Diamonds Dataset

The Diamonds dataset contains information pertaining to the physical properties of diamonds and their prices. There are 53,940 rows and 11 columns in the dataset. One column is simply an unnamed index column and was dropped for analysis. Carats, depth, table, price, and dimensions are numeric features. Diamond depth and table are percentages describing the diamond's height and length of flat face relative to the overall width. Cut, color, and clarity are all categorical features. Cut is described as *Ideal, Premium, Very Good, Good*, or *Fair*. Color is indicated by a single letter that captures degrees of color in the range from D (colorless) to J (near colorless) (GIA, 2025a). Clarity can be described by any of the following codes: *IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1*. These ratings refer to degree of inclusions, or imperfections in the stone (GIA, 2025b). *IF* corresponds to "internally flawless," while *I1* corresponds to "included." One-hot encoding was used to transform these qualitative variables for cut, color, and clarity into categorical features with 0/1 values indicating absence/presence of the feature for each sample.

No null values were flagged, however, there are 20 rows that contain a value of 0 for at least one of the x, y, or z dimensions. Given that the entire dataset has more than 53,000 entries, these rows with missing dimensional data were simply dropped since such a small number of data points is unlikely to affect the overall analysis.

The statistics for the continuous variables were obtained before and after cleaning the data. The below table displays the basic statistics calculated for the cleaned dataset.

Feature	Count	Mean	Standard Deviation	Minimum	Median	Maximum
Carat	53920	0.80	0.47	0.20	0.70	5.01
Depth	53920	61.7	1.4	43.0	61.8	79.0
Table	53920	57.5	2.2	43.0	57.0	95.0
Price	53920	3931	3987	326	2401	18823
X	53920	5.73	1.12	3.73	5.70	10.74
Y	53920	5.73	1.14	3.68	5.71	58.90
Z	53920	3.54	0.70	1.07	3.53	31.80

There is clearly a lot of variability in price. The distribution of price is shown in Figure 1 below. Although the distribution is heavily skewed, it does not look like the samples in the upper price range should be considered outliers. These data points were retained.

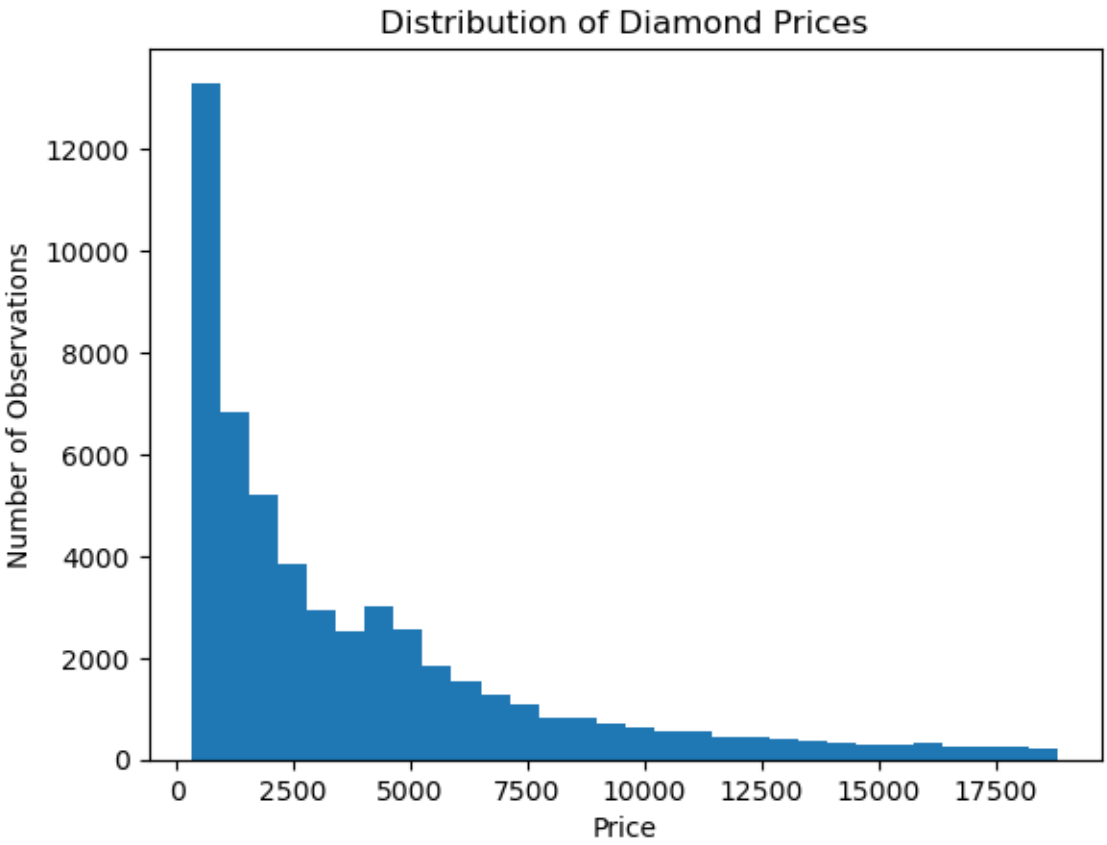

Figure 1

The remaining numeric features include some variables that would benefit from additional pre-processing to remove outliers. Based on the boxplots in Figure 2, it seems reasonable to remove data points with:

- Depth values less than 50 or greater than 75;
- Table values greater than 80;
- Y dimension values greater than 20; and
- Z dimension values greater than 10.

