Diamond price prediction 2024

(Kaggle Competition)

Team Members:

Omar Alnounou (Team Leader)

Sameer Alatrash **Omar Mardini**

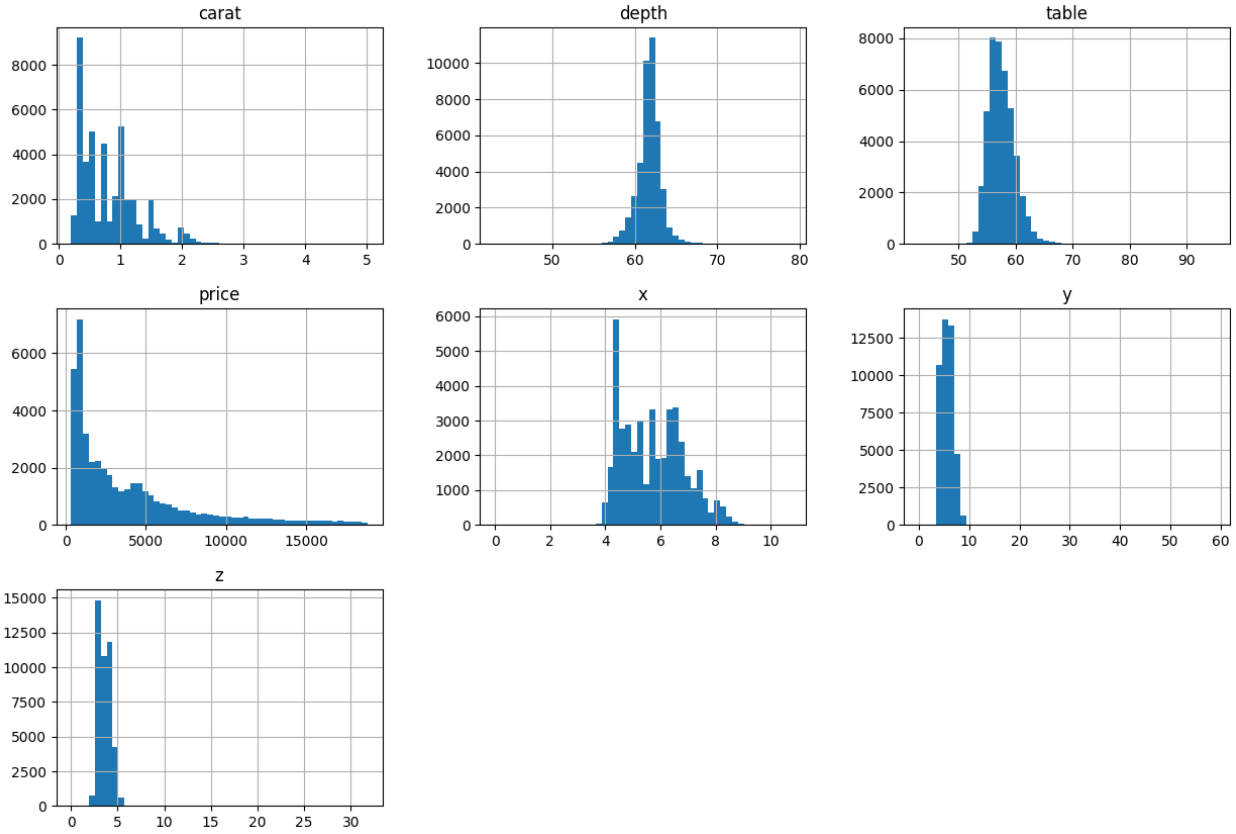
Nouran Mostafa **Hagar Ali Abdelrahman**

Coach: Maryam Asha

EDA & Data visualization:

- First, we took a quick look at our data, exploring each features value types and checking for any null values.

