Problem 1: Linear Regression

Problem Statement

You are hired by a company Gem Stones co ltd, which is a cubic zirconia manufacturer. You are provided with the dataset containing the prices and other attributes of almost 27,000 cubic zirconia (which is an inexpensive diamond alternative with many of the same qualities as a diamond). The company is earning different profits on different prize slots. You have to help the company in predicting the price for the stone on the bases of the details given in the dataset so it can distinguish between higher profitable stones and lower profitable stones so as to have better profit share. Also, provide them with the best 5 attributes that are most important.

1.1. Read the data and do exploratory data analysis. Describe the data briefly. (Check the null values, Data types, shape, EDA). Perform Univariate and Bivariate Analysis.

Let us import the data into a DataFrame and display the top 5 records –

	carat	cut	color	clarity	depth	table	x	y	z	price
1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

Data Dictionary:

Variable Name	Description
Carat	Carat weight of the cubic zirconia.
Cut	Describe the cut quality of the cubic zirconia. Quality is increasing order Fair, Good, Very Good, Premium, Ideal.
Color	Colour of the cubic zirconia.With D being the best and J the worst.
Clarity	cubic zirconia Clarity refers to the absence of the Inclusions and Blemishes. (In order from Best to Worst, FL = flawless, I3= level 3 inclusions) FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3
Depth	The Height of a cubic zirconia, measured from the Culet to the table, divided by its average Girdle Diameter.
Table	The Width of the cubic zirconia's Table expressed as a Percentage of its Average Diameter.
Price	the Price of the cubic zirconia.
X	Length of the cubic zirconia in mm.
Y	Width of the cubic zirconia in mm.
Z	Height of the cubic zirconia in mm.

Now let us check the shape and datatypes of the variables –

```
Shape - (26967, 10)


dtypes
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 26967 entries, 1 to 26967
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    26967 non-null  float64
 1   cut      26967 non-null  object 
 2   color    26967 non-null  object 
 3   clarity  26967 non-null  object 
 4   depth    26270 non-null  float64
 5   table    26967 non-null  float64
 6   x        26967 non-null  float64
 7   y        26967 non-null  float64
 8   z        26967 non-null  float64
 9   price    26967 non-null  int64  
dtypes: float64(6), int64(1), object(3)
memory usage: 2.3+ MB
```

The data consists of 10 columns (9 independent and 1 dependent variable). There are 3 categorical columns in the dataset. In total, there are 26967 observations, with some Null values in 'depth' column.

Let us describe the dataset to get a statistical description of the data –

	carat	depth	table	x	y	z	price
count	26967.00	26270.00	26967.00	26967.00	26967.00	26967.00	26967.00
mean	0.80	61.75	57.46	5.73	5.73	3.54	3939.52
std	0.48	1.41	2.23	1.13	1.17	0.72	4024.86
min	0.20	50.80	49.00	0.00	0.00	0.00	326.00
25%	0.40	61.00	56.00	4.71	4.71	2.90	945.00
50%	0.70	61.80	57.00	5.69	5.71	3.52	2375.00
75%	1.05	62.50	59.00	6.55	6.54	4.04	5360.00
max	4.50	73.60	79.00	10.23	58.90	31.80	18818.00

One specific point to note is the 0.00 values for 'x', 'y' and 'z' columns. This shouldn't be the case as Length, Width and Height can't be zero. We will see further in our analysis how we can handle these values.

Next let us also check for Null and Duplicate values –

```
carat        0
cut          0
color        0
clarity      0
depth      697
table        0
x            0
y            0
z            0
price        0
dtype: int64
```

```
dupes = df.duplicated()
print(sum(dupes))
```

```
34
```

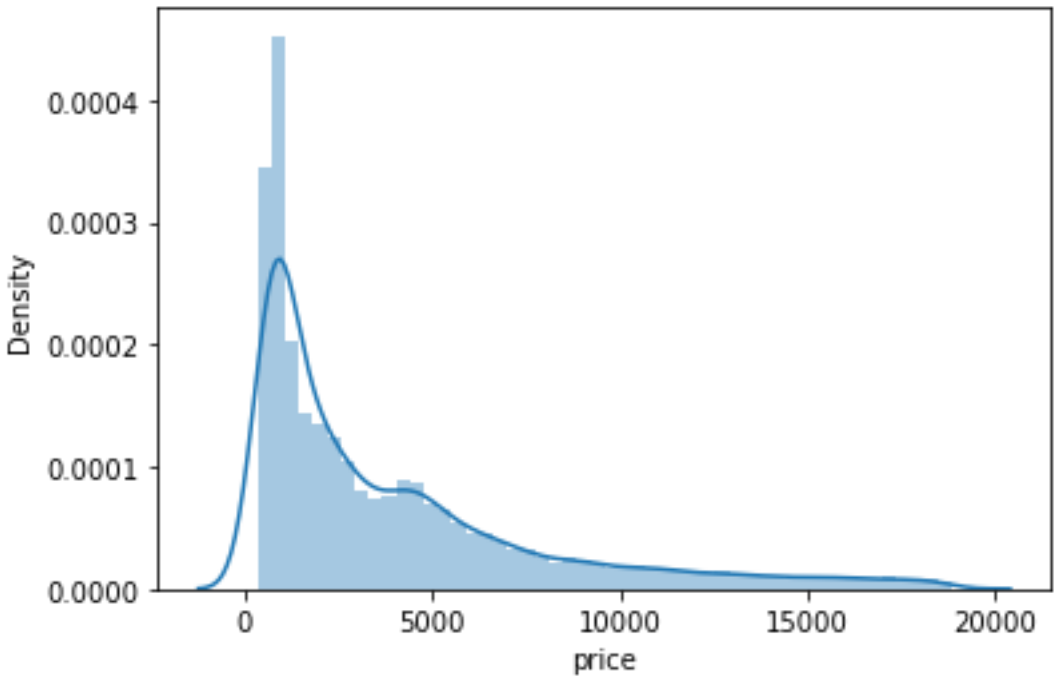
There are 697 null values inside 'depth' column and 34 duplicated records. We will go ahead and drop the duplicated records.

Now let's extract the numerical and categorical columns and save in 2 separate lists as it'll be easier for us to use them for further transformations.

```
#list of discrete variables
num_vars = [var for var in df.columns if df[var].dtypes != 'O']

#list of discrete variables
discrete_vars = [var for var in df.columns if len(df[var].unique())<20]

Number of discrete variables:  3
cut        5
color      7
clarity    8
dtype: int64
**************


cut
Ideal        10805
Premium       6886
Very Good     6027
Good          2435
Fair           780
Name: cut, dtype: int64

color
G    5653
E    4916
F    4723
H    4095
D    3341
I    2765
J    1440
Name: color, dtype: int64

clarity
SI1     6565
VS2     6093
SI2     4564
VS1     4087
VVS2    2530
VVS1    1839
IF       891
I1       364
Name: clarity, dtype: int64
```

