Diamond Price Prediction

Data Wizards team

Monzer Hweit

Alaa Odat

Mariam Zaki

المحتويات

Figures

Discover the Data:

We started by reading the data using pandas method *read_csv()*, then we printed the data shape and the first few rows of the data using *head()* method.

```
(43152, 11)
```

	Id	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	1.06	Ideal	I	SI2	61.8	57.0	4270	6.57	6.60	4.07
1	2	1.51	Premium	G	VVS2	60.9	58.0	15164	7.38	7.42	4.51
2	3	0.32	Ideal	F	VS2	61.3	56.0	828	4.43	4.41	2.71
3	4	0.53	Ideal	G	VS2	61.2	56.0	1577	5.19	5.22	3.19
4	5	0.70	Premium	H	VVS2	61.0	57.0	2596	5.76	5.72	3.50

Figure 1: data reading

The *Info()* method was useful to get a quick description of the data, in particular the total number of rows, each attribute's type, and the number of non-null values, it turned out that there are 43,152 instances in the dataset, and all of the values are non-null values and all the columns are numerical except for three categorical columns ('cut', 'color', 'clarity').

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43152 entries, 0 to 43151
Data columns (total 11 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Id       43152 non-null  int64
 1   carat    43152 non-null  float64
 2   cut      43152 non-null  object
 3   color    43152 non-null  object
 4   clarity  43152 non-null  object
 5   depth    43152 non-null  float64
 6   table    43152 non-null  float64
 7   price    43152 non-null  int64
 8   x        43152 non-null  float64
 9   y        43152 non-null  float64
 10  z        43152 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 3.6+ MB
```

Figure 2: data info

Then we used the *describe()* method that shows a summary of the numerical attributes:

	Id	carat	depth	table	price	x	y	z
count	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000
mean	21576.500000	0.797855	61.747177	57.458347	3929.491912	5.731568	5.735018	3.538568
std	12457.053745	0.473594	1.435454	2.233904	3985.527795	1.121279	1.148809	0.708238
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	10788.750000	0.400000	61.000000	56.000000	947.750000	4.710000	4.720000	2.910000
50%	21576.500000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	32364.250000	1.040000	62.500000	59.000000	5312.000000	6.540000	6.540000	4.040000
max	43152.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

Figure 3: data describe

Based on these results we can see that the price std is higher than its mean which indicates the presence of outliers. we also noticed that the min value for x, y, z is zero, in some cases zero is considered a missing value, but in our case since the rang of values for these three columns starts from zero, we won't consider it as a missing value.

Visualize & Analyze the Data:

In this step we started by plotting the histogram of the numerical attributes to gain a basic knowledge about our attribute's distribution

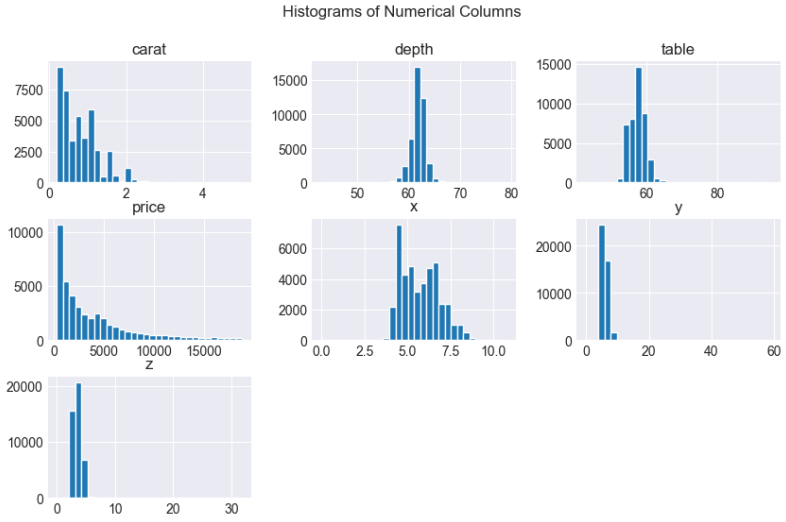

Figure 4: histogram

Then computed the standard correlation coefficient (also called *Pearson's r*) between every pair of attributes using the *corr()* method, and to make the visualization better we used the heatmap plot from seaborn, the results showed that that there is high correlation between price, carat, x, y, z features, And that there is a small negative correlation between the price and the depth.

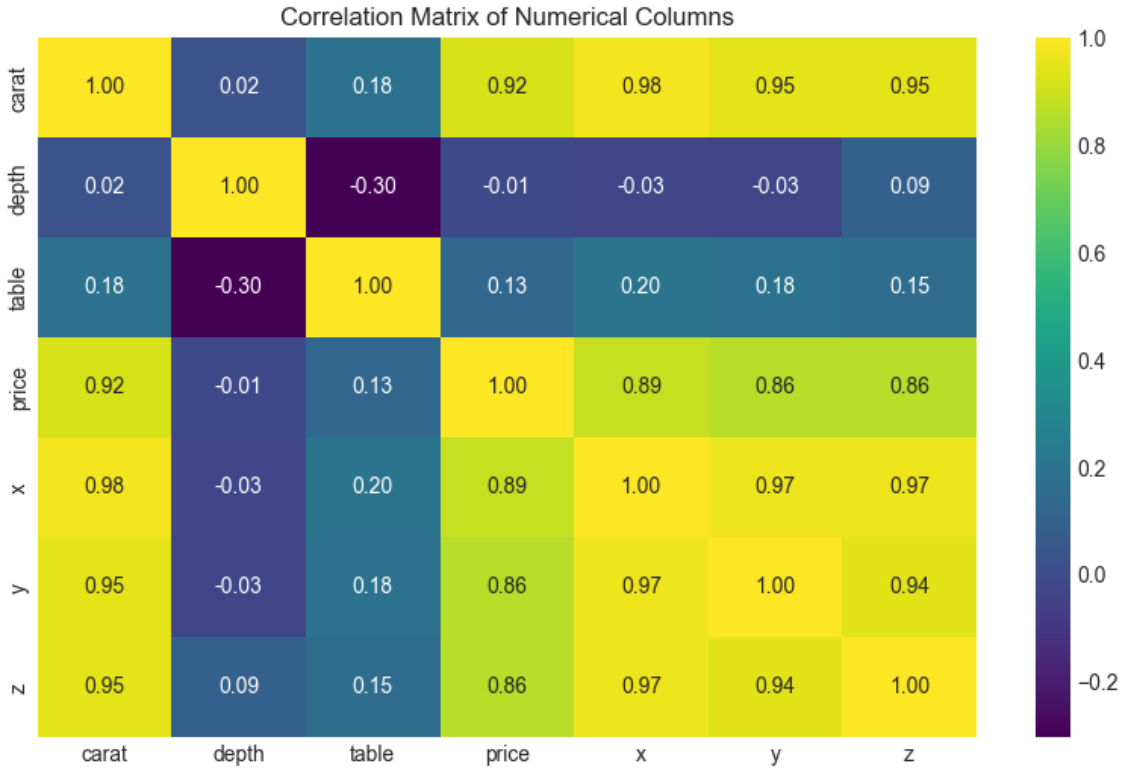

Figure 5: correlation matrix

