Context This classic dataset contains the prices and other attributes of almost 54,000 diamonds. It's a great dataset for beginners learning to work with data analysis and visualization.

Content price price in US dollars (USD 326 -- USD 18,823)

carat weight of the diamond (0.2--5.01)

cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)

color diamond colour, from J (worst) to D (best)

clarity a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x length in mm (0--10.74)

y width in mm (0--58.9)

z depth in mm (0--31.8)

depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)

table width of top of diamond relative to widest point (43--95)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
arqPath = "diamonds.csv"
data = pd.read_csv(arqPath, index_col = 0)
```

In [3]:
```
data
```

Out[3]:

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [4]:
```
data.describe()
```

Out[4]:

	carat	depth	table	price	x	y	
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.0(
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.5:
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.7(
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.0(
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.9
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.5:
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.0
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.8(

In [5]:
```
data['volume'] = data['x'] * data['y'] * data['z']
```

In [6]:
```
data
```

Out[6]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	38.202030
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	34.505856
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	38.076885
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	46.724580
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	51.917250
...	...	...	...	...	...	...	...	...	...	...	...
53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50	115.920000
53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61	118.110175
53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56	114.449728
53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74	140.766120
53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64	124.568444

53940 rows × 11 columns

This is a block of text!

In [7]:
```
dataSorted = data.sort_values('price')
dataSorted = dataSorted.reset_index()
dataSorted
```

Out[7]:

	index	carat	cut	color	clarity	depth	table	price	x	y	z	volume
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	38.202030
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	34.505856
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	38.076885
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	46.724580
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	51.917250

	index	carat	cut	color	clarity	depth	table	price	x	y	z	volume
...	...	...	...	...	...	...	...	...	...	...	...	...
53935	27746	2.00	Very Good	H	SI1	62.8	57.0	18803	7.95	8.00	5.01	318.636000
53936	27747	2.07	Ideal	G	SI2	62.5	55.0	18804	8.20	8.13	5.11	340.663260
53937	27748	1.51	Ideal	G	IF	61.7	55.0	18806	7.37	7.41	4.56	249.029352
53938	27749	2.00	Very Good	G	SI1	63.5	56.0	18818	7.90	7.97	5.04	317.333520
53939	27750	2.29	Premium	I	VS2	60.8	60.0	18823	8.50	8.47	5.16	371.494200

53940 rows × 12 columns

In [8]:
```
dataSorted.tail(1)
```

Out[8]:

	index	carat	cut	color	clarity	depth	table	price	x	y	z	volume
53939	27750	2.29	Premium	I	VS2	60.8	60.0	18823	8.5	8.47	5.16	371.4942

In [9]:
```
data = data.sample(frac=1).reset_index(drop=True)
data
```

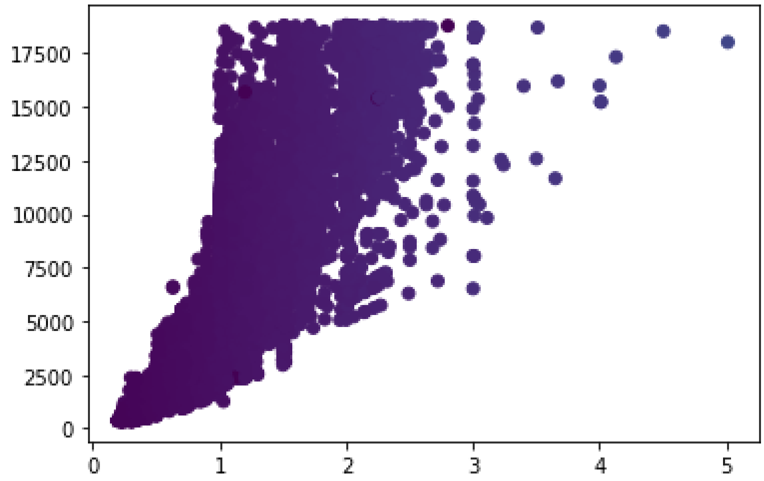
Out[9]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
0	0.31	Premium	H	VVS2	62.6	60.0	802	4.33	4.29	2.70	50.154390
1	0.58	Very Good	D	VS2	61.0	58.0	1750	5.37	5.42	3.29	95.756766
2	1.22	Ideal	F	SI2	62.5	56.0	6518	6.85	6.81	4.27	199.189095
3	0.30	Good	H	SI1	63.4	58.0	421	4.21	4.24	2.68	47.839072
4	0.30	Premium	D	SI2	62.3	58.0	447	4.29	4.32	2.68	49.667904
...	...	...	...	...	...	...	...	...	...	...	...
