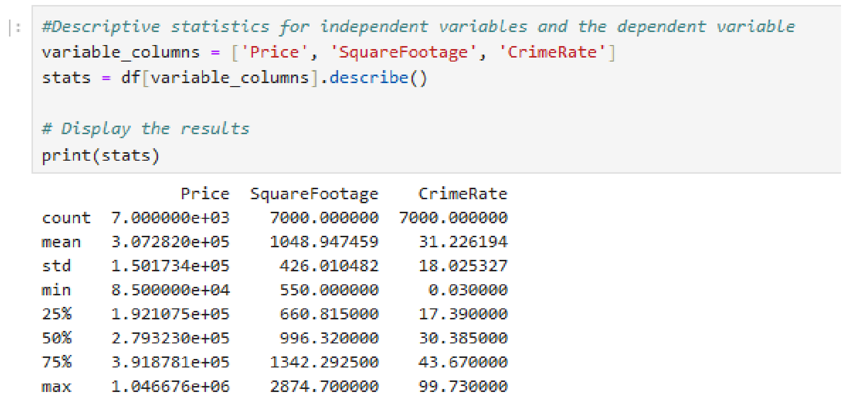
D600 Task 1
By Eric Williams

A:GITLAB REPOSITORY
Link provided for the gitlab repository was provided in the link submission.

B1:PROPOSAL OF QUESTION
We are required to use three variables for this analysis, so I've chosen my question to be about how well two variables predict a third variable. My research question is: Do square footage and crime rate affect the value of a home, and if so, how much? How much does increasing a square foot increase value, and how much does decreasing crime rate increase value?

B2:DEFINED GOAL
The goal of this data analysis is to quantify how much square footage and crime affect the cost of home prices. If I were doing this analysis for a real estate company, a multiple linear regression would help provide an estimate for the value of currently unlisted houses or for future houses. Being able to estimate a house's value before building it will help the company make better investment decisions and optimize use of land.

C1:VARIABLE IDENTIFICATION
In my regression, the independent variables are square footage and crime rate. The dependent variable is the price of a house. In this analysis, I expect that when we change square footage and crime rate, price will be affected (meaning it is dependent on the other two variables).

