Diamond Prices Prediction

The Diamond Complete Case Study involves analyzing a dataset containing information about diamonds, with separate files for training and testing data. In this case study we focused on three main parts and they are:

1- Part 1: Exploratory Data Analysis (EDA).

2- Part 2: Data preprocessing and cleaning.

2- Part 3: ML and training the models.

As for the data it contained 54000 records with 11 features and they are:

* Id: The unique identifier for each diamond.
* Carat: The carat weight of the diamond (numeric).
* Depth: The depth percentage of the diamond, calculated as the depth divided by the average width of the diamond (numeric).
* Table: The width of the top of the diamond relative to the widest point (numeric).
* Price: The price of the diamond in USD (numeric).
* X, Y, Z: The dimensions of the diamond in millimeters (numeric).
* Cut quality of the cut.
* Clarity a measurement of how clear the diamond is.

With Price being the value that we want to predict.

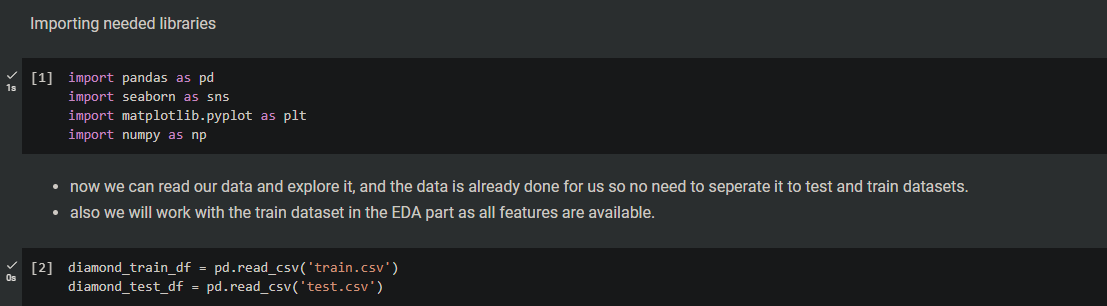
Now we will talk about each part in detail and all the steps that we did till we reached our goal and achieved a low error rate.