These points can be considered “extreme outliers.” There were 10 additional rows of data removed by excising these extreme outliers, which is a very small number compared to the overall size of the dataset. Technically, there are still outliers present as shown by the dots on the box plot, but there are so many of these data points that it does not make sense to blindly use the interquartile range cutoff to remove all of these data points. The updated box plots are displayed in Figure 3.

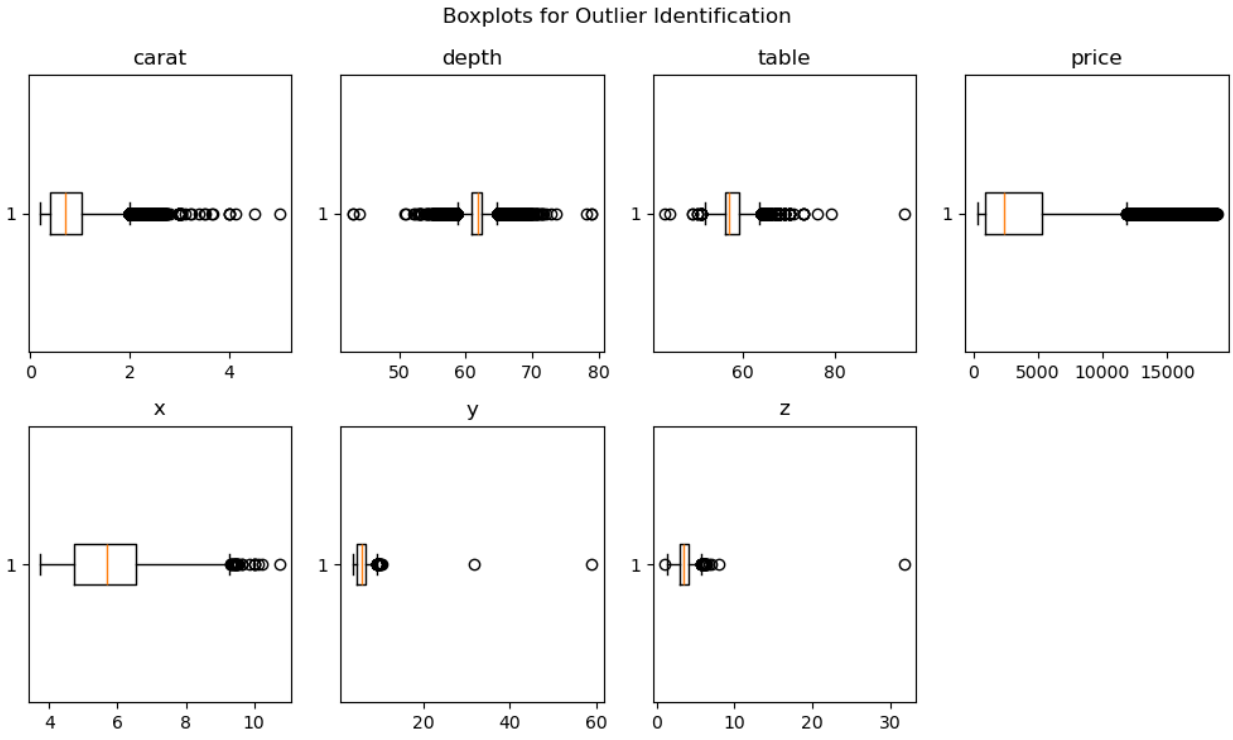

Figure 2

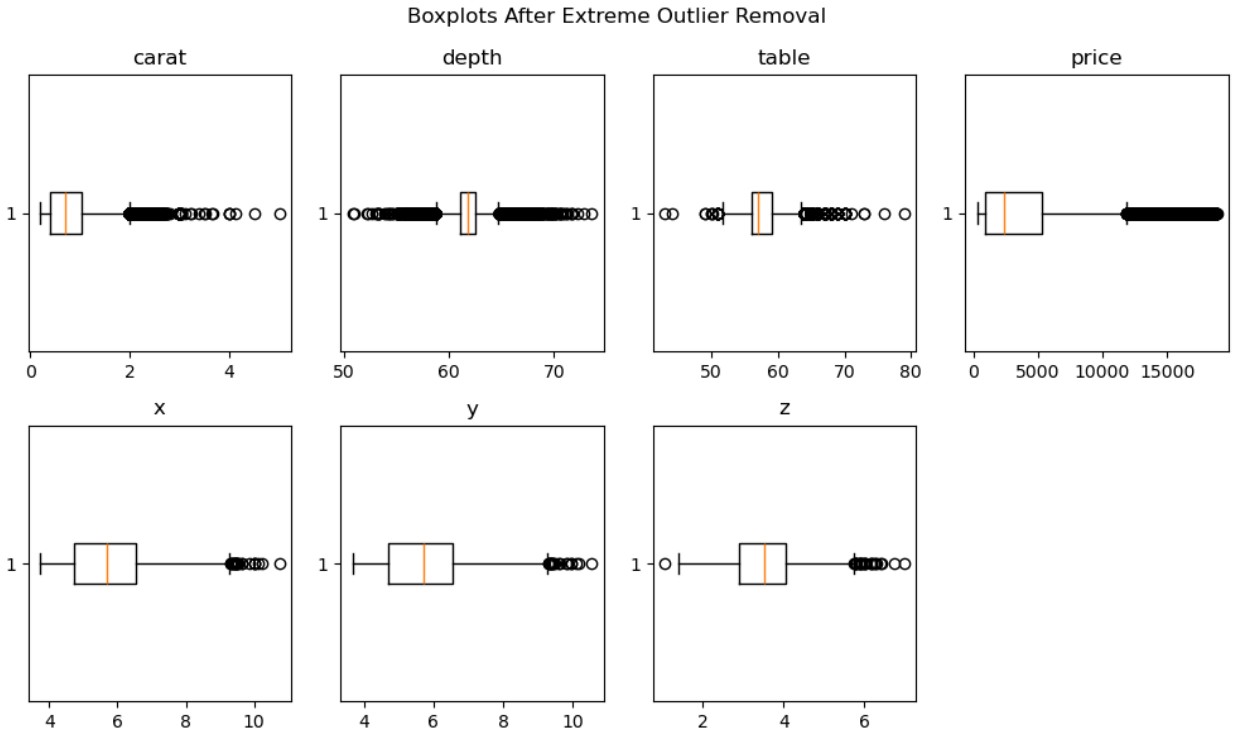

Figure 3

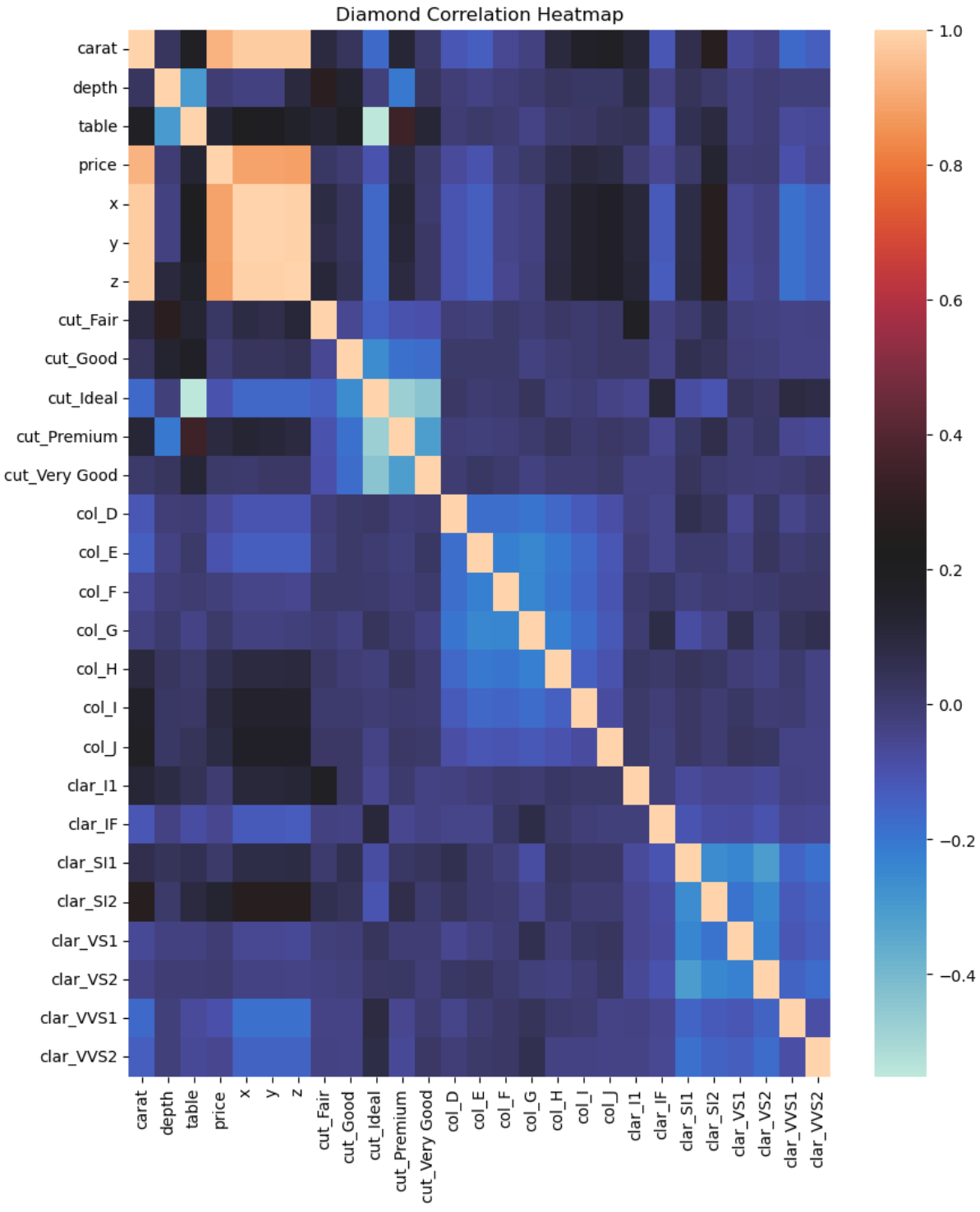

Figure 4