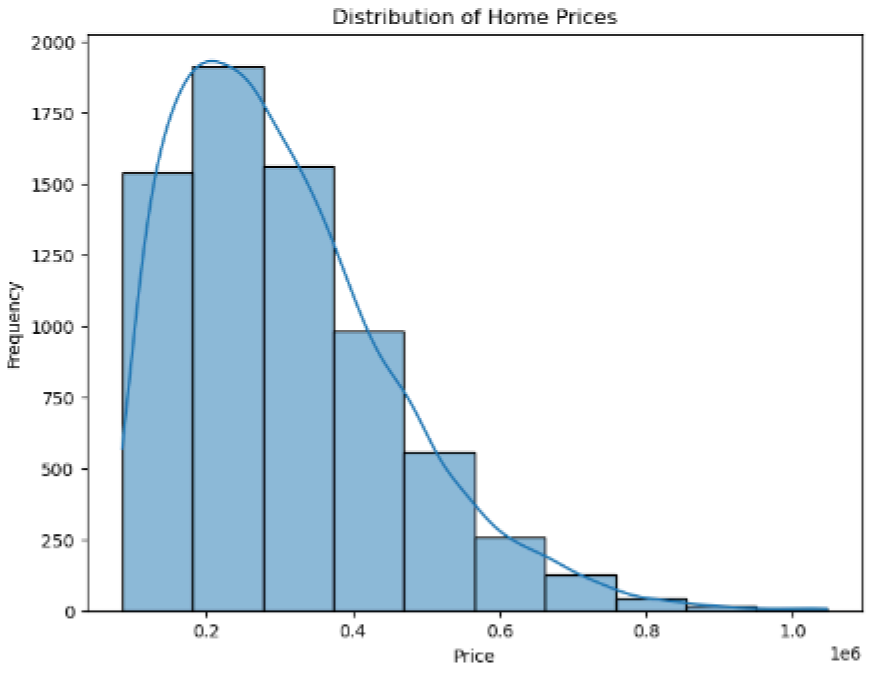
C2:DESCRIPTIVE STATISTICS


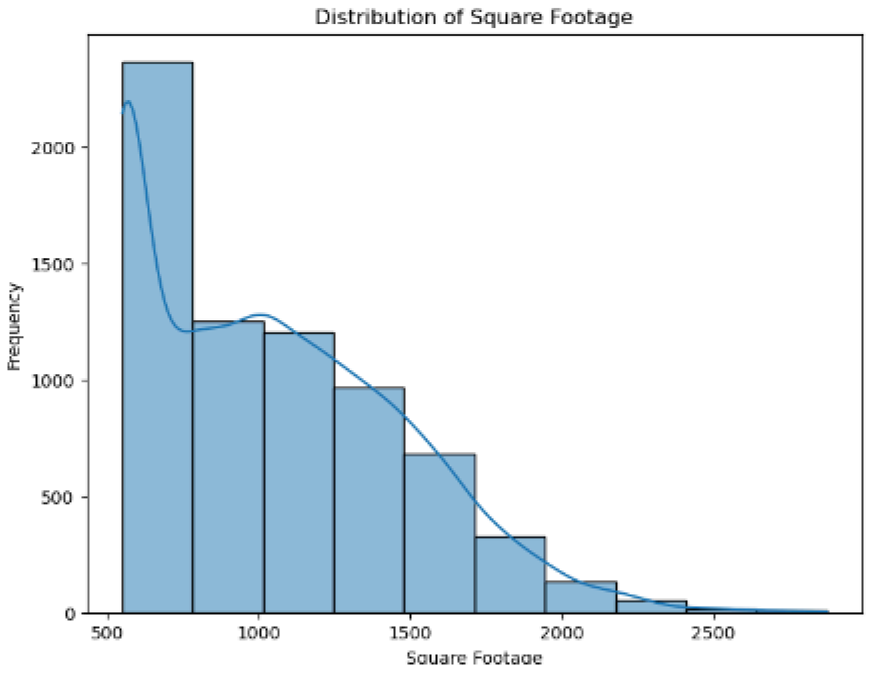
```
#Descriptive statistics for independent variables and the dependent variable
variable_columns = ['Price', 'SquareFootage', 'CrimeRate']
stats = df[variable_columns].describe()

# Display the results
print(stats)
```

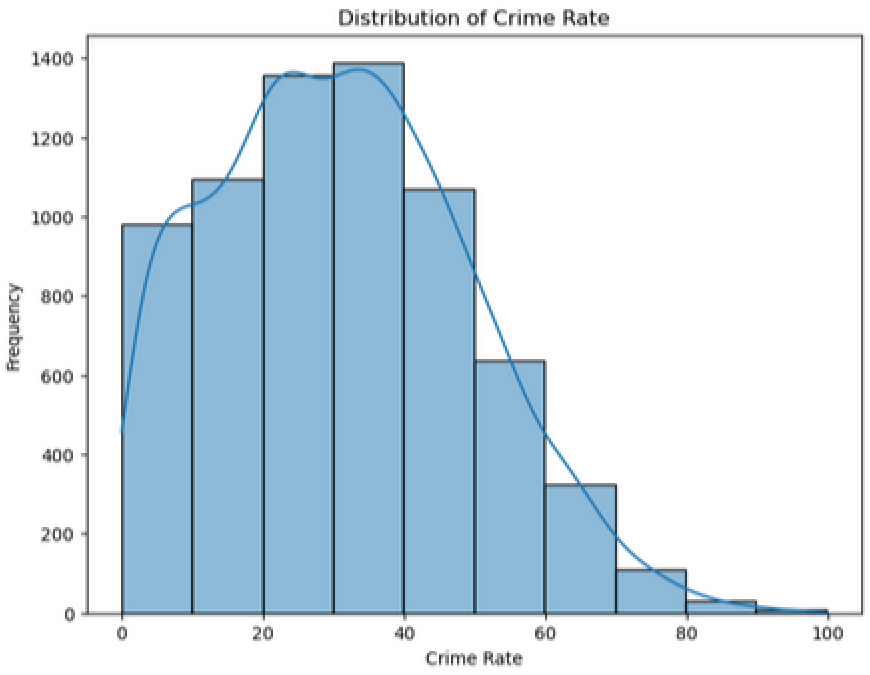
```
              Price  SquareFootage    CrimeRate
count  7.000000e+03    7000.000000  7000.000000
mean   3.072820e+05    1048.947459    31.226194
std    1.501734e+05     426.010482    18.025327
min    8.500000e+04     550.000000     0.030000
25%    1.921075e+05     660.815000    17.390000
50%    2.793230e+05     996.320000    30.385000
75%    3.918781e+05    1342.292500    43.670000
max    1.046676e+06    2874.700000    99.730000
```