**Part 1: Exploratory Data Analysis (EDA):**

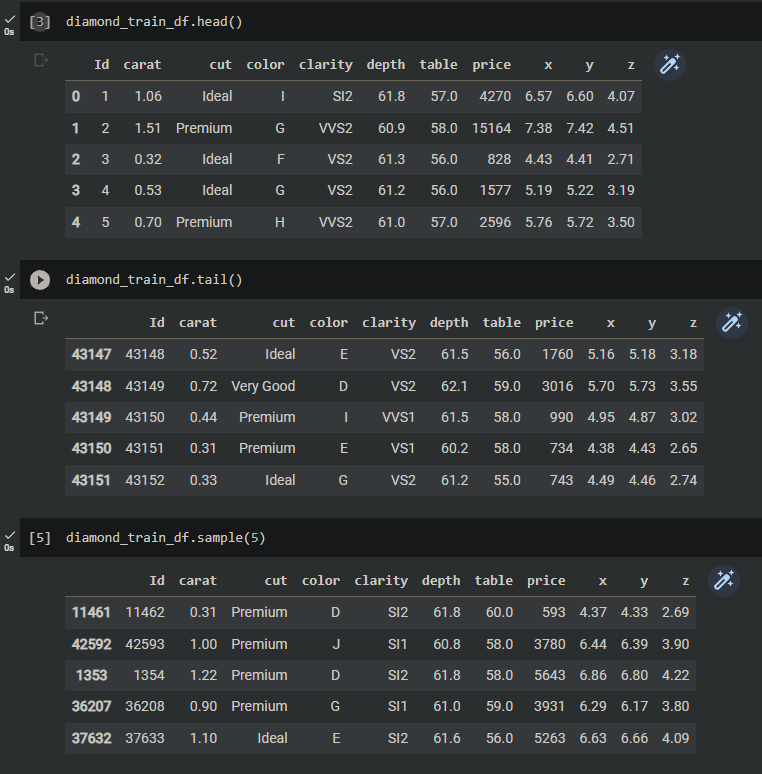
* **Data exploration:**

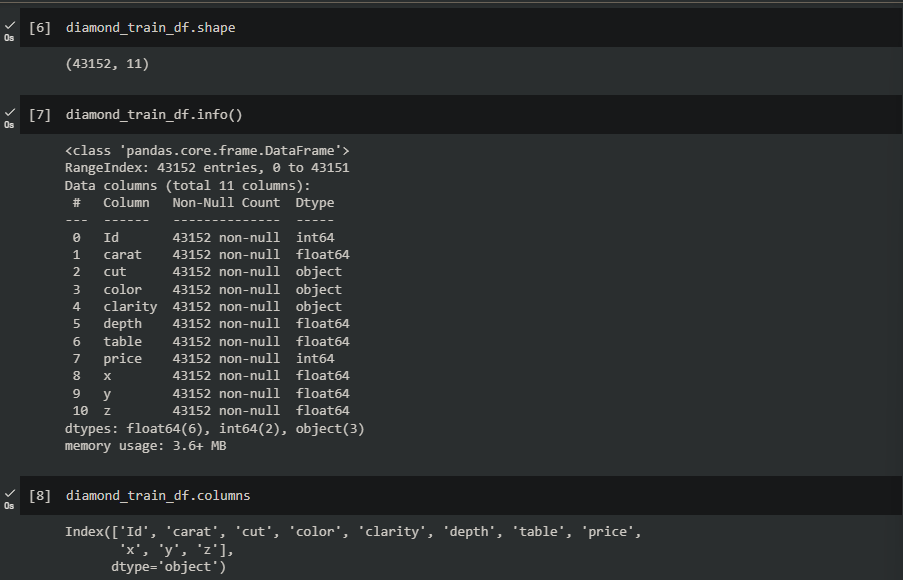
Our data set is divided into two files, one for training and the other for testing,

The first step that we did was reading the files into a dataframe using the pandas library:

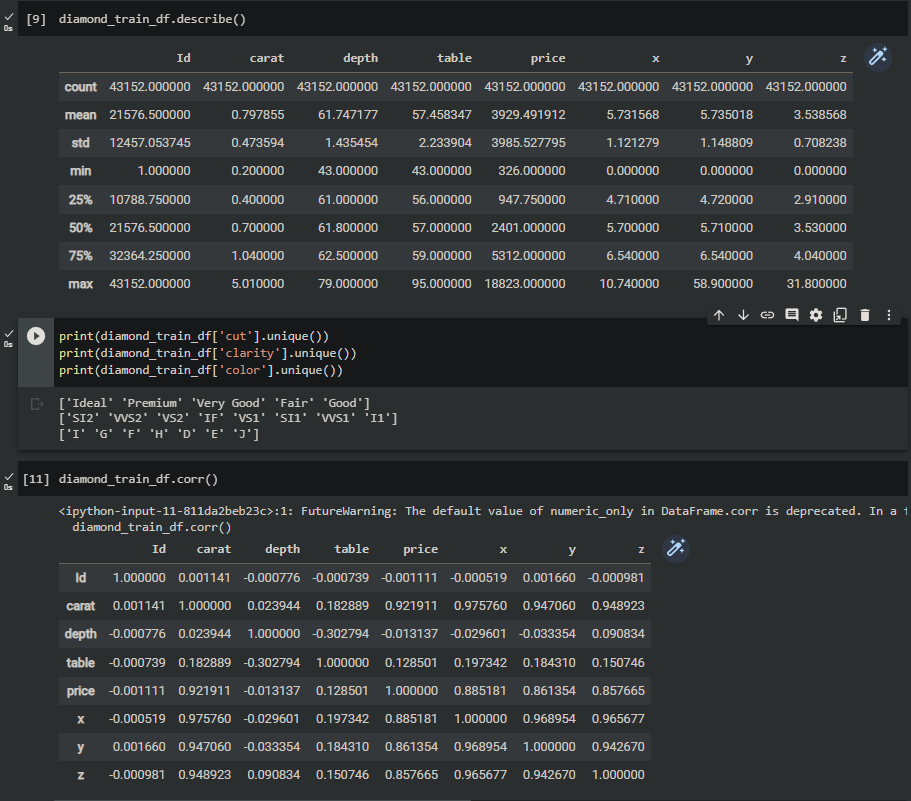
 Now we can work with our data and explore it.

