**Final Project  
Erica Volden  
Intro to Data Science (DS 210)**

**Introduction and Stating the Question  
10 pts**

This analysis will be conducted using a modified version of the Boston Housing dataset in which data was obtained from the US Census of housing around the Boston Massachusetts area. I will analyze and interpret the data to answer the following question, “Can nitric oxide levels (NOX) be predicted by the proportion of non-retail business acres per town (INDUS)?”

The Boston Housing dataset was retrieved from <https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html> and modified ([modified\_BostonHousing.csv](https://canvas.jccc.edu/courses/76484/files/16418376?wrap=1)) by Alice Chen, instructor of Intro to Data Science at Johnson County Community College. This dataset contains 506 cases and looks at 14 different features that can contribute to changes in housing prices and nitric oxide concentrations. The following list of attributes and their descriptions was pulled directly from the [Toronto.edu](https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html) website listed above.

“There are 14 attributes in each case of the dataset. They are:

1. CRIM - per capita crime rate by town
2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS - proportion of non-retail business acres per town.
4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX - nitric oxides concentration (parts per 10 million)
6. RM - average number of rooms per dwelling
7. AGE - proportion of owner-occupied units built prior to 1940
8. DIS - weighted distances to five Boston employment centers
9. RAD - index of accessibility to radial highways
10. TAX - full-value property-tax rate per $10,000
11. PTRATIO - pupil-teacher ratio by town
12. B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT - % lower status of the population
14. MEDV - Median value of owner-occupied homes in $1000's

Note

Variable #14 seems to be censored at 50.00 (corresponding to a median price of $50,000); Censoring is suggested by the fact that the highest median price of exactly $50,000 is reported in 16 cases, while 15 cases have prices between $40,000 and $50,000, with prices rounded to the nearest hundred. Harrison and Rubinfeld do not mention any censoring.”

**Exploratory Data Analysis  
40 pts**

Before jumping straight into the next step in the data science life cycle, I needed to ensure that the data was clean. This meant that I needed to check for any incorrect or missing data that would skew the analysis and handle those values as I saw fit. To begin, I first had to import a few libraries of code from which I could borrow the functions and operations I would be using to do the analysis. To accomplish that, I executed the code: import pandas as pd, import seaborn as sns and import matplotlib.pyplot as plt, then came reading in the csv file that contained the data using



df = pd.read\_csv('modified\_BostonHousing.csv'), last I ran the simple df command that verified the csv file had been loaded in correctly. Now that the libraries were imported and the data had been loaded, I could begin the exploratory data analysis.

First, I checked for any incorrect data or outliers. An outlier is a value that is mathematically outside the range of the other values. To check for outliers, I used the command: df.describe(), where df is the name I have given my data frame or table and the describe function quite literally describes the data. This command showed me a table consisting of the 14 variables across the top as columns and 8 rows which contain statistical data for each variable such as the average, the minimum and maximum values, and the standard deviation and quartiles which we will look at later. The min (minimum or lowest value) and max (maximum or highest value) are the values I am looking at to determine if any outliers exist. For outliers to be obvious, they need to be either extremely high or extremely low.

A screenshot of a computer

AI-generated content may be incorrect.



In the case of this housing data, I immediately noticed that the min value for the column labeled ‘tax’ was a negative value. I ran the command: df.sort\_values('tax', ascending=True).head(5), to sort the table by the values in the ‘tax’ column in ascending order and look at the top 5 rows. This showed me that the ‘tax’ column actually had 3 negative values. While not all negative values are considered outliers, in this instance the values represented the full-value property tax rates. People don’t pay negative taxes on the property they own so I decided the negative values were likely just input incorrectly.  I could have replaced the values with the mean, or average, tax value but the entries themselves brought the mean value down, so it wouldn't be very accurate. Instead, to fix these errors, I opted to delete these three rows of data all together by using the command: df = df[df.tax > 1]. This snippet of code basically says that the data frame now consists of ONLY rows where the values in the ‘tax’ column are greater than 1, effectively deleting the rows that contained the negative values.