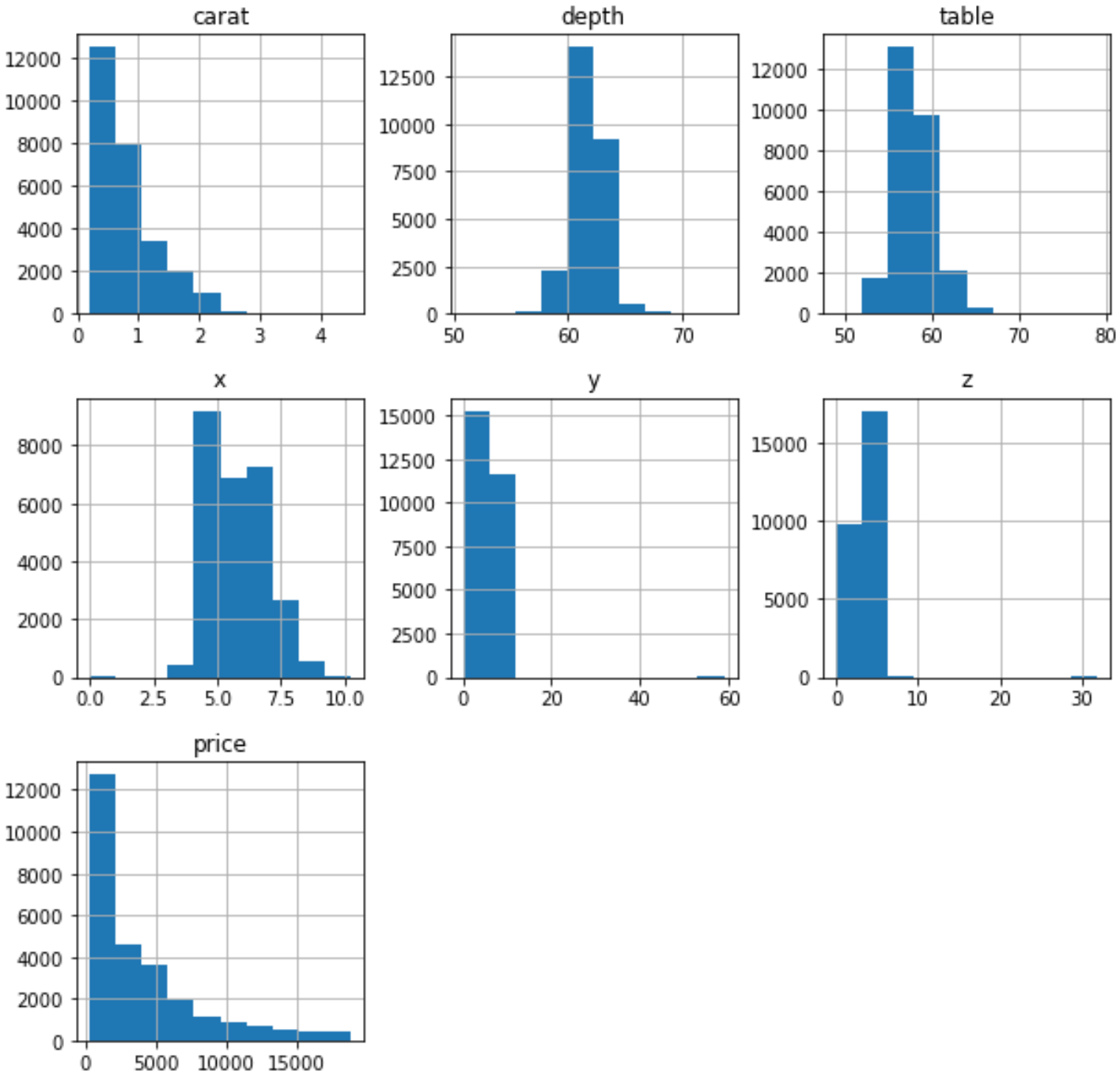
UNIVARIATE ANALYSIS –

Next, let us inspect the distribution of the target variable ‘price’ –



The distribution is right-skewed with most of the datapoints between 0-5000 range.

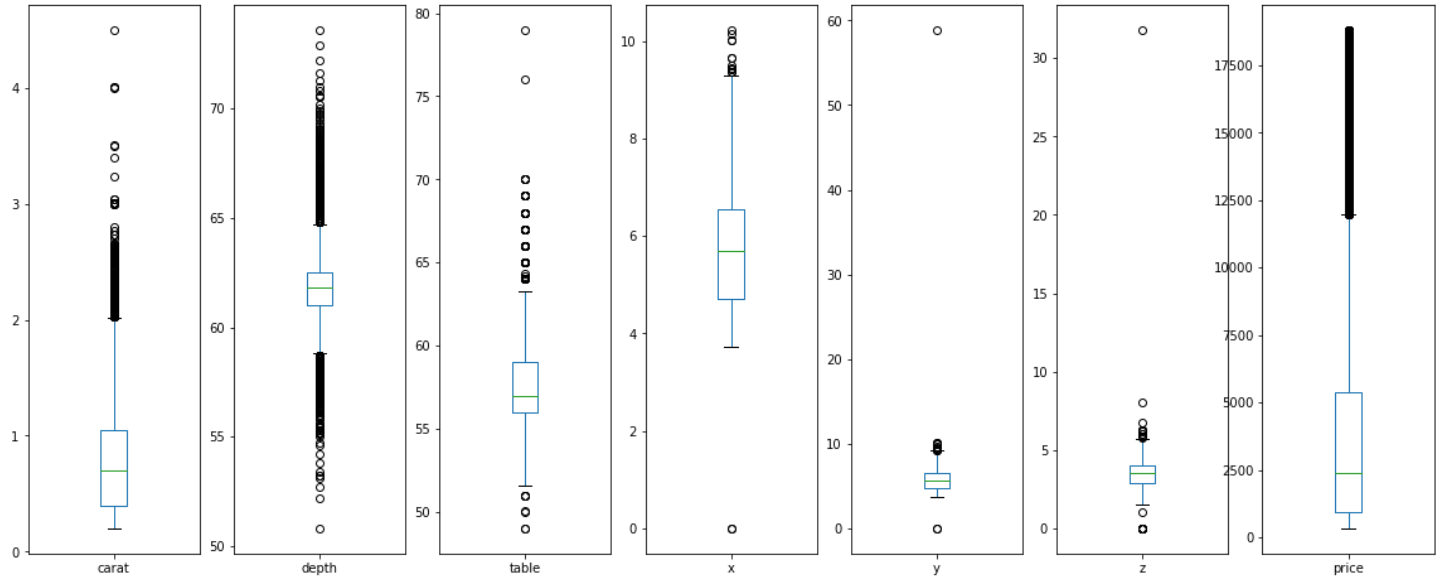
Let us also check the distribution of all numerical variables –



Observations –

'Carat' variable shows right-skewed distribution, so as 'y' and 'z' but they seem to have few outliers due to which the distribution is skewed. 'depth' shows almost normal distribution. 'table' and 'x' columns also show slight normal trend but are slightly right and left skewed respectively.