C3:VISUALIZATIONS

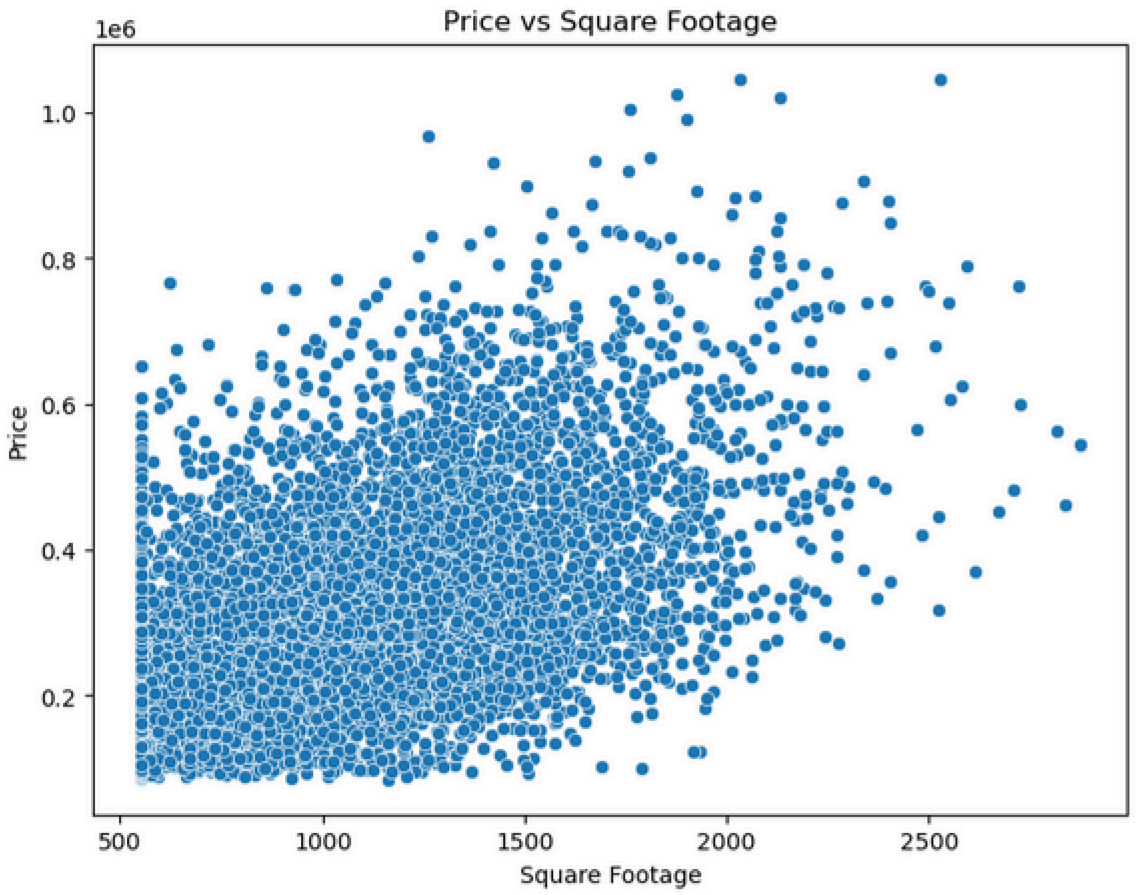
Here are the univariate visualizations of each variable (Price, Square Footage, and Crime Rate):