The correlation matrix for the cleaned dataset is shown above in Figure 4. Correlation coefficients are generally low, however there are rather strong positive correlations between price, carats, and x-, y-, and z-dimensions. There are weak negative correlations between carat and some of the cut, color, and clarity variables.

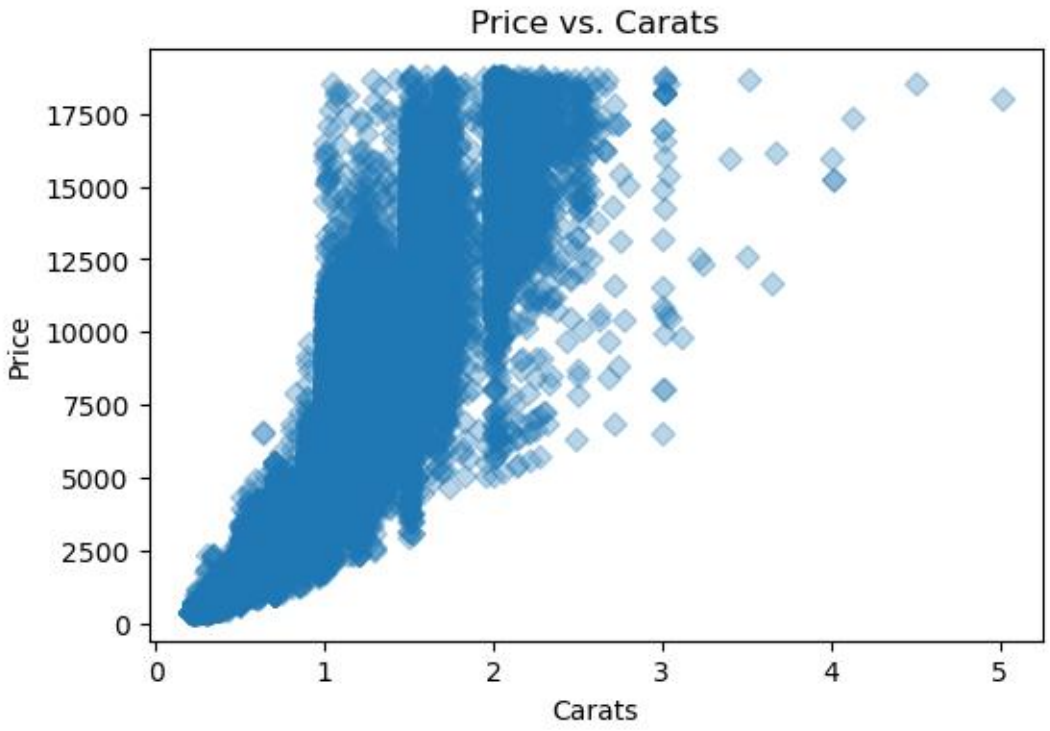

Figure 5

The general relationship between price and number of carats is captured visually by Figure 5. As expected, diamonds with more carats are generally more expensive. There tends to be more variation in price as the number of carats increases beyond 1.

Figure 6 shows the distribution of diamonds with each color code and their relative mean prices. Most diamonds in the dataset have a color classification of *G*, which is the best ranking in the "near colorless" category (GIA 2025a). The fewest number of diamonds are classified as *J*, which corresponds to the most color in the "near colorless" category. Surprisingly, these diamonds have the highest mean price. Diamonds with a color of *E*, which would traditionally be considered the second-best color after *D*, have the lowest mean price. Figure 7 helps to clarify this phenomenon. Diamonds at the higher end of the color rating scale have lower mean carats, while *H*, *I*, and *J* diamonds are sold at higher carats. As shown previously in Figures 4 and 5, there is a strong relationship between price and carats, which explains why the relative mean prices appear counterintuitive when only considering diamond color.

