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: Diamond Price Prediction
Programme and Tutorial Class: RDS G4
Tutor Name: NOOR AIDA BINTI HUSAINI
Submission Date: 17 September 2023
Team Member:

[redacted] **Student Name: TAI QI ZHENG** **Student ID:** [redacted] **Module-in-Charge:** **1.0, 2.0, 4.0, 5.0, 6.0**	[redacted]
[redacted]	 **Student Name:** **Student ID:** **Module-in-Charge:**

TABLE OF CONTENT

1.0 BUSINESS UNDERSTANDING

1.1 Company's Background

CupidJewel is a diamond jewelry company founded by a pair of couples, Amy and Josh in 2000. The business idea started when Josh was looking for the perfect engagement ring, he had difficulty finding a ring that was beautiful and affordable. He felt that most jewelry companies prioritized making profit over producing quality diamond jewelry that was actually worth the price tag. Hence, Josh decided to design the diamond ring for their engagement and started a business selling diamonds and jewelry shortly after.

Just after a few years into the diamond industry, CupidJewel stood on par with the other organization and became one of the leaders within the industry. Since then, CupidJewel has expanded and established multiple branches around the world such as Malaysia, Indonesia, Thailand, China, South Korea, Singapore and Philippines. The type of diamond jewelry event offered is vast and it includes diamond jewelry, including engagement rings, necklaces, earrings, and bracelets. The company also offers a variety of services, such as custom jewelry design and diamond grading and engraving services. CupidJewel's mission is to help people celebrate their love by finding the perfect diamond jewelry for their loved ones. It is committed to provide its customers with the most top tiered diamonds and jewelry at best prices.

To achieve that, they hired our team of data scientists to help them build a machine learning model that can predict the price of diamond based on its attributes. With the accurate prediction of diamond price, CupidJewel are able to ensure diamonds are neither undervalued or overvalued.

1.2 Aims

Our company's aim is to help us make informed decisions about the values of diamonds. By knowing the predicted price of a diamond, CupidJewel can sell the diamonds at realistic prices. Furthermore, the company can use this model to identify the trends in the diamond market. This information can be used to make predictions about the diamonds' price based on its characteristics. Besides, this model can enhance operational efficiency since we can reduce the amount of analysts to analyze the diamonds and its price. They can save time on lower level tasks researching the price based on the characteristics of a diamond and focus on making high level tasks such as developing marketing strategies, negotiating deals and building relationships with customers.

1.3 Objectives

The primary objective of our company is to develop a predictive model that is capable of predicting the price of diamonds based on their characteristics using the dataset. By doing so, it can help CupidJewel to make informed decisions when selling the diamonds. The predictive model will use a machine algorithm to make the predictions of diamond price.

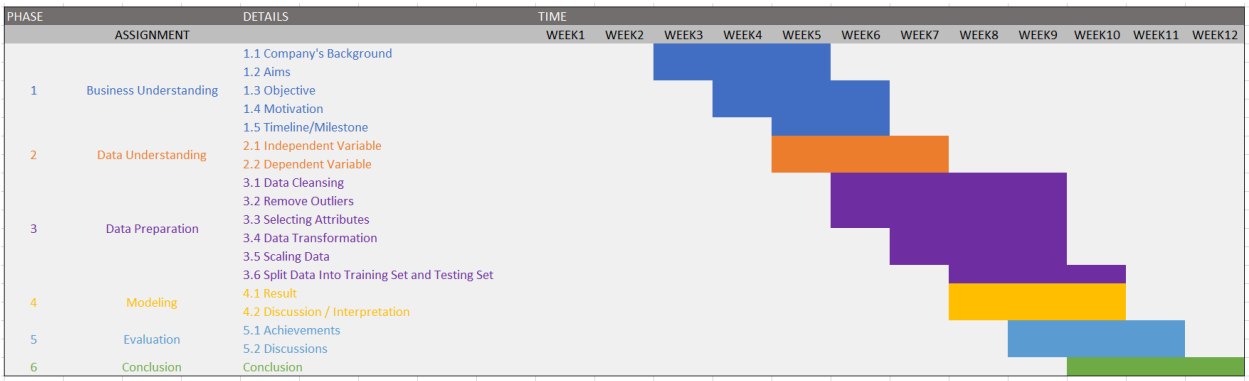
Secondly, it is also our objective to identify which machine algorithm is most efficient and accurate in making diamond price prediction. For this purpose, training and evaluation of different algorithms on our dataset by comparing their performance metrics such as R Square, Adjusted R Square, Mean Absolute Error, Mean Square Error, Root Mean Square. This ensures that our predictive model is using the best machine learning algorithm available to us. Among the machine algorithms evaluated in this assignment are Support Vector Machine, Random Forest and Decision Tree. Lastly, the objective will be finding the relationship between diamonds' characteristics and its market values. Understanding these relationships will allow us to leverage this knowledge to buy diamonds in a good deal.

1.4 Motivation

The commercial value of the diamonds prediction is to improve efficiency of the diamonds market. By applying our diamonds prediction, it would need a lesser expert to predict which could take some time for each expert to evaluate the diamonds. Lesser expert to do evaluation meaning lesser costs spent on human experts. This could lead to a lower transaction cost and faster turnaround time since it requires less diamond experts to interact with the diamonds. Furthermore, diamond prediction will be able to reduce the risk of buyers or sellers to buy overestimated diamonds in the market. By using our diamonds prediction, the buyer or seller will be more informed to be able to accurately predict fair market value of the diamonds.

Nevertheless, there are many positive social impacts of this diamond price prediction model. These social impacts include reducing fraud and corruption in the diamond industry as well as promoting sustainable mining practices. In terms of fraud and corruption in the industry, consumers can be assured that they are getting the fair price for their diamonds. Diamond miners can also optimize their production strategies which can minimize the environmental impact of mining. In addition, diamond investors can use diamond price prediction models to make more informed investment decisions. Through usage of this model, they can identify diamonds that are likely to increase in value over time which helps them to generate higher returns on their investments.

1.5 Timeline/Milestone



PHASE	ASSIGNMENT	DETAILS
1	Business Understanding	1.1 Company's Background 1.2 Aims 1.3 Objective 1.4 Motivation 1.5 Timeline/Milestone
2	Data Understanding	2.1 Independent Variable 2.2 Dependent Variable
3	Data Preparation	3.1 Data Cleansing 3.2 Remove Outliers 3.3 Selecting Attributes 3.4 Data Transformation 3.5 Scaling Data 3.6 Split Data Into Training Set and Testing Set
4	Modeling	4.1 Result 4.2 Discussion / Interpretation
5	Evaluation	5.1 Achievements 5.2 Discussions
6	Conclusion	Conclusion

2.0 DATA UNDERSTANDING

For this project, the dataset chosen is diamonds.csv. This dataset contains only 1 file with a table of 10 variables (columns) and 53940 observations (rows). In this dataset, there are 7 variables of float64 data type and the remaining 3 variables are object data type.

Name	Type	Dependent / Independent Variable	Description
CARAT	Continuous	Independent Variable	Carat is the unit to measure the physical weight of diamonds. 1 carat or 'ct' equals to ⅕ gram.
CUT	Categorical (Ordinal)	Independent Variable	Diamond cut means how well a diamond's facets interact with light.
COLOR	Nominal	Independent Variable	Color measures how the diamond reflects or emit light
CLARITY	Ordinal	Independent Variable	Clarity is to measure the purity and rarity of the stone, graded by the visibility of these characteristics under 10-power magnification.
DEPTH	Continuous	Independent Variable	Depth is to measure the height(mm) of the diamond. To measure we need to start measuring from the bottom tip (culet) to the flat top surface (table).
TABLE	Continuous	Independent Variable	A diamond's table is the facet which can be seen when the stone is viewed face up. It is measured as a percentage.
PRICE	Continuous	Dependent Variable	Price of the diamonds
'X'	Continuous	Independent Variable	X fracture, diameter(mm) of diamond
'Y'	Continuous	Independent Variable	Y fracture, width(mm) of diamond
'Z'	Continuous	Independent Variable	Z fracture, pavilion depth(mm) of diamond. Measured from girdle to culet

Table 2.1 Characteristic of each features

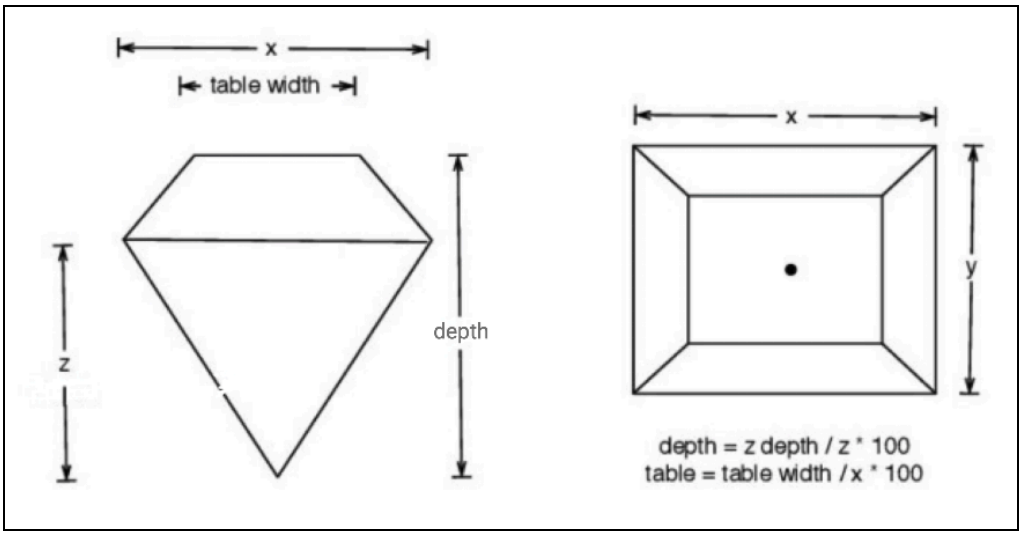


Figure 2.1 table, 'x', 'y', 'z' of a diamond

```
# Plot the correlation matrix
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
plt.show()
```

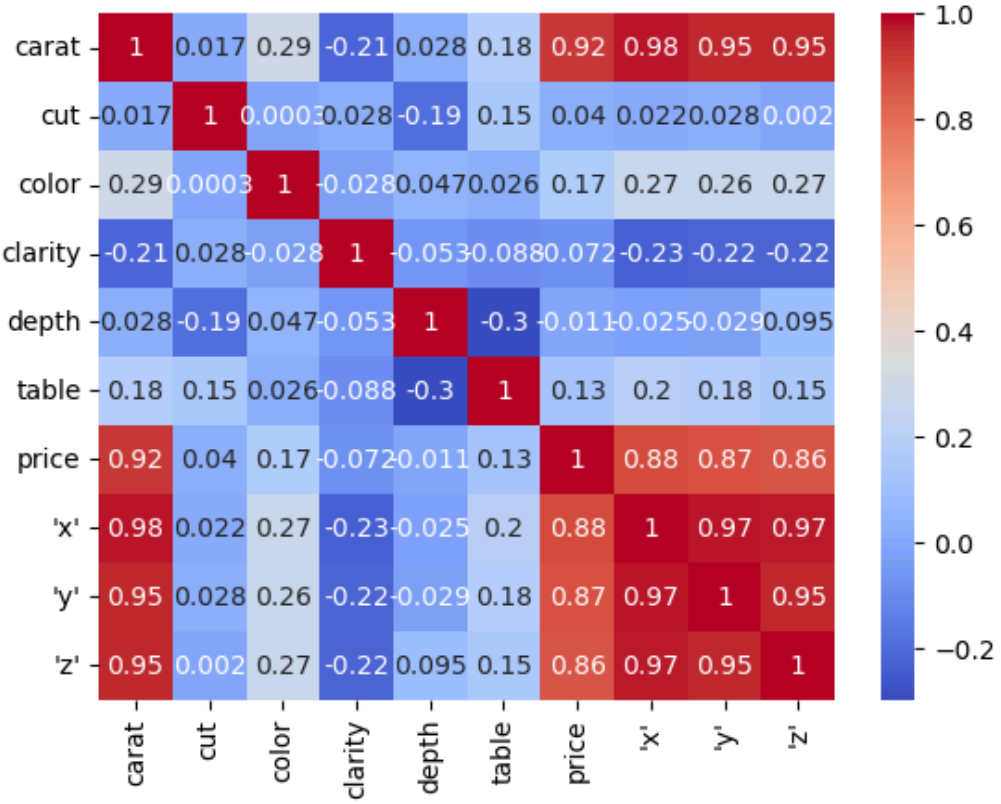


Figure 2.2 The Correlation Matrix

Based on Figure 2.2, it is evident that the relationship between carat, price, ‘x’, ‘y’, ‘z’ has high correlation while other variables such as cut, color, clarity, depth and table have low correlation. The relationship between these variables will be further discussed in the following section.

2.1. Independent Variable

2.1.1. Carat

Figure 2.3 Types of diamond carat
(Source: *Learn what carat means and what diamond carat measures | 4Cs of Diamond Quality by GIA*, 2019)

```
Carat = data['carat'].describe()

print('Carat')
print('======================')
print(Carat)
```

```
Carat
======================
count    53940.000000
mean         0.797940
std          0.474011
min          0.200000
25%          0.400000
50%          0.700000
75%          1.040000
max          5.010000
Name: carat, dtype: float64
```

Figure 2.4 Statistical data for Carat

```
plt.subplots(figsize=(20,10))
sns.histplot(data["carat"])
plt.show()
```

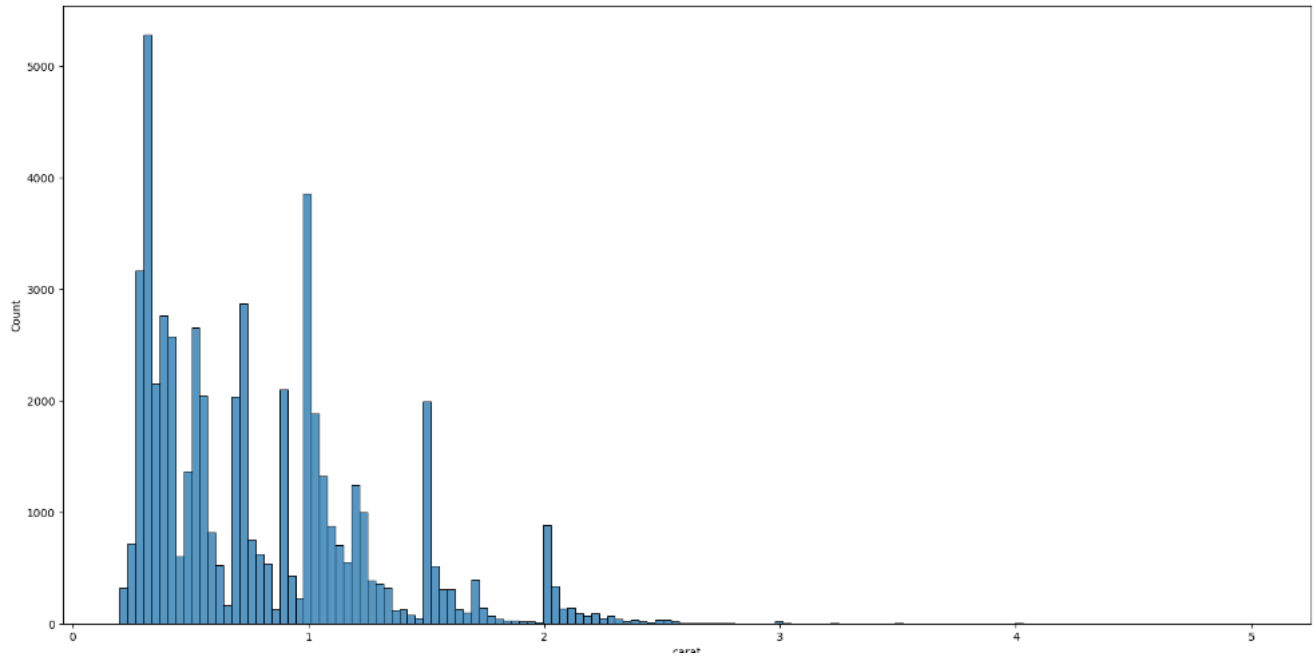


Figure 2.5 Histogram of Carat

Carat is a variable that measures the weight of the diamonds. For carats the type of the variable is continuous. Based on Figure 2.2, it can be seen that the correlation between the price of diamonds and the carat of diamonds are very high (0.92%). This is because the carat of a diamond has a significant impact on its price. For instance, diamonds with higher carat value tend to be larger in size which are considered more valuable and rare than smaller diamonds (International Gem Society LLC, 2023). Hence, the larger the diamond, the higher the price for it.

From Figure 2.5, it is evident that the count is notably high when carats of diamond are at 0.3ct and 1.0ct. Since 0.3ct is considered as small sized diamonds, the price for it might be more affordable for the public. On the other hand,1.0ct falls under the category of medium sized diamonds; it could most likely be purchased by individuals from the upper class as such diamonds act as a status symbol or it could be bought as engagement or wedding rings by couples.

2.1.2. Cut

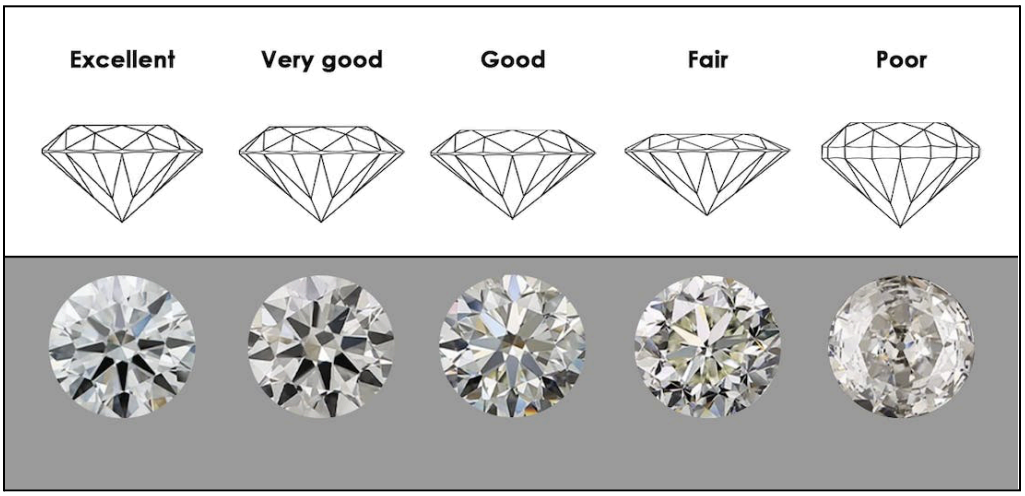


Figure 2.6 Difference types of cut in diamond
(Source: *All about Cut | My Diamond Ring*, 2021)

```
Cut = data['cut'].describe()

print('Cut')
print('======================')
print(Cut)
```

```
Cut
======================
count        53940
unique           5
top       b'Ideal'
freq         21551
Name: cut, dtype: object
```

Figure 2.7 Statistical data for Cut

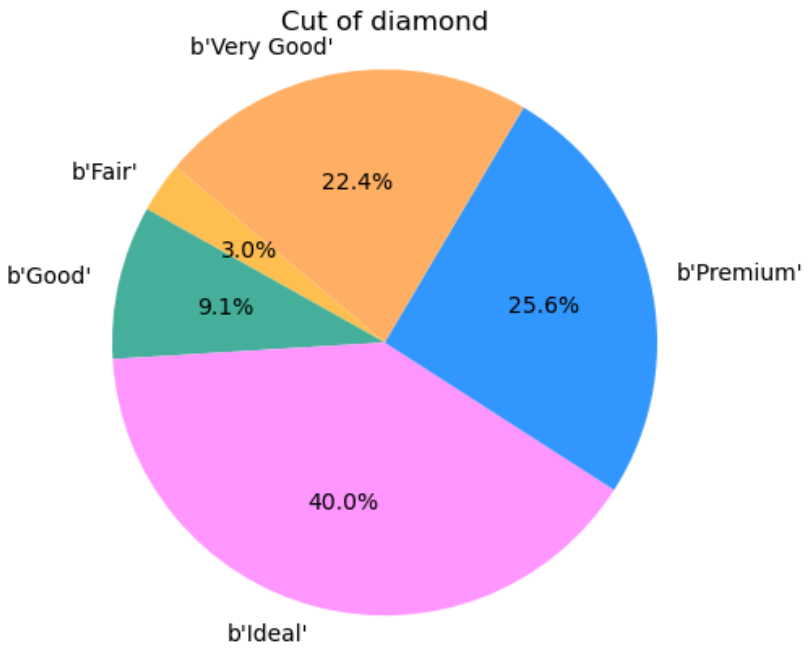


Figure 2.8 Histogram of Cut

Cut represents how well a diamond's facets interact with light. For Cut, the type of the variable is ordinal. Based on Figure 2.2, it is evident that the correlation between the price of diamonds and the cut of diamonds is very low. The explanation for this could be due to the quantifiability of diamond cut. Unlike carat, there does not exist a universally accepted standard for grading diamond cut, this makes it harder to understand how the cut affects the price. Thus, a diamond with a good cut may be more expensive than one with a poor cut but the difference in price may not be as significant.

According to Figure 2.8, it can be seen that the percentage for cut b'Ideal' is the highest. This could be because it is the middle ground for diamonds which are better than b'Good' but lower in cut quality than b'Premium'. The price for these diamonds could be more affordable, which explains the higher count. On the contrary, b'Fair' has the least percentage which could be attributed to the diamond's poor performance in reflecting light, resulting in a less attractive diamond.

2.1.3. Color

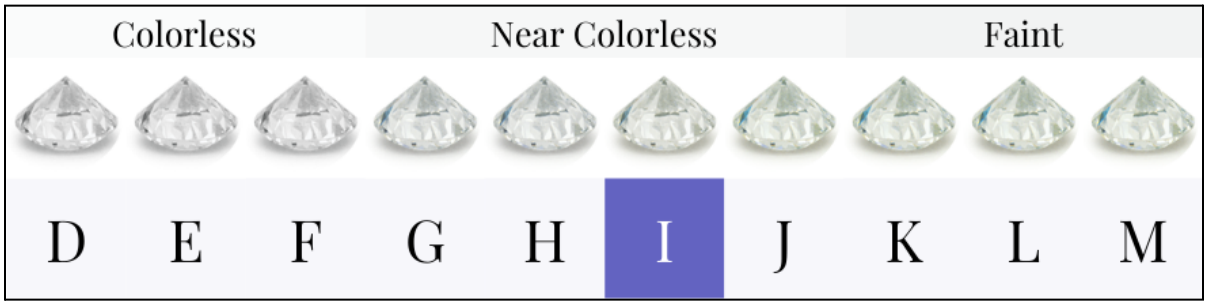


Figure 2.9 Diamond color grade scale
(Source: Devin, 2022)

```
Color = data['color'].describe()

print('Color')
print('======================')
print(Color)
```

```
Color
======================
count     53940
unique        7
top         b'G'
freq      11292
Name: color, dtype: object
```

Figure 2.10 Statistical data for Color

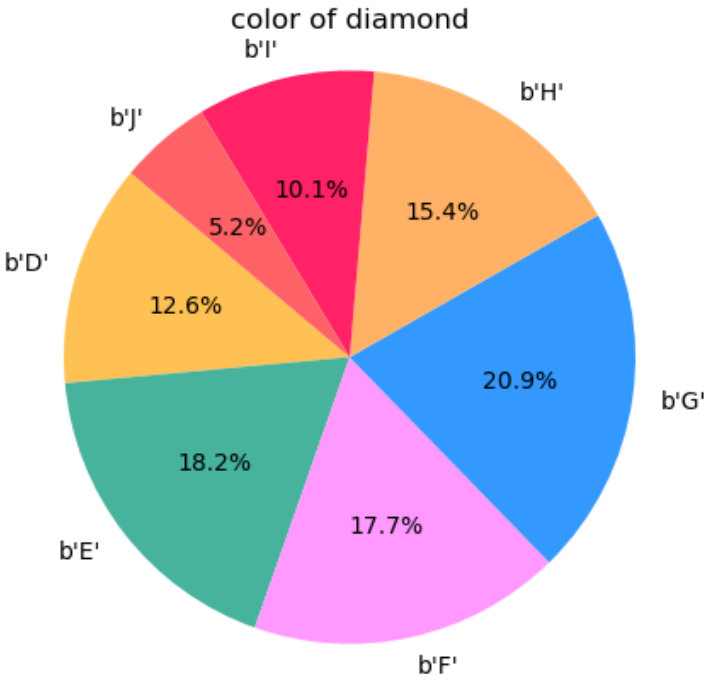


Figure 2.11 Histogram of Color

Color of diamond is to measure how the diamond reflects or emits light. For Color, the type of the variable is nominal. From Figure 2.2, it can be seen that the correlation between the price of diamonds and the color of diamonds is very low. This could be due to the minimal difference between colorless and faint diamonds. However, people are willing to pay a premium price for slight slight color improvement.

Furthermore, Figure 2.11 shows that the color grade with highest percentage is b'G' while the lowest percentage is b'j'. From this, we can conclude that the majority of the individuals prefer diamonds that are in between colorless and near colorless. Another assumption is that b'j' is not people's choice due to it having a slight color tint.

2.1.4. Clarity

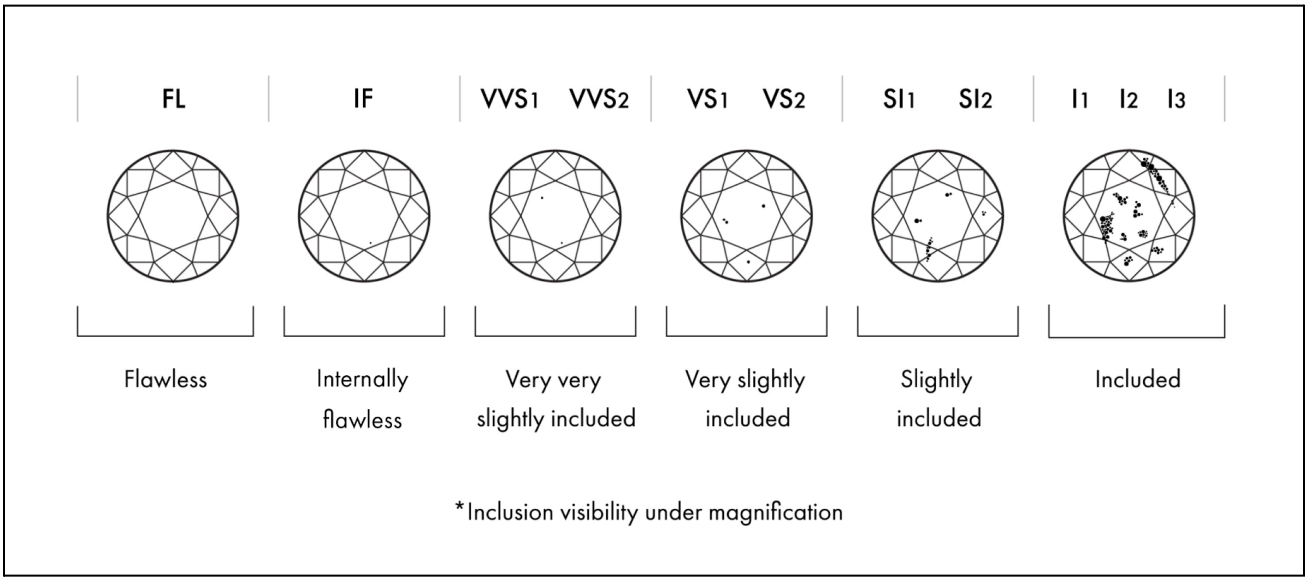


Figure 2.12 Diamond Clarity Chart
(Source: Vrai, 2023)

```
clarity = data['clarity'].describe()

print('Clarity')
print('================')
print(clarity)
```

```
Clarity
================
count      53940
unique         8
top       b'SI1'
freq       13065
Name: clarity, dtype: object
```

Figure 2.13 Statistical data for Clarity

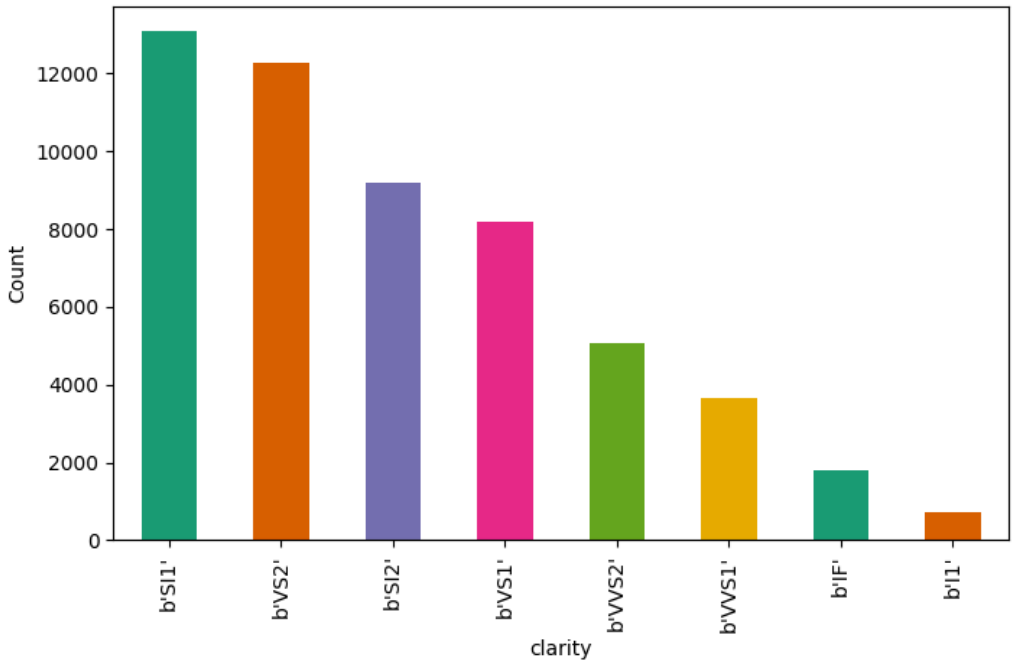


Figure 2.14 Histogram of Clarity

Clarity of diamond is to measure the purity and rarity of the stone, graded by the visibility of these characteristics under 10-power magnification. The variable type for clarity is ordinal. It can be seen from Figure 2.2 that the correlation between the price of diamonds and the clarity of diamonds is the lowest among all. This could be because of consumer's behavior as they are more concerned about other factors such as carat or color. In addition, diamond inclusions are not visible to the human eye, hence, it can be easily dismissed by consumers.