Then to visualize the relationships between pairs of numerical attributes in our dataset, and examine both the individual distributions of each attribute and the pairwise relationships between attributes, we used pair plot and the results showed that there is a positive correlation between carat weight and price. This means that diamonds with a higher carat weight tend to cost more (this is likely because larger diamonds are rarer and more difficult to find), there is also a positive correlation between carat, price and all three diamond measurements (x, y, z).

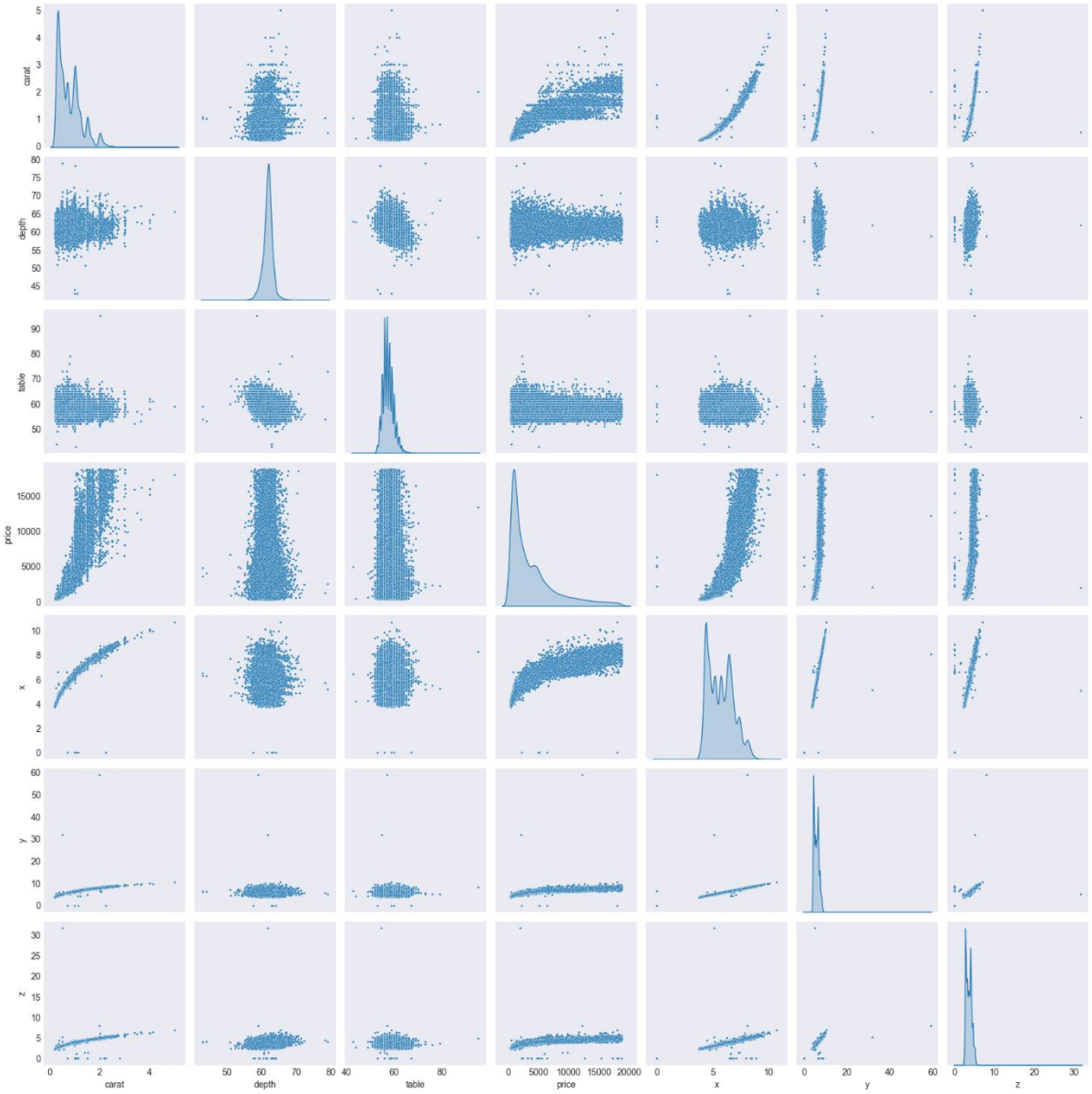

Figure 6: Pair Plot

We also used the reg plot to assess the direction, strength, and uncertainty of the relationship between the price and the other numerical attributes, and since we suspected the presence of outliers previously the reg plot is helpful in identifying outliers, we can see data points that are far away from the main cluster of points or that cause the regression line to deviate from the general trend.

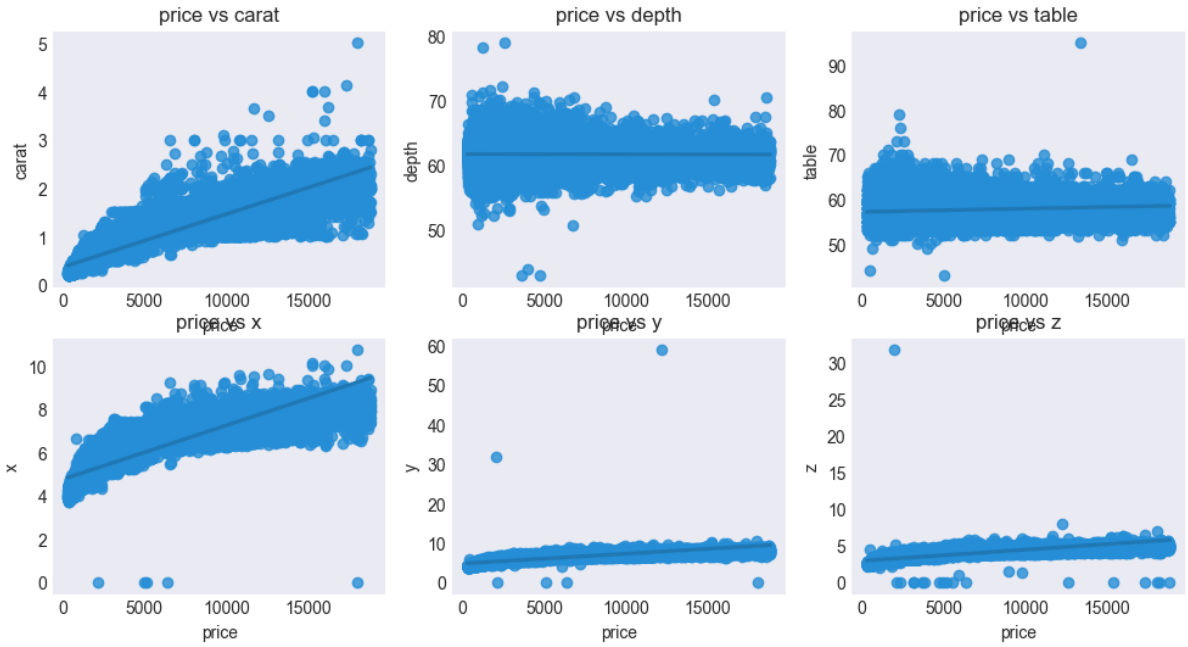

Figure 7: Reg plot

To visualize the distribution of categorical data we used seaborn count plot, and the results were:

Figure 8: Count Plot

For clarity, the most frequent category is SI1, followed by VS2 and SI2, for cut, the most frequent category is Ideal, followed by Premium and Very Good, for color, the most frequent category is G, followed by E and F.

Then we used violin plot to explore the relationships between these categorical features (clarity, cut, color) and price, and here are the resulting plots:

Figure 9: Violin (price for cut)