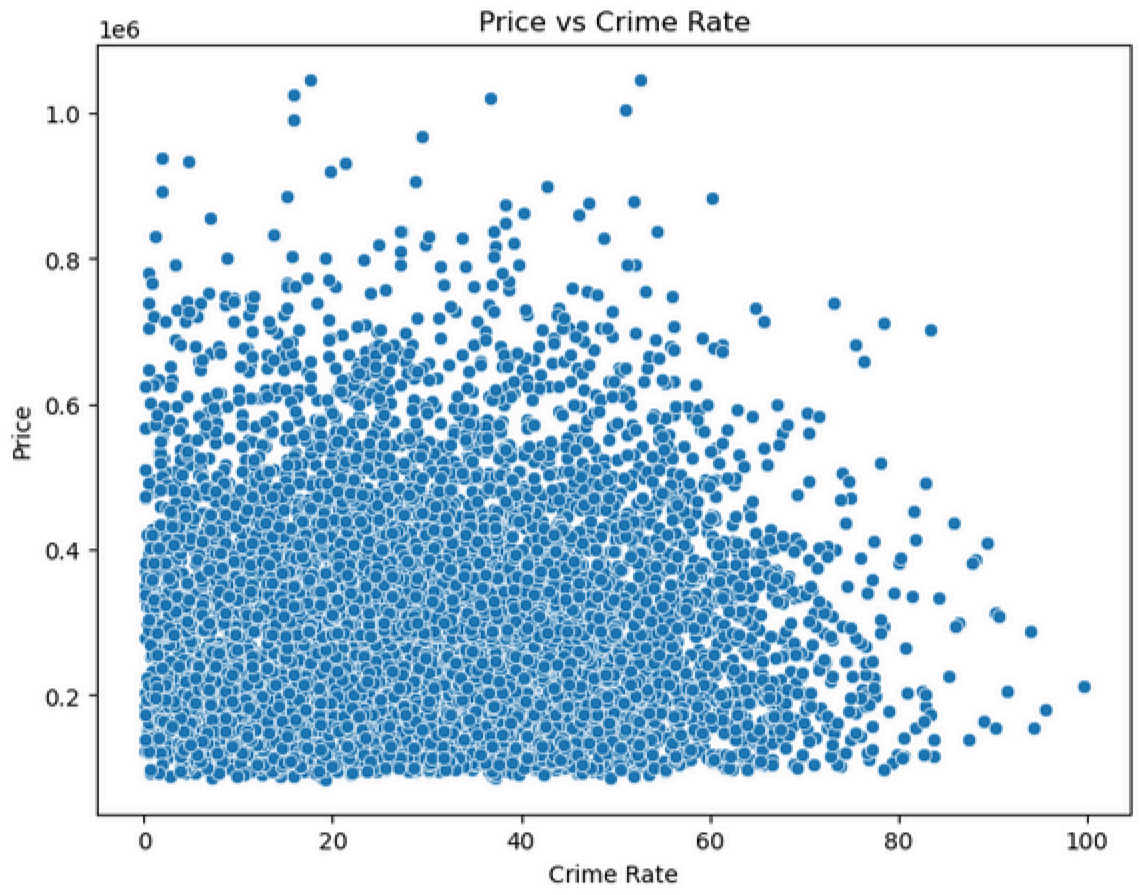






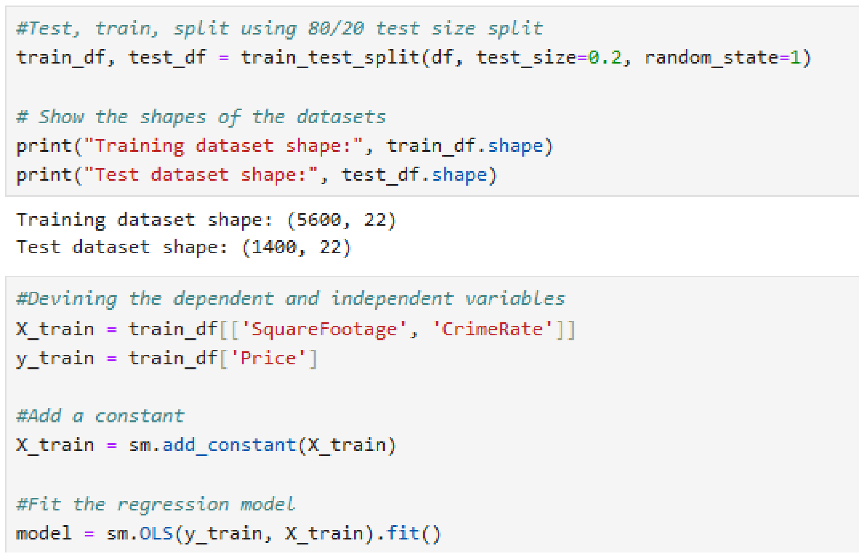
Here are the bivariate visualizations variable of Price against Square Footage and Crime Rate:





D1:SPLITTING THE DATA

This is the code I used to split the data. Note the 80/20 split, with the larger percentage assigned to training and the smaller percentage assigned to test the data.