We did the main and obvious operations, so we know what we are working with,

And you can see the output of each operation below: 

* *head(), tail(), and simple() to see some examples of our dataset.*
* *shape, info(), columns to know how many records we have, name of each column(feature), and to check datatype of each column and if there’re any null values*

From the output above we can see that our training dataset contains 43152 records with 11 columns, it doesn't contain any null values but we have some object data type columns that we want to deal with, and lastly we knew the names of our columns.

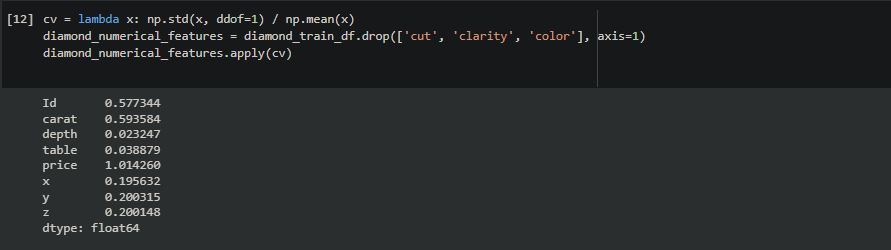


* *describe(), unique(), and corr() to show a summary of statistics, to show unique values of the column, and to show correlation of each column.*

From the output above we can see that the statistics of our numerical data type columns, and from it we can assume that there might be some outliers that we want to deal with based on the max and min values compared to the mean, then we wanted to know the categorical values that we are dealing with, and finally we wanted to see the relationships between our columns.

Also based on the corr() operation we were able to know the following:

* **When the value is close to 1 or -1 that means there's a strong relationship .**
* **Carat had a high correlation with Price, X, Y, and Z.**
* **Price had a high correlation with Carat, X, Y, and Z.**
* **X had a high correlation with Carat, Price, Y, and Z.**
* **Y had a high correlation with Carat, X, Price, and Z.**
* **Z had a high correlation with Carat, X, Price, and Y.**



And the last thing we did in exploring our data was defining a function to calculate the Coefficient of Variation to determine how much the data is spread, the closer the value to 0 the less the data is spread and it means sometimes no outliers or there are few in our data.

* **Data visualization:**

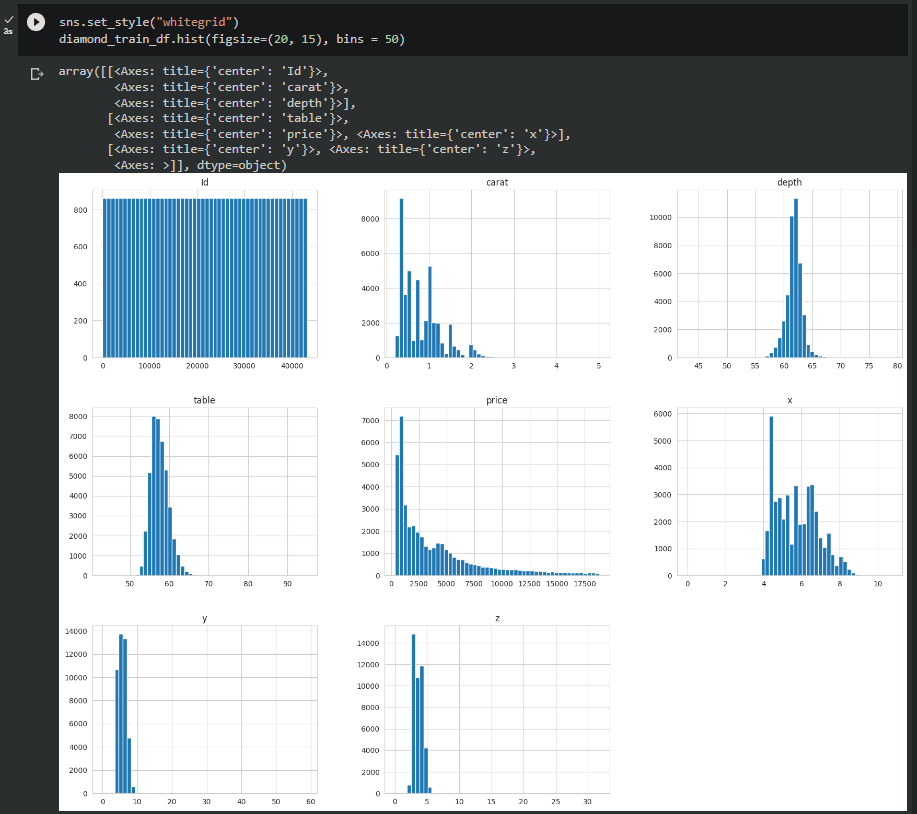
Now we can start visualizing our data and try to discover useful insights and relationships.