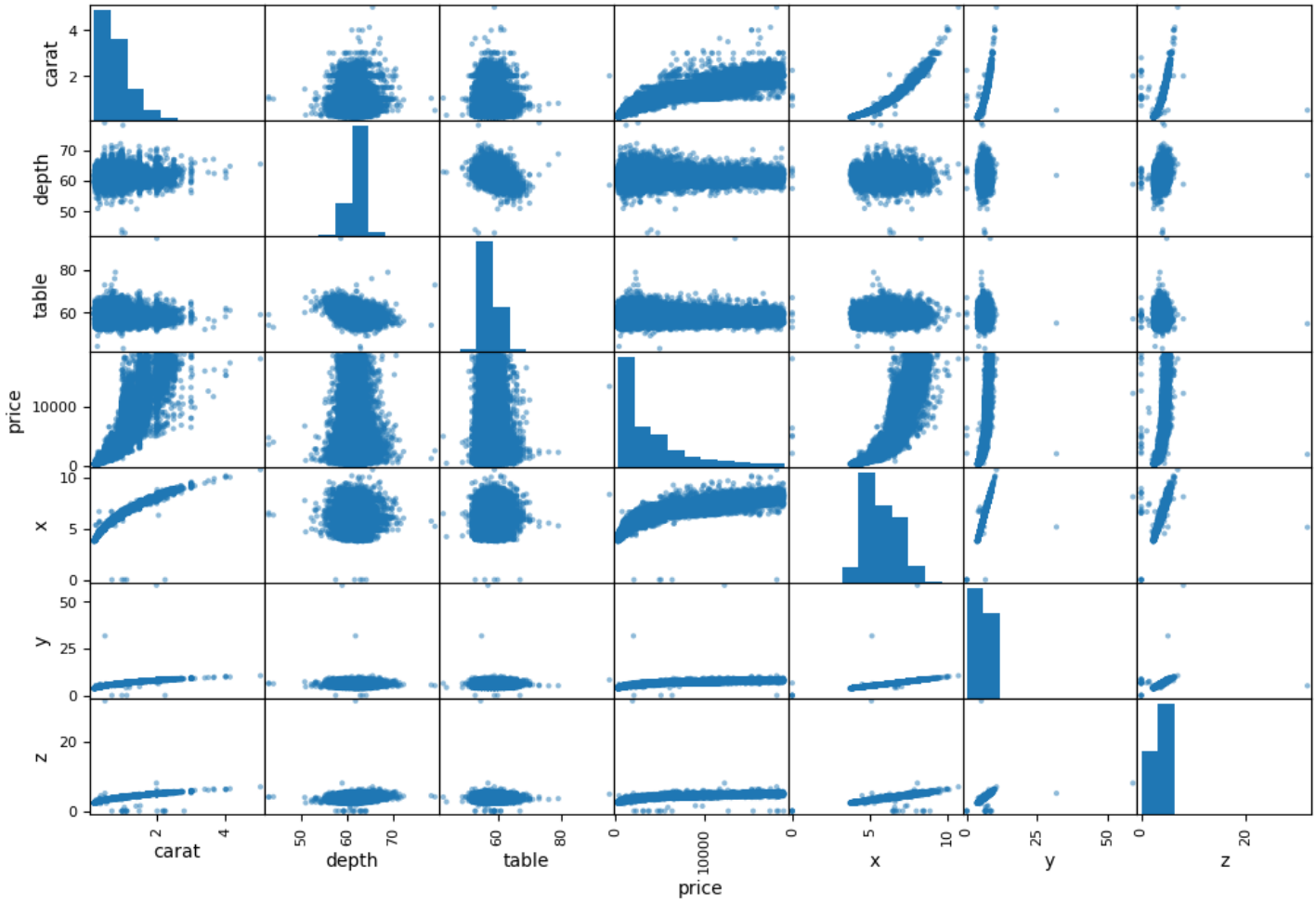
- Using *describe()* and a simple histogram, we check the distribution of each of the numeric values inside my dataset.



We got to know where the majority of my target value [price] is located, we also noticed how the [table, depth] both represent a normal distribution where the majority of their values are located around the median, and lastly, we got a broader look of how the rest the features' values are distributed, most importantly the [carat] feature where 75% of it is under 1 carat.

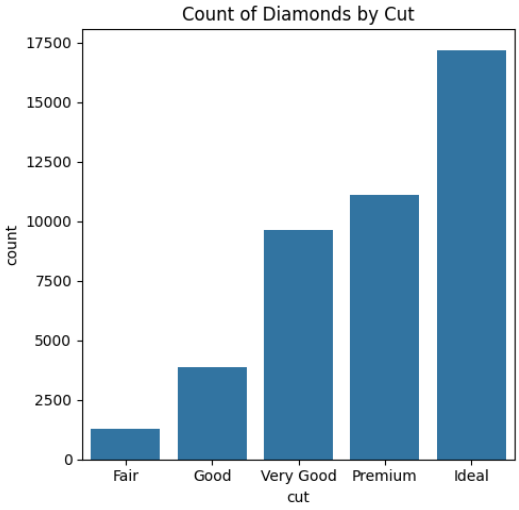
- Using *nlargest(20, 'price')* & *df.loc[df['carat'].idxmax()]* we got to see what attributes the most expensive diamonds have, we can see the best colors (D, E) are not present. also, the clarity quality isn't exactly the best measure for price considering most of them have an (SI) clarity which is not the best.
  Additionally, the highest [carat] value does indeed yield a relative high price. (Just some information to better know my data)

