.290967   0.070701   4.115 4.72e-05 ***
TAX          -0.011738   0.003980  -2.949 0.003376 **
PTRATIO      -1.011033   0.144684  -6.988 1.21e-11 ***
B             0.009800   0.002870   3.414 0.000707 ***
LSTAT        -0.558486   0.055520 -10.059  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.594 on 390 degrees of freedom
Multiple R-squared:  0.7628,    Adjusted R-squared:  0.7549
F-statistic:  96.5 on 13 and 390 DF,  p-value: < 2.2e-16
```

```
> mse <- mean(residuals(multi.model)^2)
> mse
[1] 20.37362
> rss <- sum(residuals(multi.model)^2)
> rss
[1] 8230.944
```



Residual vs Fitted: We can use this plot to check the linearity assumption of our model. Since the residuals for a range of fitted values are non-zero on average and the smooth line joining mean residuals from different vertical regions is a curve, we can conclude that untransformed linear regression is inappropriate for this data. This plot also shows unequal variance with minimum variance in the middle with high variance at both the ends.

Normal Q-Q: Since low values are too low and high values are too high as compared to the reference line, we can say that the Normality assumption is not satisfied.

Scale-Location: Since the line through the center is curved, we can say that the assumption of homoscedasticity is not satisfied for the model.

Residual vs Leverage: We can see that none of the points fall outside of Cook's distance, hence none of the points are influential and their removal would not affect the coefficients.

3. Running the multiple linear regression on testing set

```
#Run multiple linear regression on testing data
multi.model.predictions <- predict(multi.model, testing_set)
```

4. Calculating testing errors

```
#Calculating testing errors
```

```
test.multi.model.ssl <- sum((testing_set$MEDV - multi.model.predictions)^2)
test.multi.model.mse <- test.multi.model.ssl / nrow(testing_set)
test.multi.model.rmse <- sqrt(test.multi.model.mse)
sprintf("SSL/SSR/SSE: %f", test.multi.model.ssl)
sprintf("MSE: %f", test.multi.model.mse)
sprintf("RMSE: %f", test.multi.model.rmse)
```

```
> sprintf("SSL/SSR/SSE: %f", test.multi.model.ssl)
[1] "SSL/SSR/SSE: 2921.923153"
> sprintf("MSE: %f", test.multi.model.mse)
[1] "MSE: 28.646305"
> sprintf("RMSE: %f", test.multi.model.rmse)
[1] "RMSE: 5.352224"
```

5. Running the model on the entire dataset

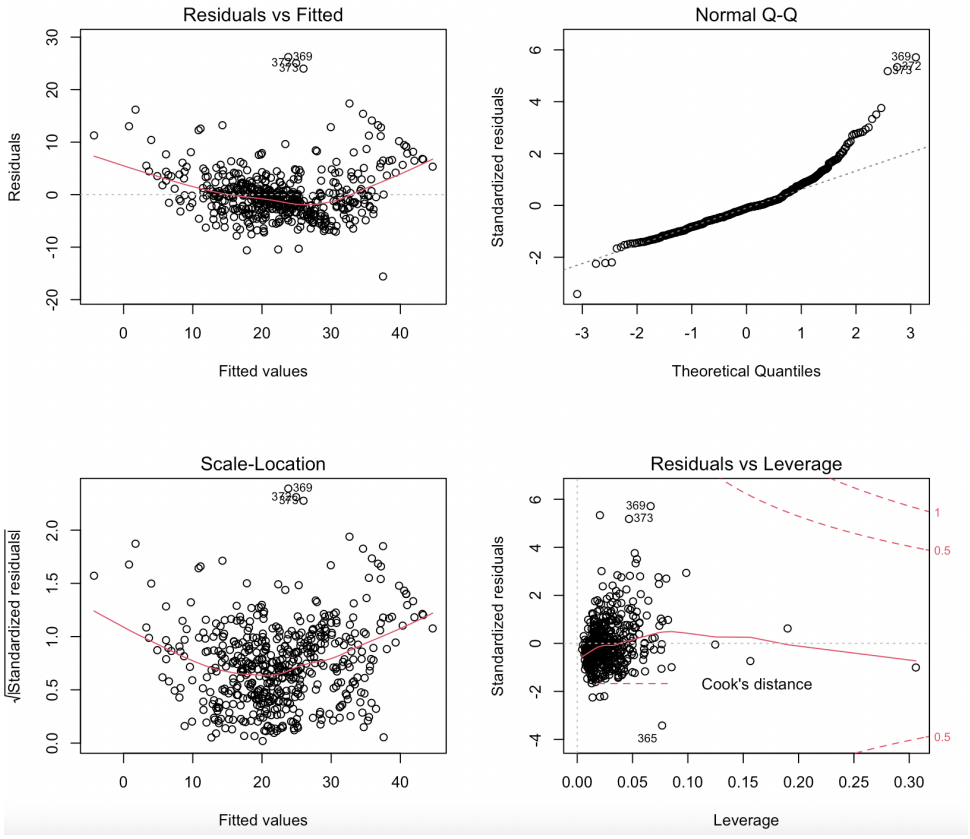
```
#Running the model on the entire dataset
multi.model.full <-
lm(MEDV~CRIM+ZN+INDUS+CHAS+NOX+RM+AGE+DIS+RAD+TAX+PTRATIO+B+LSTAT, data =
data)
summary(multi.model.full)
mse <- mean(residuals(multi.model.full)^2)
mse
rss <- sum(residuals(multi.model.full)^2)
rss
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-15.595  -2.730  -0.518   1.777  26.199