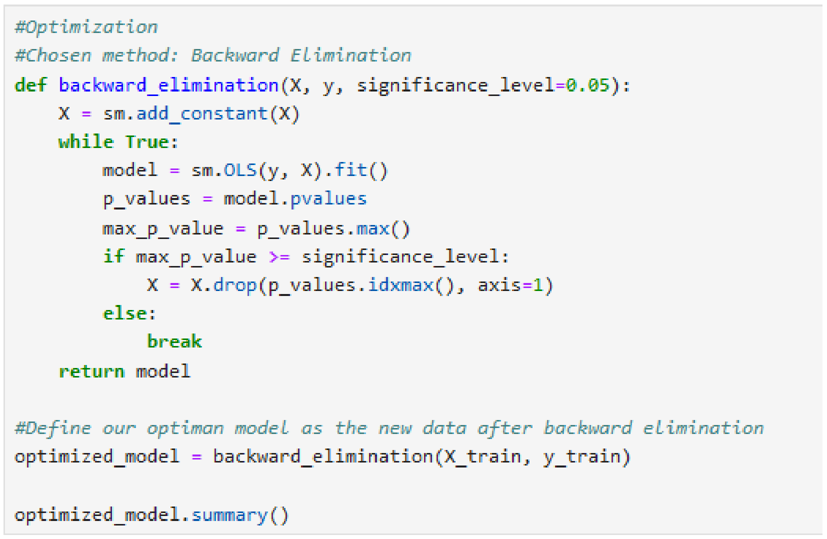
```
#Test, train, split using 80/20 test size split
train_df, test_df = train_test_split(df, test_size=0.2, random_state=1)

# Show the shapes of the datasets
print("Training dataset shape:", train_df.shape)
print("Test dataset shape:", test_df.shape)
```

```
Training dataset shape: (5600, 22)
Test dataset shape: (1400, 22)
```

```
#Devining the dependent and independent variables
X_train = train_df[['SquareFootage', 'CrimeRate']]
y_train = train_df['Price']

#Add a constant
X_train = sm.add_constant(X_train)

#Fit the regression model
model = sm.OLS(y_train, X_train).fit()
```

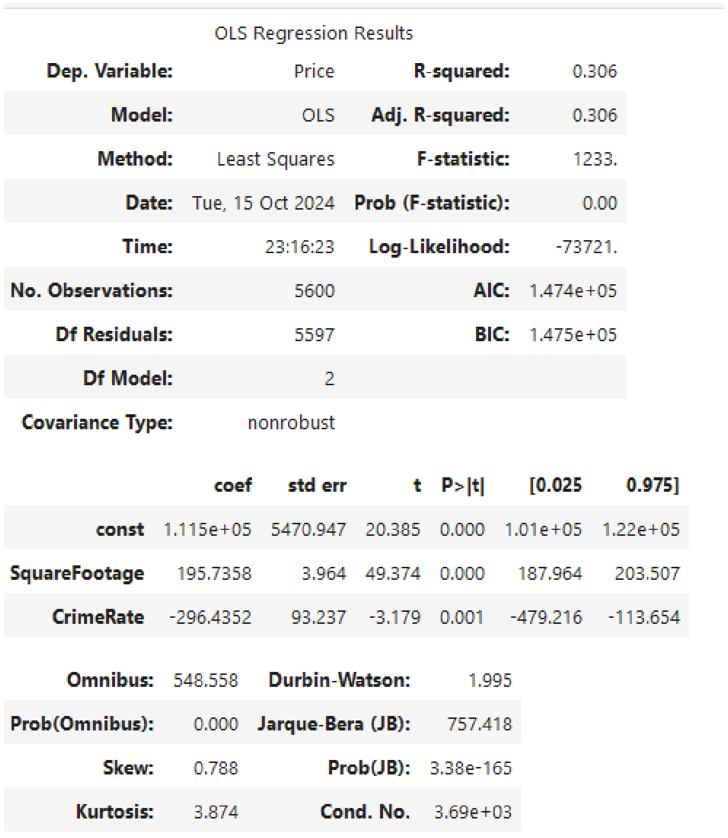
D2:MODEL OPTIMIZATION

Here is the code I used to optimize my model using the backward elimination method.



```python
#Optimization
#Chosen method: Backward Elimination
def backward_elimination(X, y, significance_level=0.05):
    X = sm.add_constant(X)
    while True:
        model = sm.OLS(y, X).fit()
        p_values = model.pvalues
        max_p_value = p_values.max()
        if max_p_value >= significance_level:
            X = X.drop(p_values.idxmax(), axis=1)
        else:
            break
    return model

#Define our optiman model as the new data after backward elimination
optimized_model = backward_elimination(X_train, y_train)

optimized_model.summary()
```

