Diamond Price Analysis & Prediction

Import library

In [1]:
```
from autoviz import AutoViz_Class
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

```
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
Imported v0.1.731. After importing autoviz, execute '%matplotlib inline' to display charts
    AV = AutoViz_Class()
    dfte = AV.AutoViz(filename, sep=',', depVar='', dfte=None, header=0, verbose=1, lowess=
               chart_format='svg',max_rows_analyzed=150000,max_cols_analyzed=30, save_plot_
```

Read DataSet

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

Out[2]:

	id	carat	cut	color	clarity	depth	table	x	y	z	price
0	0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

In [3]:
```
row, col = df.shape
print("This Dataset have",row,"rows and",col,"columns.")
```

```
This Dataset have 193573 rows and 11 columns.
```

In [4]:
```
print("Number of duplicate data : ",df.duplicated().sum())
```

```
Number of duplicate data :  0
```

In [5]:
```
print(df.isnull().sum())
```

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

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193573 entries, 0 to 193572
Data columns (total 11 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   id       193573 non-null  int64  
 1   carat    193573 non-null  float64
 2   cut      193573 non-null  object 
 3   color    193573 non-null  object 
 4   clarity  193573 non-null  object 
 5   depth    193573 non-null  float64
 6   table    193573 non-null  float64
 7   x        193573 non-null  float64
 8   y        193573 non-null  float64
 9   z        193573 non-null  float64
 10  price    193573 non-null  int64  
dtypes: float64(6), int64(2), object(3)
memory usage: 16.2+ MB
```

In [7]:
```
df.describe()
```

Out[7]:

	id	carat	depth	table	x	y	
count	193573.000000	193573.000000	193573.000000	193573.000000	193573.000000	193573.000000	193573.0
mean	96786.000000	0.790688	61.820574	57.227675	5.715312	5.720094	3.5
std	55879.856166	0.462688	1.081704	1.918844	1.109422	1.102333	0.6
min	0.000000	0.200000	52.100000	49.000000	0.000000	0.000000	0.0
25%	48393.000000	0.400000	61.300000	56.000000	4.700000	4.710000	2.9
50%	96786.000000	0.700000	61.900000	57.000000	5.700000	5.720000	3.5
75%	145179.000000	1.030000	62.400000	58.000000	6.510000	6.510000	4.0
max	193572.000000	3.500000	71.600000	79.000000	9.650000	10.010000	31.3

In [8]:
```
df.describe(include = "object")
```

Out[8]:

	cut	color	clarity
count	193573	193573	193573
unique	5	7	8
top	Ideal	G	SI1
freq	92454	44391	53272

Data Visualization

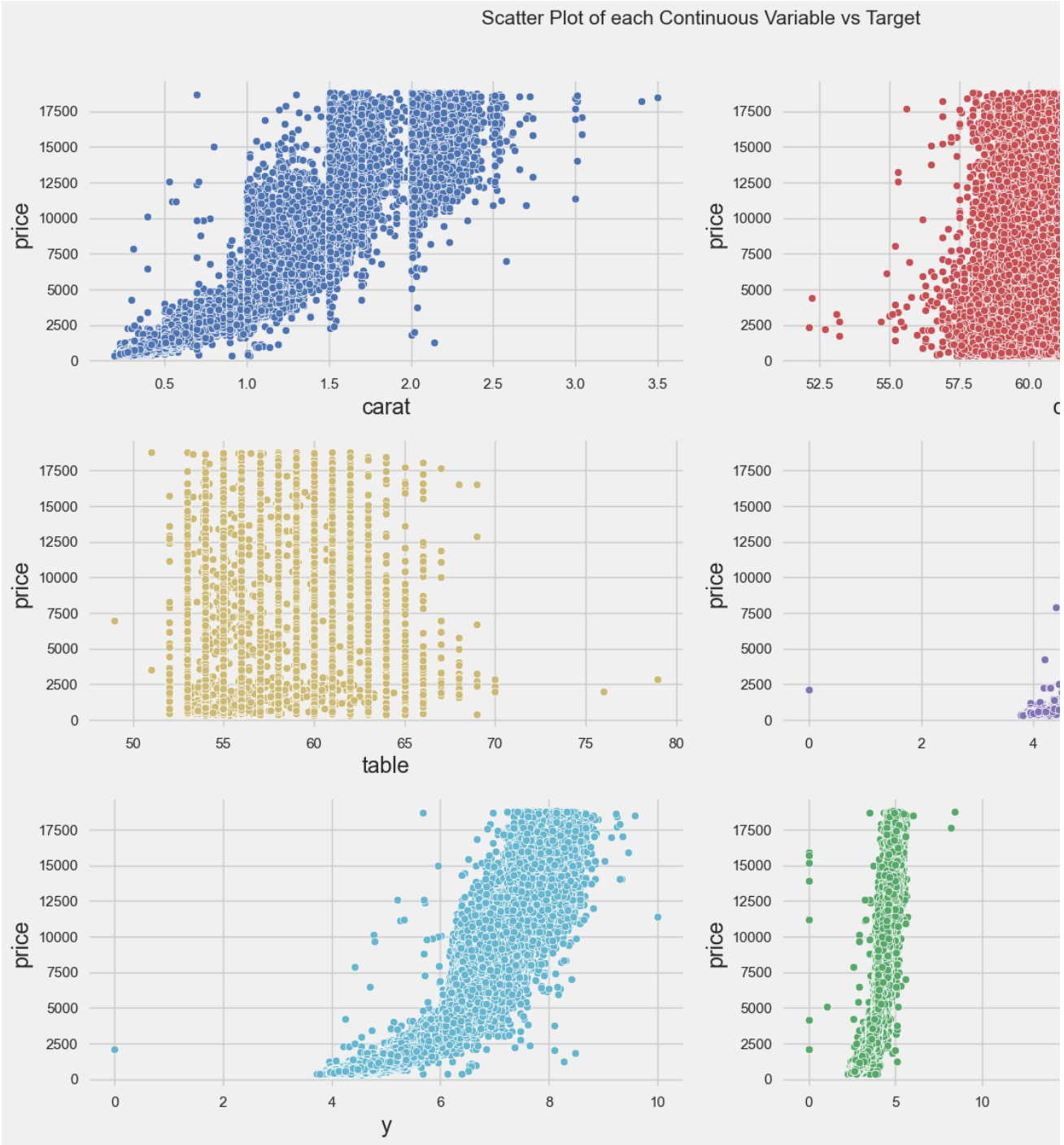
In [9]:
```
AV = AutoViz_Class()
%matplotlib inline
df = pd.read_csv("gemstone.csv")
df
filename = df
target_variable = "price"
dft = AV.AutoViz(
    "gemstone.csv",
    sep=",",
    depVar="price",
    dfte=None,
    header=0,
    verbose=1,
```

```
    lowess=False,
    chart_format="svg",
    max_rows_analyzed=150000,
    max_cols_analyzed=30,
    save_plot_dir=None)
```