We mainly used scatterplot, histogram, boxplot, displot, and countplot.

A quick explanation of each:

1. Histograms: Histograms were used to visualize the diversity of values in each attribute, providing insights into their distributions.
2. ScatterPlots: Scatter plots were created to visualize the relationships between attributes and if there is a correlation between them or not.
3. Countplot: Plots were generated to visualize the count of unique values for each attribute, identifying any biases in the data.
4. Boxplots: Box plots were used to detect outliers in the dataset. High ratios of outliers were observed.

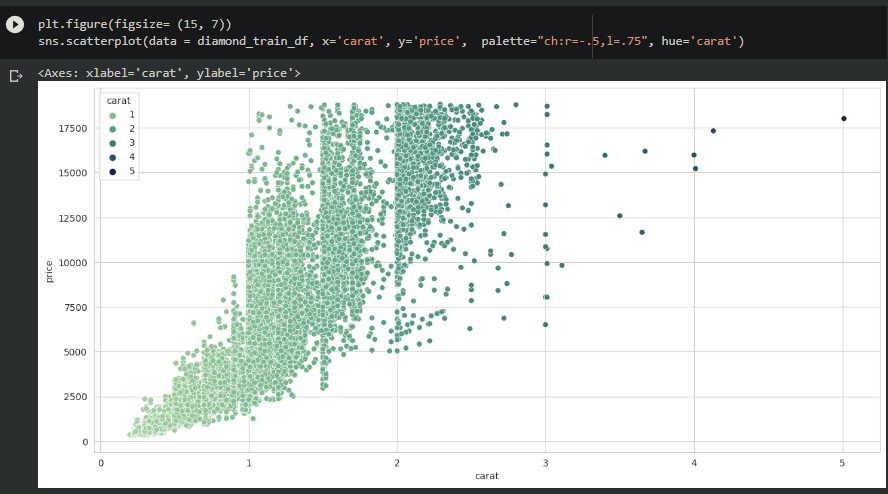
And now we will talk about each plot and what did we notice and what insights did we gain,



