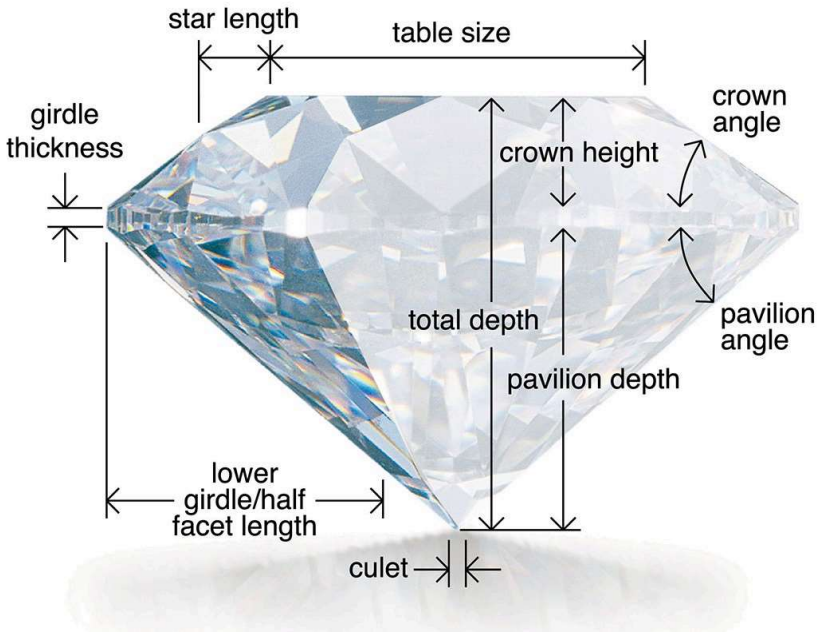
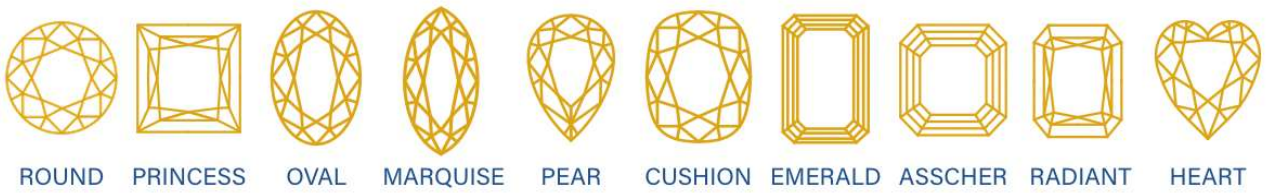
```
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_palette('plasma', n_colors=None, desat=None, color_codes=False)
pc = ['#FFB6C1', '#FFA07A', '#FFD700', '#98FB98', '#DDA0DD', '#87CEEB', '#F0E68C', '#FFE4E1'
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
```

Description of the Diamonds Data set

> **Description of the all Attributes of dataset**



1. **Cut** : It refers to one of the 10 or so most common diamond cuts. This dataset has an additional one called the 'Cushion Modified'.

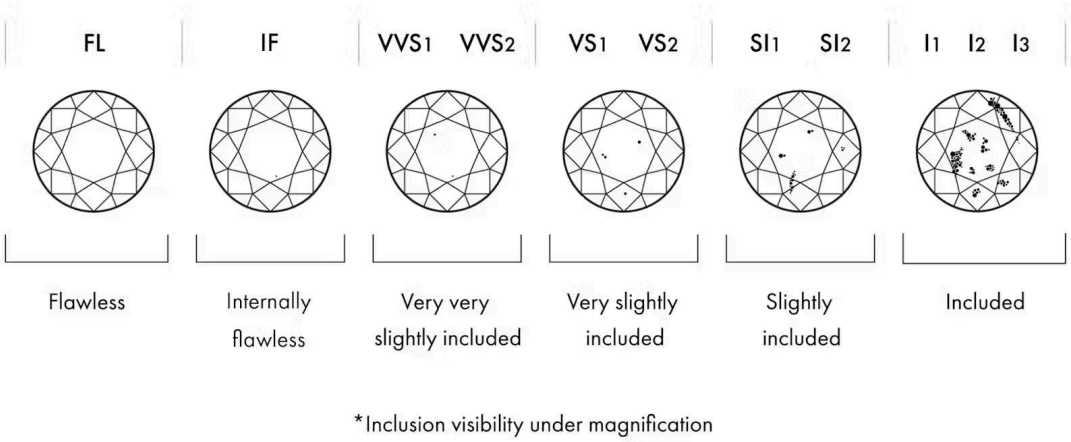


2. **Color:** Indicates the color grade or hue of the gemstone, typically ranging from D (colorless) to Z (light yellow or brown) in the GIA color grading scale for diamonds.

3. **Clarity:** Represents the level of clarity or purity of the gemstone, often graded from Flawless (no inclusions or blemishes) to Included (inclusions or blemishes visible to the naked eye) using GIA's clarity scale.

```
It has following values...
FL - Flawless
IF - Internally Flawless
VVS1/VVS2 - Very, Very Slightly Included
VS1/VS2 - Very Slightly Included
```

```
SI1/SI2 - Slightly Included
I1/I2/I3 - Included
```



4. **Carat Weight:** Denotes the weight of the gemstone in carats, a standard unit of measurement for gemstones.
5. **Cut Quality:** A more detailed evaluation of the cut, which might include factors like proportions, angles, and other aspects that affect how the gemstone reflects light.

```
The GIA assigns the cut of a diamond one of the following grades...
Excellent (the best grade)
Very Good
Good
Fair
Poor (the worst grade)
```

6. **Lab:** Indicates the laboratory or entity that has graded or certified the gemstone, such as GIA (Gemological Institute of America) or EGL (European Gemological Laboratory).
7. **Symmetry:** Reflects the symmetry of the gemstone's facets, which can influence its overall appearance.

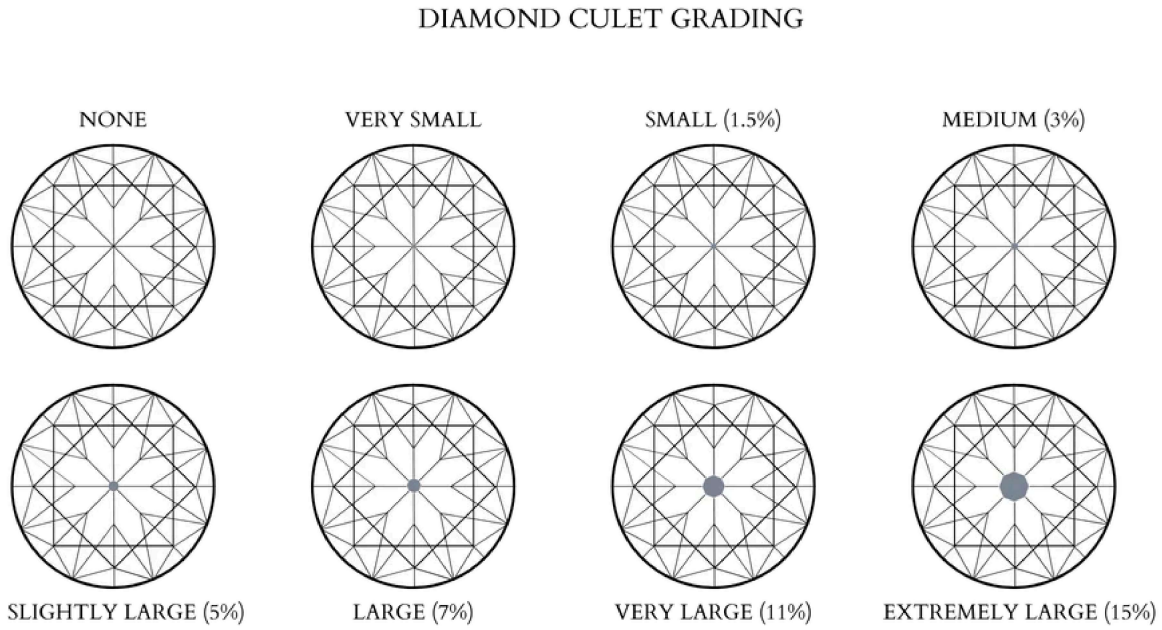
```
It has following values...
Poor, Fair, .... , Very Good
```

8. **Polish:** Refers to the quality of the gemstone's surface finish, which can impact its luster and appearance.
9. **Eye Clean:** Indicates whether the gemstone appears clean to the naked eye, with no visible inclusions or blemishes.

```
It has following values...
Yes, No, Borderline, E1, unknown
```

10. **Culet Size:** Describes the size of the culet, a facet on the bottom of the gemstone.

    The GIA includes several different grades for culet size: None, Very Small, Small, Medium, Slightly Large, Large, Very Large, Extremely Large



11. **Culet Condition:** Specifies the condition of the culet, which could be chipped or damaged etc.

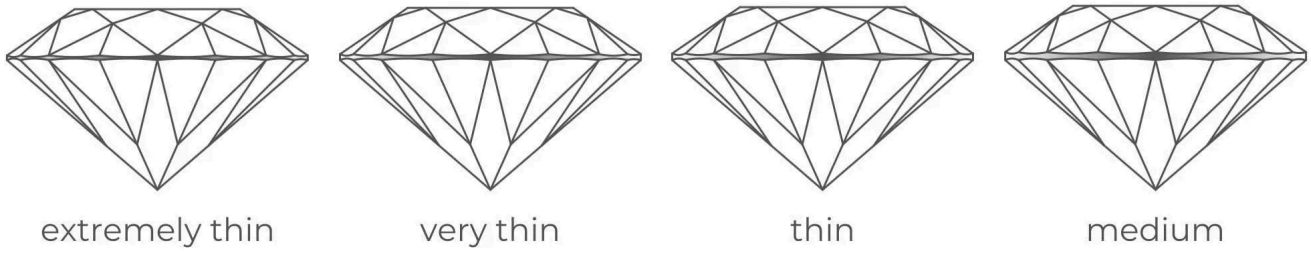
```
Values...
unknown
Abraded
Chipped
Pointed
```

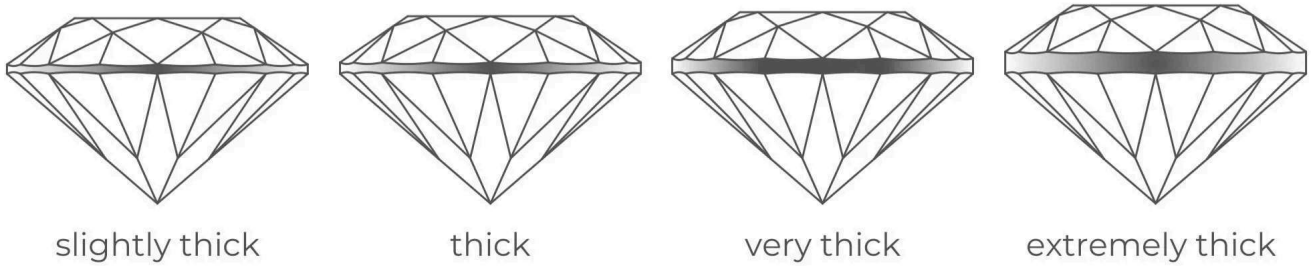
12. **Depth Percent:** Represents the depth of the gemstone as a percentage of its overall dimensions.
13. **Table Percent:** Denotes the size of the flat facet on the top of the gemstone as a percentage of its overall dimensions.

14. **Meas Length, Meas Width, Meas Depth:** The length, width, and depth measurements of the gemstone.
15. **Girdle Min, Girdle Max:** Refers to the minimum and maximum thickness of the girdle, which is the edge where the top and bottom parts of the gemstone meet.

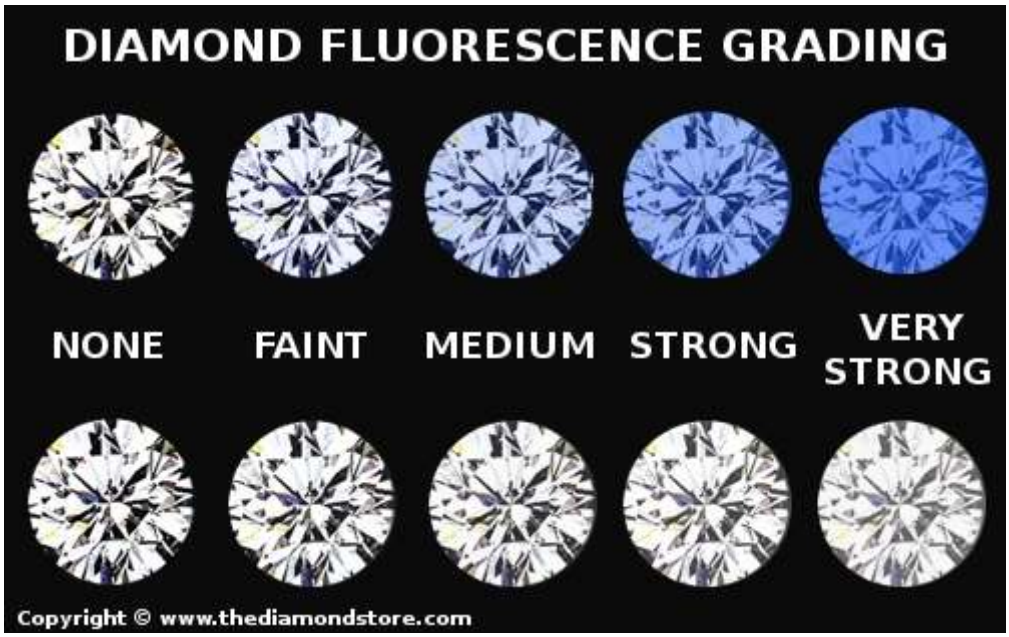
    Girdle thickness is classified into categories ranging from extremely thin to extremely thick.

```
In this dataset it has following values..
M - Medium
STK - Slightly Thick
TN - Thin
TK - Thick
VTN - Very Thin
VTK - Very Thick
XTK - Extremely Thick
XTN - Extremely Thin
```





16. **Fluor Color and Fluor Intensity:** Describes any fluorescence observed in the gemstone under ultraviolet (UV) light,including the color and the intensityfluorscence.



17. **Fancy Color Dominant Color and Secondary Color:** Applicable if the gemstone is a fancy colored diamond, indicating the primary and secondary colors present.
18. **Fancy Color Intensity:** Specifies the intensity or saturation of the fancy color.

```
Values...
Fancy
Very Light
Faint
Fancy Light
Light
Fancy Deep
Fancy Intense
```

Fancy Dark

Fancy Vivid

19. **Total Sales Price:** Represents the total price or value of the gemstone in a given transaction or context

Exploratory Data Analysis (EDA)

```
df = pd.read_csv('diamonds.csv')
df.drop('Unnamed: 0', axis=1,inplace=True) #Removing first column
df.head()
```

	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean
0	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown
1	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Very Good	unknown
2	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown
3	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown
4	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Excellent	unknown

5 rows × 25 columns

```
df.shape #Format of this function : (Numbers of rows,Number of columns)
```

```
(219703, 25)
```

```
df.info() #It Shows the datatype of each attribute and count of null, non null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 219703 entries, 0 to 219702
Data columns (total 25 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   cut                         219703 non-null  object
 1   color                       219703 non-null  object
 2   clarity                     219703 non-null  object
 3   carat_weight                219703 non-null  float64
 4   cut_quality                 219703 non-null  object
 5   lab                         219703 non-null  object
```

```
 6   symmetry                    219703 non-null  object
 7   polish                      219703 non-null  object
 8   eye_clean                   219703 non-null  object
 9   culet_size                  219703 non-null  object
 10  culet_condition             219703 non-null  object
 11  depth_percent               219703 non-null  float64
 12  table_percent               219703 non-null  float64
 13  meas_length                 219703 non-null  float64
 14  meas_width                  219703 non-null  float64
 15  meas_depth                  219703 non-null  float64
 16  girdle_min                  219703 non-null  object
 17  girdle_max                  219703 non-null  object
 18  fluor_color                 219703 non-null  object
 19  fluor_intensity             76212 non-null   object
 20  fancy_color_dominant_color  219703 non-null  object
 21  fancy_color_secondary_color  219703 non-null  object
 22  fancy_color_overtone        218053 non-null  object
 23  fancy_color_intensity       219703 non-null  object
 24  total_sales_price           219703 non-null  int64
dtypes: float64(6), int64(1), object(18)
memory usage: 41.9+ MB
```

```
df.isnull().sum() #Shows count of Null values of each attribute
```

```
cut                           0
color                         0
clarity                       0
carat_weight                  0
cut_quality                   0
lab                           0
symmetry                      0
polish                        0
eye_clean                     0
culet_size                    0
culet_condition               0
depth_percent                 0
table_percent                 0
meas_length                   0
meas_width                    0
meas_depth                    0
girdle_min                    0
girdle_max                    0
fluor_color                   0
fluor_intensity               0
fancy_color_dominant_color    0
fancy_color_secondary_color   0
fancy_color_overtone          0
fancy_color_intensity         0
total_sales_price             0
dtype: int64
```

```
df.describe() #It will show summary of numaric attributes like mean, max, min, standard devi
```

	carat_weight	depth_percent	table_percent	meas_length	meas_width	meas
count	219703.000000	219703.000000	219703.000000	219703.000000	219703.000000	219703.
mean	0.755176	61.683768	57.747585	5.548853	5.135626	3.
std	0.845894	9.915266	9.959928	1.763924	1.374529	2.
min	0.080000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.310000	61.200000	57.000000	4.350000	4.310000	2.
50%	0.500000	62.400000	58.000000	5.060000	4.800000	3.
75%	1.000000	63.500000	60.000000	6.350000	5.700000	3.
max	19.350000	98.700000	94.000000	93.660000	62.300000	76.

```
df.describe(include='O').T #It will show summary of data which is object(string) type
```

	count	unique	top	freq
cut	219703	11	Round	158316
color	219703	11	E	33103
clarity	219703	11	SI1	38627
cut_quality	219703	6	Excellent	124861
lab	219703	3	GIA	200434
symmetry	219703	5	Excellent	131619
polish	219703	5	Excellent	175806
eye_clean	219703	5	unknown	156916
culet_size	219703	9	N	131899
culet_condition	219703	4	unknown	204384
girdle_min	219703	10	unknown	83432
girdle_max	219703	10	unknown	84295
fluor_color	219703	6	unknown	203977
fluor_intensity	76212	7	Faint	38302
fancy_color_dominant_color	219703	13	unknown	210539
fancy_color_secondary_color	219703	11	unknown	218641
fancy_color_overtone	218053	8	unknown	217665
fancy_color_intensity	219703	10	unknown	210541