Here are the results after the optimization. Note that adjusted R2, R2, F statistics, probability F statistics, coefficient estimates, and p-value of each independent variable are included:



OLS Regression Results

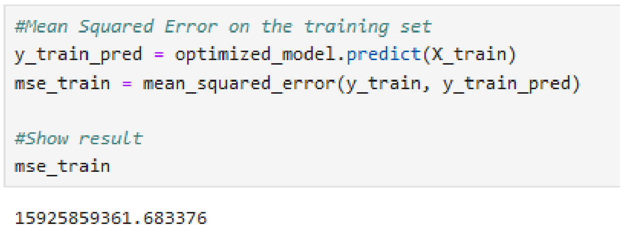
Dep. Variable:	Price	**R-squared:**	0.306
Model:	OLS	**Adj. R-squared:**	0.306
Method:	Least Squares	**F-statistic:**	1233.
Date:	Tue, 15 Oct 2024	**Prob (F-statistic):**	0.00
Time:	23:16:23	**Log-Likelihood:**	-73721.
No. Observations:	5600	**AIC:**	1.474e+05
Df Residuals:	5597	**BIC:**	1.475e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	1.115e+05	5470.947	20.385	0.000	1.01e+05	1.22e+05
SquareFootage	195.7358	3.964	49.374	0.000	187.964	203.507
CrimeRate	-296.4352	93.237	-3.179	0.001	-479.216	-113.654

Omnibus:	548.558	**Durbin-Watson:**	1.995
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	757.418
Skew:	0.788	**Prob(JB):**	3.38e-165
Kurtosis:	3.874	**Cond. No.**	3.69e+03

<u>D3:MEAN SQUARED ERROR</u> and **<u>D4:MODEL ACCURACY</u>**

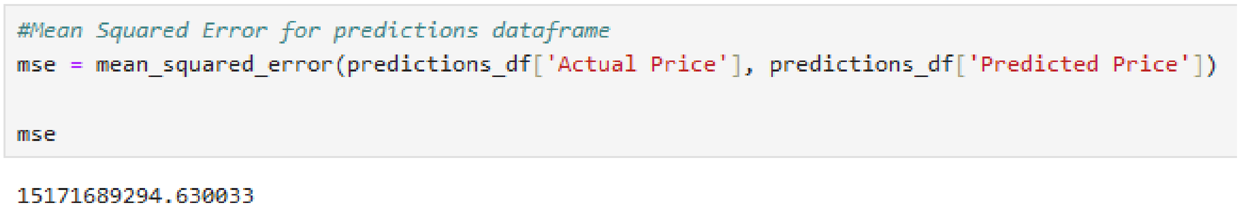
Here is the code and the results of the mean squared error BEFORE optimization:



```
#Mean Squared Error on the training set
y_train_pred = optimized_model.predict(X_train)
mse_train = mean_squared_error(y_train, y_train_pred)

#Show result
mse_train
```

```
15925859361.683376
```

And here is the result AFTER optimization:



```
#Mean Squared Error for predictions dataframe
mse = mean_squared_error(predictions_df['Actual Price'], predictions_df['Predicted Price'])

mse
```

```
15171689294.630033
```

If we take the square root of these values, that means the original model was off by an average of $126,197.69. After optimization, the average estimate was off by $123,173.41. That means the backwards elimination method improved our model by $3,024.28.

Below, I will discuss these numbers in further detail.