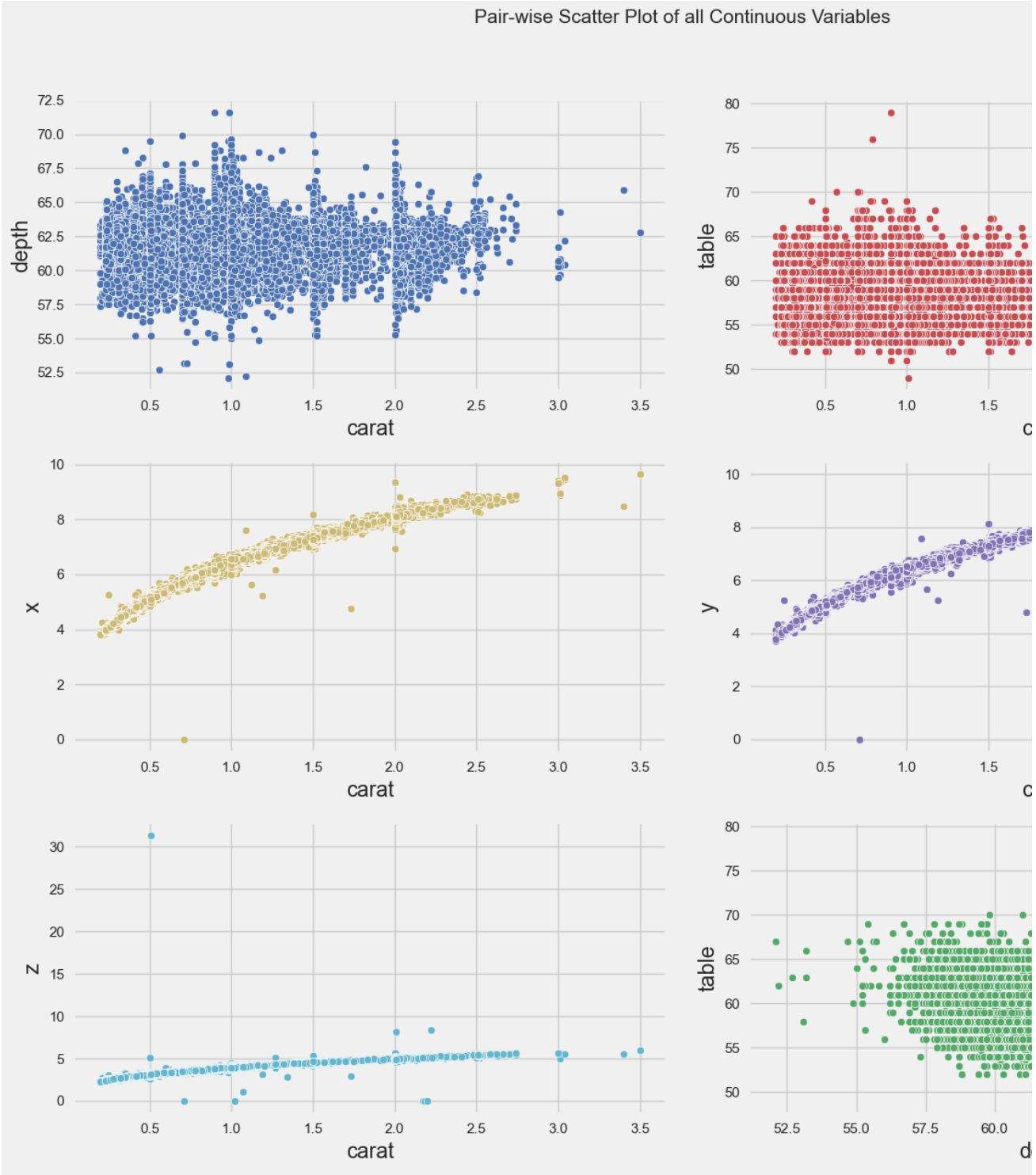
```
    max_rows_analyzed is smaller than dataset shape 193573...
        randomly sampled 150000 rows from read CSV file
Shape of your Data Set loaded: (150000, 11)
################################################################################################
######################## C L A S S I F Y I N G  V A R I A B L E S  ####################
################################################################################################
Classifying variables in data set...
    10 Predictors classified...
        1 variable(s) removed since they were ID or low-information variables
        List of variables removed: ['id']
Since Number of Rows in data 150000 exceeds maximum, randomly sampling 150000 rows for EDA.

################# Regression problem #####################
To fix data quality issues automatically, import FixDQ from autoviz...
```

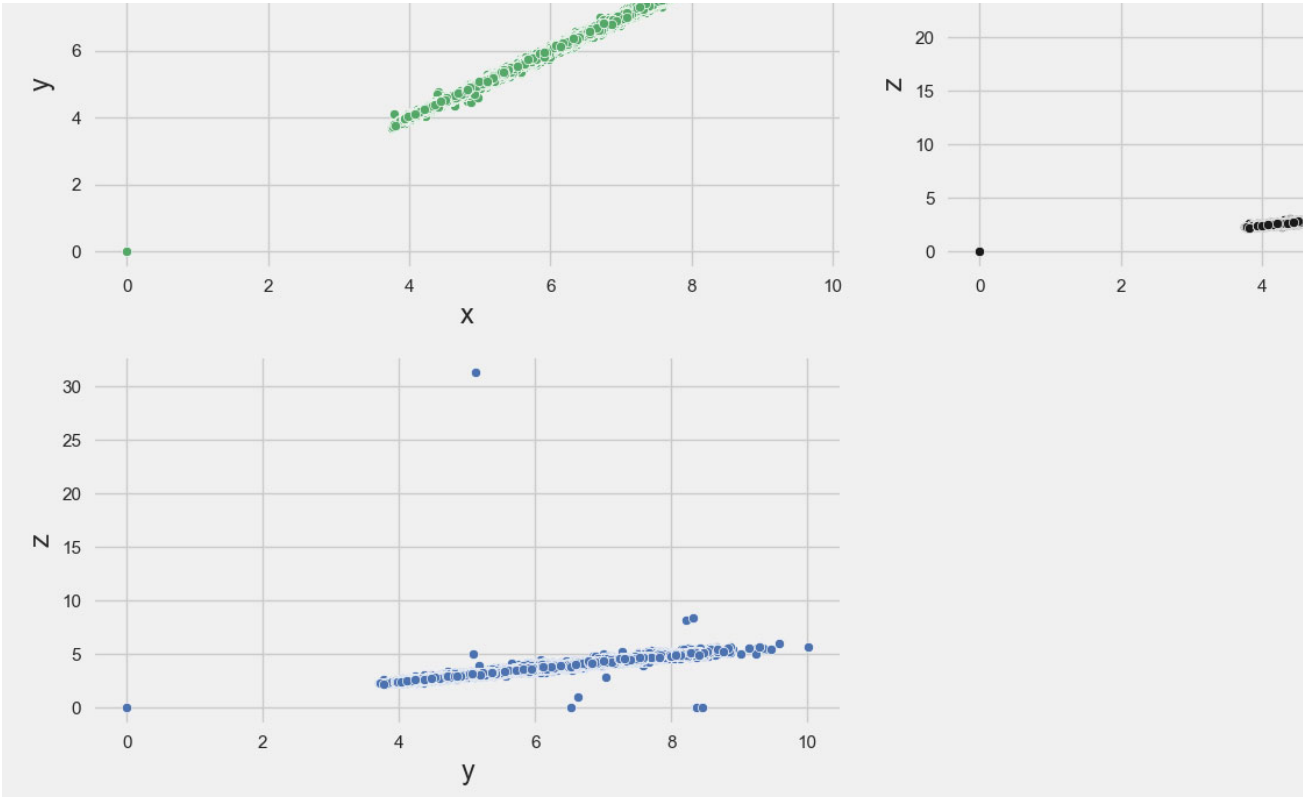
	Data Type	Missing Values%	Unique Values%	Minimum Value	Maximum Value	
carat	float64	0.000000	NA	0.200000	3.500000	has 5922 outliers greater than upper bound(-0.55). Cap them or remove them., price. Possible data l
cut	object	0.000000	0	nan	nan	
color	object	0.000000	0	nan	nan	
clarity	object	0.000000	0	nan	nan	1 rare categories: ['I1']. Group them
depth	float64	0.000000	NA	52.100000	71.600000	has 7752 outliers greater than upper bound
table	float64	0.000000	NA	49.000000	79.000000	has 3374 outliers greater than upper bound
x	float64	0.000000	NA	0.000000	9.650000	has 12 outliers greater than upper bound(1.99). Cap them or remove them., Consider dropping one of them., x has a cor data l
y	float64	0.000000	NA	0.000000	10.010000	has 11 outliers greater than upper bound(2.01). Cap them or remove them., 'x']. Consider dropping one of them., y Possible data l
z	float64	0.000000	NA	0.000000	31.300000	has 13 outliers greater than upper bound(1.23). Cap them or remove them., has 'y']. Consider dropping one of them., z Possible data l
price	int64	0.000000	5	326.000000	18818.000000	has 9720 outliers greater than upper bou bound(-5738.12). Cap them or remove ['carat', 'x', 'y', 'z