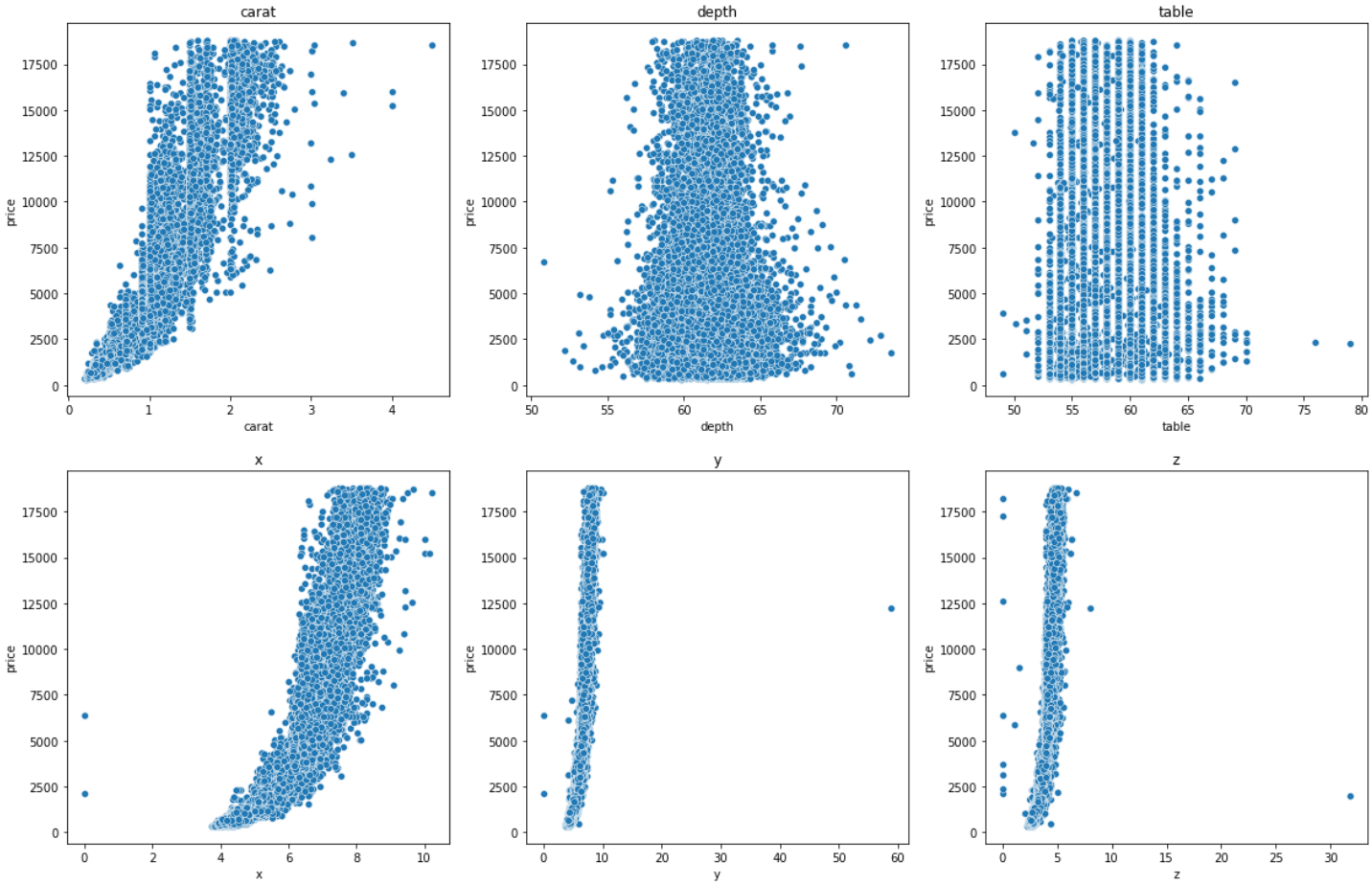
To add further clarity, lets plot boxplots for each of these variables to check the presence of outliers –



All the above variables show the presence of outliers.

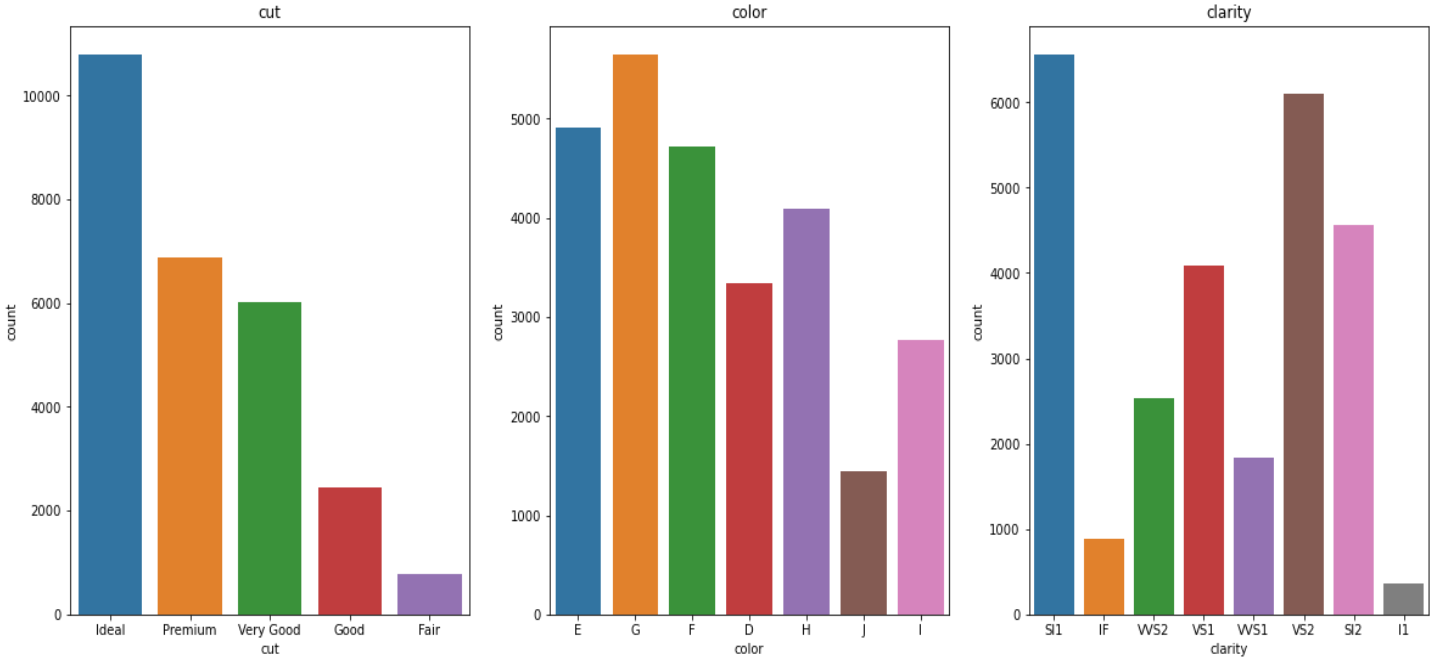
BIVARIATE ANALYSIS –

Let us now plot independent variables with the dependent to check the relationship between the data. We will use scatterplot for doing this –

