



**TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY**

**FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY  
ACADEMIC YEAR 2023/2024 Session 202305**

**BACS3013 DATA SCIENCE: Assignment Documentation**

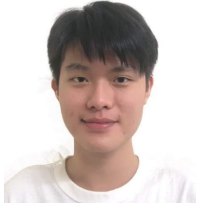


**Title: Diamonds**

**Programme and Tutorial Class: RDS Group 3**

**Tutor Name: Professor NOOR AIDA BINTI HUSAINI**

**Submission Date: 17/9/2023**

**Team Member:**

 <p><b>Student Name: Yam Jason</b> <b>Student ID: 22WMR13662</b> <b>Module-in-Charge:</b> <b>1. Random Forest Model</b></p>	 <p><b>Student Name: Wong Yee En</b> <b>Student ID: 22WMR13659</b> <b>Module-in-Charge:</b> <b>1. Decision Tree Model</b> <b>2. Gradient Boosting Regression Model</b></p>
 <p><b>Student Name: Ashantha Rosary James K Arokiasamy</b> <b>Student ID: 22WMR14161</b> <b>Module-in-Charge:</b> <b>1. K Neighbours Regression Model</b></p>	

## TABLE OF CONTENT

<b>1. BUSINESS UNDERSTANDING</b>	<b>4</b>
1.1 Problem Background	4
1.2 Aim	4
1.3 Objectives	5
1.4 Motivation	5
1.5 Timeline/Milestone	6
<b>2. DATA UNDERSTANDING</b>	<b>7</b>
2.1 Analysis of the selected tool with any other relevant tools	7
2.2 Overview of the chosen dataset	10
2.3 Visualization of each attribute	15
2.4 Visualization of the relationship between numerical attributes and the ‘y label’	25
2.5 Outliers detection	31
2.6 Correlation Heatmap	32
<b>3. DATA PREPARATION</b>	<b>33</b>
3.1 Removing Duplicates	33
3.2 Removing Dimensionless Diamond Records	34
3.3 Removing Outliers	34
3.4 Handling Categorical Variables	36
3.5 Correlation Heatmap	38
3.6 Splitting Datasets	39
3.7 Scaling	40
<b>4. MODELING</b>	<b>41</b>
4.1 Decision Tree Regression Model	42
4.2 Random Forest Regression Model	46
4.3 Gradient Boosting Regression Model	49
4.4 K Neighbours Regression Model	52
<b>5. EVALUATION</b>	<b>56</b>
5.1 Results	56
5.2 R-squared	58
5.3 Adjusted R-squared	59
5.4 Mean Absolute Error (MAE)	60
5.5 Mean Squared Error (MSE)	61
5.6 Root Mean Squared Error (RMSE)	62
5.7 Achievements	62
5.8 Discussions	63
<b>6. DEPLOYMENT</b>	<b>64</b>

<b>7. CONCLUSION</b>	<b>67</b>
7.1 Advantages and disadvantages of models	67
7.2 Thoughts	68
<b>References</b>	<b>69</b>
<b>APPENDIX</b>	<b>70</b>

# 1. BUSINESS UNDERSTANDING

## 1.1 Problem Background

Shiny Diamond Company has been a key player in the global diamond industry since its establishment in 1978. Specializing in sourcing, manufacturing, and distributing high-quality diamonds, the company has consistently maintained a stellar reputation for excellence and innovation within the gemstone sector. From the very beginning, Shiny Diamond Company's primary mission has been to provide its customers with the most exquisite diamonds available in the market. One of the factors that sets Shiny Diamond Company apart is its exceptional expertise in dealing with diamonds of various sizes, particularly its proficiency in understanding and handling diamond carat weights. This has made them highly respected and well-known in the industry for their ability to work with diamonds of all sizes.

Shiny Diamond Company takes its commitment to delivering exceptional services and top-tier product quality very seriously. To ensure that they remain at the forefront of the diamond industry, they have embraced modern technology, particularly machine learning algorithms. These algorithms play a crucial role in the company's operations. By analyzing visual data, they can accurately predict diamond prices and detect even the tiniest imperfections. This level of precision enables Shiny Diamond Company to grade diamonds with remarkable accuracy, ensuring that each diamond they offer not only meets but often exceeds the expectations of their customers.

Shiny Diamond Company's dedication to continuous improvement through machine learning algorithms serves the ultimate goal of providing customers with the best possible experience. By predicting diamond prices accurately, they can offer competitive prices to their customers. Their ability to detect imperfections with precision means that customers can trust the quality of their diamonds. This commitment to excellence and innovation allows Shiny Diamond Company to maintain its position as a trusted and respected name in the diamond industry, assuring customers that they are receiving the finest diamonds that not only meet their expectations but also provide exceptional value for their investment.

## 1.2 Aim

The aim of our project is to develop a robust predictive model that predicts the price of diamonds.

## 1.3 Objectives

In order to achieve the aim, the first objective is to collect a comprehensive dataset containing relevant features and diamond price information. This dataset should be carefully cleaned, processed, and preprocessed to handle missing values, outliers, and any inconsistencies. Specific tasks include feature selection, scaling, and encoding categorical variables to ensure the data is ready for machine learning analysis. A clean and well-structured dataset is essential for training an accurate predictive model.

The second objective is to find the best model by identifying the R square (main performance metric), Adjusted R square, Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE). We will be using 4 regression models for comparison which are the Decision Tree Regression Model, Random Forest Regression Model, Gradient Boosting Regression Model, and K Neighbors Regression Model.

The third objective is to deploy the selected predictive model with the best performance into a practical and user-friendly application for Shiny Diamond Company. This deployment involves integrating the model into an existing system that allows stakeholders and customers to input diamond information and receive real-time price predictions.

## 1.4 Motivation

Firstly, it can help to accurately predict diamond prices for businesses in the diamond industry by optimizing the pricing strategies. This can help the company to increase its profit and offer competitive prices to its customers while maintaining a healthy profit margin.

Moreover, it can help the company to assist investors in making informed decisions about buying and selling diamonds as assets. This can stimulate investment in the diamond market, potentially increasing the marketability and value of diamonds as alternative investments.

Furthermore, it can help consumers to be more transparent about the pricing in the diamond market. This can help to prevent overpricing and make sure consumers can buy diamonds at reasonable prices.

Lastly, it can help consumers make more ethical and socially responsible choices when buying diamonds by integrating data from legitimate sources.

## 1.5 Timeline/Milestone

PROCESS	AUG 1- 3	AUG 4 - 12	AUG 13 - 19	AUG 20 - 26	AUG 27 - 31	SEPT 1 - 7
Business Understanding						
Data Understanding						
Data Preparation						
Modeling						
Evaluation						
Deployment						

## 2. DATA UNDERSTANDING

### 2.1 Analysis of the selected tool with any other relevant tools

Tools comparison	Remark	Jupyter Package	Numpy	Scikit-learn	Pandas
Type of license and open source license	State all types of license	Open-source (modified BSD)	Open-source (BSD)	Open-source (BSD)	Open-source (BSD)
Year founded	When is this tool being introduced?	2014	2005	2007	2008
Founding company	Owner	Fernando Pérez and Brian Granger.	Travis Oliphant	INRIA(French Institute for Research in Computer Science and Automation)	Wes McKinney
License Pricing	Compare the prices if the license is used for development and business/commercialization	Free	Free	Free	Free
Supported features	What features that it offers?	- It offers a collaborative environment where users can create and share documents with live code, equations, graphics, and text.	-NumPy provides functions to create arrays from various sources, including Python lists, tuples, and other	-Data preprocessing tools like scaling, normalization, encoding categorical variables, and handling missing values	-In order to enhance model performance and decrease dimensionality, the library provides

		<ul style="list-style-type: none"> <li>- Python, R, Julia, and other programming languages are among those supported by Jupyter.</li> </ul>	<p>array-like objects.</p> <ul style="list-style-type: none"> <li>-Numerous mathematical operations, such as arithmetic, trigonometric, exponential, and logarithmic functions, are available on arrays using NumPy.</li> </ul>	<p>are all provided.</p> <ul style="list-style-type: none"> <li>-In order to enhance model performance and decrease dimensionality, the library provides tools for feature selection, extraction, and transformation.</li> </ul>	<p>tools for feature selection, extraction, and transformation.</p> <ul style="list-style-type: none"> <li>-With the help of Pandas' robust indexing features, you can choose, filter, and alter data based on labels or positional indices.</li> </ul>
Common applications	In what areas is this tool usually used?	<ul style="list-style-type: none"> <li>-Data analysis</li> <li>-Scientific computing</li> </ul>	<ul style="list-style-type: none"> <li>-Scientific</li> <li>-Computing</li> </ul>	<ul style="list-style-type: none"> <li>-Machine learning</li> <li>-Data mining</li> </ul>	<ul style="list-style-type: none"> <li>-Data analysis</li> <li>-Manipulation</li> </ul>
Customer support	How the customer support is given, e.g. proprietary, online community, etc.	<ul style="list-style-type: none"> <li>-Online community</li> <li>-Commercial support</li> </ul>	<ul style="list-style-type: none"> <li>- Online community</li> <li>-Commercial support</li> </ul>	<ul style="list-style-type: none"> <li>-Online community</li> <li>-Commercial support</li> </ul>	<ul style="list-style-type: none"> <li>-Online community</li> <li>-Commercial support</li> </ul>



Limitations	The drawbacks of the software	<p>-Limited to Python-based languages</p> <p>-Less secure</p> <p>-Long asynchronous tasks are challenging to test.</p> <p>-No linting, no code-style correction, and no IDE integration are present.</p> <p>-State is shared between cells in a Jupyter Notebook, which can occasionally result in unexpected behaviour if cells are not executed sequentially or if variables or outputs from earlier cells are kept in memory.</p>	<p>-Because NumPy arrays are mutable, changing an array's elements may have unanticipated effects when the same array is used multiple times.</p> <p>-Due to its emphasis on numerical calculations, NumPy lacks some of the more advanced data analysis features offered by libraries like Pandas.</p> <p>-Because NumPy arrays are homogeneous, they can only hold elements of the same type of data. It may become less adaptable when dealing with heterogeneous data due to this restriction.</p>	<p>-Due to the lack of built-in GPU acceleration for training models, Scikit-learn's performance may be constrained when dealing with large datasets and intricate models that would benefit from GPU parallelism.</p> <p>-Scikit-learn can handle a wide range of tasks, but because of memory and computational constraints, it may have trouble processing extremely large datasets.</p> <p>-Basic interpretability features are provided by some models in Scikit-learn (such as feature importances for decision trees), but more sophisticated</p>	<p>-While Pandas supports time series, it might not be as feature-rich or fast as databases or libraries that are specifically designed for time series.</p> <p>-Pandas is intended for batch data processing, so it might not be the best tool for dealing with high-frequency data or real-time data streams.</p> <p>-Pandas uses mutable DataFrames, and when working with multiple references to the same DataFrame, modifications to DataFrames can occasionally have</p>
-------------	-------------------------------	--	--	--	--

				interpretability techniques might call for different tools or libraries.	unexpected side effects.
--	--	--	--	--	--------------------------

## 2.2 Overview of the chosen dataset

Variables Name	Type	Dependent/ Independent variables	Description
carat	Numeric	Independent variable	The unit of measurement used to quantify the weight of diamond
cut	Categorical	Independent variable	Fair, Good, Very Good, Premium, Ideal The cut grade is determined by how effectively the diamond reflects and refracts light.
color	Categorical	Independent variable	from J (worst) to D (best) The color of the diamond.
clarity	Categorical	Independent variable	I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best) Inclusions and blemishes are features that can be found in diamonds, with inclusions being internal and blemishes being external. It's quite uncommon to find diamonds that are completely free from these features, but in

			most cases, they are only visible when examined under magnification.
depth	Numeric	Independent variable	total depth percentage = $z / \text{mean}(x, y) = 2 * z / (x + y)$ (43--79)
table	Numeric	Independent variable	The term "diamond table" denotes the flat facet of a diamond that becomes visible when the diamond is viewed face-up. Its primary function is to bend incoming light rays and enable the reflected light rays from inside the diamond to reach the observer's eye. In the case of an ideally cut diamond table, it enhances the diamond's dazzling sparkle and brilliance.
price	Numeric	Dependent variable	Price in US dollars (\$326--\$18,823)
x	Numeric	Independent variable	length in mm
y	Numeric	Independent variable	width in mm
z	Numeric	Independent variable	depth in mm

```
In [2]: data = pd.read_csv("diamonds.csv")
data.head()
```

Out[2]:

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
0	0.23	b'Ideal'	b'E'	b'SI2'	61.5	55.0	326.0	3.95	3.98	2.43
1	0.21	b'Premium'	b'E'	b'SI1'	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	b'Good'	b'E'	b'VS1'	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	b'Premium'	b'I'	b'VS2'	62.4	58.0	334.0	4.20	4.23	2.63
4	0.31	b'Good'	b'J'	b'SI2'	63.3	58.0	335.0	4.34	4.35	2.75

**Figure 2.1:** Load and display the first 5 rows of the dataset.

```
In [7]: data.tail()
```

Out[7]:

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
53935	0.72	b'Ideal'	b'D'	b'SI1'	60.8	57.0	2757.0	5.75	5.76	3.50
53936	0.72	b'Good'	b'D'	b'SI1'	63.1	55.0	2757.0	5.69	5.75	3.61
53937	0.70	b'Very Good'	b'D'	b'SI1'	62.8	60.0	2757.0	5.66	5.68	3.56
53938	0.86	b'Premium'	b'H'	b'SI2'	61.0	58.0	2757.0	6.15	6.12	3.74
53939	0.75	b'Ideal'	b'D'	b'SI2'	62.2	55.0	2757.0	5.83	5.87	3.64

**Figure 2.2:** Display the last 5 rows of the dataset.

```
data.shape
```

```
(53940, 10)
```

**Figure 2.3:** Display the dataset's shape.

**Insight:** Our dataset has 53940 rows and 10 columns.

```
In [72]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  -
 0   carat       53940 non-null  float64
 1   cut         53940 non-null  object  
 2   color       53940 non-null  object  
 3   clarity     53940 non-null  object  
 4   depth       53940 non-null  float64
 5   table       53940 non-null  float64
 6   price       53940 non-null  float64
 7   'x'         53940 non-null  float64
 8   'y'         53940 non-null  float64
 9   'z'         53940 non-null  float64
dtypes: float64(7), object(3)
memory usage: 4.1+ MB
```

**Figure 2.4:** Display the dataset's information.

**Insight:** We have 10 attributes with the records without any missing values. The probability of data input errors or omissions was reduced by the efficient design and implementation of the data collection procedure. (Collecting Data | CYFAR, n.d.)

```
In [69]: data.describe()

Out[69]:
```

	carat	depth	table	price	'x'	'y'	'z'
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

**Figure 2.5:** Display the statistical information of the numerical attributes.

**Insight:** We notice that the minimum values of attributes 'x', 'y', and 'z' are zero, which is impossible for 3-dimensional diamonds. So, we assume it is caused by human input error and will remove these outliers.

```
In [65]: data.duplicated().sum()
```

```
Out[65]: 146
```

```
In [67]: duplicate_rows = data[data.duplicated(keep=False)]
```

```
In [68]: # Print the duplicate rows
print(duplicate_rows)
```

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
1004	0.79	b'Ideal'	b'G'	b'SI1'	62.3	57.0	2898.0	5.90	5.85	3.66
1005	0.79	b'Ideal'	b'G'	b'SI1'	62.3	57.0	2898.0	5.90	5.85	3.66
1006	0.79	b'Ideal'	b'G'	b'SI1'	62.3	57.0	2898.0	5.90	5.85	3.66
1007	0.79	b'Ideal'	b'G'	b'SI1'	62.3	57.0	2898.0	5.90	5.85	3.66
1008	0.79	b'Ideal'	b'G'	b'SI1'	62.3	57.0	2898.0	5.90	5.85	3.66
...	...	...	...	...	...	...	...	...	...	...
49557	0.71	b'Good'	b'F'	b'SI2'	64.1	60.0	2130.0	0.00	0.00	0.00
50078	0.51	b'Ideal'	b'F'	b'VVS2'	61.2	56.0	2203.0	5.19	5.17	3.17
50079	0.51	b'Ideal'	b'F'	b'VVS2'	61.2	56.0	2203.0	5.19	5.17	3.17
52860	0.50	b'Fair'	b'E'	b'VS2'	79.0	73.0	2579.0	5.21	5.18	4.09
52861	0.50	b'Fair'	b'E'	b'VS2'	79.0	73.0	2579.0	5.21	5.18	4.09

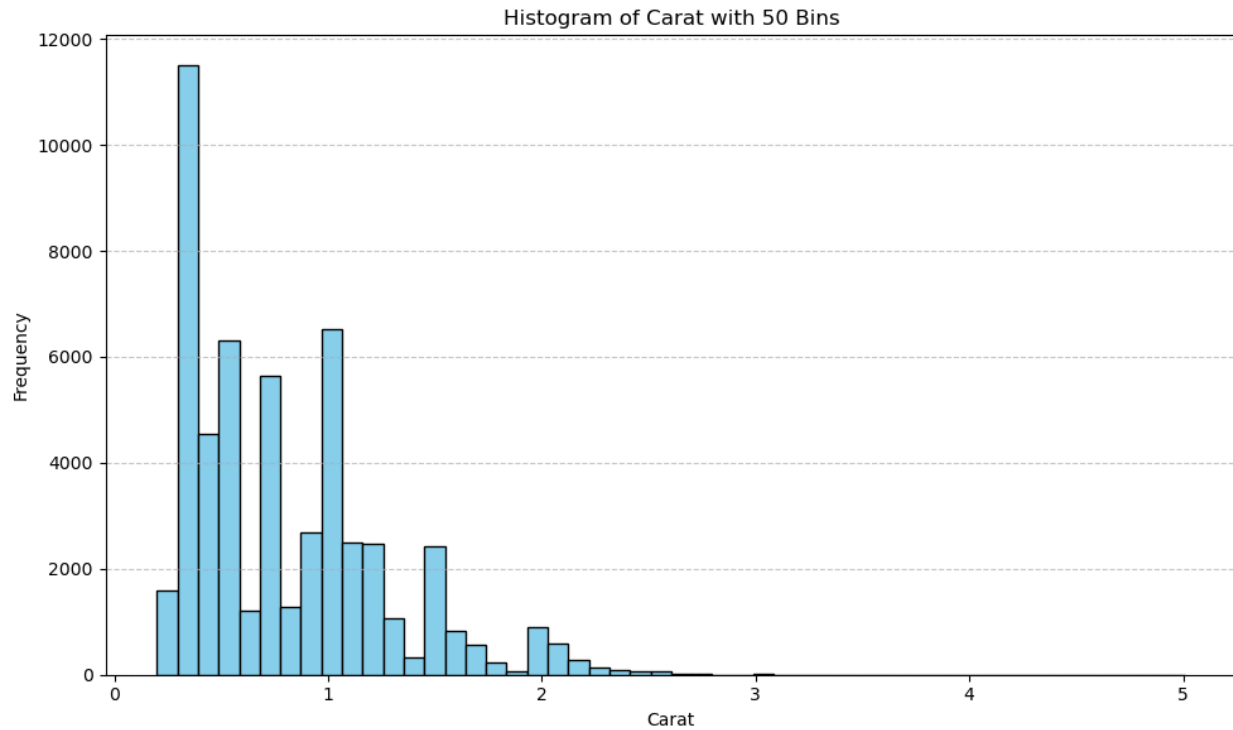
[289 rows x 10 columns]

**Figure 2.6:** Show and display the number of duplicated records.

**Insight:** It shows that there are 146 duplicated records which we will be removing during the data preparation step. Duplicate records exist as a result of problems with software system operations for data storage, retrieval, or export.