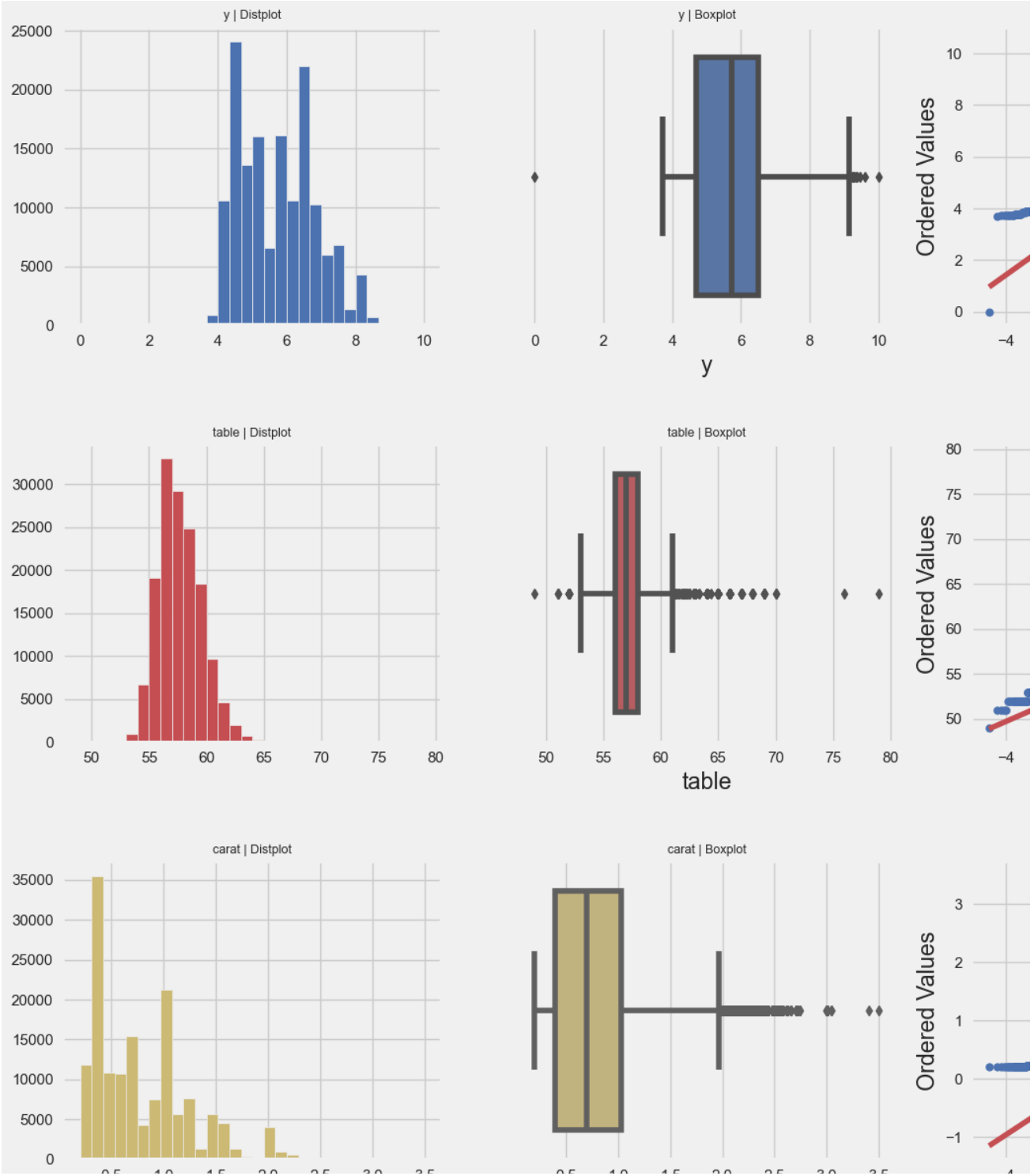


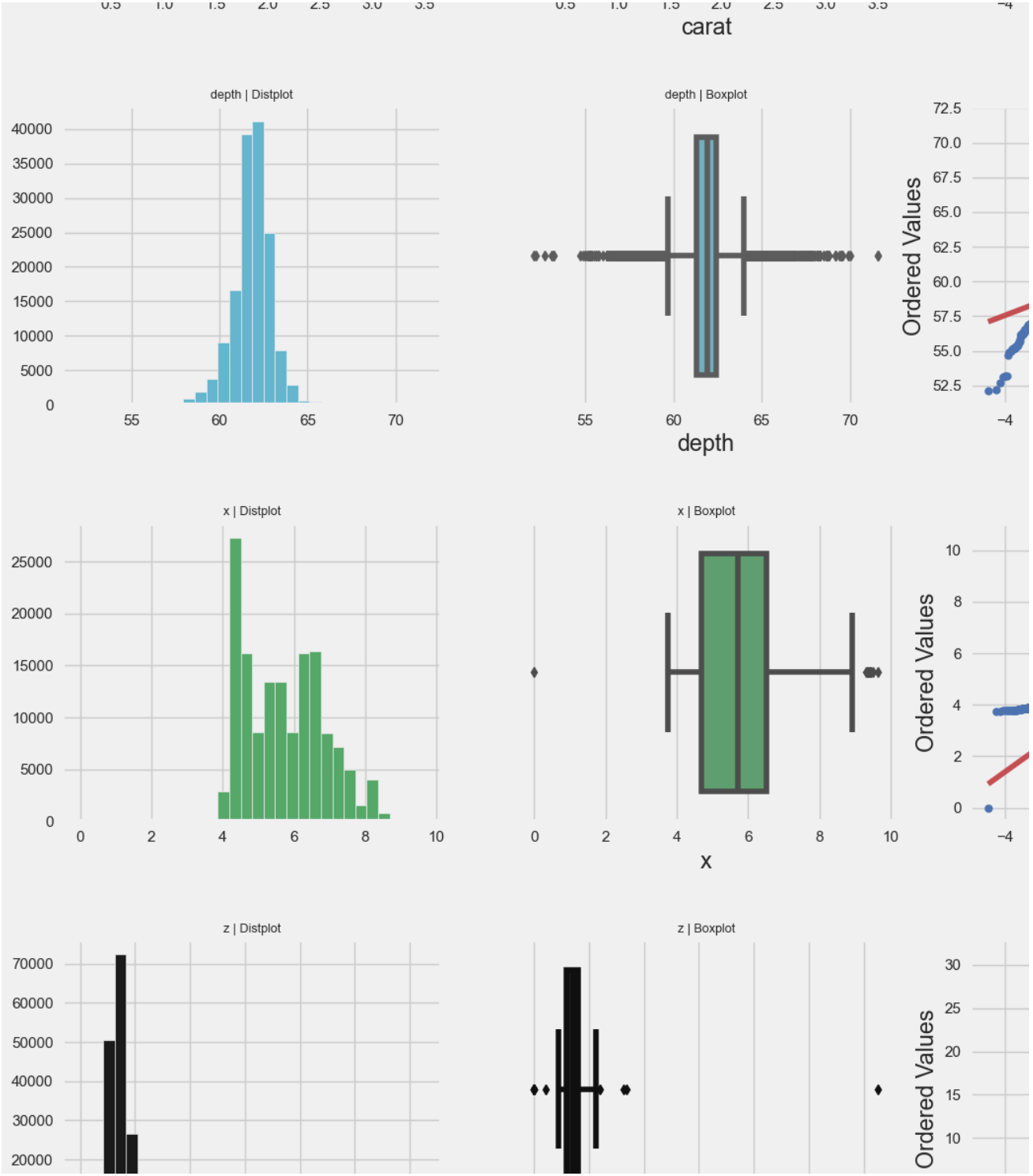
Number of All Scatter Plots = 21

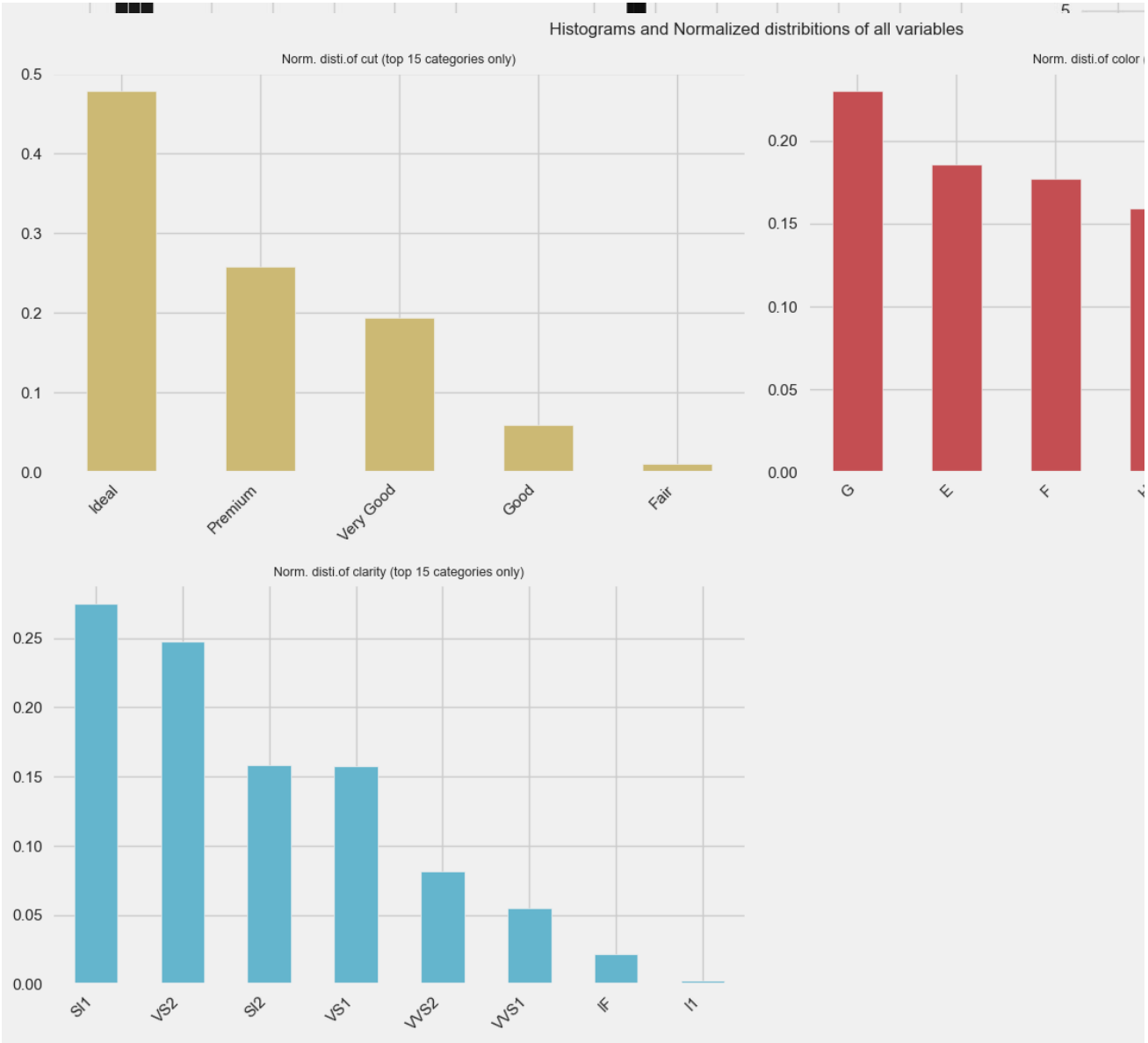

Pair-wise Scatter Plot of all Continuous Variables
depth
carat
table
carat
x
carat
y
carat
z
carat
table