<u>E1:PACKAGES OR LIBRARIES LIST</u>

Here is a list of packages I imported and why they were essential:

- Pandas: useful for making dataframes to store the data in
- Seaborn: helpful for data visualizations, such as the scatterplots i made above
- Matplotlib: essential for visually plotting the univariate and bivariate statistics
- Sklearn: essential for the specific tools I imported, namely mean squared error, linear regression, and train test split

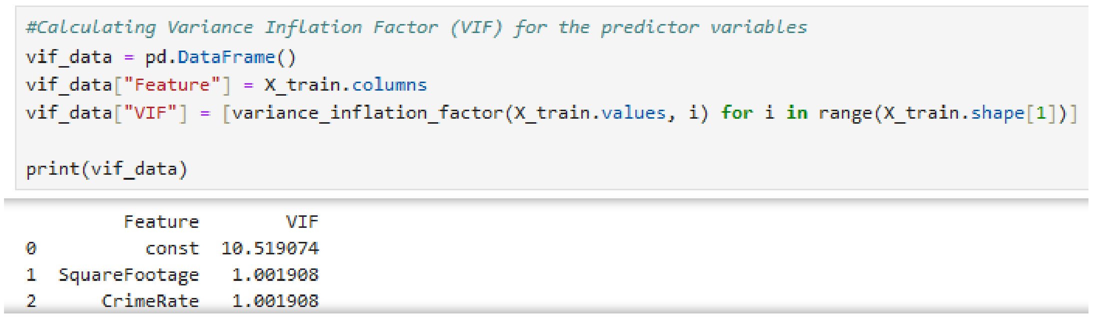
<u>E2:METHOD JUSTIFICATION</u>

For optimization, I chose backward elimination. The goal of backward elimination is to remove the less significant variables in a series of steps. Eventually, this will leave us with just the variables that are the most important since the less helpful predictors are eliminated. The effectiveness of backward elimination of an optimization model was shown above when comparing the mean squared error before and after optimization. It was helpful in reducing the error in our model.

E3:VERIFICATION OF ASSUMPTIONS

However, backward elimination requires that a) models do not include collinear sets of data for the predicting columns and b) that the data set is an adequate size.

To verify these assumptions, we need to make sure the dataset is not too small and that there is no multicollinearity between our predictor variables, Square Footage and Crime rate. I ran a variance inflation factor on the two columns to check for collinearity:



```
#Calculating Variance Inflation Factor (VIF) for the predictor variables
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]

print(vif_data)
```

```
         Feature        VIF
0          const  10.519074
1  SquareFootage   1.001908
2      CrimeRate   1.001908
```

Because the VIF is very close to 1 for both variables, this indicates that SquareFootage and CrimeRate are not multicollinear. Also because our data set is 7000 rows and is much larger than the standard 500 rows required, we can verify our assumptions that a) the variables are not collinear and b) the data is an adequate size.

E4:EQUATION

Here is the multiple predictor model equation for two variables is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

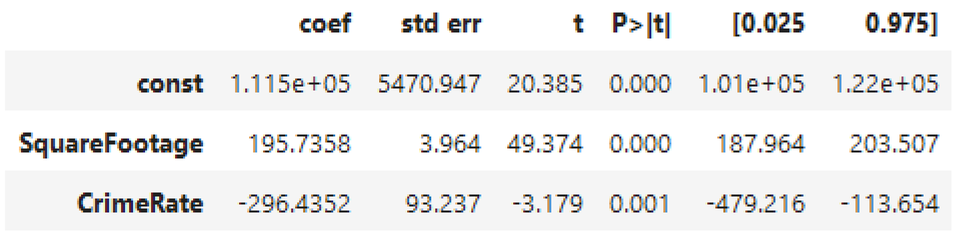
In this case, the outcome value Y is the price.

β_0 is the intercept when the independent variables are 0.

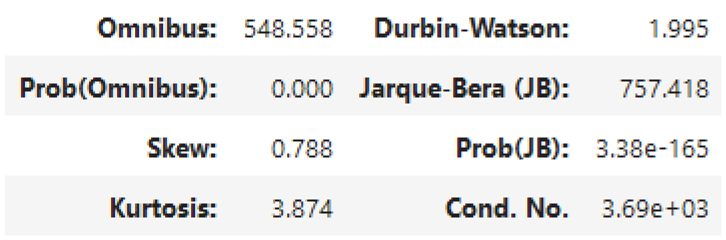
β_1 is the coefficient for square footage.

β_2 is the coefficient for crime rate.