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
CRIM        -1.080e-01  3.286e-02  -3.287 0.001087 **
ZN           4.642e-02  1.373e-02   3.382 0.000778 ***
INDUS        2.056e-02  6.150e-02   0.334 0.738288
CHAS         2.687e+00  8.616e-01   3.118 0.001925 **
NOX         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
RM           3.810e+00  4.179e-01   9.116  < 2e-16 ***
AGE          6.922e-04  1.321e-02   0.052 0.958229
DIS         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
RAD          3.060e-01  6.635e-02   4.613 5.07e-06 ***
TAX         -1.233e-02  3.760e-03  -3.280 0.001112 **
PTRATIO     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
B            9.312e-03  2.686e-03   3.467 0.000573 ***
LSTAT       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.745 on 492 degrees of freedom
Multiple R-squared:  0.7406,    Adjusted R-squared:  0.7338
F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

```
> mse <- mean(residuals(multi.model.full)^2)
> mse
[1] 21.89483
> rss <- sum(residuals(multi.model.full)^2)
> rss
[1] 11078.78
```

6. Plotting accuracy testing plots

```
par(mfrow=c(2,2))
plot(multi.model.full)
```



Residual vs Fitted: We can use this plot to check the linearity assumption of our model. Since the residuals for a range of fitted values are non-zero on average and the smooth line joining mean residuals from different vertical regions is a curve, we can conclude that untransformed linear regression is inappropriate for this data. This plot also shows unequal variance with minimum variance in the middle with high variance at both the ends.
Normal Q-Q: Since low values are too low and high values are too high as compared to the reference line, we can say that the Normality assumption is not satisfied.
Scale-Location: Since the line through the center is curved, we can say that the assumption of homoscedasticity is not satisfied for the model.
Residual vs Leverage: We can see that none of the points fall outside of Cook's distance, hence none of the points are influential and their removal would not affect the coefficients.

7. The final estimated model is:

$$MEDV = 36.46 - 0.110 * CRIM + 0.046 * ZN + 0.020 * INDUS + 2.678 * CHAS - 17.77 * NOX + 3.810 * RM + 0.0007 * AGE - 1.476 * DIS + 0.306 * RAD - 0.012 * TAX - 0.952 * PTRATIO + 0.009 * B - 0.525 * LSTAT$$

The three predictors with the most effect on the median prices of houses are:

1. **NOX (Nitric Oxide concentration parts per 10 million)** - is negatively related to the median price of houses. This is true as a large value of NOX would decrease the price of the houses as it would be unsafe for people to live in that area.
2. **RM (Average number of rooms)** - is positively related to median prices of houses as a house with a greater number of rooms would have a higher price.
3. **DIS (weighted distances to five Boston employment centers)** - is negatively related to the median price of houses as localities close to employment centers would have a higher price since they would be in greater demand as compared to houses that are farther away from employment centers. People would prefer houses close to areas with greater potential and opportunity for employment.