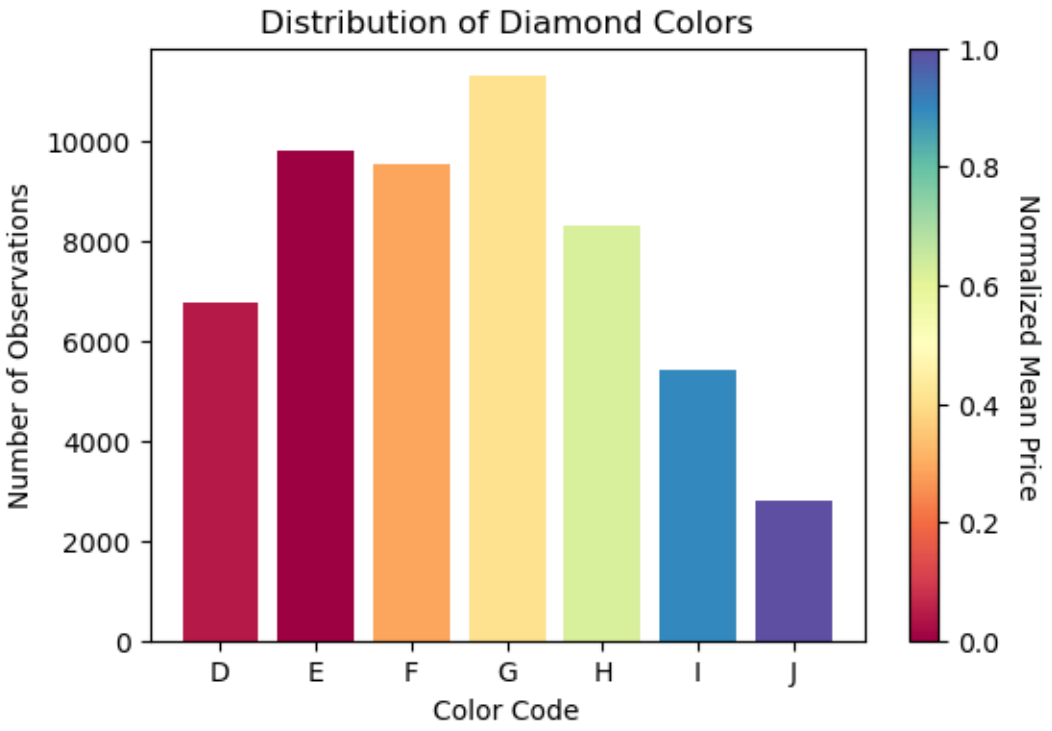

Figure 6

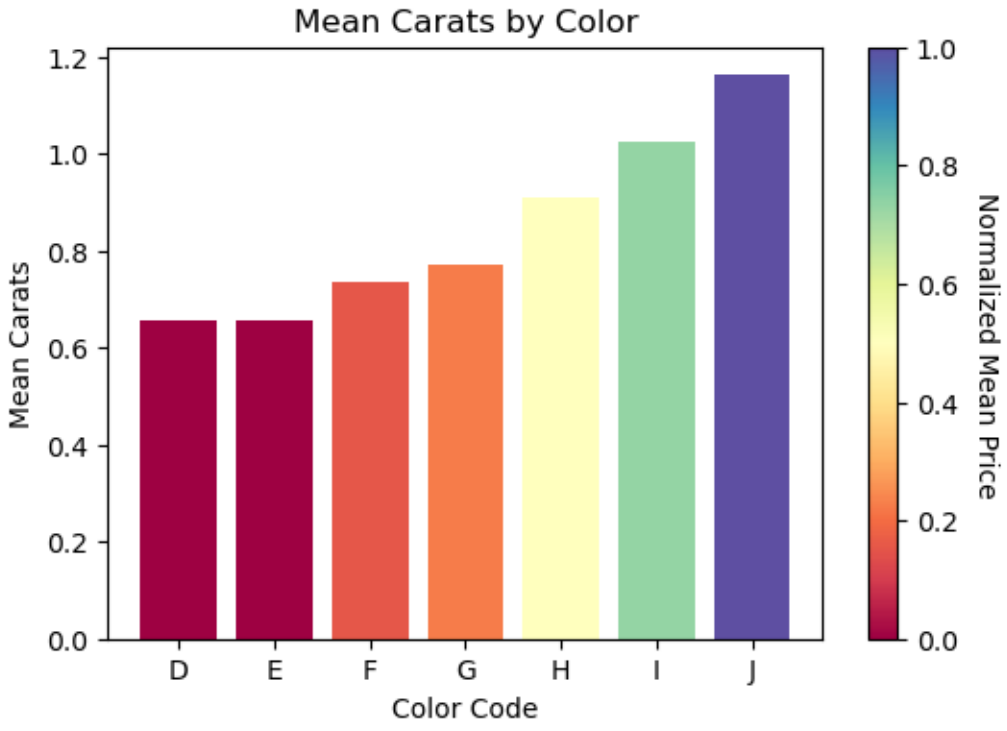

Figure 7

Linear Regression

The pre-processed, normalized data were split into test and training sets. Price is the target (y); the remaining normalized features comprise the input matrix (X). A linear regression model was fit to the training data using the ordinary least squares (OLS) method to minimize squared loss. The resulting weight vector (labeled with corresponding features) is shown in the table below.