## 2.3 Visualization of each attribute

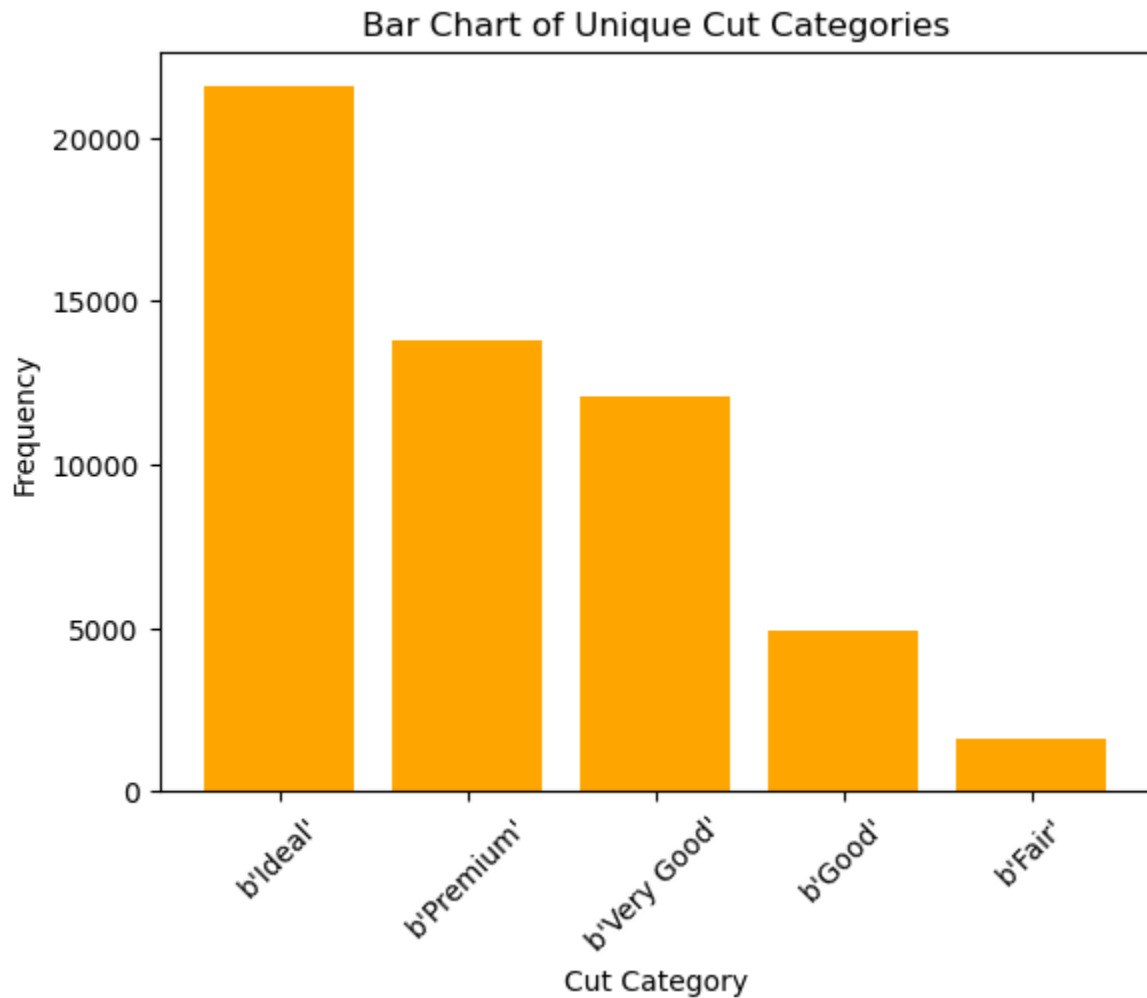
### 1. Carat



**Figure 2.7:** The histogram of carat attribute with a bin size of 50.

**Insight:** The histogram above shows a positively skewed distribution because of the longer tail on the right side of the distribution, with most of the data concentrated on the left. This means that smaller diamonds are more prevalent than larger ones. Only a few of the diamonds available are extraordinarily huge; most are rather small in carat weight. It is clear that diamonds weighing about 0.3 carats are the most highly demanded. In comparison to larger diamonds, smaller diamonds, such as those under 0.3 carats, are more reasonably priced for the average buyer.

## 2. Cut

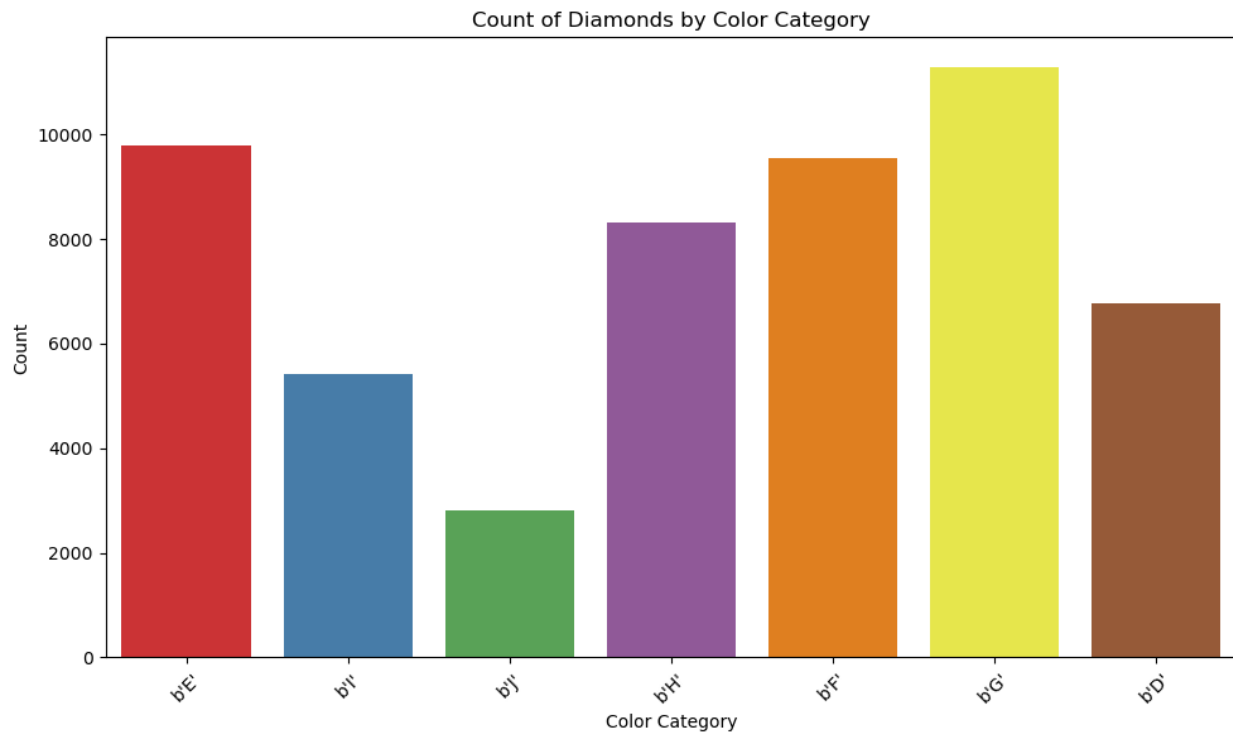


**Figure 2.8:** The bar chart of the cut attribute.

**Insight:** The bar chart above could reflect a current market trend where consumers prefer diamonds with an "Ideal" cut, perhaps due to their superior sparkle and brilliance. This cut maximizes the reflection and refraction of light within the diamond, resulting in a stunning display of brightness and sparkle. Consumers often prioritize the visual appeal of a diamond, making "Ideal" cuts highly desirable.



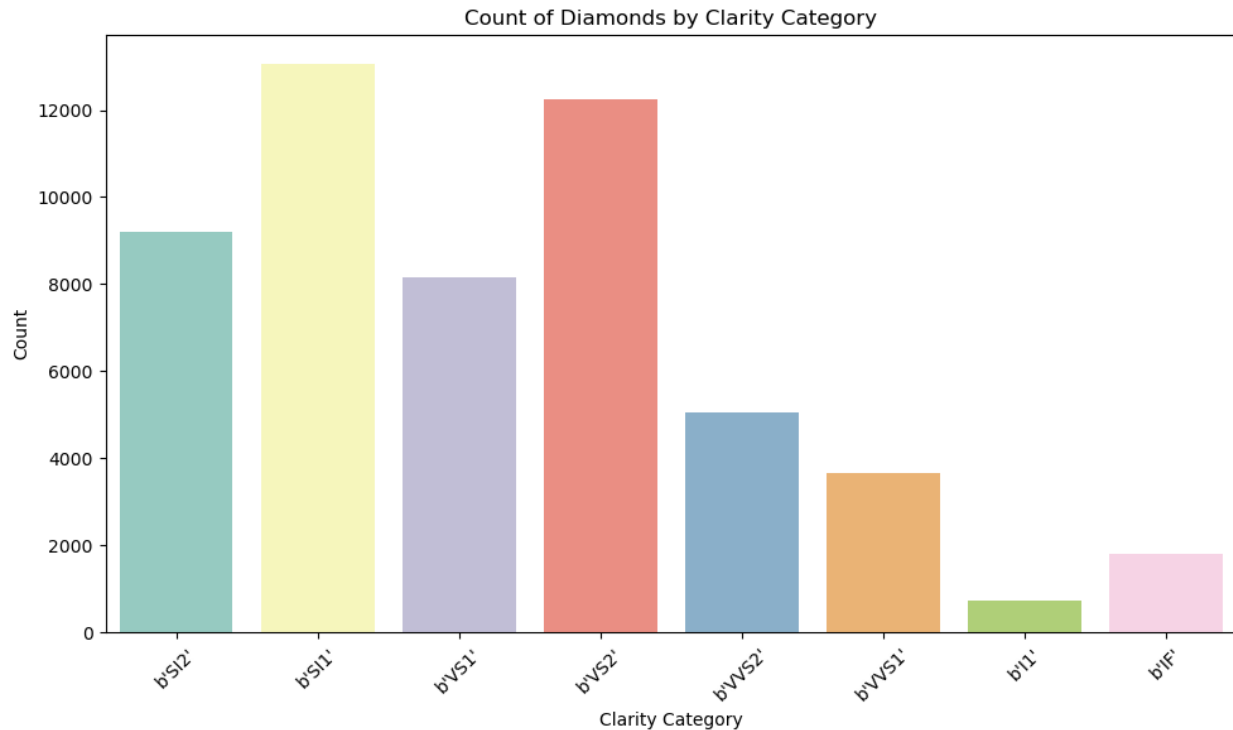
### 3. Color



**Figure 2.9:** The bar chart of the color attribute.

**Insight:** The bar chart shows that the number of diamonds with the color b'G' is the highest as compared to the other colors. In the nearly colorless color range, G-color is the highest grade of diamond color. (G Color Grade Diamonds, n.d.) So, customers tend to buy colorless diamonds which will look more high-class.

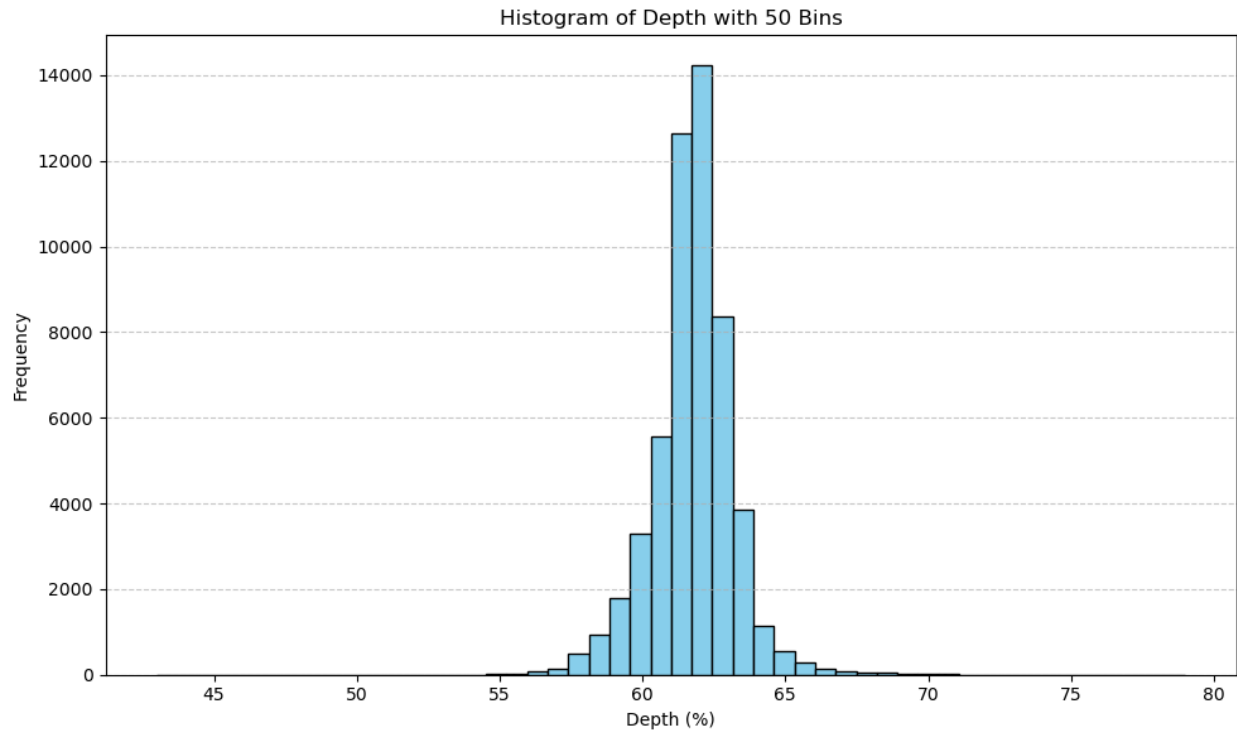
#### 4. Clarity



**Figure 2.10:** The bar chart of the clarity attribute.

**Insight:** The bar chart above shows the number of diamonds with a clarity of b'SI1' is the highest as compared to the other categories while the number of diamonds with a clarity of b'I1' is the lowest. This shows that consumers prefer diamonds with b'SI1' clarity over those with other clarity grades. This may be because "SI1" diamonds typically have small flaws that cannot be seen with the naked eye. They have a nearly as pristine appearance as those with higher clarity grades (such as "VS1" or "VVS1") but cost less because of this. (Is an SI1 Clarity Diamond a Good Choice? - International Gem Society, 2023) Customers usually seek a balance between affordability and quality.

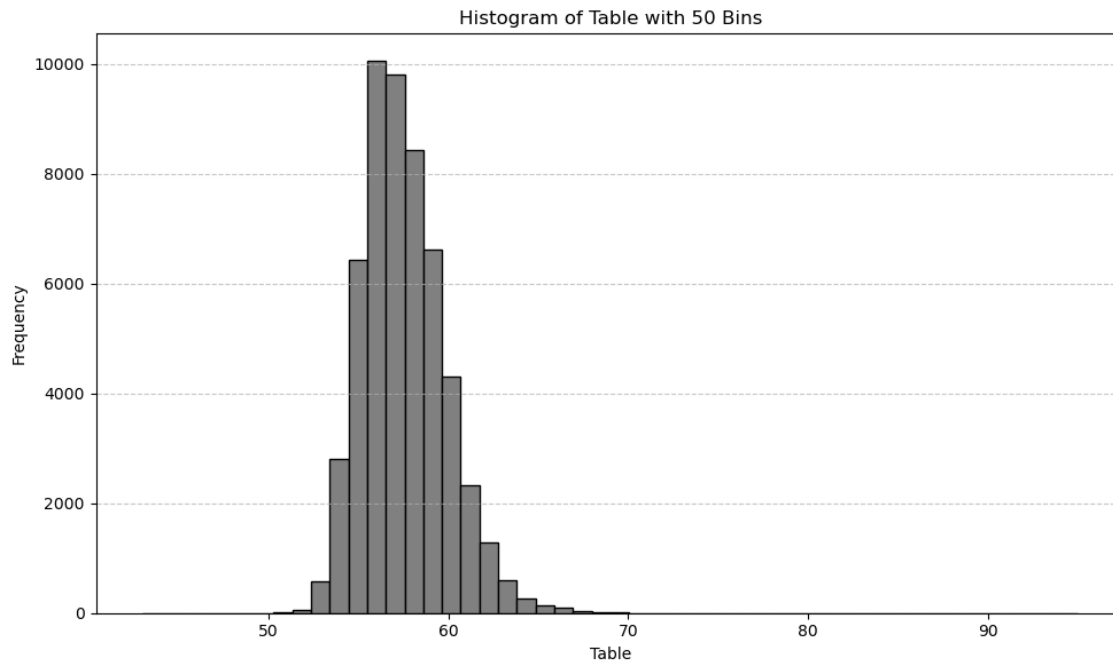
## 5. Depth



**Figure 2.11:** The histogram for depth attribute.

**Insight:** The range between 60% and 65% for the total depth percentage of diamonds in the histogram's normal distribution suggests that this range is typical and highly favored in the business. The reason for this is that diamonds in this price category are often cut to enhance their brilliance, conform to quality requirements, and match consumer preferences for balanced proportions.

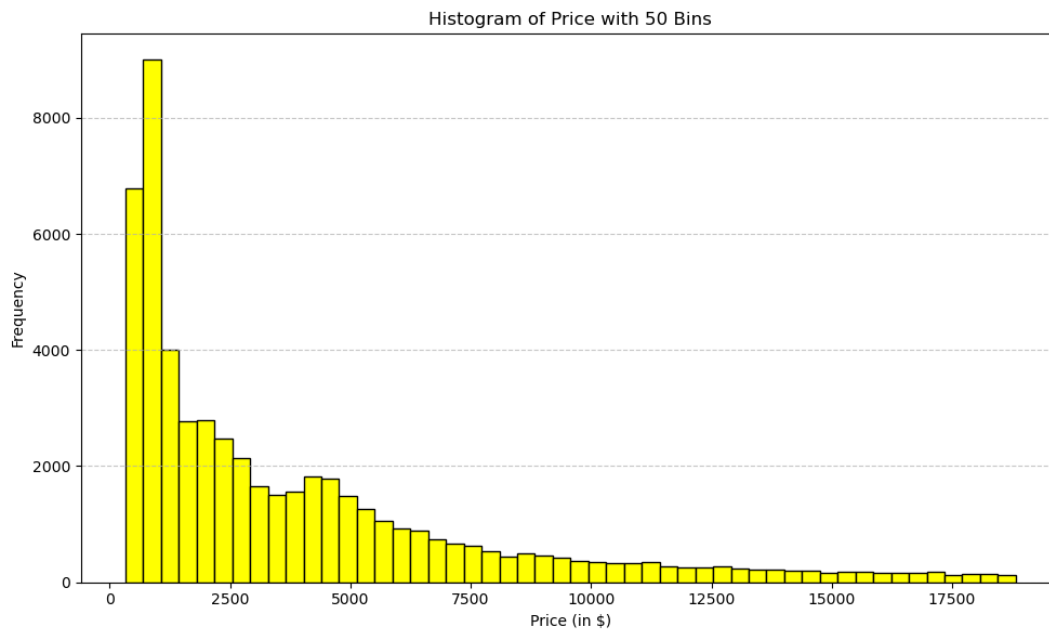
## 6. Table



**Figure 2.12:** The histogram of table attribute.

**Insight:** The histogram shows that the table of diamonds ( Width of the top of the diamond relative to the widest point) exists most frequently around 57. Many diamonds in the dataset have a table width close to 57 because this measurement is often considered visually appealing and balances the diamond's brilliance. It aligns with industry standards and meets market demand for attractive and versatile diamonds suitable for various jewelry designs. Quality control measures help maintain consistency in this specification.