Nevertheless, it is evident that diamond with clarity b'Sl1' has the highest number of counts (>12k) by observing Figure 2.14. Another observation is that the top three clarity are b'Sl1', b'VS2' and b'Sl2', which implies that diamonds with such clarity are much more marketable as they are in demand.

2.1.5. Depth

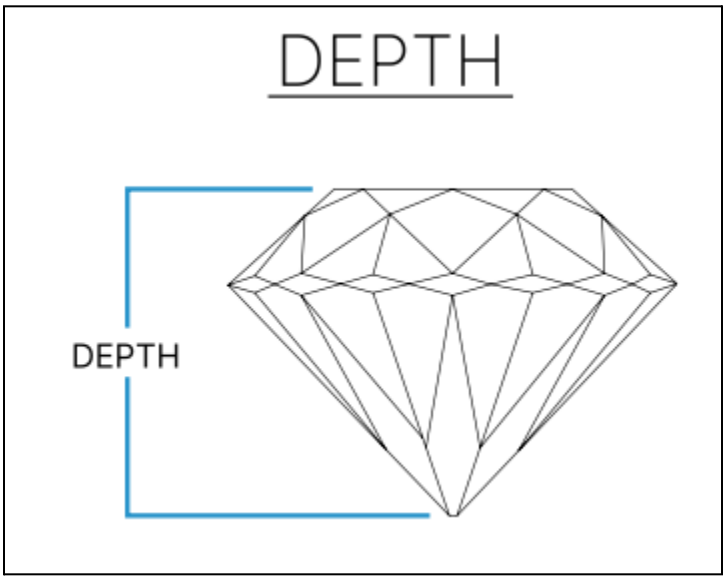


Figure 2.15 Depth of a diamond
(Source: Shiv Shambu, n.d.)

```
depth = data['depth'].describe()

print('depth')
print('================')
print(depth)
```

```
depth
================
count    53940.000000
mean        61.749405
std          1.432621
min         43.000000
25%         61.000000
50%         61.800000
75%         62.500000
max         79.000000
Name: depth, dtype: float64
```

Figure 2.16 Statistical data of Depth

```
plt.subplots(figsize=(10,5))
sns.histplot(data["depth"])
plt.show()
```

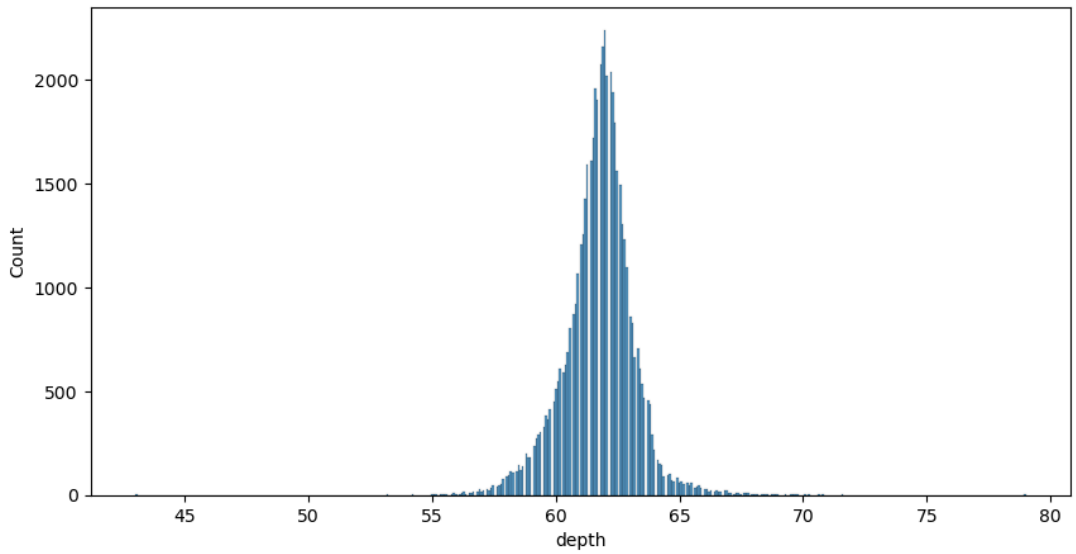


Figure 2.17 Histogram of Depth

Depth of diamond is to measure the height of the diamond. To measure we need to start measuring from the bottom tip (culet) to the flat top surface (table) by using mm. For depth the type of the variable is continuous. Based on Figure 2.2, it is evident that the price of diamonds and the depth of diamonds have a low correlation. This is because it is not a major determining factor in determining a diamond's overall quality. Instead, the diamond price is largely affected by the 4Cs which are cut, color, clarity and carat.

Besides, it can be observed from Figure 2.17 that the depth with highest count is 62 which means that most of the diamonds have depth of 62mm. It is also noted the distribution of depth is almost a normal distribution. Hence, data near mean depth which is 62 will occur more frequently.

2.1.6. Table

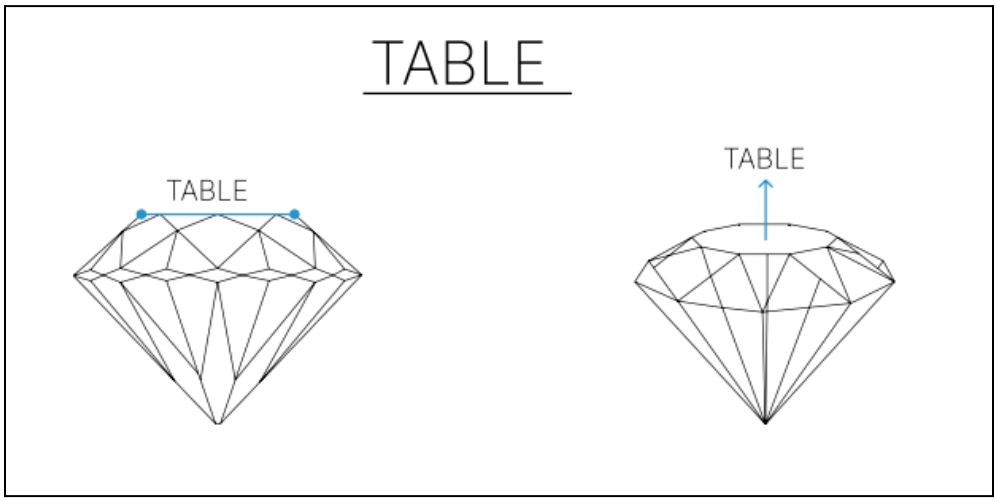


Figure 2.18 Table of a diamond
(Source: Shiv Shambu, n.d.)

```
table = data['table'].describe()

print('table')
print('================')
print(table)
```

```
table
================
count    53940.000000
mean        57.457184
std          2.234491
min         43.000000
25%         56.000000
50%         57.000000
75%         59.000000
max         95.000000
Name: table, dtype: float64
```

Figure 2.19 Statistical data of diamond

```
plt.subplots(figsize=(10,5))
sns.histplot(data["table"])
plt.show()
```

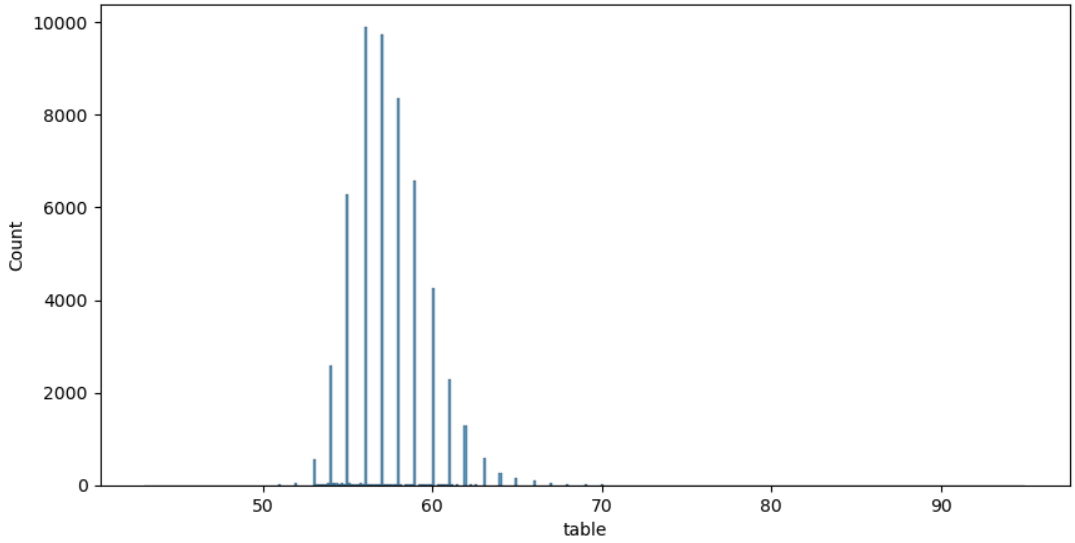


Figure 2.20 Histogram of Table

Table of diamonds refers to the width of a diamond's table expressed as average diameter in percentage and is calculated by dividing diameter by average girdle diameter. It determines the diamond's degree of brilliance. The variable table and price has a low correlation from Figure 2.2, as the table of a diamond is not a good indicator of a diamond's quality. For instance, a diamond with a large table may have low color or clarity grade.

Based on Figure 2.20, the distribution graph is slightly skewed to the right which indicates the presence of outliers at the right side of distribution. Therefore, median will be used instead of mean to calculate the center of distribution. As a result, diamonds with a 57.00% table have the highest count.

2.1.7. ‘x’

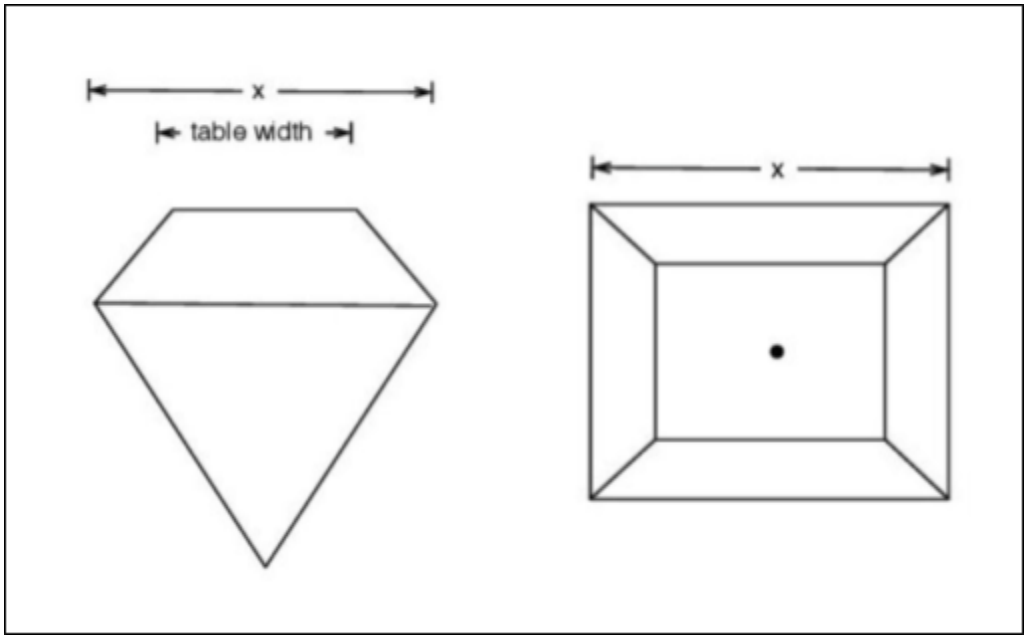


Figure 2.21 ‘x’ of a diamond

```
x = df["'x'"].describe()

print('x')
print('======================')
print(x)
```

```
x
======================
count    53940.000000
mean         5.731157
std          1.121761
min          0.000000
25%          4.710000
50%          5.700000
75%          6.540000
max         10.740000
Name: 'x', dtype: float64
```

Figure 2.22 Statistical data of ‘x’

```
sns.histplot(df["'x'"])
plt.show()
```

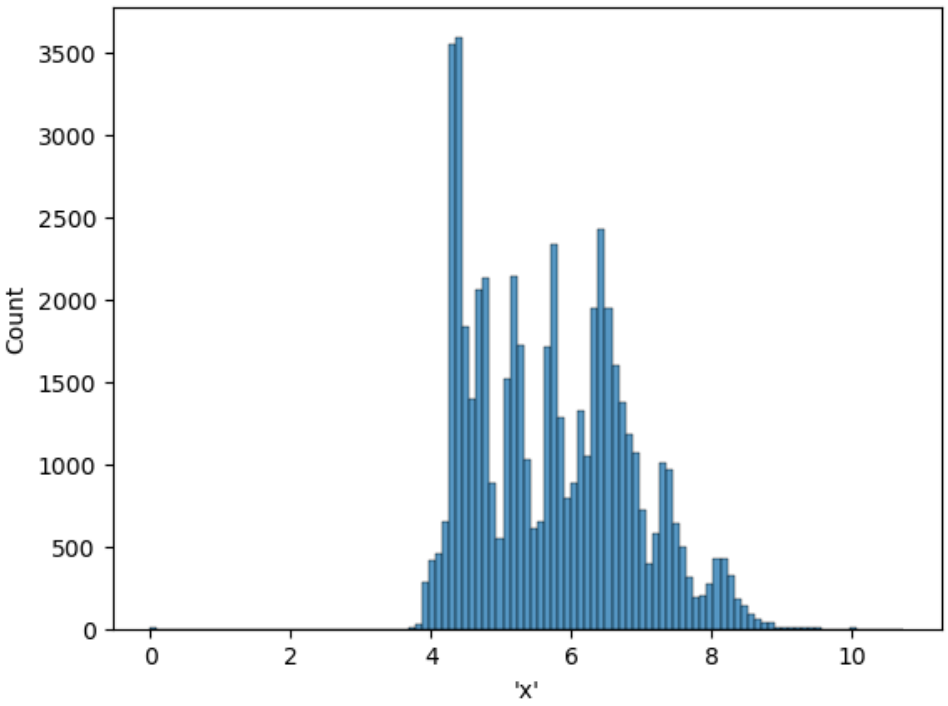


Figure 2.23 Histogram of 'x'

'x' is to measure the diameter of a diamond in millimeters. This 'x' variable is in continuous form. Based on Figure 2.2, it can be seen that 'x' has a very strong correlation with the price of diamond. Given that the length of a diamond affects the overall size, when a diamond is longer in length, the diamond size is larger. Due to the rarity of larger diamonds, the price is also higher for large diamonds than smaller ones.

According to Figure 2.22, the min for 'x' is '0.000000'. This may be due to human input error whereby un-identified 'x' is temporarily replaced with "0.000000". The handling of this missing value will be explained in the data preparation step. On the other hand, Figure 2.23 shows a graph with positive distribution. Hence, the central tendency obtained is 5.7 using median which means that most diamonds have length of 5.7mm. Median was used as it can provide a more accurate description for our dataset where the distribution is skewed to the right, unlike mean which is affected by outliers.

2.1.8. ‘y’

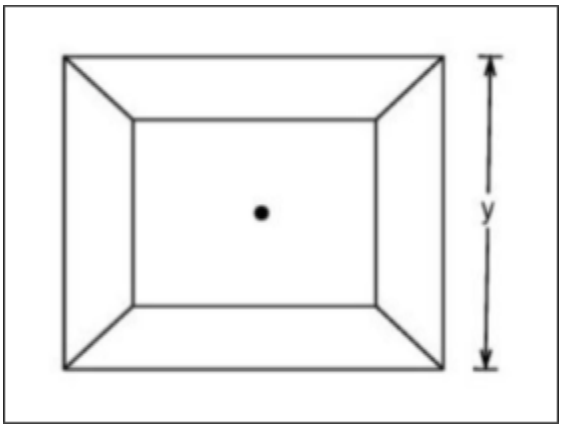


Figure 2.24 ‘y’ of a diamond

```
y = df["'y'"].describe()

print('y')
print('======================')
print(y)
```

```
y
======================
count    53940.000000
mean         5.734526
std          1.142135
min          0.000000
25%          4.720000
50%          5.710000
75%          6.540000
max         58.900000
Name: 'y', dtype: float64
```

Figure 2.25 Statistical data of ‘y’

```
sns.histplot(df["'y'"])
plt.show()
```

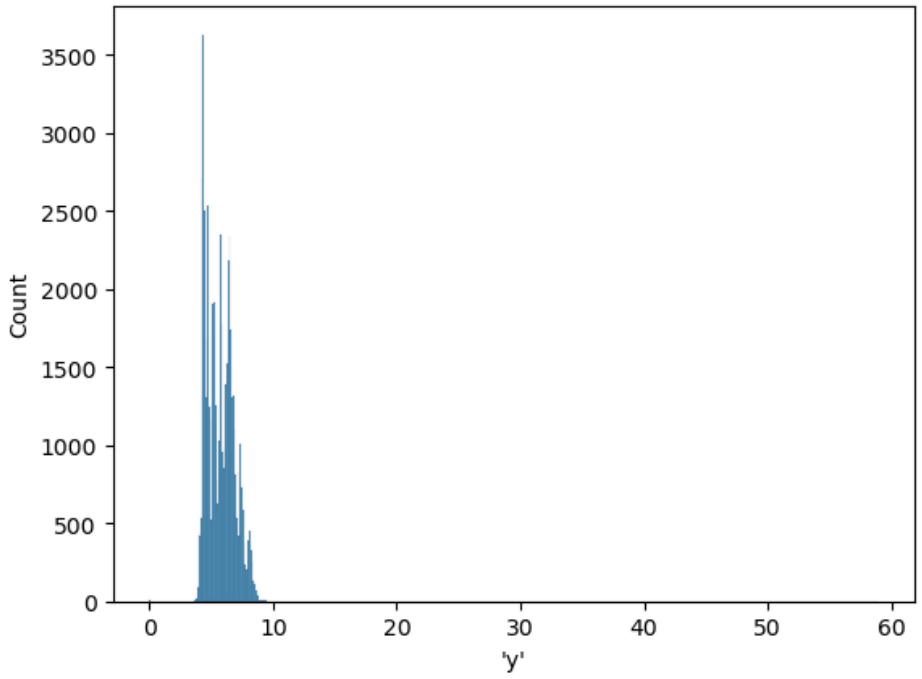


Figure 2.26 Histogram of 'y'

'y' is used to measure the width of a diamond in millimeters. The type of this variable is in continuous form. According to Figure 2.2, the correlation coefficient between price and 'y' has a very strong relationship with the value of 0.87. Similar to 'x', 'y' as the width has a huge impact on price due to its effect on the size of diamond.

In reference to Figure 2.23, the min for 'y' is '0.000000' which could be caused by human input error whereby "0.000000" is used as a temporary replacement for unknown value. The handling of this missing value will be explained in the data preparation step. Furthermore, the graph in Figure 2.24 shows a positive distribution. Since our distribution is skewed to the right, median was used to provide a more accurate description of the dataset because it is unaffected by outliers. Therefore, the central tendency obtained is 5.71, this implies that most diamonds have a width of 5.71mm.

2.1.9. ‘z’

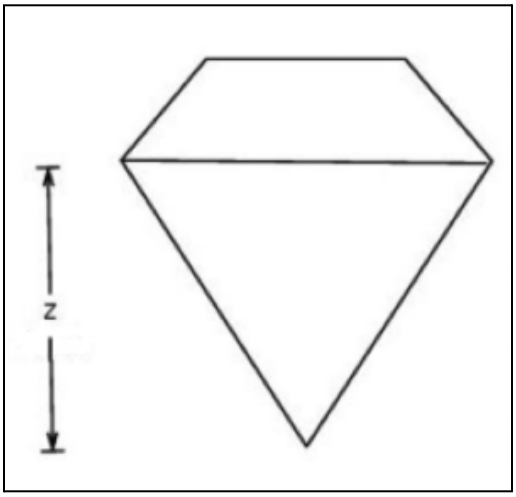


Figure 2.27 ‘z’ of a diamond

```
z = df["'z'"].describe()

print('z')
print('======================')
print(z)
```

```
z
======================
count    53940.000000
mean         3.538734
std          0.705699
min          0.000000
25%          2.910000
50%          3.530000
75%          4.040000
max         31.800000
Name: 'z', dtype: float64
```

Figure 2.28 Statistical data of ‘z’

```
sns.histplot(df["'z'"])
plt.show()
```

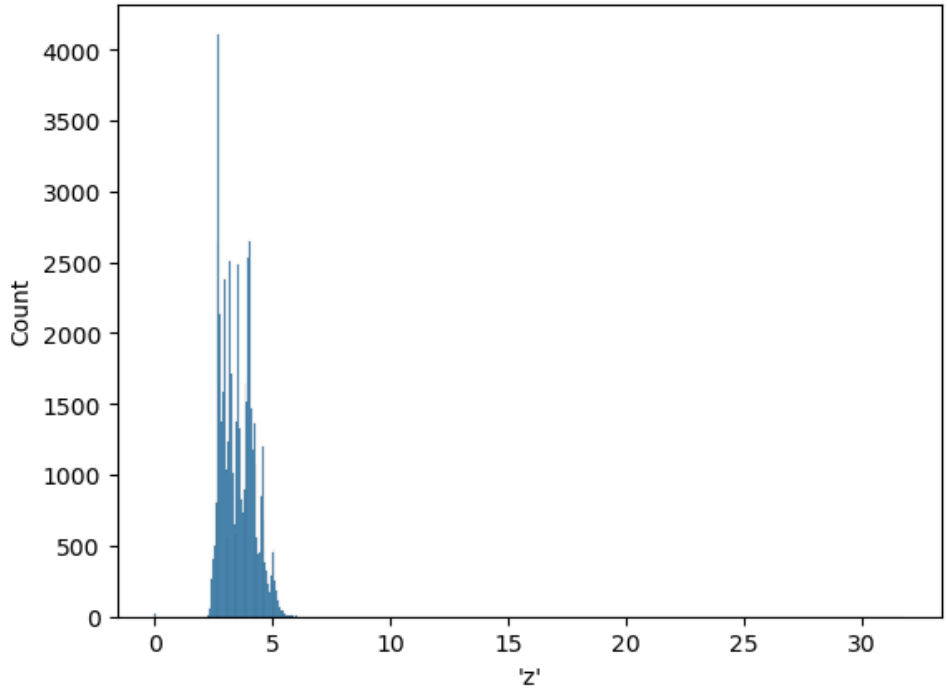


Figure 2.29 Histogram of 'z'

'z' is used to measure the pavilion depth of a diamond in millimeters . The type of this variable is in continuous form. From Figure 2.2, the correlation coefficient between price and 'z' has a very strong relationship with the value of 0.86. Similar to 'x' and 'y', 'z' also has a large influence on the price of diamond because it determines the depth of the diamond which affects the overall diamond size.

In reference to Figure 2.28, human input error could have been the cause for the min value of 'z' being "0.000000". The handling of this missing value will be explained in the data preparation step. Moreover, the graph in Figure 2.29 shows a positive distribution. So, the central tendency obtained is 3.53 using median which means that most diamonds have a pavilion depth of 3.53 mm. Median was used as it can provide a more accurate description than mean because mean is affected by outliers.

2.2. Dependent Variable

2.2.1. Price

```
price = data['price'].describe()

print('Price')
print('================')
print(price)
```

```
Price
================
count    53940.000000
mean      3932.799722
std       3989.439738
min        326.000000
25%        950.000000
50%       2401.000000
75%       5324.250000
max      18823.000000
Name: price, dtype: float64
```

Figure 2.30 Statistical data of Price

```
plt.subplots(figsize=(10,5))
sns.histplot(data["price"])
plt.show()
```

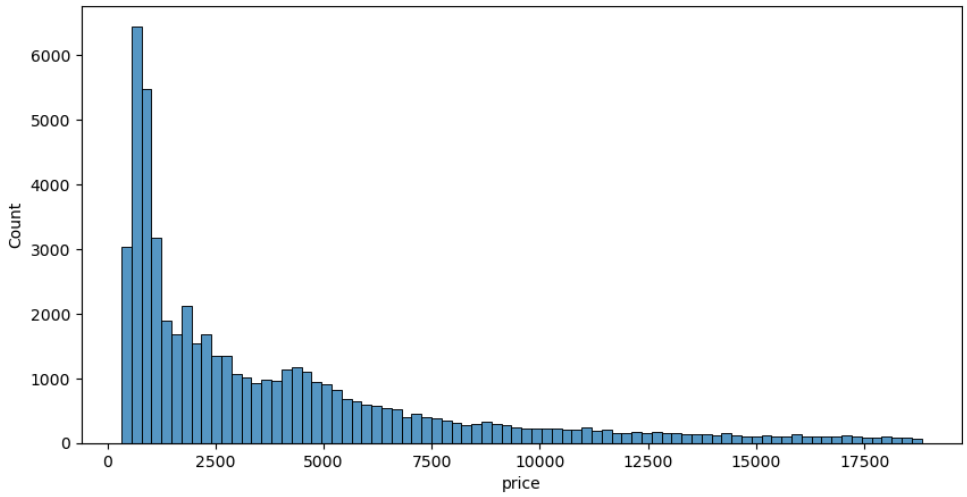


Figure 2.31 Histogram of Price

"Price" refers to the price of a diamond. It is selected as the dependent variable because it meets the objective of our assignment which is to create a diamond price prediction model. Figure 2.31 shows that most diamond's price ranges are usually below 2500 USD. This variable has the distribution that's skewed to the right. Thus, median was used to provide a more accurate description of the dataset because it is unaffected by outliers. Using the median, the central tendency obtained is 2401.0, this implies that most diamonds cost 2401.00 USD.