Need for Further Analysis

While the multiple linear model gives a good estimate of the linear model that can be used to fit the Boston Housing Dataset, there are certain improvements that can be made using further analysis.

1. From the summary of the multiple linear model, we can see that certain predictors with a small value of coefficient are included in the final model despite having very low significance on the response variable. By performing variable selection, we can reduce the unnecessary complexity in our model and make it easier to interpret.

2. A drawback of the multiple linear model is that if the number of observations, n is approximately equal to or smaller than the number of variables, p, then the model has a lot of variability (infinite in the latter case). It leads to overfitting and poor predictions. By constraining/shrinking our estimated coefficients, we can significantly reduce the variance in the predicted outcome.

These two possibilities lead us to perform the Ridge and Lasso regression with the aim of achieving a better fit for the data.

Ridge Regression

Ridge regression is a technique to improve upon the multiple linear model by shrinking our estimated coefficients towards 0.

It is very similar to the Least Squares Method except it minimizes a different quantity.
The quantity to minimize for the Ridge regression model is:

$$\sum_{i=i}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Here $\lambda \geq 0$ is a tuning parameter.

The first term $\sum_{i=i}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$ is the Loss function for the multiple linear model.

The second term $\lambda \sum_{j=1}^{p} \beta_j^2$ is called the shrinkage penalty. It is small when the estimated coefficients are small or close to zero. So, it leads to a shrinkage in the estimated coefficients.
Ridge regression would produce a different set of coefficients for each value of the tuning parameter. Therefore, choosing the optimal value of the tuning parameter is important. We choose that value using cross-validation.

The procedure to carry out Ridge regression is as follows:

1. Import necessary libraries and separate the independent and dependent variables

```
library(glmnet)
library(plotmo)

# separating the independent and the dependent variables

set.seed(490)
# x is the data
x <- data[1:13]
# median_income is the dependent variable
median_income <- data$MEDV
```

2. Create a grid of values of the tuning parameter to run our model over

```
# Creating a grid of possible values of lambda to test
grid <- 10^seq(6, -3, length=10)
```

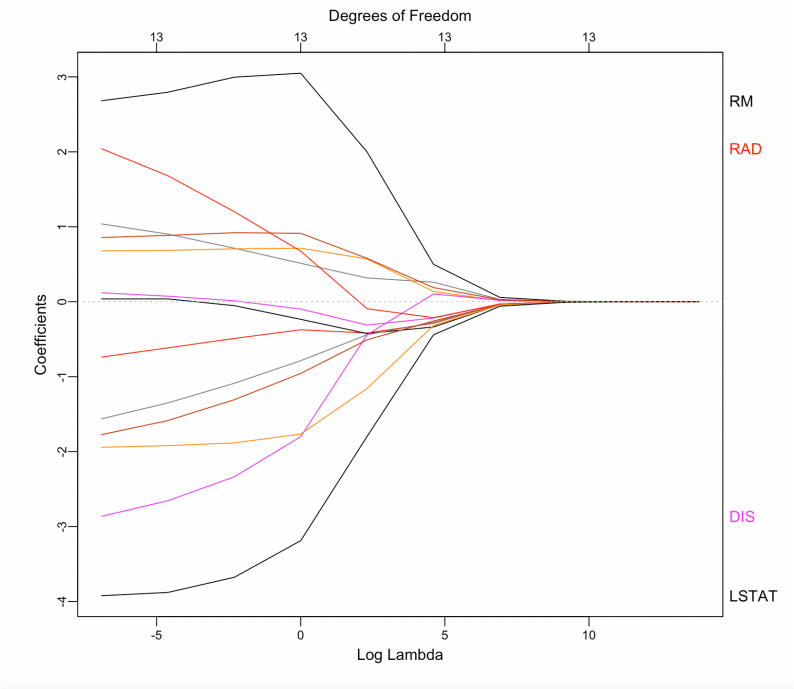
3. Construct a linear model with the following parameters:

a. A scale function to center the mean of each predictor column around the same value.
b. $\alpha = 0$, to specify ridge regression
c. $\lambda = grid$, to run the model over every value in the grid
d. $thresh = 1e - 2$, defines the stopping criterion for optimization
e. $standardize = TRUE$, converts all predictor columns into the same range

```
ridge.mod <- glmnet(scale(x), median_income, alpha=0, lambda=grid,
thresh=1e-2, standardize = TRUE)
```