53935	0.71	Premium	E	SI1	59.1	60.0	2441	5.87	5.84	3.46	118.611568
53936	1.51	Ideal	H	SI2	61.9	56.0	8951	7.31	7.36	4.54	244.259264
53937	0.33	Premium	G	VVS2	59.5	60.0	752	4.52	4.55	2.70	55.528200
53938	0.32	Very Good	E	SI1	63.3	56.0	720	4.35	4.34	2.75	51.917250
53939	0.52	Premium	E	VS2	61.6	59.0	1694	5.16	5.14	3.17	84.076008

53940 rows × 11 columns

In [10]:
```
import itertools
plt.scatter(data['carat'], data['price'], c=list(data['volume']))
##plt.viridis()
```

Out[10]: <matplotlib.collections.PathCollection at 0x221c20b5220>



In [11]:
```
data['color']
```

Out[11]:
```
0        H
1        D
2        F
3        H
4        D
        ..
53935    E
53936    H
53937    G
53938    E
53939    E
Name: color, Length: 53940, dtype: object
```

In [12]:
```
## color diamond colour, from J (worst) to D (best)
colorsDict = {'J':1, 'I':2, 'H':3, 'G':4, 'F':5, 'E':6, 'D':7}
colorsDict
```

Out[12]:
```
{'J': 1, 'I': 2, 'H': 3, 'G': 4, 'F': 5, 'E': 6, 'D': 7}
```

In [13]:
```
## cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)
cutDict =  {'Fair':1, 'Good':2, 'Very Good':3, 'Premium':4, 'Ideal':5}
cutDict
```

Out[13]:
```
{'Fair': 1, 'Good': 2, 'Very Good': 3, 'Premium': 4, 'Ideal': 5}
```

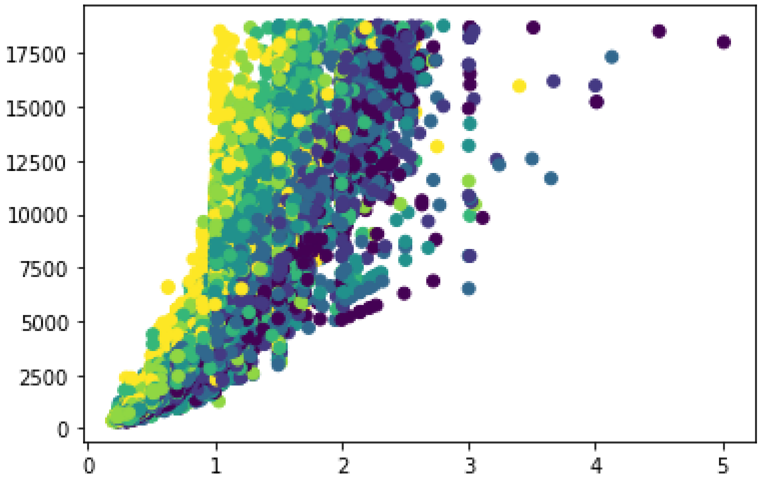
In [14]:
```
## clarity a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1
clarityDict = {'I1':1, 'SI2':2, 'SI1':3, 'VS2':4, 'VS1':5, 'VVS2':6, 'VVS1':7, 'IF':
clarityDict
```

Out[14]:
```
{'I1': 1,
 'SI2': 2,
 'SI1': 3,
 'VS2': 4,
 'VS1': 5,
 'VVS2': 6,
 'VVS1': 7,
 'IF': 8}
```

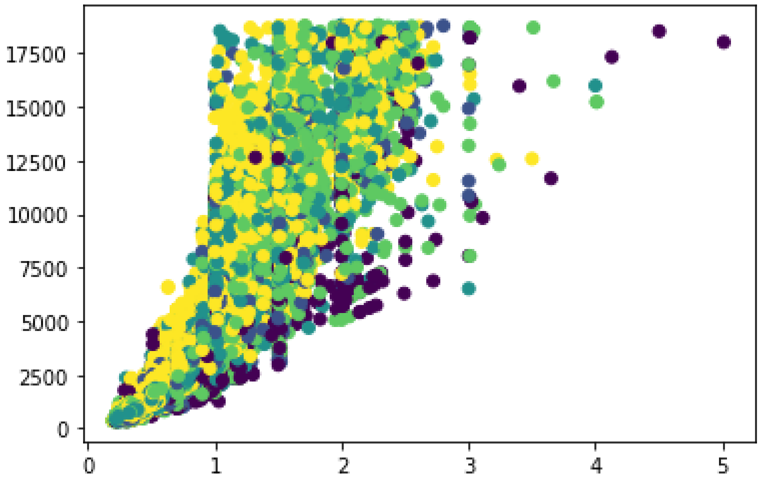
In [15]:
```
color = [colorsDict[i] for i in data['color']]
cut = [cutDict[i] for i in data['cut']]