Based on this plot we can see a wider violin for "Ideal" cuts compared to "Fair" cuts, indicating a larger spread of prices for higher quality diamonds, The median price follows a clear trend (increasing from high quality cuts to low quality cuts), which suggests a correlation between cut quality and price.

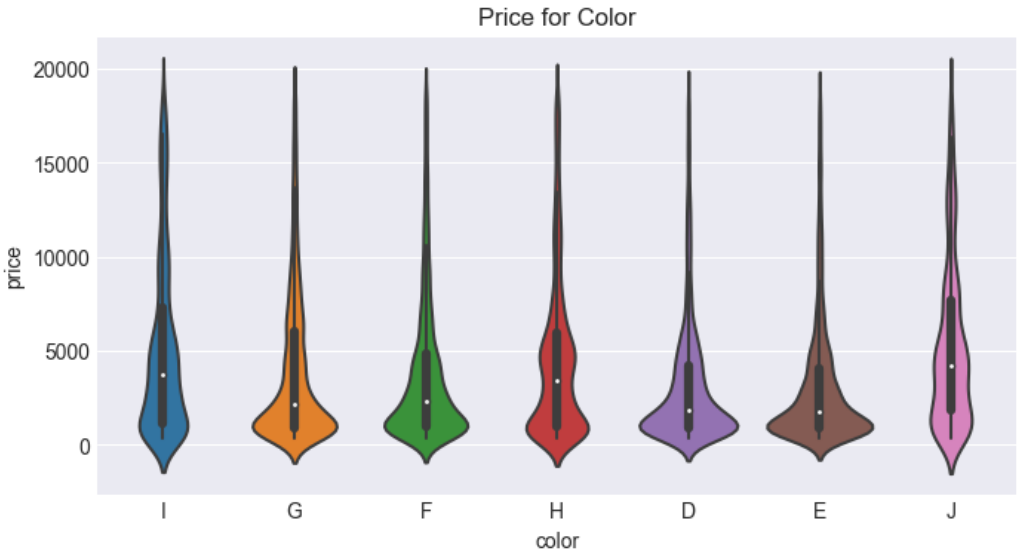

Figure 10: Violin (price for color)

Here we can see a wider violin for "D", "E" colors compared to "J", "I" colors, indicating a larger spread of prices for better color diamonds.

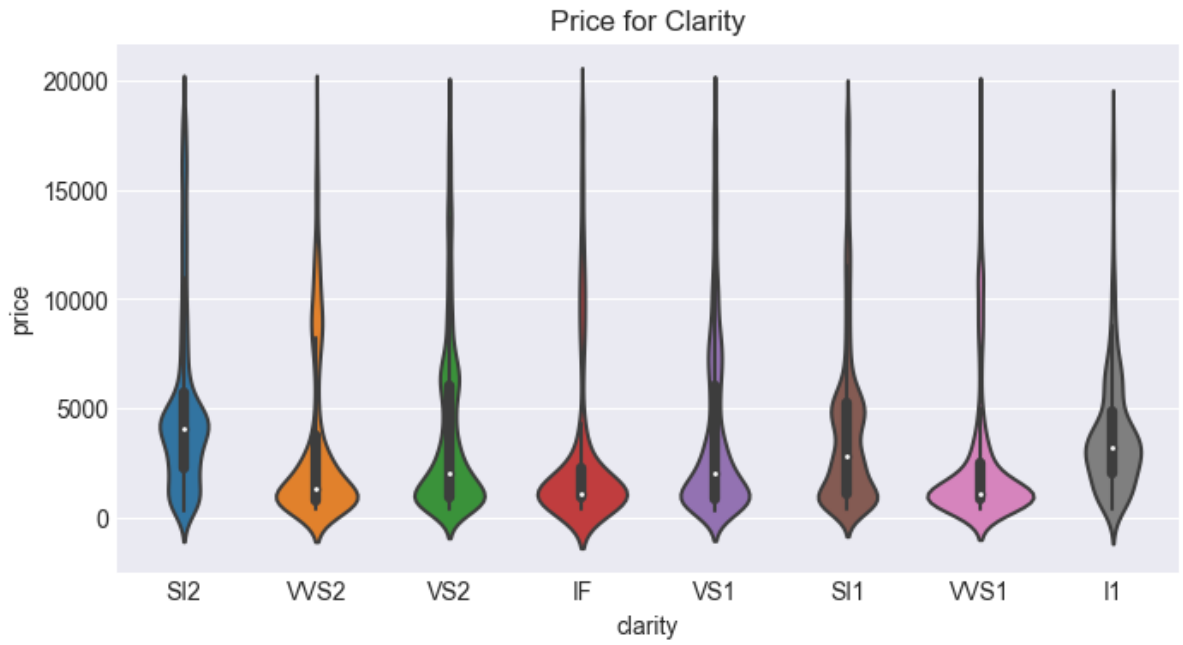

Figure 11: Violin (price for clarity)

From this plot we can tell that there is a wider violin for better clarity "IF","VVS1" compared to worse clarity "I1", "SI2" indicating a larger spread of prices for clear diamonds, the more the diamond clarity gets better the more the median price decrease, this suggests a correlation between diamond clarity and price.

Preprocess the Data:

In this step we first built our own preprocessor that drops the Id column since it won't be beneficial for the training process, then creates a new feature called volume through multiplying x, y, z, and use ordinal encoder to encode the categorical features, we chose the ordinal encoder since there's a clear order or ranking between the categories of the categorical features, the ordinal encoder assigns numerical labels to categories of a categorical features, preserving the inherent order or ranking between them.

Then we handled the outliers by eliminating data points that are far from the main cluster using the reg plot.

After that we separated the data into features (carat, depth, table, cut, clarity, color, x, y, z, volume) and label (price), then splitted them into training and validation sets with a proportion of 75% for the train set and 25% for the validation set in order to asses our model performance.