4. Plotting the results over different values of λ

```
plot_glmnet(ridge.mod, xvar = "lambda", label = 4)
```



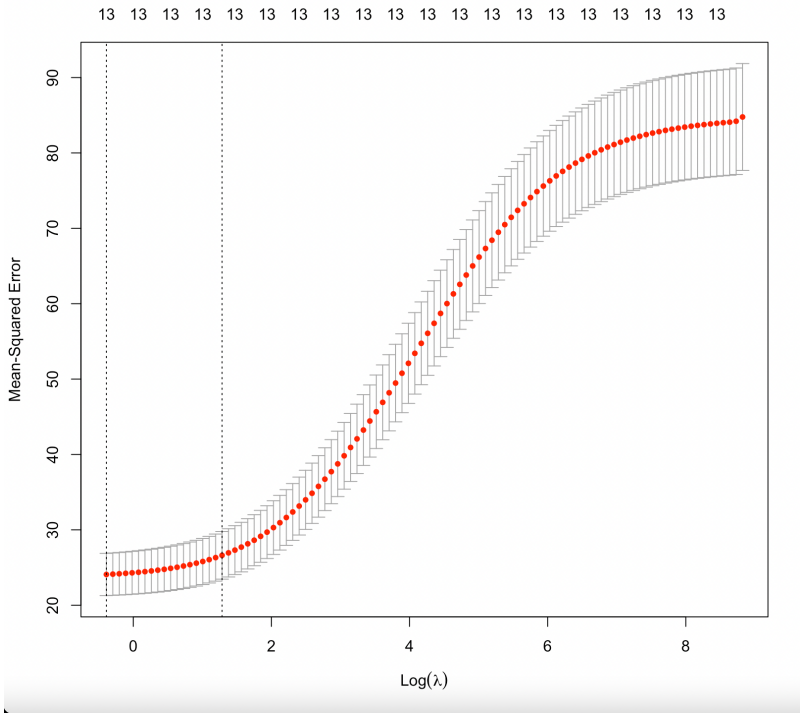
When the tuning parameter is small, the coefficients have a value that is equal to the value of the coefficients when calculated using the least-squares loss function. The effect of the addition of the penalty term can be seen as the value of the tuning parameter increases.
Here, we can see that as the value of the tuning parameter λ increases, the coefficients shrink towards zero. At the same time, the number of coefficients remains the same. The model does not eliminate any of the coefficients despite a very small value as well.

5. Perform cross-validation to find the optimal λ

```
# Perform cross-validation
cv.out <- cv.glmnet(scale(x), median_income, alpha=0, nfolds = 10)
cv.out
```

6. Plot the MSE for each value of λ

```
# Plot the MSE for each lambda value
plot(cv.out)
```



We can notice that as the value of the tuning parameter increases, the MSE increases and reaches a minimum value. We allow for a range of 1se from the minimum and choose the optimal value within that range.

For our case, the optimal value of the tuning parameter is the one at 1se from the minimum of 3.617.

7. Determine the most optimal λ

```
# Determine best lamda
best.lambda <- cv.out$lambda.1se
best.lambda
```

```
> best.lambda
[1] 3.61703
```