The next potential outlier I noticed was the min value for the ‘b’ column at 0.32. This number compared to the mean of 356.67 seemed to be a no brainer of an outlier. However, I decided to do the math to make sure. The equation 1000(Bk-0.63)^2 was given in the variable description of the ‘b’ column, where Bk is the proportion of blacks by town. I did the math in reverse to make sure the value of 0.32 was possible to achieve and it turns out that it is entirely possible. I then looked at the df.describe() table to see what the quartile values were. The quartile values consist of the value at the point where a percentage of data falls below a specified quarter. First all of the values in a given column are ordered least to greatest, then the middle value or median is found, from there the data is split into 4 equal parts (quarters). The values at which the data is split are labeled, 25% at the first split, 50% at the second split and 75% at the third. Quartiles are useful for helping to visualize the data and see the ranges without having to look at every row. For this instance, looking at the first quartile (25% on the table) gave me a reason to delve further into the data because the range between the first quartile value of 375.38 and the minimum value of 0.32 was so large. I decided to look at the top 30 rows of data by executing the command: df.sort\_values('b', ascending=True).head(30), similar to the previous code where I looked at the sorted ‘tax’ column. This showed that the values in the ‘b’ column steadily increased and that the cause for the drastic range of values was attributed to the difference in the proportion of blacks by town as the description indicated. With all this information in mind, I chose to keep the min value row in place and continue cleaning the data.

From here I again consulted the df.describe() table to make sure I hadn’t missed anything. I noticed two max values that seemed a little high compared to the third quartile (75%), located in the ‘crim’ and ‘zn’ columns. I ran a version of the code from before for each column, but instead of .head() at the end I used .tail() to look at the bottom of the column to see the highest values. With the ‘crim’ column I had to input 20 in the parentheses after tail because ‘crim’ had null values, which I will explain later, that I needed to see beyond. For both columns the max value ended up being within range and no further action needed to be taken.

When viewing the max values on the df.describe() table I noticed a value that was not an outlier in the traditional sense, however it was off in a different way. The ‘medv’ column description from the U.S. census website had a note attached to it that stated the variable medv “seems to be censored at 50.00 (corresponding to a median price of $50,000); Censoring is suggested by the fact that the highest median price of exactly $50,000 is reported in 16 cases, while 15 cases have prices between $40,000 and $50,000, with prices rounded to the nearest hundred.” The max value listed was 55.0 so I took a closer look by running the code to sort the values in that column using the .tail(20) ending. I ran it with a 20 row view because that column has 15 null values I had to see past. This output showed me that there was only one entry of 55.0 with the next few being 50.0 as expected. To fix this, I executed the command: df.medv[df.medv == 55.0] = df.medv.mean(), which locates the desired value within the ‘medv’ column and replaces it with mean ‘medv’ value. Another glance at the df.describe() table verifies that it had, in fact, been changed.

Once the outliers had been taken care of, it was time to check for any missing or null values. I looked at the df.describe() table and noted the values in the 8 rows provided. One set of values we haven’t discussed yet is the one listed in the first row of the table labeled ‘count’. These values tell me how many of the remaining 503 cases that were observed have a value entered into each column. It was here that I noticed not all 14 columns had the expected value of 503. The 4 columns that had a count less than 503 were ‘crim’(493), ‘rm’(498), ‘lstat’(484) and ‘medv’(488), these are the missing/null values I was checking for.

A screenshot of a computer screen

AI-generated content may be incorrect.



For each column with missing values, I isolated the column to verify how many missing values it had using the command: sum(df['column\_name'].isnull()) where I inserted the column name where it says ‘column\_name’. The outcome of running this code was an integer equal to the number of missing values. Next, I ran the code: df.sort\_values('column\_name', ascending=True).tail(n), again inserting the column name and, in the parentheses after .tail, I entered the number of missing values plus 10(n). This gave me a table that still had all 14 variables, but now the rows were sorted in ascending order according to the column name I input, and the table contained the last (n) rows of the dataset after sorting. Sorting the table in this manner allowed me to not only see the null values, but also the 10 highest values. This helped me understand the data and make my decision on how to further handle the null values. For handling purposes, I decided it was best to fill the missing values in each column with the mean of that column by executing the command: df.column\_name = df.column\_name.fillna(df.column\_name.mean()), inserting the column names as done previously. I ran the sum(df['column\_name'].isnull()) command after filling each column to verify the code worked, then continued to the next column and .isnull() command. Once all the columns were complete, I ran a final df.describe() command to view my clean data and was satisfied with the outcome.