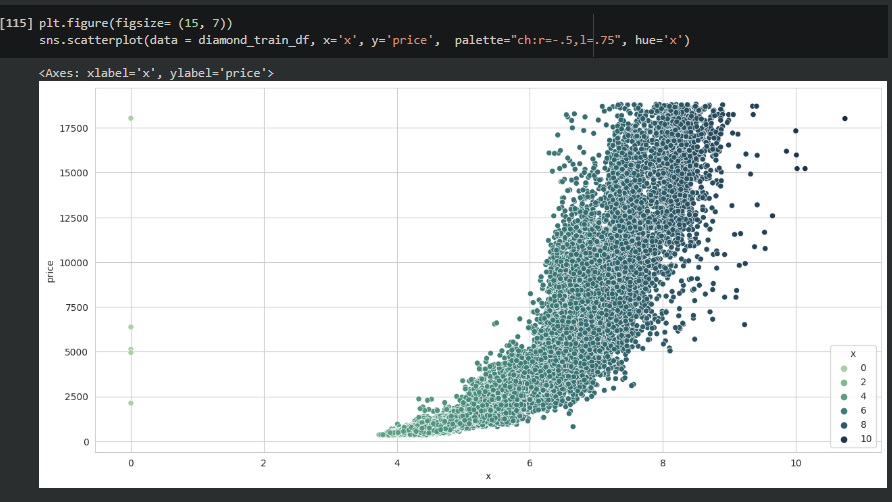
We can notice many things from the above graph, we can see that depth values aren’t spread and they are around the mean value.

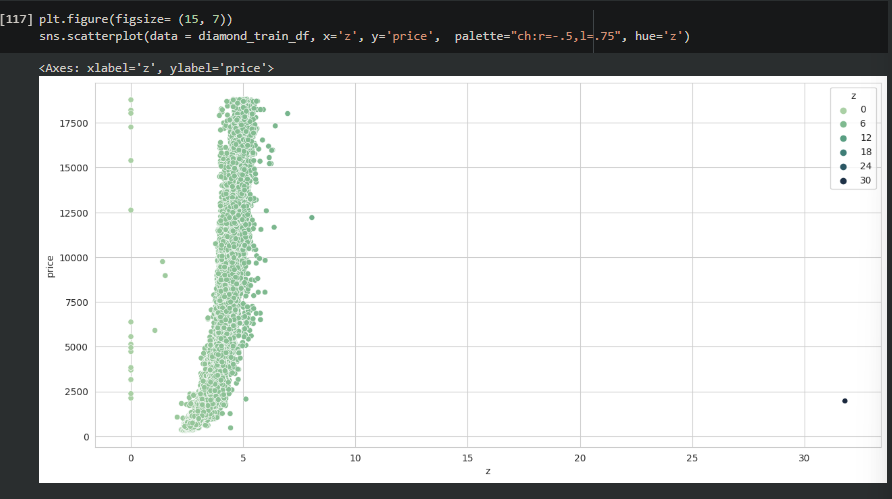
We can see that price is skewed, while y, z, and carat have outliers so they look skewed, but in reality they aren’t.

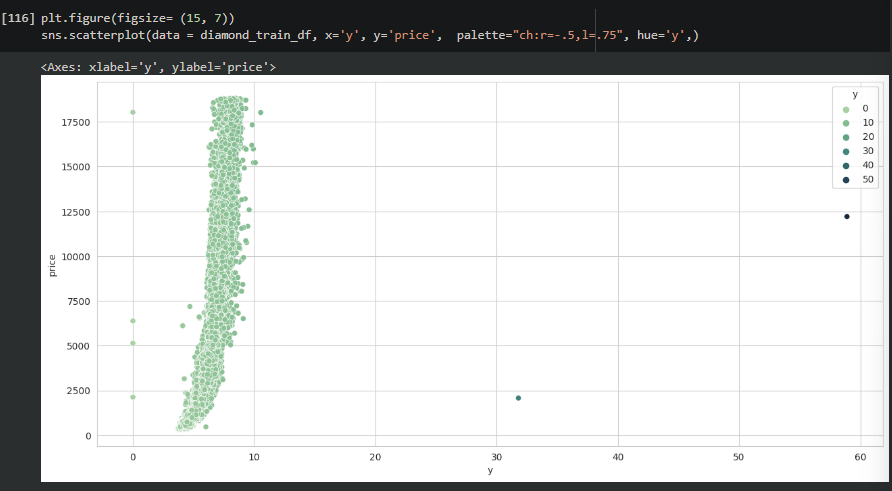
Finally x and table are almost normally distributed.