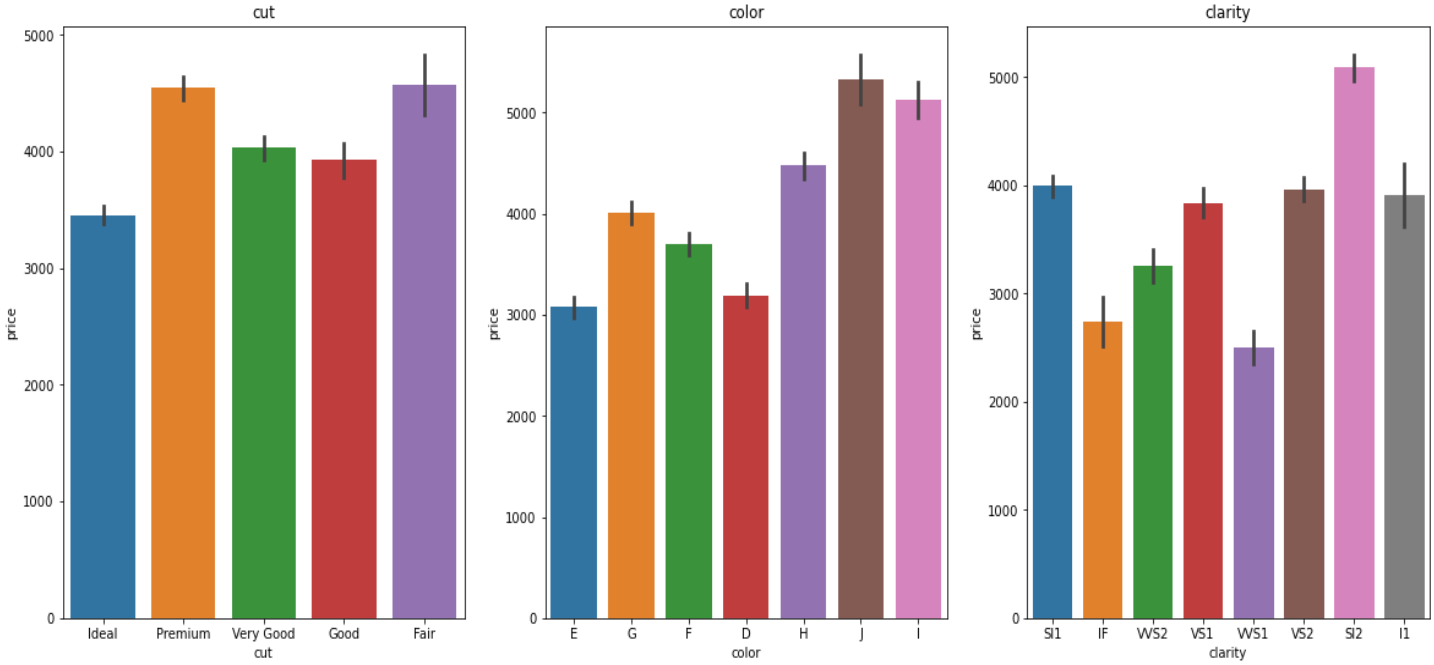


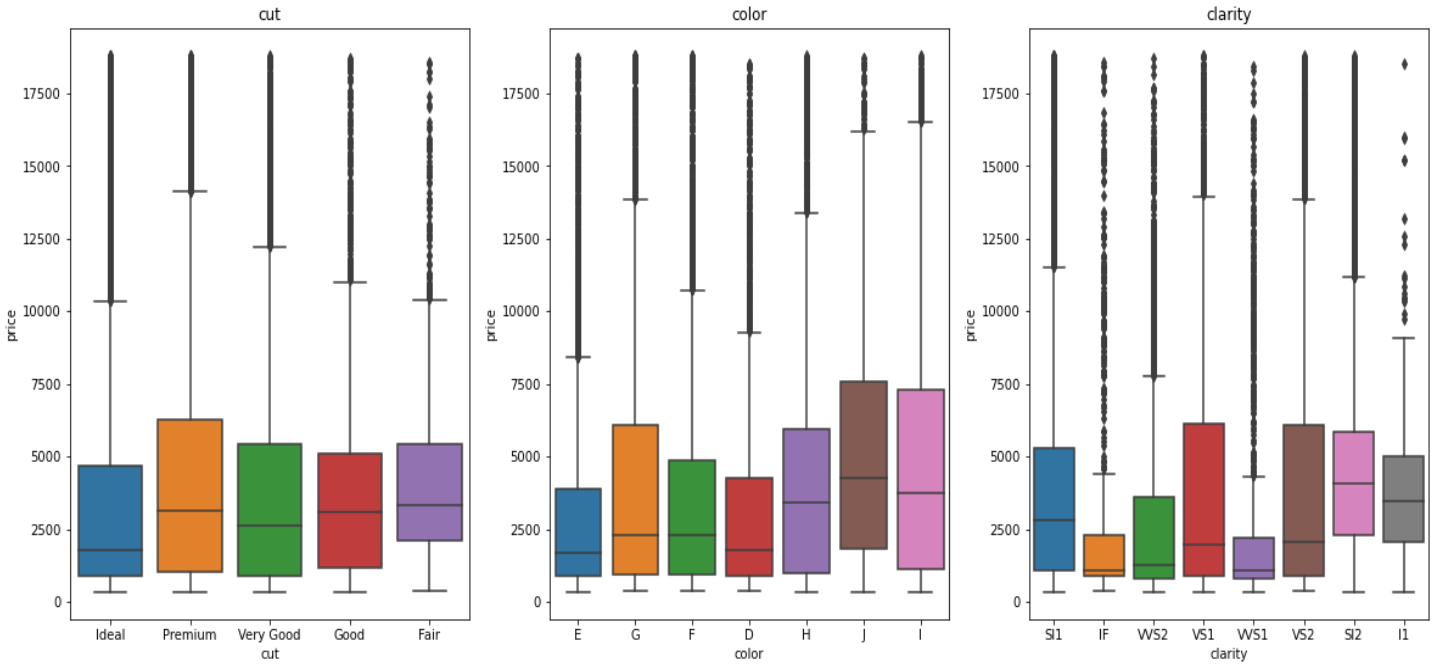
The above plot shows some relationship positive relationship between carat and price. Also, on the positive side for x, y and z with price.

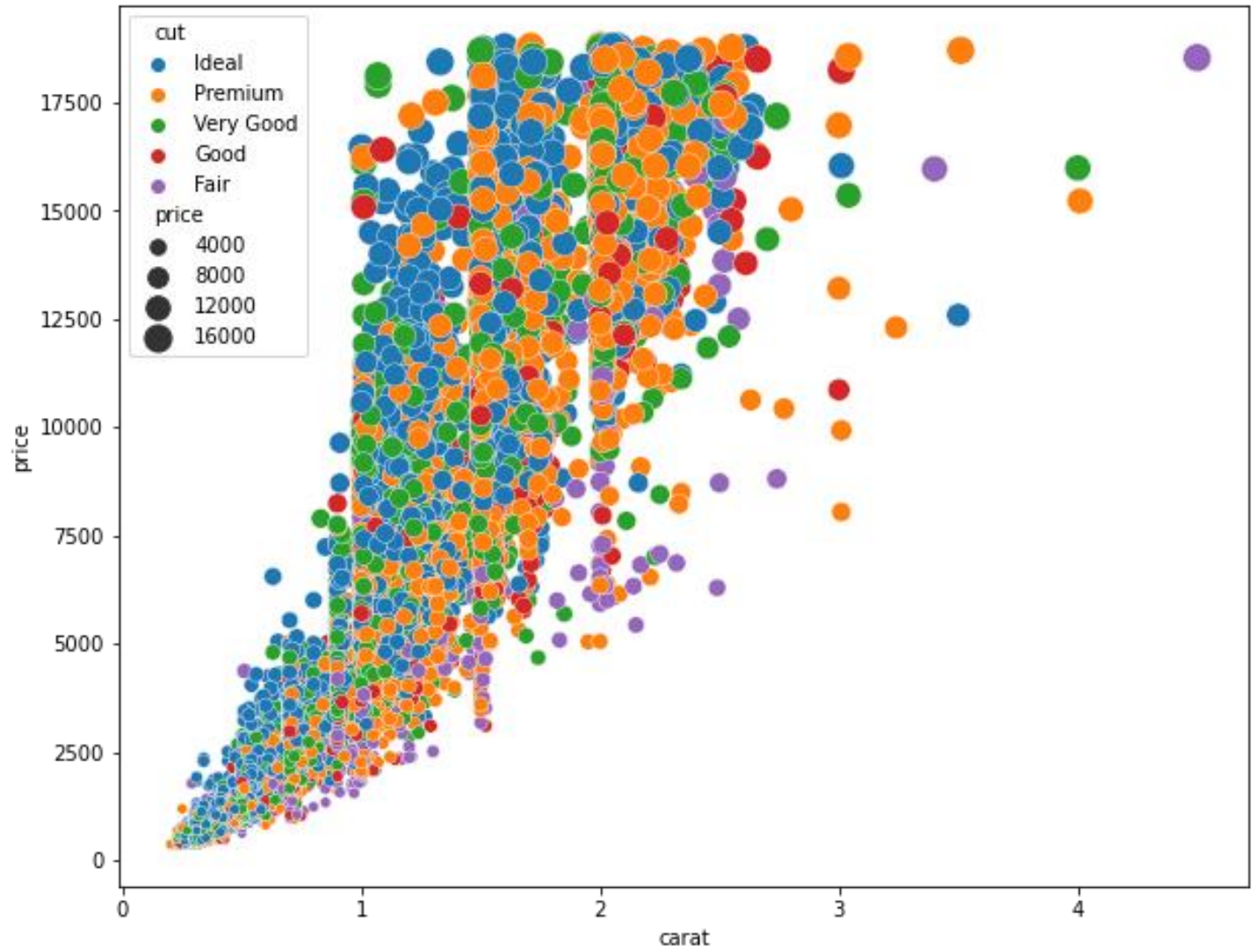
Countplots for the categorical variables –

