John Dixon
CS 4375.501
17 February 2019

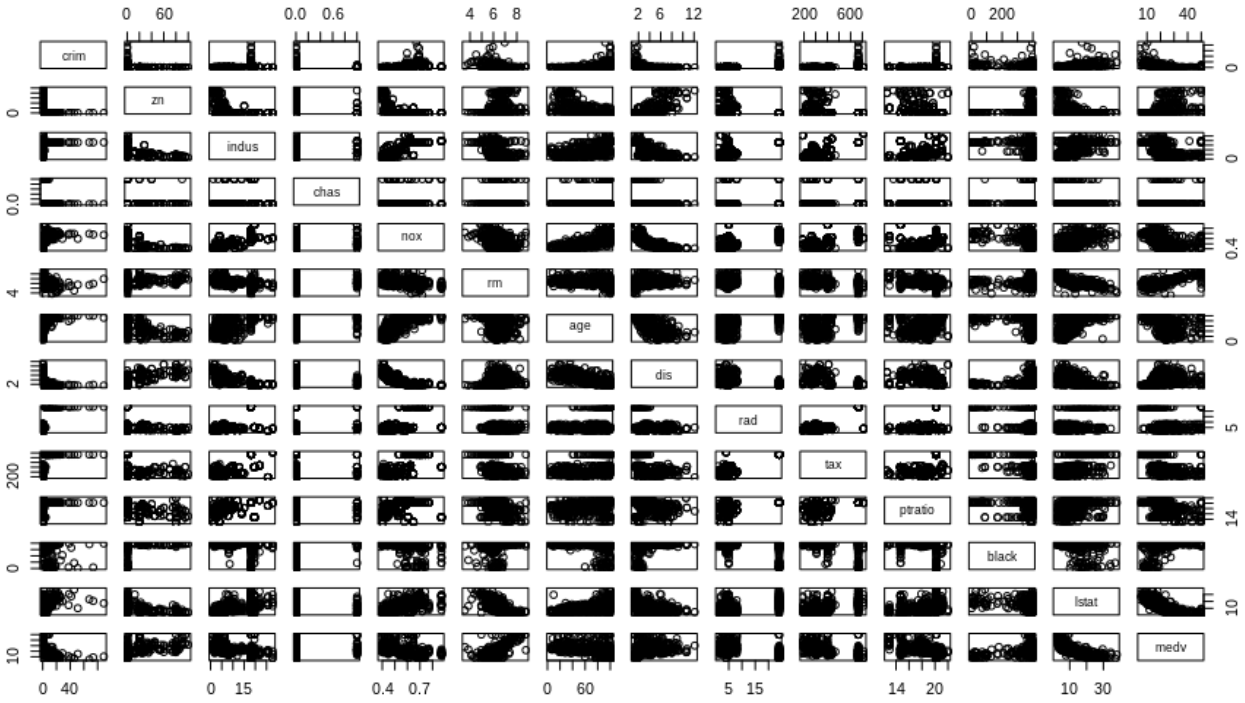
Project 1 Report

After loading the Boston data set in RStudio, I learned from the R documentation that this data set has information about 506 different suburbs in Boston. The data frame has 14 different predictors, including crime rate, residential zoning proportions, average number of rooms per house, and median home value. To count the number of NA values for each predictor, I used the sapply() function, which output the following results in RStudio:

```
> sapply(df, function(x) sum(is.na(x) == TRUE)) # count NAs in each column
   crim      zn   indus    chas     nox      rm     age     dis     rad     tax ptratio   black   lstat    medv
      0       0       0       0       0       0       0       0       0       0       0       0       0       0
```