We noticed that there’s a strong relationship between the price and carat, where the heavier the diamond is, the more expensive it gets



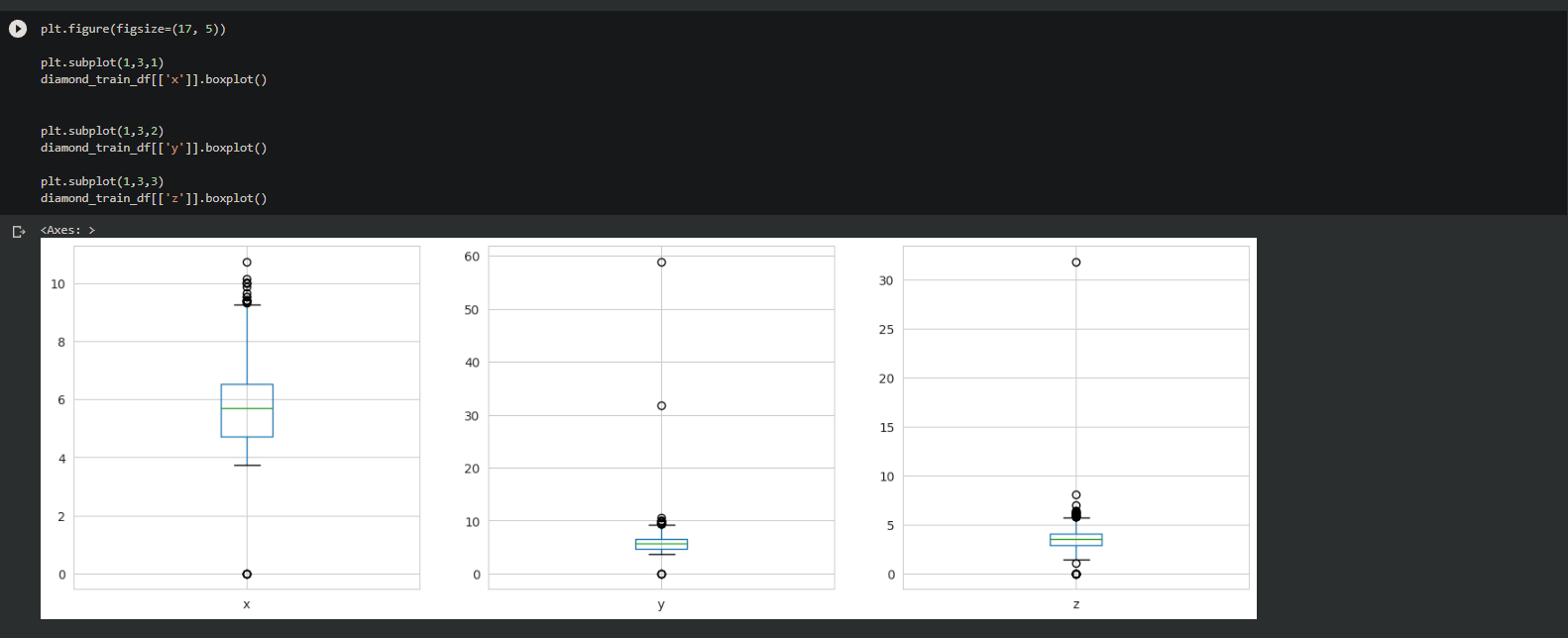




We noticed the same thing for x, y, and z.

There's a strong positive relationship between them and the price.



From the graph above we just wanted to know for each categorical value which had the most count in each column.

And finally we were able to notice there are outliers in x, y, and z columns that needs to be dealt with

Now that visualization is done, we want to make sure our data is clean and ready for training.

**Part 2: Data Preprocessing And Cleaning :**

First we create a function to detect the outliers which’s *detect\_outliers\_zscore()*, and this function uses the mean value and standard division to calculate the z score for each value.

After that for each outlier we replaced it with the median value of the column, we were dealing with x, z, and y columns.

Although there are many ways to deal with outliers, we chose to go with the median method.