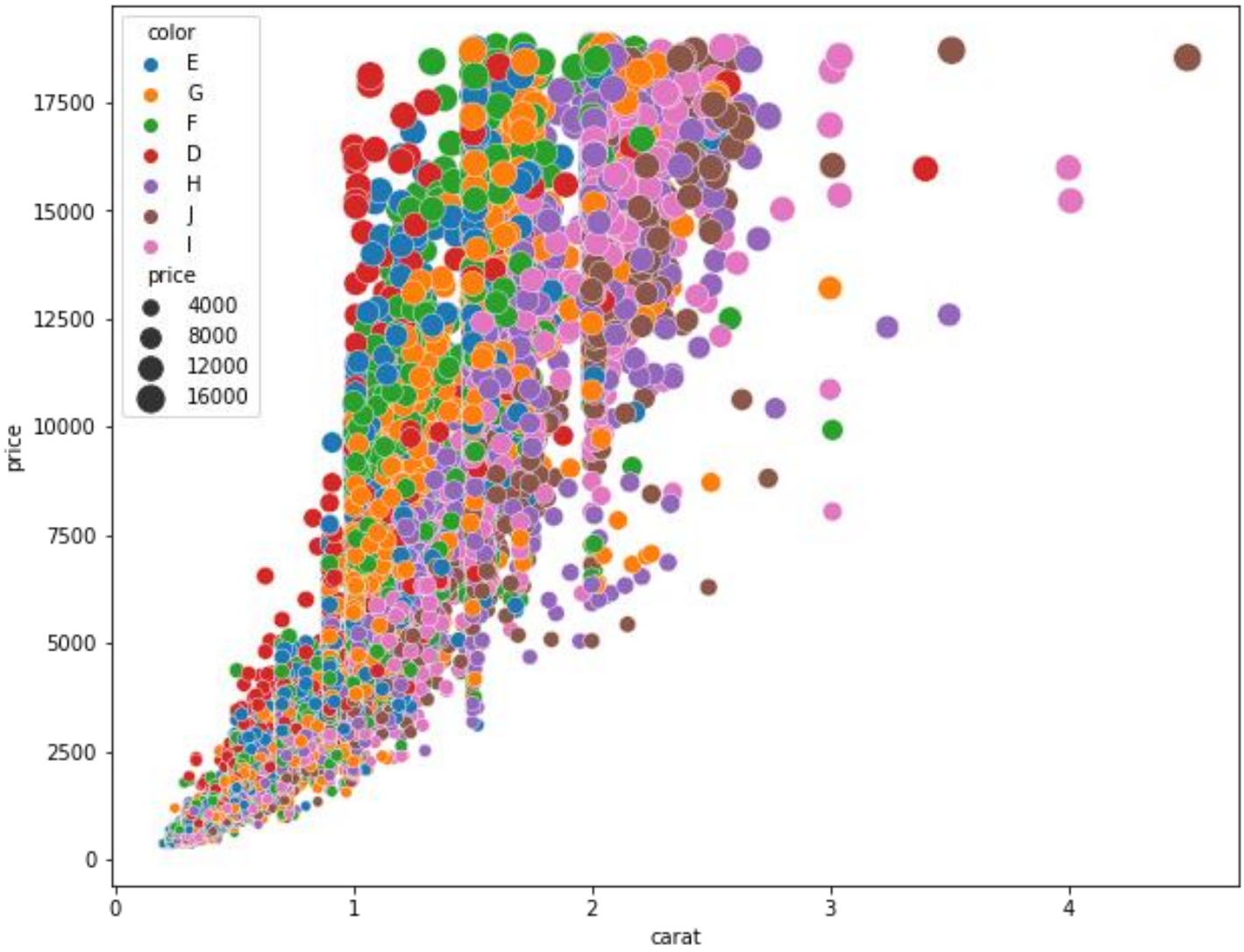


Categorical variables with price –




cut
color
clarity
price
Ideal
Premium
Very Good
Good
Fair
E
G
F
D
H
J
I
SI1
IF
VVS2
VS1
VVS1
VS2
SI2
I1


cut
Ideal
Premium
Very Good
Good
Fair
price
4000
8000
12000
16000
price
carat



Observations –

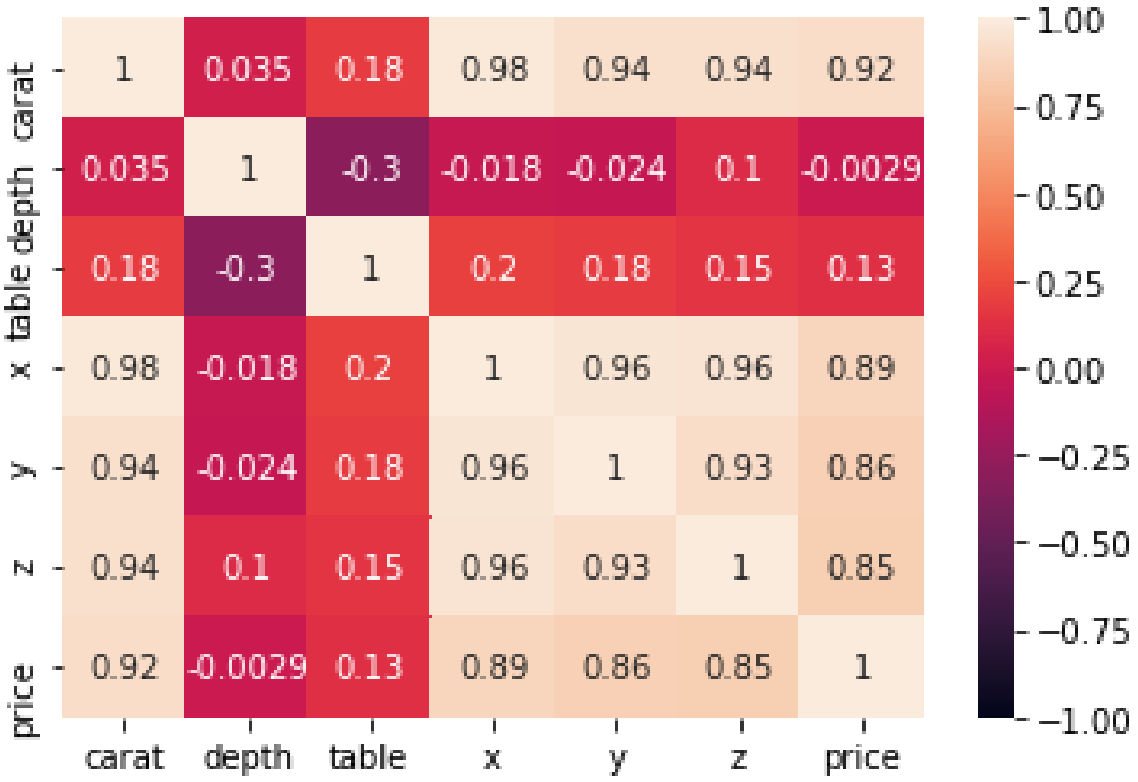
From the plots, we can observe that Ideal, followed by Premium and Very Good are the most preferred cuts by the customers. This could possibly be because of lower price range of Ideal cut.