clarity = [clarityDict[i] for i in data['clarity']]
```

In [16]:
```
data['cut'] = cut
data['color'] = color
data['clarity'] = clarity
```

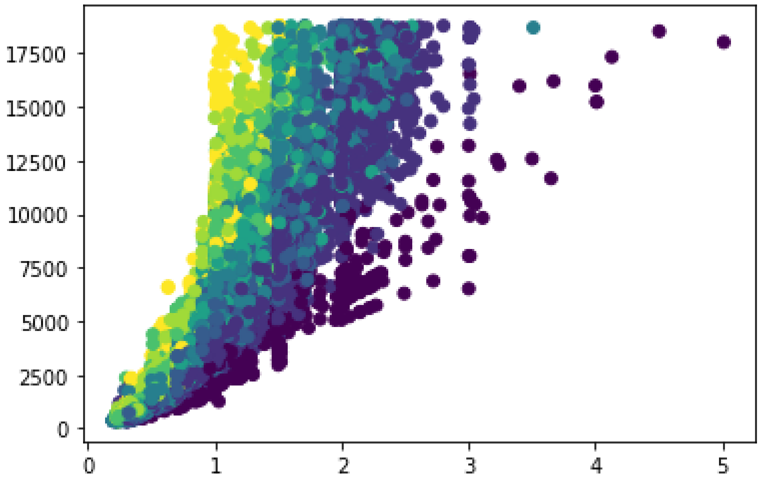
In [17]:
```
plt.scatter(data['carat'], data['price'], c=color, cmap='viridis')
plt.show()
```



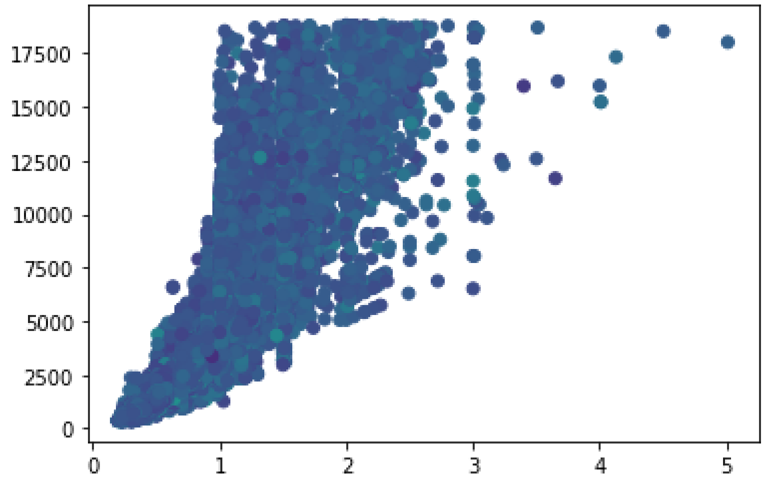
In [18]:
```
plt.scatter(data['carat'], data['price'], c=cut, cmap='viridis')
plt.show()
```



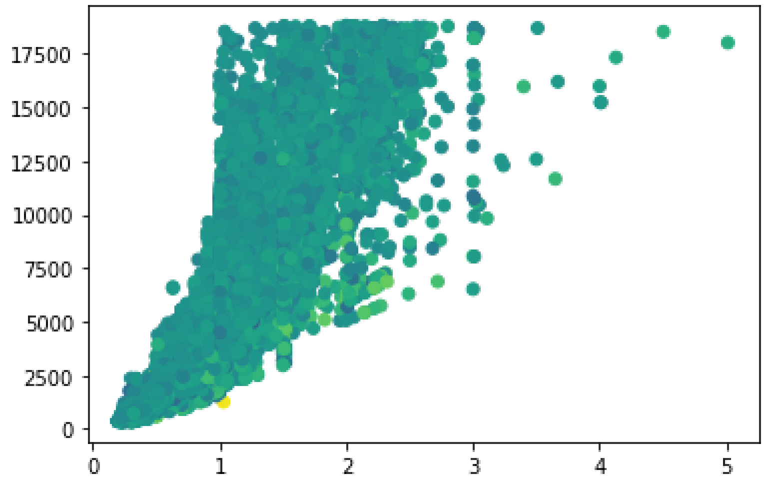
In [19]:
```
plt.scatter(data['carat'], data['price'], c=clarity, cmap='viridis')
plt.show()
```



In [20]:
```
plt.scatter(data['carat'], data['price'], c=data['table'], cmap='viridis')
plt.show()
```



In [21]:
```
plt.scatter(data['carat'], data['price'], c=data['depth'], cmap='viridis')
plt.show()
```



In [22]:
```
data['table']
```

Out[22]:
```
0        60.0
1        58.0
2        56.0
3        58.0