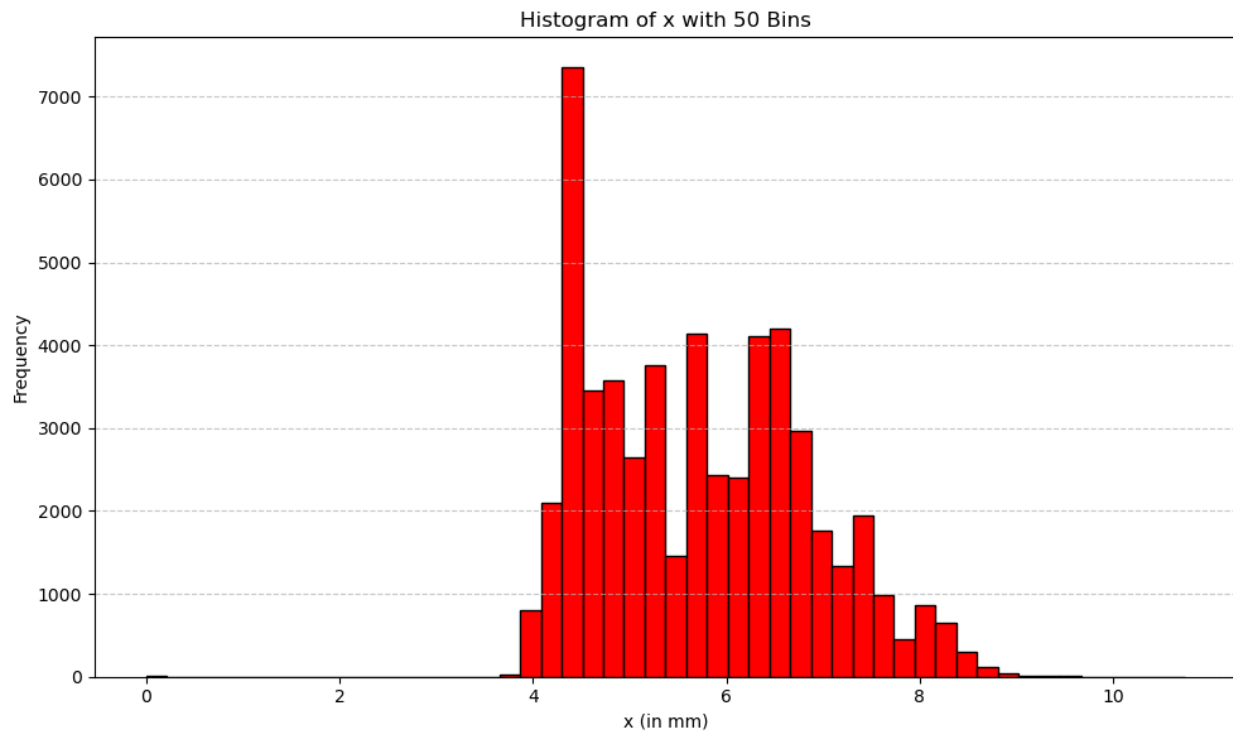
## 7. Price



**Figure 2.13:** The histogram for price attribute.

**Insight:** The histogram shows a downward trend. It may be because intense competition among sellers can lead to price reductions as businesses try to attract customers with lower prices.

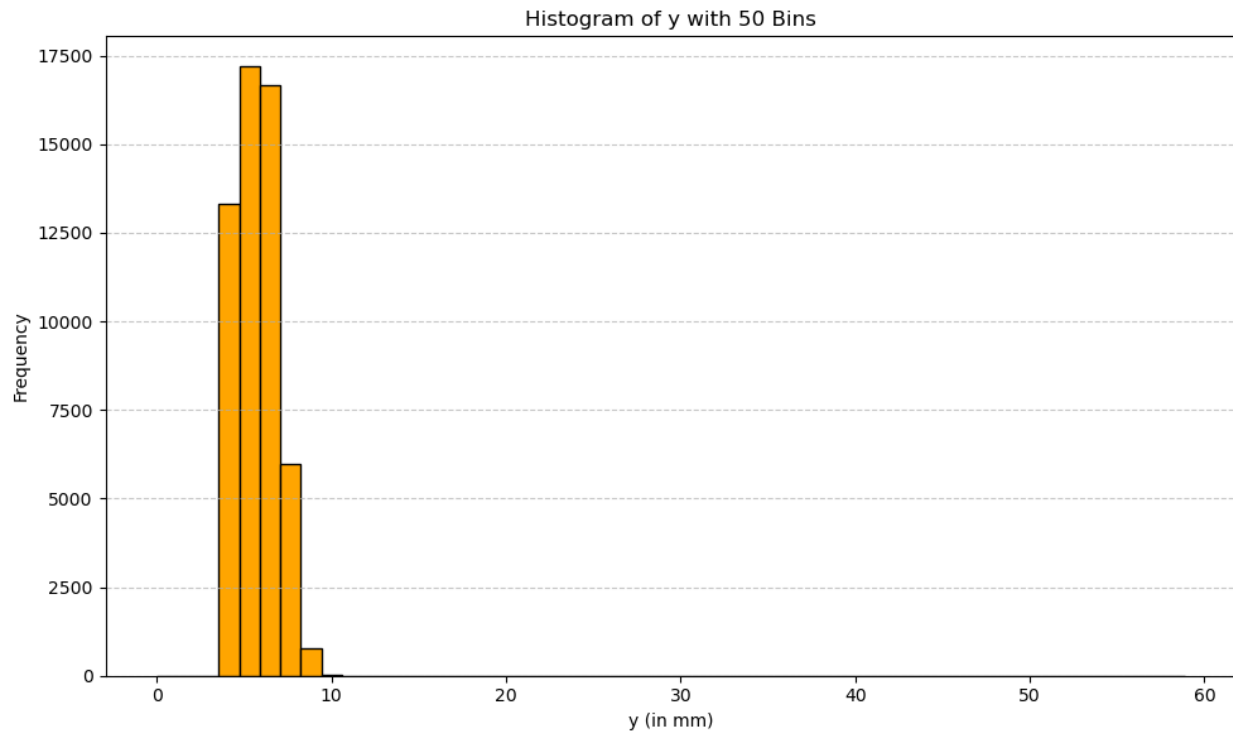
## 8. X (X fracture of the diamond)



**Figure 2.14:** The histogram of the x attribute.

**Insight:** The histogram demonstrates that the dataset's diamonds range in length (x) from about 4 to 9 millimeters. For x, the peak frequency occurs between 4 and 5 millimeters, indicating that this size range is especially well-liked. This choice might be influenced by consumer demand for adaptable, reasonably priced diamonds that work well in a variety of jewelry styles. By manufacturing additional diamonds in this size range, diamond producers might also satisfy this need.

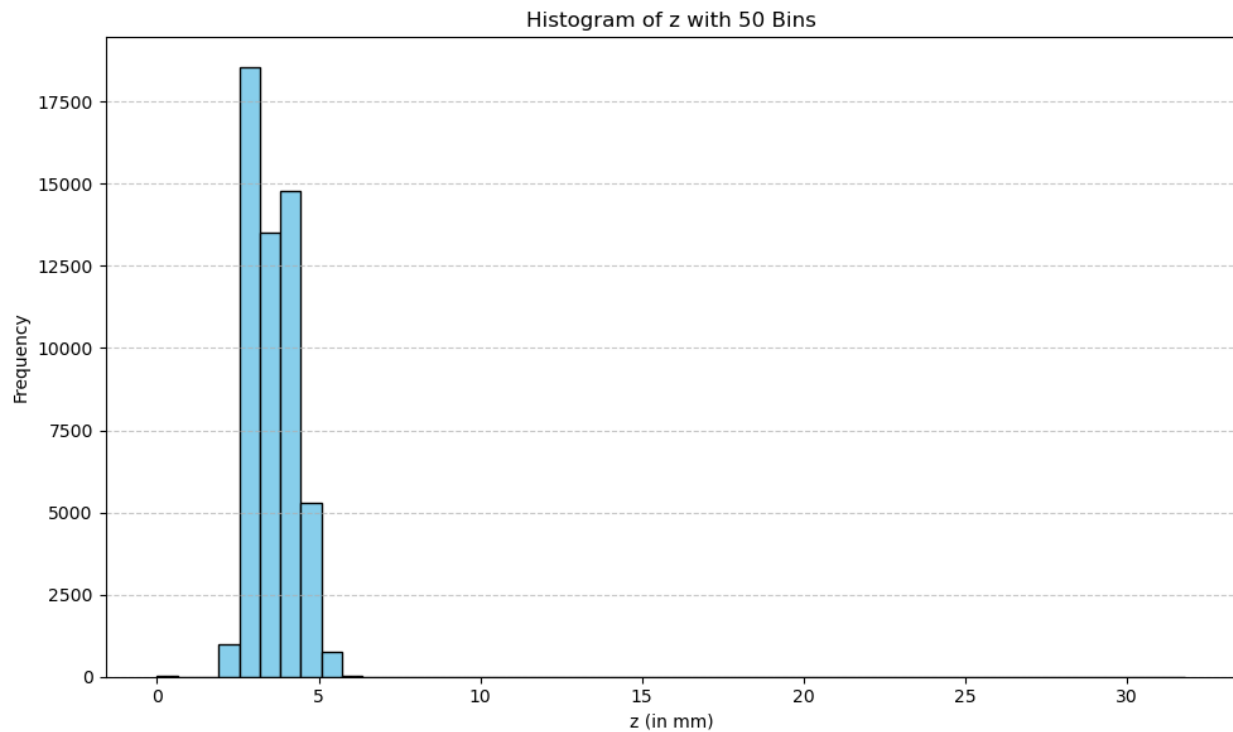
## 9. Y (Y fracture of the diamond)



**Figure 2.15:** The histogram of the y attribute.

**Insight:** The width (y) of diamonds, which ranges from around 4mm to 10mm, is shown in the **Figure 2.15** histogram. The dataset contains many diamonds with a width close to this value, as indicated by the maximum frequency, which occurs slightly below 17,500. This concentration could be the result of manufacturing procedures and consumer preferences that favor this diamond width range.

## 10. Z (Z fracture of the diamond)



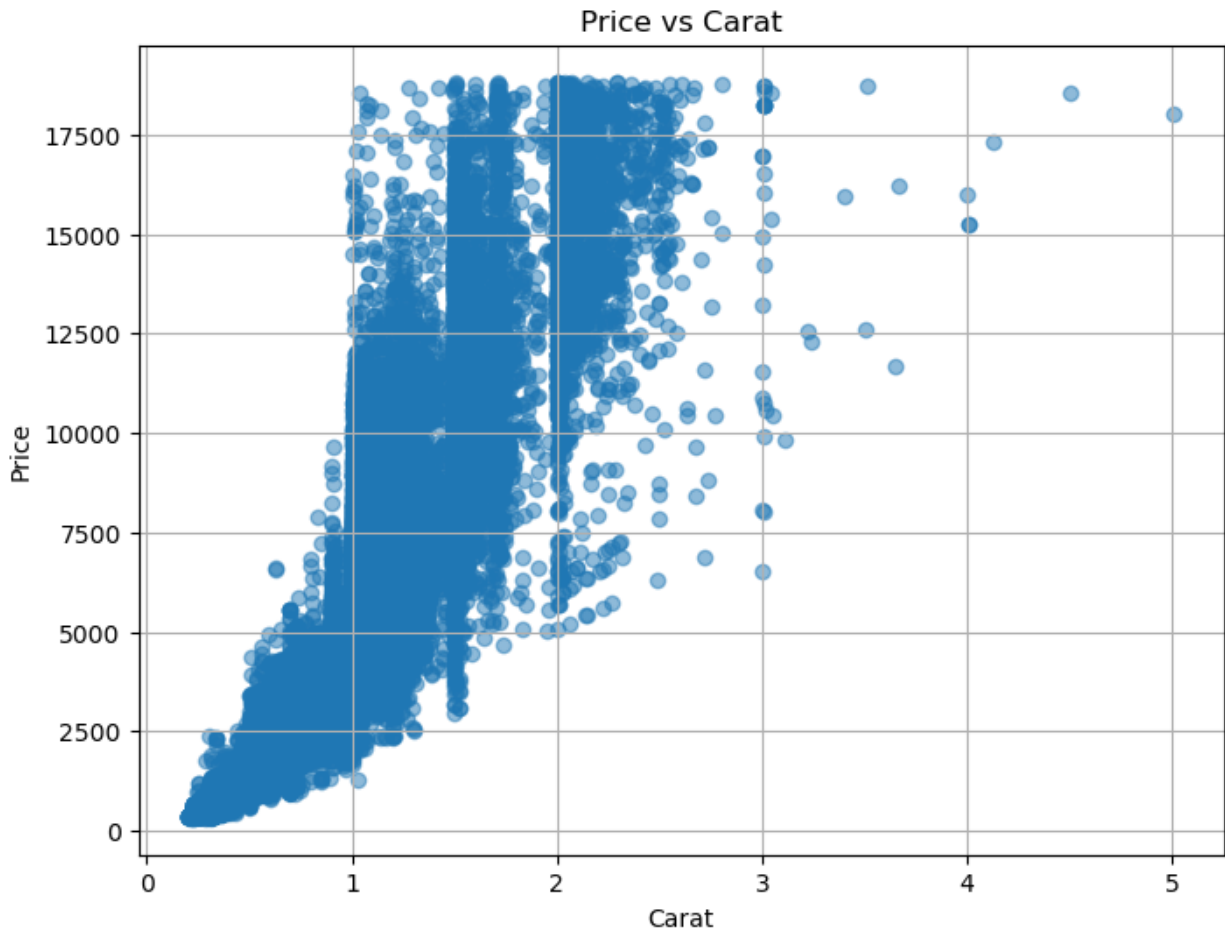
**Figure 2.16:** The histogram of z attribute.

**Insight:** The histogram in **Figure 2.16** shows the diamonds' depth (z), which ranges from about 4mm to 6mm. For z, the frequency measured in the 3mm to 4mm region is just over 17,500. This distribution points to either a widespread consumer desire for depth or typical manufacturing procedures in the diamond business as a major portion of the dataset's diamonds have a depth around this figure.



## 2.4 Visualization of the relationship between numerical attributes and the 'y label'

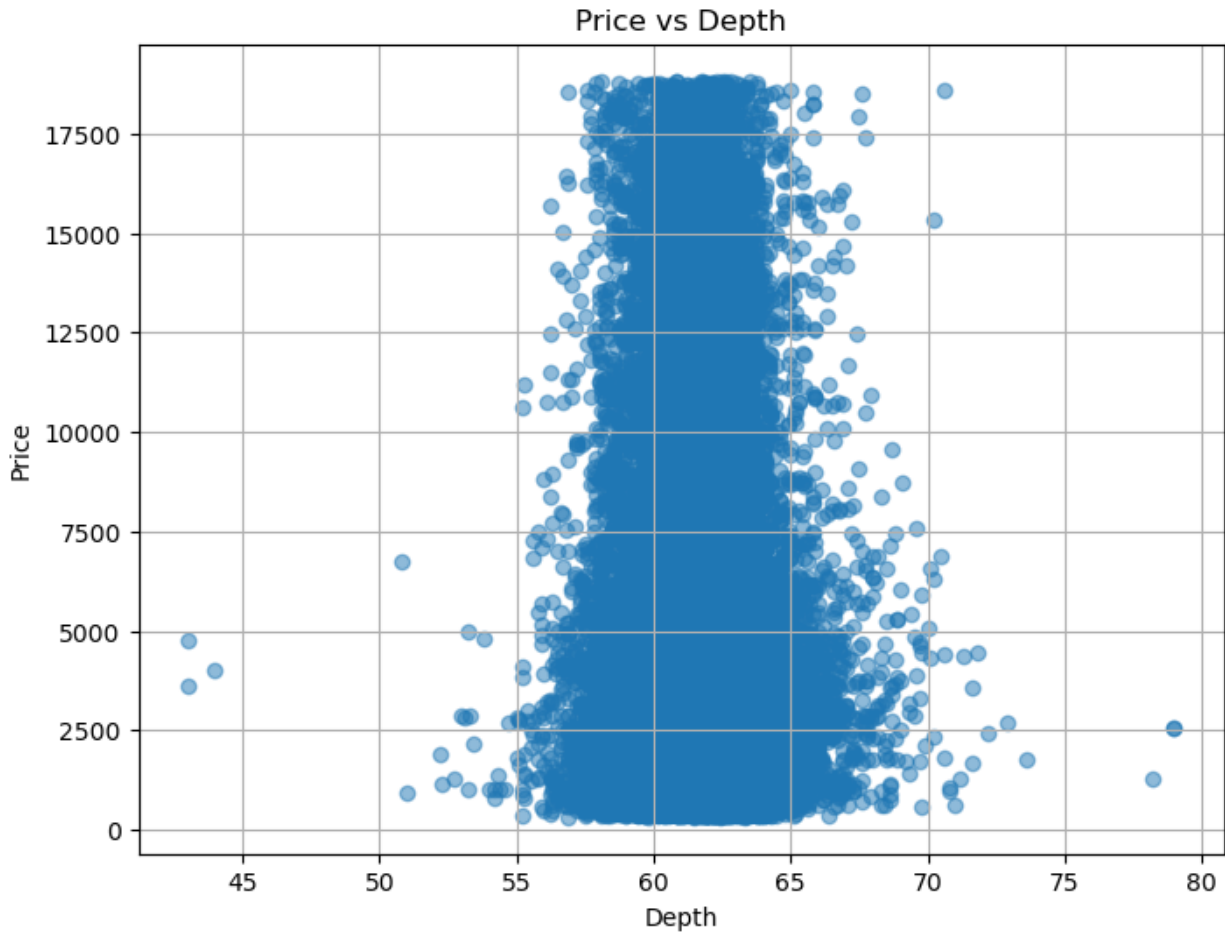
### 1. price vs. carat



**Figure 2.17:** The scatter plot of price vs. carat.

**Insight:** It shows that the attributes of price and carat are highly correlated. As the carat increases, the price also increases. The scarcity and value of bigger diamonds are the main causes of this. Larger diamonds are more expensive because they are less prevalent in nature and require more thorough cutting and polishing. Additionally, larger diamonds cost more since they look more impressive and are frequently valued higher.

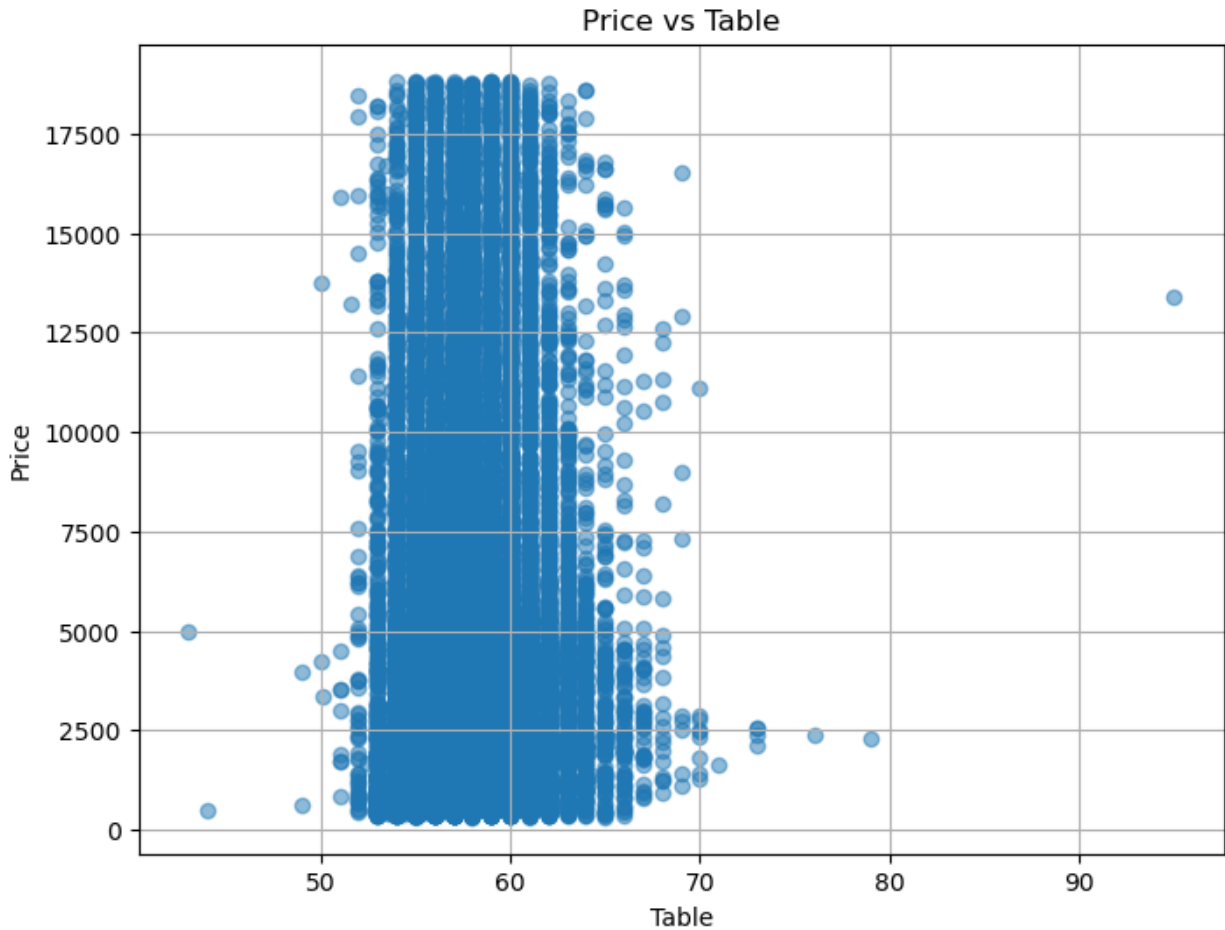
## 2. price vs. depth



**Figure 2.18:** The scatter plot of price vs. depth.

**Insight:** It shows that the price and depth are lowly correlated. Because the depth percentage is not the main factor in determining a diamond's price, even if it is crucial for appearance and cut quality, there is only a minor correlation between price and depth. The value of a diamond is frequently more significantly influenced by other criteria such as carat weight, color, and clarity. As a result, changes in depth percentage frequently have little impact on cost, resulting in a weak relationship between the two characteristics.