Now that the data had been cleaned, it was time for the exploratory data analysis. To explore my data, I wanted to see how all the values in the data set correlated to each other, so I ran the code: df.corr(numeric\_only =True), which returned this table.

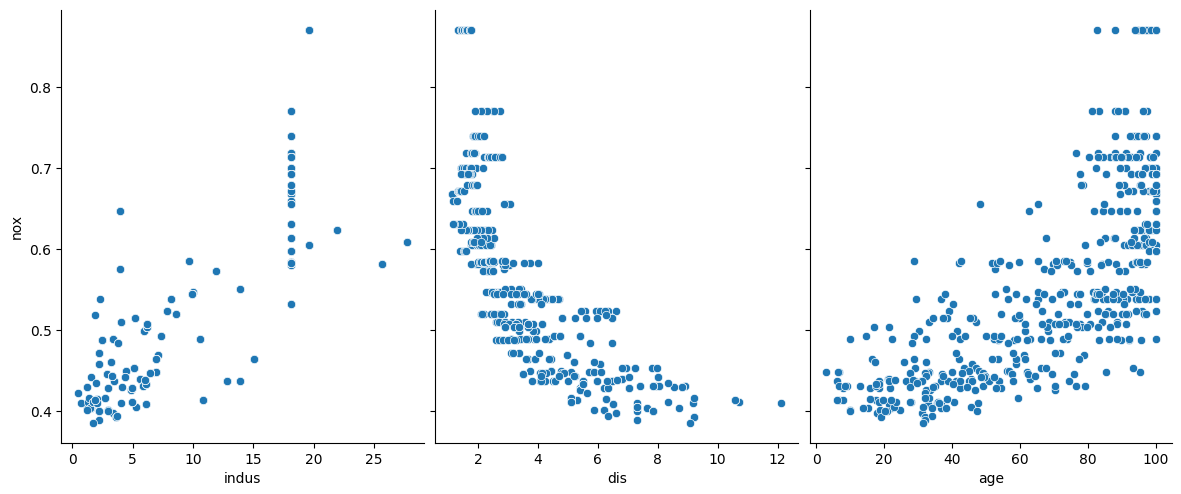
A screenshot of a computer

AI-generated content may be incorrect.



To analyze this table, I needed to refer back to my original question, “Can nitric oxide levels (NOX) be predicted by the proportion of non-retail business acres per town (INDUS)?” This not only gave me a starting point, but it helped focus my attention to the features that would potentially answer the question, ‘nox’ and ‘indus’. To understand this table you first need to know that a correlation is when one variable causes or is related to the change in another variable. For example, when one variable increases, the other variable increases; this is called a positive correlation. On the other hand, a negative correlation is when one variable increases, the other variable decreases. You also need to know that the closer a value is to 1, either positive or negative, the stronger the relationship between the two variables. Looking at the table where ‘nox’ and ‘indus’ meet, I noted that they have a correlation value of 0.766, which is actually pretty close to 1 as far as correlations go. I decided I needed to see what the correlation looked like, so I made a scatter plot with the df.plot.scatter('indus', 'nox') command. The plot was somewhat linear, but I wanted to view the two other variables that had a correlation above 0.7, ‘dis’ and ‘age’, alongside ‘indus’. I did this by accessing the Seaborn library using the command: sns.pairplot(df, x\_vars=['indus', 'dis', 'age'], y\_vars='nox',height=5, aspect=0.8, kind='scatter'), which allows you to see a pairplot, or a scatter plot with multiple ’X’ and/or ‘Y’ variables (vars in the code) at the same time. This is something I learned from my Statistical Programming class. Here is what the three scatter plots looked like side by side:





Upon seeing this, I realized that neither of the plots appeared to be as linear as I would have liked. The plot with ‘indus’ was sporadic except for the cluster in the bottom left and, while the ‘age’ plot was closer together, at a certain point the dots curved upward and were almost stacked on top of each other. The plot with the ‘dis’ variable showed the negative correlation which was clustered even closer together than the ‘age’ data but was stacked on top of each other closer to the y-axis. Since it can be difficult to know how linear some data is just by looking at a scatter plot, I decided to delve further into which variable would be best to predict the nitric oxide levels in the towns around Boston by creating linear regression models for each variable.

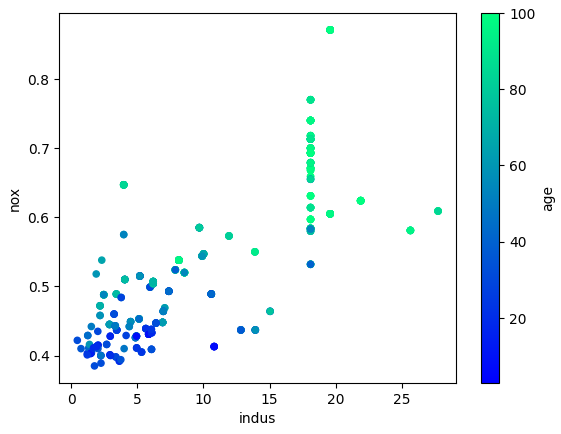
To start, I had to import a portion of the library Scikit-Learn using the command: from sklearn.linear\_model import LinearRegression. Then I needed to create, fit and run the models, make predictions, and finally plot the data and the regression line by running the following section of code for each of the three variables, ‘indus’, ‘dis’, and ‘age’.A computer screen shot of a program code

AI-generated content may be incorrect. The green text explains what is happening, while the orange text displays the names of the different variables. For each model I kept the ‘nox’ variable in place and substituted ‘age’ and ‘dis’ in where ’indus’ is shown. I chose to execute the model for age second since it was the next positive correlation. After running the models, I also ran the code: print('Correlation coefficient:', model.score(X\_train, y\_train)) to check the accuracy, again the closer to 1 the better. Here are the three models along with their correlation coefficients:A screen shot of a graph

AI-generated content may be incorrect. A screen shot of a graph

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.The red line on the plot represents the regression line, known as the line of best fit, and can also be represented by the equation y = mx + b where m = slope and b = y-intercept (or where the line crosses the Y axis). I ran the code to find the intercept: print("Intercept : ", model.intercept\_) and slope: print("Slope\_coef : ", model.coef\_) for each model. Looking at the first graph, ‘nox’ and ‘indus’, the correlation coefficient is 0.586, the intercept is 0.410 and the slope is 0.013, all rounded to 3 decimal places. The correlation coefficients for all three variables were barely above 0.5 which meant that the model would have technically worked, but only with a 50-60 percent accuracy rate. I knew there had to be something better, so I created a scatter plot using both ‘indus’ and ‘age’ using df.plot.scatter(x='indus', y='nox', c = 'age', colormap="winter"). This is set up like a typical scatter plot with ‘indus’ as the X variable and ‘nox’ as the Y variable, however, it adds the third variable, ‘age’, as a color indicator. With this I was able to see how the age of the homes, the nitric oxide concentration levels and the proportion of non-retail business acres per town related to one another. I decided I should look back at the correlation table, specifically the ‘nox’ column, this time in descending order so I can see which variables have the correlation values closest to 1. I ran the command: df.corr(numeric\_only= True).nox.sort\_values(ascending=False) to get the table. I took note of the values and that, with the models, convinced me it was time to rethink my original question.