Our data has features with different value ranges, for example the carat feature has values ranging from 0 to 5, while other features like depth might range around 60, x could be ranging from 0 to 10 and the new feature volume ranges for 30 to 790 and so on.

These disparate ranges can cause problems for machine learning algorithms because features with larger values will dominate the model's learning process, even if they aren't necessarily more important so, in order to ensure that all features contribute equally to the model's learning process regardless of their original units or scales we added a Standard Scaler. Then created a pipeline called 'full_pipeline' that combines the preprocessor and the standard scaler and applied it to the X_train through *fit_transform()* method and to the x_valid through *transform()* method.

Model Building:

We started with a **Linear Regression** model as a baseline model, and after fitting the training data and computing the predictions, the root mean squared error (RMSE) result was 1181.90 which is not satisfying but good as a starting point.

Then we tried **Polynomial Regression** by adding polynomial features from the third order using the function *PolynomialFeatures(degree=3)* from scikit-learn, the RMSE score was 636.26 which is far better than the previous model, that indicates the presence of complex patterns (nonlinear relationships) in the data that the linear regression model couldn't capture.

After that we went with more complex model which is **Decision Tree Regressor** that yielded a 747.31 RMSE, then applied cross validation

to ensure the model results is reasonable and that it doesn't overfit the data.