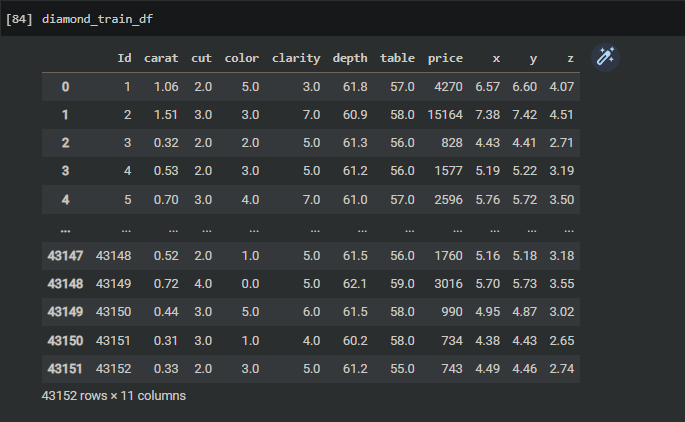
Then we made sure that our data is clean from null value, we already knew the form *info()*, but we are just making sure.

Finally we want to deal with categorical values.

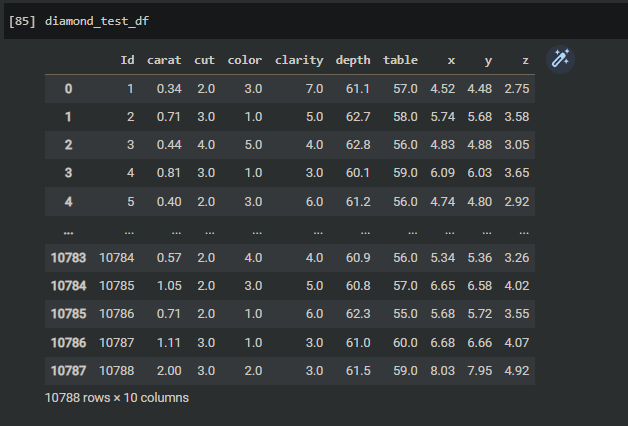
We used ordinal encoding since our categorical data is in order(from worst to best), one-hot encoding could lead to worse results.

After applying the ordinal encoding we display the train and test files again to see the changes.

The train data set:



The test data set:

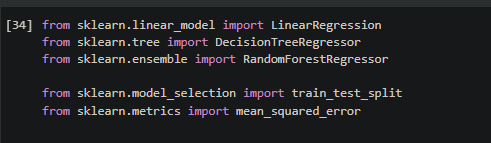


We notice above all categorical data become numerical data and we are ready to train our models.

**Part 3: Bulding And Training The models:**

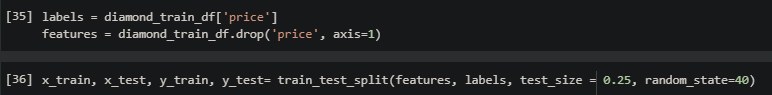
Now comes the most important part which’s building and training the model and lastly evaluating the model does it performs,

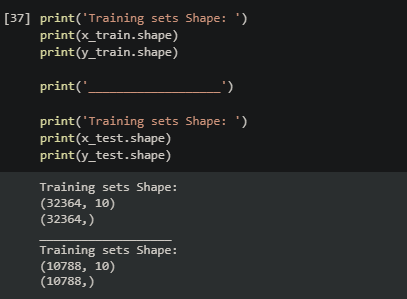
First of all we import the needed libraries:



Now we want to split our labels and features, so we put or copy the price column to a dataframe then drop it and add the remaining features in another dataframe.

Then we split our new data frames into train and test data frames using *trian\_test\_split()*, and we made the test size equal to 25%, higher than that would lead to bad results as the model won't have enough data to learn from, and the random state just so we keep having the same output each time we run our code

****

Then we check the shapes of our train and test data frames to make sure

Now the fun part begins which’s building and training the models!