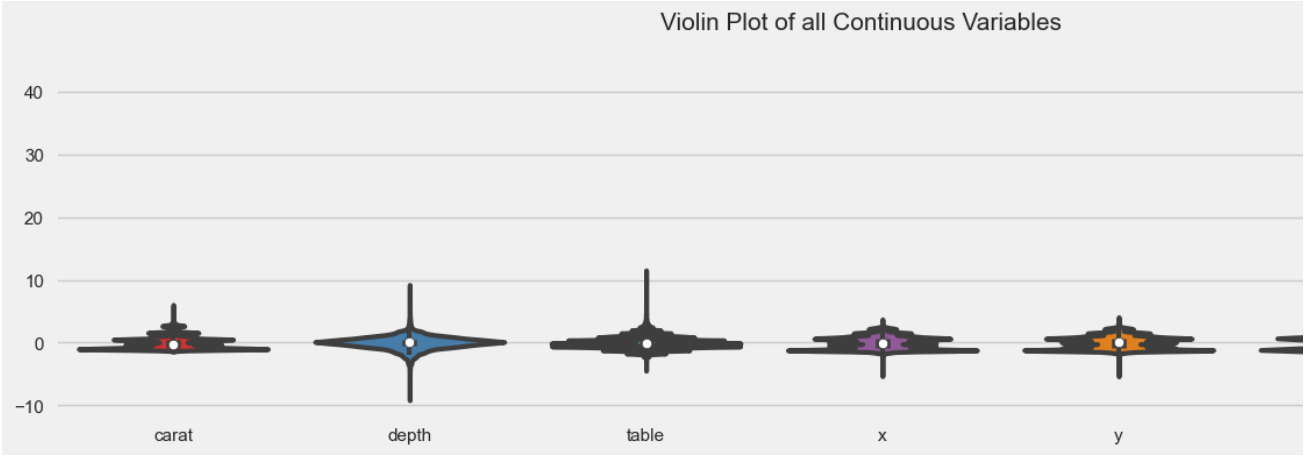

x
depth
y
z
depth
x
y
table
z

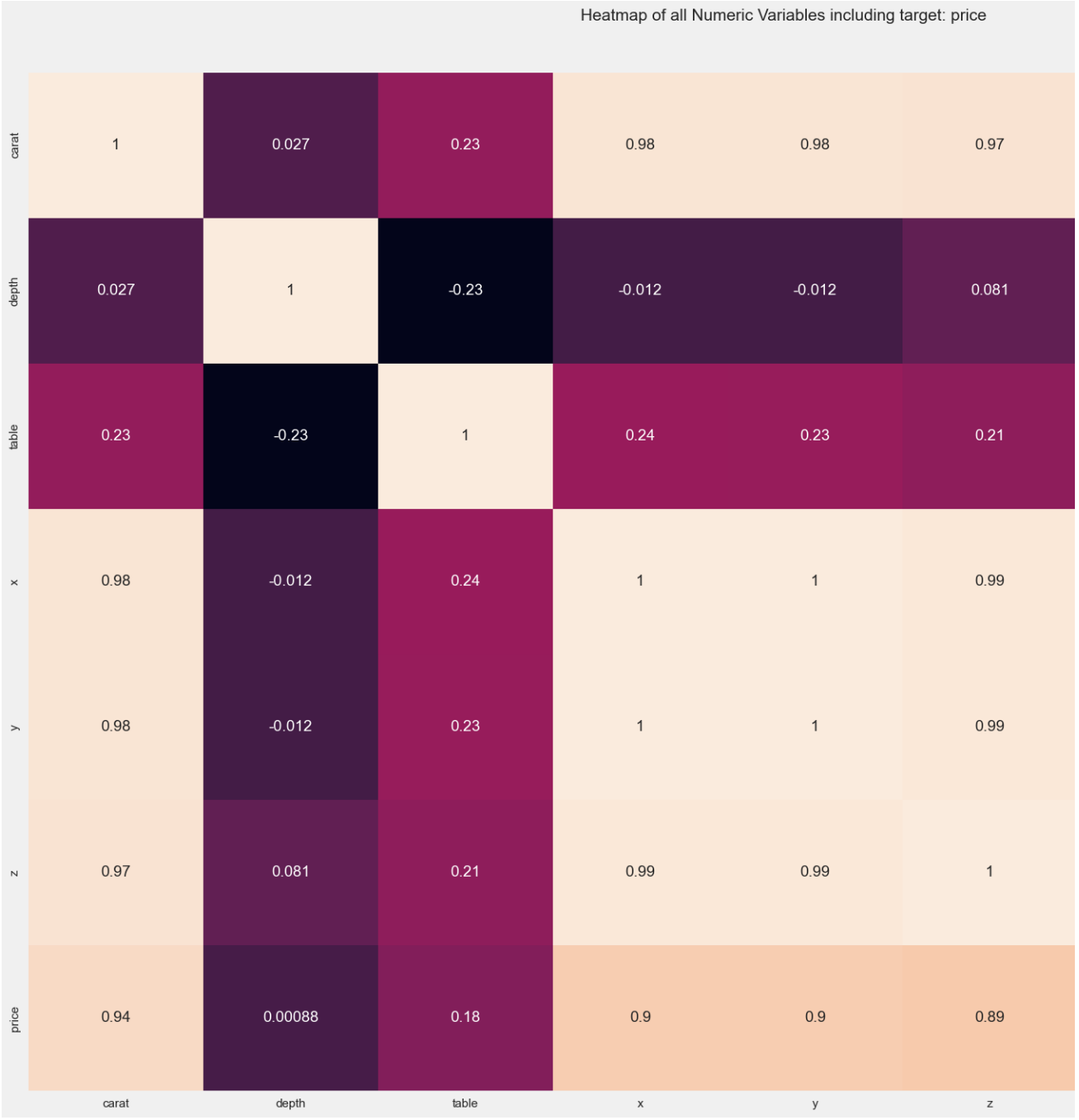

y
x
z
z
y

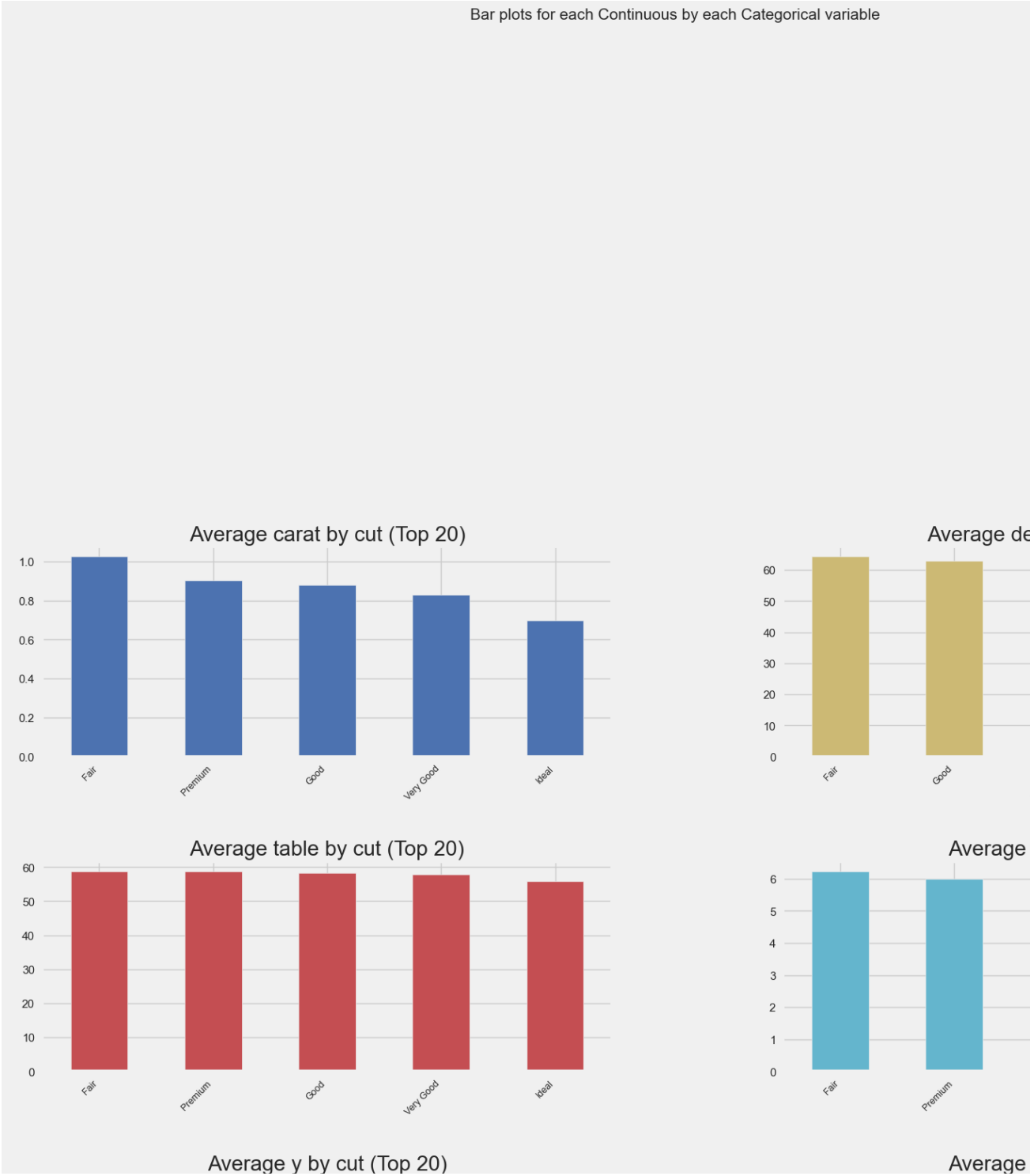

y | Distplot
y | Boxplot
Ordered Values
y
table | Distplot
table | Boxplot
Ordered Values
table
carat | Distplot
carat | Boxplot
Ordered Values

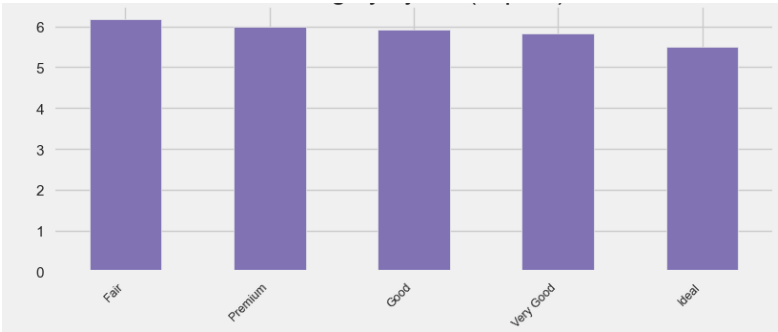

carat
depth | Distplot
depth | Boxplot
Ordered Values
depth
x | Distplot
x | Boxplot
Ordered Values
x
z | Distplot
z | Boxplot
Ordered Values

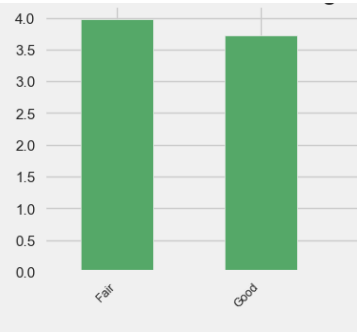

Histograms and Normalized distribitions of all variables
Norm. disti.of cut (top 15 categories only)
0.5
0.4
0.3
0.2
0.1
0.0
Ideal
Premium
Very Good
Good
Fair
Norm. disti.of color
0.20
0.15
0.10
0.05
0.00
G
E
F
Norm. disti.of clarity (top 15 categories only)
0.25
0.20
0.15
0.10
0.05
0.00
SI1
VS2
SI2
VS1
VVS2
VVS1
IF
I1