Feature	**Weight**
Bias	-2162.8306
Carat	56005.5457
Depth	1913.5117
Table	-848.2743
x	-10095.0258
y	12483.8582
z	-15014.7133
cut_Good	502.4065
cut_Ideal	778.1789
cut_Premium	746.3637
cut_Very Good	645.1481
col_E	-220.9213
col_F	-268.5410
col_G	-484.4009
col_H	-989.8122
col_I	-1478.5149
col_J	-2413.2108
clar_IF	5233.9810
clar_SI1	3571.8405
clar_SI2	2620.6093
clar_VS1	4477.5018
clar_VS2	4177.5641
clar_VVS1	4900.0057
clar_VVS2	4848.5938

Predictions were generated for both the test and training sets using these weights. The root mean squared error (RMSE) for the training data was 1121.4385 and the RMSE for the test data was calculated to be 1132.9440. The model performed slightly better on the training data. Figure 8 shows the actual and predicted price for each data point. There appears to be larger deviation between the predictions and truth at larger prices. However, many predictions at lower prices are negative, which is not realistic.

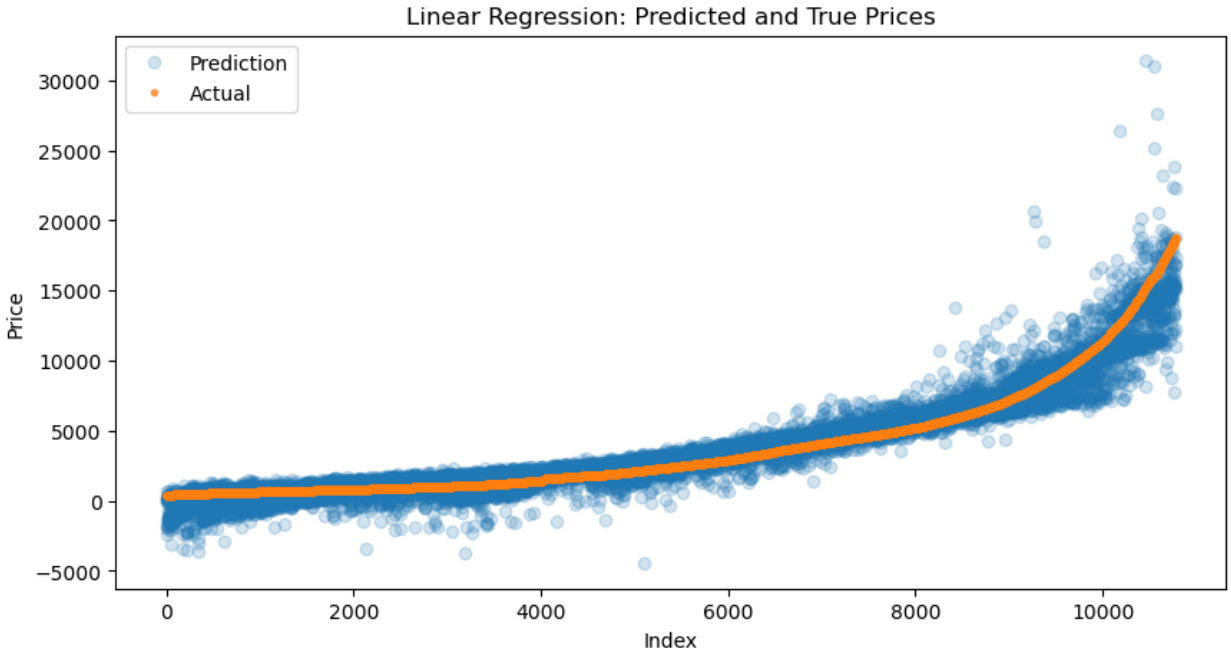

Figure 8

The OLS estimate for computing weights is straightforward to carry out. Due to its convex loss function, OLS is guaranteed to converge at the global minimum. In other words, we can be confident that the weights computed by the model represent the best possible result given the inputs; the model has not gotten stuck in any suboptimal local minima. Another advantage of OLS is that it results in a highly interpretable model. Based on the weights, we can conclude that more carats, a larger weight, and better ratings for cut, color, and clarity will likely substantially drive up the price of a diamond. This information is consistent with domain knowledge and general intuition.

Notwithstanding its benefits, using OLS to compute weights does have a few drawbacks. The method is inherently highly susceptible to outliers. While this issue can be mitigated somewhat by adequate preprocessing, other methods are able to deal with outliers more effectively. Linear regression via the OLS estimate is also heavily impacted by correlations between features as well as unnecessary features. The presence of one or both of these phenomena may lead to poor model performance. Regularized regression methods like ridge or lasso can more efficiently reduce or eliminate the influences of correlated attributes and superfluous features.