3.0 DATA PREPARATION

In this section, the dataset is prepared to predict the price of the diamond. The purpose of data preparation is to ensure the quality of all the data to perform better accuracy and consistency for the results. The dataset used is diamond.csv

3.1. Data Cleansing

	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 3.1 Statistical data of the dataset

```
[ ]  df.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 3.2 Checking for null values for all variable

```
#Dropping dimentionless diamonds
df= df.drop(df[df["'x'"]==0].index)
df = df.drop(df[df["'y'"]==0].index)
df = df.drop(df[df["'z'"]==0].index)
df.shape
```

```
(53920, 10)
```

Figure 3.3 Total number of row and column after deleting missing values

In some cases, missing data will affect the quality, validity and reliability of the data analysis and modeling. Identification of errors or inconsistencies found in the database is carried out in this step. The reason behind missing values can be caused by a variety of factors, human errors, wrong data entry or processing errors. Therefore, the first thing to do is to verify if there is any missing data in our diamond dataset.

Based on Figure 3.2, there are no missing values in our dataset but Figure 3.1 shows that the variables ‘x’, ‘y’, ‘z’ have ”0.000000” which could be due to some diamond being 2-dimensional or it could be faulty values in the dataset. Since there exists a logical explanation for this, it will be treated as a missing value. There are multiple ways to deal with these values, including deletion of data or imputation of data to reduce bias. However, caution was taken as dropping of large amounts of data could lead to misrepresentation of the population. Since our dataset is quite large with 50k rows, the idea of deleting the data was considered, provided that the amount of missing values is small enough. The reason for this is because dropping is a simpler, faster and cost effective method to deal with missing value compared to imputation which requires complex implementation.

Results from Figure 3.3 showed that the remaining rows after deleting the missing values were 53920, this indicates that only 20 rows were removed. In terms of percentage, only 3.70% is removed from the population. Hence, our team decided to delete data as the small amount of data would not have a significant impact on the representation of the population.

3.2. Remove Outliers

```
ax = sns.pairplot(df, hue="cut", palette="inferno")
```

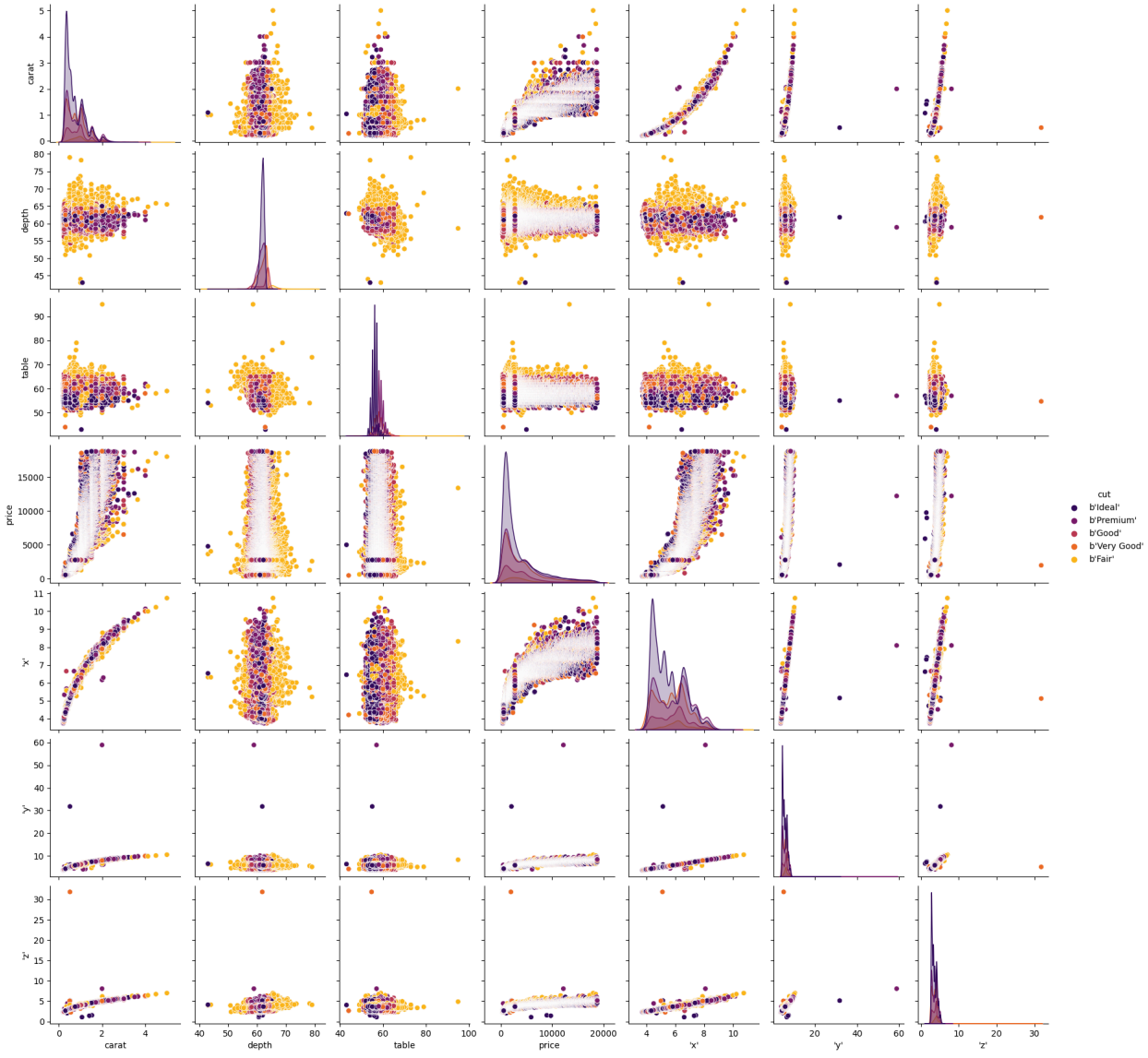


Figure 3.4 Pairplot for all variables

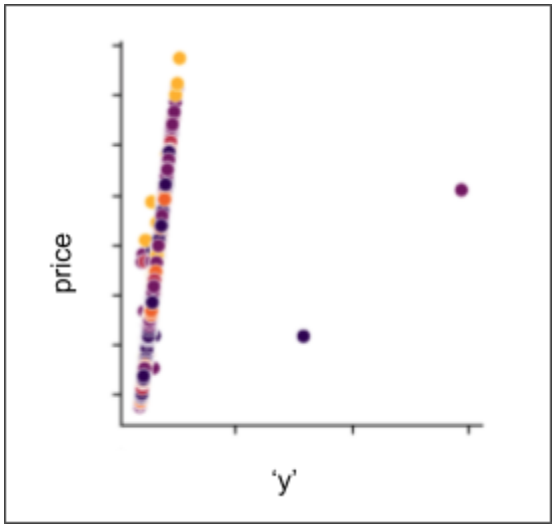


Figure 3.5 Pairplot for 'y' vs Price

Outliers are data points that are far away from the majority of the data points. It can cause the results to be skewed. This may lead to misinterpretation of patterns in plotted graphs and change the visual resolution of remaining data (Arimie, Biu and Ijomah, 2020). It also introduces bias to the machine learning models during the training process. To identify outliers, we plot the dataset using pairplot which is a module of seaborn library. The plots follow a matrix format where x-axis is represented by row name and y-axis is represented by column name.

For each attribute, univariate histograms are displayed in the main-diagonal subplots. Based on Figure 3.2.2, it is evident that some data points are far from the rest of the data points. Hence, we will plot a regression line on the scatterplots to better understand what are the changes in the independent variables as the price changes.

3.2.1 Regression Line on Price vs ‘x’

```
ax=sns.regplot(x="price", y="'x'", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'x'", color="#4e4c39")
```

Figure 3.6 Regression Line on Price vs ‘x’ (Price range: full)

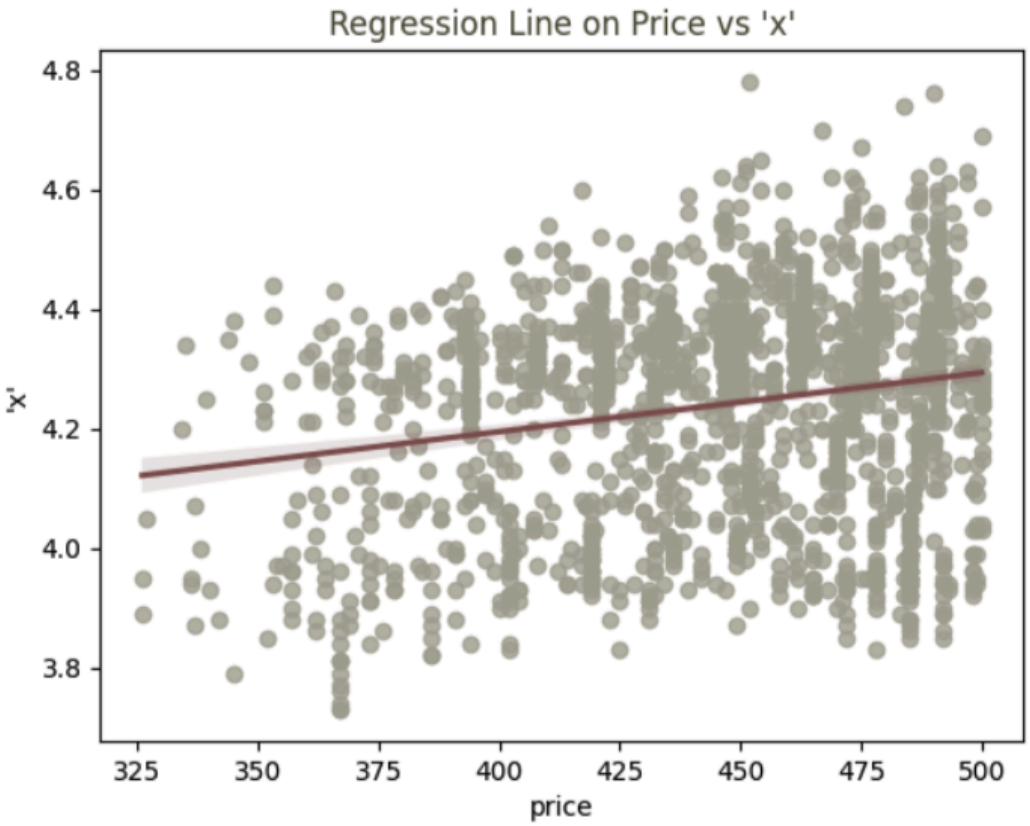


Figure 3.7 Regression Line on Price vs ‘x’ (Price range: below or equal to 500)

Based on Figure 3.6, there seems to be a moderate positive linear relationship between price and ‘x’ (length of diamond). The correlation of relationship is indicated by the distribution of the data points and steepness of the regression line. From there, it is evident that the data points are not too far or near each other indicating moderate strength. The relationship is classified as positive because both variables move in the same direction. For example, when price increases, ‘x’ also increases. The relationship is as such because ‘x’ (length in diamond) affects the overall size of a diamond and diamonds larger in size are typically considered to be rare, thus, the price for such diamonds are more expensive.

In comparison with Figure 3.7, it can be seen that the regression line in Figure 3.7 is less steep than the regression line in Figure 3.6. This shows that the relationship between price and ‘x’ for diamonds below 500 USD has a weaker positive relationship compared to diamonds of full price ranges. Furthermore, the data points in Figure 3.7 are more spread out than in Figure 3.6, which indicates more variability in the relationship between price and ‘x’. Finally, there are no outliers in this graph

3.2.2 Regression Line on Price vs ‘y’

```
ax = sns.regplot(x="price", y="'y'", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'y'")
```

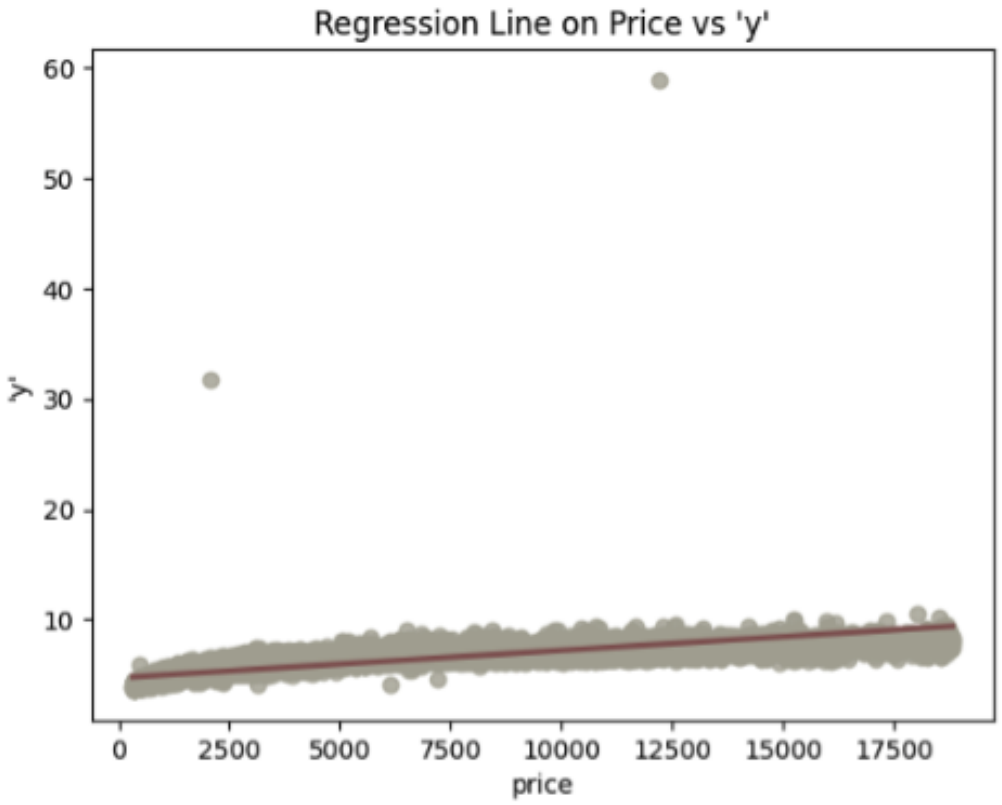


Figure 3.8 Regression Line on Price vs ‘y’ (Price range: full)

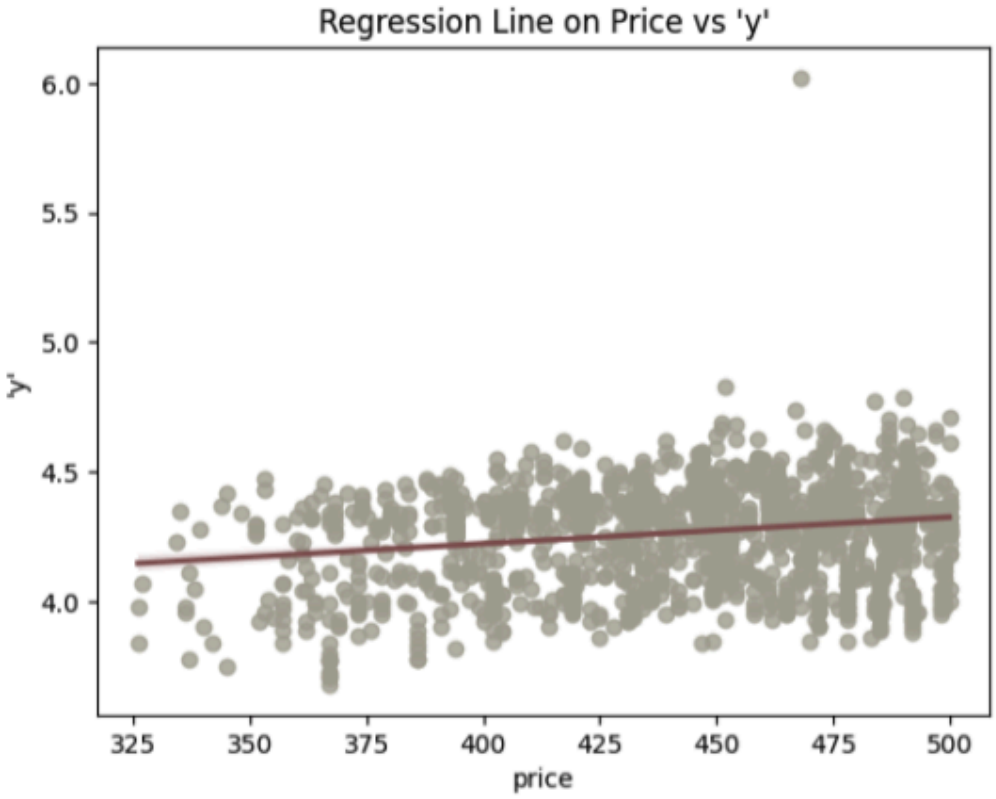


Figure 3.9 Regression Line on Price vs 'y' (Price range: <= 500)

According to Figure 3.8, it can be seen that the data points are close together and the regression line has a positive slope. Thus, there exists a very strong positive linear relationship between price and 'y' (width of diamond). This could be because 'y' (width in diamond) affects the overall size of a diamond and diamonds larger in size are typically considered to be rare, thus, the price for such diamonds are more expensive. This scatterplot shows that there are a few notable outliers when price is around 1250 to 2500, 5000 to 7500, and at 12500, the y value for these points are greater than 20. So, these outliers will be removed. The difference in size and shapes of diamonds may be the cause of this. Similarly, Figure 3.9 shows that distribution of data points is consistent and the line shows an upward trend.

3.2.3 Regression Line on Price vs ‘z’

```
ax= sns.regplot(x="price", y="'z'", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'z'")
```

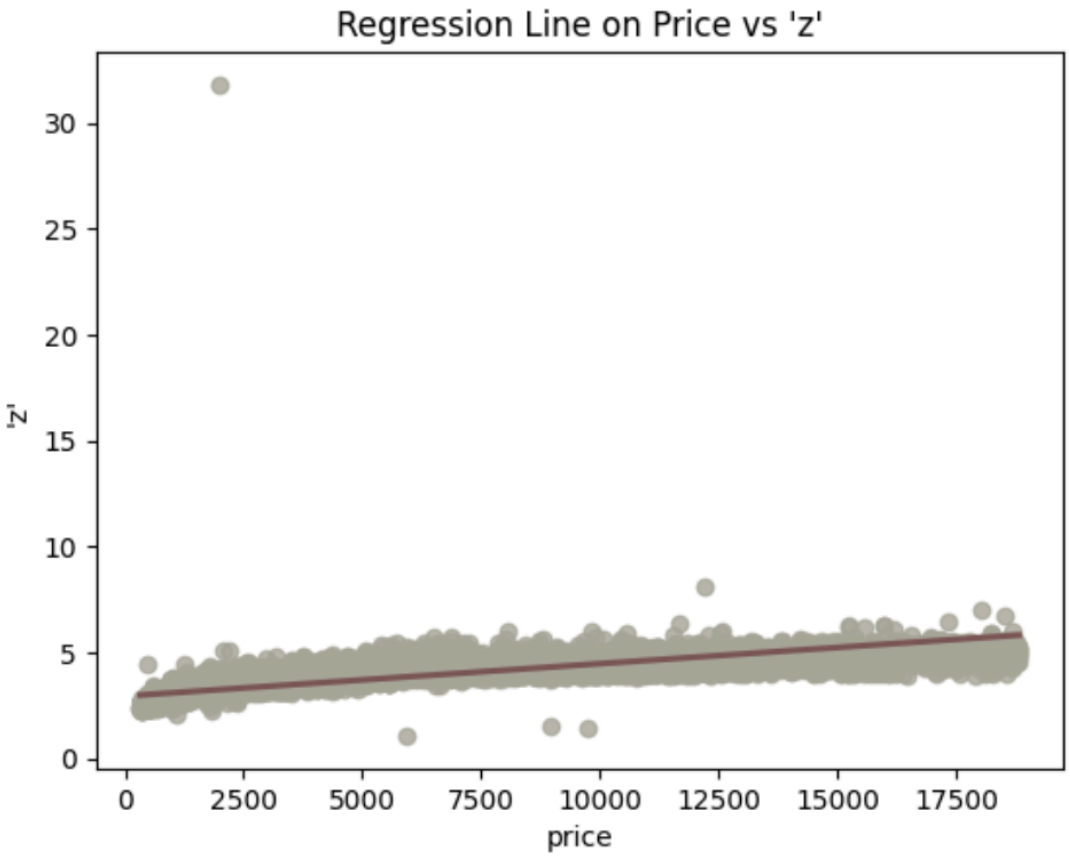


Figure 3.10 Regression Line on Price vs ‘z’ (Price range: full)

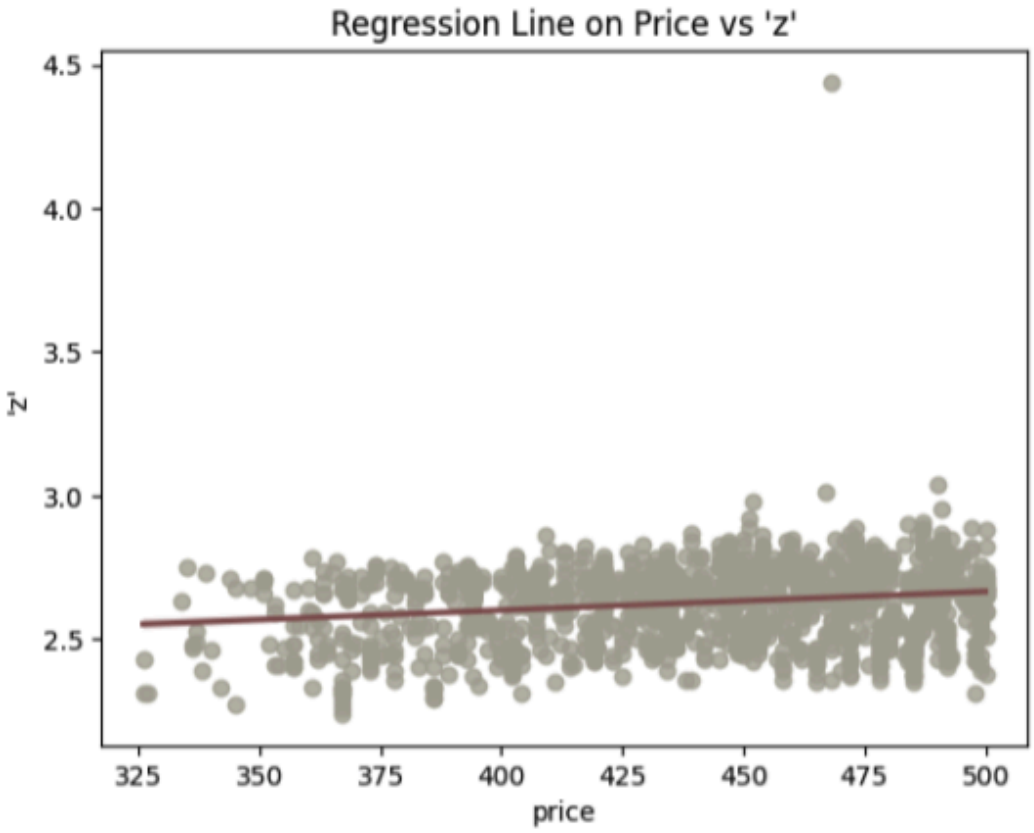


Figure 3.11 Regression Line on Price vs 'z' (Price range: <=500)

From Figure 3.10, it is evident that price and 'z' (width of diamond) have a very strong positive linear relationship. This is due to the closeness between data points and the positive trend in the regression line. Since the overall size of a diamond is affected by 'z' (width in diamond) and diamonds larger in size are typically considered to be rare, therefore, the price for such diamonds are more expensive. This scatterplot shows that there are a few notable outliers when price is around 1250 to 2500, 5000 to 7500, and at 12500 where z value is greater than 10 and less than 2. Thus, these outliers will be removed. As mentioned for 'y', the difference in size and shapes of diamonds may be the causing factors. For Figure 11, the same relationship can be used to describe even though the line has little upward trend.

3.2.4 Regression Line on Price vs Depth

```
ax= sns.regplot(x="price", y="depth", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Depth")
```

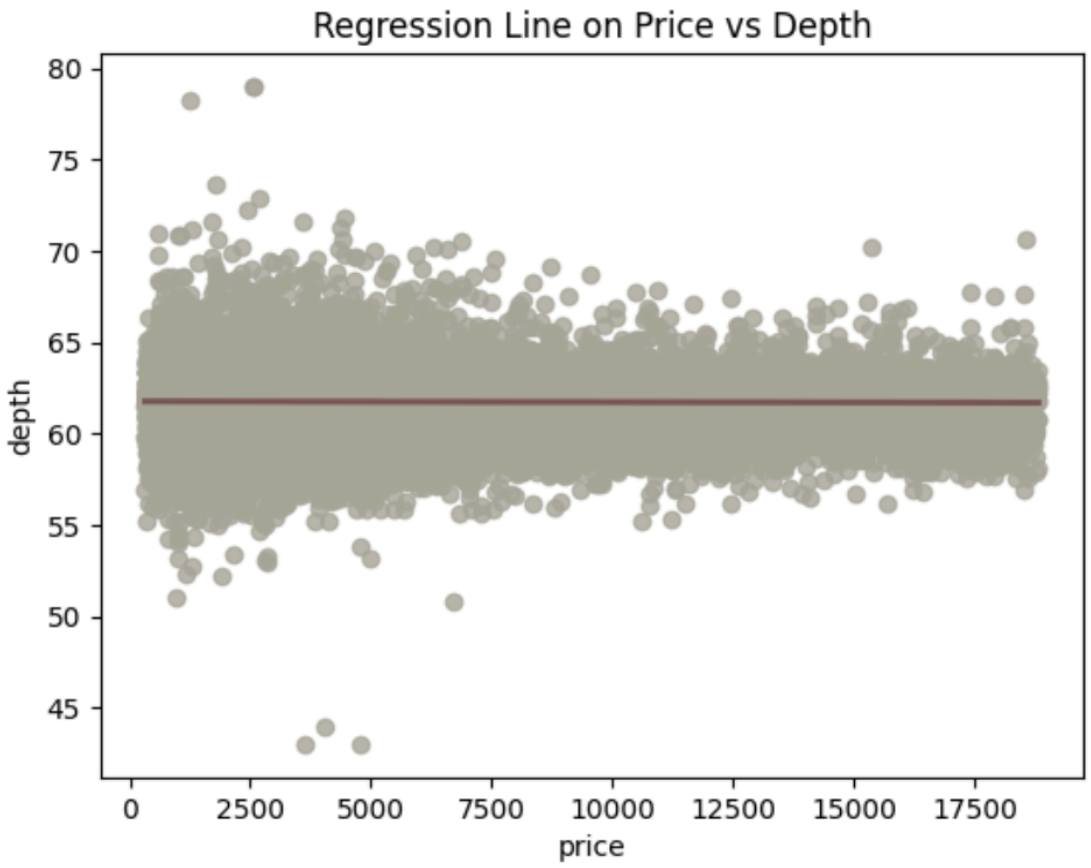


Figure 3.12 Regression Line on Price vs Depth (Price range: full)

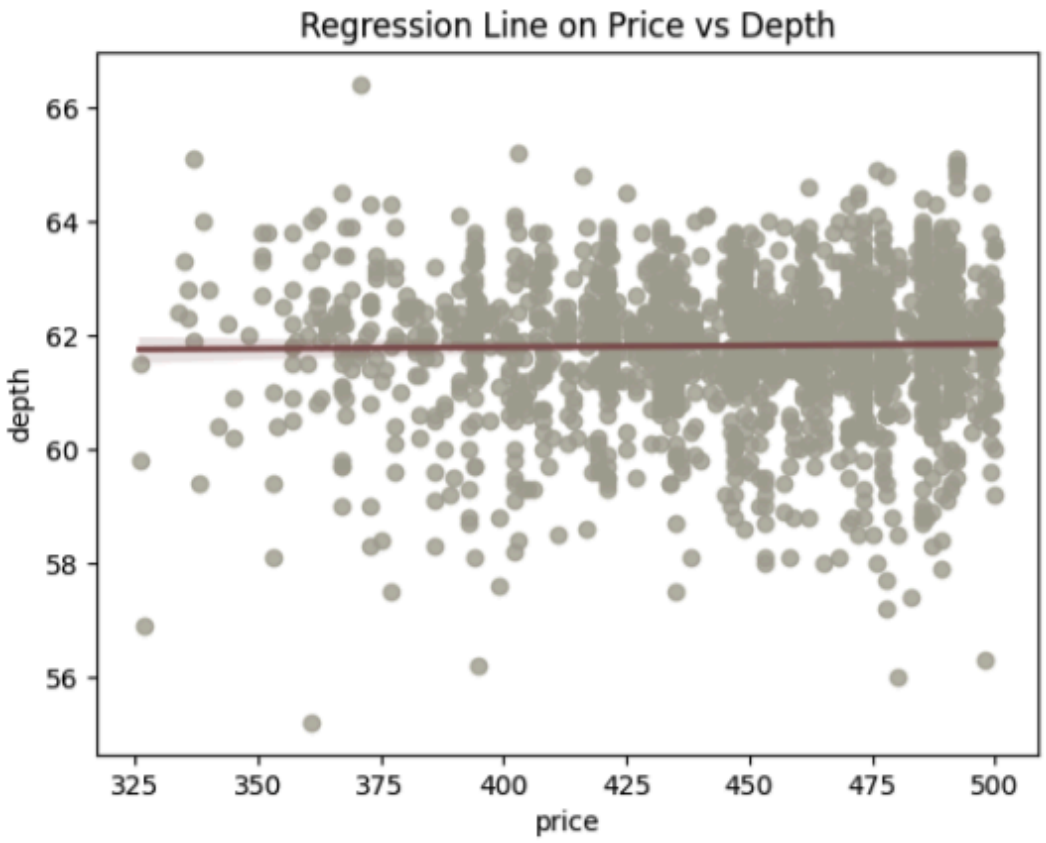


Figure 3.13 Regression Line on Price vs Depth (Price range: <= 500)

Based on Figure 3.12 and 3.13, there seems to be no relationship between price and depth which is consistent with correlation score in Figure 2.2. Depth has little effect on price because there are other factors such as the 4Cs which are more important metrics. There are prominent outliers for diamonds within the price range of 0 to 7500 and 15000 to 175000 where. Besides, the number of outliers are higher for the price range between 0 to 7500.

3.2.5 Regression Line on Price vs Table