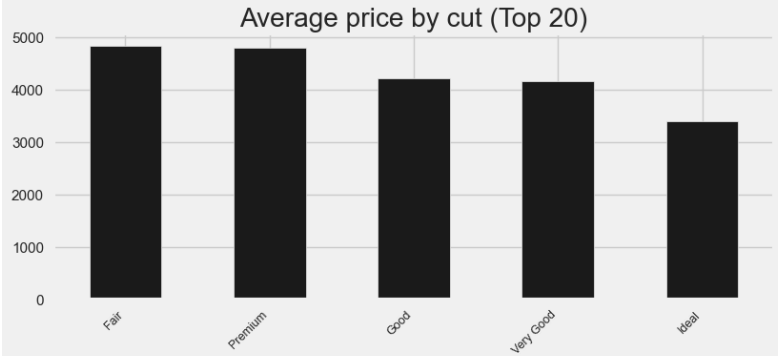

Violin Plot of all Continuous Variables
40
30
20
10
0
−10
carat
depth
table
x
y

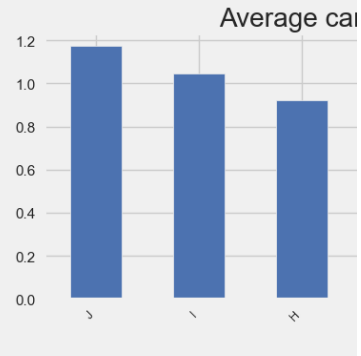

Heatmap of all Numeric Variables including target: price
carat
depth
table
x
y
z
price
1
0.027
0.23
0.98
0.98
0.97
0.027
1
-0.23
-0.012
-0.012
0.081
0.23
-0.23
1
0.24
0.23
0.21
0.98
-0.012
0.24
1
1
0.99
0.98
-0.012
0.23
1
1
0.99
0.97
0.081
0.21
0.99
0.99
1
0.94
0.00088
0.18
0.9
0.9
0.89
carat
depth
table
x
y
z

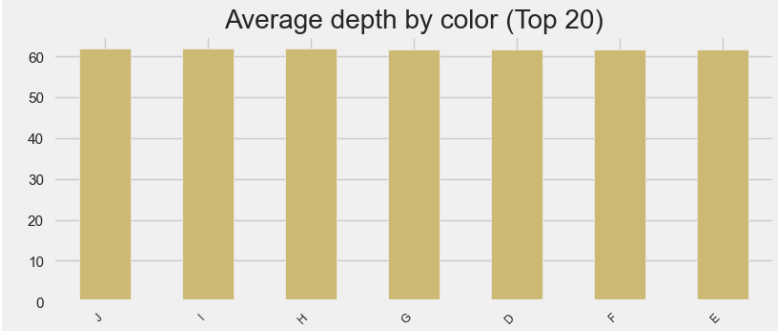

Bar plots for each Continuous by each Categorical variable
Average carat by cut (Top 20)
1.0
0.8
0.6
0.4
0.2
0.0
Fair
Premium
Good
Very Good
Ideal
Average de
60
50
40
30
20
10
0
Fair
Good
Average table by cut (Top 20)
60
50
40
30
20
10
0
Fair
Premium
Good
Very Good
Ideal
Average
6
5
4
3
2
1
0
Fair
Premium
Average y by cut (Top 20)
Average