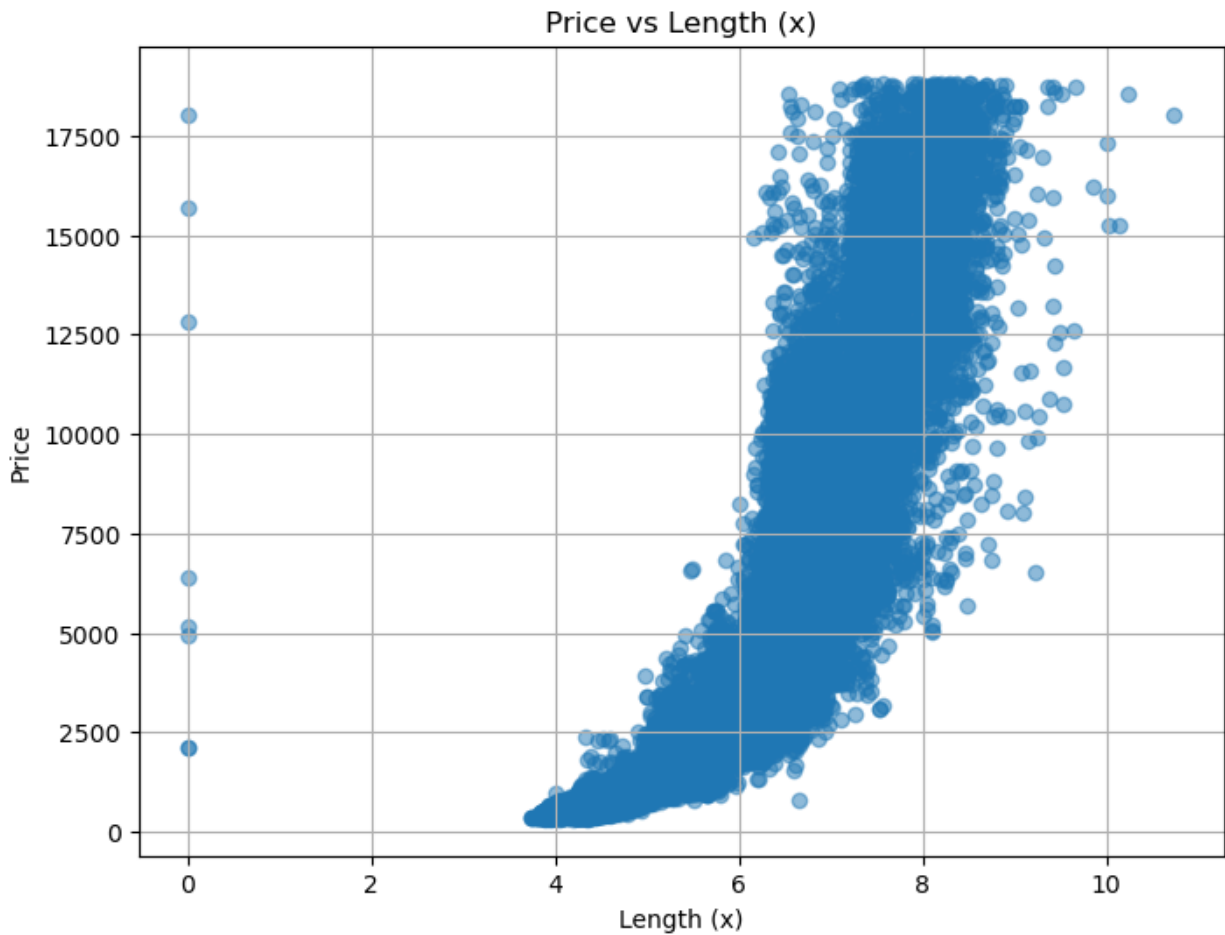
## 3. price vs. table



**Figure 2.19:** The scatter plot of price vs. table.

**Insight:** It shows that the price and table are lowly correlated. The low association between price and table percentage shows that changes in a diamond's table percentage have only a negligible effect on pricing. More important determinants of a diamond's price are its size (carat weight), color, and clarity. Although the table percentage affects cost, it has less of an impact on appearance, which is why there is a lower correlation between the two aspects.

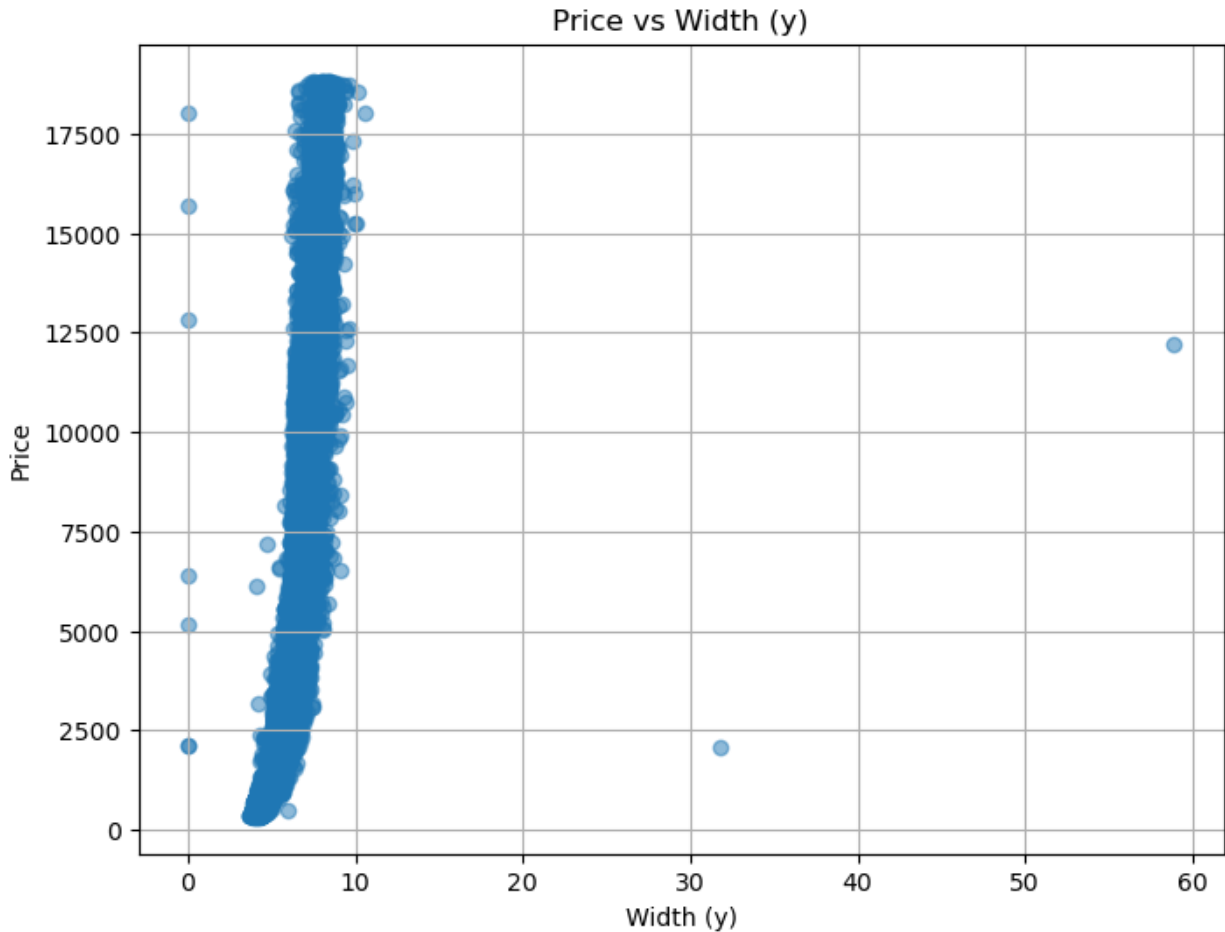
## 4. price vs. x



**Figure 2.20:** The scatter plot of price vs. x.

**Insight:** It shows that the attributes of price and x are highly correlated. As the x increases, the price also increases. Price and "x" have a strong positive correlation because larger diamonds with larger "x" dimensions are often more expensive in the diamond market. This is mainly because they are more expensive and have higher carat weights, aesthetic appeal, uniqueness, and quality.

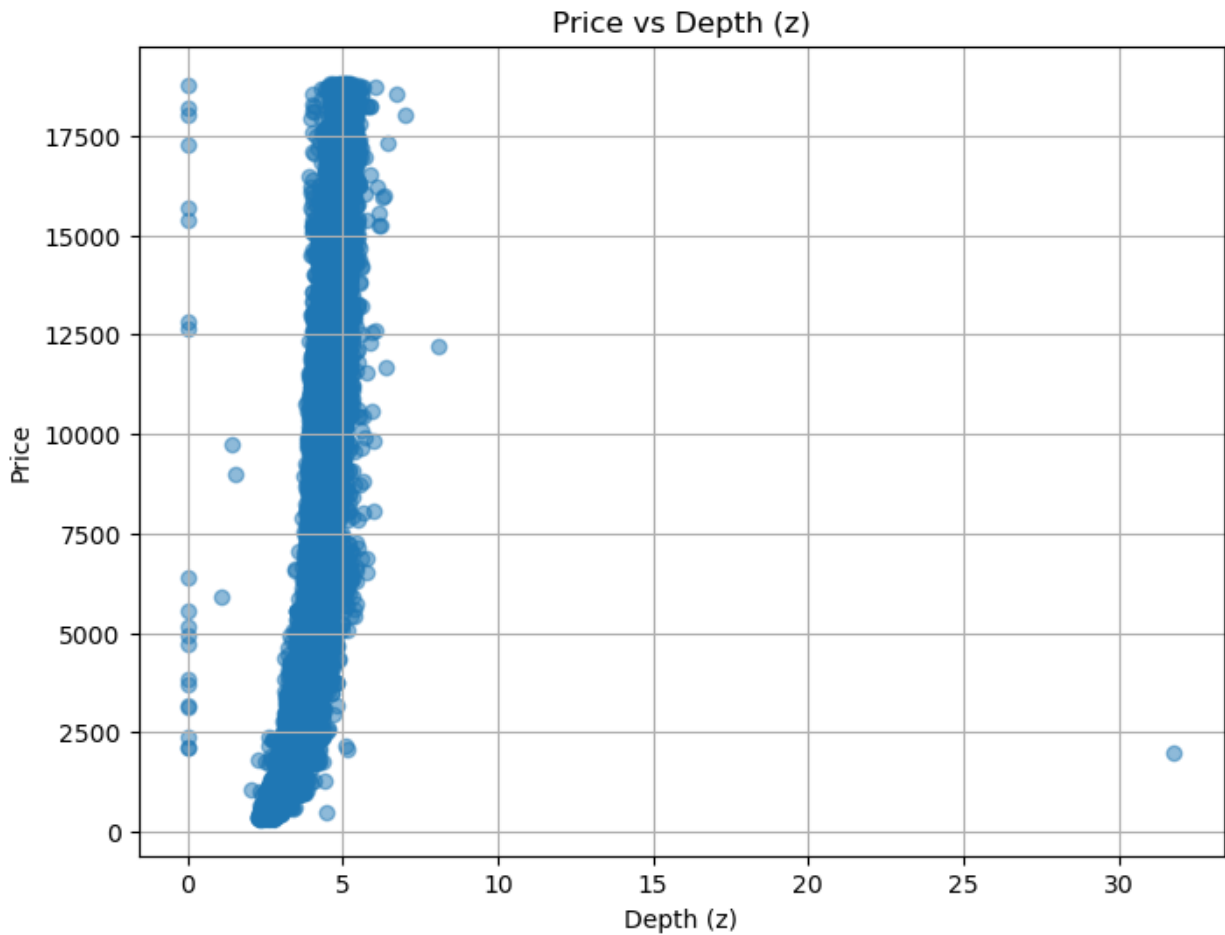
## 5. price vs. y



**Figure 2.21:** The scatter plot of price vs. y.

**Insight:** It shows that the attributes of price and y are highly correlated. As the y increases, the price also increases. The fact that larger or broader diamonds are typically valued more highly in the diamond market can be used to explain this correlation. Larger diamonds are more expensive because they are rare and frequently have more facets that can increase their shine and brilliance.

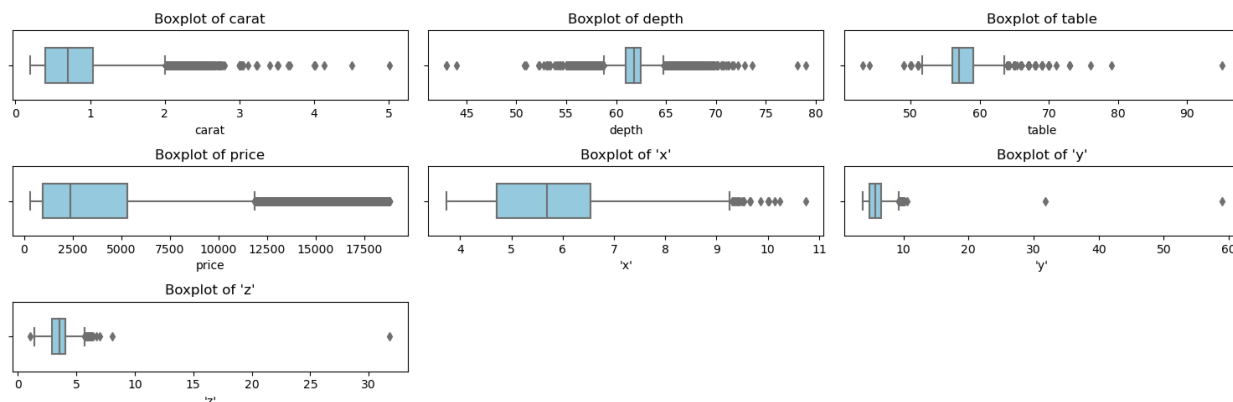
## 6. price vs. z



**Figure 2.22:** The scatter plot of price vs. z.

**Insight:** It shows that the attributes of price and z are highly correlated. As the z increases, the price also increases. The reason for this relationship is that deeper diamonds frequently require more material to be cut, which results in more waste during the cutting and shaping process. A deeper diamond may also weigh more carats, which affects its price because larger diamonds are typically more expensive.

## 2.5 Outliers detection



**Figure 2.23:** Boxplots of each numerical attribute.

**Insight:** From the visualization above, we can see that every one of the numerical variables has extreme outliers. These may indicate the unreliability of data. (Wada, 2020) However, we should consider their truthfulness before handling them. Hence, we will be calculating their skewness of data, deciding whether to deal with the outliers or just leave it after consideration.

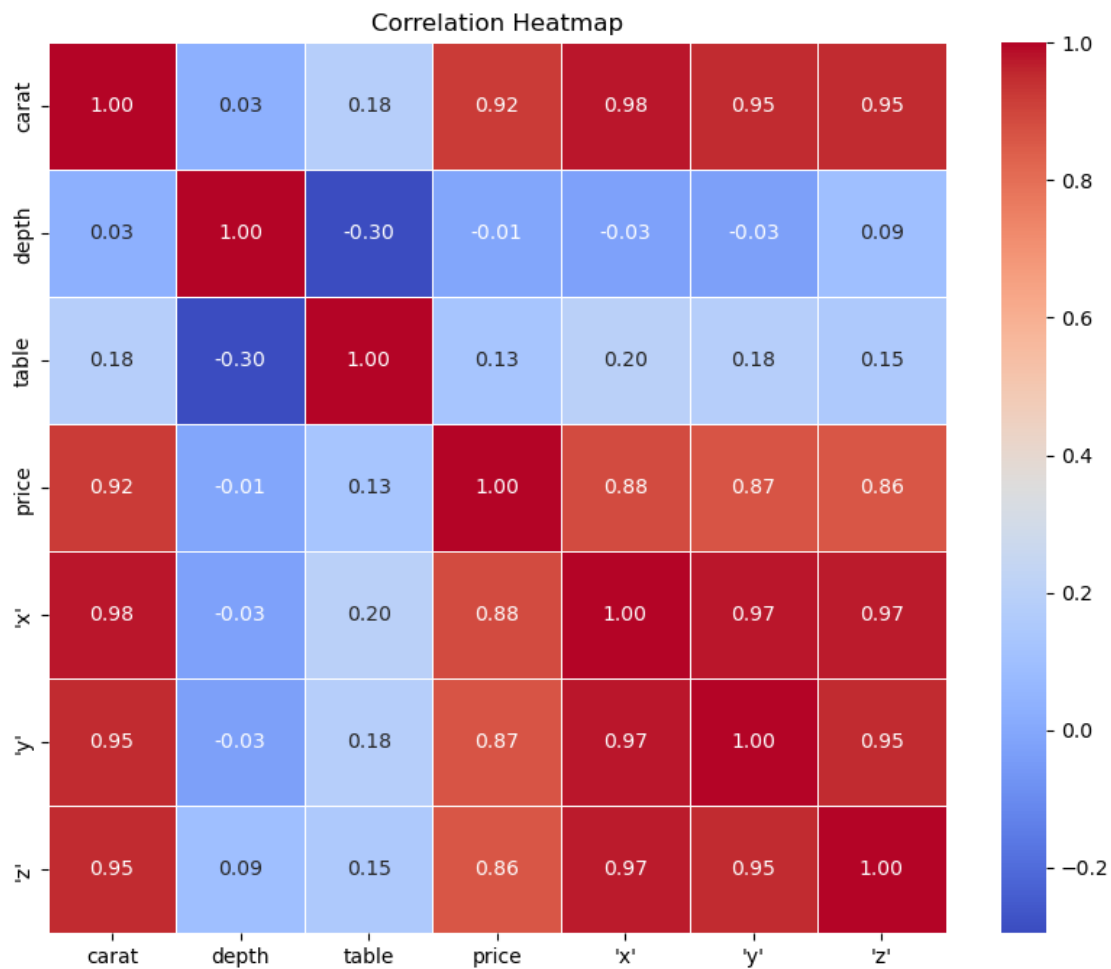
```
In [594]: # Check the skewness for each columns
for col in numeric_cols:
    skewness = data[col].skew()
    print("Skewness of {}: {}".format(col, skewness))

Skewness of carat: 1.1132176108381227
Skewness of depth: -0.11371107011617339
Skewness of table: 0.7920686095882784
Skewness of price: 1.6182203665466373
Skewness of 'x': 0.39690785208661533
Skewness of 'y': 2.470200340127712
Skewness of 'z': 1.5893087544943625
```

**Figure 2.24:** Finding skewness of each numerical variable.

**Insight:** The data is considered normal if the skewness is between -2 to 2 (Fuey & Idris, 2018). If the skewness value of the numerical features is still at this range, then we either remove or impute the outliers to prevent loss of information, which can affect the accuracy of the model. We can also replace them with a lower or upper limit. Thus, following the inferences, we are going to replace the outliers of attributes 'y' by lower/upper limit in the data preparation stage as merely removing them will cause large losses in our datasets.