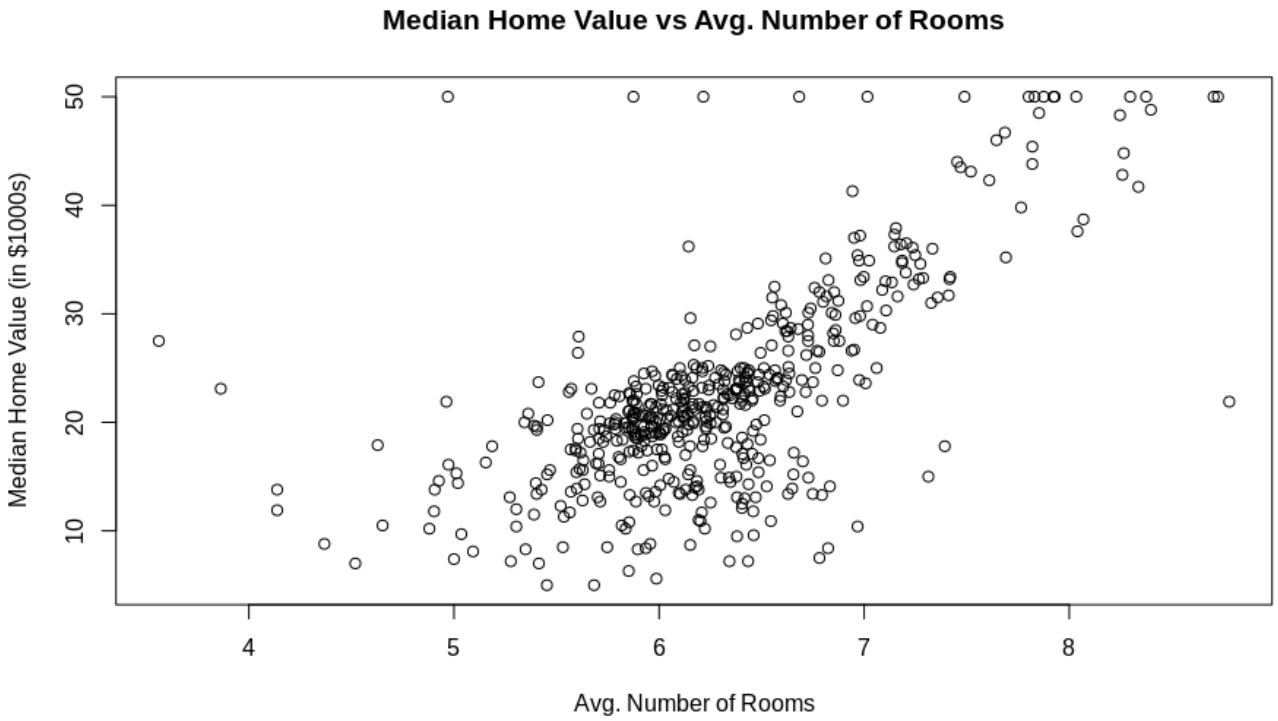
The above output indicates that none of the predictors in the data set had any NA values.

After exploring the Boston data set, I decided to make graphs of the data to help me visualize it and identify any possible correlations between predictors. First, I used the pairs() function to help see correlations between all the predictors, which output as follows:



While this graph is useful for identifying all possible correlations between predictors, I was primarily interested in determining if a correlation existed between the average number of rooms per house (rm) and the median home value (medv). Thus, I created a scatterplot graph using only these two predictors as follows:

John Dixon
CS 4375.501
17 February 2019



Based on the scatterplot, there is a strong indication of a positive correlation between the predictors rm and medv.