ϵ is the error in the model. Here are the results given by the model:



	coef	std err	t	P>|t|	[0.025	0.975]
const	1.115e+05	5470.947	20.385	0.000	1.01e+05	1.22e+05
SquareFootage	195.7358	3.964	49.374	0.000	187.964	203.507
CrimeRate	-296.4352	93.237	-3.179	0.001	-479.216	-113.654



Omnibus:	548.558	**Durbin-Watson:**	1.995
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	757.418
Skew:	0.788	**Prob(JB):**	3.38e-165
Kurtosis:	3.874	**Cond. No.**	3.69e+03

The constant coefficient β_0= 111,500, which means when crime rate and square footage are 0 the value of a home is $115,00.

The square footage coefficient is $\beta_1 = 195.73$, which means when a square foot is added to a home, the house value increases by $195.73.

The crime rate coefficient $\beta_2 = -296.45$ means that for every unit increase in the crime rate, the value of the house goes down by $296.45.

The error ϵ = $122,000

Thus our final equation is
$Price = 111,500 + 195.73\, x\, (Square\ Footage) - 296.43\, x\, (Crime\ Rate) + 122,000$

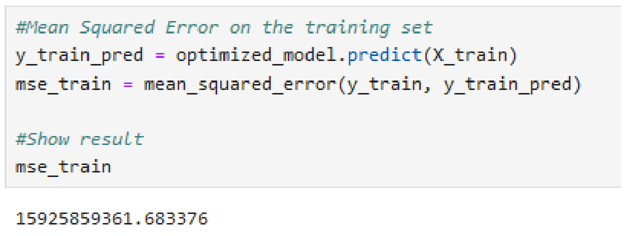
<u>E5:MODEL METRICS</u>

As given in the results above:
R-squared = 0.306
Adjusted R-squared = 0.306

R-squared generally represents how much variance the Price had in our model that can be explained by the independent variables (Square Footage and Crime Rate). A score of 0.306 means that 30.6% of the variance we see in price can be explained by our model. This relatively low score means that our independent variables do trend with price, however, there are clearly more factors that determine the other 70% of the variance. The adjusted R-squared score was exactly the same as the R-squared score, which is to be expected with only two independent variables.

A comparison of the MSE for the training set to the MSE of the test set was given above, but will be summarized again here:

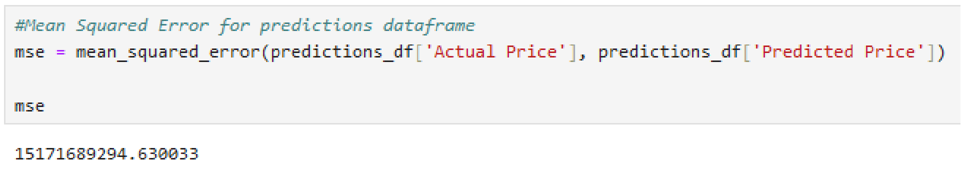
Here is the code and the results of the mean squared error BEFORE optimization:



```
#Mean Squared Error on the training set
y_train_pred = optimized_model.predict(X_train)
mse_train = mean_squared_error(y_train, y_train_pred)

#Show result
mse_train
```

```
15925859361.683376
```

And here is the result AFTER optimization:



```
#Mean Squared Error for predictions dataframe
mse = mean_squared_error(predictions_df['Actual Price'], predictions_df['Predicted Price'])

mse
```

```
15171689294.630033
```

If we take the square root of these values, that means the original model was off by an average of $126,197.69. After optimization, the average estimate was off by $123,173.41. That means the backwards elimination method improved our model by $3,024.28.

<u>**E6:RESULTS AND IMPLICATIONS**</u> and <u>**E7:COURSE OF ACTION**</u>

Our results tell us that increased square footage increases value in a home, while increased crime rate decreases value in a home. However, our model was off in predicting price by $123,173.41. This is a very large number, but this makes sense when considering our R-squared value. According to our R-squared value, square footage and crime rate only account for about 30% of variance in home price. This does not mean our model is incorrect, but it does mean it could be more complete by adding more variables.

Although we can take away the fact that these two factors affect home price, we have to acknowledge that these two variables do not tell the whole story. The course of action I would recommend would be another Multiple Linear Regression model run with more variables to better predict the value of a home. With the results we do have, however, I would recommend to a builder to create more homes in low crime rate areas with larger square footage in order to create houses with more value.

Sources

No sources were used except for official WGU course materials.