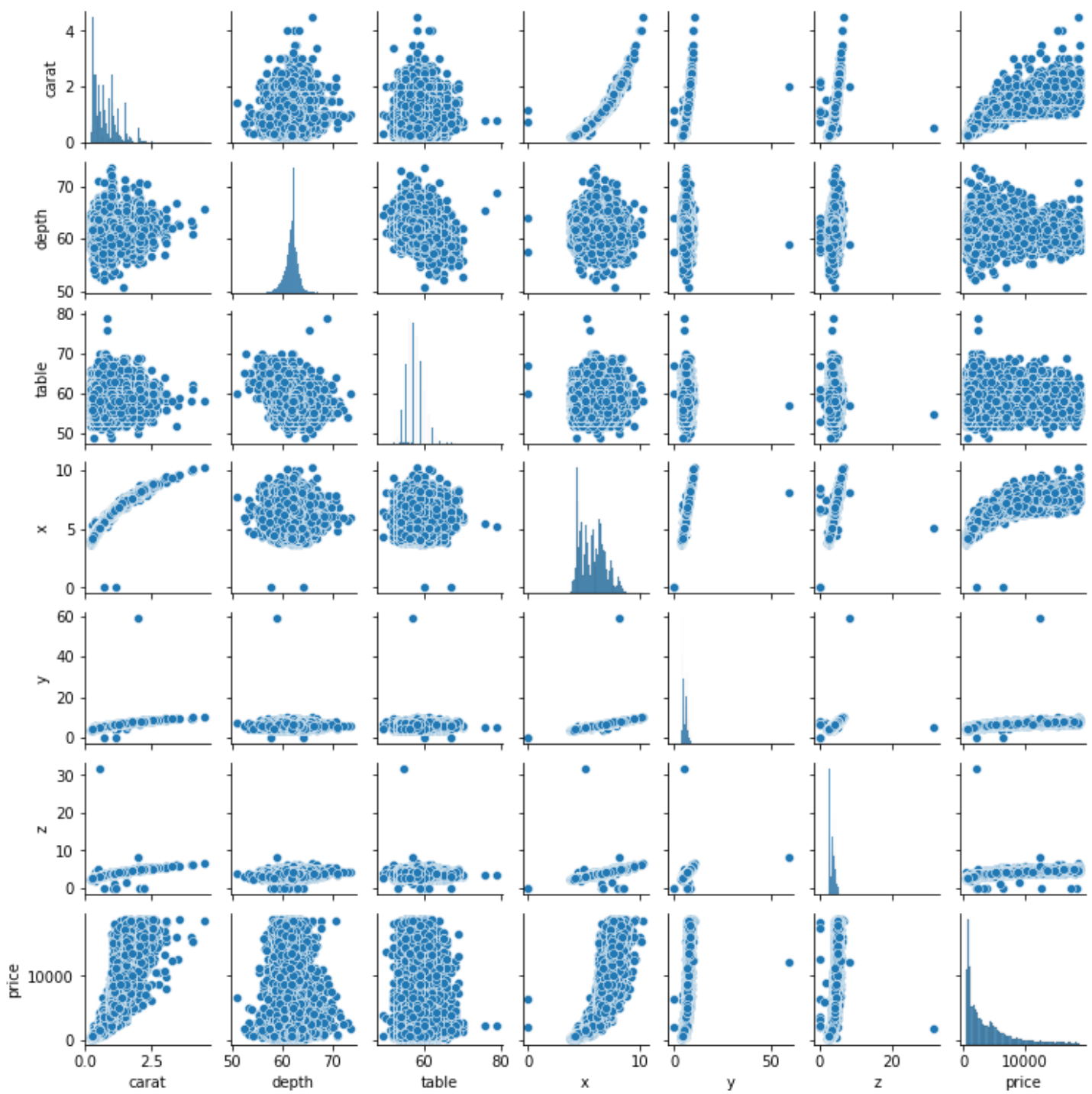
Similarly, G, E, F are the most preferred colors. However, H, J and I seem to be bringing more profits.

S1I and VS2 are the most preferred clarity in diamonds.

Diamonds with I1 clarity seem to be least preferred.

CORRELATION BETWEEN THE VARIABLES USING PAIRPLOT AND HEATMAP –


carat
depth
table
x
y
z
price
1
0.035
0.18
0.98
0.94
0.94
0.92
0.035
1
-0.3
-0.018
-0.024
0.1
-0.0029
0.18
-0.3
1
0.2
0.18
0.15
0.13
0.98
-0.018
0.2
1
0.96
0.96
0.89
0.94
-0.024
0.18
0.96
1
0.93
0.86
0.94
0.1
0.15
0.96
0.93
1
0.85
0.92
-0.0029
0.13
0.89
0.86
0.85
1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



Observations –

1. Carat and price have strong positive correlation, which makes sense as the price of the diamond would normally increase with carat size.
2. There is also strong positive correlation of price with x, y, and z variables. These variables are also strongly correlated with each other which indicates the presence of multi-collinearity.
3. There is no correlation between price and table, also price and depth are negatively correlated.

1.2 Impute null values if present, also check for the values which are equal to zero. Do they have any meaning, or do we need to change them or drop them? Do you think scaling is necessary in this case?

Missing data Imputation

Since 'depth' is normally distributed, we will go ahead and replace the null values in the column with its mean value.

```
df['depth'] = df['depth'].fillna(mean_depth)
```

We will also treat the zero values from x, y and z columns with their respective median values. Before doing that, let us check the count of zero values –

```
Column x has 2 rows with values equal to 0
Column y has 2 rows with values equal to 0
Column z has 8 rows with values equal to 0
```

```
df['x'].replace(0.0, x_median, inplace=True)
df['y'].replace(0.0, y_median, inplace=True)
df['z'].replace(0.0, z_median, inplace=True)
```

Checking the dataset again,

```
Column x has 0 rows with values equal to 0
Column y has 0 rows with values equal to 0
Column z has 0 rows with values equal to 0
```