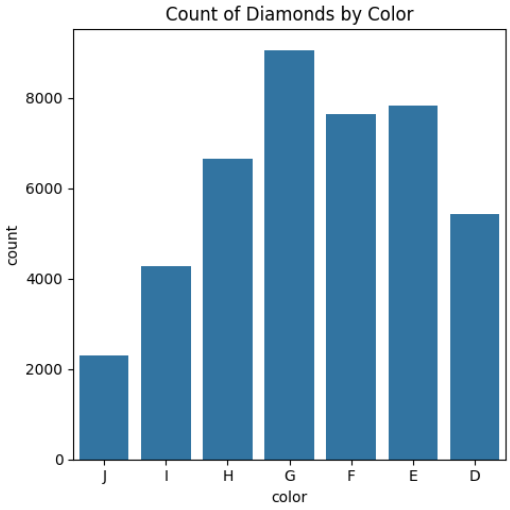
- Using a *scatter_matrix* to show the correlation between each of my numerical values:

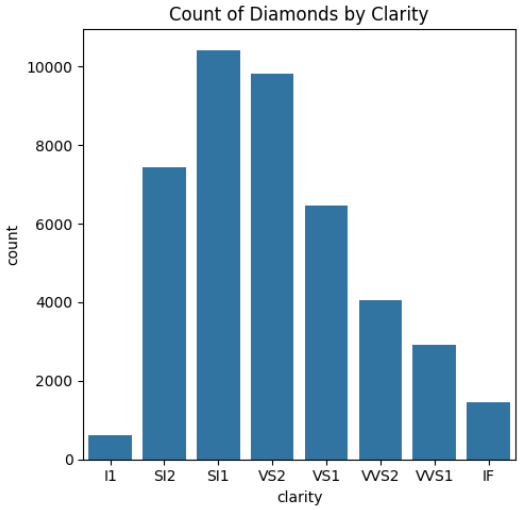


We can clearly tell the strongest correlations with our target label [Price] are the [carat] and the dimensions columns [x, y,z] while both the [table] & [depth] don't have a strong Linear correlation. Fortunately, through a small google search around diamonds we know that the best values for these two attributes are in the 50-60s% range, but as we can see from the previous plots their effect on price can differ where diamonds with perfect depth/table can still cost anywhere on the price range spectrum.

- Moving into our categorical attributes, we checked how many unique values are present and how much each value is present in my dataset:

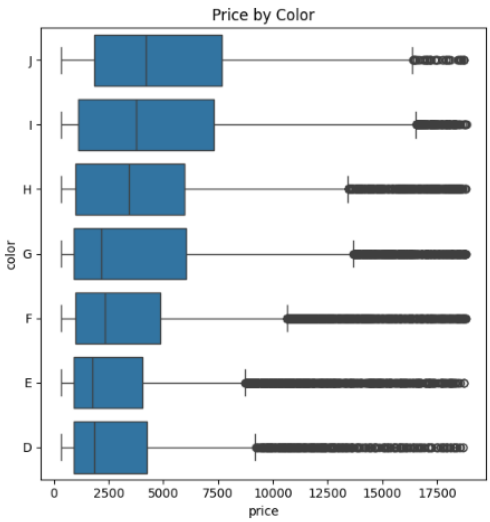


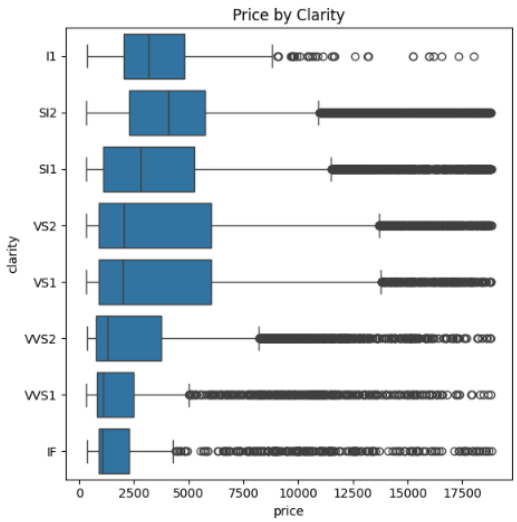


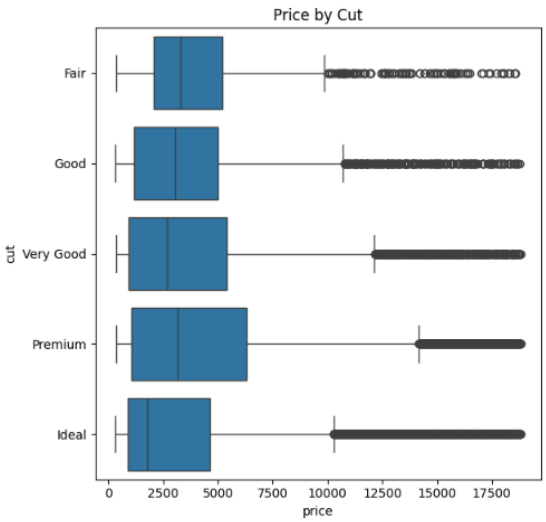


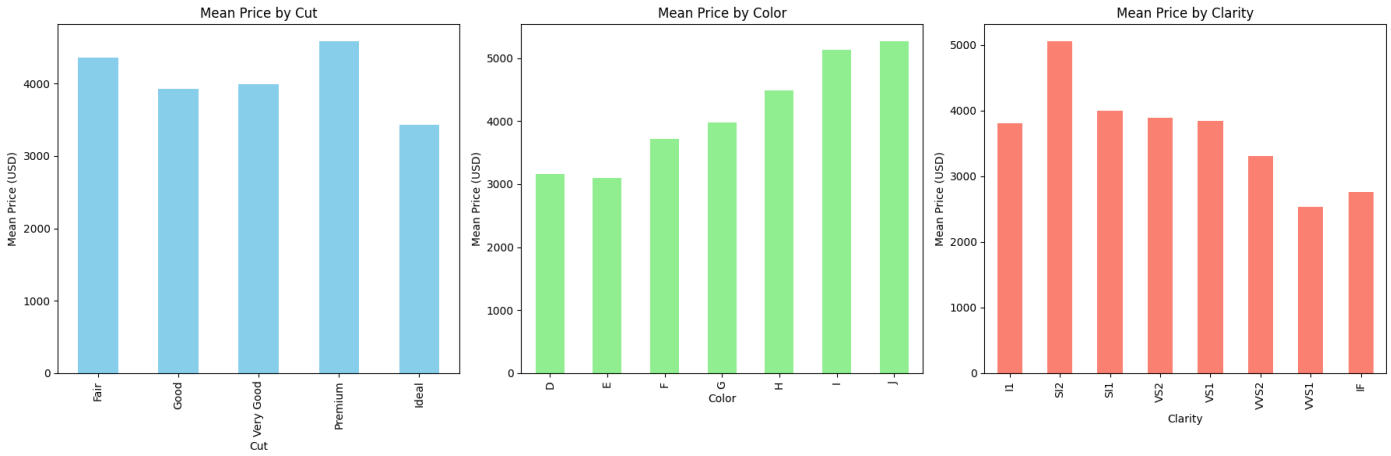
Most of the [cut] values are ideal which is the best value, while both the [color] & [clarity] values are focused around the middle best values.

- To better know how well these categorical values are placed according to my [price] target value:



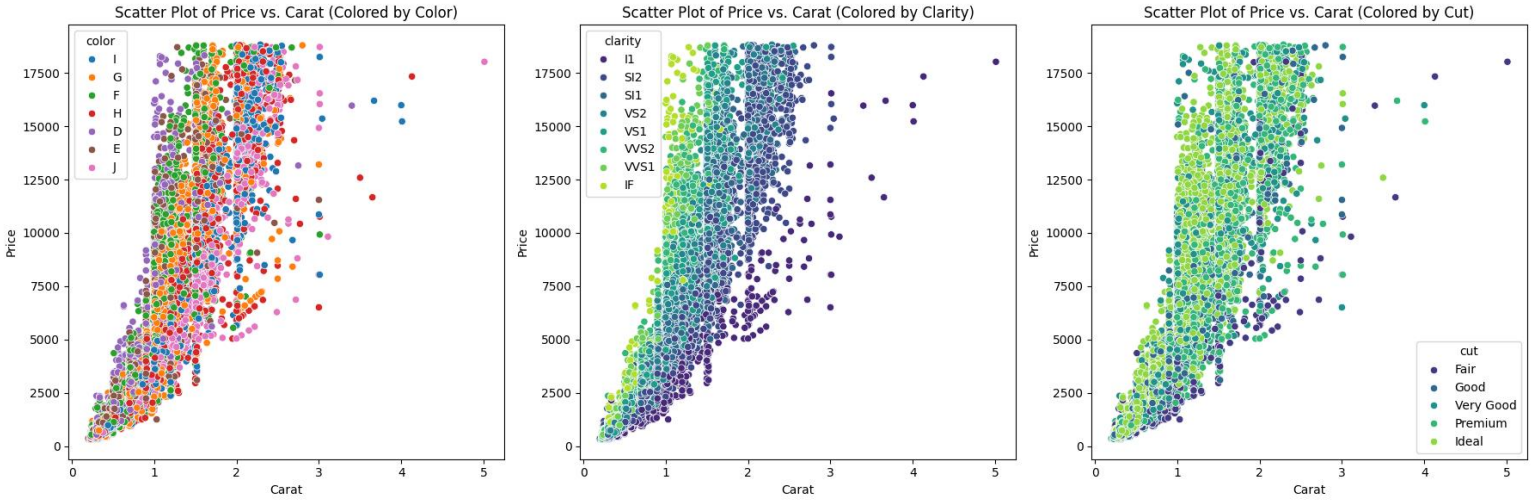






We know these categorical values are of ordinal type where they follow a natural order or ranking, however seeing how they range in my price, we could tell that this is not quite the case here as they don't necessarily have any correlations with price. Specially in my [color, clarity] attributes where the higher prices are present in somewhat “bad” values, which can lead us to believe that these features aren’t the most important regarding our target [price] value.

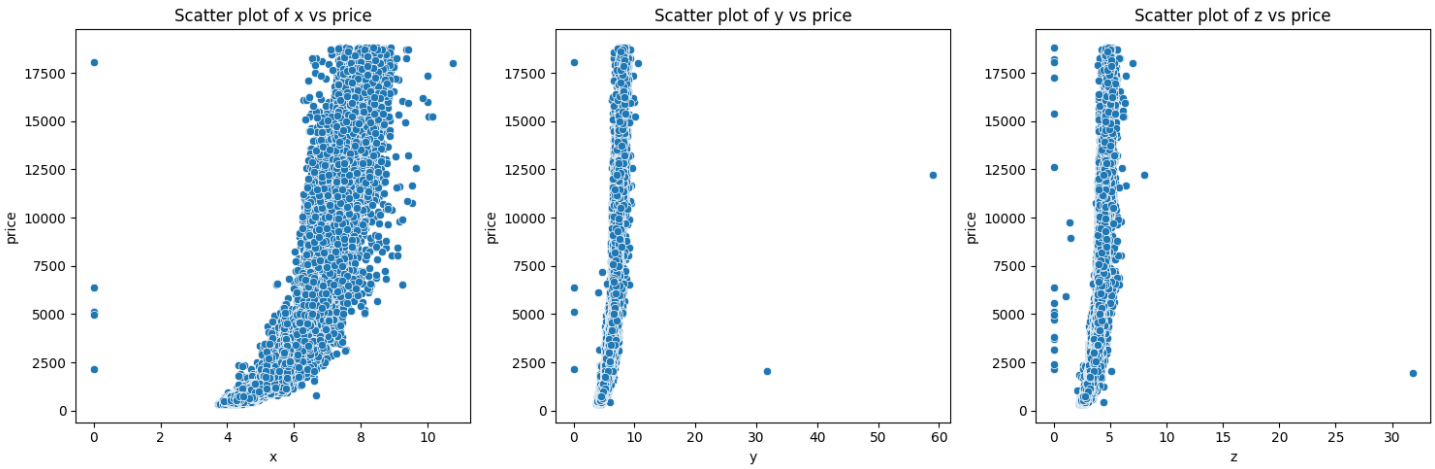
- To get a better understanding of these categorical attributes, we plotted them against the numerical feature [carat] which we know have the best correlation to the [price]



Nothing much was known regarding the [color, cut] values as we know they are scattered around my entire data in varying prices. But the most important takeaway here is how the [clarity] values are correlated to my [carat] feature, where the higher the carat value is, the lower the diamonds clarity gets, which can be quite ironic.