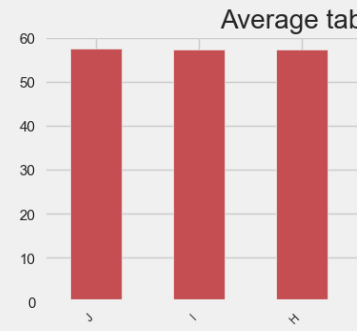

6
5
4
3
2
1
0
Fair
Premium
Good
Very Good
Ideal

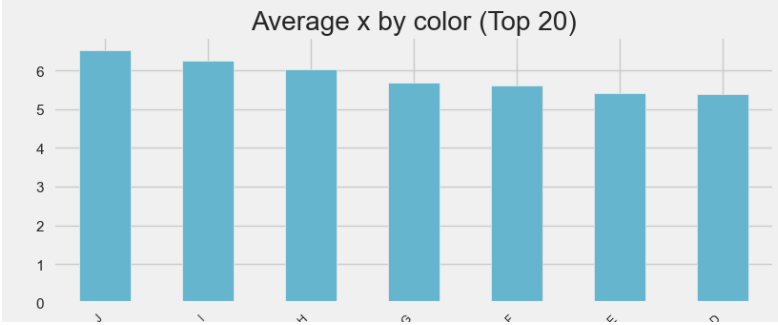

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Fair
Good

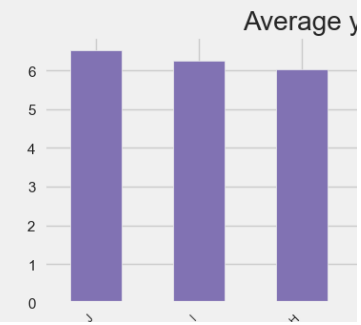

Average price by cut (Top 20)
5000
4000
3000
2000
1000
0
Fair
Premium
Good
Very Good
Ideal


Average ca
1.2
1.0
0.8
0.6
0.4
0.2
0.0
J
I
H


Average depth by color (Top 20)
60
50
40
30
20
10
0
J
I
H
G
D
F
E


Average tab
60
50
40
30
20
10
0
J
I
H


Average x by color (Top 20)
6
5
4
3
2
1
0

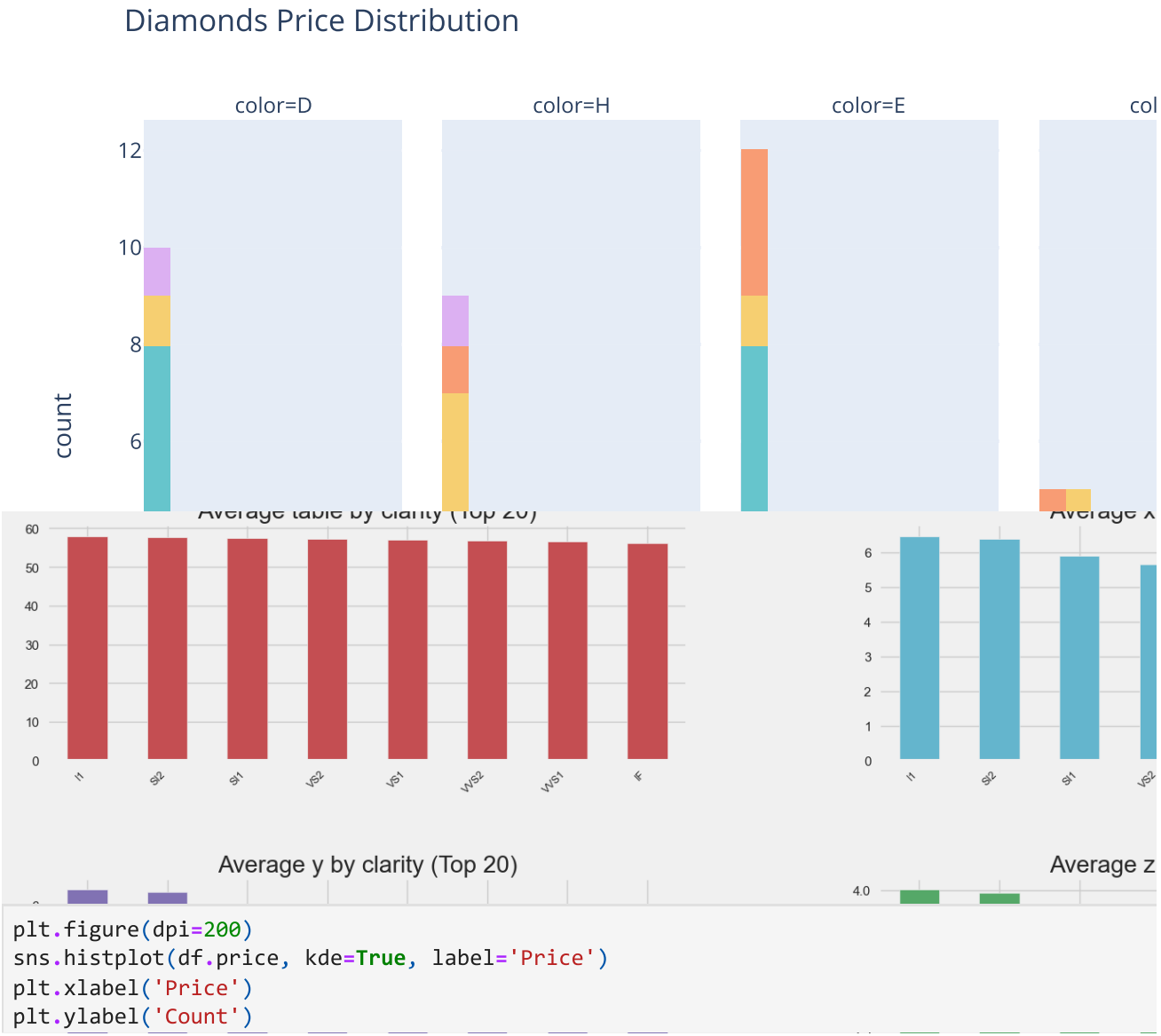

Average y
6
5
4
3
2
1
0

```
All Plots done
Time to run AutoViz = 32 seconds
```

Average z by color (Top 20) Average pri

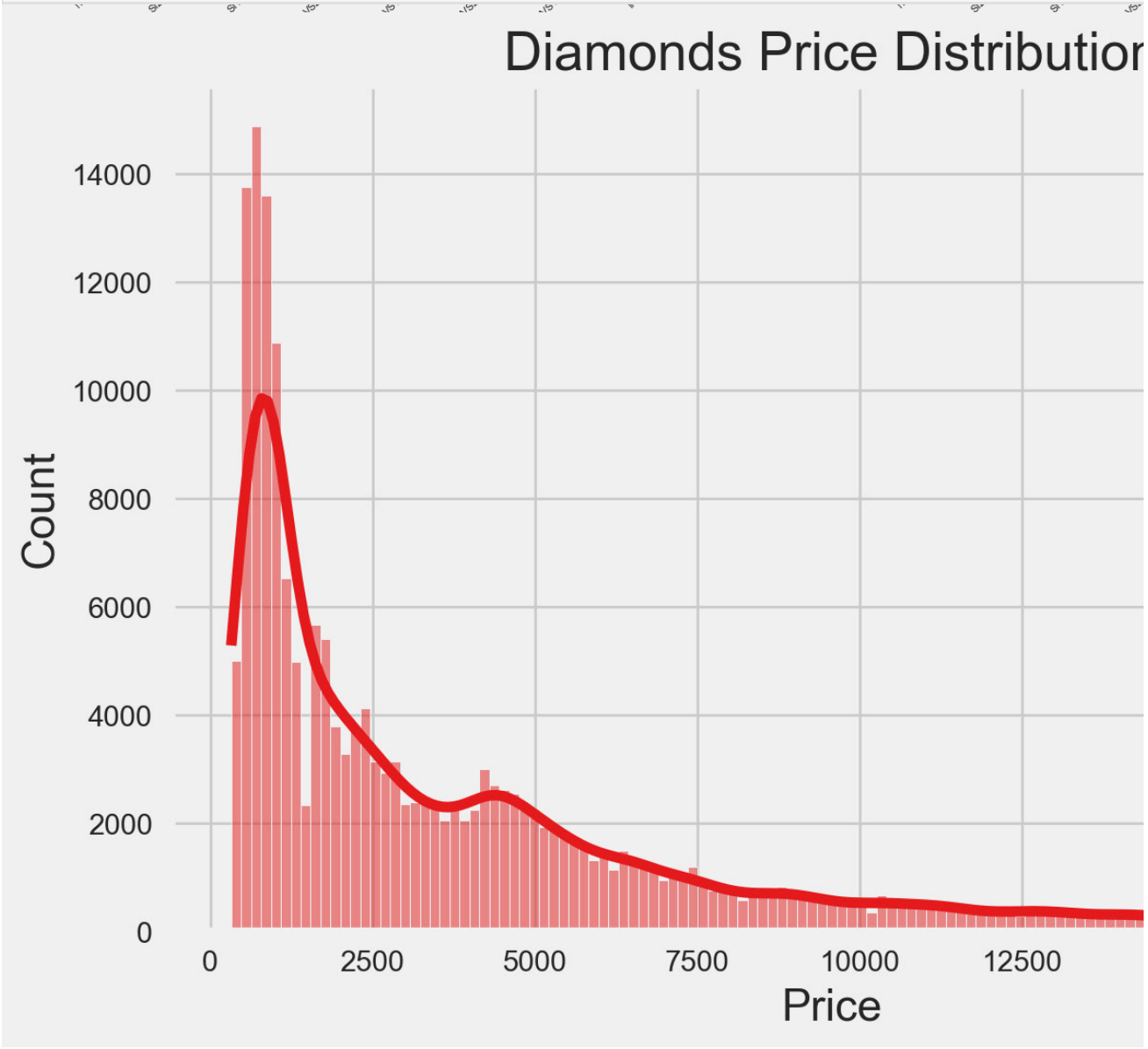
```
######################## AUTO VISUALIZATION Completed ########################
```