8. Run the Ridge regression at the best λ and evaluate MSE and RMSE

```
# Run ridge regression at the best lambda and evaluate using MSE
ridge.final <- glmnet(scale(x), median_income, alpha=0, lambda=best.lambda,
thresh=1e-2, standardize = TRUE)
predict(ridge.final, type="coefficients", s=best.lambda)

# Evaluate MSE and RMSE of the ridge model with the optimal lambda
ridge.pred <- predict(ridge.final, s=best.lambda, newx=scale(x))

print(paste('MSE:', mean((ridge.pred - median_income)^2)))
print(paste('RMSE:', sqrt(mean((ridge.pred - median_income)^2))))
print(paste('SSE:', sum(((ridge.pred - median_income)^2))))
print(paste('SST:', sum(((median_income - mean(median_income))^2))))
print(paste('R^2:', 1 - sum(((ridge.pred -
median_income)^2))/sum(((median_income - mean(median_income))^2))))
```

```
14 x 1 sparse Matrix of class "dgCMatrix"
                    s1
(Intercept) 22.5328063
CRIM        -0.4252222
ZN           0.2754390
INDUS       -0.4086552
CHAS         0.7202377
NOX         -0.7937939
RM           2.7547333
AGE         -0.4247211
DIS         -1.1335269
RAD          0.2332076
TAX         -0.5121380
PTRATIO     -1.5373380
B            0.7450672
LSTAT       -2.3868428
```

```
>
> print(paste('MSE:', mean((ridge.pred - median_income)^2)))
[1] "MSE: 25.5517665382136"
>
> print(paste('RMSE:', sqrt(mean((ridge.pred - median_income)^2))))
[1] "RMSE: 5.05487552153499"
```

```
> print(paste('SSE:', sum(((ridge.pred - median_income)^2))))
[1] "SSE: 12929.1938683361"
> print(paste('SST:', sum(((median_income - mean(median_income))^2))))
[1] "SST: 42716.2954150198"
> print(paste('R^2:', 1 - sum(((ridge.pred - median_income)^2))/sum(((median_income - mean(median_inc
ome))^2))))
[1] "R^2: 0.697324083403778"
```

9. The final estimated model is

$$MEDV = 22.533 - 0.425 * CRIM + 0.275 * ZN - 0.408 * INDUS + 0720 * CHAS - 0.793 * NOX + 2.754 * RM - 0.424 * AGE - 1.133 * DIS + 0.233 * RAD - 0.512 * TAX - 1.537 * PTRATIO + 0.745 * B - 2.386 * LSTAT$$

The three predictors with the most effect on the median prices of houses according to the ridge model are

1. **LSTAT (% lower status of the population)** - is negatively related to the median prices because the greater the percentage of the lower status of a population in an area, the lower will the prices of houses in that area. As the people living there would not be able to afford costly houses, it would stand to reason that areas with a lower LSTAT, would have a higher percentage of people from the higher status of the population and hence houses with a greater median price.
2. **RM (Average number of rooms)** - is positively related to median prices of houses as a house with a greater number of rooms would have a higher price.

3. **PTRATIO (pupil-teacher ratio by town)** - is negatively related to the median price of houses as areas with a smaller pupil to teacher ratio would be in greater demand as compared to towns with a large pupil to teacher ratio. A smaller ratio would ensure that a student gets an adequate amount of attention and people would prefer that as compared to a town with very few teachers for a large number of students.