```
ax=sns.regplot(x="price", y="table", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Table")
```

Figure 3.14 Regression Line on Price vs Table (Price range: full)

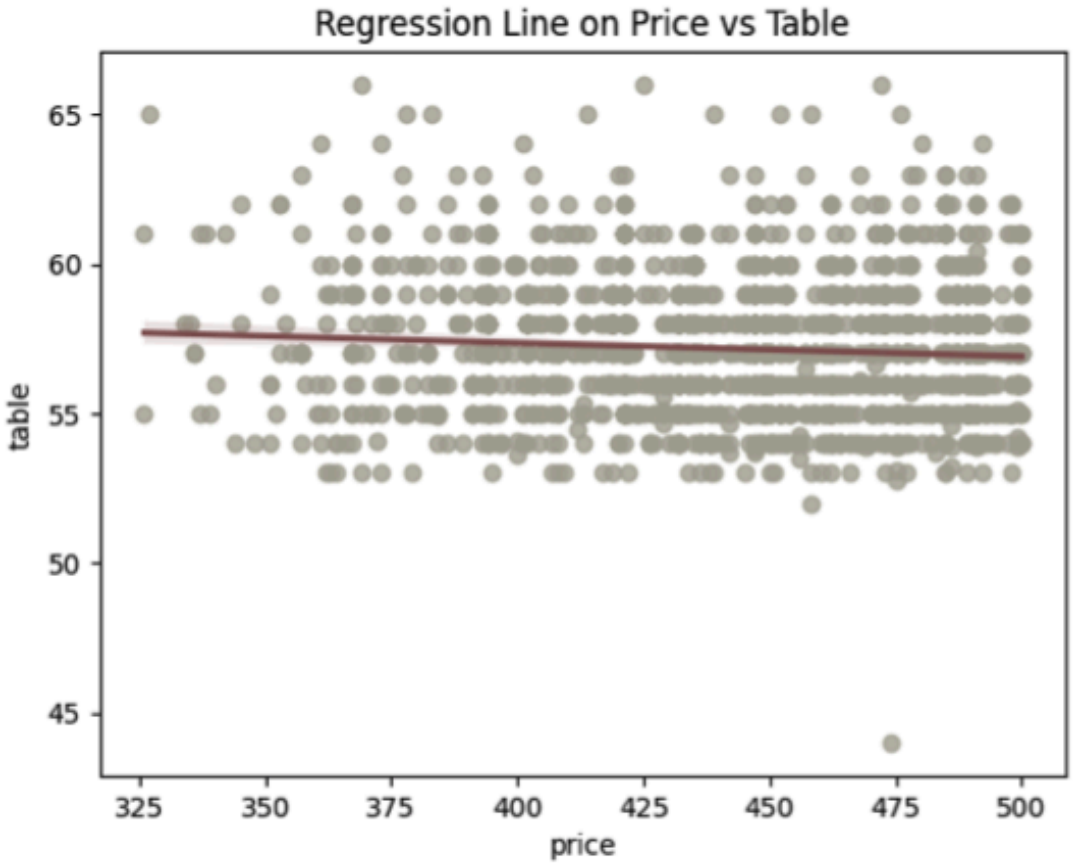


Figure 3.15 Regression Line on Price vs Table (Price range: <= 500)

According to Figure 3.14, there exists a strong positive linear relationship between price and table. This is because the data points are close together and the line is moving upwards. From the graph, diamonds with a price range around 2500 USD and 13500 to 14000 contain most outliers. The implication from this is most diamonds have the same table regardless of the price. However, there seems to be a moderate negative relationship between price and 'table' in Figure 3.15. This could be due to lower price diamonds having higher table percentage while higher price diamonds have lower table percentage. Nevertheless, more research is needed to discover the cause of this relationship.

3.2.6 Regression Line on Price vs Carat

```
ax=sns.regplot(x="price", y="carat", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Carat")
```

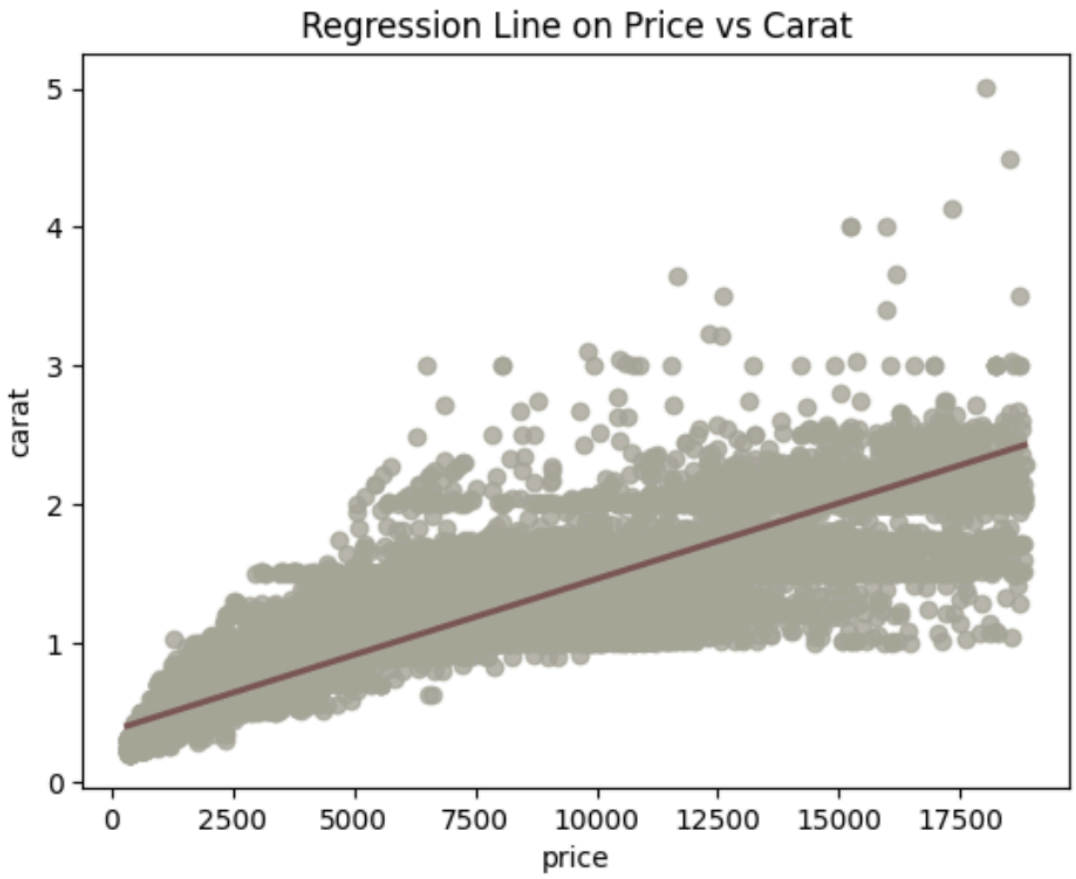


Figure 3.16 Regression Line on Price vs Carat (Price range: full)

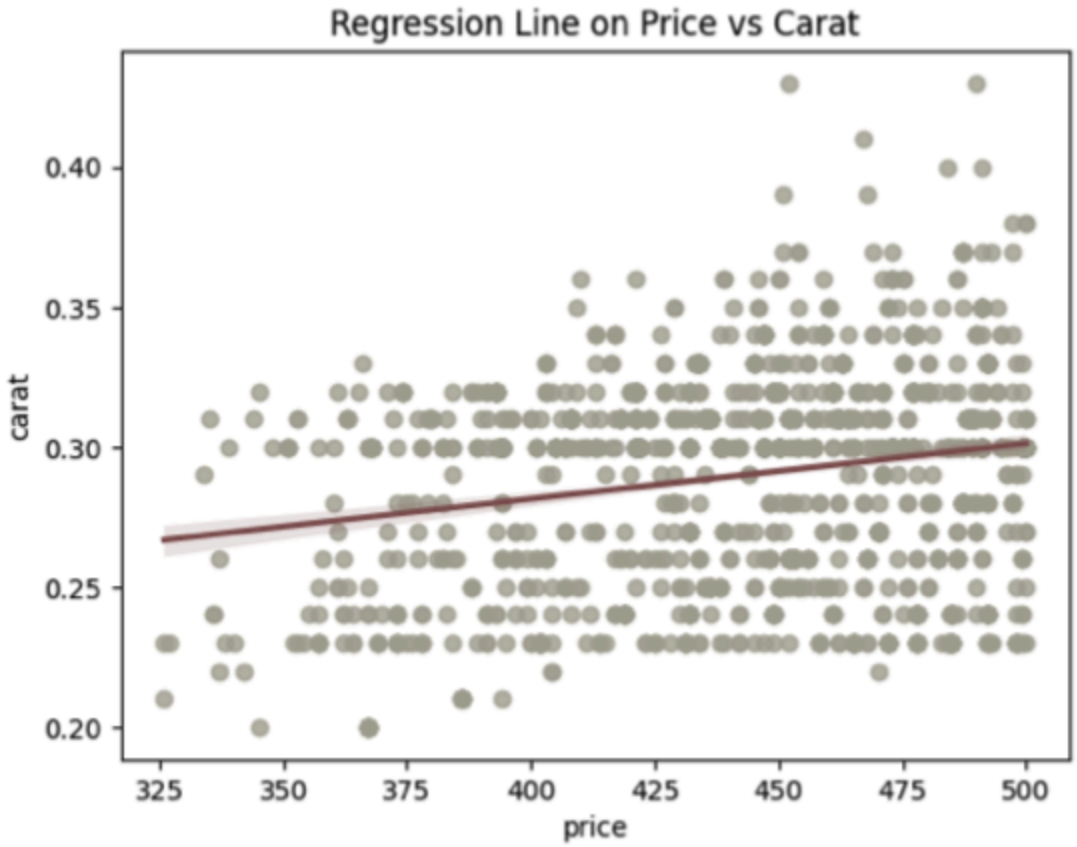


Figure 3.17 Regression Line on Price vs Carat (Price range: <= 500)

From Figure 3.16, it is evident that as the price of the diamond increases, the number of outliers for carats also increases. Thus, there exists a strong positive linear relationship between price and carat. The strength of the relationship is shown by the distribution of data points where data points which are close to each other indicate strong relationships.

3.2.7 Drop outlier

```
#dropping outlier
df = df[(df["'y'"]<20)]
df = df[(df["'z'"]<10)&(df["'z'"]>2)]
df = df[(df["depth"]<72)&(df["depth"]>52)]
df = df[(df["table"]<78)&(df["table"]>45)]
df = df[(df["carat"]<3)]
df.shape
```

Figure 3.18 Drop outliers

Based on the regression analysis, the value set for each attribute to remove the outliers were determined.

3.3. Selecting attributes (columns)

Based on Figure 2.2, a comparison between price and each of the variables including carat, cut, color, clarity, depth, ‘x’, ‘y’ and ‘z’. It is evident that carat, ‘x’, ‘y’ and ‘z’ have high correlation to price while the correlation for other variables are low. Despite the low correlation, these variables are still kept in the dataset as they believe they would have an impact on the prediction model. However, a testing is also done where attributes only with more than 0.5 are kept and used to train the Support Vector Machine (SVM).

3.4. Data Transformation

```
# Create a label encoder object
label_encoder = LabelEncoder()

# Fit the label encoder object to the category data
# Transform the category data using the transform() method of the label encoder object
label_encoder.fit(df['cut'])
df['cut'] = label_encoder.transform(df['cut'])

label_encoder.fit(df['color'])
df['color'] = label_encoder.transform(df['color'])

label_encoder.fit(df['clarity'])
df['clarity'] = label_encoder.transform(df['clarity'])

# Print the data
df
```

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
0	0.23	2	1	3	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	5	62.4	58.0	334.0	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335.0	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	2	0	2	60.8	57.0	2757.0	5.75	5.76	3.50
53936	0.72	1	0	2	63.1	55.0	2757.0	5.69	5.75	3.61
53937	0.70	4	0	2	62.8	60.0	2757.0	5.66	5.68	3.56
53938	0.86	3	4	3	61.0	58.0	2757.0	6.15	6.12	3.74
53939	0.75	2	0	3	62.2	55.0	2757.0	5.83	5.87	3.64

53859 rows × 10 columns

Figure 3.19 Encoded data

In order to conduct machine learning, the data has to be converted into a format suitable for analysis. This can be achieved by applying label encoder or one-hot encoders. Both encoders convert categorical data into numerical data. However, a label encoder would assign a unique numerical label for each category while a one-hot encoder creates a new binary vector for each category (Sethi, 2023). Since the categorical variables in our dataset have a small number of categories (max: 8) and the dataset is large (>53k row), Label Encoder was used since they are generally more simple and faster than one-hot encoders. From Figure 3.19, it can be seen that the categorical values for cut, color and clarity have been transformed into numerical values such as 1, 2, 4 and more.

3.5. Scaling Data

```
In [99]: from sklearn.preprocessing import MinMaxScaler
         df_scaled = df.copy()

         col = df.columns

         scaler = MinMaxScaler()
         df_scaled = scaler.fit_transform(df_scaled)
         df_scaled = pd.DataFrame(data=df_scaled,columns=[col])
```

```
In [100]: print(df_scaled)
```

```
          carat   cut     color   clarity     depth     table     price  \
0      0.011538  0.50  0.166667  0.428571  0.474490  0.222222  0.000000
1      0.003846  0.75  0.166667  0.285714  0.387755  0.444444  0.000000
2      0.011538  0.25  0.166667  0.571429  0.239796  0.592593  0.000054
3      0.034615  0.75  0.833333  0.714286  0.520408  0.333333  0.000433
4      0.042308  0.25  1.000000  0.428571  0.566327  0.333333  0.000487
...         ...   ...       ...       ...       ...       ...       ...
53854  0.200000  0.50  0.000000  0.285714  0.438776  0.296296  0.131427
53855  0.200000  0.25  0.000000  0.285714  0.556122  0.222222  0.131427
53856  0.192308  1.00  0.000000  0.285714  0.540816  0.407407  0.131427
53857  0.253846  0.75  0.666667  0.428571  0.448980  0.333333  0.131427
53858  0.211538  0.50  0.000000  0.428571  0.510204  0.222222  0.131427

            'x'       'y'       'z'
0      0.040441  0.055046  0.090244
1      0.029412  0.029358  0.060976
2      0.058824  0.071560  0.060976
3      0.086397  0.100917  0.139024
4      0.112132  0.122936  0.168293
...         ...       ...       ...
53854  0.371324  0.381651  0.351220
53855  0.360294  0.379817  0.378049
53856  0.354779  0.366972  0.365854
53857  0.444853  0.447706  0.409756
53858  0.386029  0.401835  0.385366

[53859 rows x 10 columns]
```

Figure 3.20 Scale data

Based on Figure 3.20 , it can be seen that the diamond data set has a wide range of values. This may cause some features to contribute to the model more than other features such as features with larger values dominating the data. To produce an unbiased model performance, there are a few methods that can be used to scale the features, including normalization, min-max scaling and standardization (Bhandari, 2020). Since the distribution of data is not guaranteed to be normal after removing the outliers, Min-Max scaling was used to reduce the impact of outliers by scaling down the outliers to be closer to other data points. After applying the min-max scaling to the diamond dataset, the range of values for all variables is scaled to 0 to 1.

3.6. Split Data Into Training Set and Testing Set

```
#split data into training and testing
x = data.drop('price', axis = 1)
y = data['price']


from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size = 0.30, random_state = 1)
```

Figure 3.21 Split train and test data

We will create a function that separates the training set and testing set into X and Y respectively. For testing, the complete dataset was used except for the target attribute 'price'. The features of the dataset include carat, cut, color, clarity, depth, table, X, Y, Z was assigned to the variable 'x' and the target variable was assigned to variable 'y'. After reviewing multiple journals and articles from the web, our team decided to split 70% of the dataset for training and 30% for testing (Afshin Gholamy, Vladik Kreinovich , and Olga Kosheleva, 2018). The reason for this is because the 70:30 ratio provides a good balance that reduces chances of overfitting and improves generalization of the model. Another study was done on prediction of diamond price, however the training to test the dataset ratio was 80:20 (Chaijunla & Taninpong, n.d.).

4.0 MODELING

There are 3 types of modeling techniques used which includes Support Vector Machine (SVM), Random Forest Regression and Decision Tree Regression.
The table below is the modeling techniques that we used and the result.

No.	Model	Name
1.	Support Vector Machine (SVM)	**TAI QI ZHENG**
2.	Random Forest Regression	**LAW QIAN WEN**
3.	Decision Tree Regression	**WONG WEI JIE**

Table 1

4.1. Results

Result Table

		Result					
No.	**Model**	**R^2**	**Adj R^2**	**MAE**	**MSE**	**RMSE**	**EVS**
1.	Support Vector Machine 1	**0.927310**	**0.92727**	**0.043134**	**0.003252**	**0.06**	**0.93**
2.	Support Vector Machine 2	**0.854041**	**0.85400**	**0.057637**	**0.006531**	**0.08**	**0.86**
3.	Random Forest Regression	**0.979517**	**0.979505**	**0.014778**	**0.000916**	**0.03**	**0.98**
4.	Decision Tree Regression	**0.961705**	**0.961684**	**0.020004**	**0.001713**	**0.04**	**0.96**

Table 2 Performance result for all algorithms

Based on the Result Table it briefly shows the result of R^2, Adj R^2, MAE, MSE, RMSE and EVS for every model.

4.1.1 Support Vector Machine (SVM)

```
: LL_svr = SVR().fit(X_train, y_train)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explained_variance_score

predSVM = LL_svr.predict(X_test)

print('The R^2 is: %f'% r2_score(y_test, predSVM))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test, predSVM))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test, predSVM))
print('The MSE is: %f'% mean_squared_error(y_test, predSVM))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test, predSVM)))
print('The EVS is: %.2f'% explained_variance_score(y_test, predSVM))
```

```
The R^2 is: 0.927310
The Adjusted R^2 is: 0.9272696361944092
The MAE is: 0.043134
The MSE is: 0.003252
The RMSE is: 0.06
The EVS is: 0.93
```

Figure 4.1 SVM model's code and result

```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test)
TestingDataResults['TargetColumn']=y_test
TestingDataResults['Prediction']=predSVM
TestingDataResults
```

	carat	cut	color	clarity	depth	table	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.75	0.500000	0.857143	0.515306	0.370370	0.040441	0.044037	0.095122	554.0	1038.238678
47338	0.161538	0.25	0.500000	0.714286	0.448980	0.481481	0.308824	0.333945	0.309756	1850.0	3277.024085
42573	0.115385	0.00	0.666667	0.714286	0.209184	0.592593	0.277574	0.280734	0.214634	1332.0	2158.496273
32163	0.042308	0.50	0.500000	0.857143	0.505102	0.222222	0.110294	0.124771	0.156098	789.0	654.471480
40040	0.046154	0.50	0.000000	0.571429	0.443878	0.296296	0.137868	0.141284	0.160976	1114.0	-10.724709
...	...	...	...	...	...	...	...	...	...	...	...
29498	0.080769	0.25	0.333333	0.285714	0.566327	0.333333	0.180147	0.194495	0.226829	705.0	973.947273
11697	0.053846	0.50	0.500000	0.714286	0.474490	0.296296	0.136029	0.152294	0.170732	596.0	497.070099
19191	0.534615	0.75	1.000000	0.714286	0.530612	0.370370	0.683824	0.688073	0.634146	7953.0	9117.996103
4195	0.311538	1.00	0.833333	0.285714	0.556122	0.333333	0.487132	0.486239	0.475610	3563.0	4403.891031
53079	0.150000	0.50	0.333333	0.857143	0.520408	0.259259	0.299632	0.311927	0.314634	2625.0	2626.989792

16158 rows × 11 columns

Figure 4.2 Table of original value VS prediction value

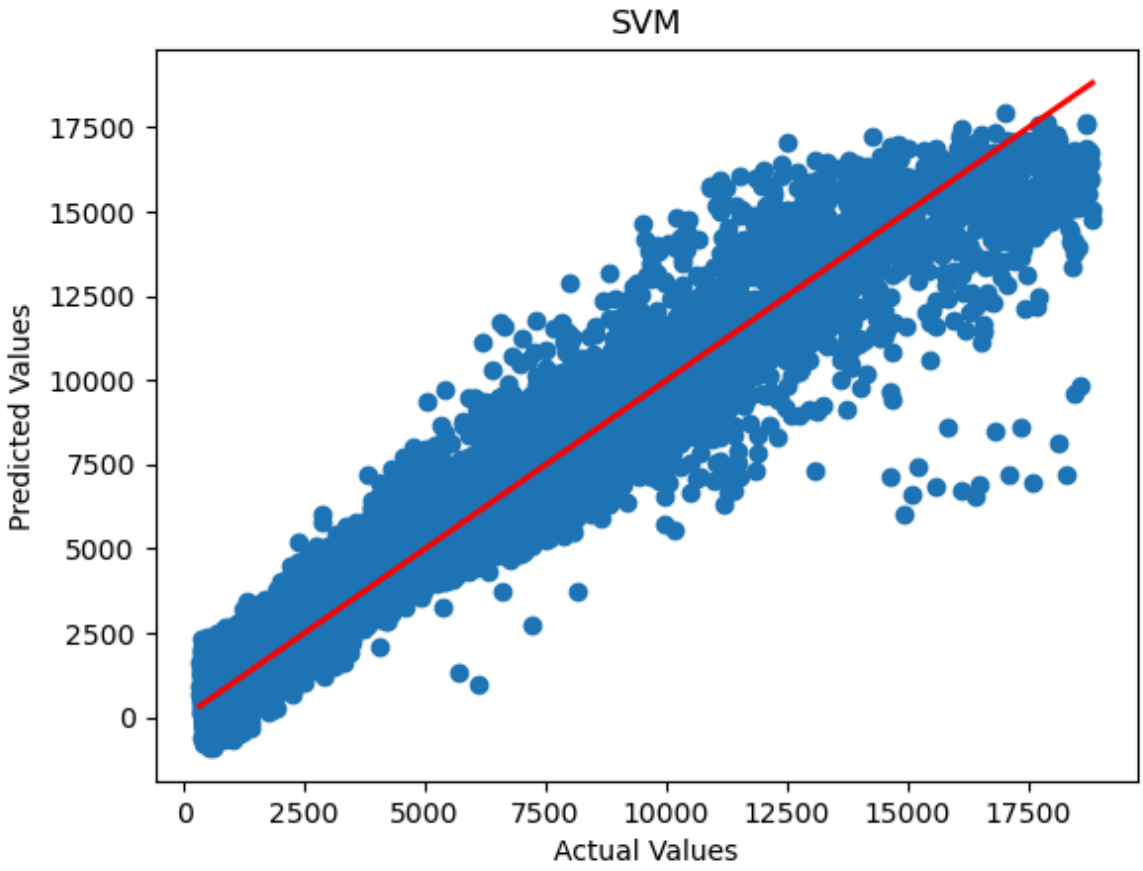


Figure 4.3 Scatter plot of the predicted and actual values

Based on the figure 4.3, it is showing a scatter plot of the predicted values VS actual values by using the Support Vector Machine (SVM). The x-axis is the actual values and the y-axis is showing the predicted values. The red line represents the perfect prediction, where the predicted value matches the actual value. The plots that are generated in the scatter plot are close to the line of prefect prediction, which showed that the SVM model is working well in predicting the value. However, some of the plots are far away from the line, which shows that the model will not be predicted perfectly. Other than that, the predicted value is closer to the line of perfect prediction compared with the actual values.

4.1.2 Support Vector Machine (SVM) with the variable that have a high correlation

Support Vector Machine (SVM) With all variable that correlation is more than 0.5

```
]: LL_svr = SVR().fit(X_train_a, y_train_a)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explained_variance_score

predSVM_a = LL_svr.predict(X_test_a)

print('The R^2 is: %f'% r2_score(y_test_a, predSVM_a))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test_a, predSVM_a))*(len(y_test_a)-1)/(len(y_test_a)-X_test_a.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test_a, predSVM_a))
print('The MSE is: %f'% mean_squared_error(y_test_a, predSVM_a))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test_a, predSVM_a)))
print('The EVS is: %.2f'% explained_variance_score(y_test_a, predSVM_a))
```

```
The R^2 is: 0.854041
The Adjusted R^2 is: 0.854004930090795
The MAE is: 0.057637
The MSE is: 0.006531
The RMSE is: 0.08
The EVS is: 0.86
```

Figure 4.4 Code and result of SVM with the variable that correlation is higher than 0.5

```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test_a)
TestingDataResults['TargetColumn']=y_test_a
TestingDataResults['Prediction']=predSVM_a
TestingDataResults
```

	carat	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.040441	0.044037	0.095122	554.0	-413.330587
47338	0.161538	0.308824	0.333945	0.309756	1850.0	2823.277549
42573	0.115385	0.277574	0.280734	0.214634	1332.0	2594.717789
32163	0.042308	0.110294	0.124771	0.156098	789.0	649.409633
40040	0.046154	0.137868	0.141284	0.160976	1114.0	1015.610118
...	...	...	...	...	...	...
29498	0.080769	0.180147	0.194495	0.226829	705.0	2113.065812
11697	0.053846	0.136029	0.152294	0.170732	596.0	1169.684400
19191	0.534615	0.683824	0.688073	0.634146	7953.0	11013.244519
4195	0.311538	0.487132	0.486239	0.475610	3563.0	5233.884892
53079	0.150000	0.299632	0.311927	0.314634	2625.0	2665.351994

16158 rows × 6 columns

Figure 4.5 Table of original value VS prediction value

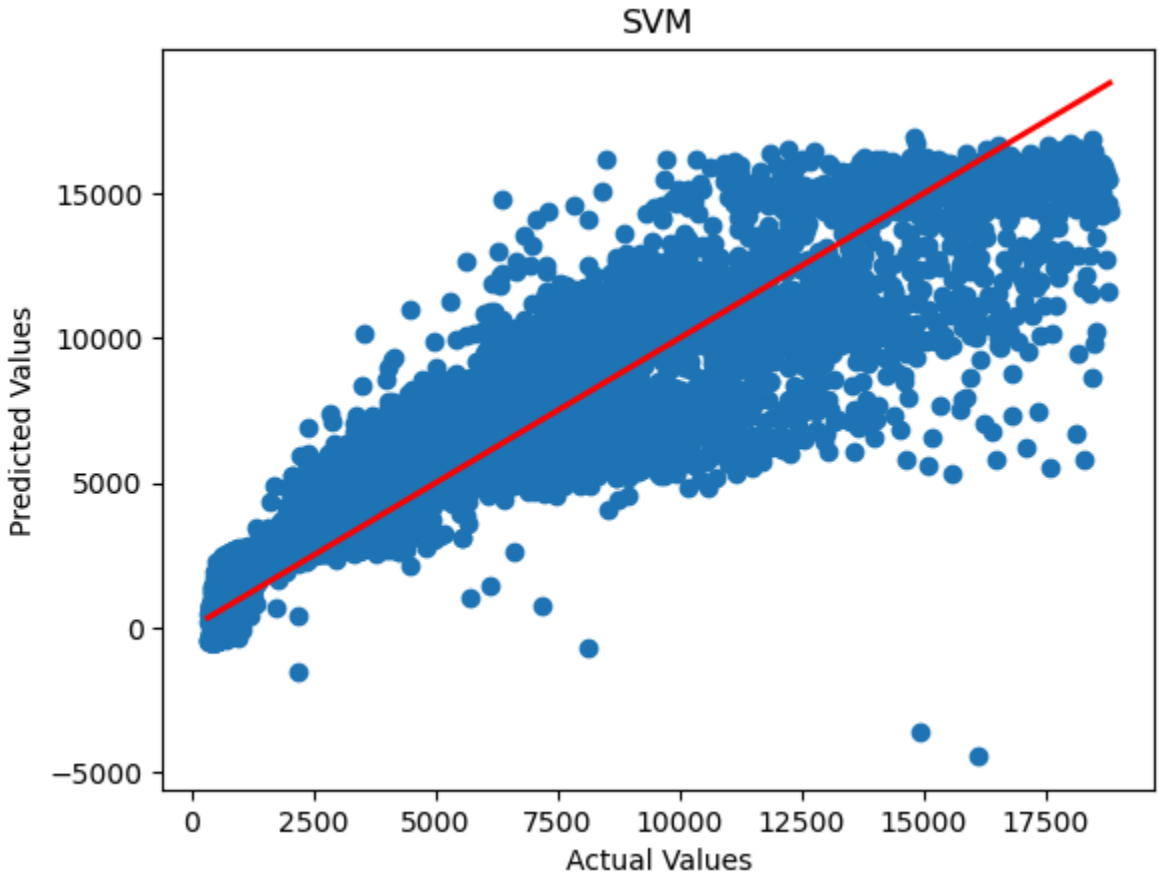


Figure 4.6 Scatter plot of the predicted and actual values

Based on the figure 4.6, it is showing a scatter plot of the predicted values VS actual values by using the Support Vector Machine (SVM) and the variable is only considered high correlation. The x-axis is the actual values and the y-axis is showing the predicted values. The red line represents the perfect prediction, where the predicted value matches the actual value.

4.1.2 Random Forest Regression

Random Forest Regression