Then we used **Random Forest Regressor** with no parameters, we got 535.57 RMSE score, so we applied grid search to get the best values for the parameters max_features, n_estimators and bootstrap the best estimator parameters were:

max_features=8, n_estimators=300 and bootstrap=True

after using the best estimator to fit the training data and compute the predictions we had a 530.93 RMSE result, then we set the n_estimators parameter to 200 achieving 530.64 RMSE score which is slightly better than the previous one.

We also used **KNN Regressor** to fit the training data, the RMSE score for this model was 650.64 which is worse than the previous model.

For our final model we used **XGBoost Regressor** which is common for regression problems with no parameters set, and we got RMSE score of 535.25 (still worse than our previous random forest score) so, we applied grid search to get the best estimator with the following params:

max_depth=6, min_child_weight=3, n_estimators=300

the grid search's best estimator achieved 512.69 RMSE score, we then added more parameters and tried different values to get a better score, and we achieved 509.54 RMSE with the following params:

objective='reg:squarederror', max_depth=8, n_estimators=100,
learning_rate=0.1, min_child_weight=3, reg_alpha=2,
reg_lambda=1, random_state=42

and that is by far the best model we had.

Model Evaluation:

We read the test data and created the submission data frame that has two columns ‘Id’ and ‘price’, then we processed the data using the full_pipeline *transform()* function, and computed the predictions using our best model ‘XGBoost’, finally after submitting our solution we got a 538.39 RMSE which is relatively good for new unseen data.