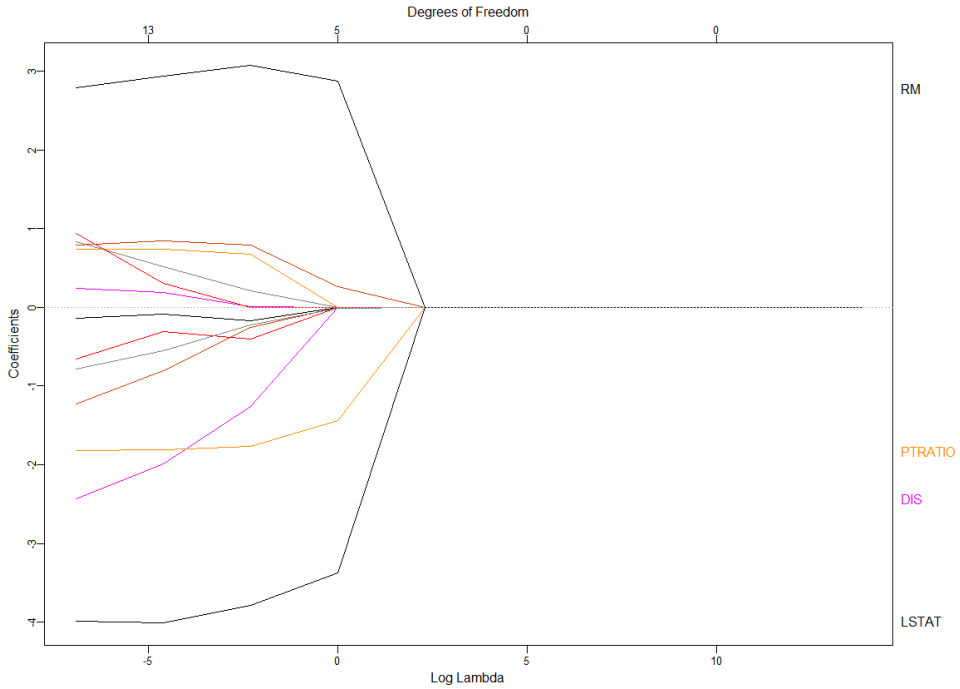
Lasso Regression

Lasso regression is a technique used to solve the problem of overfitting caused by collinearity. It basically shrinks regression coefficients with some shrunk to zero. Thus, it also helps with feature selection.

$$SSE_{Lasso} = \Sigma(Y - \hat{Y})^2 + \lambda \Sigma |\beta|$$

```
grid <- 10^seq(6, -3, length=10)
x <- data[,-14]
y <- data[, 'MEDV']
lasso.mod <- glmnet(scale(x), y, alpha=1, lambda=grid, thresh=1e-2,
standardize = TRUE)
plot_glmnet(lasso.mod, xvar="lambda", label = 4)
lasso.cv.out <- cv.glmnet(scale(x), y, alpha=1, nfolds = 10)
lasso.cv.out
plot(lasso.cv.out)

lasso.best.lambda <- lasso.cv.out$lambda.1se
lasso.best.lambda
lasso.final <- glmnet(scale(x), y, alpha=1, lambda= lasso.best.lambda,
thresh=1e-2, standardize = TRUE)
predict(lasso.final, type="coefficients", s=lasso.best.lambda )

lasso.pred <- predict(lasso.final, s= lasso.best.lambda, newx=scale(x))
print(paste('MSE:', mean((lasso.pred - y)^2)))
print(paste('RMSE:', sqrt(mean((lasso.pred - y)^2))))
print(paste('SSE:', sum(((lasso.pred - median_income)^2))))
print(paste('SST:', sum(((median_income - mean(median_income))^2))))
print(paste('R^2:', 1 - sum(((lasso.pred -
median_income)^2))/sum(((median_income - mean(median_income))^2))))


plot_glmnet(lasso$finalModel, xvar = 'dev', label = 4)
plot(varImp(lasso, scale = F))
predict(lasso$finalModel, type="coefficients", s=lambda )
```

For lasso regression we first define our grid that will store our chosen sequence of values for λ. In this case, our range is from 10^{-3} to 10^{6}. We then partition the dataset into x and y, where x contains all attributes except our target variable i.e. MEDV, and y contains only our target attribute. After that, we use the glmnet function from the glmnet library to make our lasso model. Standardized values of x are provided so that all attributes contribute equally. For lasso regression, α=1 is set and for λ we supply our grid created earlier.