```
# Create a random forest regressor model
rfr = RandomForestRegressor()

# Train the model on the training data
rfr.fit(X_train, y_train)

# Make predictions on the test data
rfr_predictions = rfr.predict(X_test)

print('The R^2 is: %f'% r2_score(y_test, rfr_predictions))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test, rfr_predictions))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test, rfr_predictions))
print('The MSE is: %f'% mean_squared_error(y_test, rfr_predictions))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test, rfr_predictions)))
print('The EVS is: %.2f'% explained_variance_score(y_test, rfr_predictions))
```

```
The R^2 is: 0.979595
The Adjusted R^2 is: 0.9795835497107316
The MAE is: 0.014736
The MSE is: 0.000913
The RMSE is: 0.03
The EVS is: 0.98
```

Figure 4.7 Random Forest Regression model's code and result

```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test)
TestingDataResults['TargetColumn']=y_test
TestingDataResults['Prediction']=rfr_predictions
TestingDataResults
```

	carat	cut	color	clarity	depth	table	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.75	0.500000	0.857143	0.515306	0.370370	0.040441	0.044037	0.095122	554.0	524.78
47338	0.161538	0.25	0.500000	0.714286	0.448980	0.481481	0.308824	0.333945	0.309756	1850.0	1918.79
42573	0.115385	0.00	0.666667	0.714286	0.209184	0.592593	0.277574	0.280734	0.214634	1332.0	1275.75
32163	0.042308	0.50	0.500000	0.857143	0.505102	0.222222	0.110294	0.124771	0.156098	789.0	773.30
40040	0.046154	0.50	0.000000	0.571429	0.443878	0.296296	0.137868	0.141284	0.160976	1114.0	981.45
...	...	...	...	...	...	...	...	...	...	...	...
29498	0.080769	0.25	0.333333	0.285714	0.566327	0.333333	0.180147	0.194495	0.226829	705.0	720.37
11697	0.053846	0.50	0.500000	0.714286	0.474490	0.296296	0.136029	0.152294	0.170732	596.0	596.00
19191	0.534615	0.75	1.000000	0.714286	0.530612	0.370370	0.683824	0.688073	0.634146	7953.0	8099.13
4195	0.311538	1.00	0.833333	0.285714	0.556122	0.333333	0.487132	0.486239	0.475610	3563.0	3936.57
53079	0.150000	0.50	0.333333	0.857143	0.520408	0.259259	0.299632	0.311927	0.314634	2625.0	2980.00

16158 rows × 11 columns

Figure 4.8 Table of original value VS prediction value

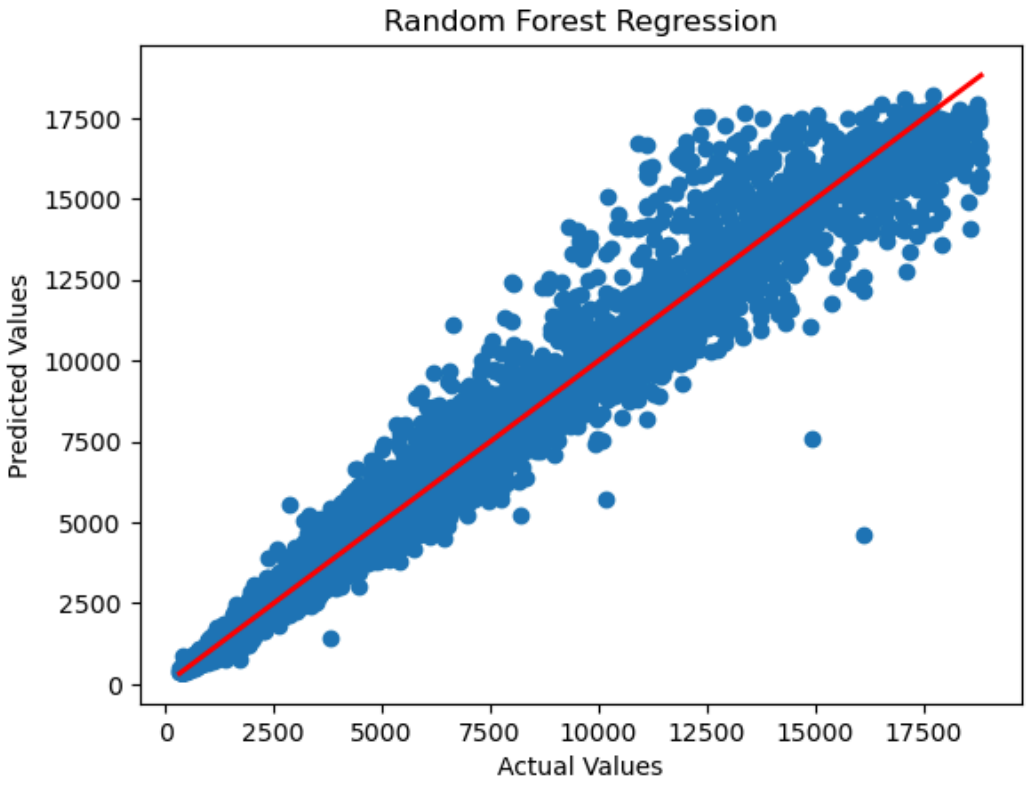


Figure 4.9 Scatter plot of the predicted and actual values

Based on the figure 4.9, it is showing a scatter plot of the predicted values VS actual values by using the Random Forest Regression. The x-axis is the actual values and the y-axis is showing the predicted values. The red line represents the perfect prediction, where the predicted value matches the actual value. The plots in the scatter plot are very close to the line of prefect prediction, which is showing that it is doing an excellent job for predicting the values. Even though some of the plot is far away from the line of prefect prediction, it still can do a prefect prediction. So we can see that the scatter plot that is showing is saying that the Random Forest Regression model is good and it can make a good prediction.

4.1.3 Decision Tree Regression

Decision Tree Regression

```
]: # Create a decision tree regressor model
dtf = DecisionTreeRegressor()

# Train the model on the training data
dtf.fit(X_train, y_train)

# Make predictions on the test data
dtf_predictions = dtf.predict(X_test)

print('The R^2 is: %f'% r2_score(y_test, dtf_predictions))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test, dtf_predictions))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test, dtf_predictions))
print('The MSE is: %f'% mean_squared_error(y_test, dtf_predictions))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test, dtf_predictions)))
print('The EVS is: %.2f'% explained_variance_score(y_test, dtf_predictions))
```

```
The R^2 is: 0.961861
The Adjusted R^2 is: 0.9618392842860237
The MAE is: 0.020130
The MSE is: 0.001707
The RMSE is: 0.04
The EVS is: 0.96
```

Figure 4.10

```
: LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test)
TestingDataResults['TargetColumn']=y_test
TestingDataResults['Prediction']=dtf_predictions
TestingDataResults
```

:

	carat	cut	color	clarity	depth	table	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.75	0.500000	0.857143	0.515306	0.370370	0.040441	0.044037	0.095122	554.0	680.0
47338	0.161538	0.25	0.500000	0.714286	0.448980	0.481481	0.308824	0.333945	0.309756	1850.0	1821.0
42573	0.115385	0.00	0.666667	0.714286	0.209184	0.592593	0.277574	0.280734	0.214634	1332.0	1239.0
32163	0.042308	0.50	0.500000	0.857143	0.505102	0.222222	0.110294	0.124771	0.156098	789.0	789.0
40040	0.046154	0.50	0.000000	0.571429	0.443878	0.296296	0.137868	0.141284	0.160976	1114.0	972.0
...	...	...	...	...	...	...	...	...	...	...	...
29498	0.080769	0.25	0.333333	0.285714	0.566327	0.333333	0.180147	0.194495	0.226829	705.0	719.0
11697	0.053846	0.50	0.500000	0.714286	0.474490	0.296296	0.136029	0.152294	0.170732	596.0	596.0
19191	0.534615	0.75	1.000000	0.714286	0.530612	0.370370	0.683824	0.688073	0.634146	7953.0	7811.0
4195	0.311538	1.00	0.833333	0.285714	0.556122	0.333333	0.487132	0.486239	0.475610	3563.0	4242.0
53079	0.150000	0.50	0.333333	0.857143	0.520408	0.259259	0.299632	0.311927	0.314634	2625.0	3090.0

16158 rows × 11 columns

Figure 4.11 Table of original value VS prediction value

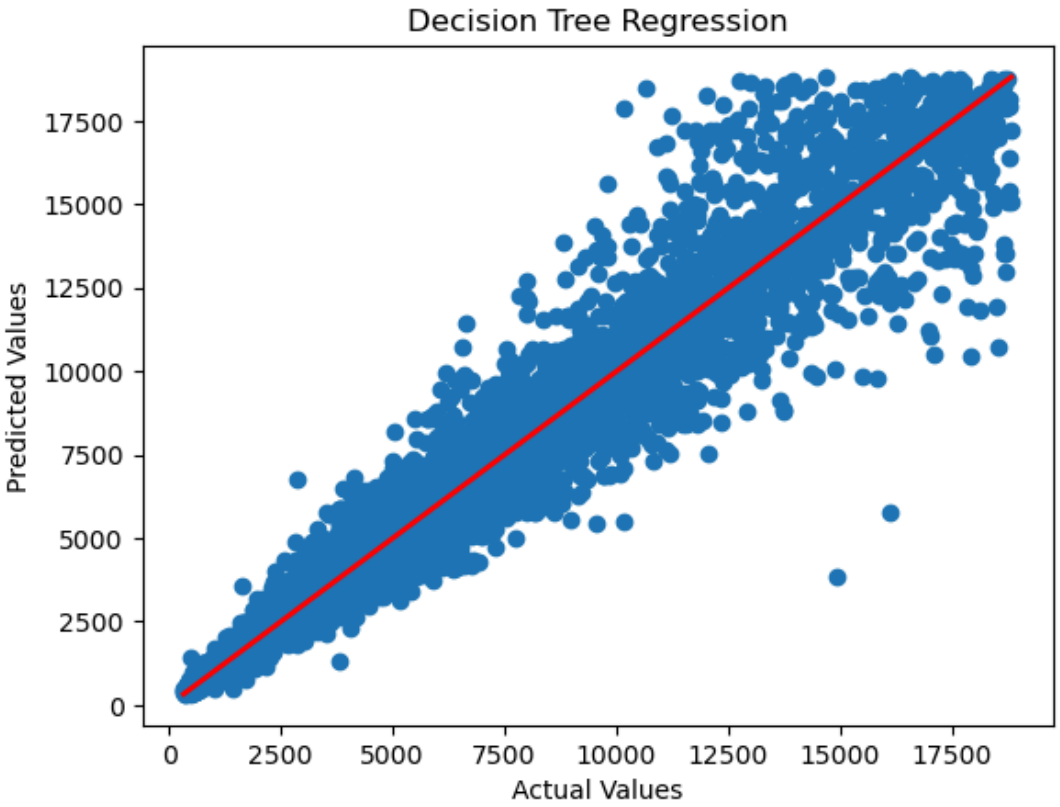


Figure 4.12 Scatter plot of the predicted and actual values

Based on the figure 4.12, it is showing a scatter plot of the predicted values VS actual values by using the Decision Tree Regression. The x-axis is the actual values and the y-axis is showing the predicted values. The red line represents the perfect prediction, where the predicted value matches the actual value. The plots in the scatter plot are very close to the line of prefect prediction, which is showing that it is doing an excellent job for predicting the values. Even though some of the plot is far away from the line of prefect prediction, it still can do a prefect prediction. Other than that we also can see that the scatter plot that is showing is saying that the Random Forest Regression model is almost similar to the Random Forest Regression.

4.2. Discussion/Interpretation

4.2.1. R^2

R^2 is to measure how well a regression model fits a set of data. The value range for R^2 is from 0 to 1. Higher numbers indicate a better fit of the model to the data and lower numbers indicate that the model does not explain any of the variation in the data.

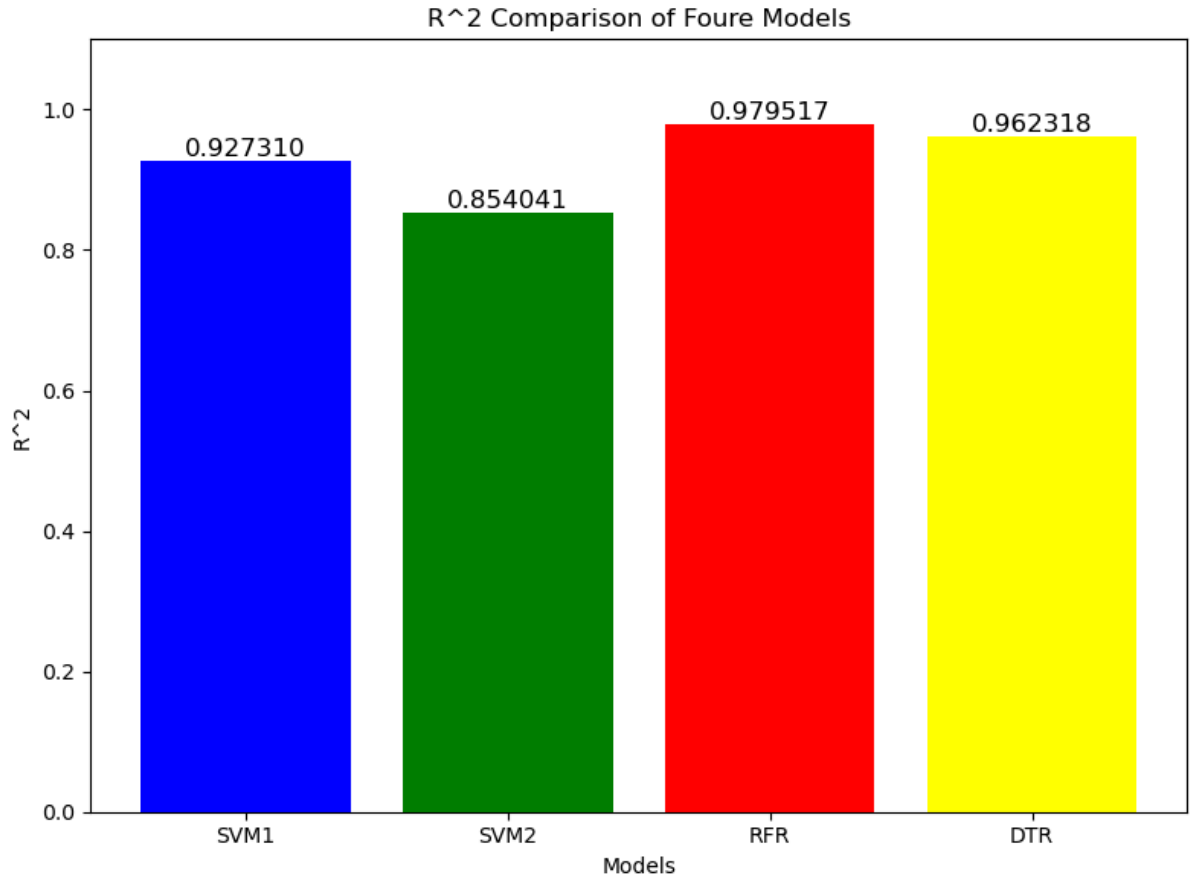


Figure 4.13 R^2 Comparison of Four model

Based on the figure 4.13, the Random Forest Regression(RFR) has the highest score 0.979517. Followed by Decision Tree Regression(DTR), Support Vector Machine1 (SVM1) and Support Vector Machine2 (SVM2) which have lesser variables than SVM1. This figure shows that the RFR are able to predict diamond price more accurately among those 4 models.

4.2.2. Adjusted R^2

Adjusted R^2 is a modified version of R^2 that is adjusted for the number of predictors in the model. R^2 can be increased when the predictors are added even if the predictors do not improve the model ability to predict the outcome variable. To make it simple, R^2 will be affected if the model added a new variable but it wouldn't necessarily affect the model. While Adjusted R^2, will penalize the model for having too many variables. The value range for Adjusted R^2 is the same as R^2.

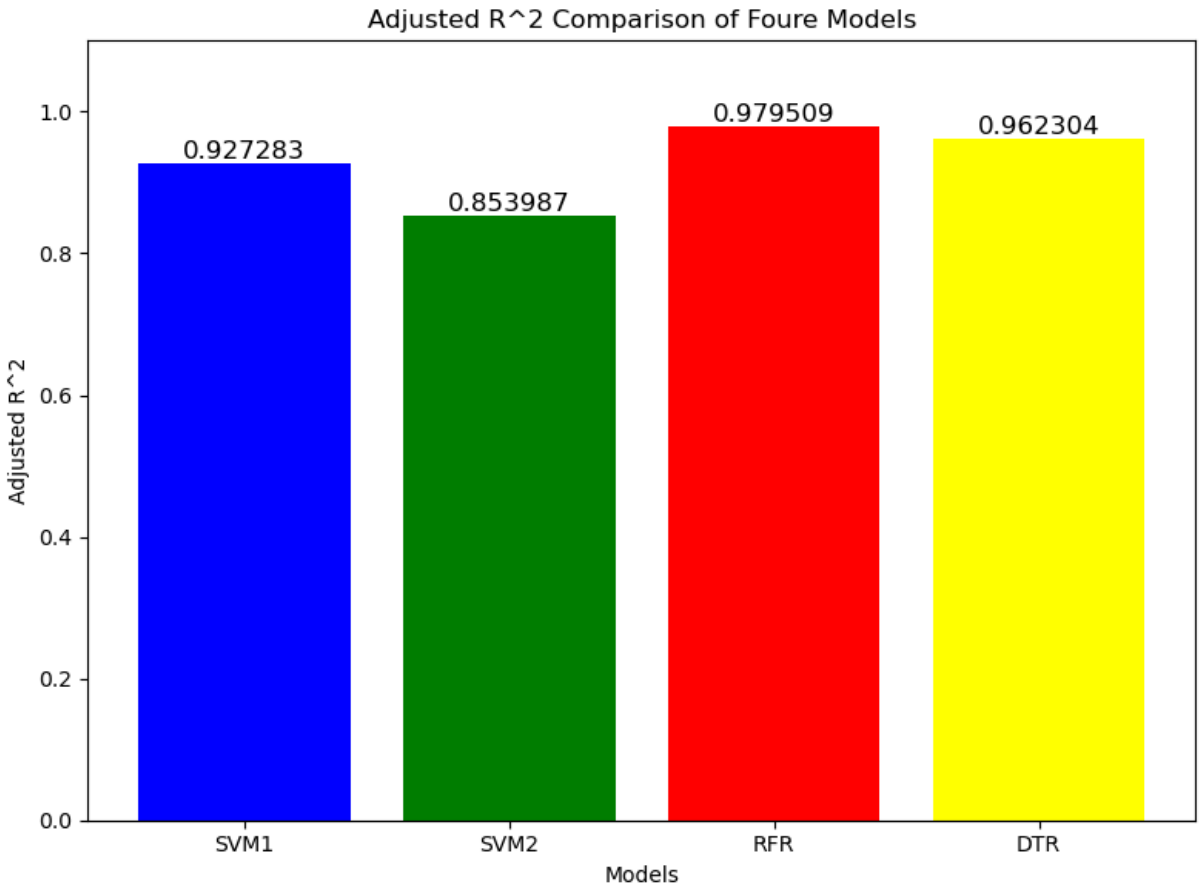


Figure 4.14 Adjusted R^2 Comparison of Four model

The Figure 4.14 in above shows that Random Forest Regression has the highest Adjusted R^2 which is 0.979509. This is showing that the Random Forest Regression model is the best to fit the data and the next is only Decision Tree Regression. One possible explanation for random forest are ensemble models, meaning that the model combines the predictions of multiple decision trees. It makes this model more robust to overfitting and be able to learn more complex relationships for the data.

4.2.3. Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is to measure the average magnitude of the errors in a set of predictions, without considering their direction. It is calculated by taking the average of the absolute differences between the predicted values and the actual values. In short the MAE value will be the average difference between the predicted values and the actual value.

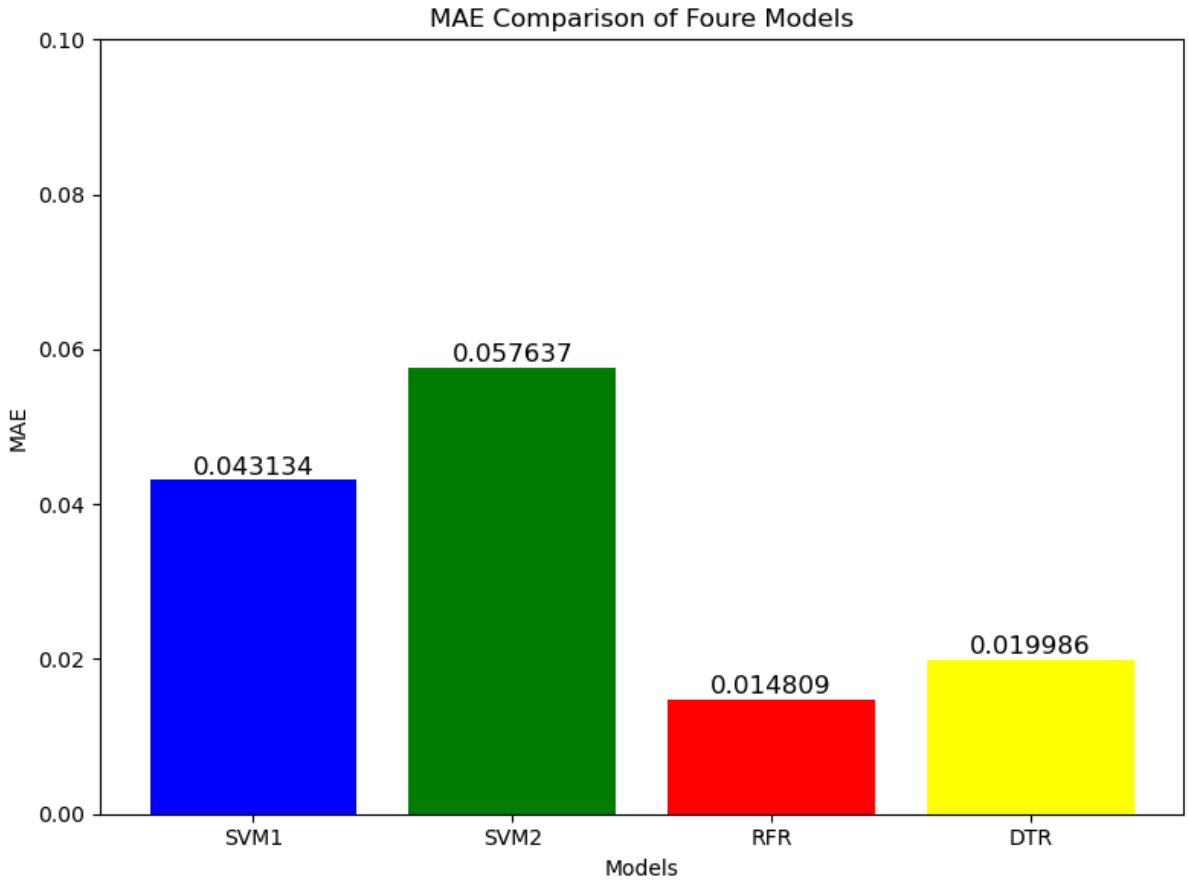


Figure 4.15 MAE Comparison of Four model

Mean Absolute Error (MAE) is to measure the average difference between the predicted values and the actual value. Based on the Figure 4.14, SVM2 or Support Vector Machine that has drop variables if the correlation is lower than 0.5 has the highest score which is 0.057637. Followed by SVM1 with a score of 0.043134. Besides, the second lowest score for the MAE is Decision Tree Regression with a score of 0.019986. Lastly, the lowest MAE score is 0.014809 with the model name Random Forest Regression(RFR). So, the RFR model is the best in this MAE evaluation.

4.2.4. Mean Square Error(MSE)

Mean Square Error(MSE) is to check the average squared difference between the predicted values and the actual values. It is calculated by taking the average of the squared errors. Lower value for MSE indicates a better fit of the model to the data and 0 indicates that the model perfectly predicts the actual values.

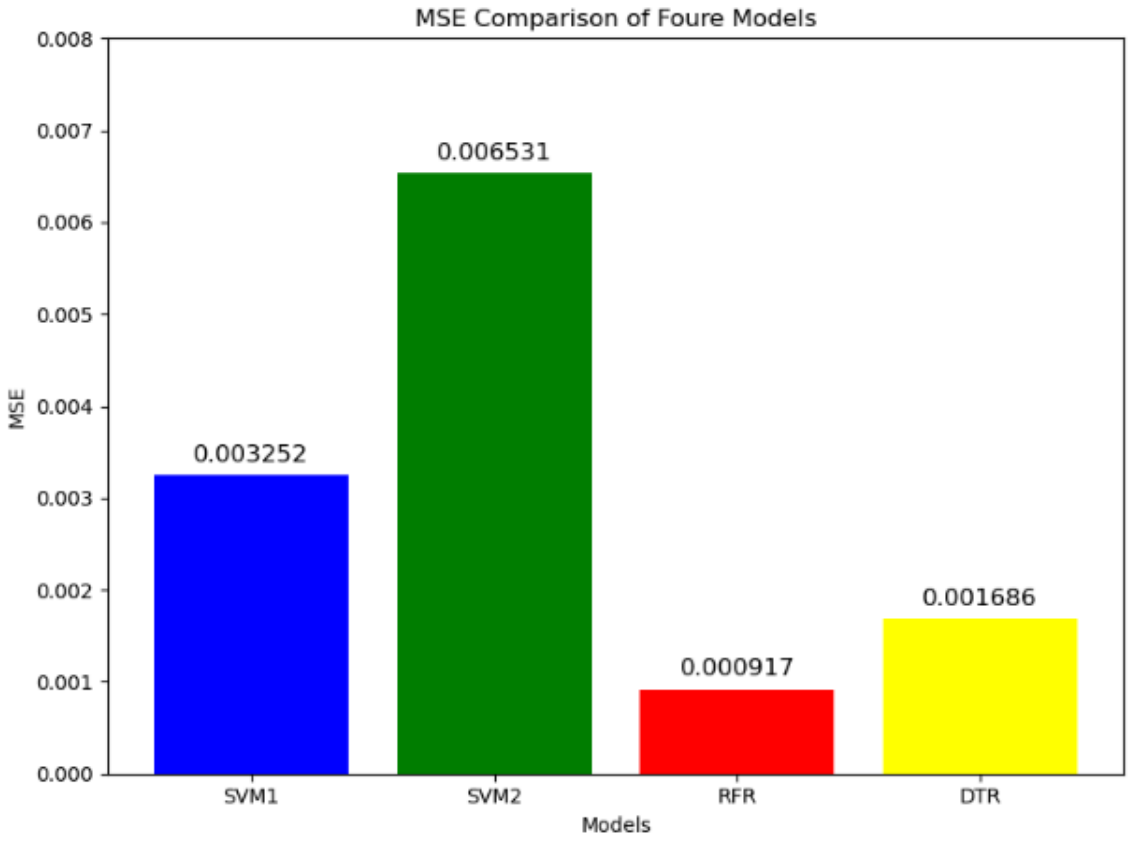


Figure 4.16 MSE Comparison of Four Models

Mean Squared Error(MSE) is a way to measure the average squared difference between predicted and observed values. MSE is a popular metric that is used in regression models. Other than that, MSE also is used for machine learning and statistics. MSE will always be greater than or equal to zero, and the lower of MSE is better to fit the model to the dataset. Based on the figure 4.1 we can conclude that the best model that can get the lowest MSE is the Random Forest Regression (RFR) and the result is 0.000917. Next, for the second lowest MSE is Decision Tree Regression (DTR) and the result is 0.001686. Other than that we also can based on the figure 4.1 to know that the highest MSE result is from Support Vector Machine (SVM). One of the reasons for getting this result is because it may not learn and understand the relationship between the data as well as the Random Forest Regression (RFR) and Decision Tree Regression (DTR). Other than that, it is also possible that the SVM is not really suitable for this task compared with RFR and DTR.

4.2.5. Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is a measure of the average magnitude of the errors and a set of predictions, which direction of the error are taken into account. It is a more intuitive measure of model accuracy than the MSE since it is using the same units. It is also not affected by the scale of the data. The lower the RMSE value the better the fit for the model to the data.

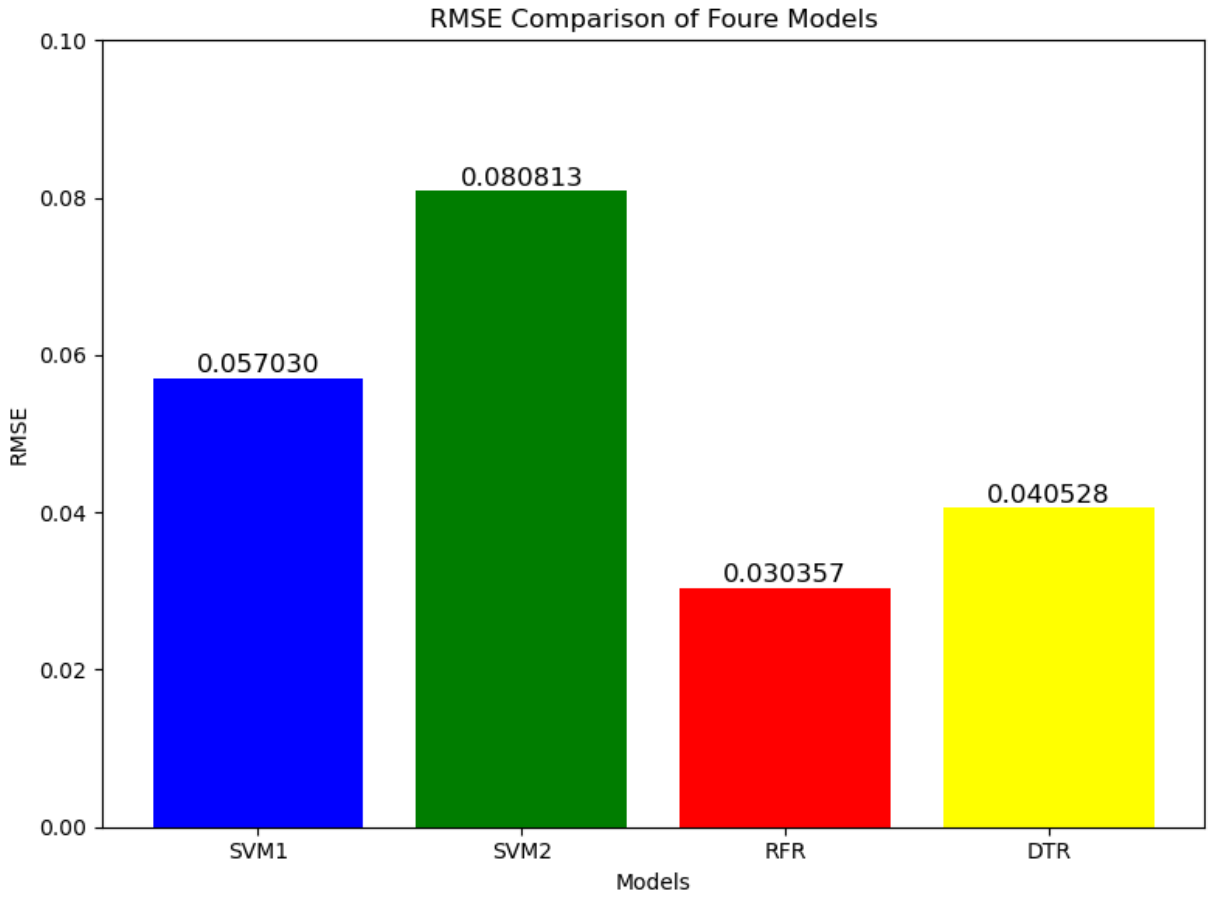


Figure 4.17 RMSE Comparison of Four Models

For the RMSE evaluation, the SVM2 has scored 0.080813 as the highest score among 4 models. For SVM1 who has the RMSE score of 0.057030 which is the second highest after SVM2. Next, DTR has scored 0.040528 for RMSE as the third highest. Lastly, the lowest score among these 4 models is RFR with a score of 0.030357.Overall, RFR has the best performance in this RMSE evaluation.

4.2.6. Explain Variance Score(EVS)

Explain Variance Score(EVS) is a metric for regression that measures how well a model explains the variation in the target variable. It is calculated by taking the ratio of the variance shown by the model to total variance in the target variable. The EVS range is from 0 to 1, the higher the values the closer to the perfect model which explains the variation in the target variable. While the lower the values the model will closer to does not explain any of the variation in the target variable.

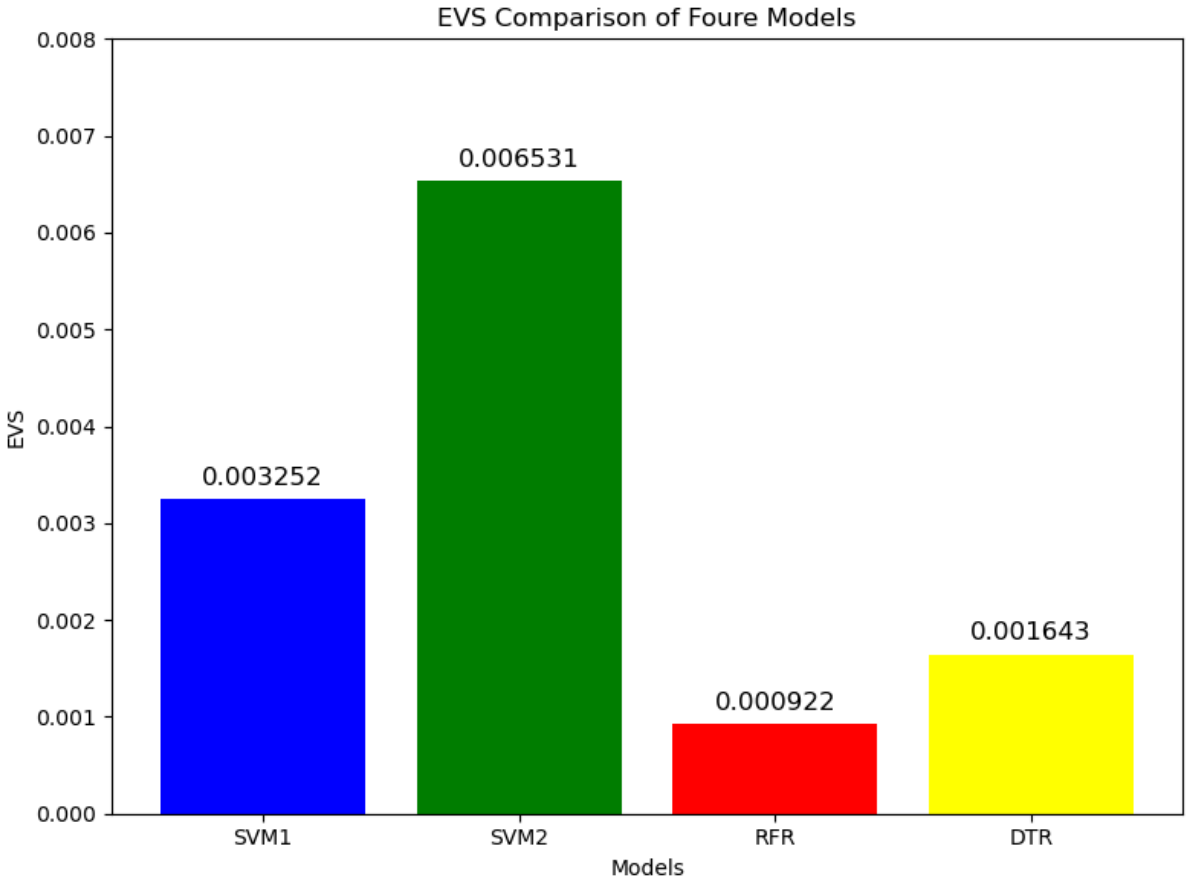


Figure 4.18 EVS Comparison of Four Models

In this EVS evaluation, SVM2 has scored 0.0006531 in EVS which could be considered the best performance. Followed by SVM1, with a score of 0.003252 which could be considered in the second. Next, the DTR has a score of 0.001643 making it the third highest among the 4 models. Lastly, the lowest score within this 4 is RFR with a score of 0.000922.

4.3. Overall

Overall, Random Forest Regression (RFR) has better performance than other 3 models. This is because the Random Forest models are ensemble models. By combining the predictions of multiple decision trees, this will make the model more robust to overfitting and able to learn complex relationships in the data better. Ensemble model in RFR will combine all decision trees from random forest regression, average the prediction to prevent the model from overfit. Besides that, RFR are less sensitive to hyperparameters which means that it is less important to tune the parameters in order to achieve good results.

5.0 EVALUATION

5.1. Achievements

This project has been able to develop a predictive model that is capable of predicting diamond price based on the characteristics. Allowing users to enter the diamond's characteristics and predicting the price from the model. The model that we choose for this diamond prediction is Random Forest Regression (RFR). The reason we choose this is because the RFR are ensemble models that combine prediction models and leverage the strength of each model and minimize the impact of the weakness. RFR are also more robust to overfitting than decision trees or SVM.

Besides, the project is also able to identify the relationship between diamonds' characteristics and its market values. Each of the diamonds characteristics will be able to affect the price of the diamonds. From the figure 2.2, 'x', 'y', 'z' and carat have higher influence on diamond price. Without being able to identify the relationship, this model will not be able to predict diamond price accurately. Therefore, the project has fulfilled the objective.

5.2. Discussions

The limitations of our assignment is our model does not guarantee prediction with high accuracy on the price of diamonds. This could be due to the data quality, complexity of the model and other factors. Moreover, this prediction model might no longer be relevant in the future as the diamond market is constantly changing as well as the factors that affect price of diamond may be subjected to changes too. For example, discovery of new diamond mines can lead to an increase in supply of diamonds and decrease in their price. In the same way, changes in consumer behavior may affect the price of diamonds.

For future improvements, the quantity of data, usage of machine learning algorithms and accessibility of models can be improved. In terms of quantity of data, larger quantities can be used as it will help the model to learn the relationship between different diamond characteristics and price. It is expected that more data will be available in the future, this will allow for development of more accurate and reliable models. Application of new machine learning algorithms that are more suitable can be applied to diamond price prediction in order to produce models that are more efficient and accurate. In addition, there are many diamond prediction models available in the market but not all of them are accessible to small businesses and consumers. Hence, it is recommended that diamond prediction models be made accessible to the public by developing web-based applications or mobile apps that allow users to predict the price of diamond.

6.0 CONCLUSION

In conclusion, the best model to use on this dataset is the Random Forest Regression (RFR). Despite applying the same data preparation steps for all models, the results obtained from each model were different. Random Forest Regression manages to get the lowest Mean Squared Error (MSE) and lowest Mean Absolute Error compared to Support Vector Machine (SVM). Even though it did not achieve the best result for Mean Squared Error and Mean Absolute Error, it still achieved the overall best performance. On the contrary, Support Vector Machine 2 which only used features with more than 0.5 correlation had the overall worst performance. This could be due to a lack of data, resulting in poor and inaccurate predictions.

Being able to predict diamond price is one of the most important objectives that were achieved throughout this assignment. Finding the highest score within 4 models is also one of the objectives that were achieved. Using Random Forest Regression(RFR) to predict diamond price since this RFR has the highest overall score within 4 models. The third objective that was achieved is finding the relationship between all variables.

In order to be able to predict the diamond price, few phases need to be gone through. In the exploratory data analysis phase, understanding and exploring the data is a must for us to understand the patterns, anomalies and trends in the data. Finding the distribution of the data is also included in this phase.

In the data preparation phase, few of the steps will be done in this phase. The first one will be data cleaning, the first thing will be finding all null values for all variables. Then, removing value ‘0’ from the variables ‘x’, ‘y’ and ‘z’ since any variable will mean that the diamond is 2 -dimensional (refer to figure 2.1 to understand the variable). After that, we will use a pair plot to find outliers. The data will be removed if the data is too far from the regression line. Next, all of the variables will remain even if some of the correlation values are not high but will use SVM to train and test the performance of removing variables that have low correlation. Since the dataset has a wide range of values. Therefore, min max scaling will be applied to scale the data between a certain value which is 0 to 1. Lastly, the dataset will be split with the ratio of 70:30 by following the Afshin Gholamy et al. (2018)’s research.

Furthermore, in the modeling phase a total 3 + 1 will be used. The 3 algorithms that will be used for this assignment are Support Vector Machine (SVM), Random Forest Regression (RFR) and Decision Tree Regression (DTR). The other 1 algorithm is using SVM but with lesser variables. So the highest score between these 4 models is RFR. Since RFR is using an ensemble model which will combine all predictions and leverage it.

Through this assignment, we were able to put what we have learnt in class into practice.

This allows us to gain a better understanding of the data science process, including handling data, data understanding, data preprocessing, data transformation, data scaling, model training and deployment. Finally, we would also like to thank our tutor, Ms Noor Aida, for her feedback and guidance in completing this assignment.

REFERENCE

Hashmi, F. (2020) *Diamond price prediction case study in python - Thinking Neuron*. Available at: https://thinkingneuron.com/diamond-price-prediction-case-study-in-python/

International Gem Society LLC (2023) *Diamond Carat Weight, Value, and Size - International Gem Society*. Available at: https://www.gemsociety.org/article/diamond-carat-weight/#.

Vrai (2023) "What is A Diamond Clarity Chart?," *VRAI*, 21 August. Available at https://www.vrai.com/journal/post/diamond-clarity-chart.

Learn what carat means and what diamond carat measures | 4Cs of Diamond Quality by GIA (2019). Available at: https://4cs.gia.edu/en-us/diamond-carat-weight/.

All about Cut | My Diamond Ring (2021). Available at: https://mydiamondring.com/en/article/756841.

Devin (2022) "Diamond Color Grade Guide (With GIA Color Scale)," *StoneAlgo*, 27 January. Available at: https://www.stonealgo.com/blog/the-4cs-what-to-know-about-diamond-color/.

Shiv Shambu (no date) *Shiv Shambu: Diamond Engagement Rings - Diamond Depth*. Available at: https://shivshambu.net/pages/depth.

Shiv Shambu (no date) *Shiv Shambu: Diamond Engagement Rings - Diamond Table*. Available at: https://shivshambu.net/pages/table.

Sethi, A. (2023) "One Hot Encoding vs. Label Encoding using Scikit-Learn," *Analytics Vidhya* [Preprint]. Available at: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/.

Chaijunla, T., & Taninpong, P. (n.d.). *Comparative Study of Predicting Diamond Ring Prices in Online Retail Shop*. Retrieved September 16, 2023, from https://datascience.cmu.ac.th/storage/articles/18.pdf

APPENDIX

Import Library

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

import warnings
warnings.filterwarnings('ignore')
```