Ridge Regression

Ridge regression was applied to the same test and training sets that were created for the linear regression. After tuning the model, the training RMSE was 1122.5921 and the test RMSE was 1131.6520. The test error represents a marginal improvement from linear regression, but the performance is basically equivalent and the weights are very similar. The estimated weight vector along with its corresponding features is shown in the table below.

Feature	**Weight**
Bias	-2767.7848
Carat	54075.3550
Depth	833.5761
Table	-871.4798
x	-7706.7137
y	7581.3869
z	-9864.7690
cut_Good	542.7100
cut_Ideal	810.4011
cut_Premium	763.2042
cut_Very Good	690.8176
col_E	-220.1704
col_F	-271.3744
col_G	-484.1576
col_H	-984.6760
col_I	-1463.1231
col_J	-2392.3698
clar_IF	5219.7550
clar_SI1	3538.3011
clar_SI2	2591.2914
clar_VS1	4448.7432
clar_VS2	4148.1592
clar_VVS1	4882.6980
clar_VVS2	4827.9899

Figure 9 displays the actual data and the predictions resulting from ridge regression. This plot strongly resembles the one from linear regression, though slight differences are observed. Again, there are unrealistic negative predictions at the lower end and relatively wider variation toward the higher end of the actual price range.

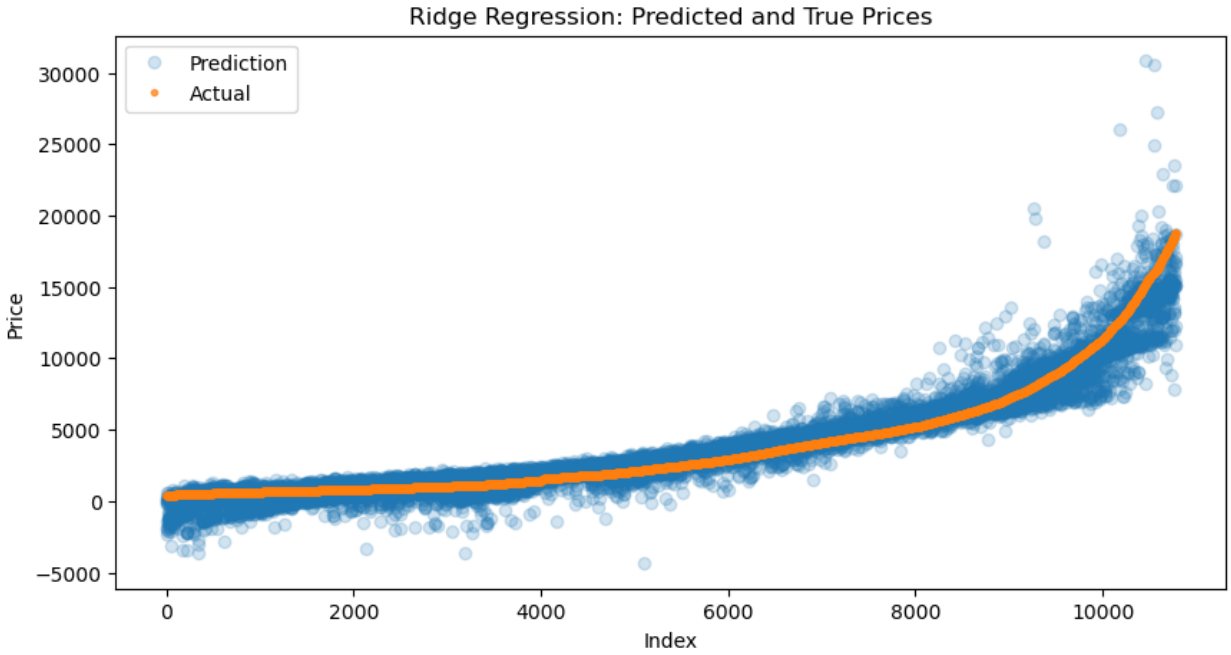

Figure 9

In general, ridge regression has the effect of decreasing variance, which can help to make the model more generalizable. Ridge regression uses the same loss function as linear regression, but also adds a penalty term to prevent the weight values from growing too large. This penalty term makes ridge regression a form of l2 regularization, which helps to reduce the impact of correlated and useless features in the dataset. Ridge regression tends to shrink the weights of correlated features proportionally. For irrelevant features, ridge regression can shrink their weights close to (but not equal to) zero, thereby reducing their influence. As discussed previously, linear regression cannot effectively deal with correlated and useless features. One drawback of ridge regression, however, is that the penalty term contains a scalar parameter that must be tuned. Therefore, the implementation of ridge regression requires a little more effort compared to linear regression.