Having observed evidence of strong correlation between these predictors, I proceeded to make a linear regression model between them. I first created a training set, which consisted of the first 400 observations in the Boston data frame, and a testing set, which consisted of the remaining 106 observations. To produce the model in R, I used the lm() function, which produced a model with the following coefficients:

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -35.2609     2.6289  -13.41   <2e-16 ***
rm            9.4055     0.4121   22.82   <2e-16 ***
```

I then used this model to predict the medv on the testing set. To measure the accuracy of the model, I compared the predicted medv values to the actual medv values in the test set by calculating the correlation, the mean squared error, and the root mean squared error. I also recorded the total time taken for the lm() function by finding the difference between proc.time() before and after the function executed. The end results were as follows:

```
> # print results
> print(paste("Elapsed time for linear regression:", lr_time, "seconds"))
[1] "Elapsed time for linear regression: 0.000999999999748979 seconds"

> print(paste("Correlation:", corr))
[1] "Correlation: 0.240602667889326"

> print(paste("MSE:", mse))
[1] "MSE: 79.6258305871682"

> print(paste("RMSE:", rmse))
[1] "RMSE: 8.92333068910752"
```

John Dixon
CS 4375.501
17 February 2019

I also implemented an algorithm to perform linear regression in C++. The algorithm estimates $\hat{w}$, the slope of the regression line, and $\hat{b}$, the y-intercept of the regression line, by using the ordinary least squares (OLS) method. The OLS method uses the following equations:

$$\hat{b} = \overline{y} - \hat{w}\overline{x} \qquad\qquad \hat{w} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

Using these equations, I wrote the following algorithm in C++:

```
template<class T>
void vector_ols(const vector<T> & x, const vector<T> & y, double coef[2]) {
	double xmean = vector_mean<T>(x);
	double ymean = vector_mean<T>(y);
	double numerator = 0;
	double denominator = 0;
	// can only use as many elements as the smallest vector has
	size_t length = min<size_t>(x.size(), y.size());

	for (size_t i = 0; i < length; ++i) {
		numerator += ((x[i] - xmean) * (y[i] - ymean));
		denominator += pow((x[i] - xmean), 2.0);
	}
	coef[0] = (numerator / denominator);	// slope
	coef[1] = (ymean - (coef[0] * xmean));	// y-intercept
}
```

This algorithm takes two vectors as input; one vector has the rm values from the training set, and the other has the medv values from the training set. The algorithm computes $\hat{w}$, the slope, and $\hat{b}$, the y-intercept, based on the OLS equations and stores these into the coefficient array. When I ran this algorithm on the training set, it calculated the model coefficients as follows:

```
Coefficients
	(Intercept) -35.260948
	rm          9.405502
```

These are the same coefficients that were estimated by the lm() function in R. To gauge the speed of the C++ algorithm, I used the clock() function to determine the number of clock cycles that passed right before vector_ols() started and right before it stopped. I then divided the difference between the number of clock cycles by the macro CLOCKS_PER_SEC, which is the number of clock cycles executed per second on a given machine. When I ran the C++ algorithm on the same machine on which I had run the R script, the C++ algorithm took about 0.000090 seconds on average. Compared to R, which took nearly 0.001 seconds to perform linear regression, the C++ algorithm is over 10 times faster.

After using vector_ols() to make a linear model, I used the model to predict the medv of the test data as I had done in R. To compare the predicted medv values to the actual values, I also implemented algorithms in C++ for calculating the Pearson correlation coefficient and the mean square error. The

John Dixon
CS 4375.501
17 February 2019

Pearson correlation coefficient of two variables x and y equals the covariance (cov(x,y)) of the variables divided by the product of their standard deviations:

$$corr(x,y)=\frac{cov(x,y)}{sd(x)*sd(y)}$$

The covariance and the standard deviation are computed as follows:

$$cov(x,y)=\frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{n-1} \qquad sd(x)=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n-1}}=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})(x_i-\overline{x})}{n-1}}=\sqrt{cov(x,x)}$$

The mean squared error is the average of the squared differences between the actual values and the predicted values:

$$mse=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y})^2$$

When I calculated the correlation, the mean squared error, and the root mean squared error, I had the following results:

```
Elapsed time for linear regression: 0.000090 seconds
Correlation:      0.240603
MSE:              79.625831
RMSE:             8.923331
```

Thus, the predictions made by the C++ algorithm had the same correlation, MSE, and RMSE values as those of the R algorithm.