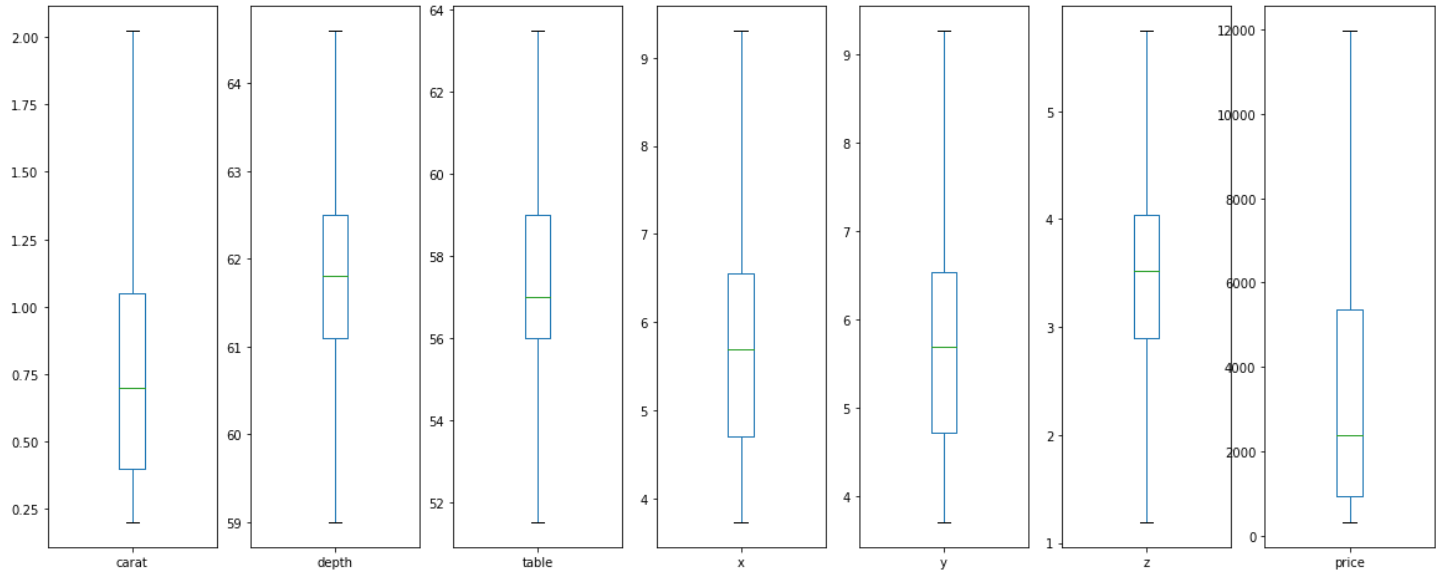
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
x          0
y          0
z          0
price      0
dtype: int64
```

All clean and clear!

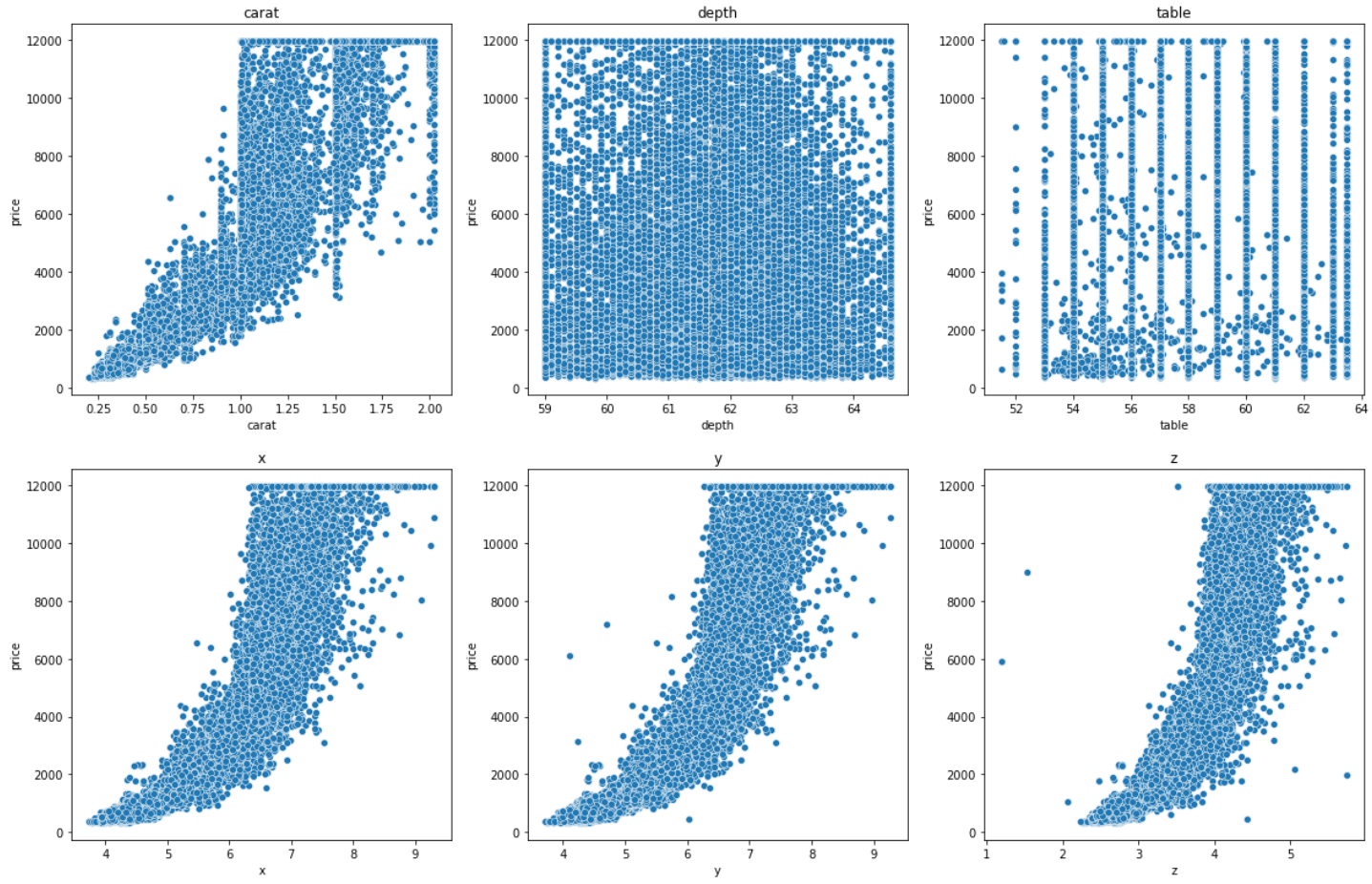
Let us also handle the outliers, however, we will try to fit the model with and without outliers to gauge the impact –

```
def find_skewed_boundaries(df, variable):
    #Let's calculate the boundary ranges for skewed distributions
    IQR = df[variable].quantile(0.75) - df[variable].quantile(0.25)

    lower_boundary = df[variable].quantile(0.25) - (IQR * 1.5)
    upper_boundary = df[variable].quantile(0.75) + (IQR * 1.5)

    return upper_boundary, lower_boundary
```

Viewing the data after treating the outliers –



Let's view the scatterplots now for better understanding –



Scaling

	carat	depth	table	x	y	z	price
count	26933.00	26933.00	26933.00	26933.00	26933.00	26933.00	26933.00
mean	0.79	61.75	57.44	5.73	5.73	3.54	3735.83
std	0.46	1.22	2.16	1.13	1.12	0.70	3468.21
min	0.20	59.00	51.50	3.73	3.71	1.19	326.00
25%	0.40	61.10	56.00	4.71	4.72	2.90	945.00
50%	0.70	61.80	57.00	5.69	5.70	3.52	2375.00
75%	1.05	62.50	59.00	6.55	6.54	4.04	5356.00
max	2.03	64.60	63.50	9.31	9.27	5.75	11972.50

I think Scaling might not be necessary here as mean and median values are more or less the same for all the independent variables. However, we will statistically confirm that by fitting the model on scaled data as well.

1.3 Encode the data (having string values) for Modelling. Data Split: Split the data into train and test (70:30). Apply Linear regression. Performance Metrics: Check the performance of Predictions on Train and Test sets using Rsquare, RMSE.

We will use LabelEncoder technique here to encode the data as the discrete variables (cut, color, clarity) seem to have an inherent ordering.