Read data

In [2]:
```
df = pd.read_csv('diamonds.csv')

df
```

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
...	...	...	...	...	...	...	...	...	...	...
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

53940 rows × 10 columns

Exploratory Data Analysis

info()

In [3]:
```
df.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
```

dtypes

In [4]:
```
df.dtypes
```

Out[4]:
```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price      float64
'x'        float64
'y'        float64
'z'        float64
dtype: object
```

shape

In [5]:
```
df.shape
```

Out[5]:
```
(53940, 10)
```

describe()

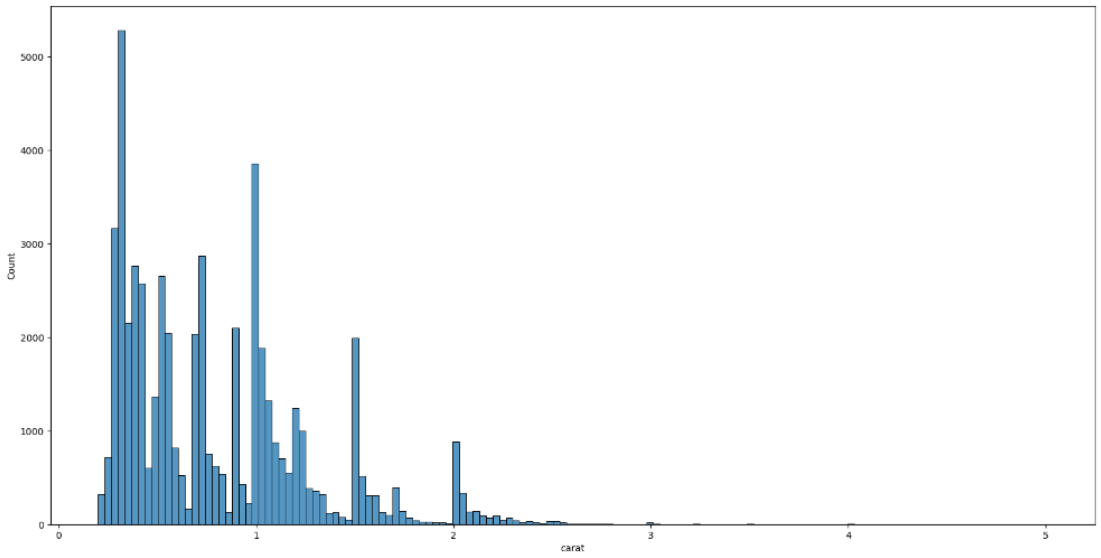
In [6]:
```
df.describe()
```

Out[6]:

	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

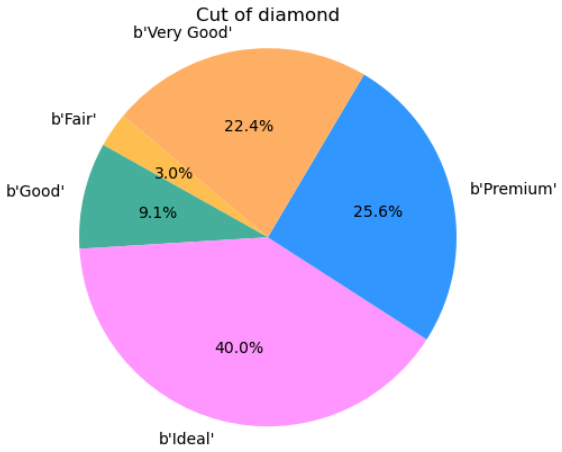
Visualizing each columns

In [7]:
```
plt.subplots(figsize=(20,10))
sns.histplot(df["carat"])
plt.show()
```



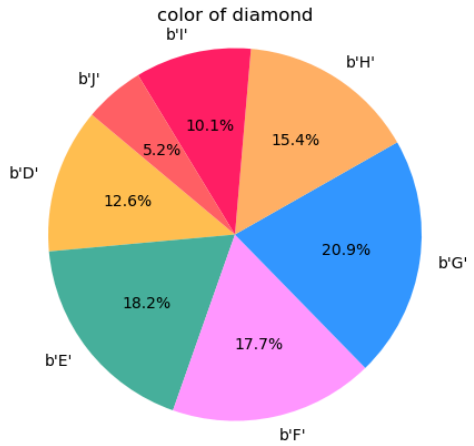
In [8]:
```
counts = df.groupby('cut').size().to_dict()

print(counts)
plt.pie(counts.values(), labels = counts.keys(), colors = ['#FFC154', '#47B39C', '#FF99FF', '#3399FF', '#FFB266'], autopct = '%1.
plt.axis('equal')
plt.title('Cut of diamond')
plt.show()
```

```
{"b'Fair'": 1610, "b'Good'": 4906, "b'Ideal'": 21551, "b'Premium'": 13791, "b'Very Good'": 12082}
```

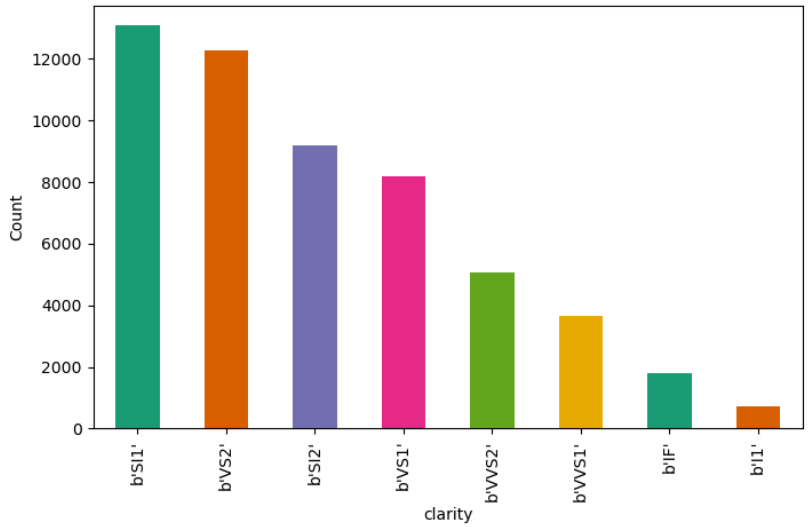


In [9]:
```
counts = df.groupby('color').size().to_dict()

print(counts)
plt.pie(counts.values(), labels = counts.keys(), colors = ['#FFC154', '#47B39C', '#FF99FF', '#3399FF', '#FFB266', '#FF2266', '#FF
plt.axis('equal')
plt.title('color of diamond')
plt.show()
```

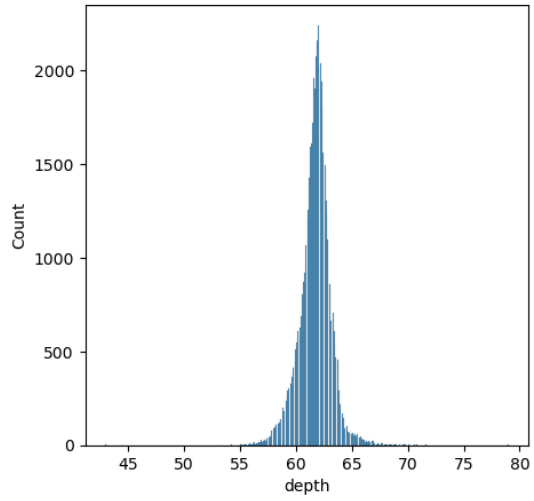
```
{"b'D'": 6775, "b'E'": 9797, "b'F'": 9542, "b'G'": 11292, "b'H'": 8304, "b'I'": 5422, "b'J'": 2808}
```



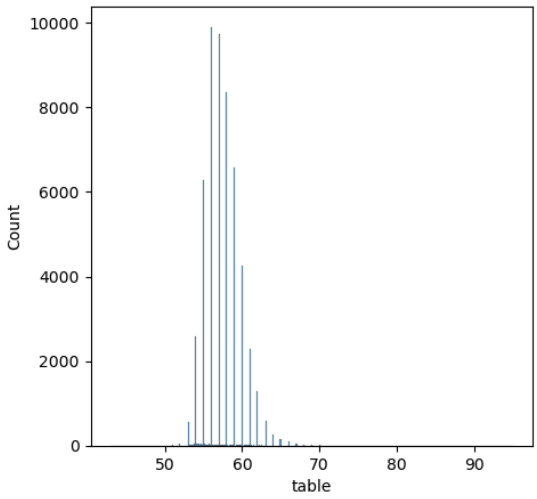
In [21]:
```
def categorical_histogram(df, colname, figscale=1, mpl_palette_name='Dark2'):
    from matplotlib import pyplot as plt
    import seaborn as sns
    count_series = df[colname].value_counts()
    ax = count_series.plot(kind='bar', color=sns.palettes.mpl_palette(mpl_palette_name), figsize=(8, 4.8))

    # Set the x-axis label to 'Clarity' and y-axis label to 'Count'
    ax.set_xlabel(colname)
    ax.set_ylabel('Count')

    return

chart = categorical_histogram(df, 'clarity', **{})
chart
```



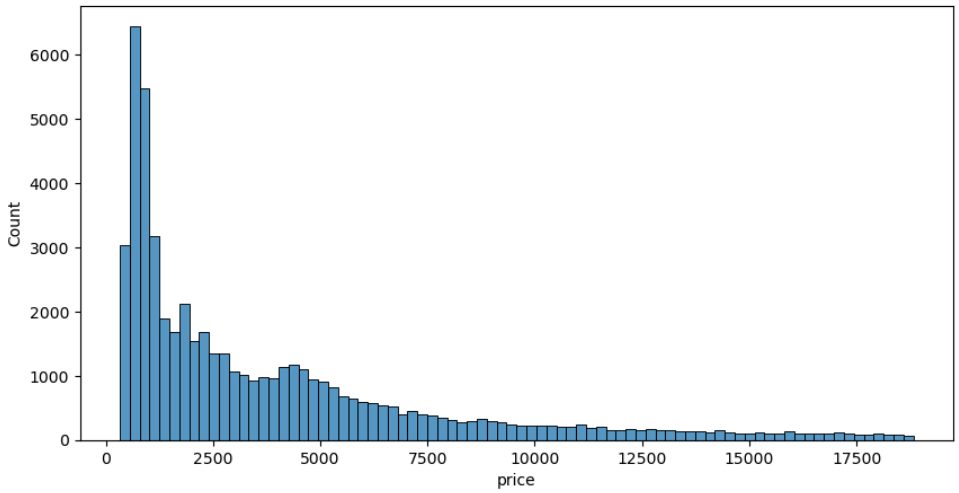
In [22]:
```
plt.subplots(figsize=(5,5))
sns.histplot(df["depth"])
plt.show()
```



In [23]:
```
plt.subplots(figsize=(5,5))
sns.histplot(df["table"])
plt.show()
```



In [24]:
```
plt.subplots(figsize=(10,5))
sns.histplot(df["price"])
plt.show()
```



Data Preprocessing

Checking for missing value

In [25]:
```
df.describe()
```

Out[25]:

	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

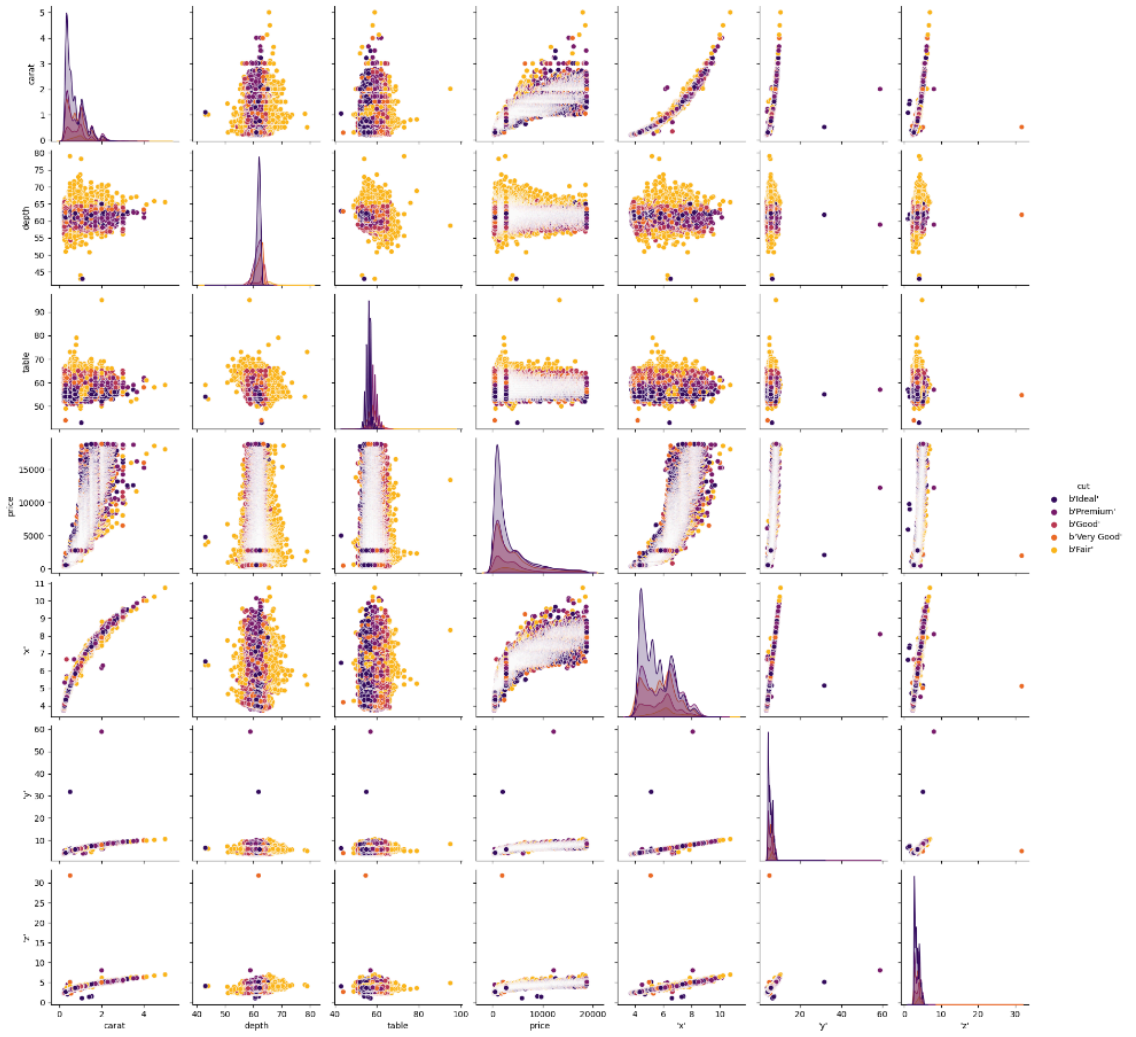
Remove missing value

In [26]:
```
#Dropping dimentionless diamonds
df = df.drop(df[df["'x'"]==0].index)
df = df.drop(df[df["'y'"]==0].index)
df = df.drop(df[df["'z'"]==0].index)
df.shape
```