## 2.6 Correlation Heatmap



**Figure 2.25:** Showing the correlation between the numerical attributes.

**Insight:** From the heat map above, we can see that x, y, and z have a very high correlation with the target variable price. Thus, it is very important to keep these variables for modeling. However, the table and depth have a low correlation with price. We will decide whether to keep these variables in the later stage.



### 3. DATA PREPARATION

#### 3.1 Removing Duplicates

```
In [22]: ► #Remove Duplicates
print("Before removing duplicates:")
data.shape
```

Before removing duplicates:

```
Out[22]: (53940, 10)
```

```
In [23]: ► data.drop_duplicates(inplace=True)
```

```
In [24]: ► # Print the duplicate rows after removing to double check
duplicate = data[data.duplicated()]
print("Duplicate Rows :")
# Print the resultant Dataframe
duplicate
```

Duplicate Rows :

```
Out[24]:
```

carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
-------	-----	-------	---------	-------	-------	-------	-----	-----	-----

```
In [25]: ► print("After removing duplicates:")
data.shape
```

After removing duplicates:

```
Out[25]: (53794, 10)
```

**Figure 3.1:** Removing duplicates.

**Explanation:** We lost 146 data points after removing duplicates. However, it is important to remove them because they can introduce noise and inaccuracies into our dataset, potentially leading to incorrect or biased results. (Mastering Data Cleaning & Data Preprocessing for Machine Learning, n.d.)

### 3.2 Removing Dimensionless Diamond Records

```
#Dropping dimensionless diamond records
data = data.drop(data[data["'x'"]==0].index)
data = data.drop(data[data["'y'"]==0].index)
data = data.drop(data[data["'z'"]==0].index)
data.shape

(53775, 10)
```

**Figure 3.2:** Removing dimensionless diamond records.

**Explanation:** We left 53775 records and lost 19 data points after removing dimensionless diamond records. This is because the presence of minimum values of zero for "x," "y," and "z" indicates potential data anomalies, suggesting the existence of diamonds with dimensionless or 2-dimensional attributes. Therefore, it is important to filter out these data points, as they likely represent faulty or incorrect data entries.

### 3.3 Removing Outliers

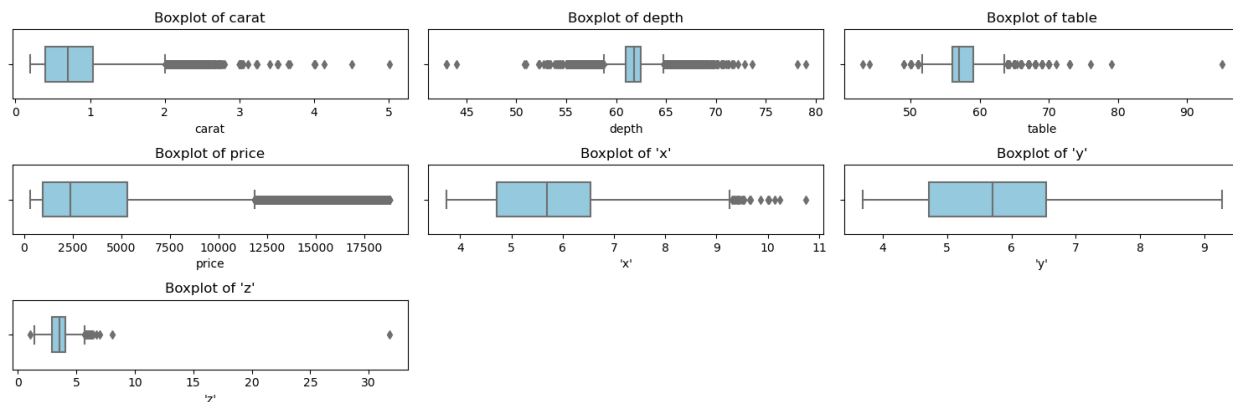
```
cols_with_outliers = [ "'y'"]
q1 = data[cols_with_outliers].quantile(0.25)
q3 = data[cols_with_outliers].quantile(0.75)
iqr = q3 - q1

lower = q1 - (iqr * 1.5)
upper = q3 + (iqr * 1.5)

for col in cols_with_outliers:
    data[col] = data[col].apply(lambda x: lower[col] if x < lower[col] else x)
    data[col] = data[col].apply(lambda x: upper[col] if x > upper[col] else x)
```

**Figure 3.3:** Removing outliers of attribute 'y'.

**Explanation:** After replacing the outliers of attribute 'y' with the lower and upper limits, we visualize the numerical features with a boxplot to peek again at the data distribution.



**Figure 3.3:** Boxplot of each numerical attribute (after removing outliers of y).

**Explanation:** It can be seen that the boxplot of attribute 'y' is free from outliers now. However, the other attributes still have outliers, but it is important to keep them since we follow the inferences as mentioned in the data understanding step.

### 3.4 Handling Categorical Variables

```
# Get list of categorical variables
s = (data.dtypes == "object")
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)
```

Categorical variables:  
['cut', 'color', 'clarity']

**Figure 3.4:** Showing categorical variables.

**Explanation:** There are 3 categorical variables which are ‘cut’, ‘color’, and ‘clarity’.

```
from sklearn.preprocessing import LabelEncoder

# Make a copy to avoid changing the original data
label_data = data.copy()

# Define custom mappings for each attribute
cut_mapping = {
    "b'Ideal'": 4,
    "b'Premium'": 3,
    "b'Very Good'": 2,
    "b'Good'": 1,
    "b'Fair'": 0
}

color_mapping = {
    "b'D'": 0,
    "b'E'": 1,
    "b'F'": 2,
    "b'G'": 3,
    "b'H'": 4,
    "b'I'": 5,
    "b'J'": 6
}

clarity_mapping = {
    "b'I1'": 0,
    "b'SI2'": 1,
    "b'SI1'": 2,
    "b'VS2'": 3,
    "b'VS1'": 4,
    "b'VVS2'": 5,
    "b'VVS1'": 6,
    "b'IF'": 7
}

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply custom Label encoding separately for each attribute
label_data['cut'] = label_encoder.fit_transform(label_data['cut'].map(cut_mapping))
label_data['color'] = label_encoder.fit_transform(label_data['color'].map(color_mapping))
label_data['clarity'] = label_encoder.fit_transform(label_data['clarity'].map(clarity_mapping))

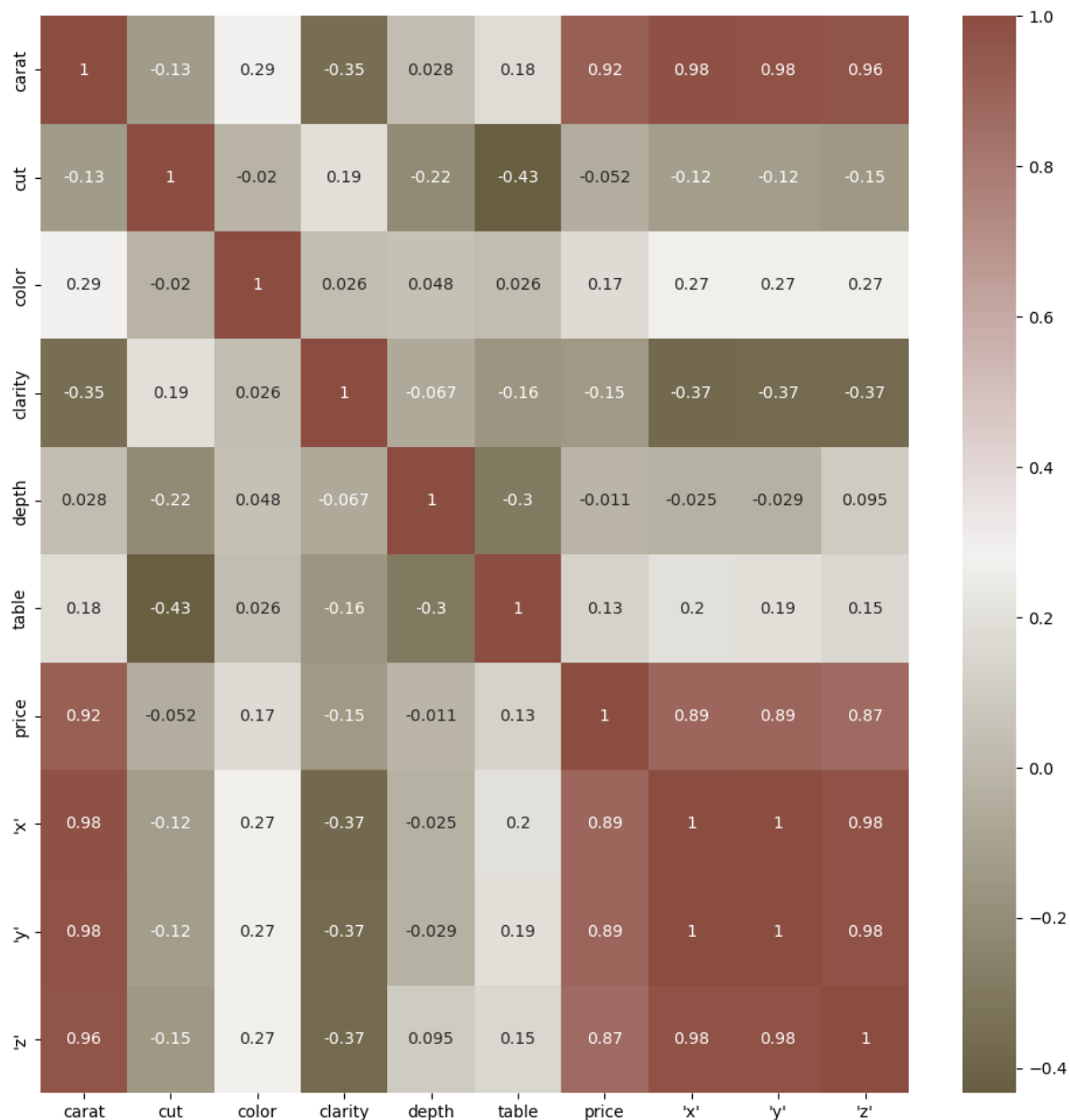
label_data.head()
```

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
0	0.23	4	1	1	61.5	55.0	326.0	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	1	1	4	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	3	5	3	62.4	58.0	334.0	4.20	4.23	2.63
4	0.31	1	6	1	63.3	58.0	335.0	4.34	4.35	2.75

**Figure 3.5:** Handling categorical variables.

**Explanation:** We encoded the categorical variables into numerical representations to facilitate their incorporation into machine learning algorithms and mathematical models.(Handling Categorical Data in Python Tutorial, n.d.)

### 3.5 Correlation Heatmap



**Figure 3.6:** Showing heatmap after encoding the categorical variables.

**Explanation:** It can be seen that carat, price, x, y, and z are highly correlated, thus they serve as strong predictors, helping to improve the accuracy of price predictions for diamonds. Even though the cut, color, clarity, depth, and table are relatively low correlated, we do not drop these attributes because machine learning algorithms can extract patterns and relationships that might not be apparent through simple correlation analysis. Combining these attributes with others in our models may improve its overall predictive accuracy.

```
label_data.corr()
```

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
carat	1.000000	0.017860	0.290943	-0.214002	0.027889	0.181113	0.921548	0.977857	0.976384	0.961030
cut	0.017860	1.000000	0.000352	0.028115	-0.193282	0.150711	0.040237	0.022393	0.028344	0.002223
color	0.290943	0.000352	1.000000	-0.027891	0.047654	0.026110	0.171746	0.270228	0.269875	0.269584
clarity	-0.214002	0.028115	-0.027891	1.000000	-0.053299	-0.087888	-0.071248	-0.225716	-0.222597	-0.225977
depth	0.027889	-0.193282	0.047654	-0.053299	1.000000	-0.297580	-0.011144	-0.025224	-0.028526	0.094678
table	0.181113	0.150711	0.026110	-0.087888	-0.297580	1.000000	0.126666	0.195451	0.189159	0.151683
price	0.921548	0.040237	0.171746	-0.071248	-0.011144	0.126666	1.000000	0.887137	0.888608	0.868030
'x'	0.977857	0.022393	0.270228	-0.225716	-0.025224	0.195451	0.887137	1.000000	0.998470	0.975382
'y'	0.976384	0.028344	0.269875	-0.222597	-0.028526	0.189159	0.888608	0.998470	1.000000	0.975131
'z'	0.961030	0.002223	0.269584	-0.225977	0.094678	0.151683	0.868030	0.975382	0.975131	1.000000

**Figure 3.7:** Showing table of correlations between attributes.

**Explanation:** Carat has a strong positive correlation with 'x', 'y', 'z', and price. This suggests that as carat weight increases, these attributes tend to increase as well. Furthermore, clarity has a negative correlation with carat, depth, and table, suggesting that diamonds with higher carat weights may have lower clarity and different depth and table percentages. Moreover, 'x', 'y', and 'z' have strong positive correlations with each other, indicating that they are closely related in terms of dimensions.

### 3.6 Splitting Datasets

```
# Assigning the features as X and target as y
X= label_data.drop(["price"],axis =1)
y= label_data["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25, random_state=7)
```

**Figure 3.8:** Split datasets into training and testing sets.

**Explanation:** We split the dataset into the proportion of 75% training set and 25% of the testing set.

### 3.7 Scaling

```
scaler = StandardScaler()  
  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)
```

**Figure 3.9:** Scaling.

**Explanation:** We scaled the training and testing set to maintain consistency and fairness in feature contributions, which can lead to more accurate and reliable model predictions.



## 4. MODELING

```
# Create dictionaries to store results for each model
results = {
    "Model": [],
    "R^2": [],
    "Adjusted R^2": [],
    "MAE": [],
    "MSE": [],
    "RMSE": [],
}
```

**Figure 4.1:** Create a dictionary.

**Explanation:** A dictionary is created to store results generated by each model. The dictionary will be used to compare the performance of each model.

## 4.1 Decision Tree Regression Model

```

from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import make_scorer, r2_score

# Define a custom scorer for R-squared
r2_scorer = make_scorer(score_func=r2_score, greater_is_better=True)

# Define the hyperparameter grid
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create a Decision Tree Regressor
dt_regressor = DecisionTreeRegressor()

# Create a GridSearchCV object with R-squared scoring
grid_search = GridSearchCV(estimator=dt_regressor, param_grid=param_grid, scoring=r2_scorer, cv=5)

# Fit the GridSearchCV object on the training data
grid_search.fit(X_train_scaled, y_train)

# Retrieve the best hyperparameters
best_params = grid_search.best_params_

# Create a new Decision Tree Regressor with the best hyperparameters
best_dt_regressor = DecisionTreeRegressor(**best_params)

# Fit the model on the training data with the best hyperparameters
best_dt_regressor.fit(X_train_scaled, y_train)

```

DecisionTreeRegressor

DecisionTreeRegressor(max\_depth=10, max\_features='auto', min\_samples\_leaf=4)

**Figure 4.2:** Hyperparameter tuning for Decision Tree Regressor.

**Explanation:** GridSearchCV, DecisionTreeRegressor, make\_scorer, and r2\_score were all imported in order to do grid search hyperparameter tuning, create custom scoring functions, and calculate R-squared scores.

Make\_scorer is used to create a special scoring function. Higher R-squared values are better, according to the greater\_is\_better=True setting, and r2\_score is utilized as the scoring function to produce the R-squared score.

In a hyperparameter grid, different values for "max\_depth," "min\_samples\_split," "min\_samples\_leaf," and "max\_features" are present. These hyperparameters will be changed throughout the grid search.

We build a GridSearchCV object with an R-squared score and a Decision Tree Regressor. Once the GridSearchCV object has been fitted to the training data, find the optimum hyperparameters, create a new Decision Tree Regressor using those, and fit the model to the data.

For a Decision Tree Regressor to perform at its best, avoid overfitting, and strike the ideal balance between complexity and simplicity, hyperparameter tuning is crucial.

```
: #Predict on the training and test data
pred_train = best_dt_regressor.predict(X_train_scaled)
pred_test = best_dt_regressor.predict(X_test_scaled)

# Calculate R-squared for training and test data
r2_train = metrics.r2_score(y_train, pred_train)
r2_test = metrics.r2_score(y_test, pred_test)

# Print the R-squared values for both training and test data
print("R-squared (Training):", r2_train)
print("R-squared (Test):", r2_test)

R-squared (Training): 0.9819260532967826
R-squared (Test): 0.9746903904794378
```

**Figure 4.3:** Evaluate model performance using R-squared.

**Explanation:** We aimed to verify the model's consistency by comparing the R-squared values between the training and testing datasets. A minimal difference suggests the model generalizes well to new data, ensuring reliable predictions.

```

# Calculate metrics
r2 = metrics.r2_score(y_test, pred_test)
adjusted_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test_scaled.shape[1] - 1)
mae = metrics.mean_absolute_error(y_test, pred_test)
mse = metrics.mean_squared_error(y_test, pred_test)
rmse = np.sqrt(mse)

# Store results in the dictionary
results["Model"].append("Decision Tree")
results["R^2"].append(r2)
results["Adjusted R^2"].append(adjusted_r2)
results["MAE"].append(mae)
results["MSE"].append(mse)
results["RMSE"].append(rmse)

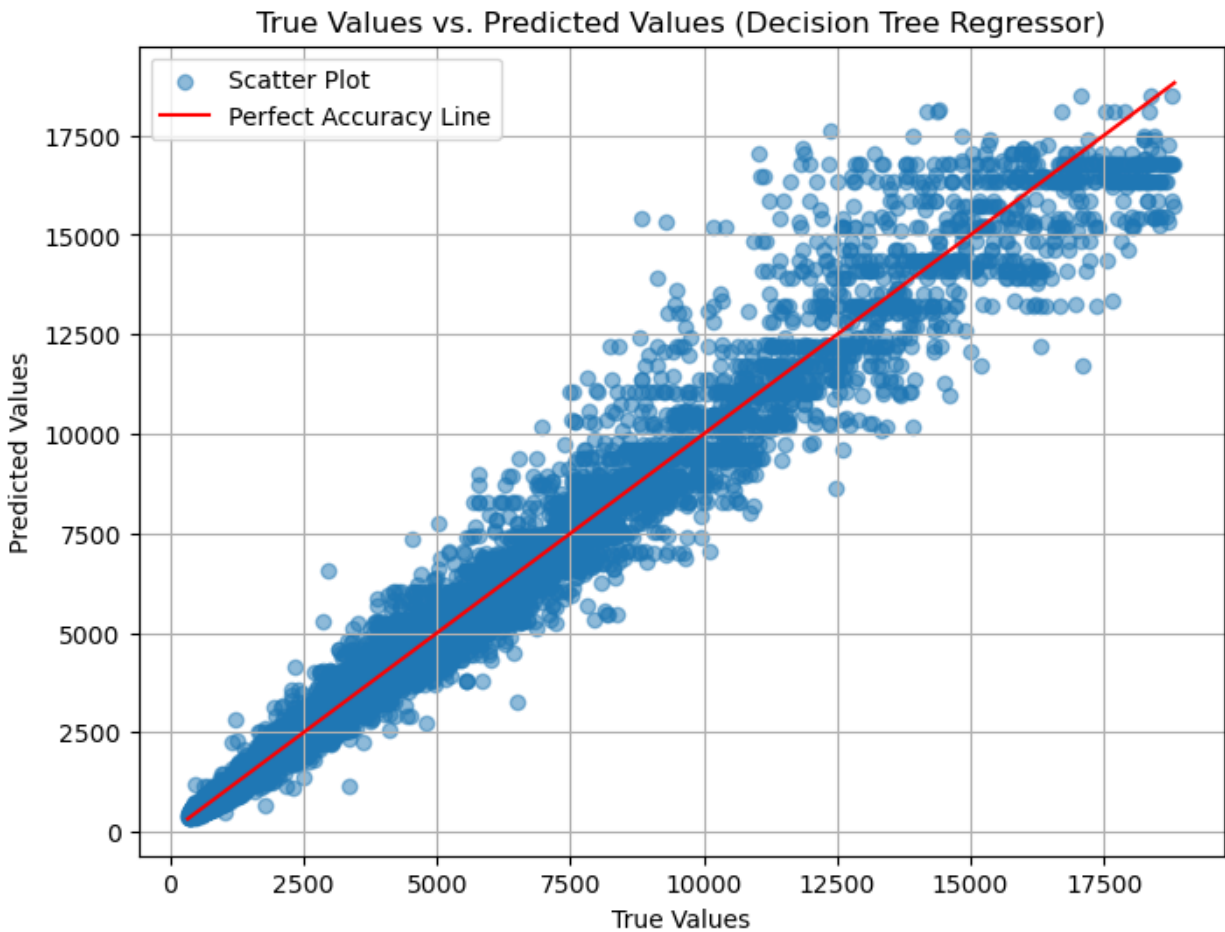
# Print the formatted results
print(f"R^2: {r2:.4f}")
print(f"Adjusted R^2: {adjusted_r2:.4f}")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")

R^2: 0.9747
Adjusted R^2: 0.9747
MAE: 338.3322
MSE: 403423.4150
RMSE: 635.1562

```

**Figure 4.4:** Calculate and store regression metrics results.

**Explanation:** We calculate the metrics and store the result in the dictionary which we defined earlier and show the results of the Decision Tree Regression Model's performance.