**Refining the Question  
10 pts**

The Data Science Life Cycle is not an entirely linear process, often finding that in completing one step you must go back to the beginning to change your course before you can continue with any remaining steps. That doesn’t always mean starting over completely, it just means that the results need to be assessed to refine your focus. In this case, my exploratory analysis revealed that while I could have answered the question, it wouldn’t have been as accurate. To explain further, I need to refer to the correlation coefficients of the three linear regression models. The correlation coefficients are similar to the values in the original correlation table, except that these values are assessed after the model has been run using the data to predict the new values of Y. To be accurate, the correlation coefficient should be close to 1, however, the value from the ‘age’ model was barely above 0.5 or a 50% accuracy, while the ‘indus’ and ‘dis’ models showed almost 60% accuracy. Both the proportion of non-retail business acres per town (indus) and the weighted distances to five Boston employment centers(dis) are viable options to predict nitric oxide level fluctuation.

Upon evaluating the various scatter plots, the table of correlation values and the linear regression models, I decided that the relationship between ‘nox’ and ‘dis’ would be more accurate in predicting nitric oxide concentration levels. Therefore, I refined the original question from “Can nitric oxide levels (NOX) be predicted by the proportion of non-retail business acres per town (INDUS)?” to “Can nitric oxide concentration levels (NOX) be predicted by the weighted distances to five Boston employment centers (DIS)?” Now that I refined the question it was time to officially build the model.

**Model Building   
  
30 pts**

Although there are numerous types of models one can build based on the data they have, I chose to build a linear regression model. This model works best when there is an obvious positive or negative correlation to the data and the data follows a linear path when plotted. This specific linear regression model is a supervised learning model because it uses one variable to predict another, for example, using the distance away from business centers (dis) to determine nitric oxide levels (nox). Since I had already imported the LinearRegression function from the Scikit-Learn library, I ran the code shown below and got the subsequent scatter plot.

A screen shot of a computer program

AI-generated content may be incorrect. A screen shot of a graph

AI-generated content may be incorrect.

Upon creating the model, I had to train it, with X representing the values from the ‘dis’ column, ensuring to reshape the data so the array could be read correctly by the model, and Y representing the values from the ‘nox’ column. Then, I fit the model using the X and Y train variables to represent the data and made the ‘nox’ predictions based on the values of the ‘dis’ column again reshaping the data. Lastly, I made a scatter plot with the regression line using the predicted values from the model. I labeled my x and y axes appropriately and with the plt.show() command the scatter plot printed. After running the model, I printed the correlation coefficient along with the Y-intercept(b) and Slope(m) so I could find the equation for the regression line Y = mx + b. The correlation coefficient showed that the regression model is 59% accurate which is not as high as I would have liked but is still the best option with the data given. Using the slope and intercept, rounded to 4 decimal points, I found the equation for the regression line: Y = -0.0423(X) + 0.7156. With this equation I can input any value as X, again representing the distance from 5 Boston business centers, and get a value for Y, the predicted nitric oxide levels. I demonstrated this concept by creating a new data set for ‘dis’ and predicting ‘nox’ using the model, showing the printed predictions. To interpret, the new data set consists of the values [6, 3, 5, 9] with their corresponding predictions along the bottom. For example, when 6 was input for X, the predicted Y value was 0.4615 and as an equation that would look like Y = -0.0423 \* 6 + 0.7156. The equation can also be performed by executing the code: model.intercept\_ + (6\*model.coef\_) where the slope and X value are put in parentheses to ensure they are multiplied, and the Y-intercept is put first since you can’t start code with a parenthesis.

A screenshot of a computer program

AI-generated content may be incorrect. A screen shot of a computer code

AI-generated content may be incorrect.

**Interpretation/Summary  
10 pts**

What does the data say? Can nitric oxide concentration levels (NOX) be predicted by the weighted distances to five Boston employment centers (DIS)? The simple answer is yes, which should make sense because the closer a town is to business areas the more emissions from cars and businesses leading to higher nitric oxide concentrations. However, the data shows that the model’s predictions are only 59% accurate, which means that 41% of the time the predictions will be off. That number is way too large for me so, while the model says that nitric oxide levels can be predicted, the accuracy level is not strong enough for me to recommend this model. A more accurate model may be possible if multiple variables are considered, such as the ones with the highest correlation values, but logic would say that those variables are arbitrary and would not lead to any sort of causation which is why I did not continue and make a more in-depth model.