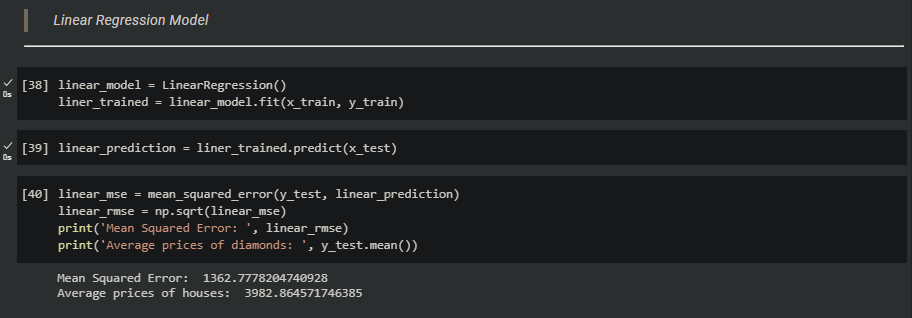
We will build three models in total, and they are:

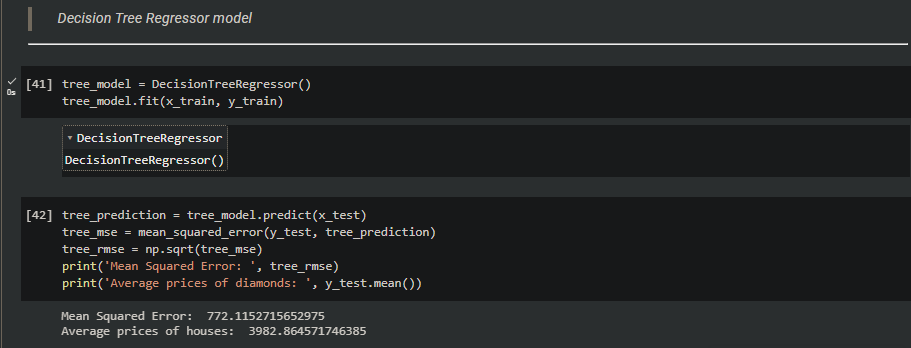
1. *Linear Regression*
2. *Decision Tree Regressor model*
3. *RandomForestRegressor*

and use the best one with the lowest root mean square error(rmse) value to predict the prices of the diamonds.

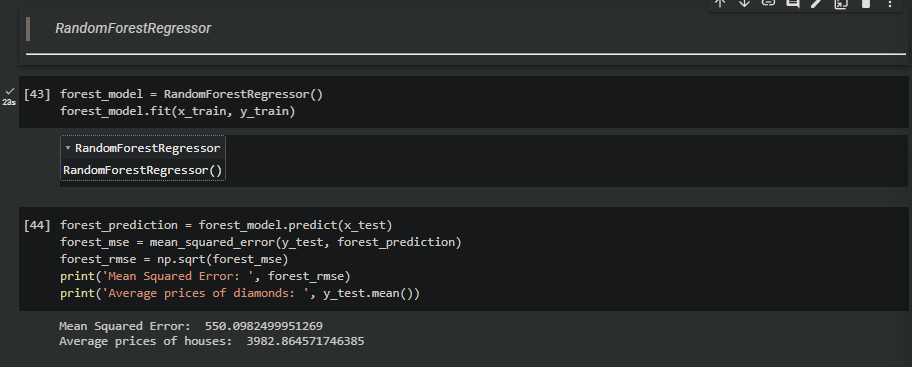
As for the process of training the model it’s the same for all the models, we make an instance of the model, then we fit it using the train datasets(y, x), and lastly we predict the values using the test dataset of x.

As for the evaluation we are using rmse for this project, so we give *mean\_squared\_error()* function the prediction that we got above and the y labels (y\_test), then we use numpy to square root the value and compare it to the mean value of the labels.

* ***Linear Regression*****:**
* ***Decision Tree Regressor*:**

****

* ***Random Forest Regressor*:**

****

As we can see, the linear model gave the worst results with 34% error rate, then came the decision tree model which was a lot better with 19% error rate, and finally random forest being the best with 13% error rate.

So now that random forest being the best model we will use it on the test data set to predict the prices of diamonds, and lastly we copy the price prediction to a csv file with the Id column.