**Figure 4.5:** Visualize true vs. predicted values (scatter plot).

**Explanation:** We plotted the scatter plot to show the relationship between the true values and the predicted values for the Decision Tree Regression Model. It can be seen that the points on the scatter plot are close to this diagonal line, which means the model's predictive performance is making accurate predictions.

## 4.2 Random Forest Regression Model

```
: from sklearn.ensemble import RandomForestRegressor

# 1. Choose model class
model_randomForest = RandomForestRegressor()

# 2. Instantiate and fit model to data
model_randomForest.fit(X_train_scaled, y_train)

# 3. Predict on training and test data
pred_train = model_randomForest.predict(X_train_scaled)
pred_test = model_randomForest.predict(X_test_scaled)

# Calculate R-squared for training and test data
r2_train = metrics.r2_score(y_train, pred_train)
r2_test = metrics.r2_score(y_test, pred_test)

# Print the R-squared values for both training and test data
print("R-squared (Training):", r2_train)
print("R-squared (Test):", r2_test)

R-squared (Training): 0.9973873009806851
R-squared (Test): 0.9813638601034886
```

**Figure 4.6:** Evaluate model performance using R-squared.

**Explanation:** RandomForestRegressor is imported in order to use the random forest model. This model is suitable for predicting the price for several reasons. Firstly, non-linearity. Prices of diamonds may not always correlate directly with their characteristics (carat, cut, color, clarity, etc.). A Random Forest is suited for modeling nonlinear interactions because it can identify intricate non-linear patterns in the data. Secondly, this model is very robust. Compared to certain other regression techniques, Random Forests are less sensitive to outliers and noise in the data. The model's predictions are less likely to be significantly impacted by diamonds with odd properties or dataset flaws. After going through several trials and errors to detect the best parameter for the model, the default parameter for the model is used since it provides the best performance.

```

# Calculate metrics
r2 = metrics.r2_score(y_test, pred_test)
adjusted_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test_scaled.shape[1] - 1)
mae = metrics.mean_absolute_error(y_test, pred_test)
mse = metrics.mean_squared_error(y_test, pred_test)
rmse = np.sqrt(mse)

# Store results in the dictionary
results["Model"].append("Random Forest")
results["R^2"].append(r2)
results["Adjusted R^2"].append(adjusted_r2)
results["MAE"].append(mae)
results["MSE"].append(mse)
results["RMSE"].append(rmse)

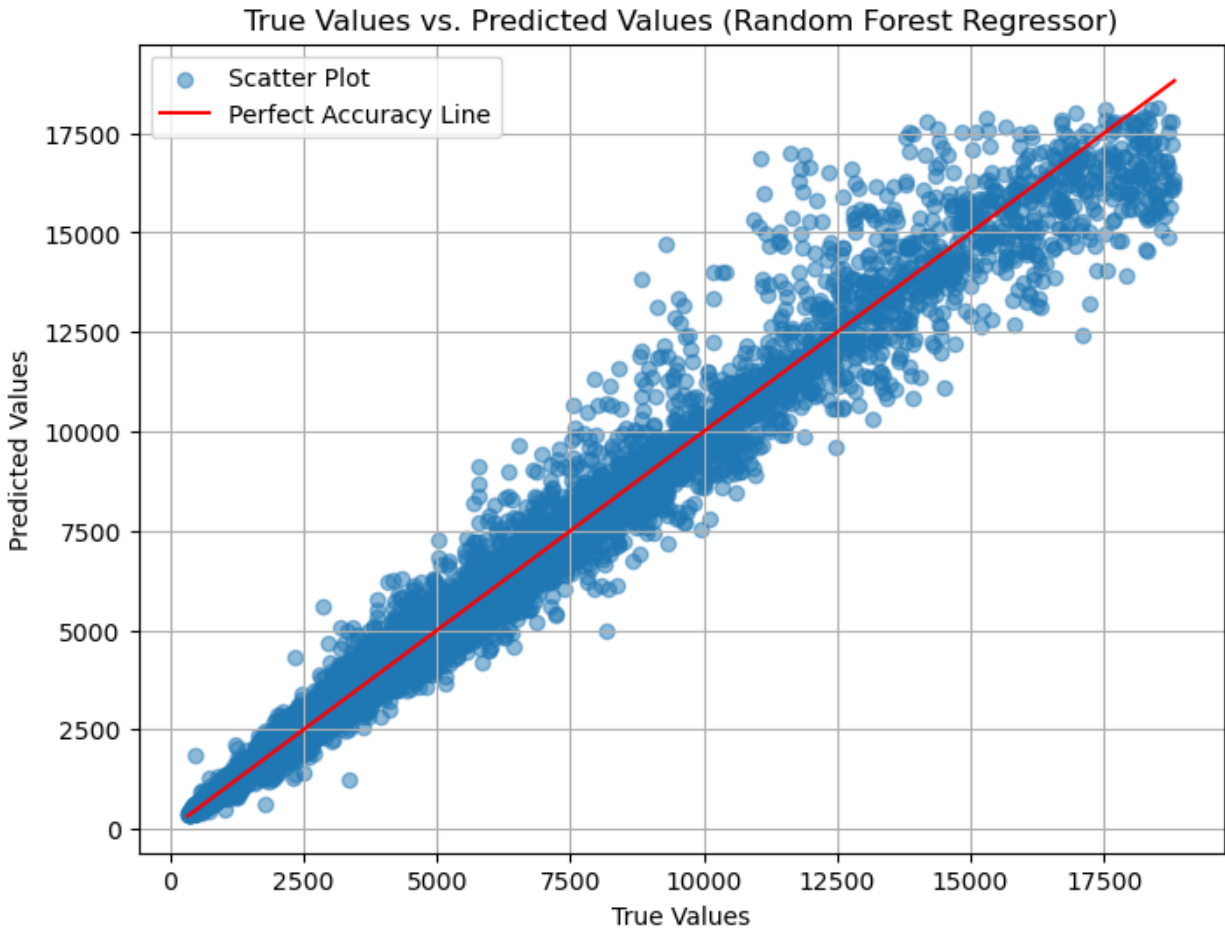
# Print the formatted results
print(f"R^2: {r2:.4f}")
print(f"Adjusted R^2: {adjusted_r2:.4f}")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")

R^2: 0.9814
Adjusted R^2: 0.9814
MAE: 270.9754
MSE: 297051.4102
RMSE: 545.0242

```

**Figure 4.7:** Calculate and store regression metrics results.

**Explanation:** We calculate the metrics and store the result in the dictionary which we defined earlier and show the results of the Random Forest Regression Model.



**Figure 4.8:** Visualize true vs. predicted values (scatter plot).

**Explanation:** A scatter plot is plotted to show the relationship between the true values and the predicted values for the Random Forest Regression Model. It can be seen that the points on the scatter plot are close to this diagonal line, which means the model's predictive performance is good.



### 4.3 Gradient Boosting Regression Model

```
from sklearn.ensemble import GradientBoostingRegressor
gb_model = GradientBoostingRegressor(n_estimators=500, learning_rate=0.1, max_depth=3, random_state=7)

# Fit the model to the training data
gb_model.fit(X_train_scaled, y_train)

# 3. Predict on training and test data
pred_train = gb_model.predict(X_train_scaled)
pred_test = gb_model.predict(X_test_scaled)

# Calculate R-squared for training and test data
r2_train = metrics.r2_score(y_train, pred_train)
r2_test = metrics.r2_score(y_test, pred_test)

# Print the R-squared values for both training and test data
print("R-squared (Training):", r2_train)
print("R-squared (Test):", r2_test)
```

R-squared (Training): 0.985045205519413  
R-squared (Test): 0.9808255991091915

**Figure 4.9:** Evaluate model performance using R-squared.

**Explanation:** Gradient Boosting Regressor is imported in order to use this model and to fit the model onto the training data. There are several compelling reasons for choosing this model. To begin with, this model has impressive high predictive accuracy. This model has excelled in numerous data science challenges and it can capture complicated correlations in the data with ease, including non-linear and interaction impacts. This model also has robustness to outliers. Gradient boosting can withstand noisy data and outliers. With a dataset like diamond pricing, where outliers can have a big impact on predictions, the ability to handle extreme values and anomalies gracefully is crucial. The values for the model's parameter are carefully chosen after going through several testing manually to identify the best parameter.

```

# Calculate metrics
r2 = metrics.r2_score(y_test, pred_test)
adjusted_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test_scaled.shape[1] - 1)
mae = metrics.mean_absolute_error(y_test, pred_test)
mse = metrics.mean_squared_error(y_test, pred_test)
rmse = np.sqrt(mse)

# Store results in the dictionary
results["Model"].append("Gradient Boosting")
results["R^2"].append(r2)
results["Adjusted R^2"].append(adjusted_r2)
results["MAE"].append(mae)
results["MSE"].append(mse)
results["RMSE"].append(rmse)

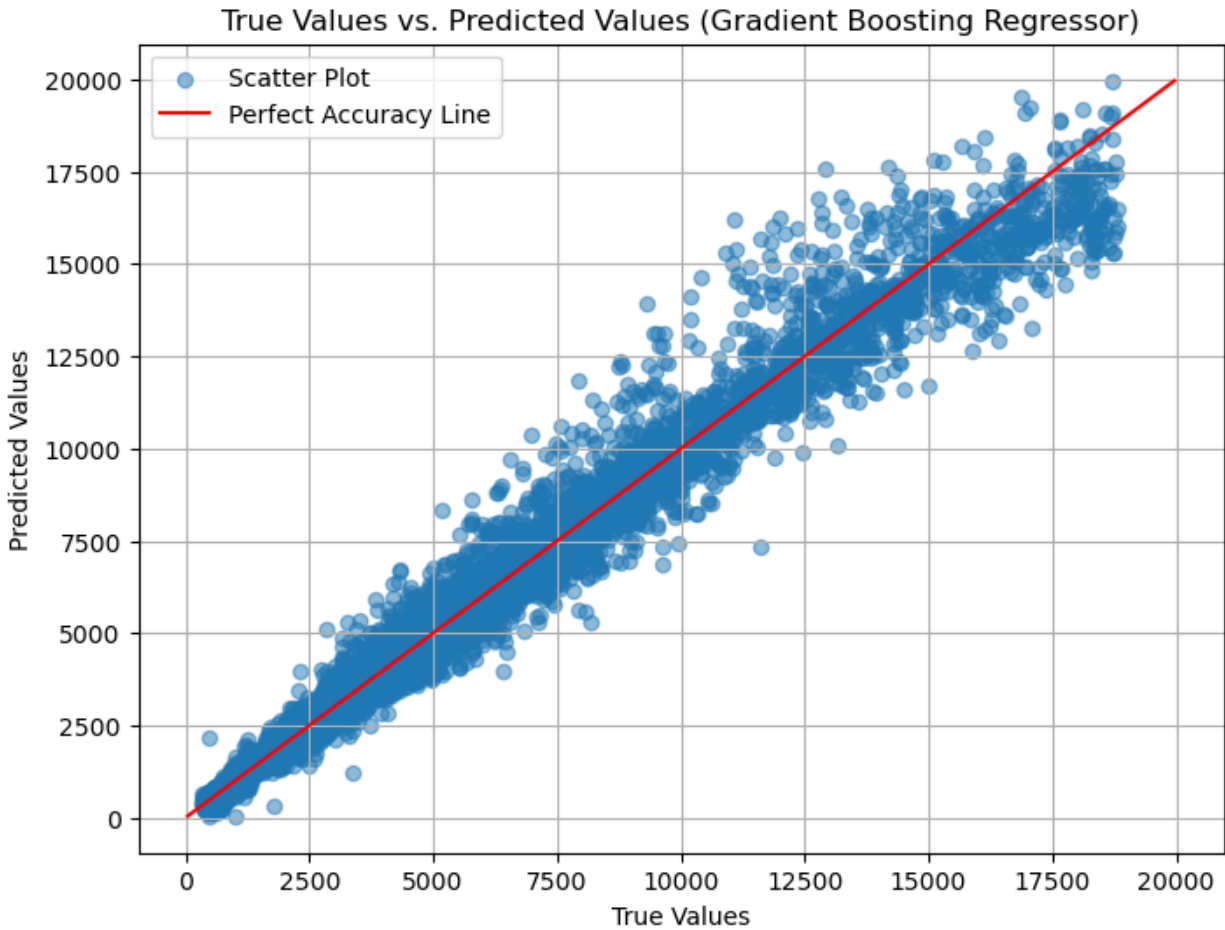
# Print the formatted results
print(f"R^2: {r2:.4f}")
print(f"Adjusted R^2: {adjusted_r2:.4f}")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")

R^2: 0.9808
Adjusted R^2: 0.9808
MAE: 301.7505
MSE: 305631.0403
RMSE: 552.8391

```

**Figure 4.10:** Calculate and store regression metrics results.

**Explanation:** We calculate the metrics of the model and store the result in the dictionary which we defined earlier and show the results of the Gradient Boosting Regressor Model.



**Figure 4.11:** Visualize true vs. predicted values (scatter plot).

**Explanation:** A scatter plot is plotted to show the relationship between the true values and the predicted values for the Gradient Boosting Regressor Model. It can be seen that the points on the scatter plot are close to this diagonal line, which means the model's predictive performance is good.

## 4.4 K Neighbours Regression Model

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor

# Create a new K Neighbors Regressor with the best k value
best_knn_model = KNeighborsRegressor(13)

# Fit the model on the training data with the best k value
best_knn_model.fit(X_train_scaled, y_train)

# 3. Predict on training and test data
pred_train = best_knn_model.predict(X_train_scaled)
pred_test = best_knn_model.predict(X_test_scaled)

# Calculate R-squared for training and test data
r2_train = metrics.r2_score(y_train, pred_train)
r2_test = metrics.r2_score(y_test, pred_test)

# Print the R-squared values for both training and test data
print("R-squared (Training):", r2_train)
print("R-squared (Test):", r2_test)
```

```
R-squared (Training): 0.973238071178297
R-squared (Test): 0.9691451837399503
```

**Figure 4.12:** Evaluate model performance using R-squared.

**Explanation:** K-Nearest Neighbors Regressor is used because it is suitable for predicting the diamond prices for several reasons. First of all, simple concepts. Conceptually, the KNeighborsRegressor model is straightforward. By averaging the values of its k nearest neighbors in the feature space, it predicts the target variable (diamond price). In circumstances when interpretability is vital, this intuitive method may be desirable. Secondly, adaptability to data. Different data patterns can be adapted to using KNeighborsRegressor. KNeighborsRegressor can efficiently capture correlations between characteristics and the target variable that are linear, non-linear, or even irregular in the data. For this model, we did trial and error to find the best 'n\_neighbours' value for this model. Using the best hyperparameter for the model results in a fine-tuned model optimized for R-squared performance, which measures how well the model fits the data and predicts the target variable.

```

# Calculate metrics
r2 = metrics.r2_score(y_test, pred_test)
adjusted_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test_scaled.shape[1] - 1)
mae = metrics.mean_absolute_error(y_test, pred_test)
mse = metrics.mean_squared_error(y_test, pred_test)
rmse = np.sqrt(mse)

# Store results in the dictionary
results["Model"].append("K Neighbors")
results["R^2"].append(r2)
results["Adjusted R^2"].append(adjusted_r2)
results["MAE"].append(mae)
results["MSE"].append(mse)
results["RMSE"].append(rmse)

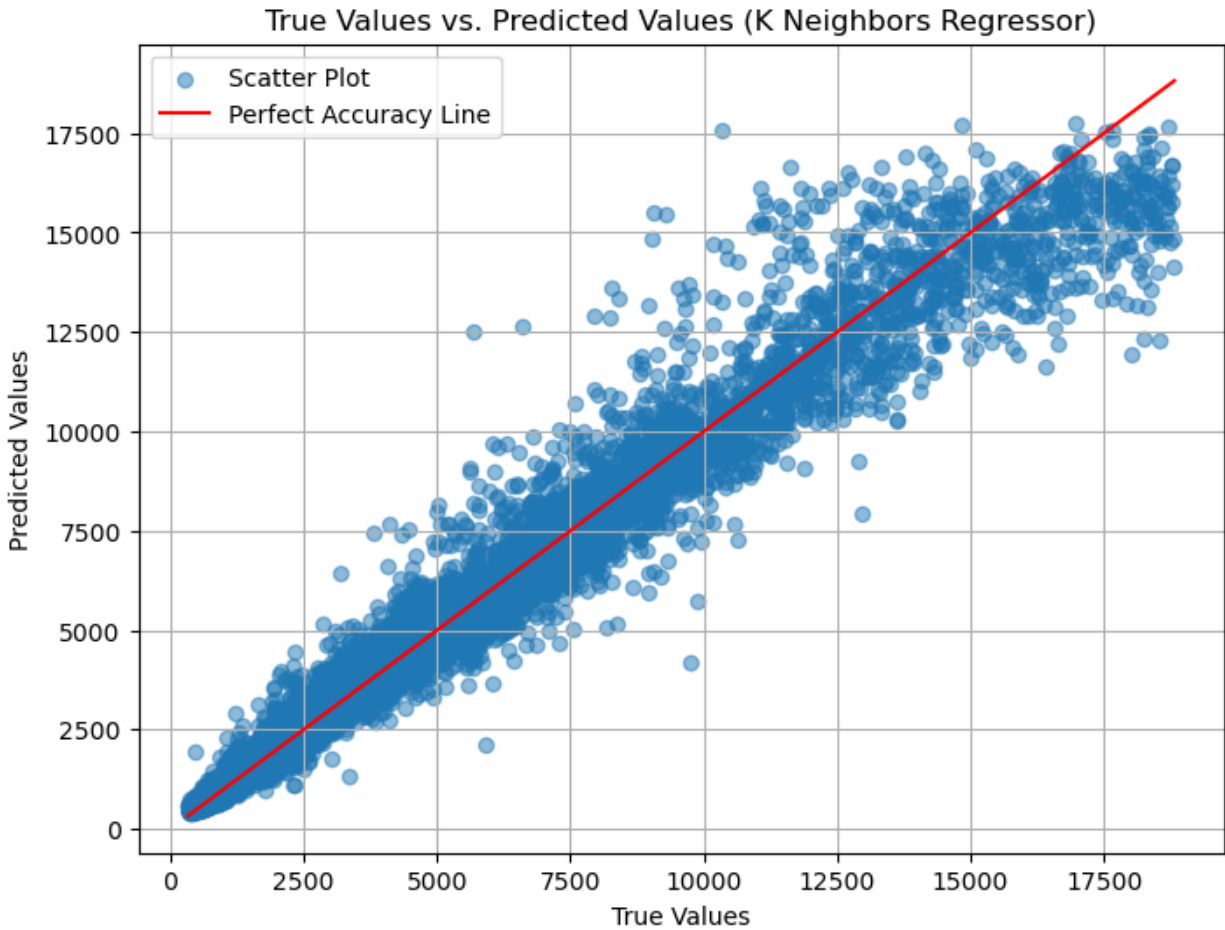
# Print the formatted results
print(f"R^2: {r2:.4f}")
print(f"Adjusted R^2: {adjusted_r2:.4f}")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")

R^2: 0.9691
Adjusted R^2: 0.9691
MAE: 370.9868
MSE: 491811.4337
RMSE: 701.2927

```

**Figure 4.13:** Calculate and store regression metrics results.

**Explanation:** We calculate the metrics of the model and store the result in the dictionary which we defined earlier and show the results of the K Neighbors Regressor Model.



**Figure 4.14:** Visualize true vs. predicted values (scatter plot).

**Explanation:** A scatter plot is plotted to show the relationship between the true values and the predicted values for the K Neighbors Regressor Model. It can be seen that the points on the scatter plot are close to this diagonal line or the middle, which means the model's predictive performance is pretty good.