In [10]:
```
px.histogram(df.sample(n=100, replace=False, random_state=123).sort_index(), x='price', fa
```

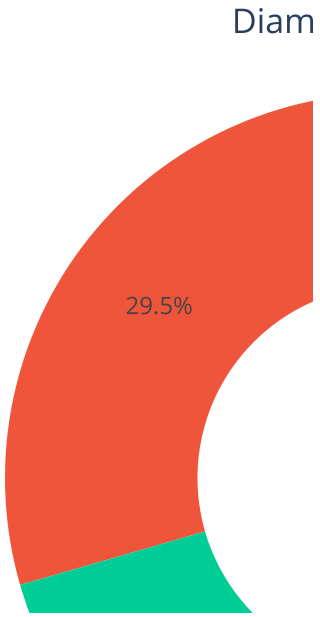


In [11]:
```
plt.figure(dpi=200)
sns.histplot(df.price, kde=True, label='Price')
plt.xlabel('Price')
plt.ylabel('Count')
```

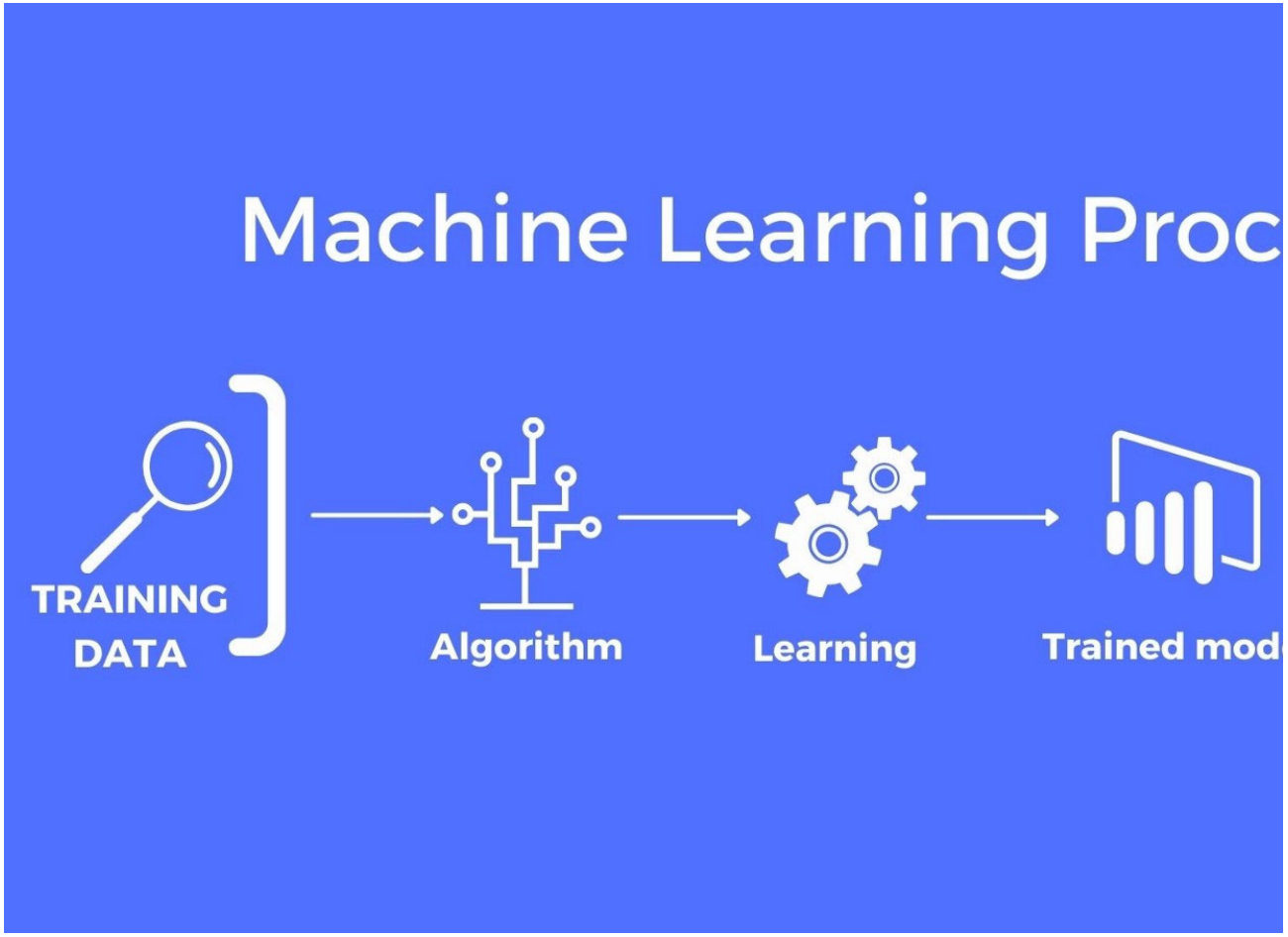
```
plt.title('Diamonds Price Distribution')
plt.legend()
plt.show();
```



In [12]:
```
fig = px.pie(df, names='cut', values='carat',hole=0.5)
fig.update_layout(title_text='Diamonds Cut Ratio', title_x=0.5)
fig.show()
```



Data Modeling



In [13]:
```
df.isnull().sum()
```

Out[13]:
```
id          0
carat       0
cut         0
color       0
clarity     0
depth       0
table       0
x           0
y           0
z           0
price       0
dtype: int64
```

In [14]:
```
df.to_excel("diamondclean_final.xlsx")
```

In [15]:
```
df = pd.read_excel("diamondclean_final.xlsx",index_col=0)
df
```

Out[15]:

	id	carat	cut	color	clarity	depth	table	x	y	z	price
0	0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453
...	...	...	...	...	...	...	...	...	...	...	...
193568	193568	0.31	Ideal	D	VVS2	61.1	56.0	4.35	4.39	2.67	1130
193569	193569	0.70	Premium	G	VVS2	60.3	58.0	5.75	5.77	3.47	2874
193570	193570	0.73	Very Good	F	SI1	63.1	57.0	5.72	5.75	3.62	3036
193571	193571	0.34	Very Good	D	SI1	62.9	55.0	4.45	4.49	2.81	681
193572	193572	0.71	Good	E	SI2	60.8	64.0	5.73	5.71	3.48	2258

193573 rows × 11 columns

In [16]:
```
df1 = df.copy()
```

In [17]:
```
from sklearn.preprocessing import LabelEncoder
```

In [18]:
```
# Encoding clarity column
encoder_clarity = LabelEncoder()
df1.clarity = encoder_clarity.fit_transform(df1.clarity)

# Encoding color column
encoder_color = LabelEncoder()
df1.color = encoder_color.fit_transform(df1.color)

# Encoding cut column
```

```
encoder_cut = LabelEncoder()
df1.cut = encoder_cut.fit_transform(df1.cut)
```