4        58.0
         ...
53935    60.0
53936    56.0
53937    60.0
53938    56.0
53939    59.0
Name: table, Length: 53940, dtype: float64
```

In [23]:
```
(data['table']).mean()
```

Out[23]:
```
57.45718390804598
```

In [24]:
```
(data['table']).std()
```

Out[24]:
```
2.234490562821323
```

In [25]:
```
data.mean()
```

Out[25]:
```
carat        0.797940
cut          3.904097
color        4.405803
```

```
clarity       4.051020
depth        61.749405
table        57.457184
price      3932.799722
x             5.731157
y             5.734526
z             3.538734
volume      129.849403
dtype: float64
```

In [26]:
```
data.std()
```

Out[26]:
```
carat         0.474011
cut           1.116600
color         1.701105
clarity       1.647136
depth         1.432621
table         2.234491
price      3989.439738
x             1.121761
y             1.142135
z             0.705699
volume       78.245262
dtype: float64
```

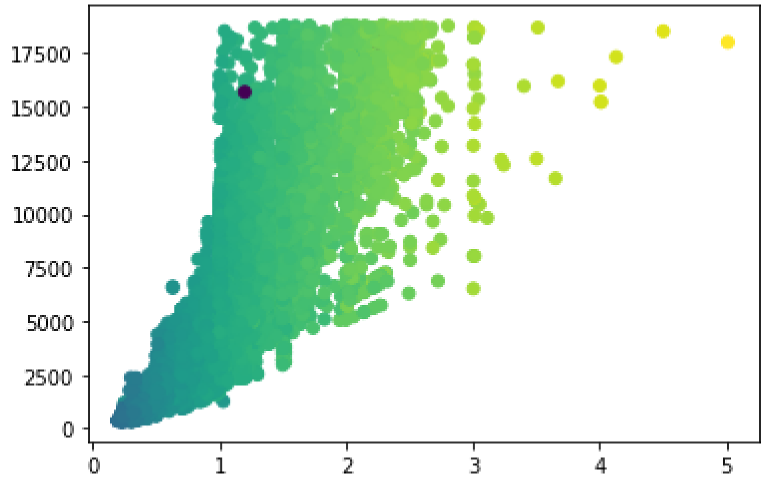
In [27]:
```
data.max()
```

Out[27]:
```
carat          5.01000
cut            5.00000
color          7.00000
clarity        8.00000
depth         79.00000
table         95.00000
price      18823.00000
x             10.74000
y             58.90000
z             31.80000
volume      3840.59806
dtype: float64
```

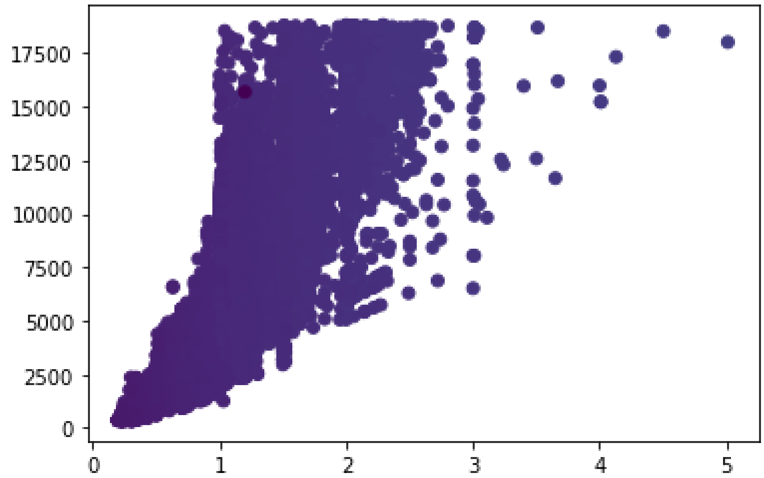
In [28]:
```
data.min()
```

Out[28]:
```
carat        0.2
cut          1.0
color        1.0
clarity      1.0
depth       43.0
table       43.0
price      326.0
x            0.0
y            0.0
z            0.0
volume       0.0
dtype: float64
```

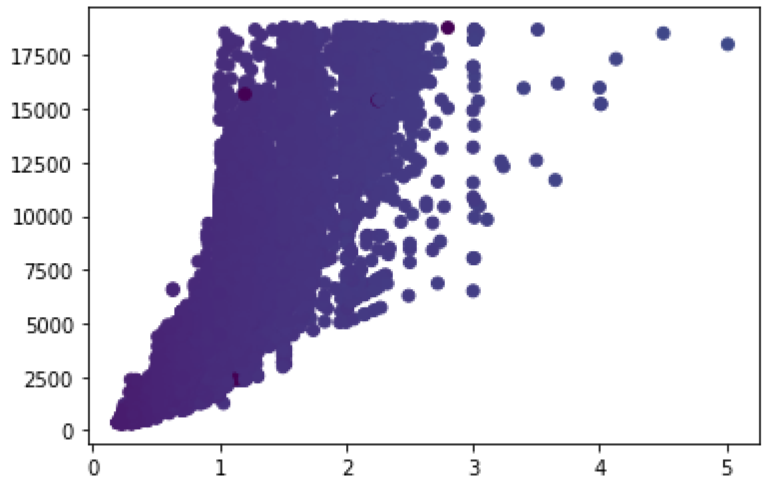
In [29]:
```
plt.scatter(data['carat'], data['price'], c=data['x'], cmap='viridis')