For the Diamonds dataset specifically, the weights computed by ridge regression were generally similar to those from linear regression. About three quarters of the weight parameters were decreased in magnitude during ridge regression. The relatively small value for lambda in the ridge regression model (0.53) helps to explain the similar performance. Since ridge regression is equivalent to linear regression when lambda equals 0, it is expected that smaller values of lambda maintain a fit somewhat close to the OLS model. The distributions of residuals for OLS linear regression and ridge regression are shown in Figure 10. These distributions are pretty similar, but ridge regression shows a greater tendency to underestimate the price compared to the OLS method.

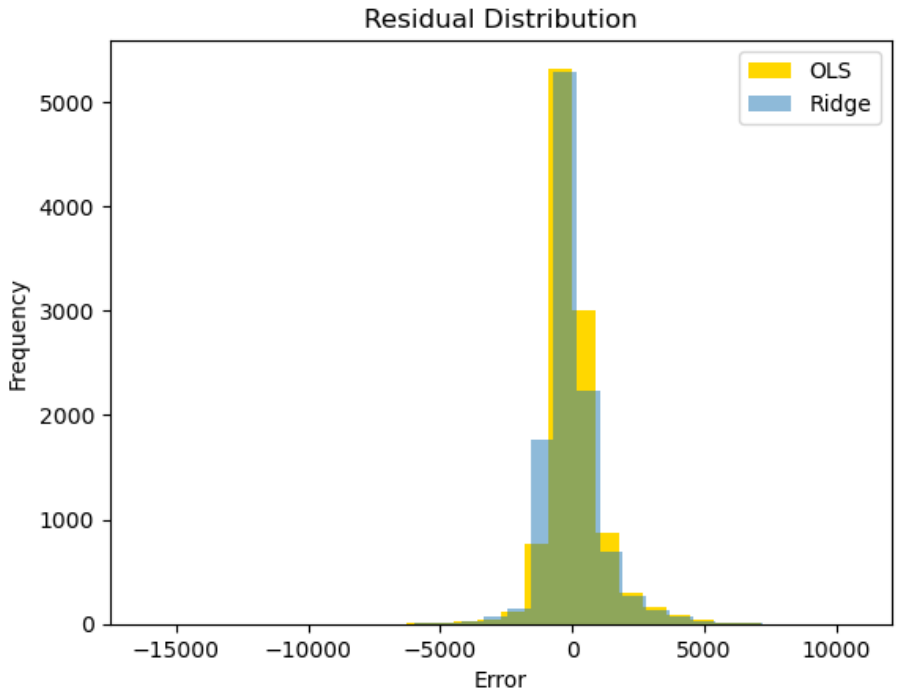

Figure 10

As a note, a sharper contrast in the performance of ridge vs. linear regression was observed during an earlier round of analysis wherein all one-hot encoded categorical variables were retained, rather than dropping the first column. Thus, there were highly correlated features present. In this case, linear regression performed terribly, whereas ridge regression obtained reasonable predictions.

Gradient Descent

Gradient descent was implemented as an alternate way to estimate the weights from ridge regression. The same lambda parameter was used. The learning rate was chosen to ensure that the method would eventually converge. The weight vector was initialized with values of 1000 for all features. The method was run until the magnitude of the gradient was equal to 0.01, which was deemed to be sufficiently close to zero. The final RMSE was 1122.5921. The weight vector is provided in the table below.

Feature	**Weight**
Bias	-2767.7851
Carat	54075.3550
Depth	833.5754
Table	-871.4799
x	-7706.7147
y	7581.3857
z	-9864.7659
cut_Good	542.7100
cut_Ideal	810.4011
cut_Premium	763.2042
cut_Very Good	690.8176
col_E	-220.1704
col_F	-271.3744
col_G	-484.1576
col_H	-984.6760
col_I	-1463.1231
col_J	-2392.3698
clar_IF	5219.7550
clar_SI1	3538.3011
clar_SI2	2591.2915
clar_VS1	4448.7432
clar_VS2	4148.1592
clar_VVS1	4882.6980
clar_VVS2	4827.9899

The weights obtained from gradient descent are nearly identical to the weights estimated in the previous implementation of ridge regression. The RMSE calculated from gradient descent is also equal to the RMSE from the closed-form solution.

References

Data Camp. What is One Hot Encoding and How to Implement It in Python. https://www.datacamp.com/tutorial/one-hot-encoding-python-tutorial

Geeks for Geeks. 2025. Matplotlib.pyplot.colorbar() function in Python. https://www.geeksforgeeks.org/python/matplotlib-pyplot-colorbar-function-in-python/

Gemological Institute of America (GIA). 2025a. The GIA D-to-Z Color Scale. https://4cs.gia.edu/en-us/diamond-color/

GIA. 2025b. GIA Clarity Scale. https://4cs.gia.edu/en-us/diamond-clarity/

Matplotlib Documentation. https://matplotlib.org/stable/gallery/color/named_colors.html; https://matplotlib.org/stable/users/explain/colors/colormaps.html; https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html

Pandas documentation. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html; https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html

Seaborn. Choosing color palettes. https://seaborn.pydata.org/tutorial/color_palettes.html