> Listing Columns which has Object DataType

```
categorical_columns = [feature for feature in df.columns if df[feature].dtypes =='O']
categorical_columns
```

```
['cut',
 'color',
 'clarity',
 'cut_quality',
 'lab',
 'symmetry',
 'polish',
 'eye_clean',
 'culet_size',
 'culet_condition',
 'girdle_min',
 'girdle_max',
 'fluor_color',
 'fluor_intensity',
 'fancy_color_dominant_color',
 'fancy_color_secondary_color',
 'fancy_color_overtone',
 'fancy_color_intensity']
```

> Listing Columns which has Numeric DataType and removing Total_Price from feature

```
numerical_columns=[feature for feature in df.columns if df[feature].dtypes !='O']
numerical_columns.pop(-1)
features=numerical_columns+categorical_columns
features
```

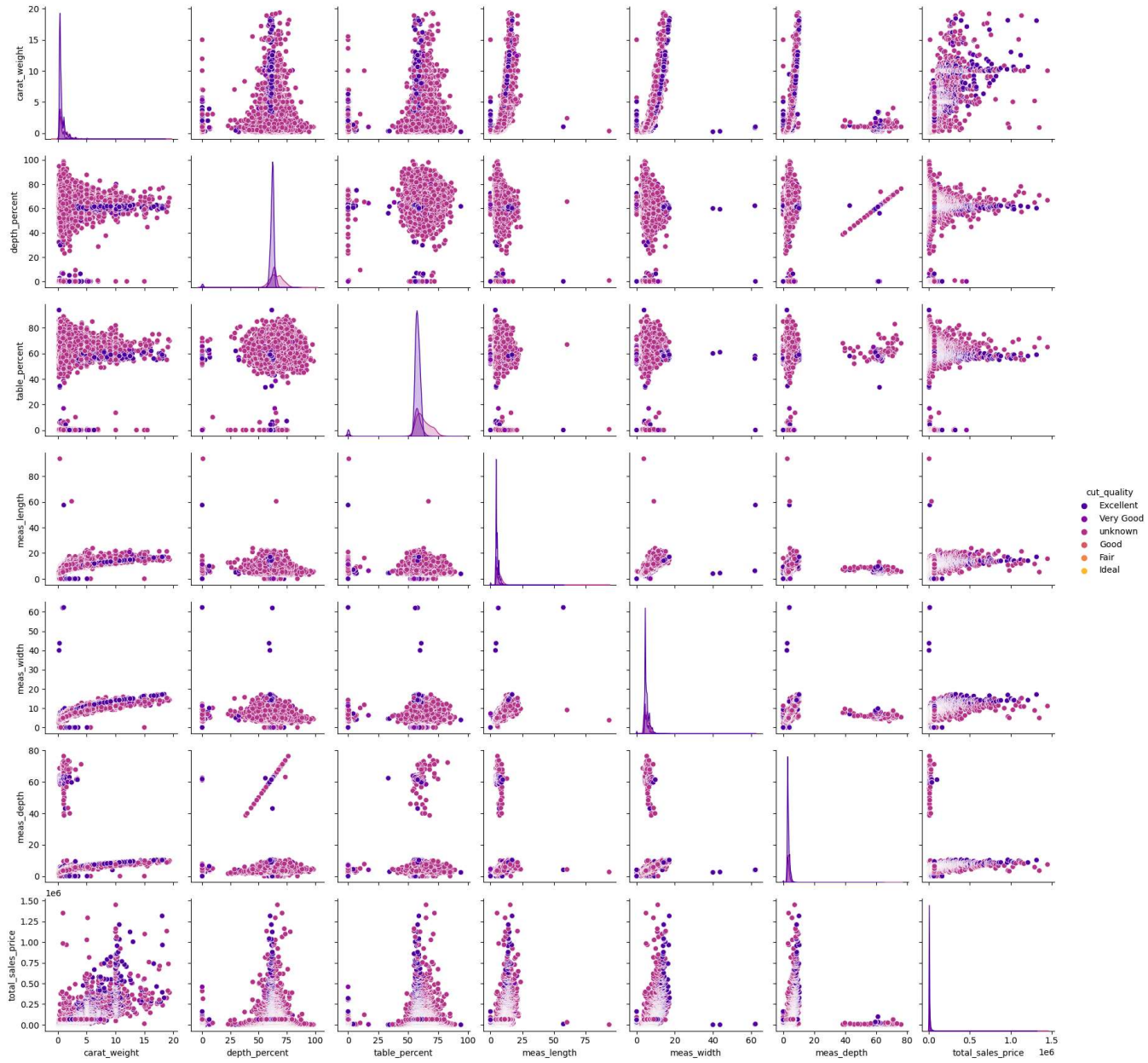
```
['carat_weight',
 'depth_percent',
 'table_percent',
 'meas_length',
 'meas_width',
 'meas_depth',
 'cut',
 'color',
 'clarity',
 'cut_quality',
 'lab',
 'symmetry',
 'polish',
 'eye_clean',
 'culet_size',
 'culet_condition',
 'girdle_min',
 'girdle_max',
 'fluor_color',
```

```
      'fluor_intensity',
      'fancy_color_dominant_color',
      'fancy_color_secondary_color',
      'fancy_color_overtone',
      'fancy_color_intensity']


cols = numerical_columns + categorical_columns
cols
```

```
['carat_weight',
 'depth_percent',
 'table_percent',
 'meas_length',
 'meas_width',
 'meas_depth',
 'cut',
 'color',
 'clarity',
 'cut_quality',
 'lab',
 'symmetry',
 'polish',
 'eye_clean',
 'culet_size',
 'culet_condition',
 'girdle_min',
 'girdle_max',
 'fluor_color',
 'fluor_intensity',
 'fancy_color_dominant_color',
 'fancy_color_secondary_color',
 'fancy_color_overtone',
 'fancy_color_intensity']
```

```
sns.pairplot(df, hue= "cut_quality" ) #help in visualizing the relationships between feature
```

<seaborn.axisgrid.PairGrid at 0x7a980bc94910>

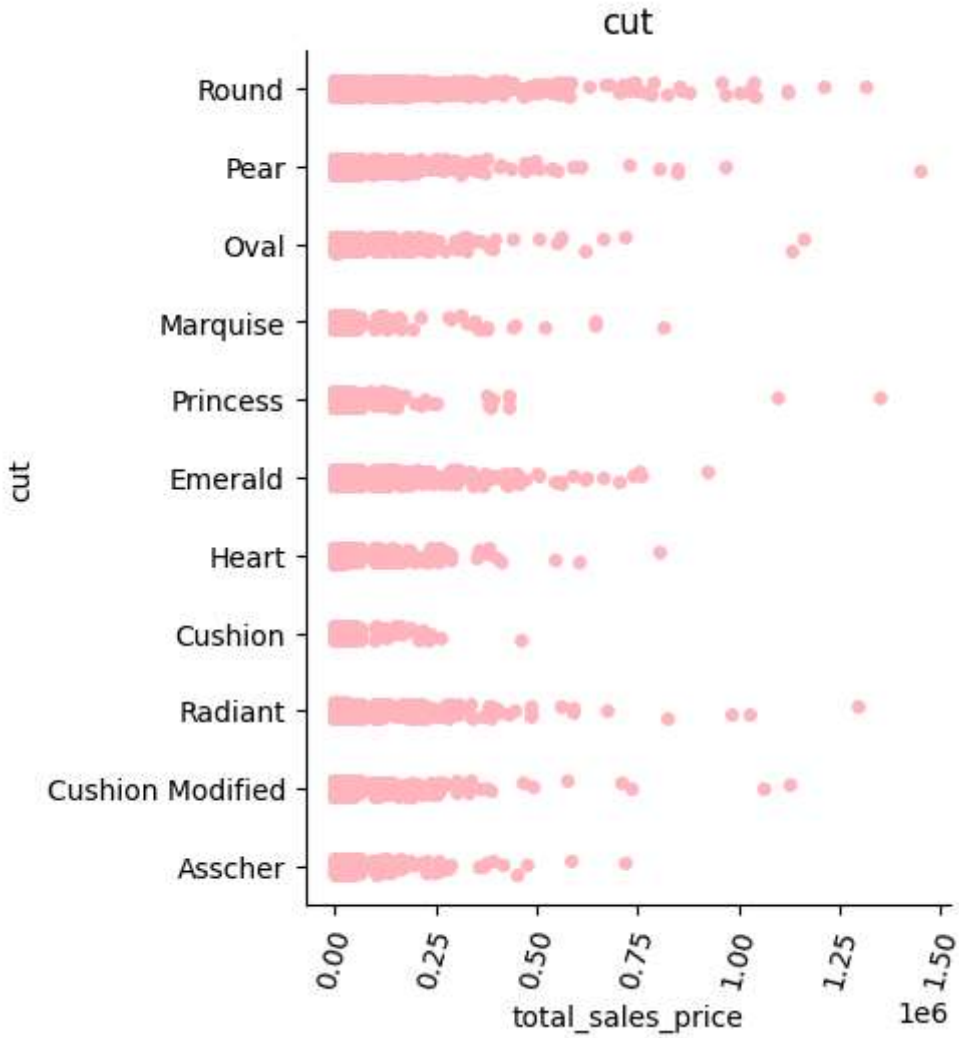


Some of the realtions are in linear Relation and some of are in Polinomial Relation.

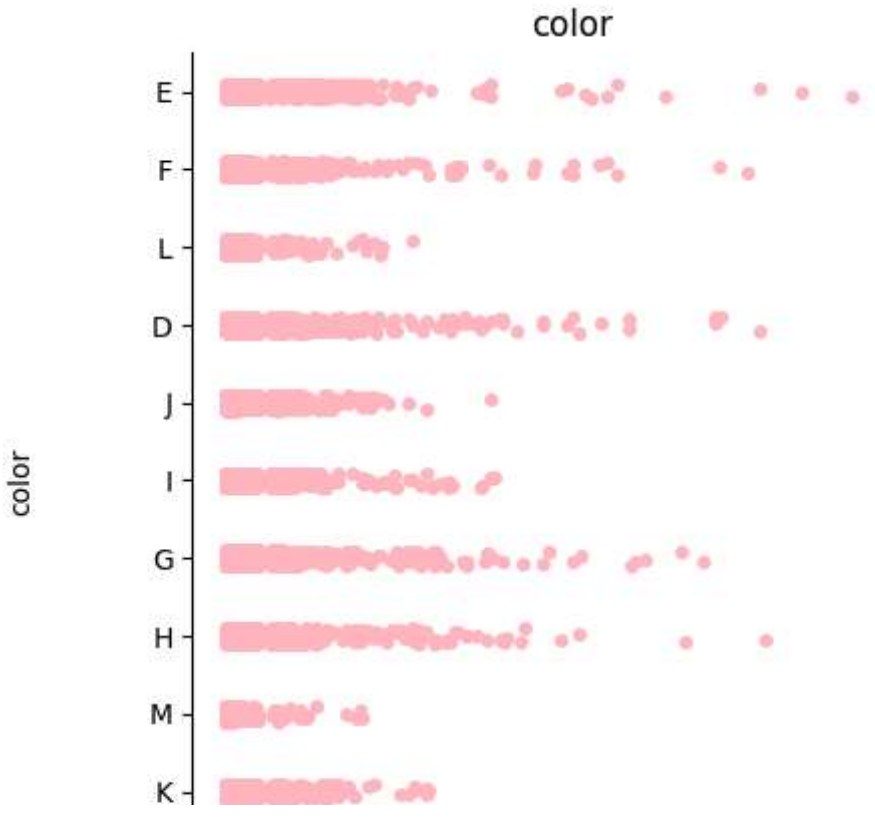
> **Below is a categorical plot of type 'strip', which helps visualize the data more efficiently. It will display all data points in the graph.**

```
for cat in categorical_columns:
    sns.catplot(df,x='total_sales_price',y=cat,color='#FFB6C1')
    plt.xticks(rotation=75)
    plt.title(cat)
    print(cat +':')
    plt.show()
```

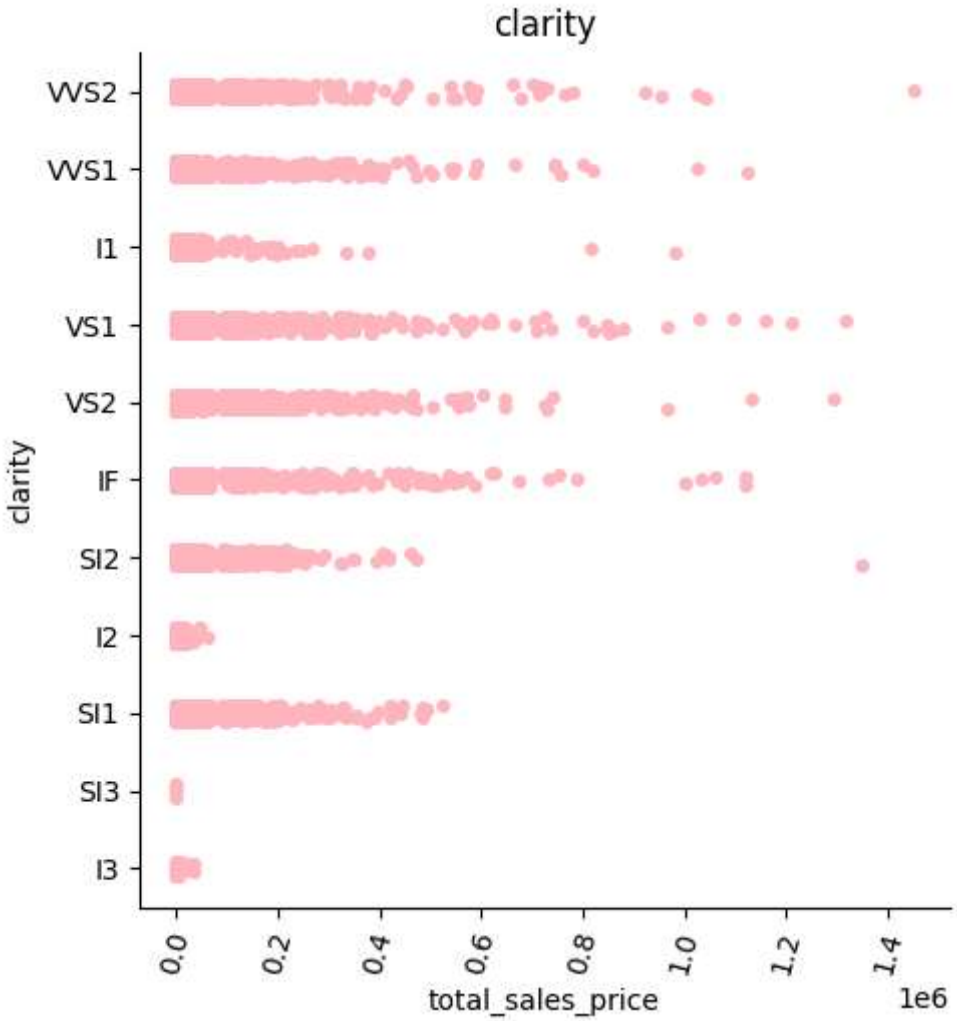
cut:


cut
Round
Pear
Oval
Marquise
Princess
Emerald
Heart
Cushion
Radiant
Cushion Modified
Asscher
cut
0.00
0.25
0.50
0.75
1.00
1.25
1.50
total_sales_price
1e6

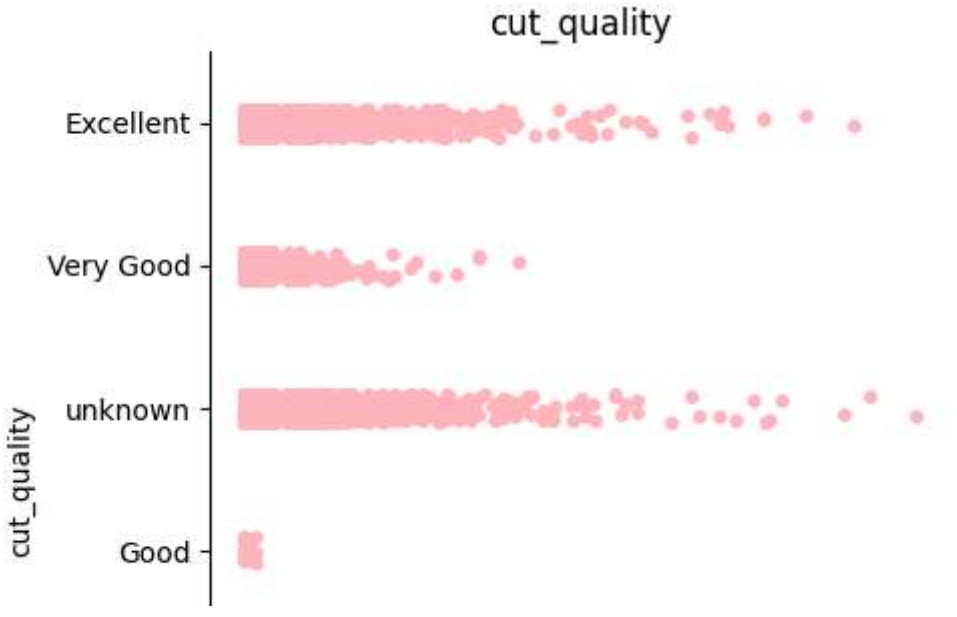
color:

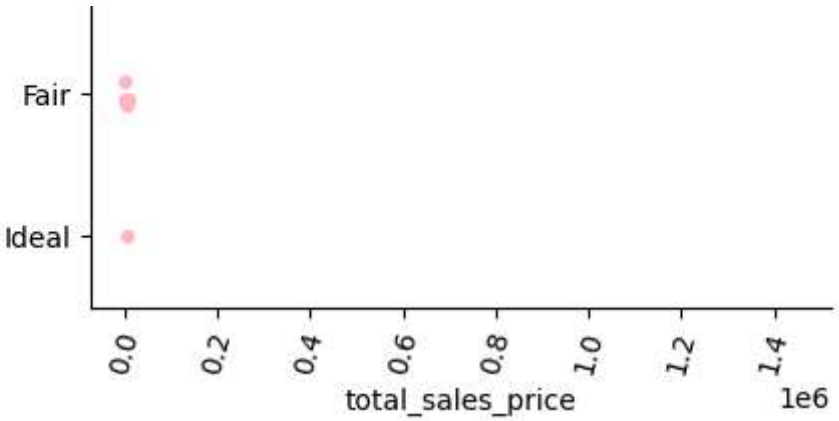

color
E
F
L
D
J
I
G
H
M
K
color

clarity:

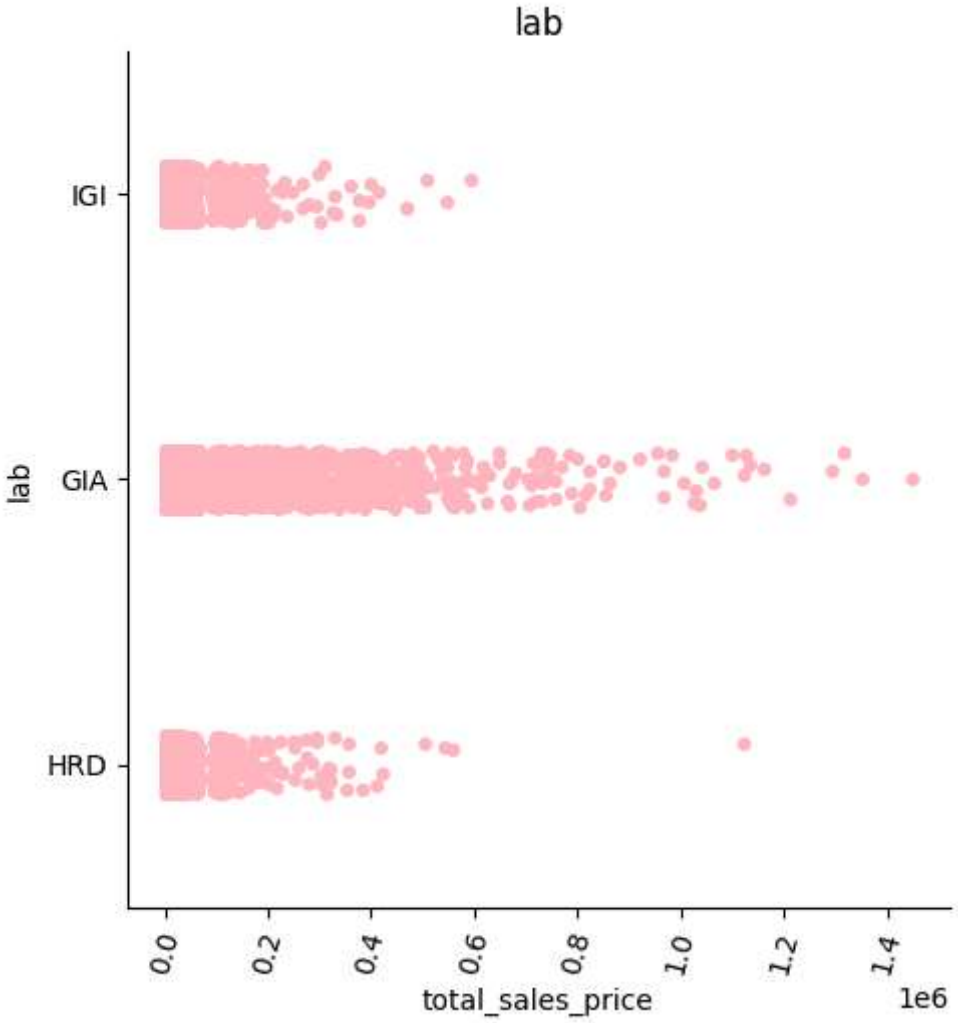


cut_quality:

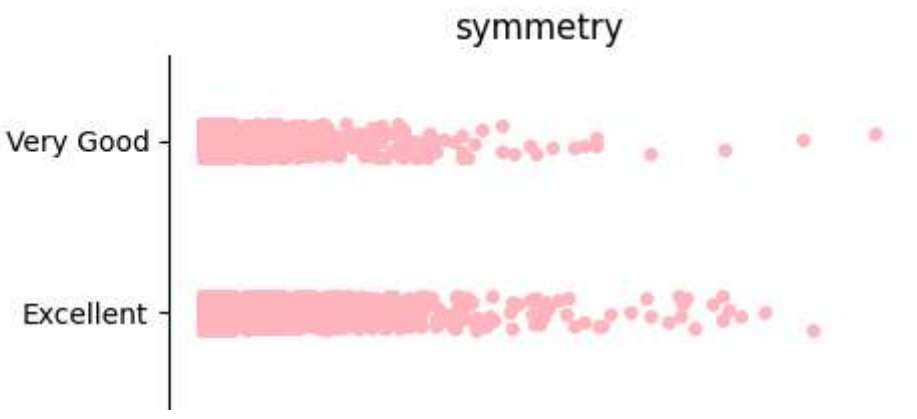


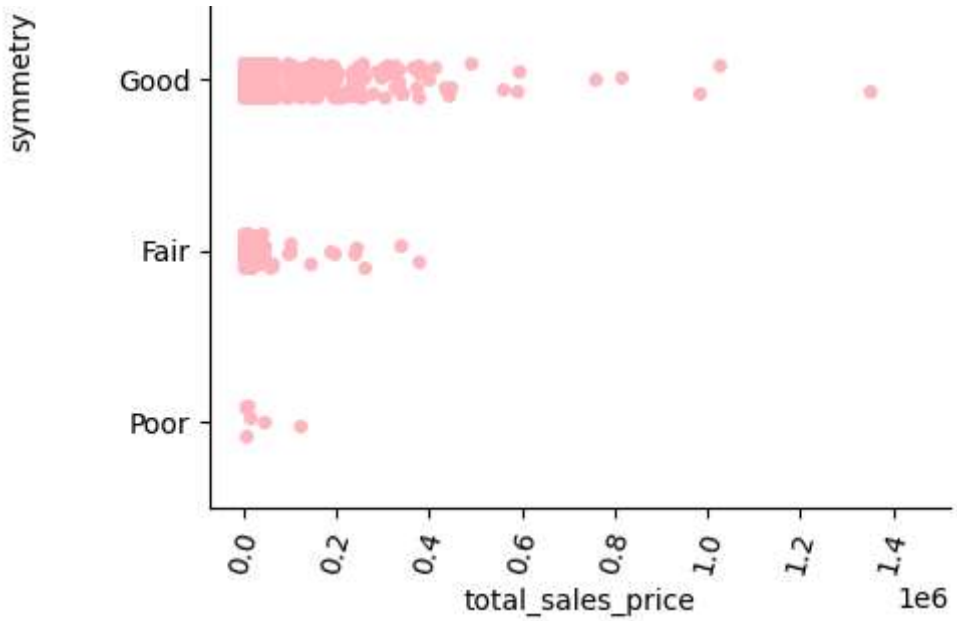


lab:

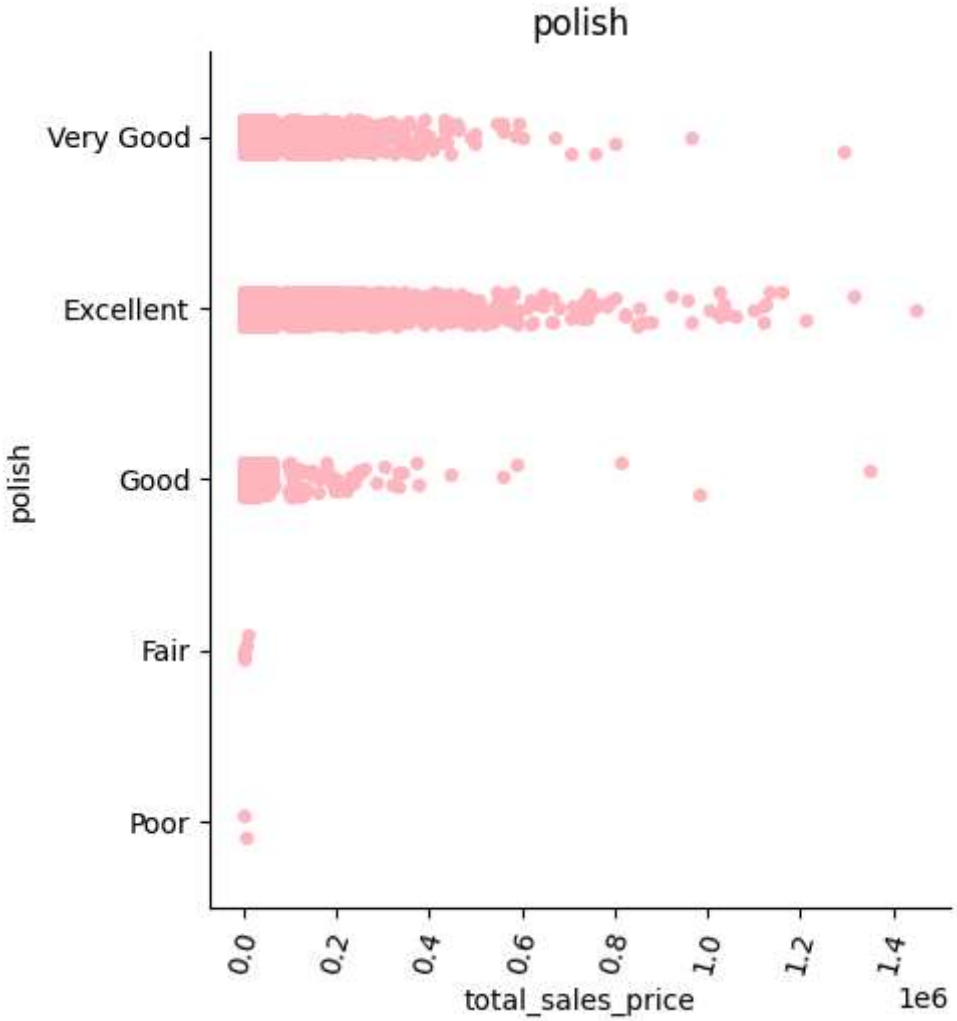


symmetry:

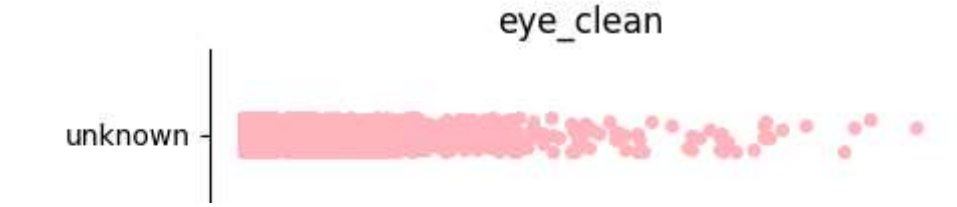


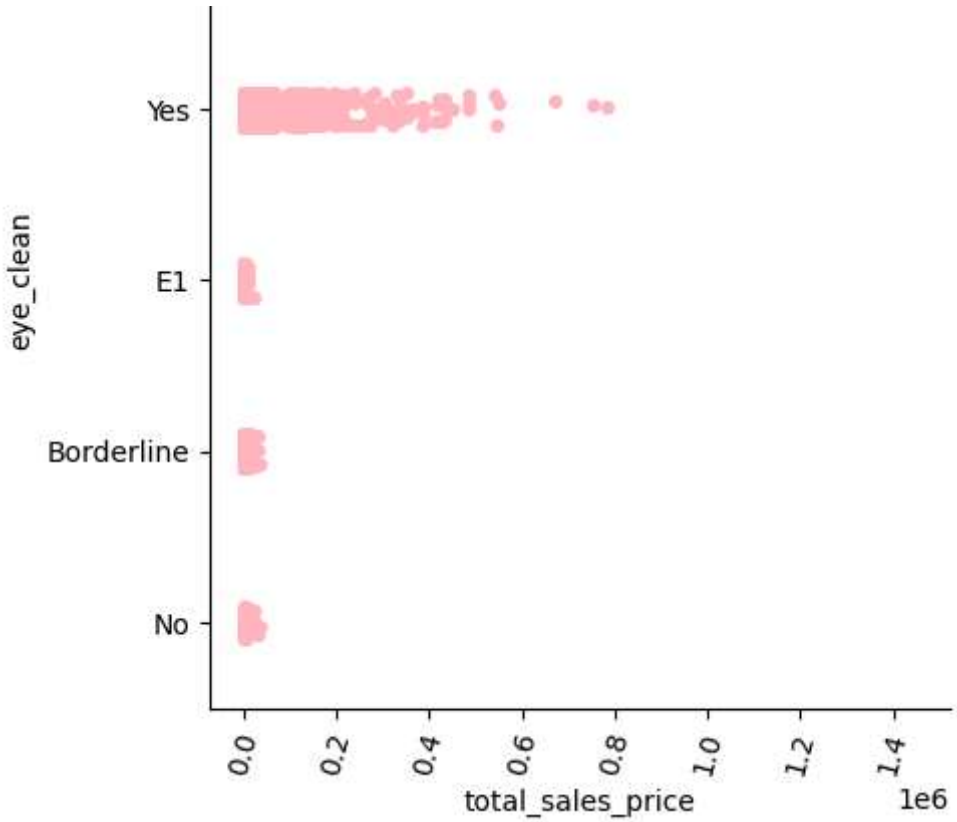


polish:

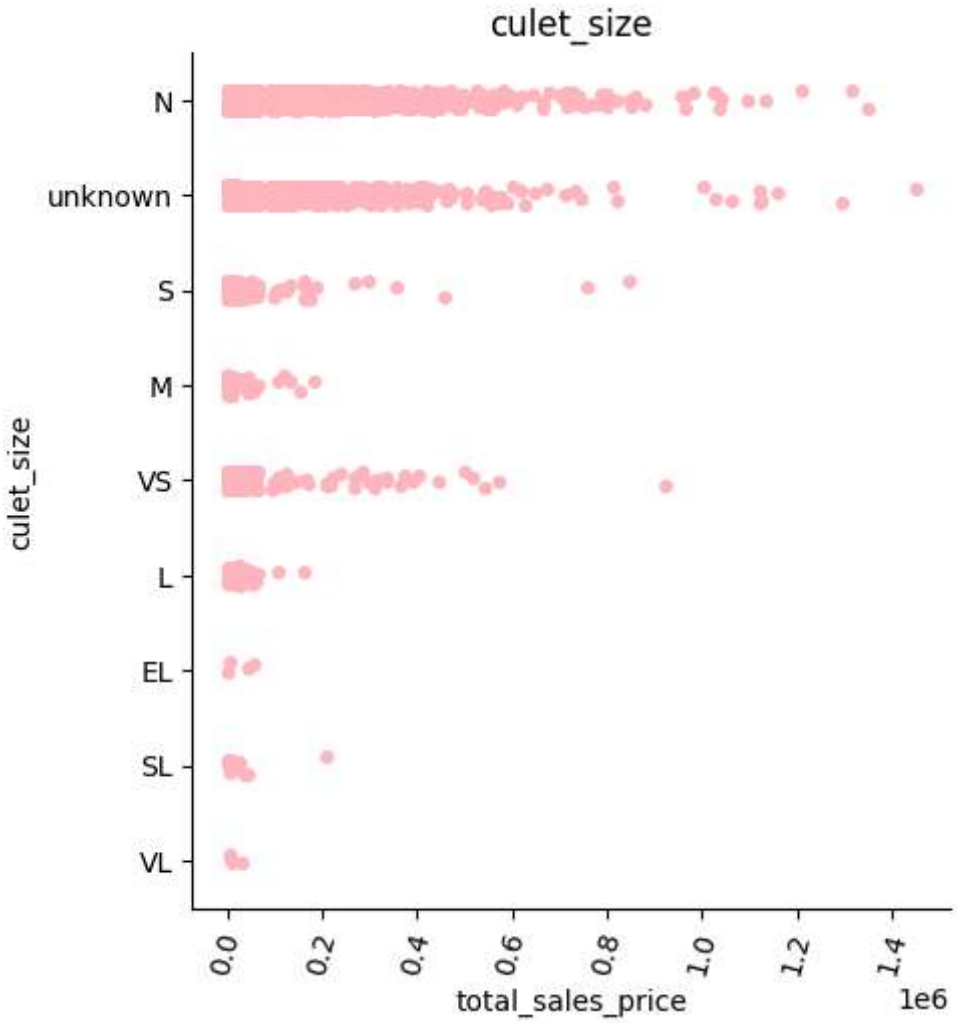


eye_clean:

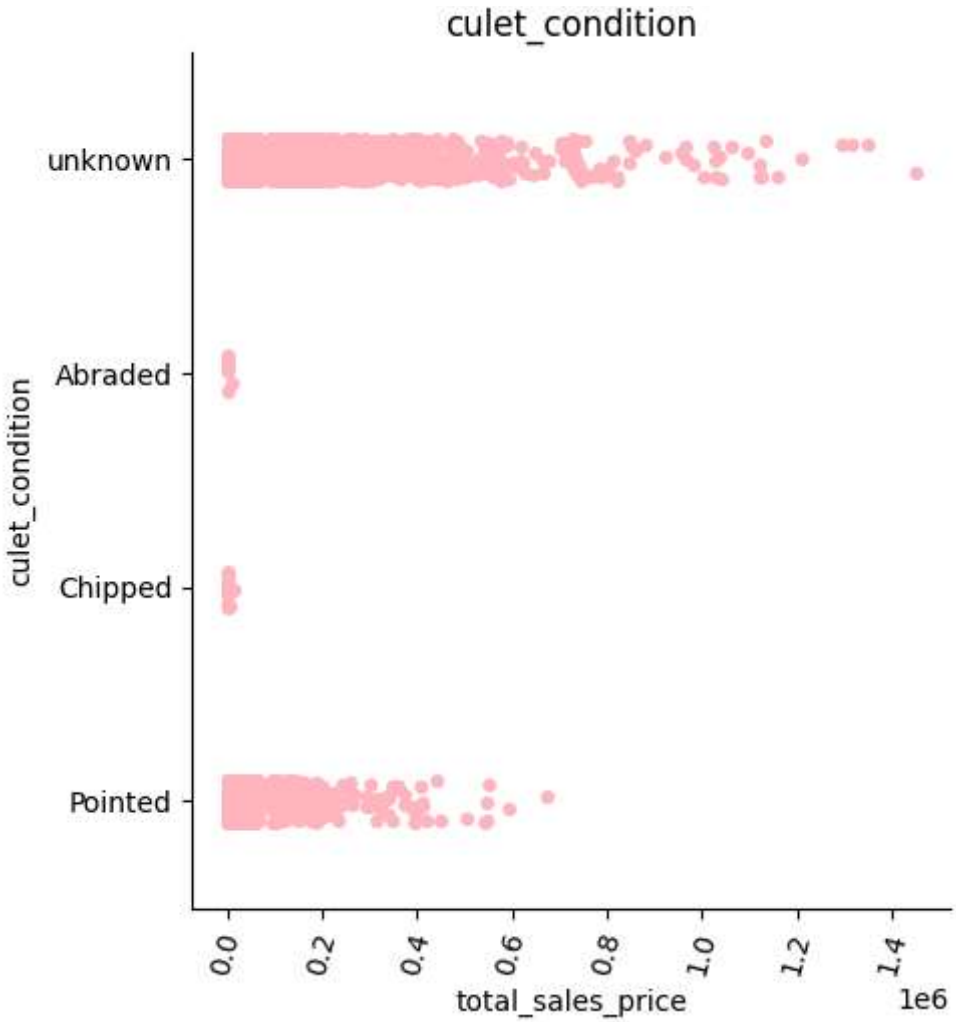




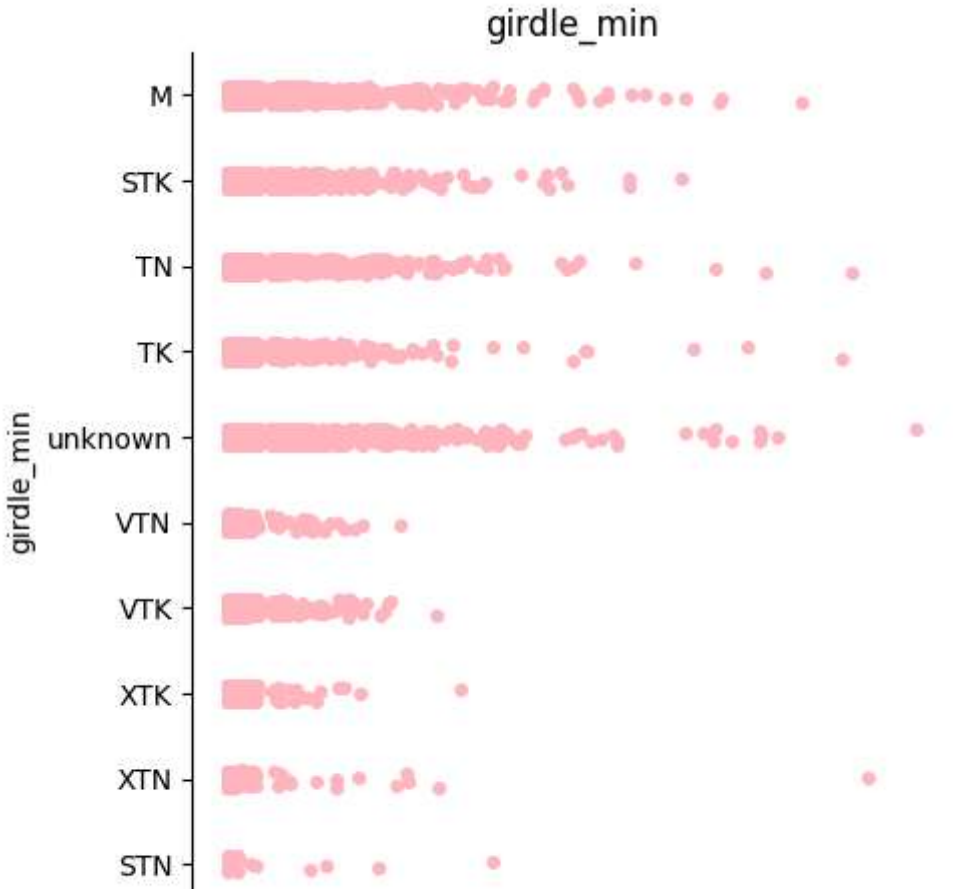
culet_size:

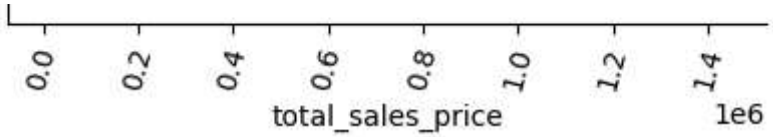


culet_condition:

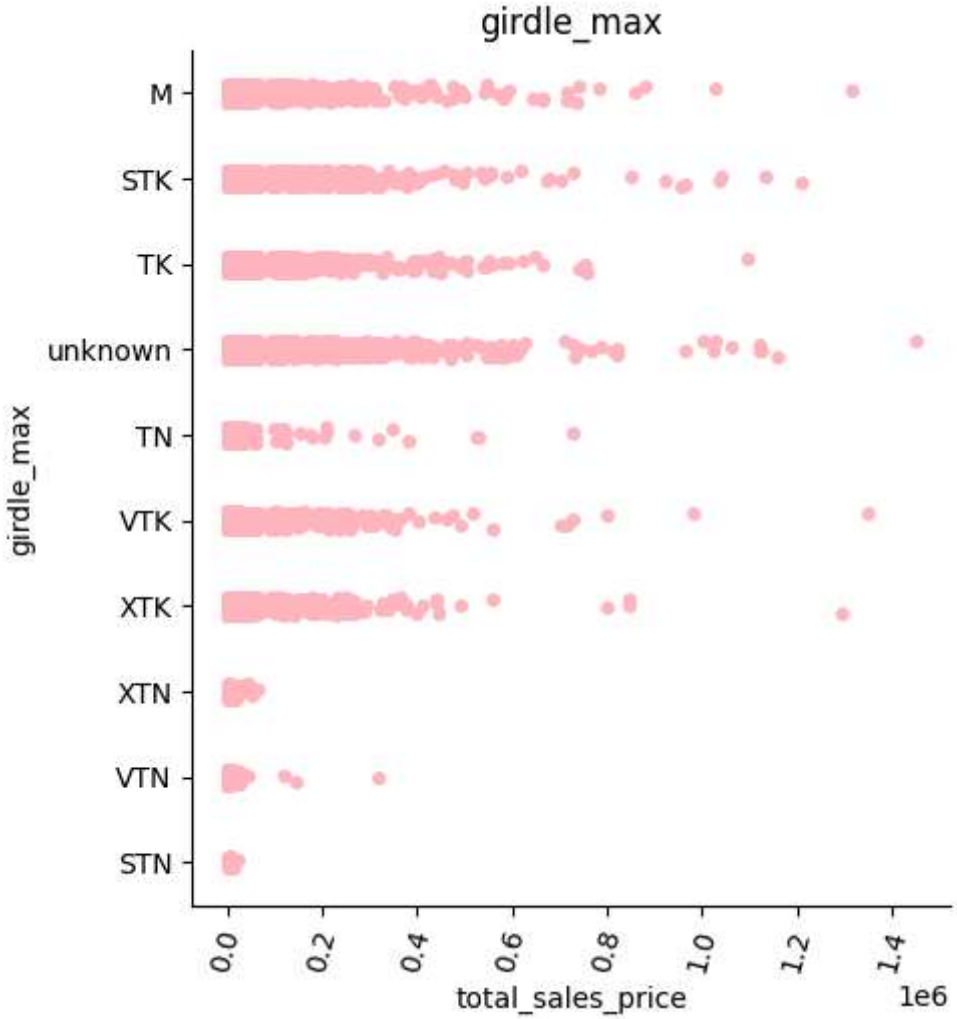

culet_condition
unknown
Abraded
Chipped
Pointed
culet_condition
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
total_sales_price
1e6

girdle_min:

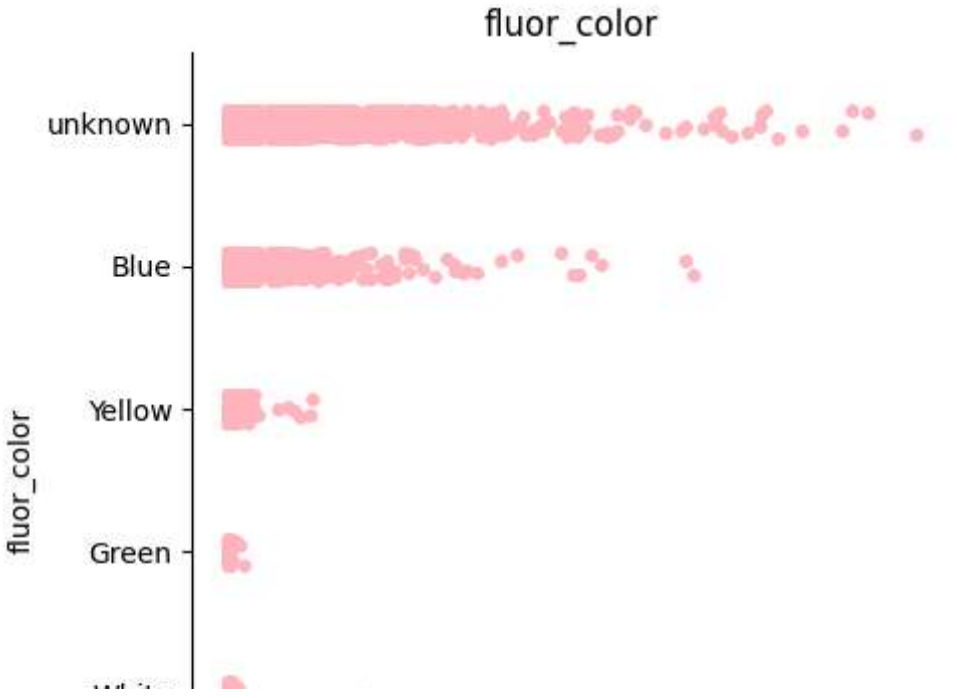

girdle_min
M
STK
TN
TK
unknown
VTN
VTK
XTK
XTN
STN
girdle_min

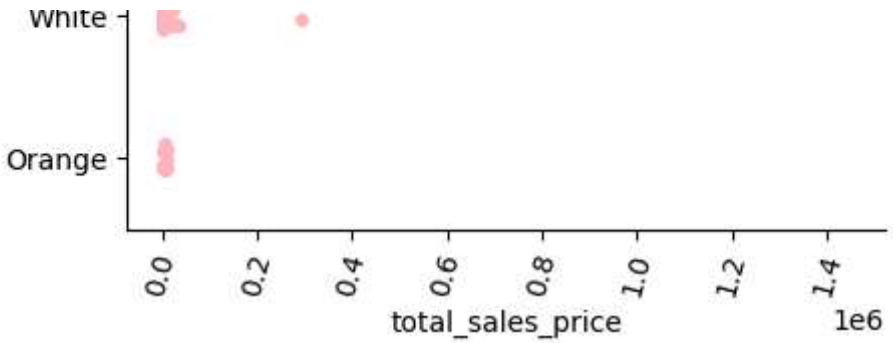


girdle_max:

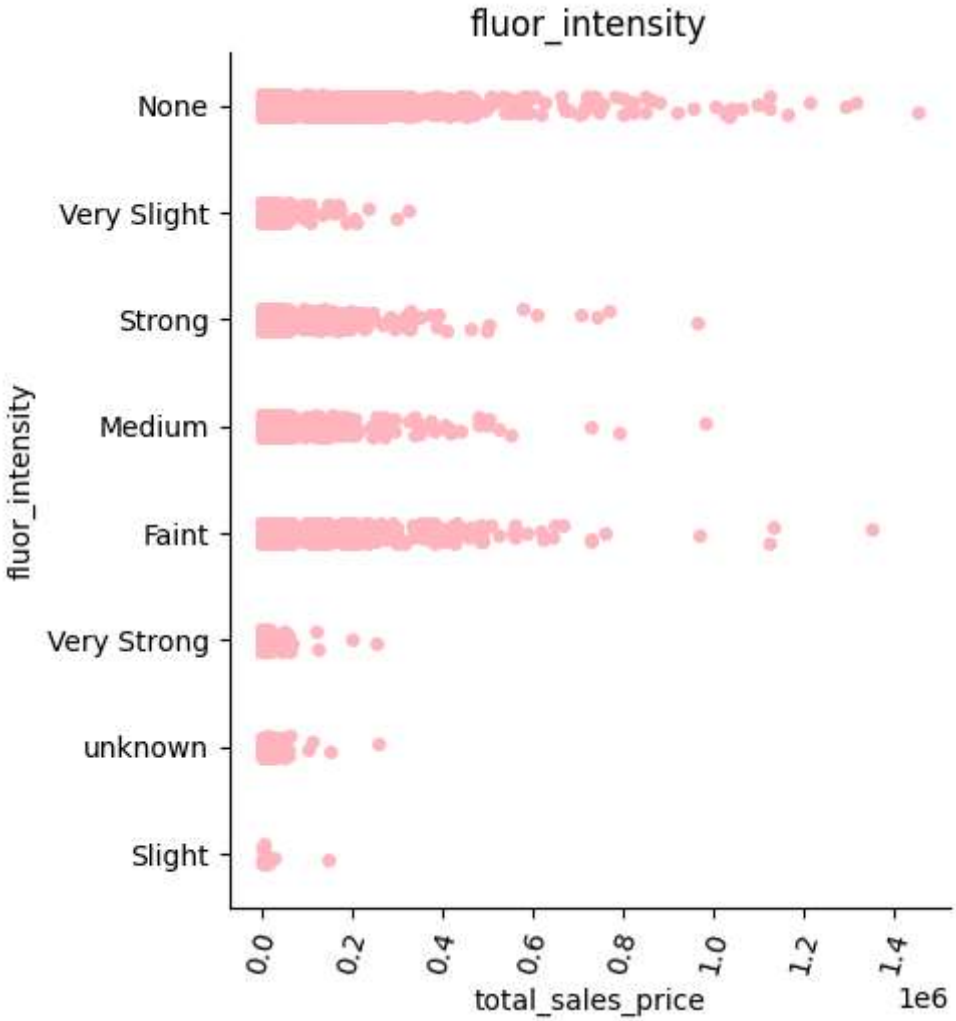


fluor_color:

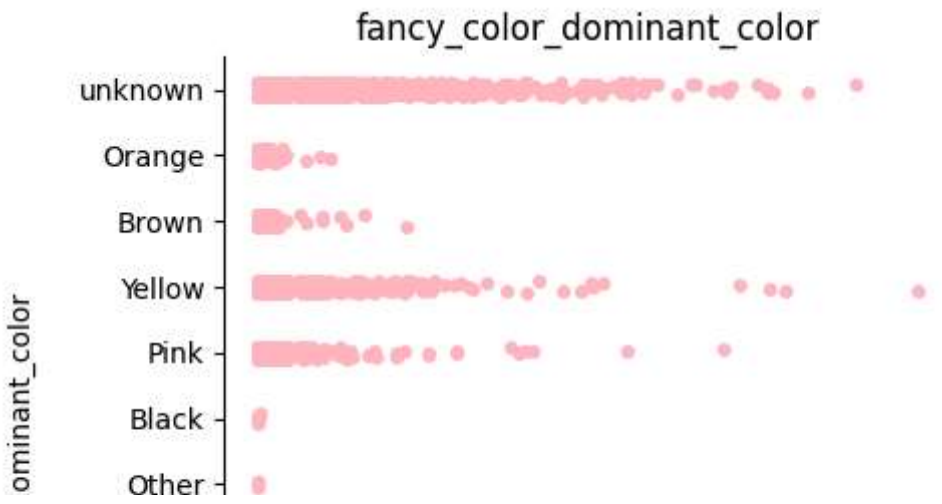


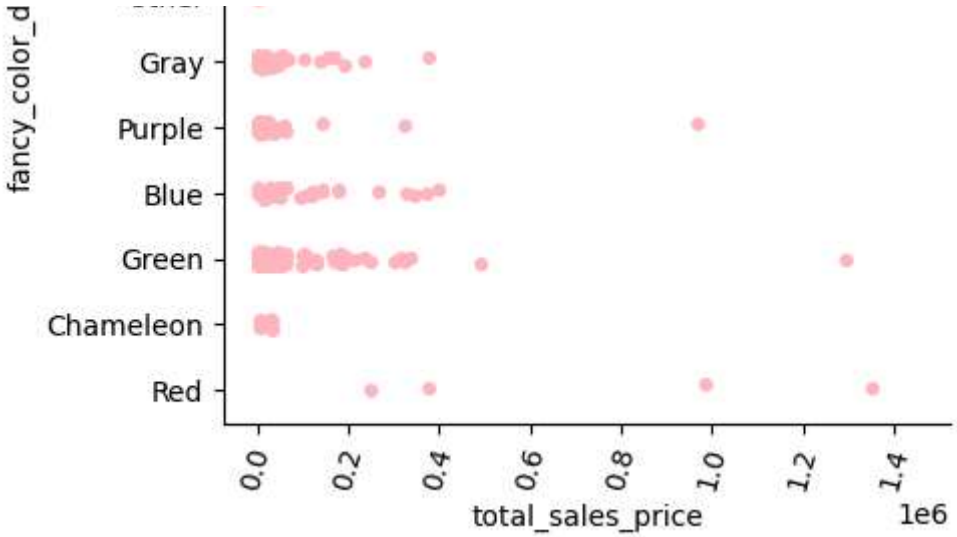


`fluor_intensity:`

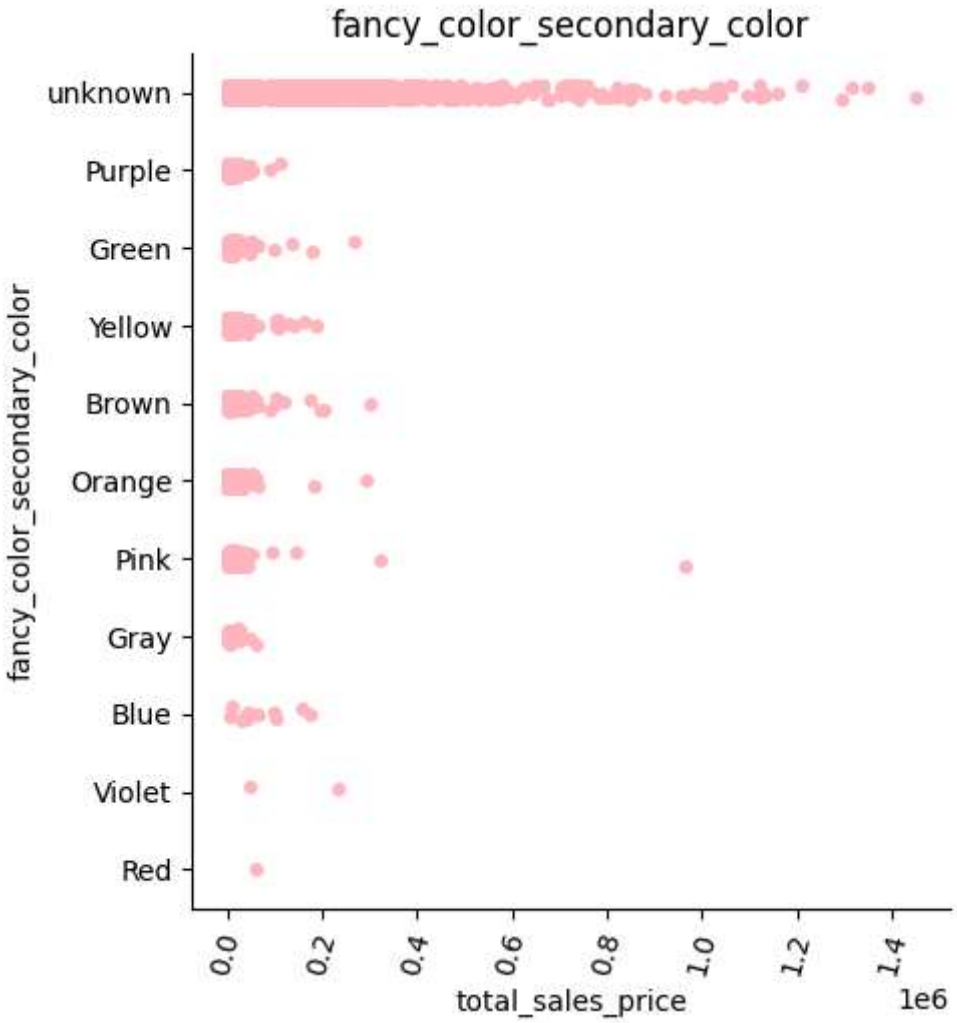


`fancy_color_dominant_color:`

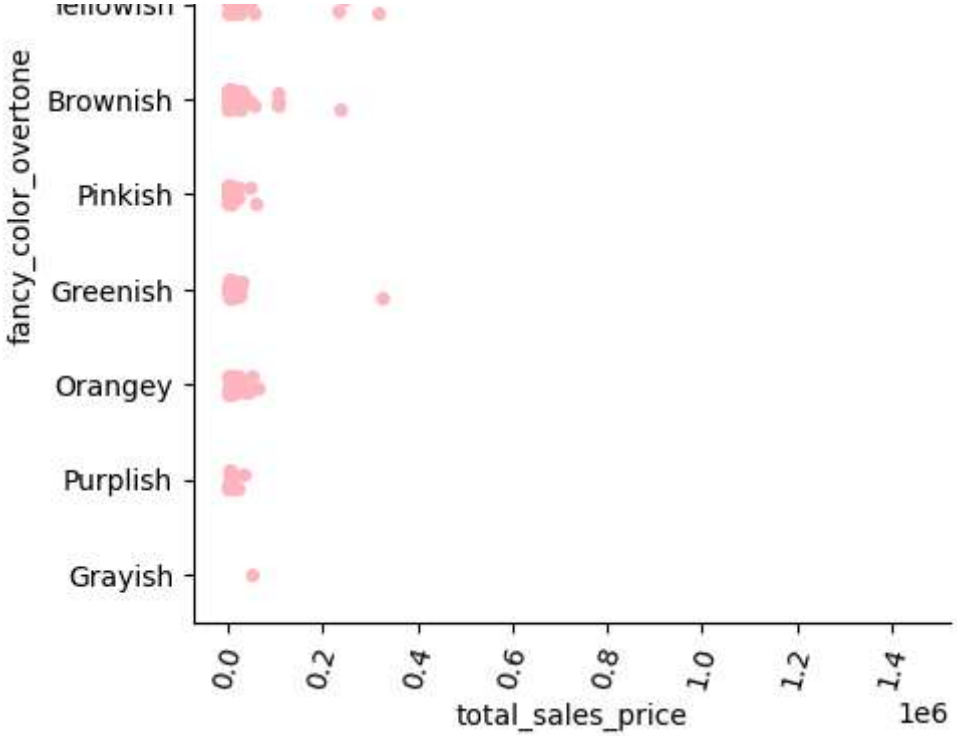




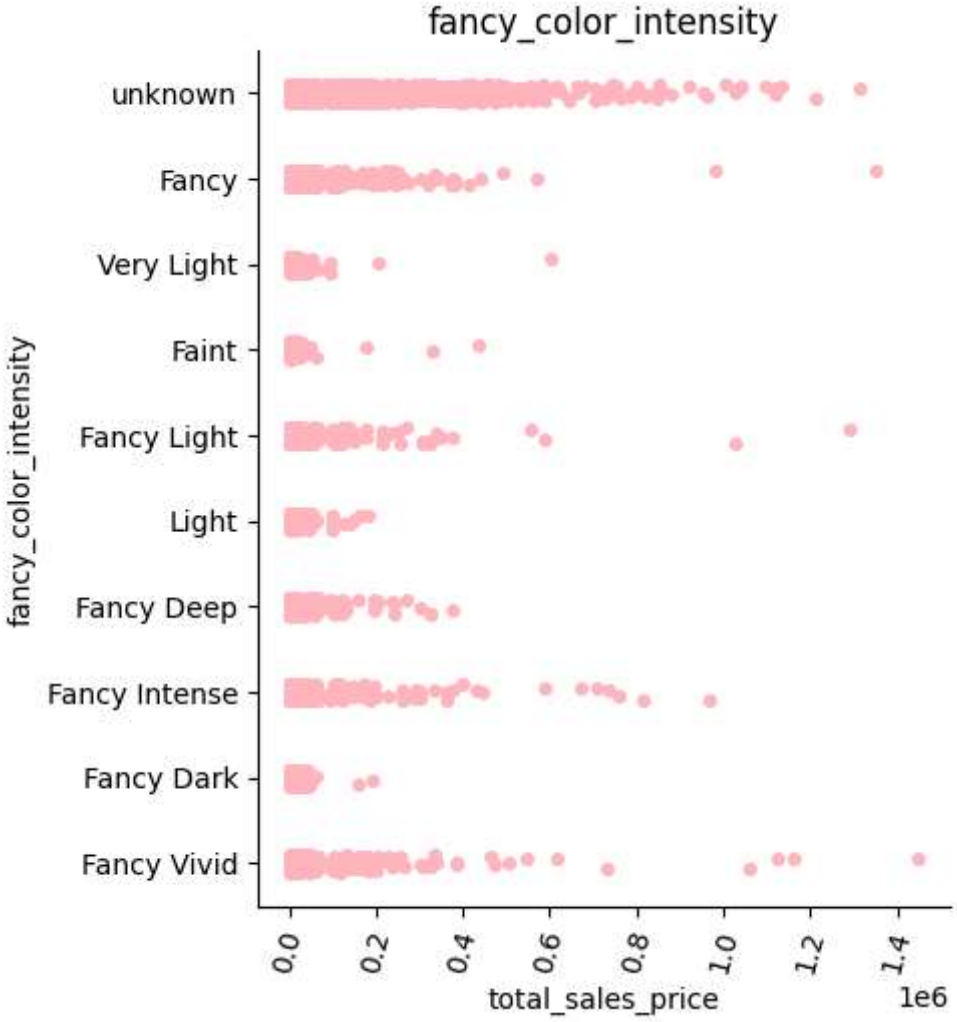
`fancy_color_secondary_color`:



`fancy_color_overtone`:

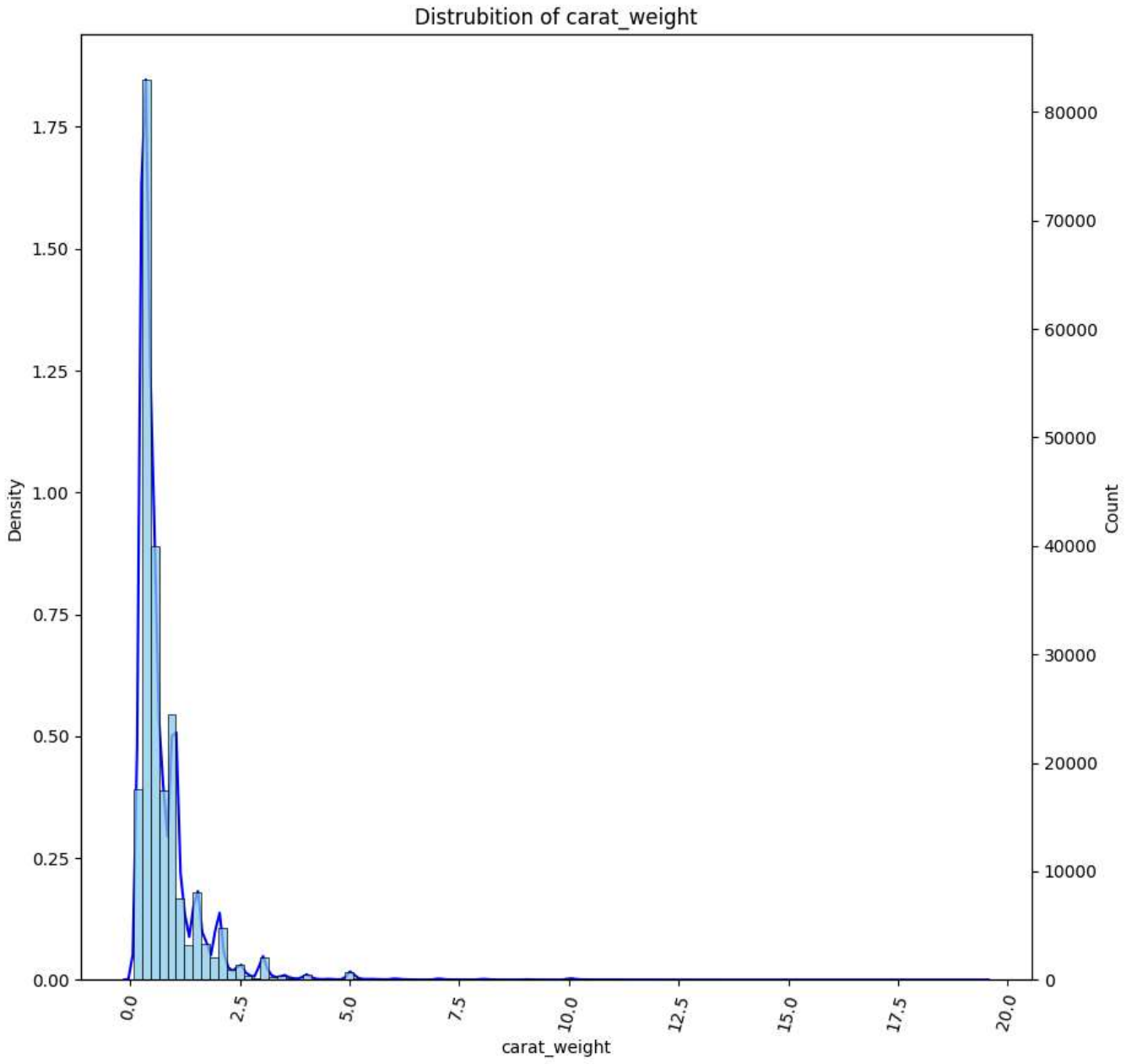


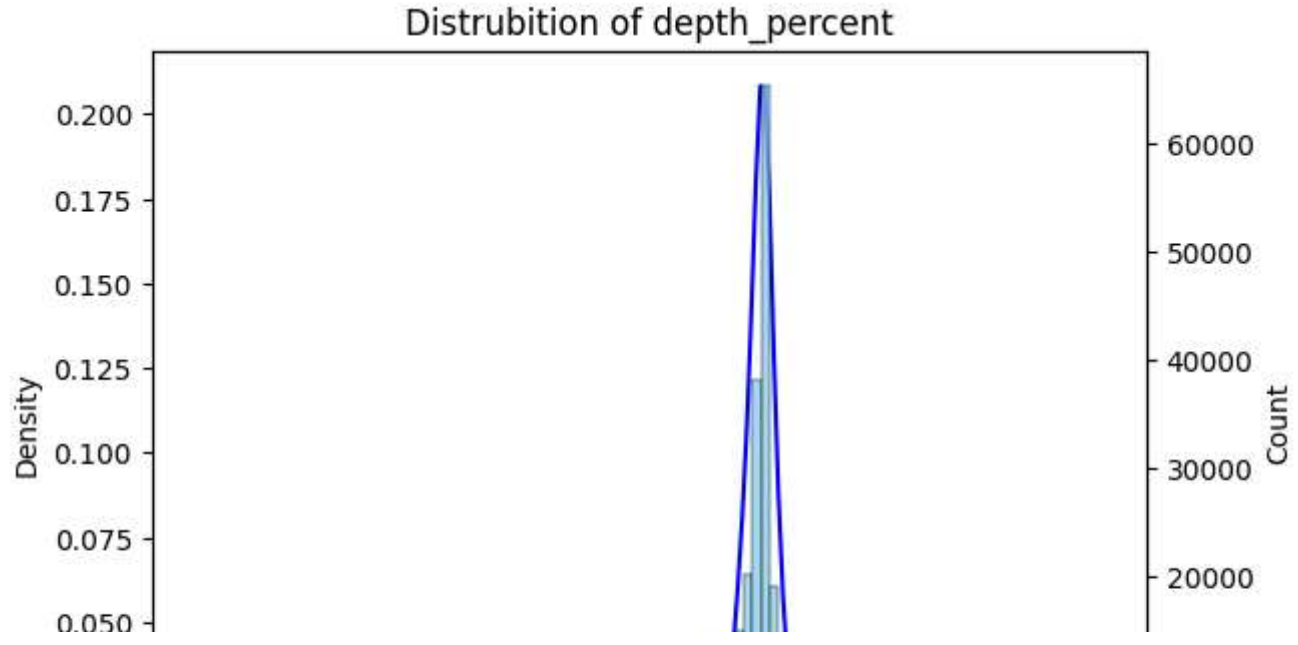
`fancy_color_intensity`:

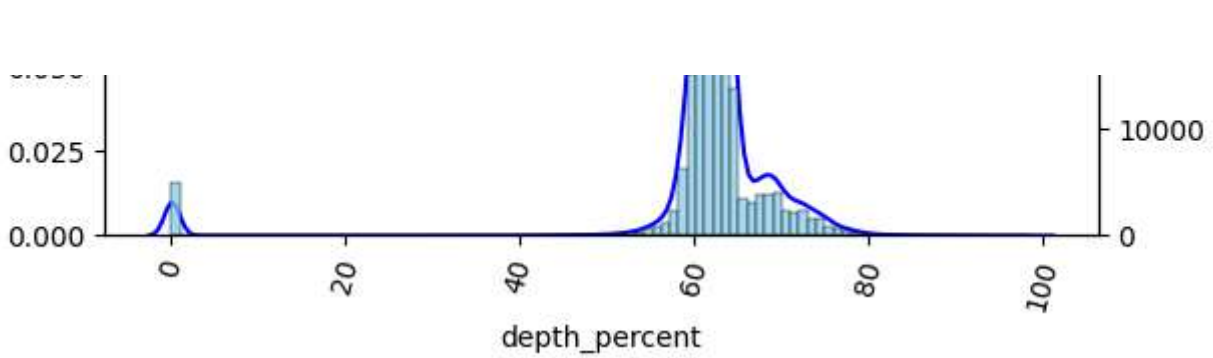


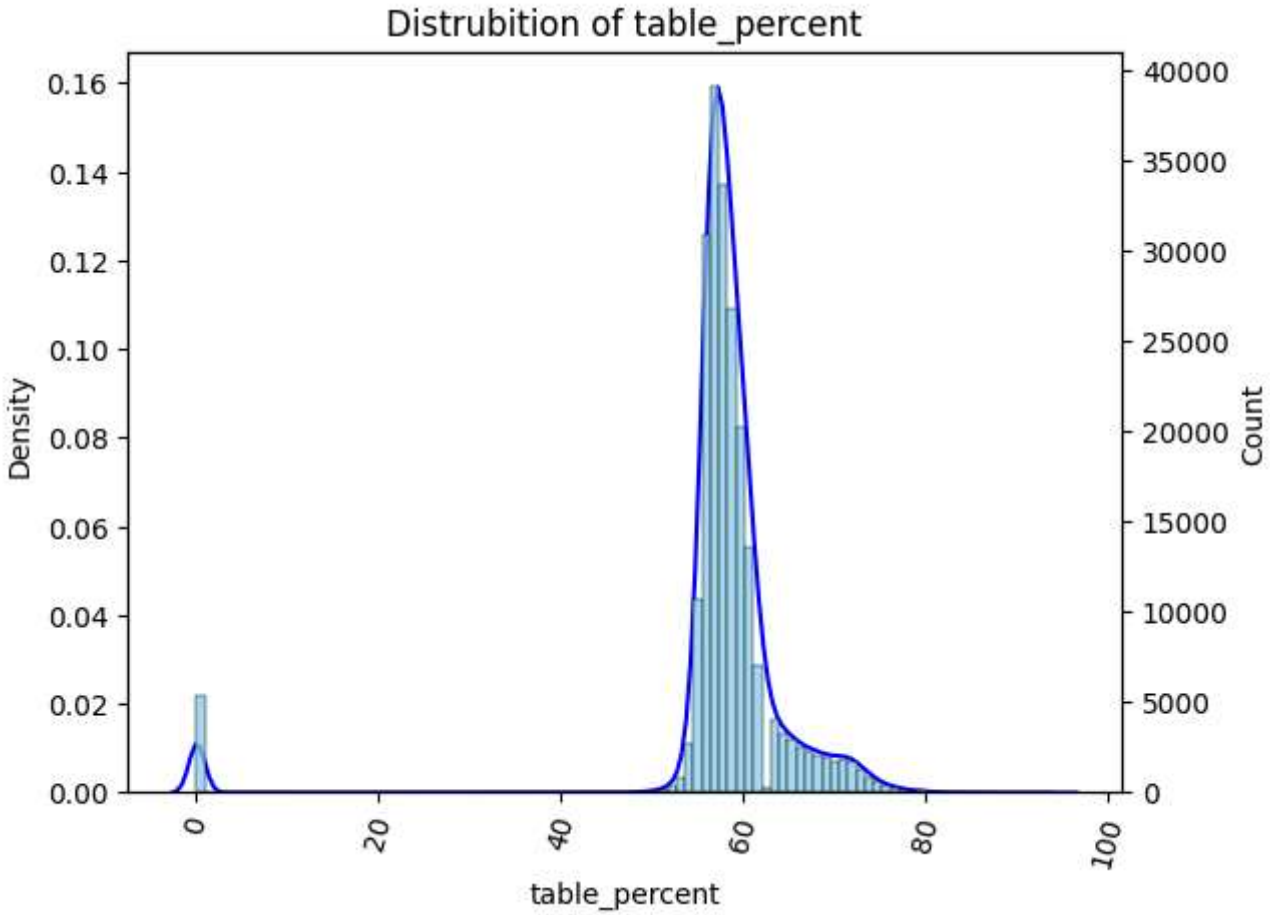
Probability Distribution Function (PDF) of the numeric attributes

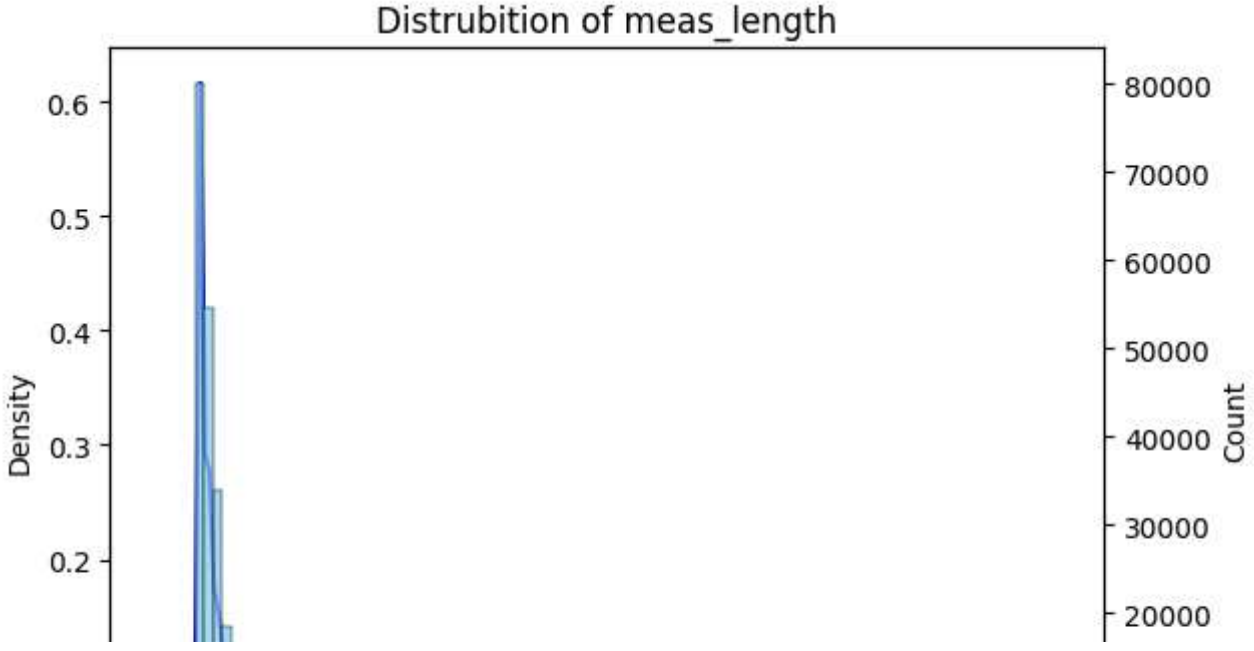
```
#It will show the Probability Distribution Function (PDF) of the numeric attributes,
#and the second plot is a histogram of the count of that numeric attribute.
#As shown below, the histogram and the distribution function appear similar
#but are scaled differently.

fig, ax = plt.subplots(figsize=(10,10))
for cat in numerical_columns:
    sns.kdeplot(df,x=cat,color='blue')
    plt.xticks(rotation=75)
    plt.title('Distrubition of {}'.format(cat))
    plt.twinx()
    sns.histplot(data=df, x=cat, bins=100,color=pc[5])
    plt.show()
    print()
sns.kdeplot(df,x='total_sales_price',gridsize=100,color='blue')
plt.title('Distrubition of total_sales_price')
plt.twinx()
sns.histplot(data=df, x='total_sales_price', bins=100,color=pc[5])
plt.show()
```

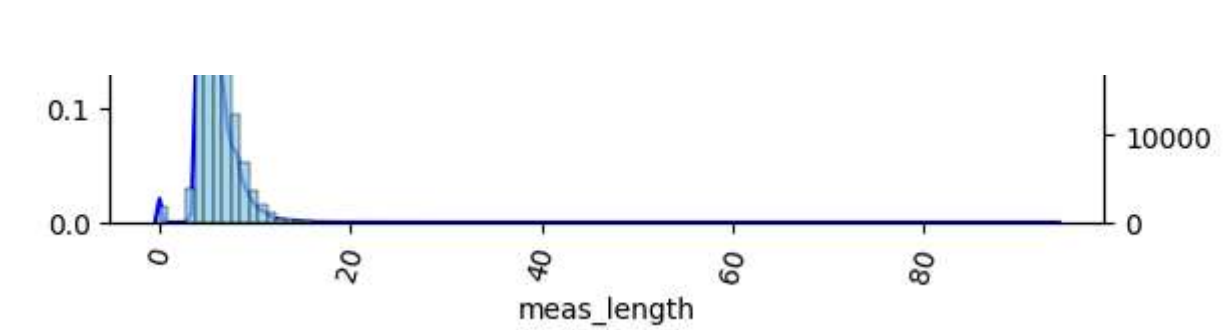

Distrubition of carat_weight
Density
Count
carat_weight
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
10000
20000
30000
40000
50000
60000
70000
80000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

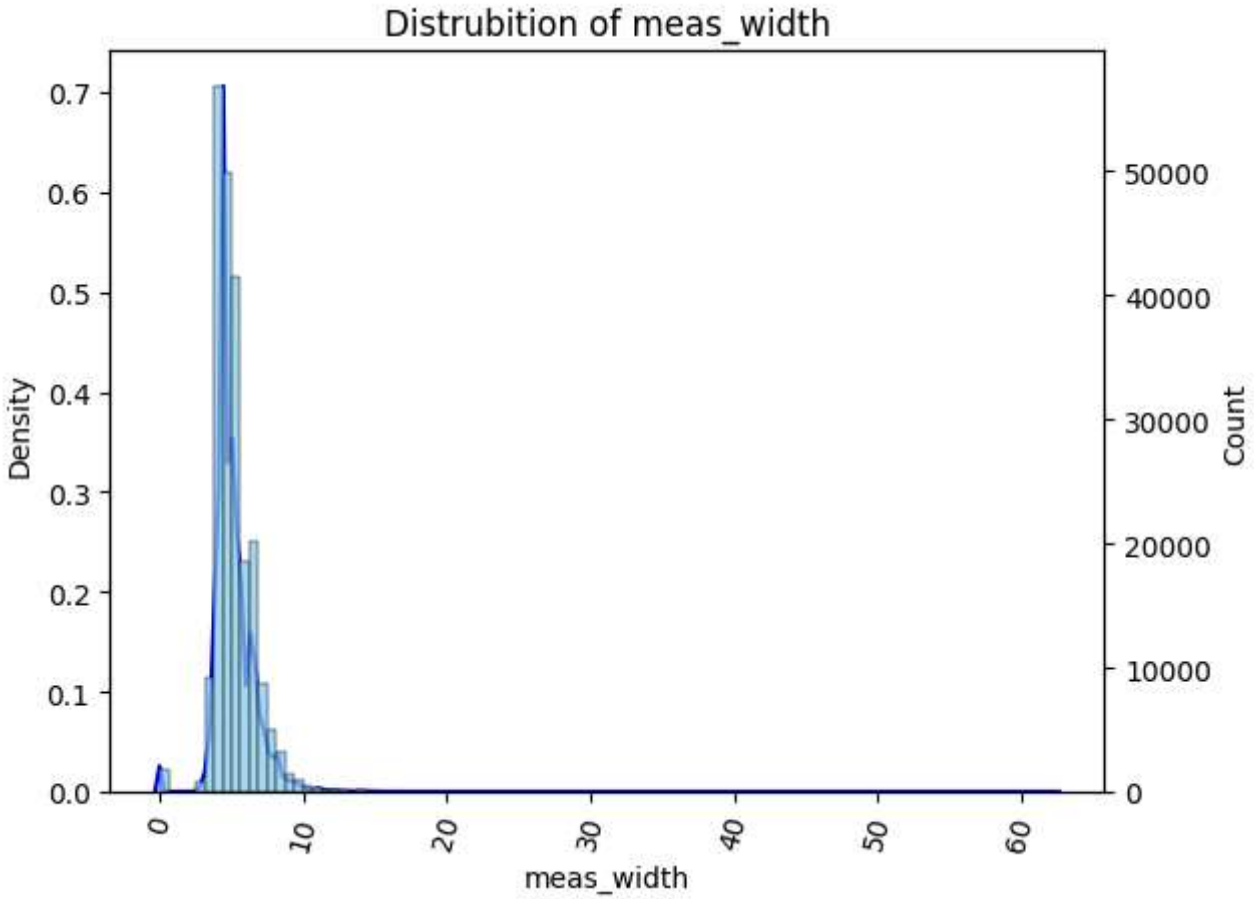

Distrubition of depth_percent
Density
Count
0.050
0.075
0.100
0.125
0.150
0.175
0.200
20000
30000
40000
50000
60000

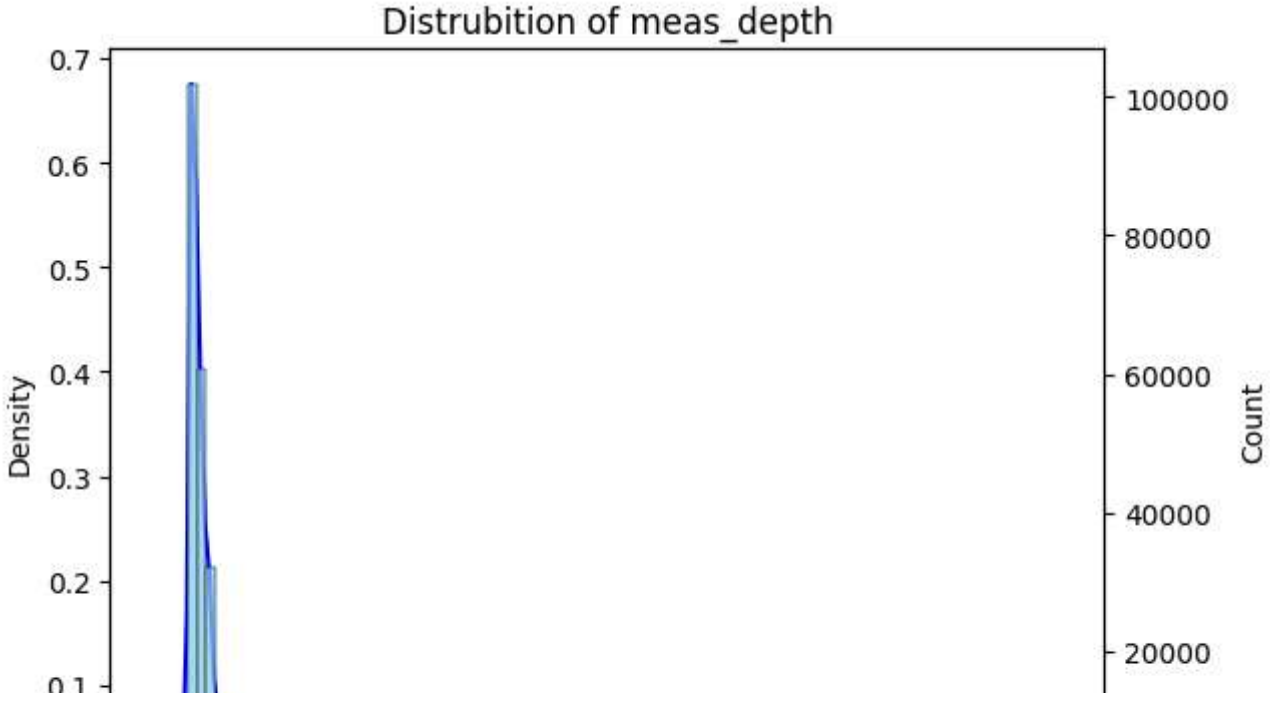

0.025
0.000
10000
0
0
20
40
60
80
100
depth_percent

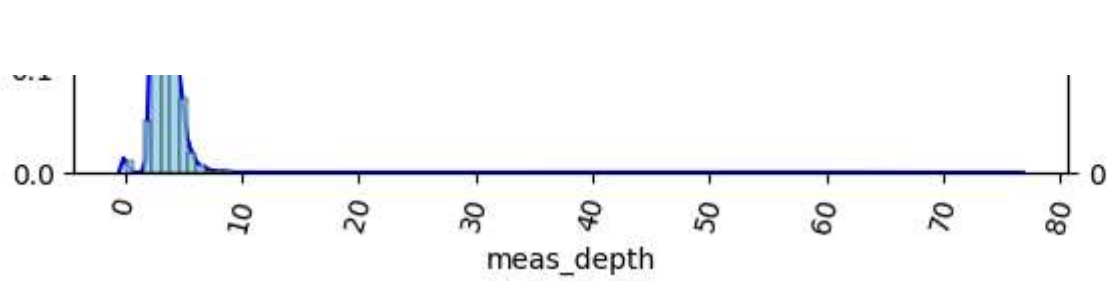

Distrubition of table_percent
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
40000
35000
30000
25000
20000
15000
10000
5000
0
Count
0
20
40
60
80
100
table_percent