plt.show()
```



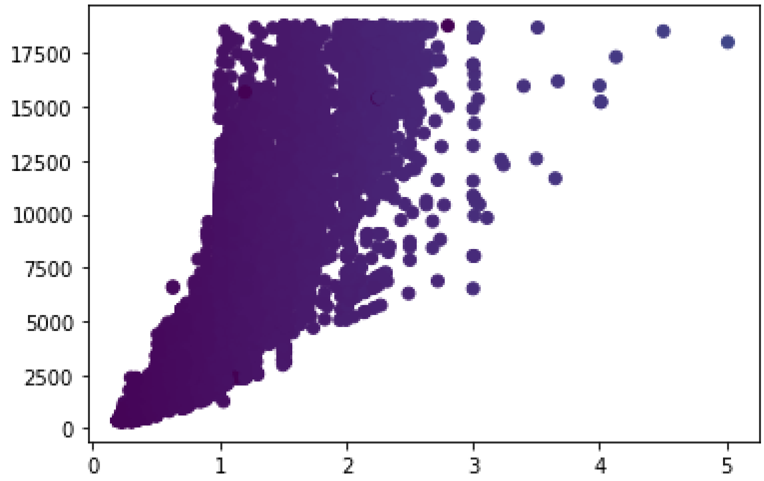
In [30]:
```
plt.scatter(data['carat'], data['price'], c=data['y'], cmap='viridis')
plt.show()
```



In [31]:
```
plt.scatter(data['carat'], data['price'], c=data['z'], cmap='viridis')
plt.show()
```



In [32]:
```
plt.scatter(data['carat'], data['price'], c=data['volume'], cmap='viridis')
plt.show()
```

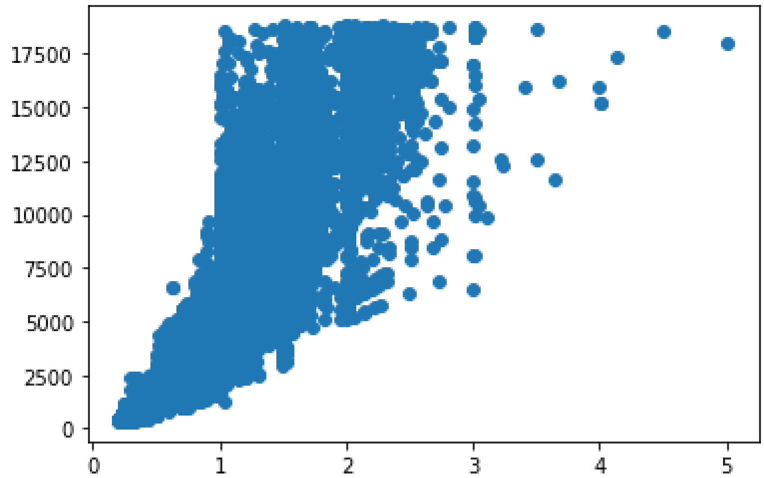


In [ ]:

In [33]:
```
from math import sqrt, log
x = data['carat']
linear = data['price'].map(lambda x: x)
square = data['price'].map(sqrt)
log = data['price'].map(log)
```

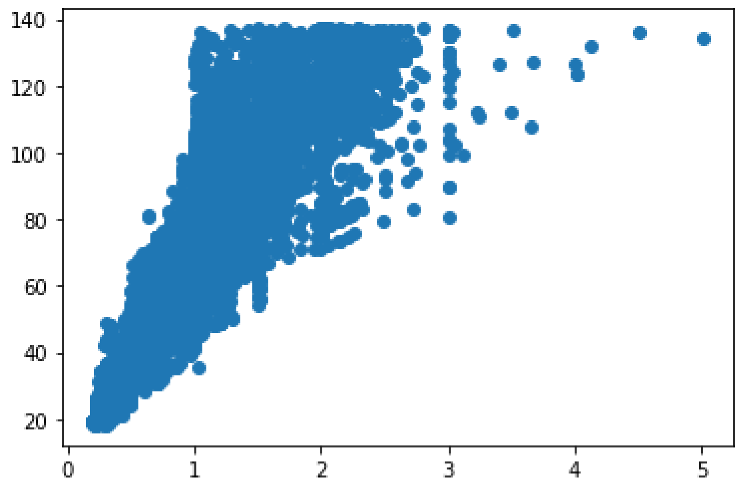
In [34]:
```
plt.scatter(x, linear)
```

Out[34]: <matplotlib.collections.PathCollection at 0x221c2507550>



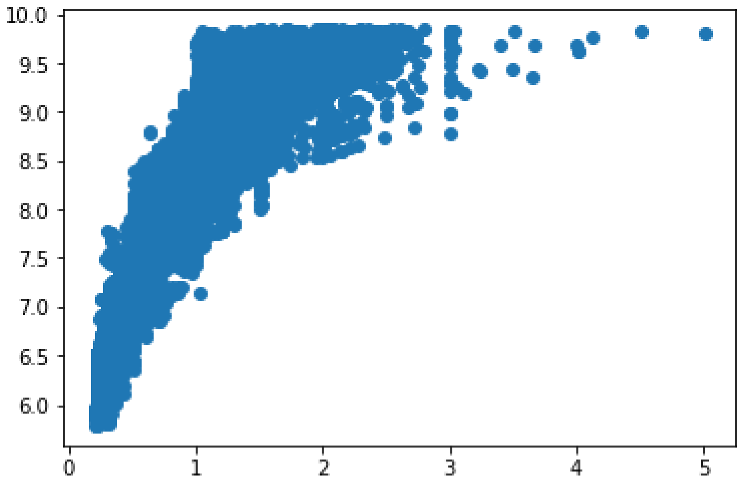
In [35]:
```
plt.scatter(x, square)
```