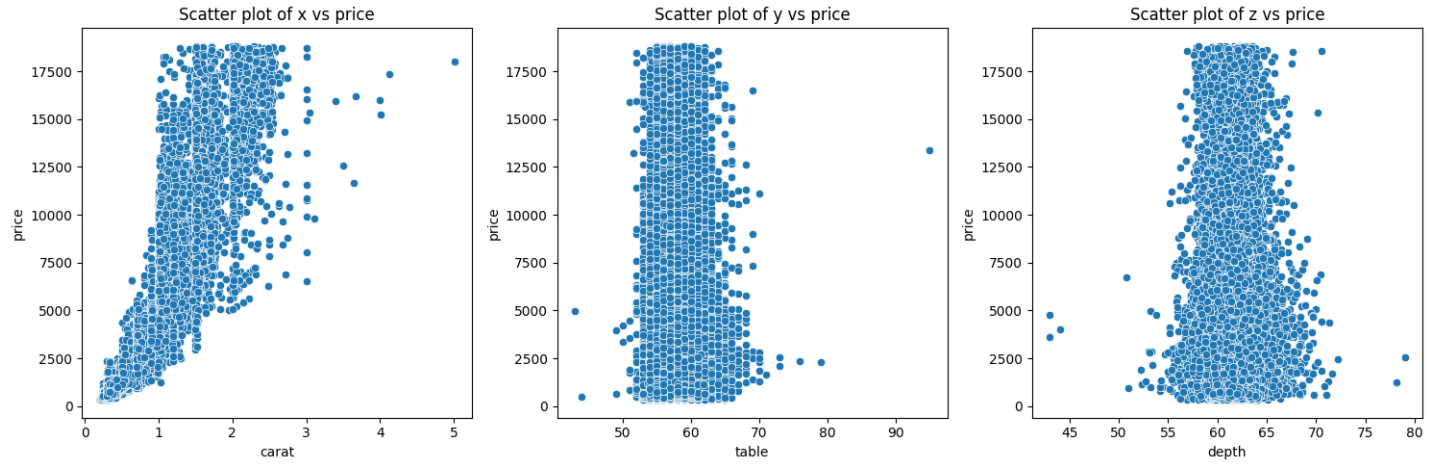
Dealing with outliers and null values:

- We know there are none specific “null” values, but looking at the plots below we noticed a few “0” values in my [x, y] features, so we removed them.



Additionally, we noticed some outliers where [y] & [z] values are above 30 so we removed those as well as they can’t affect my model’s prediction.

Plotting the same scatter plots for my other numerical features, we also noticed a few outliers so we removed them.



Handling text and data scaling:

We created a simple pipeline where our numeric features undergo standard scaling using *StandardScaler*, as for our categorical features they are transformed using *OrdinalEncoder* which converts categorical values into ordinal integers based on specified custom orders as per noted in the data description.

We dropped our target value [price] as *y*, and fit_transformed my data into *X_transformed* to use it for training my ML models.

Training ML Models:

- Using a simple *LinearRegression* as our base model we got an RMSE score of: 1197.68
- Moving into a more complex *RandomForest* model with *CrossValidation* we got a mean RMSE score of: 544
- On a more powerful *XGBOOST* approach with Hyperparameter Tuning using *BayesSearch* we got an RMSE score of: 408.5 which is a good improvement.

**RMSE score based on train data

Final Notes:

Other combinations not present on the final notebook were tried, trying some feature merging but not netting quite good results.

As per categorical values, we know they are of ordinal order. However, plotting the data and seeing how small they effect the target [price] and even sometimes contradicting other stronger features like [carat]. It maybe better to try other encoders like "*OneHotEncoding"*, or dealing with them in some other way (eg, dropping column, feature merging) will need further investigation.

Outliers detection wasn't performed in the most optimal way. Using a better density or statistical approaches would almost certainly give a better detection than manual input.

Finally, training a more complex model should also be tried, and comparing it with existing models.