```
# Create a DataFrame from the results dictionary
from tabulate import tabulate
results_df = pd.DataFrame(results)
results_df.index = range(1, len(results_df) + 1)
pd.set_option('display.max_columns', None) # Display all columns
pd.set_option('display.width', 2000)

# Display the results table
print(tabulate(results_df, headers='keys', tablefmt='fancy_grid'))
```

**Figure 4.15:** Creating a data frame from the results dictionary.

**Explanation:** A data frame is created from the ‘results’ dictionary in order to tabulate the results of each model, so that comparisons can be done easily.

## 5. EVALUATION

### 5.1 Results

	Model	R <sup>2</sup>	Adjusted R <sup>2</sup>	MAE	MSE	RMSE
1	Decision Tree	0.974677	0.97466	338.459	403641	635.327
2	Random Forest	0.981229	0.981216	271.821	299205	546.996
3	Gradient Boosting	0.980826	0.980813	301.751	305631	552.839
4	K Neighbors	0.969145	0.969125	370.987	491811	701.293

**Figure 5.1:** Results of each model in table form

The results of each model can be now viewed easily on a table, side by side. From the first row, which is the Decision Tree model row, it can be seen that the Decision Tree model has a high  $R^2$  value of 0.974677, indicating that it explains a significant portion of the variance in the target variable. The Adjusted  $R^2$  is very close to the  $R^2$ , suggesting that the model does not really suffer from overfitting. The MAE of 338.459 indicates that, on average, the model's predictions are off by approximately 338 units. This shows that the model typically predicts things with quite big mistakes even if we did hyperparameter tuning for this model. Moreover, the MSE is pretty high at 403641, which means the model is really sensitive to outliers, meaning that extreme data points have a disproportionately negative impact on the model's performance. When outliers are present in the data, decision trees have a tendency to overreact to them, which raises the MSE. The RMSE of 635.327 gives an estimate of the average prediction error in the original units of the target variable. In other words, the model's predictions deviate from the true diamond prices by approximately \$633.327. This may be because we kept outliers of some of the attributes because their skewness is within absolute 2 (Fuey & Idris, 2018) to prevent loss of information. Outliers can affect how sensitive a decision tree is. The model may struggle with adjusting for extreme values or outliers in your dataset, which would result in larger prediction errors and a higher RMSE.

For the second row, the Random Forest model. With a higher  $R^2$  of 0.981229, the Random Forest model outperforms the Decision Tree and provides a better match to the data. Good generalization is indicated by the Adjusted  $R^2$  being virtually identical to the  $R^2$ . This implies that the Random Forest model's performance on new data is consistent with that on training data. Since this model's predictions on average are more accurate than those of the Decision Tree, the MAE is lower than that of the Decision Tree. Additionally, the MSE is smaller, demonstrating



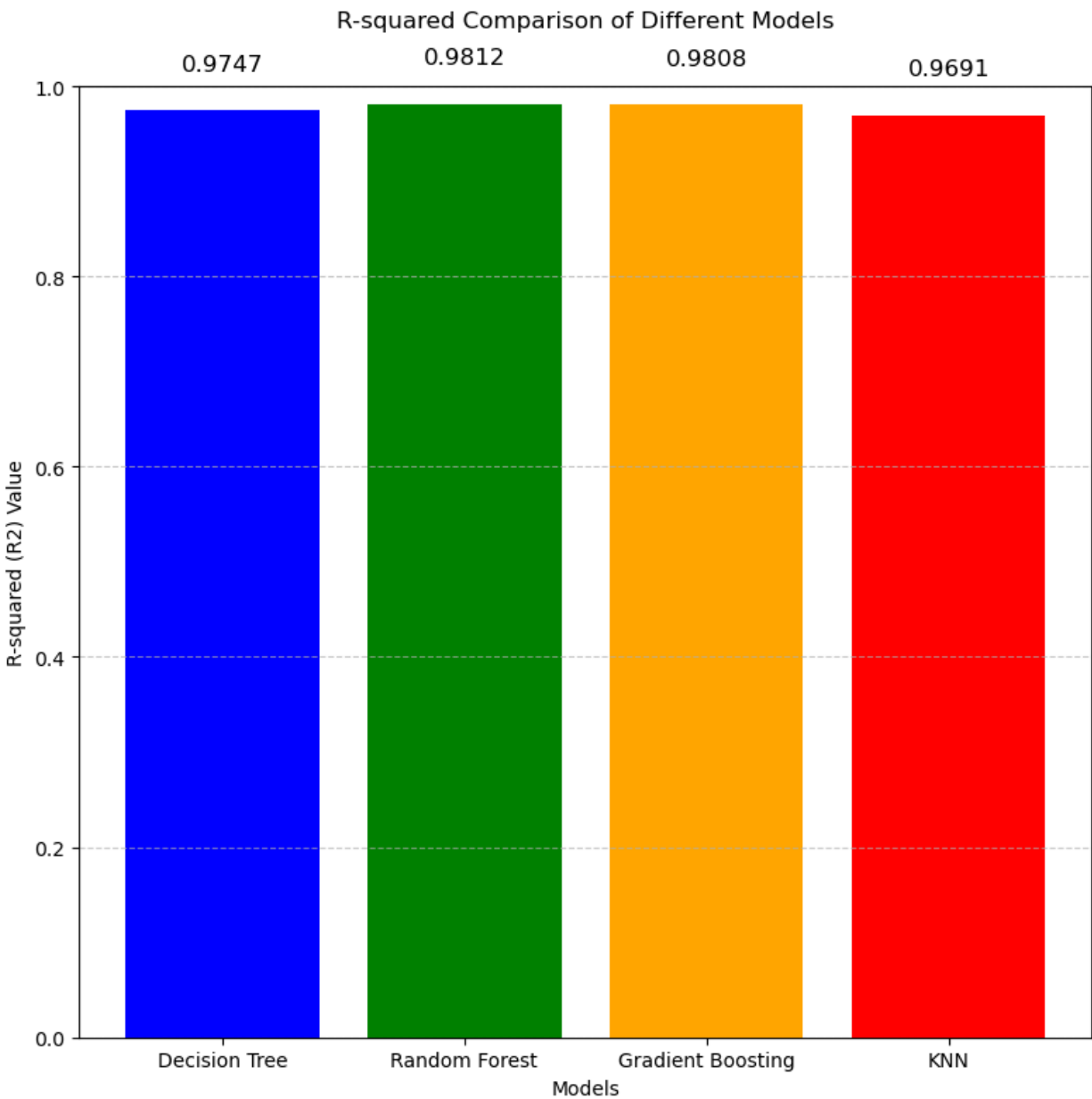
decreased susceptibility to outliers, indicating that it performs better generally. Compared to the Decision Tree, the RMSE is 546.701, indicating improved prediction accuracy.

On the other hand, similar to the Random Forest model, the third row, which is the Gradient Boosting model, has a high  $R^2$  of 0.980826 and Adjusted  $R^2$  of 0.980813, indicating good model fit and generalization. The MAE is lower than that of the Decision Tree but slightly higher than the Random Forest, indicating good predictive accuracy. The MSE of 305631 indicates that the model's predictions are closer to the actual values on average compared to the Decision Tree model. Due to its ensemble learning, sequential refinement, complexity management, and robustness to outliers, which together maximize its predictive performance, the Gradient Boosting model offers more accurate predictions with fewer mistakes. The RMSE is 552.839, which is a reasonable level of prediction error.

In comparison to the other models, the K Neighbors model explains less variance in the target variable because it has the lowest  $R^2$  of the four models. Additionally, the Adjusted  $R^2$  is the lowest, raising the possibility of overfitting. The KNN model's predictions deviate from the actual values greater than those of the other models, on average, as indicated by the largest MAE. The MSE is the highest, indicating a high sensitivity to outliers. The RMSE is also the highest, indicating a larger average prediction error in the original units.

To wrap it all up, based on the results, Random Forest Model appears to be the best performing model amongst all the models as it has the best results compared to all of the other models. Hence, Random Forest Model would be chosen to be used as the model to predict the diamond price.

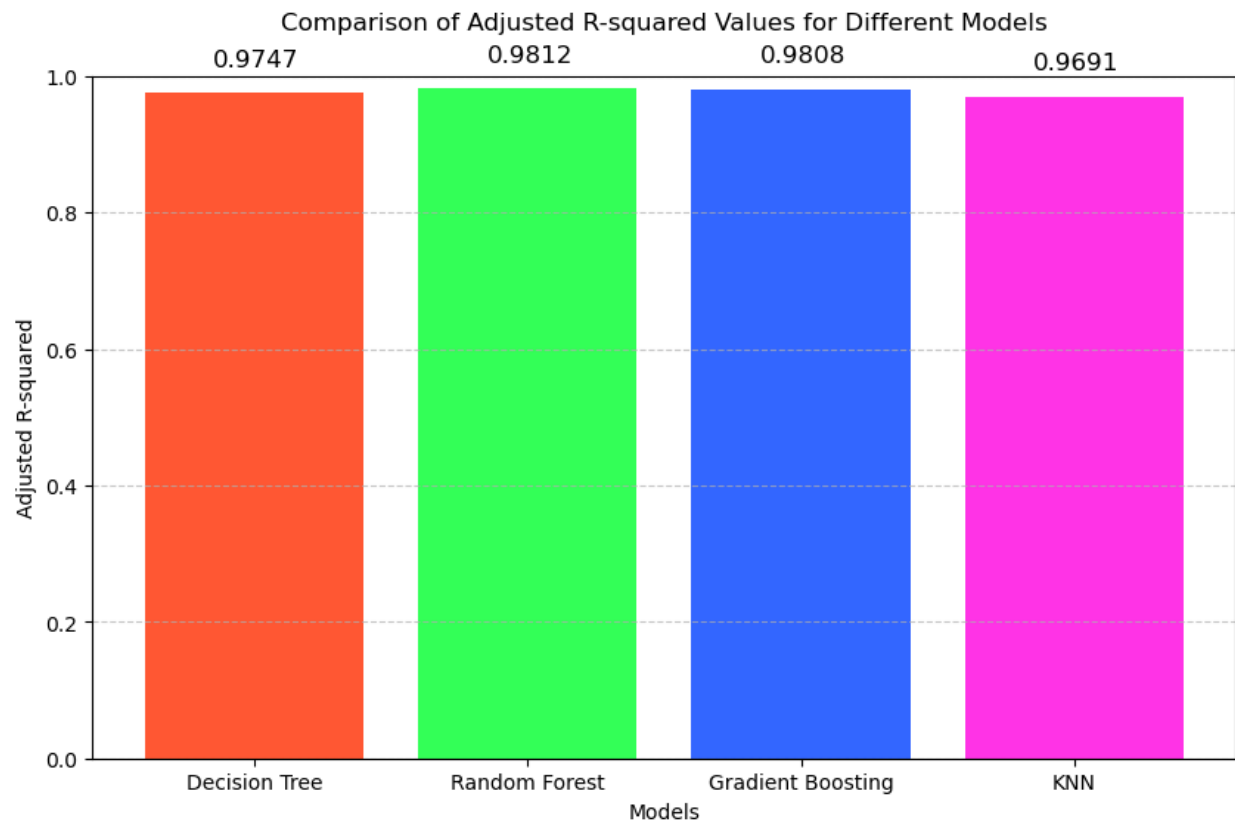
## 5.2 R-squared



**Figure 5.2:** Barplot for comparing R\_squared of all models.

**Explanation:** From **Figure 5.2** above, the Random Forest model with a high R-squared value of 0.9812 outperforms other models like Decision Tree, Gradient Boosting, and KNN in explaining the dataset's variability. To put it another way, it shows that the Random Forest model is successful in identifying the underlying trends in the data.

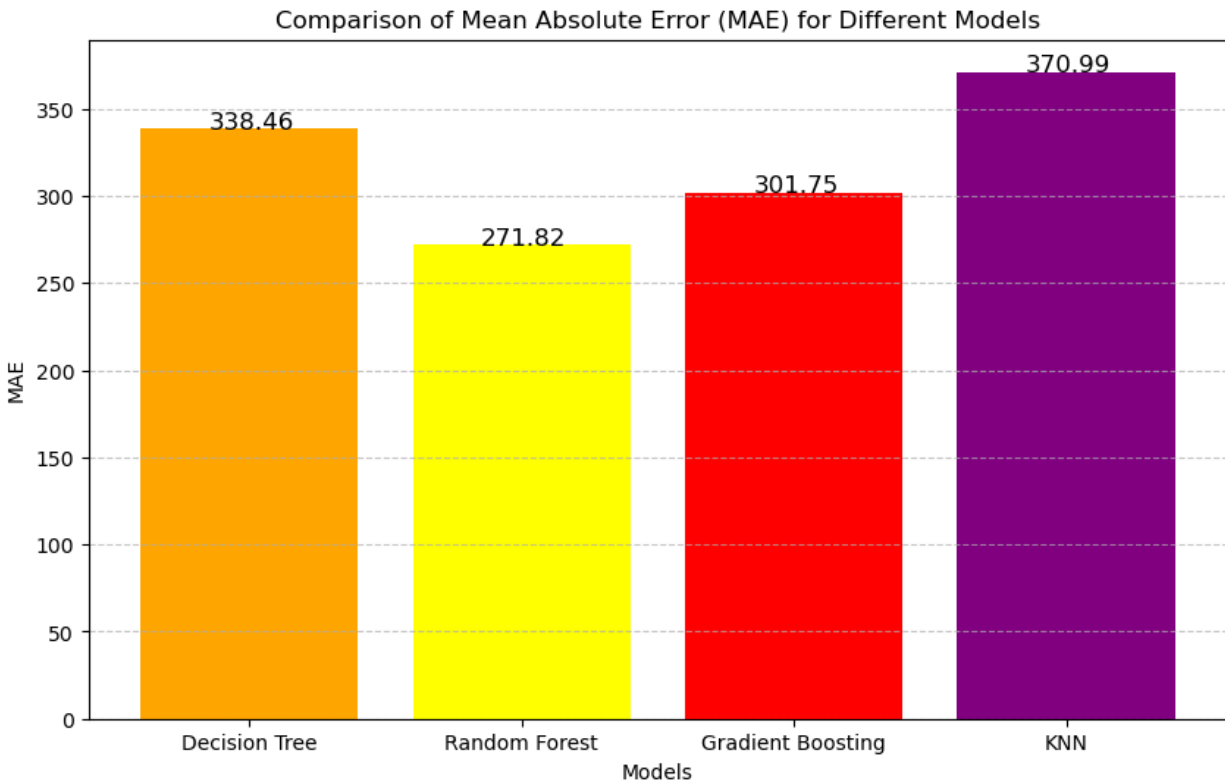
### 5.3 Adjusted R-squared



**Figure 5.3:** Barplot for comparing Adjusted R\_squared of models.

**Explanation:** The Random Forest model stands out in **Figure 5.3** above with an amazing adjusted R-squared value of 0.9812, proving its superiority to other models like Decision Tree, Gradient Boosting, and KNN in thoroughly describing the dataset's variability. Or, to put it another way, this highlights the outstanding ability of the Random Forest model to capture and reveal the underlying patterns within the data, offering a highly effective depiction of the link between the predictors and the target variable.

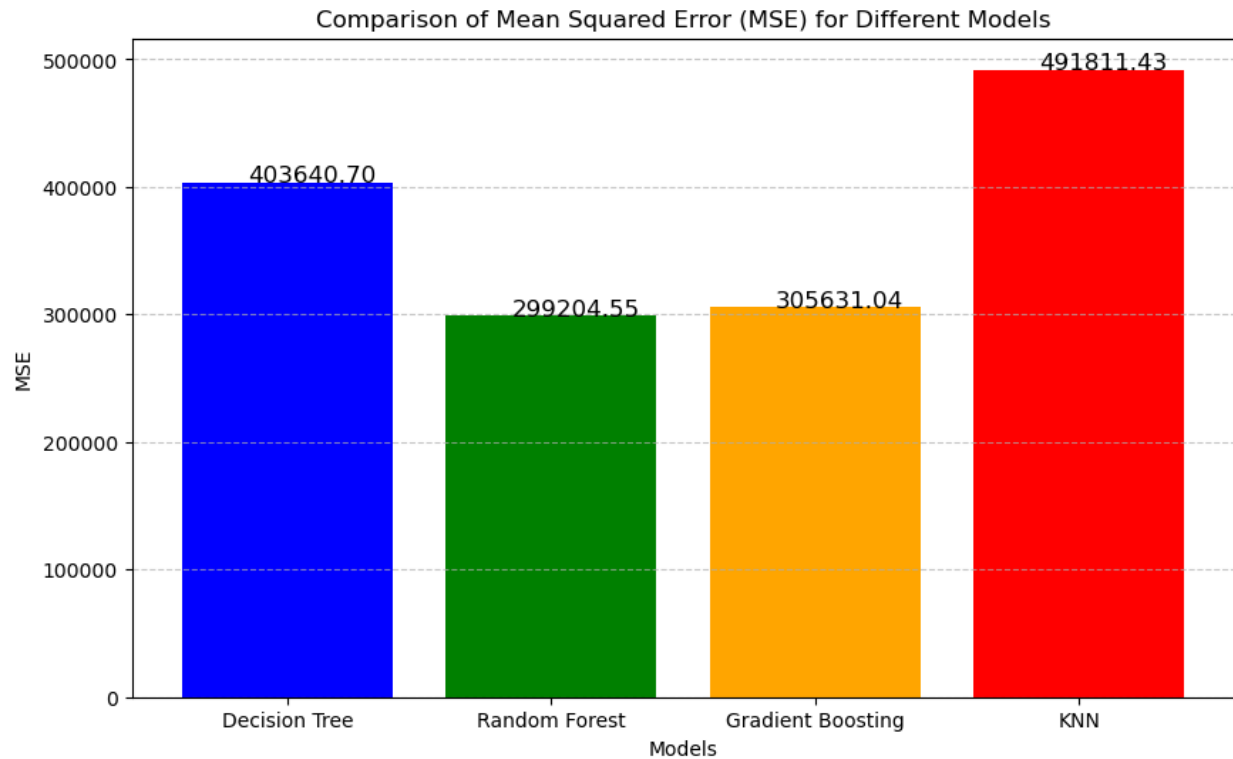
## 5.4 Mean Absolute Error (MAE)



**Figure 5.4:** Barplot for comparing Mean Absolute Error (MAE) of models.

**Explanation:** While KNN has the highest Mean Absolute Error (MAE), Random Forest has the lowest MAE as shown in **Figure 5.4**. With the least average absolute difference between its predictions and the actual values, the Random Forest model excels at making predictions, according to this observation. The lower prediction accuracy of KNN, which has a greater MAE, suggests that it typically diverges more from the true values.

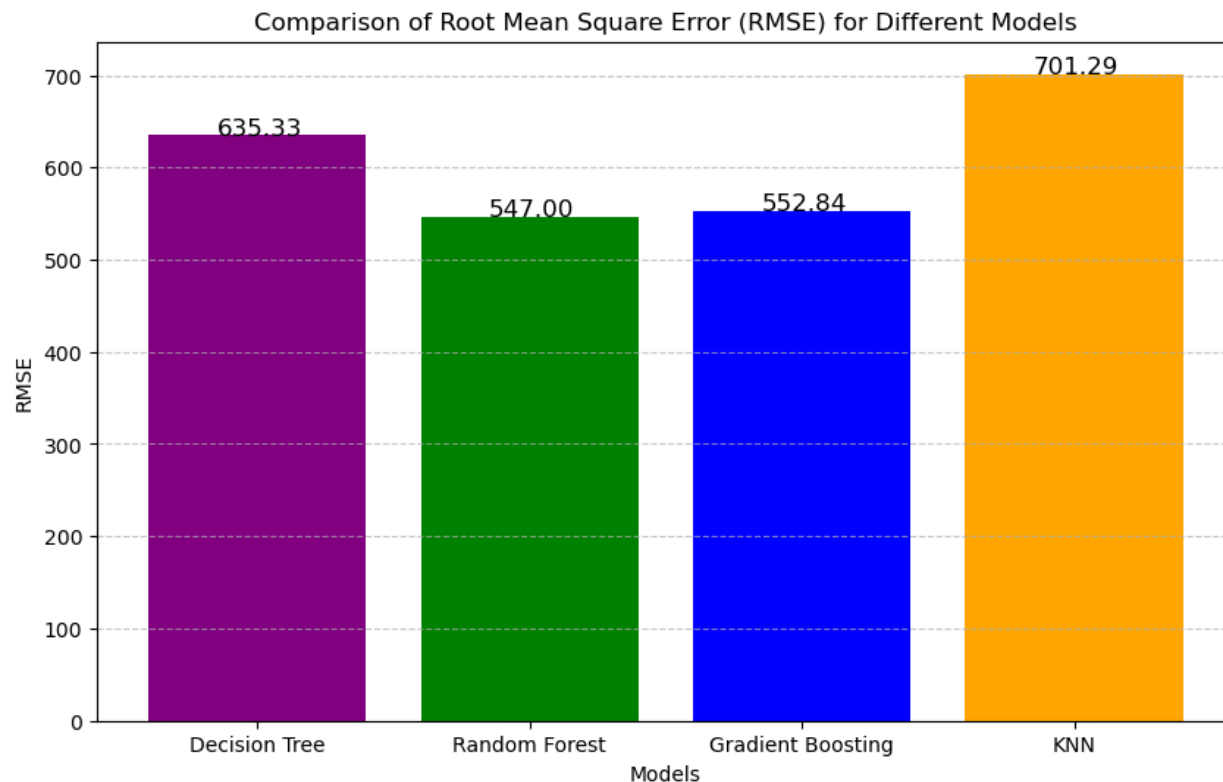
## 5.5 Mean Squared Error (MSE)



**Figure 5.5:** Barplot for comparing Mean Squared Error (MSE) of models.

**Explanation:** In **Figure 5.5** above, The Mean Squared Error (MSE) for Random Forest is 299204.55, whereas the MSE for KNN is 491811.43. Due to the difference in MSE values, it can be seen that the Random Forest model typically produces predictions with fewer squared errors, which suggests that, on average, its predictions are more accurate. In contrast, KNN tends to have larger squared errors due to its higher MSE, indicating that it predicts less precisely and accurately, which causes a wider difference between its predictions and the actual results.

## 5.6 Root Mean Squared Error (RMSE)



**Figure 5.6:** Barplot for comparing Root Mean Squared Error (RMSE) of models.

**Explanation:** From **Figure 5.6**, Random Forest has the lowest Root Mean Square Error (RMSE), 547, and KNN has the highest RMSE, 701.29. This difference in RMSE values demonstrates that, on average, the Random Forest model produces forecasts with fewer root mean squared errors, indicating that its predictions are more accurate. In contrast, KNN tends to have larger root mean squared errors because of its higher RMSE, which indicates less precision and accuracy in prediction and, thus, wider differences between its predictions and the actual outcomes.

## 5.7 Achievements

The project has successfully achieved its aim, all of the objectives stated before are achieved with an adequate level of results. The first objective is to collect a comprehensive dataset containing all relevant features and diamond price information and it is successfully achieved by carefully cleaning, processing, the data and also handling missing values, outliers and such. After achieving the first objective, the dataset became neat, clean, and just perfect for training an accurate predictive model.

Moreover, the second objective to find the best model by identifying the R square (main performance metric), Adjusted R square, Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE) is gracefully achieved. The best model that we have identified among all 4 regression models is the Random Forest Model. This model performs the best in predicting among all of them.

Lastly, our third objective is also reached flawlessly, we are able to deploy the selected predictive model with the best performance into a practical and user-friendly application for Shiny Diamond Company. It is now possible to predict diamond prices with high accuracy and with ease. The deployment involves creating a user interface (UI) or integrating the model into an existing system that allows stakeholders and customers to input diamond information and receive real-time price predictions.

## 5.8 Discussions

Discuss the limitations of the project and what improvements can be done in the future

The first limitation of our project is we are not experts in diamonds. The project's effectiveness may be constrained by a lack of subject matter expertise. This gap can be closed and decision-making can be strengthened by working with subject matter experts. Hence, the predictive ability of our project may be constrained by poor feature engineering or feature selection. Future developments may include more thorough feature selection techniques and domain knowledge to locate and create pertinent features.

Moreover, the scope and complexity of the project may be constrained by insufficient computer resources, such as insufficient processing power or memory. These restrictions can be removed by using cloud computing or purchasing better hardware.

Furthermore, under normal conditions, our predictive models may be effective, but they could be negatively impacted by unpredictable occurrences like economic downturns, natural disasters, or pandemics. These unanticipated occurrences may inject volatility and uncertainty into the data, reducing the precision of forecasts and maybe making the project less efficient during certain times. While it might not completely prevent the possible influence of extreme events, mitigating this constraint might entail embedding adaptable algorithms into the models or continuously monitoring external elements to make real-time adjustments when necessary.

## 6. DEPLOYMENT

The best model that we have identified, which is the Random Forest Regressor Model, will be used for the deployment of this project. First of all, we have created a function called `predict_price`. In this function, it will prompt the user to input the data that is used for predicting the diamond price. For instance, carat, cut, color, clarity, depth, table, x, y, z. For the categorical variables that are used to predict the diamond price, we will prompt the user to enter the encoded value that belongs to each category value. For example, if the user wants Fair for the cut of a diamond, then 0 should be entered. Options will be shown for each categorical value input when the user is prompted for input. Thus, the users just have to type in one of the options given. This provides user-friendliness and convenience to the users since they don't have to type a lot of information. In order to prevent wrong price prediction, validation functions are created to validate inputs. For instance, validation is implemented for the cut of diamond input, the user is only able to enter a value from 0 to 4, if the user enters other values, the user will be prompted again and again until the user enters the accepted value. After the user has keyed in all the required inputs, a data frame that contains the inputs of the user will be created. The data frame is then scaled prior to prediction, this helps to make the prediction more accurate. After scaling, it will proceed to make a prediction on the diamond price.