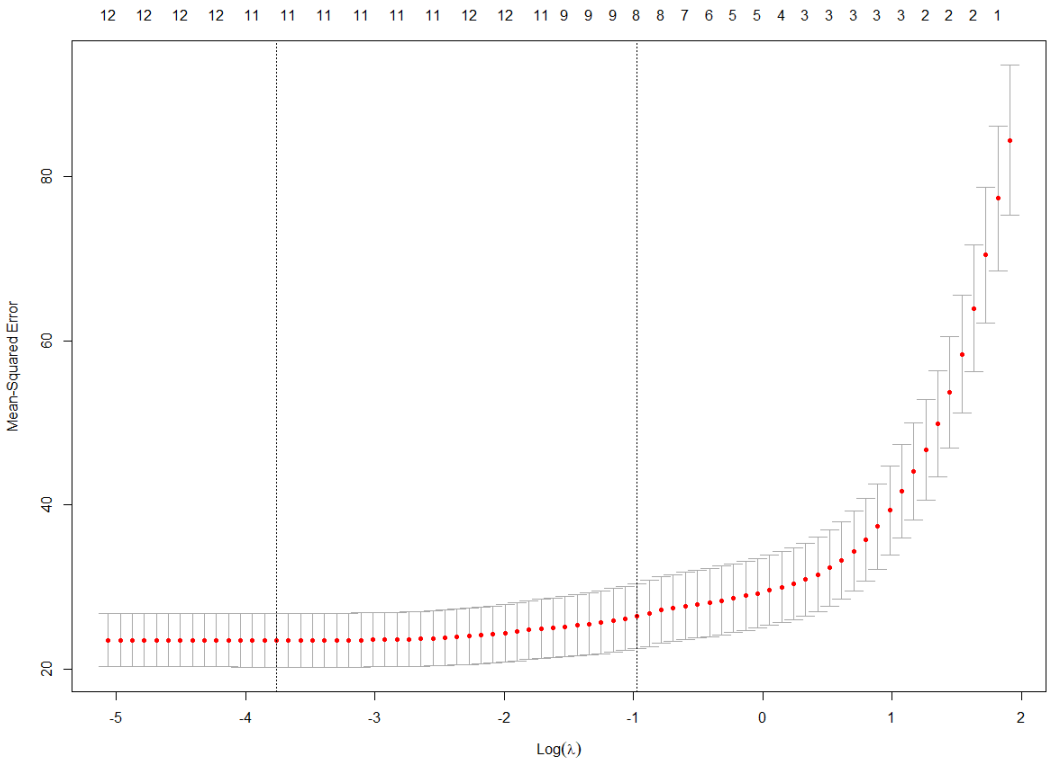


The above plot shows the top 4 most significant attributes while predicting the value of MEDV. It seems that LSTAT, RM, DIS, and PTRATIO are the most significant.

We then make use of cross-validation to estimate the most optimal value for λ. Cross-validation is a technique that partitions your dataset into pre-described equal parts (10 in our case). It then uses 9 of them for model training and the remaining one for error estimation.

```
Call:  cv.glmnet(x = scale(x), y = y, nfolds = 10, alpha = 1)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min 0.0233    62   23.50 3.251      11
1se 0.3789    32   26.46 3.971       8
> 
```

This gives us a smaller range for possible values of λ. In this case, the minimum value of λ=0.0233 and the 1 standard error value is λ=0.3789. We chose λ=0.3789 as our best λ.



This plot clearly shows that as we increase the log(λ) our mean squared error increases exponentially.

We then re-run the model but this time instead of giving it a range of values for λ, we supply our best λ=0.3789. Following are the final coefficients that we get for our model attributes.

```
> predict(lasso.final, type="coefficients", s=lasso.best.lambda )
14 x 1 sparse Matrix of class "dgCMatrix"
                      s1
(Intercept) 22.53280632
CRIM        -0.17469717
ZN           .
INDUS       -0.73779945
CHAS         0.50891194
NOX         -0.33197158
RM           3.61610775
AGE         -0.34849784
DIS         -0.83676958
RAD          .
TAX         -0.02593487
PTRATIO     -1.64025516
B            0.74027277
LSTAT       -2.74410076
>
```

As mentioned earlier lasso performs coefficient shrinkage and even reduces some coefficients to zero. In this case, the coefficients ZN and RAD are shrunk to 0 whereas LSTAT, RM, PTRATIO and DIS seem to be the most significant.

```
> print(paste('MSE:', mean((lasso.pred - y)^2)))
[1] "MSE: 25.1695185572246"
> print(paste('RMSE:', sqrt(mean((lasso.pred - y)^2))))
[1] "RMSE: 5.01692321619781"
>
```

```
> print(paste('SSE:', sum(((lasso.pred - median_income)^2))))
[1] "SSE: 12735.7763899556"
> print(paste('SST:', sum(((median_income - mean(median_income))^2))))
[1] "SST: 42716.2954150198"
> print(paste('R^2:', 1 - sum(((lasso.pred - median_income)^2))/sum(((median_income - mean(median_inc
ome))^2))))
[1] "R^2: 0.701852038754336"
```

The mean squared error for our final lasso regression model is MSE=25.170 and the root mean squared error is RMSE=5.017 when rounded off to 3 decimal places. The final equation is as follows:

$$MEDV = 22.533 - 0.175 * CRIM - 0.738 * INDUS + 0.509 * CHAS - 0.332 * NOX + 3.616 * RM - 0.348 * AGE - 0.837 * DIS - 0.026 * TAX - 1.640 * PTRATIO + 0.740 * B - 2.744 * LSTAT$$

The three predictors with the most effect on the median prices of houses according to the lasso model are