Out[26]:
```
(53920, 10)
```

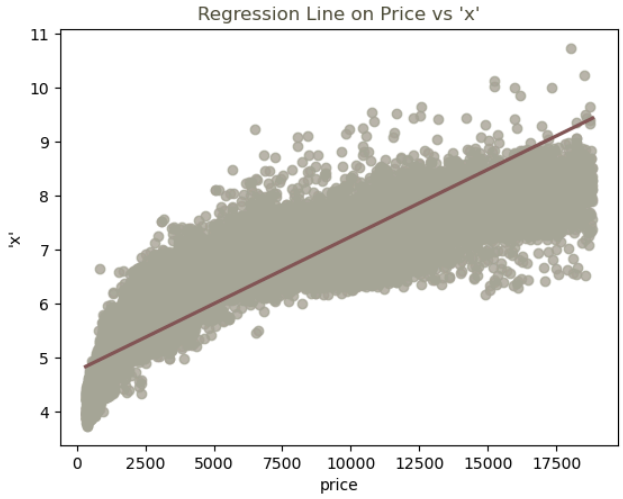
Checking Outliers

In [27]:
```
ax = sns.pairplot(df, hue="cut", palette="inferno")
```



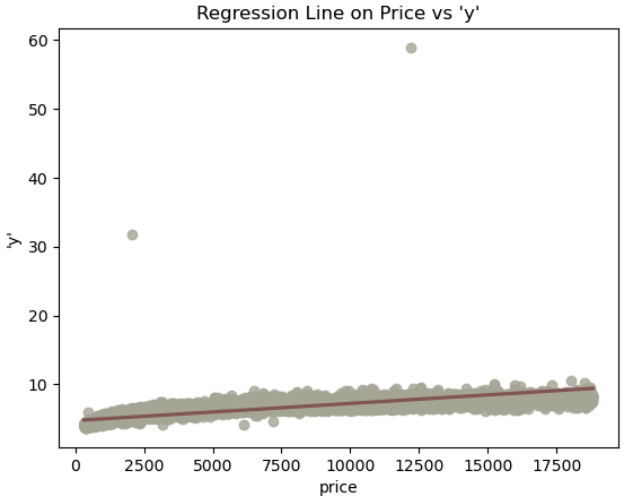
In [28]:
```
ax=sns.regplot(x="price", y="'x'", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'x'", color="#4e4c39")
```

Out[28]:
```
Text(0.5, 1.0, "Regression Line on Price vs 'x'")
```



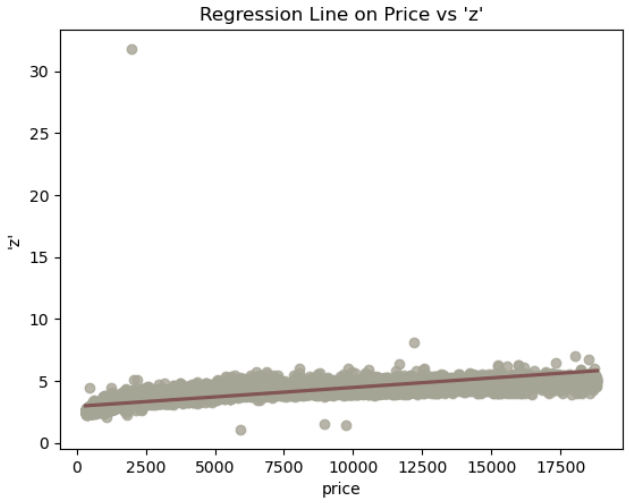
In [29]:
```
ax = sns.regplot(x="price", y="'y'", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'y'")
```

Out[29]:
```
Text(0.5, 1.0, "Regression Line on Price vs 'y'")
```



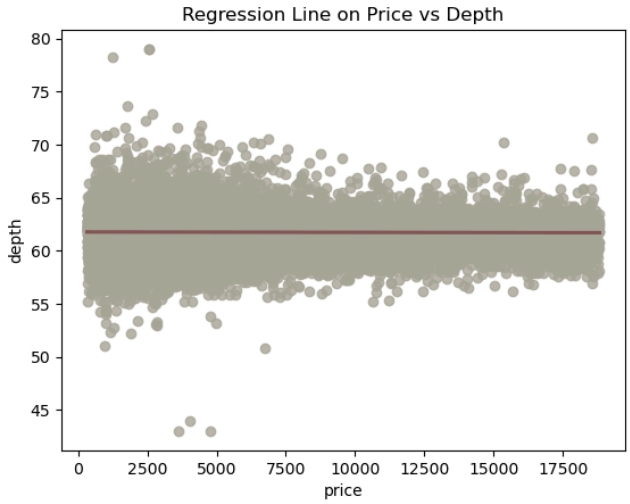
In [30]:
```
ax= sns.regplot(x="price", y="'z'", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'z'")
```

Out[30]: Text(0.5, 1.0, "Regression Line on Price vs 'z'")



In [31]:
```
ax= sns.regplot(x="price", y="depth", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Depth")
```

Out[31]: Text(0.5, 1.0, 'Regression Line on Price vs Depth')

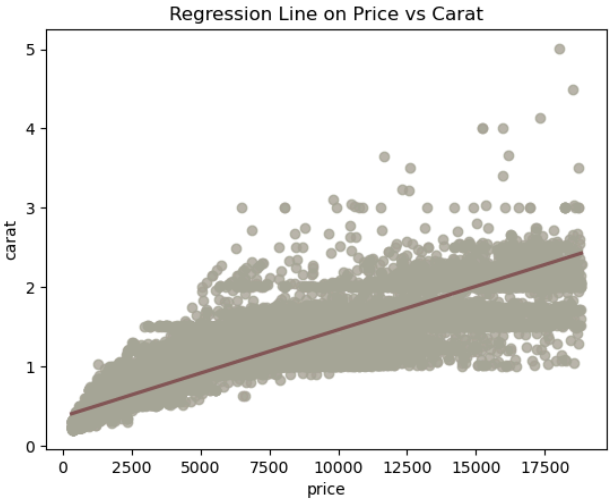


In [32]:
```
ax=sns.regplot(x="price", y="table", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Table")
```

Out[32]: Text(0.5, 1.0, 'Regression Line on Price vs Table')

In [33]:
```
ax=sns.regplot(x="price", y="carat", data=df, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Carat")
```

Out[33]: Text(0.5, 1.0, 'Regression Line on Price vs Carat')



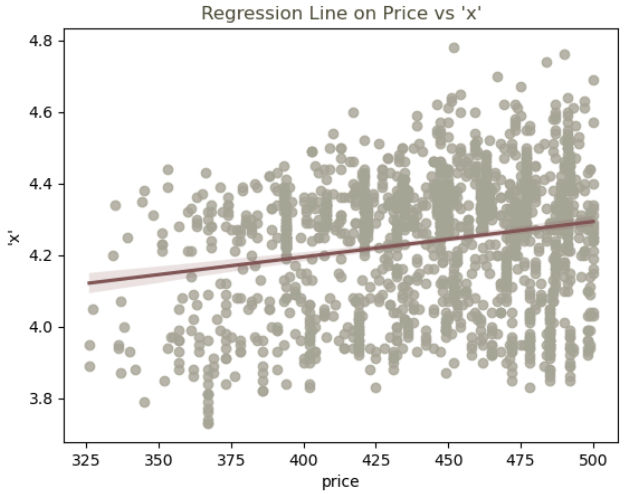
Price set below or equal $500

Demo purpose

In [34]:
```
df_shrink = df[(df["price"] <= 500)]
```

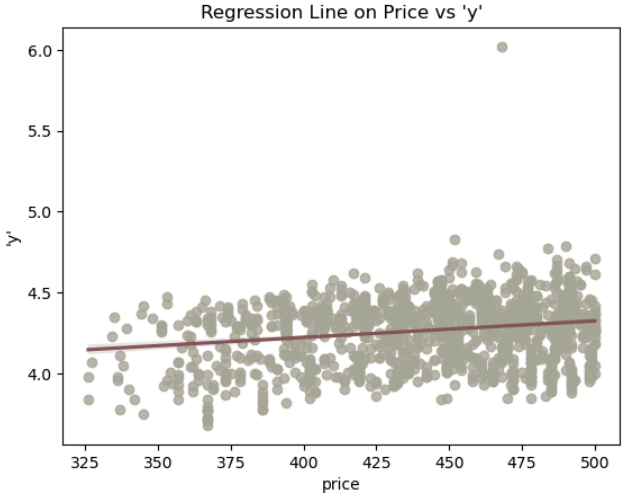
In [35]:
```
ax=sns.regplot(x="price", y="'x'", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'x'", color="#4e4c39")
```

Out[35]: Text(0.5, 1.0, "Regression Line on Price vs 'x'")



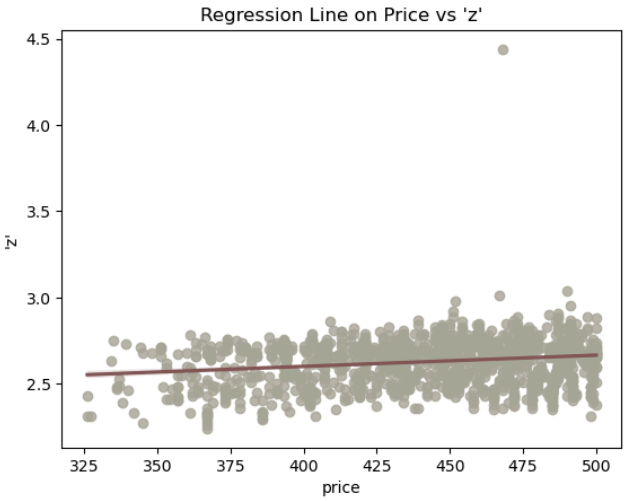
In [36]:
```
ax = sns.regplot(x="price", y="'y'", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"
ax.set_title("Regression Line on Price vs 'y'")
```

Out[36]: Text(0.5, 1.0, "Regression Line on Price vs 'y'")



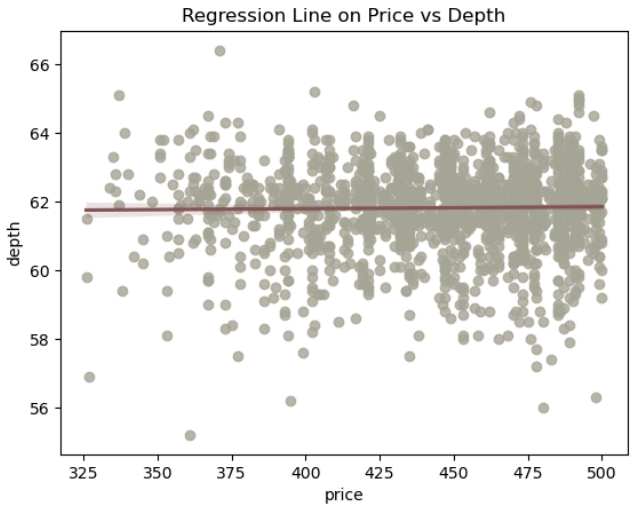
In [37]:
```
ax= sns.regplot(x="price", y="'z'", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"}
ax.set_title("Regression Line on Price vs 'z'")
```

Out[37]:
```
Text(0.5, 1.0, "Regression Line on Price vs 'z'")
```



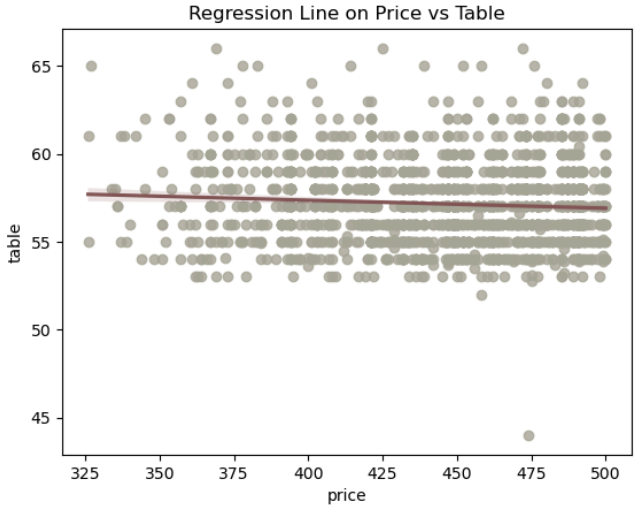
In [38]:
```
ax= sns.regplot(x="price", y="depth", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656
ax.set_title("Regression Line on Price vs Depth")
```

Out[38]:
```
Text(0.5, 1.0, 'Regression Line on Price vs Depth')
```



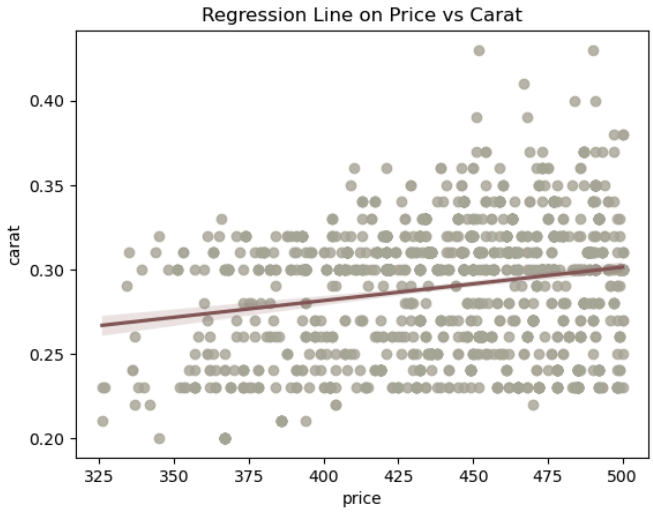
In [39]:
```
ax=sns.regplot(x="price", y="table", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"
ax.set_title("Regression Line on Price vs Table")
```

Out[39]: Text(0.5, 1.0, 'Regression Line on Price vs Table')



In [40]:
```
ax=sns.regplot(x="price", y="carat", data=df_shrink, fit_reg=True, scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"
ax.set_title("Regression Line on Price vs Carat")
```

Out[40]: Text(0.5, 1.0, 'Regression Line on Price vs Carat')



Remove Outliers

In [41]:
```
#dropping outlier
df = df[(df["'y'"]<20)]
df = df[(df["'z'"]<10)&(df["'z'"]>2)]
df = df[(df["depth"]<72)&(df["depth"]>52)]
df = df[(df["table"]<78)&(df["table"]>45)]
df = df[(df["carat"]<3)]
df.shape
```

Out[41]: (53859, 10)

Encoding character or string into numerical value

In [42]:
```
# Create a label encoder object
label_encoder = LabelEncoder()

# Fit the label encoder object to the category data
# Transform the category data using the transform() method of the label encoder object
label_encoder.fit(df['cut'])
df['cut'] = label_encoder.transform(df['cut'])

label_encoder.fit(df['color'])
df['color'] = label_encoder.transform(df['color'])

label_encoder.fit(df['clarity'])
df['clarity'] = label_encoder.transform(df['clarity'])

# Print the data
df
```

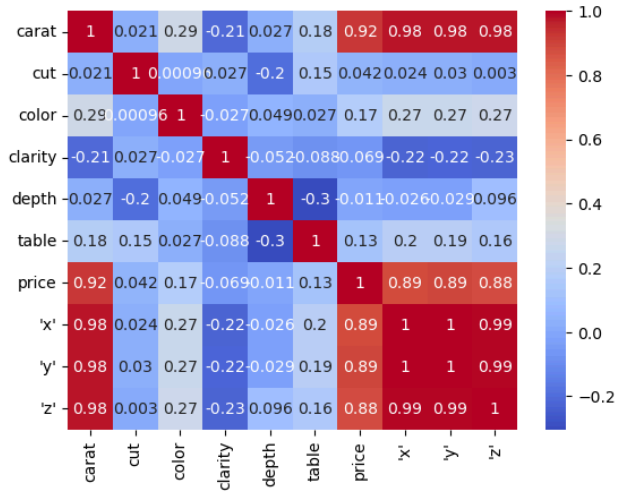
Out[42]:

	carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
0	0.23	2	1	3	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	5	62.4	58.0	334.0	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335.0	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	2	0	2	60.8	57.0	2757.0	5.75	5.76	3.50
53936	0.72	1	0	2	63.1	55.0	2757.0	5.69	5.75	3.61
53937	0.70	4	0	2	62.8	60.0	2757.0	5.66	5.68	3.56
53938	0.86	3	4	3	61.0	58.0	2757.0	6.15	6.12	3.74
53939	0.75	2	0	3	62.2	55.0	2757.0	5.83	5.87	3.64

53859 rows × 10 columns

Correlation matrix

In [43]:
```
# Plot the correlation matrix
corr_matrix = df.corr()
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
plt.show()
```



Scaler Data

In [44]:
```
from sklearn.preprocessing import MinMaxScaler
df_scaled = df.copy()

col = df.columns

scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df_scaled)
df_scaled = pd.DataFrame(data=df_scaled,columns=[col])
```

In [45]:
```
print(df_scaled)
```

```
          carat   cut     color   clarity     depth     table     price  \
0      0.011538  0.50  0.166667  0.428571  0.474490  0.222222  0.000000
1      0.003846  0.75  0.166667  0.285714  0.387755  0.444444  0.000000
2      0.011538  0.25  0.166667  0.571429  0.239796  0.592593  0.000054
3      0.034615  0.75  0.833333  0.714286  0.520408  0.333333  0.000433
4      0.042308  0.25  1.000000  0.428571  0.566327  0.333333  0.000487
...         ...   ...       ...       ...       ...       ...       ...
53854  0.200000  0.50  0.000000  0.285714  0.438776  0.296296  0.131427
53855  0.200000  0.25  0.000000  0.285714  0.556122  0.222222  0.131427
53856  0.192308  1.00  0.000000  0.285714  0.540816  0.407407  0.131427
53857  0.253846  0.75  0.666667  0.428571  0.448980  0.333333  0.131427
53858  0.211538  0.50  0.000000  0.428571  0.510204  0.222222  0.131427

           'x'       'y'       'z'
0      0.040441  0.055046  0.090244
1      0.029412  0.029358  0.060976
2      0.058824  0.071560  0.060976
3      0.086397  0.100917  0.139024
4      0.112132  0.122936  0.168293
...         ...       ...       ...
53854  0.371324  0.381651  0.351220
53855  0.360294  0.379817  0.378049
53856  0.354779  0.366972  0.365854
53857  0.444853  0.447706  0.409756
53858  0.386029  0.401835  0.385366

[53859 rows x 10 columns]
```

In [46]:
```
yS = df['price']

scaler = MinMaxScaler()
yS = pd.DataFrame(data=yS,columns=['price'])
yS = scaler.fit_transform(yS)
yS = pd.DataFrame(data=yS,columns=['price'])
```

In [47]:
```
yS
```

Out[47]:

	price
0	0.000000
1	0.000000
2	0.000054
3	0.000433
4	0.000487
...	...
53854	0.131427
53855	0.131427
53856	0.131427
53857	0.131427
53858	0.131427

53859 rows × 1 columns

Split data into training data and test data

In [48]:
```
X = df_scaled.drop('price', axis=1, level=0)

y = df_scaled['price']


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [49]:
```
Xa = df_scaled.drop(['price','cut','color','clarity','depth','table'], axis=1)

ya = df_scaled['price']


from sklearn.model_selection import train_test_split
X_train_a, X_test_a, y_train_a, y_test_a = train_test_split(Xa, ya, test_size=0.3, random_state=1)
```

In [ ]:
```

```

Support Vector Machine (SVM) (Tai Qi Zheng)

Support Vector Machine (SVM) With all variable

In [50]:
```
LL_svr = SVR().fit(X_train, y_train)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explained_variance_score

predSVM = LL_svr.predict(X_test)

print('The R^2 is: %f'% r2_score(y_test, predSVM))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test, predSVM))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test, predSVM))
print('The MSE is: %f'% mean_squared_error(y_test, predSVM))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test, predSVM)))
print('The EVS is: %.2f'% explained_variance_score(y_test, predSVM))
```

```
The R^2 is: 0.927310
The Adjusted R^2 is: 0.9272696361944075
The MAE is: 0.043134
The MSE is: 0.003252
The RMSE is: 0.06
The EVS is: 0.93
```

In [51]:
```
print(predSVM)
print(y_test)
```

```
[0.03850563 0.15954069 0.09906992 ... 0.47532011 0.22046229 0.124398  ]
          price
85     0.012326
47338  0.082392
42573  0.054387
32163  0.025031
40040  0.042602
...         ...
29498  0.020490
11697  0.014597
19191  0.412337
4195   0.175001
53079  0.124290

[16158 rows x 1 columns]
```

Support Vector Machine (SVM) With all variable that correlation is more than 0.5

In [52]:
```
LL_svr = SVR().fit(X_train_a, y_train_a)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, explained_variance_score

predSVM_a = LL_svr.predict(X_test_a)

print('The R^2 is: %f'% r2_score(y_test_a, predSVM_a))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test_a, predSVM_a))*(len(y_test_a)-1)/(len(y_test_a)-X_test_a.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test_a, predSVM_a))
print('The MSE is: %f'% mean_squared_error(y_test_a, predSVM_a))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test_a, predSVM_a)))
print('The EVS is: %.2f'% explained_variance_score(y_test_a, predSVM_a))
```

```
The R^2 is: 0.854041
The Adjusted R^2 is: 0.854004930090795
The MAE is: 0.057637
The MSE is: 0.006531
The RMSE is: 0.08
The EVS is: 0.86
```

Random Forest Regression

In [53]:
```
# Create a random forest regressor model
rfr = RandomForestRegressor()

# Train the model on the training data
rfr.fit(X_train, y_train)

# Make predictions on the test data
rfr_predictions = rfr.predict(X_test)

print('The R^2 is: %f'% r2_score(y_test, rfr_predictions))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test, rfr_predictions))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test, rfr_predictions))
print('The MSE is: %f'% mean_squared_error(y_test, rfr_predictions))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test, rfr_predictions)))
print('The EVS is: %.2f'% explained_variance_score(y_test, rfr_predictions))
```

```
The R^2 is: 0.979729
The Adjusted R^2 is: 0.9797173759661288
The MAE is: 0.014724
The MSE is: 0.000907
The RMSE is: 0.03
The EVS is: 0.98
```

In [54]:
```
print(rfr_predictions)
print(y_test)
```

```
[0.01125155 0.08396767 0.0518284  ... 0.4198762  0.19852895 0.14279559]
         price
85     0.012326
47338  0.082392
42573  0.054387
32163  0.025031
40040  0.042602
...         ...
29498  0.020490
11697  0.014597
19191  0.412337
4195   0.175001
53079  0.124290

[16158 rows x 1 columns]
```

Decision Tree Regression

In [55]:
```
# Create a decision tree regressor model
dtf = DecisionTreeRegressor()

# Train the model on the training data
dtf.fit(X_train, y_train)

# Make predictions on the test data
dtf_predictions = dtf.predict(X_test)

print('The R^2 is: %f'% r2_score(y_test, dtf_predictions))
print("The Adjusted R^2 is:",1 - (1-r2_score(y_test, dtf_predictions))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('The MAE is: %f'% mean_absolute_error(y_test, dtf_predictions))
print('The MSE is: %f'% mean_squared_error(y_test, dtf_predictions))
print('The RMSE is: %.2f'% np.sqrt(mean_squared_error(y_test, dtf_predictions)))
print('The EVS is: %.2f'% explained_variance_score(y_test, dtf_predictions))
```

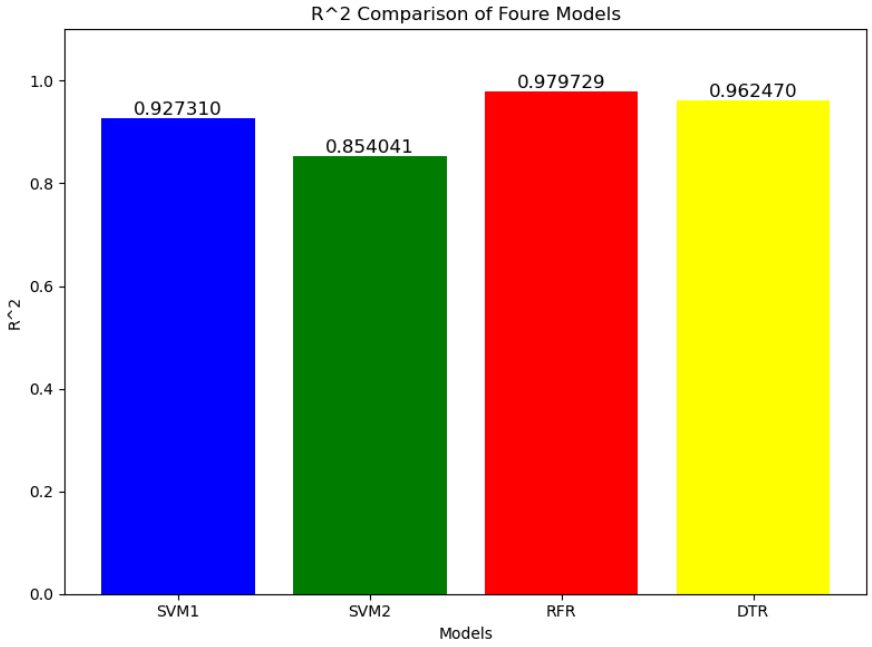
```
The R^2 is: 0.962470
The Adjusted R^2 is: 0.9624493104048226
The MAE is: 0.019981
The MSE is: 0.001679
The RMSE is: 0.04
The EVS is: 0.96
```

In [56]:
```
print(dtf_predictions)
print(y_test)
```

```
[0.01913824 0.08082392 0.04935936 ... 0.40466022 0.21171001 0.14942964]
         price
85     0.012326
47338  0.082392
42573  0.054387
32163  0.025031
40040  0.042602
...         ...
29498  0.020490
11697  0.014597
19191  0.412337
4195   0.175001
53079  0.124290

[16158 rows x 1 columns]
```

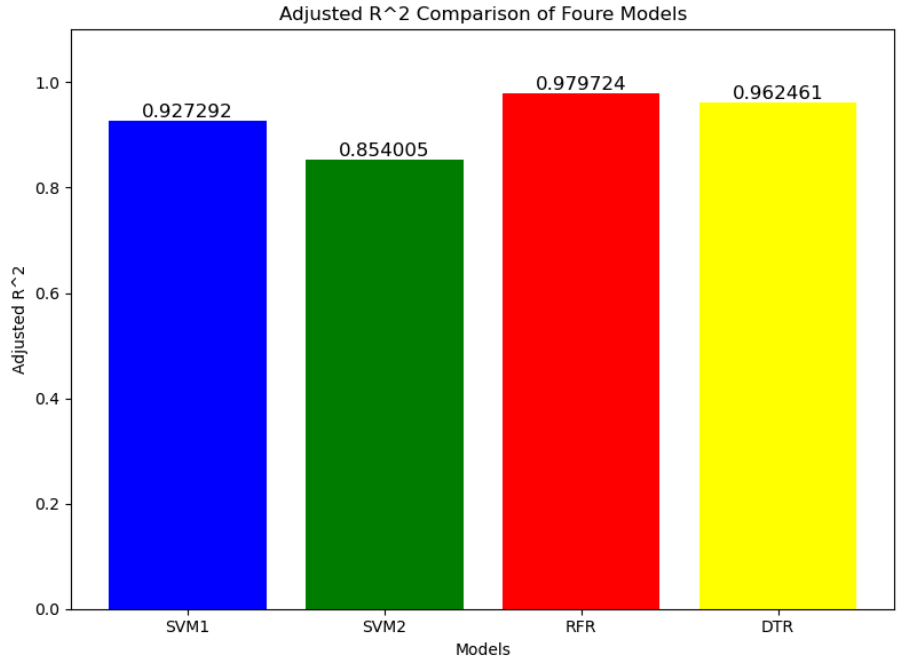
R^2 scores for the three models

In [58]:
```
#Accuracy scores for the three models
model_names = ['SVM1', 'SVM2', 'RFR', 'DTR']
R2SVM = r2_score(y_test, predSVM)
R2SVM2 = r2_score(y_test_a, predSVM_a)
R2RFR = r2_score(y_test, rfr_predictions)
R2DTR = r2_score(y_test, dtf_predictions)
R2 = [R2SVM, R2SVM2, R2RFR, R2DTR]