```
def validate_numeric_input(prompt, min_value=None, max_value=None):
    while True:
        try:
            value = float(input(prompt))
            if (min_value is None or value >= min_value) and (max_value is None or value <= max_value):
                return value
            else:
                if min_value is not None and max_value is not None:
                    print(f"Invalid input. Please enter a numerical value between {min_value}-{max_value}.")
                elif min_value is not None:
                    print(f"Invalid input. Please enter a numerical value greater than or equal to {min_value}.")
                elif max_value is not None:
                    print(f"Invalid input. Please enter a numerical value less than or equal to {max_value}.")
        except ValueError:
            print("Invalid input. Please enter a numerical value.")

def validate_categorical_input(prompt, valid_values):
    valid_range = f"({min(valid_values)}--{max(valid_values)})"
    while True:
        try:
            value = int(input(prompt))
            if value in valid_values:
                return value
            else:
                print(f"Invalid input. Please choose a number within the valid range {valid_range}.")
        except ValueError:
            print(f"Invalid input. Please enter a numerical value within the valid range {valid_range}.")
```

**Figure 6.1:** Creating the validation functions for inputs.



```
def predict_price():
    # Prompt the user for input values
    carat = validate_numeric_input("Enter carat: ")

    # Define valid ranges for cut, color, and clarity
    cut_options = {0: "Fair", 1: "Good", 2: "Very Good", 3: "Premium", 4: "Ideal"}
    color_options = {0: "D", 1: "E", 2: "F", 3: "G", 4: "H", 5: "I", 6: "J"}
    clarity_options = {0: "I1", 1: "SI2", 2: "SI1", 3: "VS2", 4: "VS1", 5: "VVS2", 6: "VVS1", 7: "IF"}

    # Prompt the user for categorical values with corresponding numeric options
    cut_num = validate_categorical_input(f"Enter cut {cut_options}: ", cut_options.keys())
    color_num = validate_categorical_input(f"Enter color {color_options}: ", color_options.keys())
    clarity_num = validate_categorical_input(f"Enter clarity {clarity_options}: ", clarity_options.keys())

    depth = validate_numeric_input("Enter depth: ")
    table = validate_numeric_input("Enter table: ")
    x = validate_numeric_input("Enter x: ")
    y = validate_numeric_input("Enter y: ")
    z = validate_numeric_input("Enter z: ")

    # Create a dictionary with the user input
    input_data = {
        'carat': [carat],
        'cut': [cut_num],
        'color': [color_num],
        'clarity': [clarity_num],
        'depth': [depth],
        'table': [table],
        'x': [x],
        'y': [y],
        'z': [z]
    }
}
```

**Figure 6.2:** Creating the function to prompt the user for inputs.

```
# Create a DataFrame from the input data
input_df = pd.DataFrame(input_data)

input_scaled = scaler.transform(input_df)

# Make the price prediction
predicted_price = model_randomForest.predict(input_scaled)

# Print the predicted price and input DataFrame
print("Predicted Price:$",predicted_price[0])
#     print(input_df)

# Call the function to predict the price based on user input
predict_price()
```

**Figure 6.3:** Scaling the input data and making a prediction on the diamond price.

```
Enter carat: 0.21
Enter cut {0: 'Fair', 1: 'Good', 2: 'Very Good', 3: 'Premium', 4: 'Ideal'}: 3
Enter color {0: 'D', 1: 'E', 2: 'F', 3: 'G', 4: 'H', 5: 'I', 6: 'J'}: 1
Enter clarity {0: 'I1', 1: 'SI2', 2: 'SI1', 3: 'VS2', 4: 'VS1', 5: 'VVS2', 6: 'VVS1', 7: 'IF'}: 2
Enter depth: 59.8
Enter table: 61
Enter x: 3.89
Enter y: 3.84
Enter z: 2.31
Predicted Price:$ 344.06
```

**Figure 6.4:** Make predictions based on the inputs

## 7. CONCLUSION

### 7.1 Advantages and disadvantages of models

In conclusion, the best model to use on this dataset is by using the Random Forest Regression Model. Despite the existence of outliers in some attributes, this model performs well and successfully captures the underlying trends in the data. It is appropriate for a wide range of regression issues because it may capture complex nonlinear correlations between features and the target variable. By merging the predictions of various decision trees, lowering overfitting, and producing reliable findings, it offers excellent predictive accuracy. Furthermore, it doesn't require considerable data preprocessing and is flexible enough to handle missing values as well as categorical and numerical data. However, it has limitations like longer training times. This is because the computational cost of Random Forests is high.(Mbaabu, 2020)

Decision trees' primary advantages include their simplicity and interpretability. They are a great option for explaining and communicating the decision-making process to non-technical stakeholders because they are simple to comprehend and visualize. However, it has its limitations too. The tendency they have to overfit the training data, particularly when the tree is deep and complicated, is a significant downside. Poor generalization to new, unforeseen facts may result from this. Second, Decision Trees can produce distinct tree architectures even when the data is slightly modified. When used with slightly different datasets, this volatility can make them less trustworthy for important decisions.

In the context of predictive modeling, the Gradient Boosting Regressor offers a number of noteworthy benefits. Its remarkable prediction accuracy is a crucial asset. It is a useful tool for precise predictions across a wide range of applications since it is excellent at capturing complicated correlations within the data. Gradient Boosting ensemble technique, which combines numerous weak models, also makes it resistant to overfitting. By preventing the model from fitting noise in the data, this ensemble learning strategy improves the model's generalization capabilities. On the other hand, its complexity and computational costs are a major disadvantage. Compared to simpler models, it can take more time and resources to train complex models. Working with enormous datasets or having few computational resources can make this difficult.

K-Nearest Neighbors (KNN) regressor has both advantages and disadvantages. Its simplicity and ease of use are two of its main advantages. KNN regression is an approachable option for those new to machine learning because it is simple to comprehend. It is also a non-parametric approach, which means it doesn't make firm assumptions about the distribution of the underlying

data, making it flexible to different regression issues. KNN regression also provides modeling versatility by capturing complex and nonlinear interactions between features and the target variable. In contrast, due to the disproportionate impact that extreme values can have on predictions, KNN regression is sensitive to outliers. Moreover, KNN regression, like its counterpart in classification, lacks interpretability and doesn't reveal insights into the significance of the features or the underlying patterns in the data, which can be a drawback when model interpretability is crucial.

## 7.2 Thoughts

In a nutshell, this assignment has been a worthwhile learning opportunity. We successfully applied our knowledge to develop a diamond price prediction model under the guidance of our beloved tutor, Miss Noor Aida, in the practical lessons. Even though we had trouble understanding and interpreting the dataset, our tutor's guidance and our group discussions helped us get through these difficulties. I am eager to take on more challenging projects and continue honing my data science skills in the future. In the future, I intend to deepen my comprehension of data analysis methods and use them to solve practical issues, ultimately making a contribution to the area of data science and producing insightful forecasts that can help numerous industries.

## References

- Fuey, G. S., & Idris, N. (2018). Assessing the Validity of the Elements for Pre-Service Mathematics Teacher Education Curriculum. *International Journal of Academic Research in Business and Social Sciences*, 7(12).  
<https://doi.org/10.6007/ijarbss/v7-i12/3611>
- Mbaabu, O. (2020, December 11). *Introduction to Random Forest in Machine Learning*. Section.  
<https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/>
- *G Color Grade Diamonds*. (n.d.). The Diamond Pro. Retrieved September 11, 2023, from <https://www.diamonds.pro/education/g-color-diamonds/>
- *Mastering Data Cleaning & Data Preprocessing for Machine Learning*. (n.d.). Encord.com. <https://encord.com/blog/data-cleaning-data-preprocessing/>
- *Handling Categorical Data in Python Tutorial*. (n.d.). Wwww.datacamp.com.  
<https://www.datacamp.com/tutorial/categorical-data>
- *Is an SI1 Clarity Diamond a Good Choice? - International Gem Society*. (2023, May 16). International Gem Society. <https://www.gemsociety.org/article/si1-clarity-diamond/>
- *Collecting Data | CYFAR*. (n.d.). Cyfar.org. <https://cyfar.org/collecting-data>
- Wada, K. (2020). Outliers in official statistics. *Japanese Journal of Statistics and Data Science*. <https://doi.org/10.1007/s42081-020-00091-y>

## APPENDIX

	A	B	C	D	E	F	G	H	I	J	K	L
1	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'		
2	0.23	b'Ideal'	b'E'	b'SI2'	61.5	55	326	3.95	3.98	2.43		
3	0.21	b'Premium'	b'E'	b'SI1'	59.8	61	326	3.89	3.84	2.31		
4	0.23	b'Good'	b'E'	b'VS1'	56.9	65	327	4.05	4.07	2.31		
5	0.29	b'Premium'	b'I'	b'VS2'	62.4	58	334	4.2	4.23	2.63		
6	0.31	b'Good'	b'J'	b'SI2'	63.3	58	335	4.34	4.35	2.75		
7	0.24	b'Very Good'	b'J'	b'VVS2'	62.8	57	336	3.94	3.96	2.48		
8	0.24	b'Very Good'	b'I'	b'VVS1'	62.3	57	336	3.95	3.98	2.47		
9	0.26	b'Very Good'	b'H'	b'SI1'	61.9	55	337	4.07	4.11	2.53		
10	0.22	b'Fair'	b'E'	b'VS2'	65.1	61	337	3.87	3.78	2.49		
11	0.23	b'Very Good'	b'H'	b'VS1'	59.4	61	338	4	4.05	2.39		
12	0.3	b'Good'	b'J'	b'SI1'	64	55	339	4.25	4.28	2.73		
13	0.23	b'Ideal'	b'J'	b'VS1'	62.8	56	340	3.93	3.9	2.46		
14	0.22	b'Premium'	b'F'	b'SI1'	60.4	61	342	3.88	3.84	2.33		
15	0.31	b'Ideal'	b'J'	b'SI2'	62.2	54	344	4.35	4.37	2.71		
16	0.2	b'Premium'	b'E'	b'SI2'	60.2	62	345	3.79	3.75	2.27		
17	0.32	b'Premium'	b'E'	b'I1'	60.9	58	345	4.38	4.42	2.68		
18	0.3	b'Ideal'	b'I'	b'SI2'	62	54	348	4.31	4.34	2.68		
19	0.3	b'Good'	b'J'	b'SI1'	63.4	54	351	4.23	4.29	2.7		
20	0.3	b'Good'	b'J'	b'SI1'	63.8	56	351	4.23	4.26	2.71		
21	0.3	b'Very Good'	b'J'	b'SI1'	62.7	59	351	4.21	4.27	2.66		
22	0.3	b'Good'	b'I'	b'SI2'	63.3	56	351	4.26	4.3	2.71		
23	0.23	b'Very Good'	b'E'	b'VS2'	63.8	55	352	3.85	3.92	2.48		
24	0.23	b'Very Good'	b'H'	b'VS1'	61	57	353	3.94	3.96	2.41		
25	0.31	b'Very Good'	b'J'	b'SI1'	59.4	62	353	4.39	4.43	2.62		
26	0.31	b'Very Good'	b'J'	b'SI1'	58.1	62	353	4.44	4.47	2.59		

Figure 9.1 above shows the dataset of diamonds, cut,color,clarity,depth and more.

```

Data Understanding & EDA & Visualization

In [2]: data = pd.read_csv("diamonds.csv")
        data.head()

Out[2]:
   carat    cut  color clarity  depth  table  price    'x'    'y'    'z'
0   0.23  b'Ideal'  b'E'  b'SI2'   61.5   55.0   326.0   3.95   3.98   2.43
1   0.21  b'Premium'  b'E'  b'SI1'   59.8   61.0   326.0   3.89   3.84   2.31
2   0.23    b'Good'  b'E'  b'VS1'   56.9   65.0   327.0   4.05   4.07   2.31
3   0.29  b'Premium'  b'I'  b'VS2'   62.4   58.0   334.0   4.20   4.23   2.63
4   0.31    b'Good'  b'J'  b'SI2'   63.3   58.0   335.0   4.34   4.35   2.75

In [3]: data.tail()

Out[3]:
   carat    cut  color clarity  depth  table  price    'x'    'y'    'z'
53935  0.72  b'Ideal'  b'D'  b'SI1'   60.8   57.0   2757.0   5.75   5.76   3.50
53936  0.72    b'Good'  b'D'  b'SI1'   63.1   55.0   2757.0   5.69   5.75   3.61
53937  0.70  b'Very Good'  b'D'  b'SI1'   62.8   60.0   2757.0   5.66   5.68   3.56
53938  0.86  b'Premium'  b'H'  b'SI2'   61.0   58.0   2757.0   6.15   6.12   3.74
53939  0.75  b'Ideal'  b'D'  b'SI2'   62.2   55.0   2757.0   5.83   5.87   3.64

In [4]: data.shape
Out[4]: (53940, 10)

```

Figure 9.2 shows the sample of python file