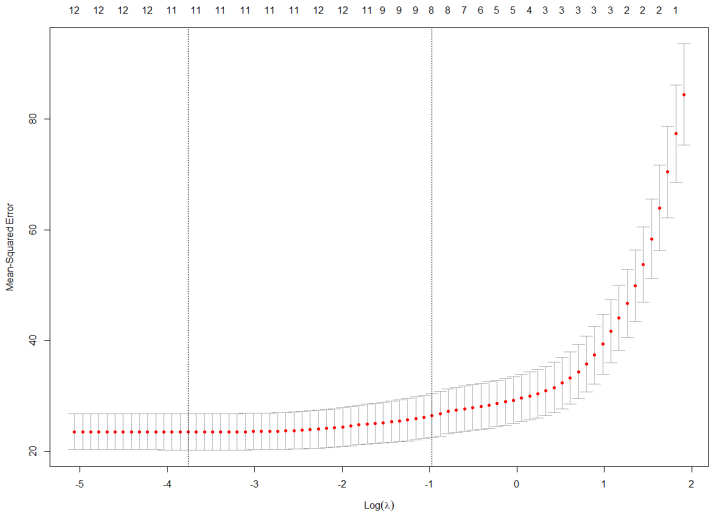
1. **RM (Average number of rooms)** - is positively related to median prices of houses as a house with a greater number of rooms would have a higher price.
2. **LSTAT (% lower status of the population)** - is negatively related to the median prices because the greater the percentage of the lower status of a population in an area, the lower will the prices of houses in that area. As the people living there would not be able to afford costly houses, it would stand to reason that areas with a lower LSTAT, would have a higher percentage of people from the higher status of the population and hence houses with a greater median price.
3. **PTRATIO (pupil-teacher ratio by town)** - is negatively related to the median price of houses as areas with a smaller pupil to teacher ratio would be in greater demand as compared to towns with a large pupil to teacher ratio. A smaller ratio would ensure that a student gets an adequate amount of attention and people would prefer that as compared to a town with very few teachers for a large number of students.

Cross-Validation Algorithm

Cross-validation is one of the most commonly used resampling techniques. It is used to calculate the test error associated with a statistical technique and evaluate its performance in the absence of a large designated testing set. It does so by holding back a subset of the training data from the fitting process and applying statistical techniques to that subset of data.

We have used the **k-fold Cross Validation** algorithm to find out the optimal value of the tuning parameter. The algorithm for the 10-fold cross-validation process is as follows:

1. Partition the entire dataset into 10 parts.
2. Use 9 of the 10 parts for model training and consider them as training dataset.
3. Use the remaining 1 dataset for error estimation and consider that as the testing dataset.
4. This process is repeated 10 times. Each time a different set is considered as testing set.
5. This process results in 10 estimates of MSE and 10-fold CV estimate is computed by taking the average of these values, $$CV_{(k)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$
6. Now, this process is repeated for each value of the tuning parameter λ, and the value of the λ for which the 10-fold CV error is the least is the tuning parameter that minimizes the loss function for Lasso and Ridge regression.



We can see the trend of MSE as a function of the tuning parameter λ. As the value of λ increases, initially the value of MSE decreases until it reaches a minimum after which it keeps on increasing. We can also see that as the value of λ increases the number of attributes, shown on top of the graph keeps on decreasing. The Lasso model sets some of the attributes to zero and performs variable selection.

Summary Table

Statistic	Multiple Linear Regression			Ridge Regression	Lasso Regression
	Train	Test	All	All	All
MSE	20.374	28.646	21.895	25.551	25.170
RMSE	4.514	5.352	4.679	5.055	5.017
1se λ (λ*)	-	-	-	3.6170	0.3789
RSS	8230.944	2921.923	11078.78	12929.193	12735.776
RSE	4.594	-	4.745	-	-
R^2	0.763	-	0.740	0.697	0.702
Adj. R^2	0.755	-	0.734	-	-
F-statistic	96.5	-	108.1	-	-

References

https://www.kaggle.com/sukeshpabba/linear-regression-with-boston-housing-data

https://www.kaggle.com/kritikseth/eda-and-linear-regression-on-boston-housing-in-r

An Introduction To Statistical Learning With Applications in R by Gareth James, Daniela Witten Trevor Hastie, Robert Tibshirani