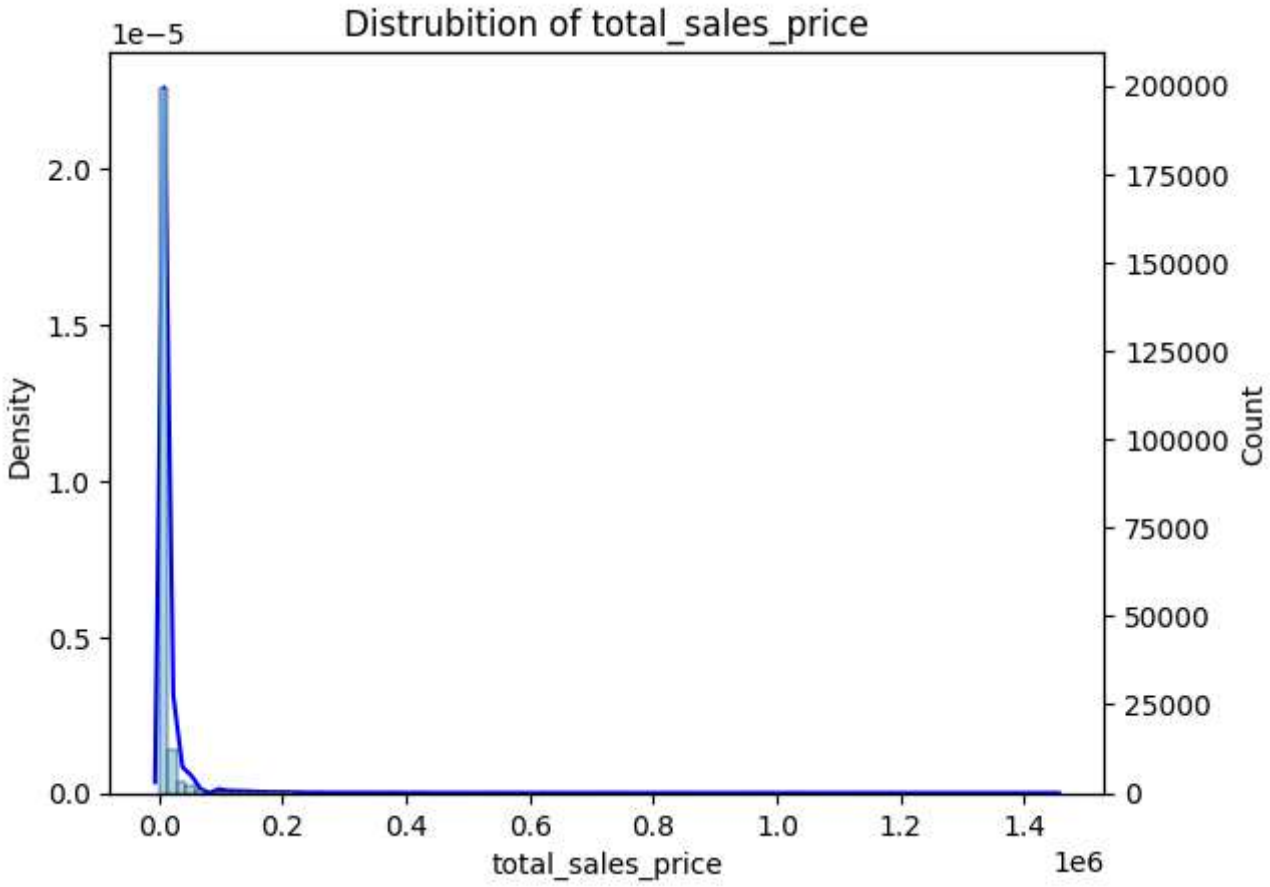

Distrubition of meas_length
0.6
0.5
0.4
0.3
0.2
Density
80000
70000
60000
50000
40000
30000
20000
Count


0.1
0.0
10000
0
0
20
40
60
80
meas_length


Distrubition of meas_width
Density
Count
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
60
meas_width


Distrubition of meas_depth
Density
Count
0.7
0.6
0.5
0.4
0.3
0.2
0.1
100000
80000
60000
40000
20000

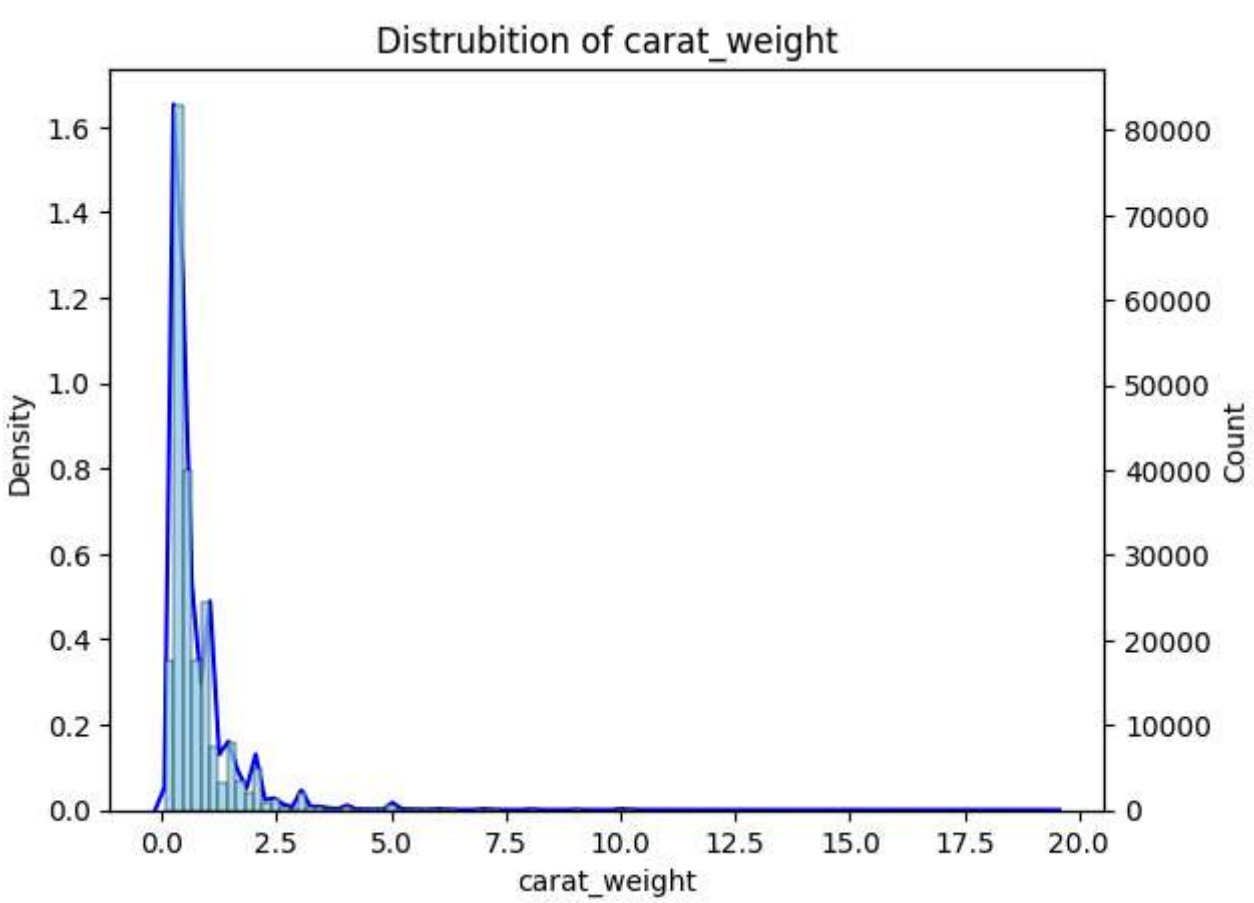




This way we can analyse the data where the most of the data are and where the least data are.

Let's See some of the Outliers and should we remove it or not ?

```
sns.kdeplot(df,x='carat_weight',gridsize=100,color='blue')
plt.title('Distrubition of carat_weight')
plt.twinx()
sns.histplot(data=df, x='carat_weight', bins=100,color=pc[5])
plt.show()
```



As per the above graph, we can see there are some data points for selling prices greater than 10. These might seem like outliers, but they are not because some people, like celebrities and influencers, are interested in buying at these prices. Therefore, we cannot consider them as outliers in this case.

```
#Count of Data Contribution/Participation In Lab Attribute
lab_dist = df["lab"].value_counts()
lab_dist
```

```
lab
GIA    200434
IGI     15865
HRD      3404
Name: count, dtype: int64
```

```
#Count of Data Contribution/Participation In Color Attribute
color_dist = df["color"].value_counts()
color_dist
```

```
color
E          33103
F          31566
D          30873
G          29184
H          26073
I          22364
J          16898
K          11750
unknown     9162
L           5683
M           3047
Name: count, dtype: int64
```

```
#Count of Data Contribution/Participation In Clarity Attribute
clarity_dist = df["clarity"].value_counts()
clarity_dist
```

```
clarity
SI1     38627
VS2     38173
VS1     36956
SI2     31105
VVS2    28985
VVS1    27877
IF       9974
I1       6961
I2        944
I3         91
SI3        10
Name: count, dtype: int64
```

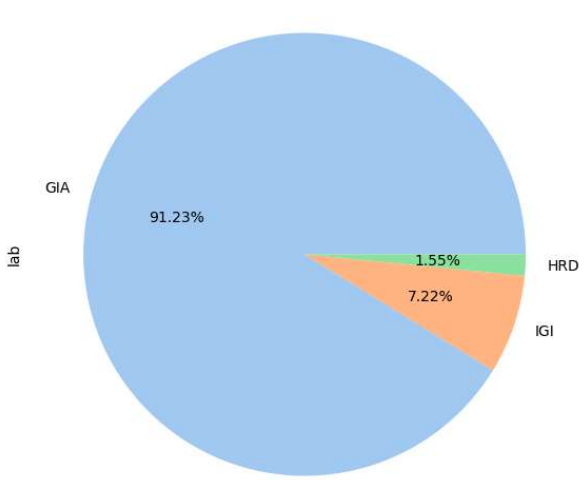
```
cut_dist = df["cut"].value_counts()
cut_dist
```

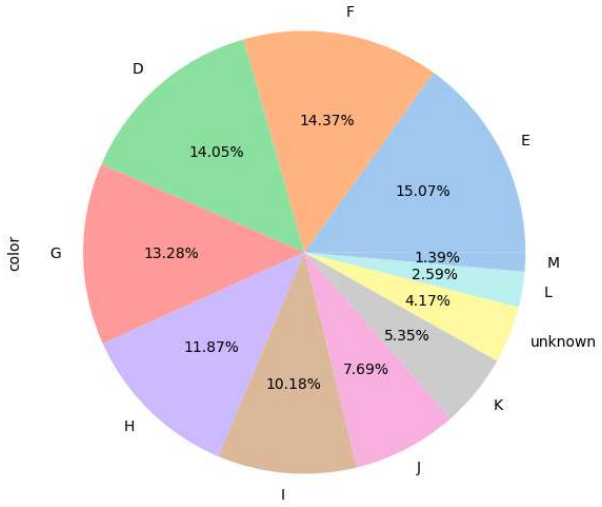
```
cut
Round               158316
Oval                 13857
Emerald              11091
Pear                  9860
Princess              7050
Radiant               5630
Heart                 4774
Cushion Modified      3984
Marquise              2916
Asscher               1696
Cushion                529
Name: count, dtype: int64
```

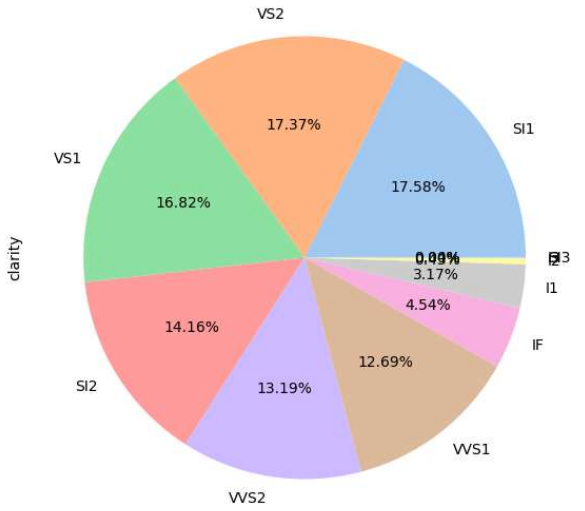
> **Below is a pie chart of these attributes, showing the percentage of their contribution:**

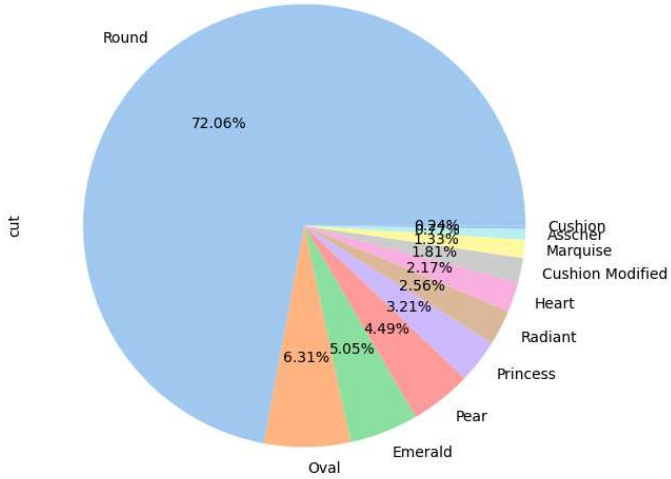
```
plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
piec_colors = sns.color_palette('pastel')[0:len(lab_dist)]
lab_dist.plot(kind = 'pie', colors=piec_colors, autopct = '%1.2f%%')
plt.subplot(2, 2, 2)
piec_colors = sns.color_palette('pastel')[0:len(color_dist)]
color_dist.plot(kind = 'pie', colors=piec_colors, autopct = '%1.2f%%')
plt.subplot(2, 2, 3)
piec_colors = sns.color_palette('pastel')[0:len(clarity_dist)]
clarity_dist.plot(kind = 'pie', colors=piec_colors, autopct = '%1.2f%%')
plt.subplot(2, 2, 4)
piec_colors = sns.color_palette('pastel')[0:len(cut_dist)]
cut_dist.plot(kind = 'pie', colors=piec_colors, autopct = '%1.2f%%')
```

<Axes: ylabel='cut'>

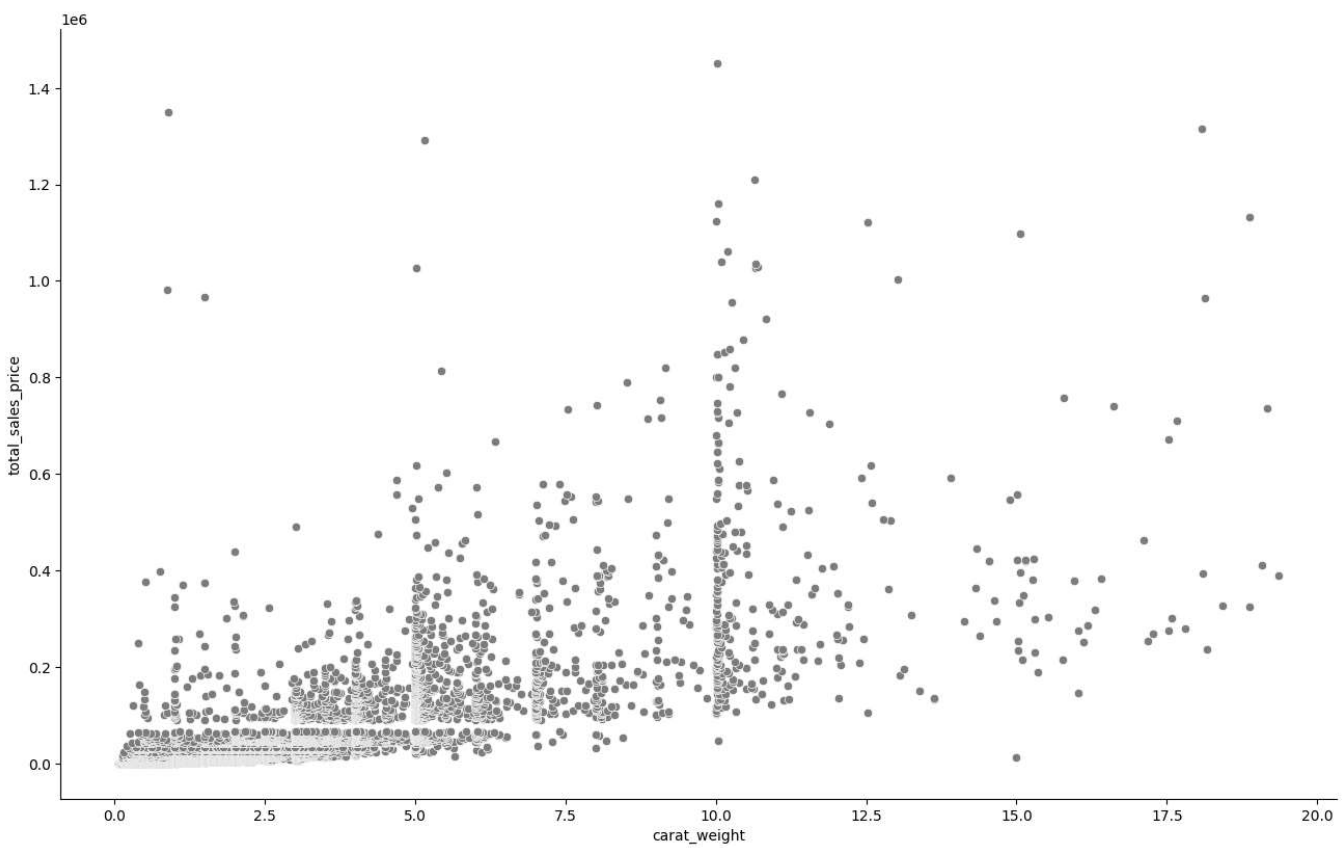






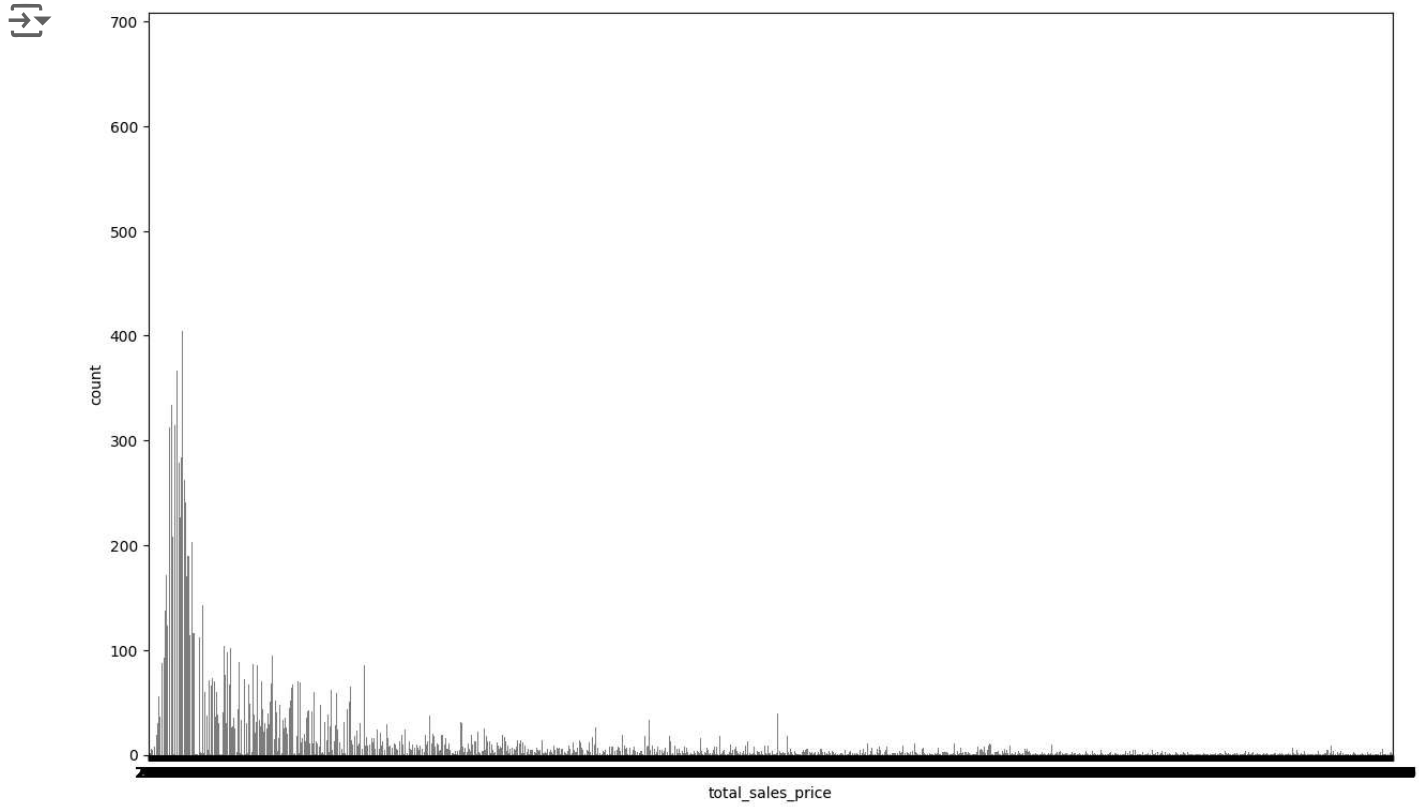


```
f = sns.relplot( data=df, x="carat_weight", y="total_sales_price",color='grey')
f.figure.set_size_inches(15, 9)
```