In [19]:
```
df1
```

Out[19]:

	id	carat	cut	color	clarity	depth	table	x	y	z	price
0	0	1.52	3	2	5	62.2	58.0	7.27	7.33	4.55	13619
1	1	2.03	4	6	3	62.0	58.0	8.06	8.12	5.05	13387
2	2	0.70	2	3	4	61.2	57.0	5.69	5.73	3.50	2772
3	3	0.32	2	3	4	61.6	56.0	4.38	4.41	2.71	666
4	4	1.70	3	3	5	62.6	59.0	7.65	7.61	4.77	14453
...	...	...	...	...	...	...	...	...	...	...	...
193568	193568	0.31	2	0	7	61.1	56.0	4.35	4.39	2.67	1130
193569	193569	0.70	3	3	7	60.3	58.0	5.75	5.77	3.47	2874
193570	193570	0.73	4	2	2	63.1	57.0	5.72	5.75	3.62	3036
193571	193571	0.34	4	0	2	62.9	55.0	4.45	4.49	2.81	681
193572	193572	0.71	1	1	3	60.8	64.0	5.73	5.71	3.48	2258

193573 rows × 11 columns

In [20]:
```
## Independent and dependent features
X = df1.drop(labels=['price'],axis=1)
Y = df1[['price']]
```

In [21]:
```
Y
```

Out[21]:

	price
0	13619
1	13387
2	2772
3	666
4	14453
...	...
193568	1130
193569	2874
193570	3036
193571	681
193572	2258

193573 rows × 1 columns

In [22]:
```
df1.drop(columns=['id'],inplace=True)
```

In [23]:
```
df1
```

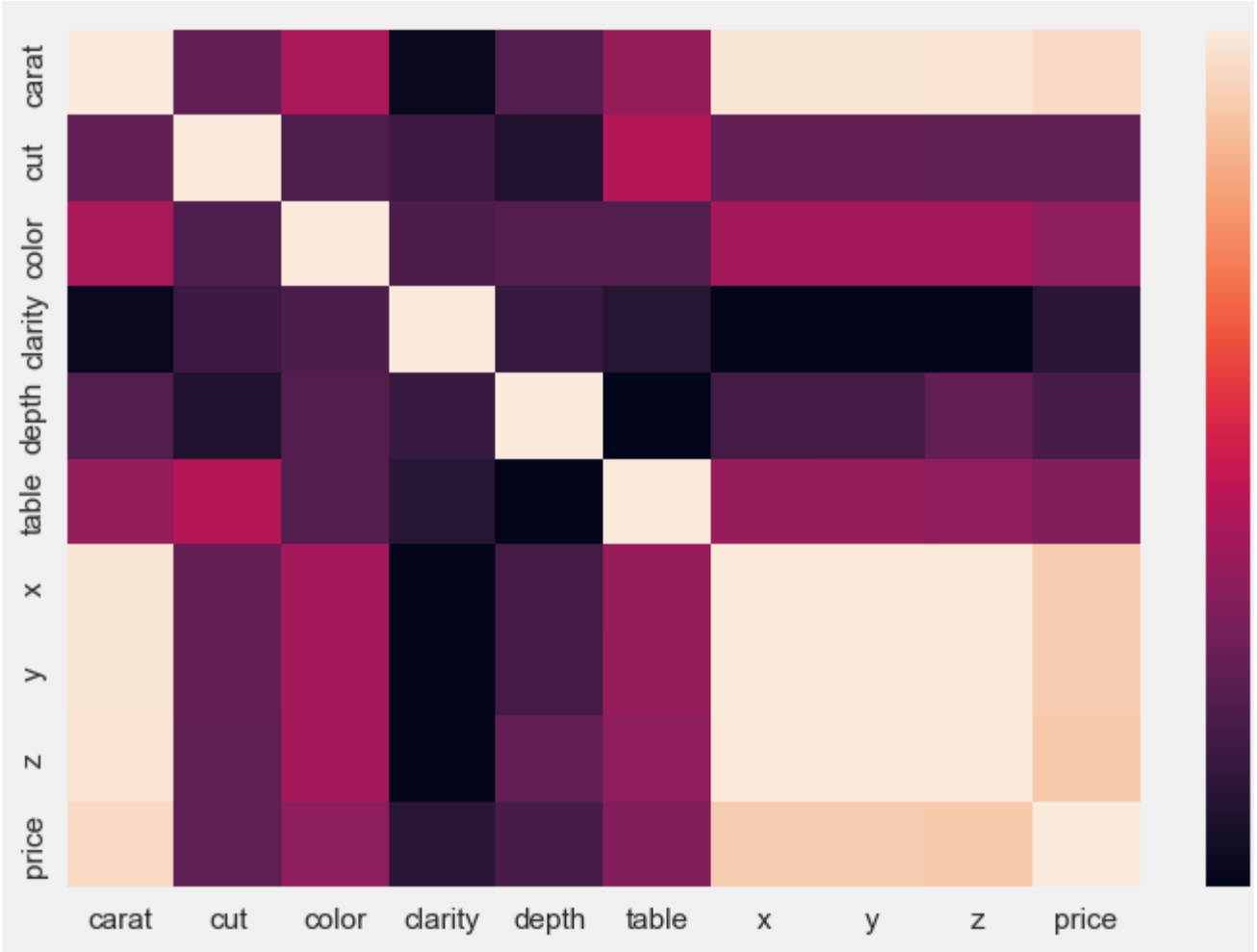
Out[23]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	3	2	5	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	4	6	3	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	2	3	4	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	2	3	4	61.6	56.0	4.38	4.41	2.71	666
4	1.70	3	3	5	62.6	59.0	7.65	7.61	4.77	14453
...	...	...	...	...	...	...	...	...	...	...
193568	0.31	2	0	7	61.1	56.0	4.35	4.39	2.67	1130
193569	0.70	3	3	7	60.3	58.0	5.75	5.77	3.47	2874
193570	0.73	4	2	2	63.1	57.0	5.72	5.75	3.62	3036
193571	0.34	4	0	2	62.9	55.0	4.45	4.49	2.81	681
193572	0.71	1	1	3	60.8	64.0	5.73	5.71	3.48	2258

193573 rows × 10 columns

In [24]:
```
# Visualizing heatmap to find coorelation
sns.heatmap(df1.corr())
```

Out[24]:
```
<AxesSubplot:>
```



In [25]:
```
from sklearn.model_selection import train_test_split
```

In [26]:
```
df1.columns
```

Out[26]:
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z',
       'price'],
      dtype='object')
```

In [27]:
```
# Splitting training and test data
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.1,random_state=10)
```

In [28]:
```
[len(x) for x in [X_train, X_test, y_train, y_test]]
```

Out[28]:
```
[174215, 19358, 174215, 19358]
```

Training and Testing of Models

In [29]:
```
## Model Training
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
import xgboost as xg
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
```

In [30]:
```
XG_Model = xg.XGBRegressor(objective ='reg:squarederror',n_estimators = 120, seed = 123)
XG_Model.fit(X_train, y_train)
```

Out[30]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=120, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In [31]:
```
regression=LinearRegression()
regression.fit(X_train,y_train)
```

Out[31]:
```
LinearRegression()
```

In [32]:
```
DecisionTree_Model = DecisionTreeRegressor(random_state=42)
DecisionTree_Model.fit(X_train, y_train)
```

Out[32]:
```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

In [33]:
```
def evaluate_model(true, predicted):
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return rmse, r2_square
```

In [34]:
```
## Train multiple models
## Model Ecaluation
models={
'LinearRegression':LinearRegression(),
'Lasso':Lasso(),
'Ridge':Ridge(),
'ElasticNet':ElasticNet(),
'DecisionTree_Model' :DecisionTreeRegressor(),
'XG_Model':xg.XGBRegressor()}

trained_model_list=[]
model_list=[]
r2_list=[]
for i in range(len(list(models))):
    model=list(models.values())[i]
    model.fit(X_train,y_train)
    #Make Predictions
    y_pred=model.predict(X_test)
    rmse, r2_square=evaluate_model(y_test,y_pred)
    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])
    print('Model Training Performance')
    print("RMSE:",rmse)
    print("R2 score",r2_square*100)
    r2_list.append(r2_square)
    print('='*35)
    print('\n')
```

```
LinearRegression
Model Training Performance
RMSE: 1121.0799792738976
R2 score 92.34722049041618
==================================


Lasso
Model Training Performance
RMSE: 1120.7553031425211
R2 score 92.35165249365922
==================================


Ridge
Model Training Performance
RMSE: 1121.0561372736738
R2 score 92.34754599017353
==================================


ElasticNet
Model Training Performance
RMSE: 1731.7482520250117
R2 score 81.73937983327164
==================================


DecisionTree_Model
Model Training Performance
RMSE: 861.0427724913033
R2 score 95.4856475962063
==================================


XG_Model
Model Training Performance
RMSE: 602.6214356192735
R2 score 97.78876368303487
==================================
```

In [35]:
```
model_list
```

Out[35]:
```
['LinearRegression',
 'Lasso',
 'Ridge',
 'ElasticNet',
 'DecisionTree_Model',
 'XG_Model']
```