#Create a bar graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, R2, color=['blue', 'green', 'red', 'yellow'])
plt.xlabel('Models')
plt.ylabel('R^2')
plt.title('R^2 Comparison of Foure Models')
plt.ylim(0, 1.1)  # Set the y-axis limit between 0 and 0.008 (accuracy range)

#Display the accuracy values on top of each bar
for i, score in enumerate(R2):
    plt.text(i, score + 0.0001, f'{score:f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```



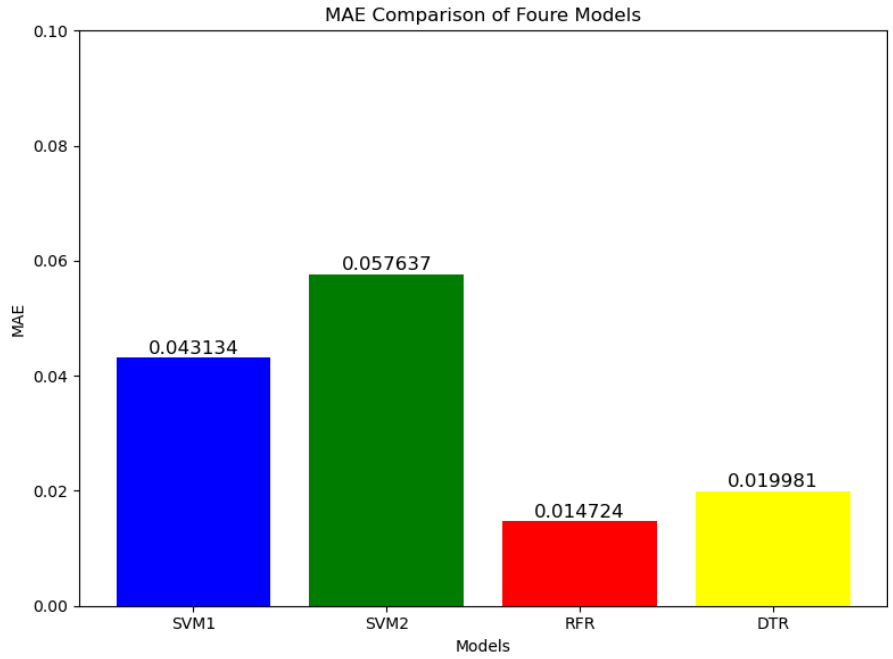
Adjusted R^2 scores for the three models

In [59]:
```
#Accuracy scores for the three models
model_names = ['SVM1', 'SVM2', 'RFR', 'DTR']
ARSVM = 1 - (1-r2_score(y_test, predSVM))*(len(y_test_a)-1)/(len(y_test_a)-X_test_a.shape[1]-1)
ARSVM2 = 1 - (1-r2_score(y_test_a, predSVM_a))*(len(y_test_a)-1)/(len(y_test_a)-X_test_a.shape[1]-1)
ARRFR = 1 - (1-r2_score(y_test, rfr_predictions))*(len(y_test_a)-1)/(len(y_test_a)-X_test_a.shape[1]-1)
ARDTR = 1 - (1-r2_score(y_test, dtf_predictions))*(len(y_test_a)-1)/(len(y_test_a)-X_test_a.shape[1]-1)
AR = [ARSVM, ARSVM2, ARRFR, ARDTR]

#Create a bar graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, AR, color=['blue', 'green', 'red', 'yellow'])
plt.xlabel('Models')
plt.ylabel('Adjusted R^2')
plt.title('Adjusted R^2 Comparison of Foure Models')
plt.ylim(0, 1.1)  # Set the y-axis limit between 0 and 0.008 (accuracy range)

#Display the accuracy values on top of each bar
for i, score in enumerate(AR):
    plt.text(i, score + 0.0001, f'{score:f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```



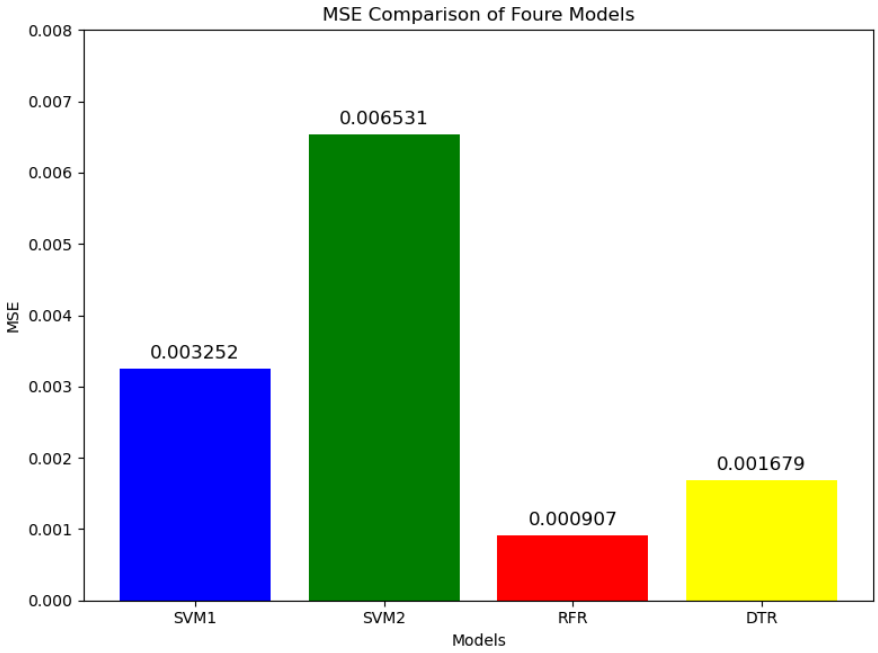
MAE scores for the three models

In [60]:
```
#Accuracy scores for the three models
model_names = ['SVM1', 'SVM2', 'RFR', 'DTR']
MAESVM = mean_absolute_error(y_test_a, predSVM)
MAESVM2 = mean_absolute_error(y_test_a, predSVM_a)
MAERFR = mean_absolute_error(y_test_a, rfr_predictions)
MAEDTR = mean_absolute_error(y_test_a, dtf_predictions)
MAE = [MAESVM, MAESVM2, MAERFR, MAEDTR]

#Create a bar graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, MAE, color=['blue', 'green', 'red', 'yellow'])
plt.xlabel('Models')
plt.ylabel('MAE')
plt.title('MAE Comparison of Foure Models')
plt.ylim(0, 0.1)  # Set the y-axis limit between 0 and 0.008 (accuracy range)

#Display the accuracy values on top of each bar
for i, score in enumerate(MAE):
    plt.text(i, score + 0.0001, f'{score:f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```



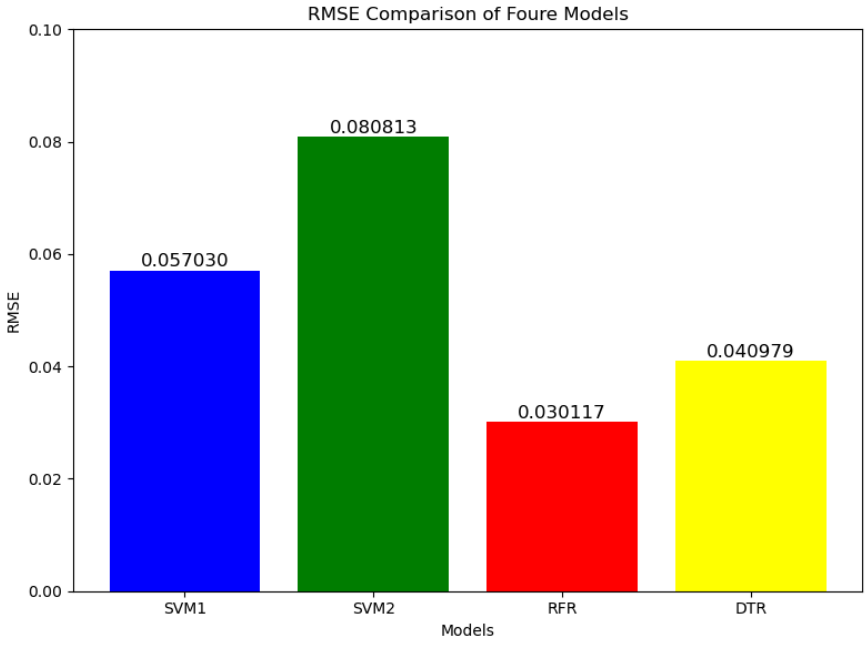
MSE scores for the three models

In [61]:
```
#Accuracy scores for the three models
model_names = ['SVM1', 'SVM2', 'RFR', 'DTR']
MSESVM = mean_squared_error(y_test, predSVM)
MSESVM2 = mean_squared_error(y_test_a, predSVM_a)
MSERFR = mean_squared_error(y_test, rfr_predictions)
MSEDTR = mean_squared_error(y_test, dtf_predictions)
MSE = [MSESVM, MSESVM2, MSERFR, MSEDTR]

#Create a bar graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, MSE, color=['blue', 'green', 'red', 'yellow'])
plt.xlabel('Models')
plt.ylabel('MSE')
plt.title('MSE Comparison of Foure Models')
plt.ylim(0, 0.008)  # Set the y-axis limit between 0 and 0.008 (accuracy range)

#Display the accuracy values on top of each bar
for i, score in enumerate(MSE):
    plt.text(i, score + 0.0001, f'{score:f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```



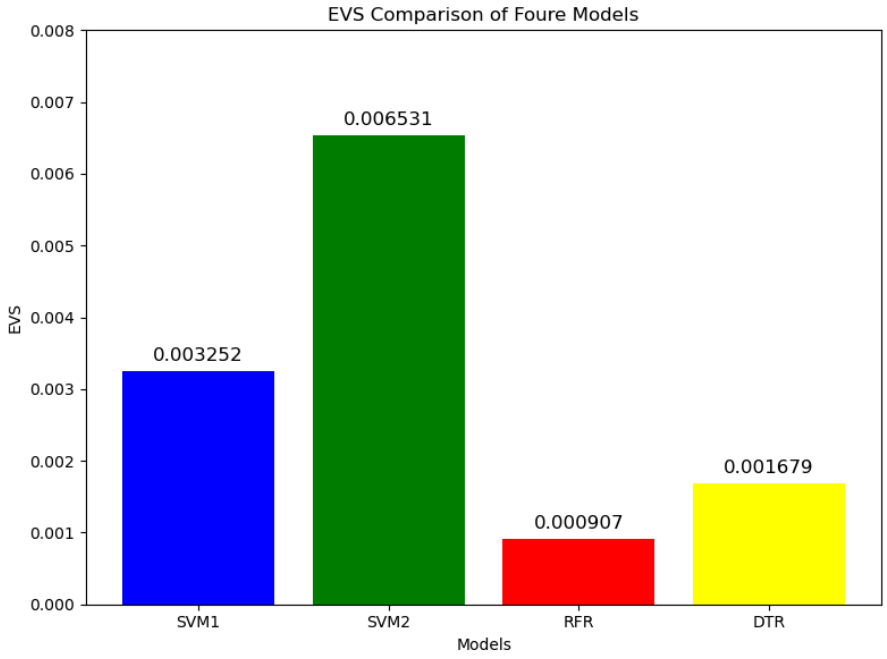
RMSE scores for the three models

In [62]:
```
#Accuracy scores for the three models
model_names = ['SVM1', 'SVM2', 'RFR', 'DTR']
RMSESVM = np.sqrt(mean_squared_error(y_test, predSVM))
RMSESVM2 = np.sqrt(mean_squared_error(y_test_a, predSVM_a))
RMSERFR = np.sqrt(mean_squared_error(y_test, rfr_predictions))
RMSEDTR = np.sqrt(mean_squared_error(y_test, dtf_predictions))
RMSE = [RMSESVM, RMSESVM2, RMSERFR, RMSEDTR]

#Create a bar graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, RMSE, color=['blue', 'green', 'red', 'yellow'])
plt.xlabel('Models')
plt.ylabel('RMSE')
plt.title('RMSE Comparison of Foure Models')
plt.ylim(0, 0.1)  # Set the y-axis limit between 0 and 0.008 (accuracy range)

#Display the accuracy values on top of each bar
for i, score in enumerate(RMSE):
    plt.text(i, score + 0.0001, f'{score:f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```



Explained Variance Score for the three models

In [64]:
```
#Accuracy scores for the three models
model_names = ['SVM1', 'SVM2', 'RFR', 'DTR']
EVSSVM = explained_variance_score(y_test, predSVM)
EVSSVM2 = explained_variance_score(y_test_a, predSVM_a)
EVSRFR = explained_variance_score(y_test, rfr_predictions)
EVSDTR = explained_variance_score(y_test, dtf_predictions)
EVS = [MSESVM, MSESVM2, MSERFR, MSEDTR]

#Create a bar graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, EVS, color=['blue', 'green', 'red', 'yellow'])
plt.xlabel('Models')
plt.ylabel('EVS')
plt.title('EVS Comparison of Foure Models')
plt.ylim(0, 0.008)  # Set the y-axis limit between 0 and 0.008 (accuracy range)

#Display the accuracy values on top of each bar
for i, score in enumerate(EVS):
    plt.text(i, score + 0.0001, f'{score:f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()
```



unscaler data

In [65]:
```
predSVM = pd.DataFrame(data=predSVM,columns=['price'])
predSVM = scaler.inverse_transform(predSVM)
predSVM
```

Out[65]:
```
array([[1038.23867818],
       [3277.02408456],
       [2158.49627262],
       ...,
       [9117.99610266],
       [4403.89103051],
       [2626.98979249]])
```

In [66]:
```
predSVM_a = pd.DataFrame(data=predSVM_a,columns=['price'])
predSVM_a = scaler.inverse_transform(predSVM_a)
predSVM_a
```

Out[66]:
```
array([[ -413.33058694],
       [ 2823.27754852],
       [ 2594.71778857],
       ...,
       [11013.2445187 ],
       [ 5233.88489171],
       [ 2665.35199446]])
```

In [67]:
```
rfr_predictions = pd.DataFrame(data=rfr_predictions,columns=['price'])
rfr_predictions = scaler.inverse_transform(rfr_predictions)
rfr_predictions
```

Out[67]:
```
array([[ 534.12],
       [1879.15],
       [1284.67],
       ...,
       [8092.45],
       [3998.19],
       [2967.29]])
```

In [68]:
```
dtf_predictions = pd.DataFrame(data=dtf_predictions,columns=['price'])
dtf_predictions = scaler.inverse_transform(dtf_predictions)
dtf_predictions
```

Out[68]:
```
array([[ 680.],
       [1821.],
       [1239.],
       ...,
       [7811.],
       [4242.],
       [3090.]])
```

In [69]:
```
y_test = scaler.inverse_transform(y_test)
y_test
```

Out[69]:
```
array([[ 554.],
       [1850.],
       [1332.],
       ...,
       [7953.],
       [3563.],
       [2625.]])
```

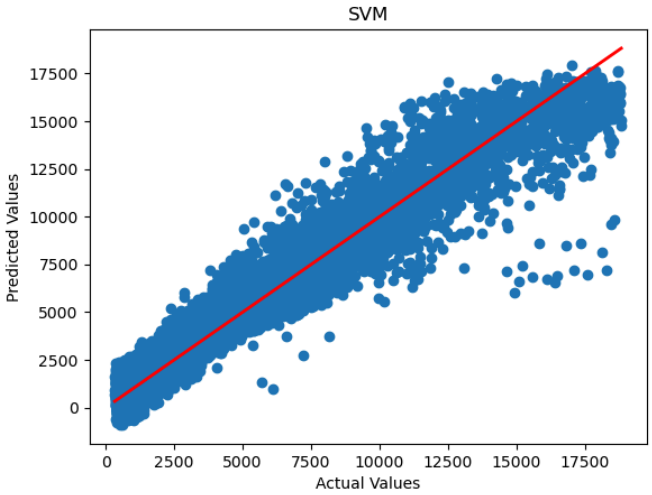
In [70]:
```
y_test_a = scaler.inverse_transform(y_test_a)
y_test_a
```

Out[70]:
```
array([[ 554.],
       [1850.],
       [1332.],
       ...,
       [7953.],
       [3563.],
       [2625.]])
```

Support Vector Machine (SVM) (Tai Qi Zheng)

Support Vector Machine (SVM) With all variable

In [71]:
```
plt.scatter(y_test, predSVM)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='r', linewidth=2, label='Perfect Prediction')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('SVM')
plt.show()
```



In [72]:
```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test)
TestingDataResults['TargetColumn']=y_test
TestingDataResults['Prediction']=predSVM
TestingDataResults
```

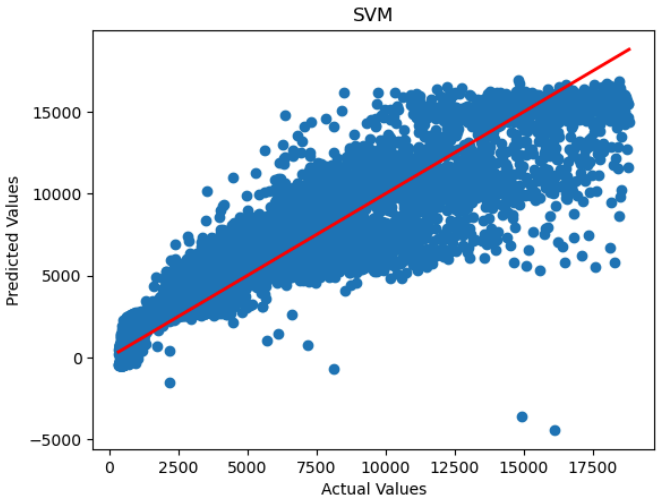
Out[72]:

	carat	cut	color	clarity	depth	table	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.75	0.500000	0.857143	0.515306	0.370370	0.040441	0.044037	0.095122	554.0	1038.238678
47338	0.161538	0.25	0.500000	0.714286	0.448980	0.481481	0.308824	0.333945	0.309756	1850.0	3277.024085
42573	0.115385	0.00	0.666667	0.714286	0.209184	0.592593	0.277574	0.280734	0.214634	1332.0	2158.496273
32163	0.042308	0.50	0.500000	0.857143	0.505102	0.222222	0.110294	0.124771	0.156098	789.0	654.471480
40040	0.046154	0.50	0.000000	0.571429	0.443878	0.296296	0.137868	0.141284	0.160976	1114.0	-10.724709
...	...	...	...	...	...	...	...	...	...	...	...
29498	0.080769	0.25	0.333333	0.285714	0.566327	0.333333	0.180147	0.194495	0.226829	705.0	973.947273
11697	0.053846	0.50	0.500000	0.714286	0.474490	0.296296	0.136029	0.152294	0.170732	596.0	497.070099
19191	0.534615	0.75	1.000000	0.714286	0.530612	0.370370	0.683824	0.688073	0.634146	7953.0	9117.996103
4195	0.311538	1.00	0.833333	0.285714	0.556122	0.333333	0.487132	0.486239	0.475610	3563.0	4403.891031
53079	0.150000	0.50	0.333333	0.857143	0.520408	0.259259	0.299632	0.311927	0.314634	2625.0	2626.989792

16158 rows × 11 columns

Support Vector Machine (SVM) With all variable that correlation is more than 0.5

In [73]:
```
plt.scatter(y_test_a, predSVM_a)
plt.plot([min(y_test_a), max(y_test_a)], [min(y_test_a), max(y_test_a)], color='r', linewidth=2, label='Perfect Prediction')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('SVM')
plt.show()
```



In [74]:
```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test_a)
TestingDataResults['TargetColumn']=y_test_a
TestingDataResults['Prediction']=predSVM_a
TestingDataResults
```

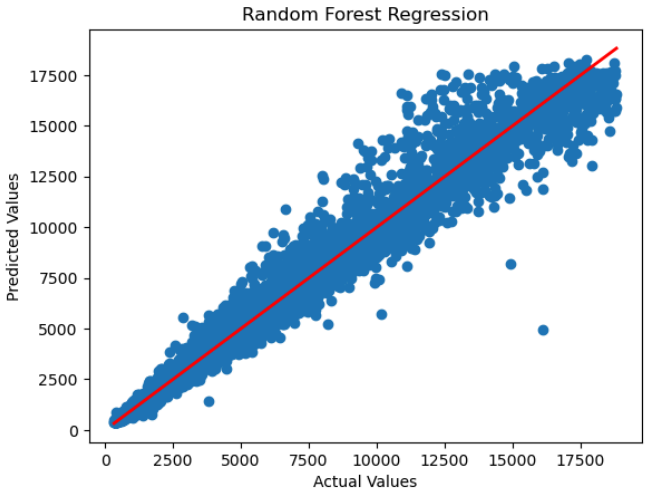
Out[74]:

	carat	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.040441	0.044037	0.095122	554.0	-413.330587
47338	0.161538	0.308824	0.333945	0.309756	1850.0	2823.277549
42573	0.115385	0.277574	0.280734	0.214634	1332.0	2594.717789
32163	0.042308	0.110294	0.124771	0.156098	789.0	649.409633
40040	0.046154	0.137868	0.141284	0.160976	1114.0	1015.610118
...	...	...	...	...	...	...
29498	0.080769	0.180147	0.194495	0.226829	705.0	2113.065812
11697	0.053846	0.136029	0.152294	0.170732	596.0	1169.684400
19191	0.534615	0.683824	0.688073	0.634146	7953.0	11013.244519
4195	0.311538	0.487132	0.486239	0.475610	3563.0	5233.884892
53079	0.150000	0.299632	0.311927	0.314634	2625.0	2665.351994

16158 rows × 6 columns

Random Forest Regression

In [75]:
```
plt.scatter(y_test, rfr_predictions)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='r', linewidth=2, label='Perfect Prediction')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Random Forest Regression')
plt.show()
```



In [76]:
```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test)
TestingDataResults['TargetColumn']=y_test
TestingDataResults['Prediction']=rfr_predictions
TestingDataResults
```

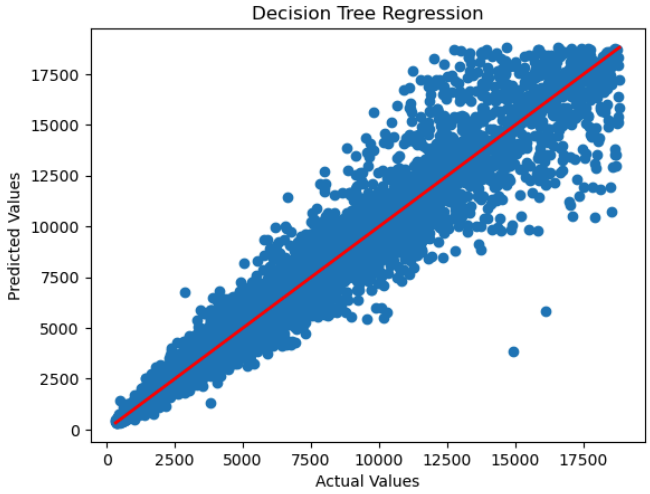
Out[76]:

	carat	cut	color	clarity	depth	table	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.75	0.500000	0.857143	0.515306	0.370370	0.040441	0.044037	0.095122	554.0	534.12
47338	0.161538	0.25	0.500000	0.714286	0.448980	0.481481	0.308824	0.333945	0.309756	1850.0	1879.15
42573	0.115385	0.00	0.666667	0.714286	0.209184	0.592593	0.277574	0.280734	0.214634	1332.0	1284.67
32163	0.042308	0.50	0.500000	0.857143	0.505102	0.222222	0.110294	0.124771	0.156098	789.0	776.80
40040	0.046154	0.50	0.000000	0.571429	0.443878	0.296296	0.137868	0.141284	0.160976	1114.0	966.49
...	...	...	...	...	...	...	...	...	...	...	...
29498	0.080769	0.25	0.333333	0.285714	0.566327	0.333333	0.180147	0.194495	0.226829	705.0	718.64
11697	0.053846	0.50	0.500000	0.714286	0.474490	0.296296	0.136029	0.152294	0.170732	596.0	595.48
19191	0.534615	0.75	1.000000	0.714286	0.530612	0.370370	0.683824	0.688073	0.634146	7953.0	8092.45
4195	0.311538	1.00	0.833333	0.285714	0.556122	0.333333	0.487132	0.486239	0.475610	3563.0	3998.19
53079	0.150000	0.50	0.333333	0.857143	0.520408	0.259259	0.299632	0.311927	0.314634	2625.0	2967.29

16158 rows × 11 columns

Decision Tree Regression

In [77]:
```
plt.scatter(y_test, dtf_predictions)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='r', linewidth=2, label='Perfect Prediction')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Decision Tree Regression')
plt.show()
```



In [78]:
```
LoanData=pd.DataFrame(data=df)
LoanData.head()
TestingDataResults=pd.DataFrame(data=X_test)
TestingDataResults['TargetColumn']=y_test
TestingDataResults['Prediction']=dtf_predictions
TestingDataResults
```

Out[78]:

	carat	cut	color	clarity	depth	table	'x'	'y'	'z'	TargetColumn	Prediction
85	0.015385	0.75	0.500000	0.857143	0.515306	0.370370	0.040441	0.044037	0.095122	554.0	680.0
47338	0.161538	0.25	0.500000	0.714286	0.448980	0.481481	0.308824	0.333945	0.309756	1850.0	1821.0
42573	0.115385	0.00	0.666667	0.714286	0.209184	0.592593	0.277574	0.280734	0.214634	1332.0	1239.0
32163	0.042308	0.50	0.500000	0.857143	0.505102	0.222222	0.110294	0.124771	0.156098	789.0	789.0
40040	0.046154	0.50	0.000000	0.571429	0.443878	0.296296	0.137868	0.141284	0.160976	1114.0	972.0
...	...	...	...	...	...	...	...	...	...	...	...
29498	0.080769	0.25	0.333333	0.285714	0.566327	0.333333	0.180147	0.194495	0.226829	705.0	683.0
11697	0.053846	0.50	0.500000	0.714286	0.474490	0.296296	0.136029	0.152294	0.170732	596.0	596.0
19191	0.534615	0.75	1.000000	0.714286	0.530612	0.370370	0.683824	0.688073	0.634146	7953.0	7811.0
4195	0.311538	1.00	0.833333	0.285714	0.556122	0.333333	0.487132	0.486239	0.475610	3563.0	4242.0
53079	0.150000	0.50	0.333333	0.857143	0.520408	0.259259	0.299632	0.311927	0.314634	2625.0	3090.0

16158 rows × 11 columns

In [80]:
```
X_u = df_scaled.drop('price', axis=1)

y_u = df['price']


from sklearn.model_selection import train_test_split
X_train_u, X_test_u, y_train_u, y_test_u = train_test_split(X_u, y_u, test_size=0.3, random_state=1)

LL_svr = SVR().fit(X_train_u, y_train_u)
rfr.fit(X_train_u, y_train_u)
dtf.fit(X_train_u, y_train_u)

predSVM_u = LL_svr.predict(X_train_u)
rfr_predictions_u = rfr.predict(X_train_u)
dtf_predictions_u = dtf.predict(X_train_u)
```

In [81]:
```
import pandas as pd

data = {
    'carat': (input('Enter the carat: ')),
    'cut': (input('Enter the cut: ')),
    'color': (input('Enter the color: ')),
    'clarity': (input('Enter the clarity: ')),
    'depth': (input('Enter the depth: ')),
    'table': (input('Enter the table: ')),
    'x': (input('Enter the x: ')),
    'y': (input('Enter the y: ')),
    'z': (input('Enter the z: ')),
    }

index = [1]

test_data = pd.DataFrame(data, index=index)

# test_data_s = scaler.fit_transform(test_data)

LL_svr = SVR().fit(X_train_u, y_train_u)
rfr.fit(X_train_u, y_train_u)
dtf.fit(X_train_u, y_train_u)

X_train_u = scaler.fit_transform(X_train_u)

predSVM_u = LL_svr.predict(X_train_u)
predSVM_a_u = LL_svr.predict(X_train_u)
rfr_predictions_u = rfr.predict(X_train_u)
dtf_predictions_u = dtf.predict(X_train_u)

predSVM_u = LL_svr.predict(test_data)
predSVM_a_u = LL_svr.predict(test_data)
rfr_predictions_u = rfr.predict(test_data)
dtf_predictions_u = dtf.predict(test_data)

TestingDataResults=pd.DataFrame(data=test_data)
TestingDataResults['Prediction for SVM']=predSVM_u
TestingDataResults['Prediction for Logistic Regression']=rfr_predictions_u
TestingDataResults['Prediction for KNN']=dtf_predictions_u

TestingDataResults.head(1)
```

```
Enter the carat: 0.5
Enter the cut: 3
Enter the color: 3
Enter the clarity: 3
Enter the depth: 30
Enter the table: 30
Enter the x: 3
Enter the y: 3
Enter the z: 3
```

Out[81]:

	carat	cut	color	clarity	depth	table	x	y	z	Prediction for SVM	Prediction for Logistic Regression	Prediction for KNN
1	0.5	3	3	3	30	30	3	3	3	3228.53096	14827.03	14399.0