Based on the above graph, we can observe specific points where certain carat weights are more popular. For example, customers prefer to buy diamonds with a carat weight of 2.5 more than those with a carat weight of 2.2. We can see this preference for specific decimal values clearly in the graph. This pattern also applies to price points, as indicated by the visible blank line at $1000.

```
gap_analysis = df.loc[df['total_sales_price'].between(0, 25000)]


f = sns.countplot(data=gap_analysis, x='total_sales_price', color='grey')
f.figure.set_size_inches(15, 9)
```



In the above graph, we have considered prices between $0 and $25,000. Within this price range, there are some gaps, indicating that diamonds at certain prices are not being sold. These gaps suggest that customers prefer to buy diamonds at specific price points.

Key Attributes for Regression Model Predictions

- **Total sales price prediction**: Predicting the total_sales_price of a diamond is valuable beacuse it is depended on cut quality, color, clarity, Symmetry and Polish. Simply it relies on every attribute
- **carat_weight prediction**: carat weight is function of length, width and depth of the diamond.
- **table percent**: table percent is function of table width and diameter of diamond. So it can be predicted easily.

Which one does seem the most interesting to you and why?

Predicting the total_sales_price of a diamond is valuable beacuse it is depended on cut quality, color, clarity, Symmetry and Polish. Comparatively total_sales_price relies on every attribute. Total sales price seems most interesting because it allows buyers to make informed purchasing decisions based on price expectations. Sellers can use it to set competitive prices, and investors can assess the potential return on investment in diamonds. Prediction of total sales price helps assess risk in diamond transactions. It can be used to evaluate whether a seller is offering a fair price for a diamond based on its characteristics. If you are in the business of buying or selling diamonds, regression models can help you optimize pricing strategies. Utilizing regression analysis to predict total sales price can give your business a competitive advantage by offering pricing that reflects market dynamics and the value of individual diamonds accurately. And as per the corealtion figure we can see the most relation between the numeric attribute till now is two attributes one is Total_sales_price and second one is carat_weight but carat_weight is direct function of meas_length*meas_width*meas_depth/166 so that will be the and feature diriving from feature and that would be diract solution and that is not a problem. Anf in other hand Total_sales_price is depending on many condition(Attribute) so that would a problem to solve.

Preprocessing and Transformation

```
df['volume'] = (df['meas_length']*df['meas_width']*df['meas_depth'])
df['volume']
```

```
0          14.641305
1          14.609528
2          14.783040
3          14.661504
4          14.370174
```

```
              ...
219698    1726.551128
219699     581.023905
219700    2975.605200
219701      88.755660
219702    1279.319111
Name: volume, Length: 219703, dtype: float64
```

> **Now let us map the values of categorical attribute to numeric values.**

We have following categorical attributes...

cut, color, clarity, cut_quality, lab, symmetry, polish, eye_clean, culet_size, culet_condition, girdle_min, girdle_max, fluor_color, fluor_intensity, fancy_color_dominant_color, fancy_color_secondary_color, fancy_color_overtone, fancy_color_intensity

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
for cat in categorical_columns:
    le = preprocessing.LabelEncoder()
    le.fit(list(df[cat]))
    df[cat+'_pre'] = df[cat].apply(lambda x: le.transform([x])[0])
    print(df[[cat, cat + '_pre']])
```

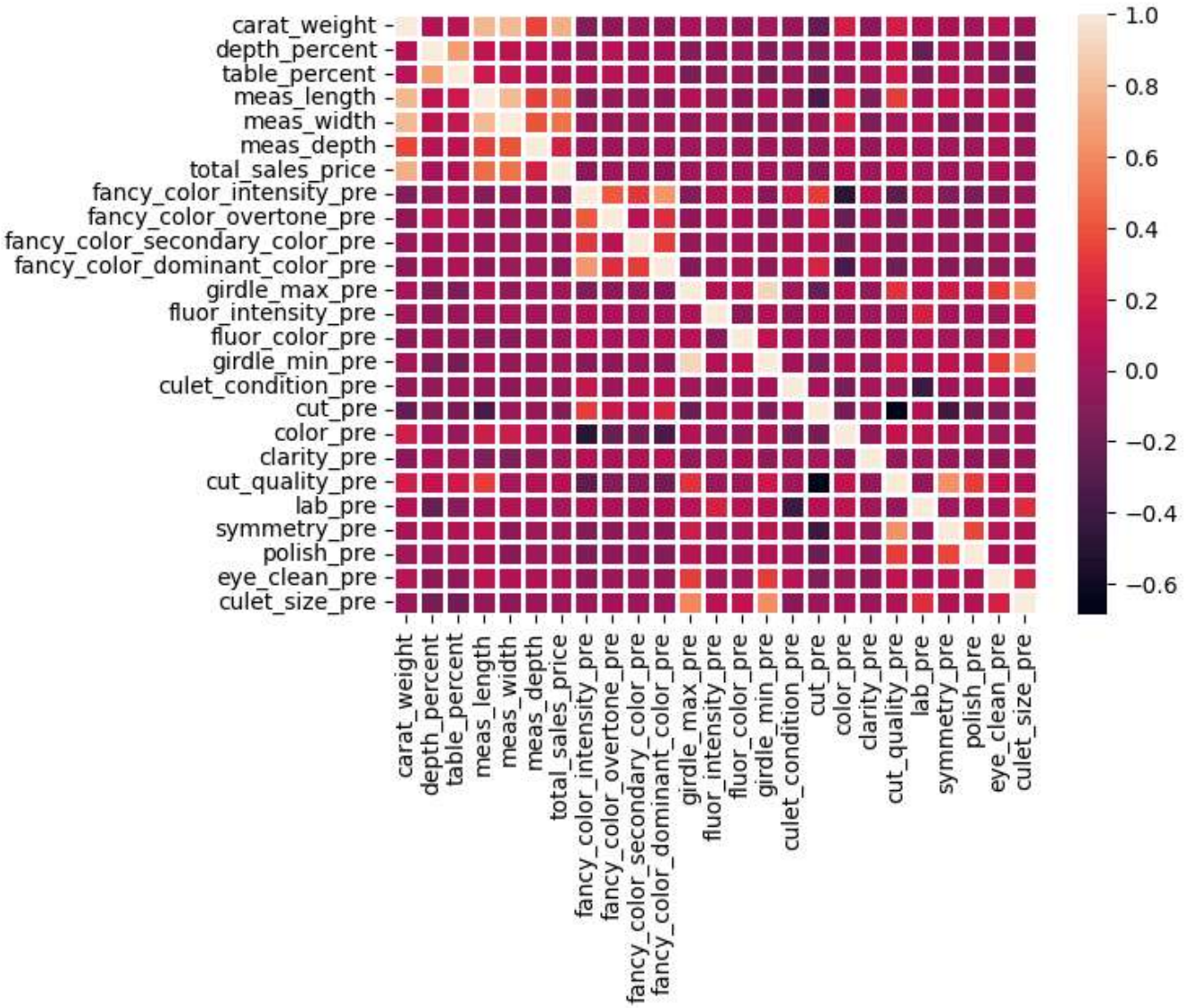
```
219700          M               0
219701        VTK               5
219702    unknown               9

[219703 rows x 2 columns]
       fluor_color  fluor_color_pre
0          unknown                5
1          unknown                5
2          unknown                5
3          unknown                5
4          unknown                5
...            ...              ...
219698     unknown                5
219699     unknown                5
219700     unknown                5
219701     unknown                5
219702     unknown                5

[219703 rows x 2 columns]
       fluor_intensity  fluor_intensity_pre
0                  NaN                    6
1                  NaN                    6
2                  NaN                    6
3                  NaN                    6
4                  NaN                    6
...                ...                  ...
219698             NaN                    6
219699             NaN                    6
219700             NaN                    6
219701           Faint                    0
219702             NaN                    6
```

Now Let us find the correlation among features

```
#select only numeric columns
data_df = df[['carat_weight', 'depth_percent', 'table_percent', 'meas_length',
       'meas_width', 'meas_depth','fancy_color_intensity_pre', 'fancy_color_overtone_pre',
       'fancy_color_secondary_color_pre', 'fancy_color_dominant_color_pre',
       'girdle_max_pre', 'fluor_intensity_pre', 'fluor_color_pre',
       'girdle_min_pre', 'culet_condition_pre', 'cut_pre', 'color_pre',
       'clarity_pre', 'cut_quality_pre', 'lab_pre', 'symmetry_pre',
       'polish_pre', 'eye_clean_pre', 'culet_size_pre']]
```

```
#heatmap
new_df = df[['carat_weight', 'depth_percent', 'table_percent', 'meas_length',
       'meas_width', 'meas_depth','total_sales_price',
       'fancy_color_intensity_pre', 'fancy_color_overtone_pre',
       'fancy_color_secondary_color_pre', 'fancy_color_dominant_color_pre',
       'girdle_max_pre', 'fluor_intensity_pre', 'fluor_color_pre',
       'girdle_min_pre', 'culet_condition_pre', 'cut_pre', 'color_pre',
       'clarity_pre', 'cut_quality_pre', 'lab_pre', 'symmetry_pre',
       'polish_pre', 'eye_clean_pre', 'culet_size_pre']]
sns.heatmap(data=new_df.corr(), lw = 1)
```

```
<Axes: >
```



Significant correlation of total_sales_price with carat_weight, meas_length, meas_width, meas_depth.

We can see that carat_weight is highly correlated with meas_length, meas_width, meas_depth, total_sales_price.

> **Let us now split the dataset into train-val-test sets**

Also We normalize the data to range [-1, 1].

We divide the dataset in two sets.

Training set which is 80% of the dataset

Test set which is of 20%

```
from sklearn.model_selection import train_test_split, cross_validate, cross_val_score, GridS
X_train, X_test, y_train, y_test = train_test_split(data_df, df["total_sales_price"], random
```

```
size_scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = size_scaler.transform(X_train)
X_test_scaled = size_scaler.transform(X_test)
X_train_scaled.shape, X_test_scaled.shape
```

```
((175762, 24), (43941, 24))
```

```
X_train_scaled
```

```
array([[-6.45174943e-01,  4.13723016e-02,  2.41819523e-02, ...,
        -4.97433847e-01,  5.97113628e-01,  1.23995298e+00],
       [-5.38654132e-01,  2.53673161e-01,  1.24922295e-01, ...,
        -4.97433847e-01, -1.41933084e+00,  1.23995298e+00],
       [-2.90105573e-01,  7.38932267e-01,  7.29364351e-01, ...,
         2.03335705e+00, -1.41933084e+00,  1.23995298e+00],
       ...,
       [ 1.12306380e-01,  1.49714962e+00,  5.27883666e-01, ...,
         7.67961603e-01, -1.41933084e+00, -8.09887836e-01],
       [-2.78269927e-01,  1.12139255e-01, -1.77298733e-01, ...,
        -4.97433847e-01,  5.97113628e-01, -8.09887836e-01],
       [-3.01941219e-01,  9.34042748e-04,  3.26402981e-01, ...,
        -4.97433847e-01, -1.41933084e+00, -8.09887836e-01]])
```

```
X_test_scaled
```

```
array([[ 0.28984107,  0.18290621, -0.17729873, ..., -0.49743385,
         0.59711363, -0.80988784],
       [-0.51498284,  0.02115317,  0.1249223 , ..., -0.49743385,
        -1.41933084,  1.23995298],
       [ 0.28984107, -0.23158595,  0.32640298, ..., -0.49743385,
         0.59711363,  1.23995298],
       ...,
```

```
[ 1.13017191,  0.71871314,  1.53528709, ...,  2.03335705,
  0.59711363, -0.80988784],
[-0.53865413,  0.09192013, -0.07655839, ..., -0.49743385,
 -1.41933084, -0.80988784],
[-0.00605008,  0.16268708, -5.81875793, ..., -0.49743385,
  0.59711363,  1.23995298]])
```

Regression Model building

```
score_type = ["neg_mean_absolute_error", "neg_mean_squared_error"]
degrees = [1,2]
```

```
lin_reg = LinearRegression() # Initialize linear regression model
```

```
# Prepare DataFrame to store model results
model_results = pd.DataFrame()
model_results['Train_MAE'] = None
model_results['Train_RMSE'] = None
model_results['Validation_MAE'] = None
model_results['Validation_RMSE'] = None
```

```
for degree in degrees:
    poly_features = PolynomialFeatures(degree=degree, include_bias=False)
    poly = poly_features.fit_transform(X_train_scaled)

    # Perform cross-validation
    score = cross_validate(lin_reg, poly, y_train, scoring=score_type, cv=5, return_train_sc

    # Calculate mean RMSE and MAE for train and validation sets
    train_rmse = np.sqrt(-score['train_neg_mean_squared_error']).mean()
    train_mae = (-score['train_neg_mean_absolute_error']).mean()
    validation_rmse = np.sqrt(-score['test_neg_mean_squared_error']).mean()
    validation_mae = (-score['test_neg_mean_absolute_error']).mean()

    # Store results in DataFrame
    model_results.loc["Polynomial_Regression " + str(degree) , ['Train_RMSE', 'Train_MAE', '
```

- Here we can see the result of the Linear Regression and Polynomial Regression of degree 2 using cross validation method.

```
model_results
```

	Train_MAE	Train_RMSE	Validation_MAE	Validation_RMSE	best_p
Polynomial_Regression 1	4593.17748	15842.631153	4595.405117	15828.428763	

- Now, let's evaluate different models like, Decision Tree, Random Forest and XGB using their predefined parameters.

```
# models to evaluate
models = {
    "DecisionTreeRegressor" : DecisionTreeRegressor(),
    "RandomForestRegressor" : RandomForestRegressor(),
    "XGBRegressor" : XGBRegressor()
}


for model_name, model in models.items():
    # Perform cross-validation
    model_score = cross_validate(model, X_train_scaled, y_train, scoring=score_type, cv=5, r

    # Calculate mean RMSE and MAE for train and validation sets
    train_rmse = np.sqrt(-model_score['train_neg_mean_squared_error']).mean()
    train_mae = (-model_score['train_neg_mean_absolute_error']).mean()
    validation_rmse = np.sqrt(-model_score['test_neg_mean_squared_error']).mean()
    validation_mae = (-model_score['test_neg_mean_absolute_error']).mean()

    # price_predict = best_model.predict(X_test_scaled)
    # price_r2 = r2_score(y_test, price_predict)
    # price_r2

    # Store results in DataFrame
    model_results.loc[model_name , ['Train_RMSE', 'Train_MAE', 'Validation_RMSE', 'Validatic
```

- Here we can see the result of all of the models we've used so far.

```
model_results
```

- As we can see above, Random Forest and XGB models have less RMSE values than all of the other models.
- We can also see that, Train RMSE is much lower than Validation_RMSE in DecisionTreeRegressor, RandomForestRegressor and XGBRegressor. This represents the overfitting in the model.
- So, now let's fine tune these two models to get the better parameters for these models.
- We've used different paramaters for both of these models to get the best score out of it!

Hyperparameter Tuning

```
model_dict = {
    "RandomForestRegressor": {'model': RandomForestRegressor(random_state=42),
                              'params': {'n_estimators': list(range(5,50, 5)), 'max_depth': ]
                                         'min_samples_split': [2, 5, 10]}},
```