```
for var in discrete_vars:
    df_encoded[var] = label_enc.fit_transform(df_encoded[var])
    for k, v in enumerate(label_enc.classes_):
        print(var, v,'==>', k)
```

```
cut Fair ==> 0
cut Good ==> 1
cut Ideal ==> 2
cut Premium ==> 3
cut Very Good ==> 4

color D ==> 0
color E ==> 1
color F ==> 2
color G ==> 3
color H ==> 4
color I ==> 5
color J ==> 6

clarity I1 ==> 0
clarity IF ==> 1
clarity SI1 ==> 2
clarity SI2 ==> 3
clarity VS1 ==> 4
clarity VS2 ==> 5
clarity VVS1 ==> 6
clarity VVS2 ==> 7
```

Now let's look at the dtypes –

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26933 entries, 1 to 26967
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    26933 non-null  float64
 1   cut      26933 non-null  int32  
 2   color    26933 non-null  int32  
 3   clarity  26933 non-null  int32  
 4   depth    26933 non-null  float64
 5   table    26933 non-null  float64
 6   x        26933 non-null  float64
 7   y        26933 non-null  float64
 8   z        26933 non-null  float64
 9   price    26933 non-null  float64
dtypes: float64(7), int32(3)
memory usage: 3.2 MB
```

We saw earlier from the Heatmap, there exists some multi-collinearity between the independent variables. So, I would like to check that using VIF (Variance Inflation Factor)

```
def calculate_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    print(X.shape)
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)

X = df_encoded.iloc[:,:-1]
calculate_vif(X)
```

	variables	VIF
0	carat	119.969141
1	cut	7.671209
2	color	3.669549
3	clarity	6.300427
4	depth	971.138472
5	table	719.564779
6	x	10317.298601
7	y	9457.920514
8	z	2621.227844

This confirms our observation, there is high multicollinearity between all variables, esp. x, y, z. Let us scale the data and check again –

```
X_scaled  = df_encoded.apply(zscore)
X = X_scaled.iloc[:,:-1]
calculate_vif(X)
```

	variables	VIF
0	carat	32.675402
1	cut	1.065715
2	color	1.101226
3	clarity	1.068987
4	depth	3.885856
5	table	1.182455
6	x	408.406194
7	y	392.474248
8	z	204.993996

We see that x, y and z have high VIF score, meaning they can probably be predicted by other independent variables in the dataset.

For now, let's proceed and create a Training and Test Data split. Then we will create a Regression Model and try to fit it on DataFrames (original and with Outlier treatment) to see the impact –

```
X = df_original.drop('price', axis=1)
y = df_original.pop('price')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_sta
te=1)

reg_model = LinearRegression()
reg_model.fit(X_train, y_train)
```

Let's look at the co-efficients –

	Coeff
carat	11468.882055
cut	59.412933
color	-282.707638
clarity	290.151385
depth	-156.516511
table	-92.217010
x	-1311.995229
y	3.791608
z	-35.546090

Predict y_train and y_test –

```
ytrain_pred = reg_model.predict(X_train)
ytest_pred = reg_model.predict(X_test)
```

Check RMSE and R2 scores of the model –

```
def get_model_scores(y, y_pred):
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    r2 = r2_score(y, y_pred)
    return rmse, r2
```


```
def get_model_scores(y, y_pred):
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    r2 = r2_score(y, y_pred)
    return rmse, r2
```


```
Linear Regression Training set RMSE: 1348.51
Linear Regression Training set R2 SCORE: 0.887
Linear Regression Test set RMSE: 1347.04
Linear Regression Test set R2 SCORE: 0.89
```

Now we will fit the model on a DataFrame with outliers treated. Applying the same steps as above, we get the below co-efficients –

	Coeff
carat	9267.945271
cut	49.471178
color	-233.397325
clarity	253.365894
depth	-50.686730
table	-73.297697
x	-1859.985309
y	1727.080748
z	-915.165019

Checking RMSE and R2 scores now,

```
Linear Regression Training set RMSE: 1046.246
Linear Regression Training set R2 SCORE: 0.908
Linear Regression Test set RMSE: 1030.189
Linear Regression Test set R2 SCORE: 0.912
```

There is definitely an improvement in RMSE and R2 scores after outlier treatment. So we will continue working on this DataFrame and see if the score can be further optimized.

Next let us scale the data and check RMSE and R2 score.


```
Linear Regression Training set RMSE: 1046.246
Linear Regression Training set R2 SCORE: 0.908
Linear Regression Test set RMSE: 1030.189
Linear Regression Test set R2 SCORE: 0.912
```


After rounding off, the scores are exactly the same, so it affirms our earlier observation that scaling might not be necessary for this dataset.

As a next step, we can try to create a regularized model using **RIDGE** or **LASSO** to see if it helps improve the model performance with lesser features.

```
ridge = Ridge(alpha=0.001, normalize=True)
ridge.fit(X_train, y_train)
#print ("Ridge model:", (ridge.coef_))

pd.DataFrame(ridge.coef_.reshape(9,1),
             index=X_train.columns, columns=['Coeff']
             )
```

	Coeff
carat	8930.373705
cut	55.065215
color	-231.293016
clarity	255.385738
depth	-57.353112
table	-75.591081
x	-1067.260044
y	964.721816
z	-750.383509

Get model scores –

```
Training RMSE: 1047.523
Training R2 SCORE: 0.908
Test RMSE: 1030.928
Test R2 SCORE: 0.912
```

```
lasso = Lasso(alpha=0.08, normalize=True)
lasso.fit(X_train,y_train)
#print ("Lasso model:", (lasso.coef_))

pd.DataFrame(lasso.coef_.reshape(9,1),
             index=X_train.columns, columns=['Coeff']
             )
```

	Coeff
carat	7790.065083
cut	52.681578
color	-219.642391
clarity	255.705598
depth	-65.249764
table	-70.064138
x	-0.000000
y	-0.000000
z	-177.977702

Get model scores –

```
Training RMSE: 1058.091
Training R2 SCORE: 0.906
Test RMSE: 1040.798
Test R2 SCORE: 0.91
```

LASSO gives a very negligible decline in the scores but helps eliminate 2 features.

1.4 Inference: Basis on these predictions, what are the business insights and recommendations.

From the EDA performed, we can observe that Diamonds with 'Ideal', 'Premium' and 'Very Good' cuts have the most sales and give good amount of profit. Price is high for 'Fair' cut, however sales are pretty low.

'H', 'J' and 'I' colors seem maximum profits so the company can target some marketing campaigns to boost the sales for these colors.

Similarly for diamond clarity, I1 seems to be the least preferred one; SI1 and VS2 are the most preferred ones. The data has no diamonds with FL (Flawless) clarity which probably indicates that FL diamonds might not be returning good profits to the company or might not have many buyers.

We also saw that outlier treatment has a positive impact on model accuracy, so I would recommend doing that. However, scaling has no impact on the score so we can choose to skip doing that in Production.

Our model can predict the diamonds price with 90% accuracy, we have similar score for both train and test datasets, which means, the model is neither overfit nor underfit.

With Lasso regularization, we were able to eliminate 2 features and get almost the same R2 scores for training and test dataset. This might be helpful as it would help increasing the model efficiency/speed in production.

Overall, my recommendations would be to go with Lasso model and target marketing campaigns for –

- The Ideal, Premium and Very Good cuts to increase profits
- H, J and I colors to boost sales
- S12 and VS2 clarity to boost sales

From the co-efficients, we can say that 'Carat', 'Clarity' and 'Cut' are the most important attributes to predict the diamonds price.