Out[35]: <matplotlib.collections.PathCollection at 0x221c251ee20>



In [36]:
```
plt.scatter(x, log)
```

Out[36]: <matplotlib.collections.PathCollection at 0x221c21fb4c0>



In [37]:
```
from scipy.stats import pearsonr
```

In [38]:
```
pearsonr(x, linear)
```

Out[38]: (0.9215913011934742, 0.0)

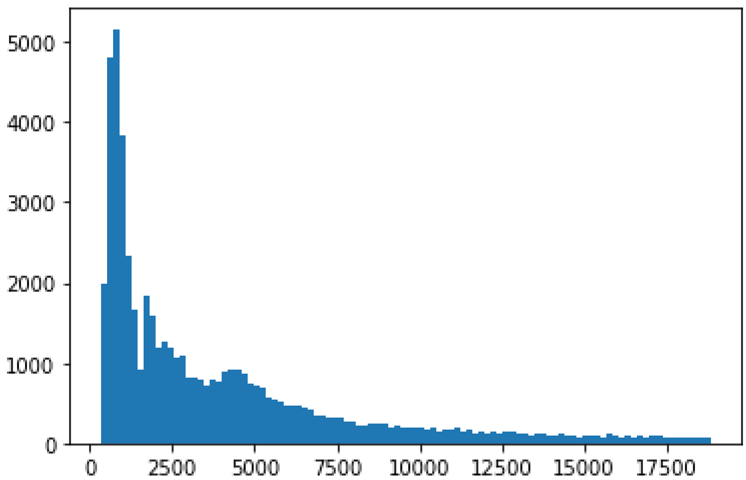
In [39]:
```
pearsonr(x, square)
```

Out[39]: (0.9465317436230122, 0.0)

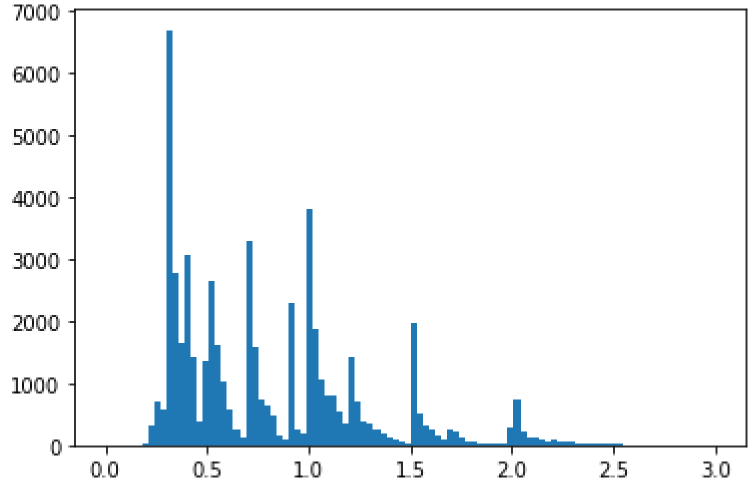
In [40]:
```
pearsonr(x, log)
```

Out[40]: (0.9202065980266558, 0.0)

In [41]:
```
plt.hist(data['price'], bins=100)
plt.show()
```

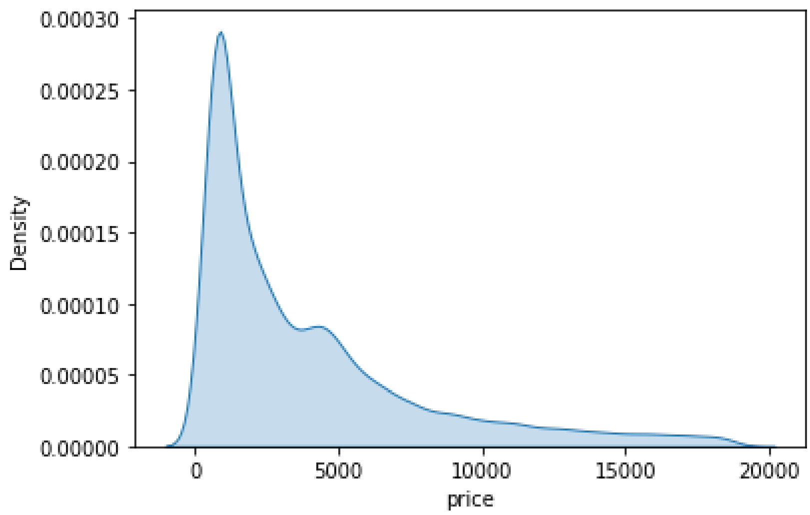


In [42]:
```
plt.hist(data['carat'], bins=100, range=(0,3))
plt.show()
```



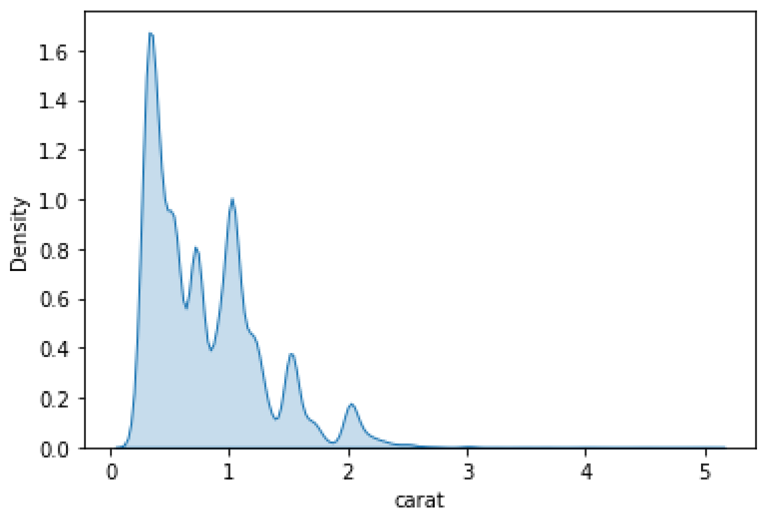
In [43]:
```
sns.kdeplot(data['price'], shade=True)
```

Out[43]: `<AxesSubplot:xlabel='price', ylabel='Density'>`

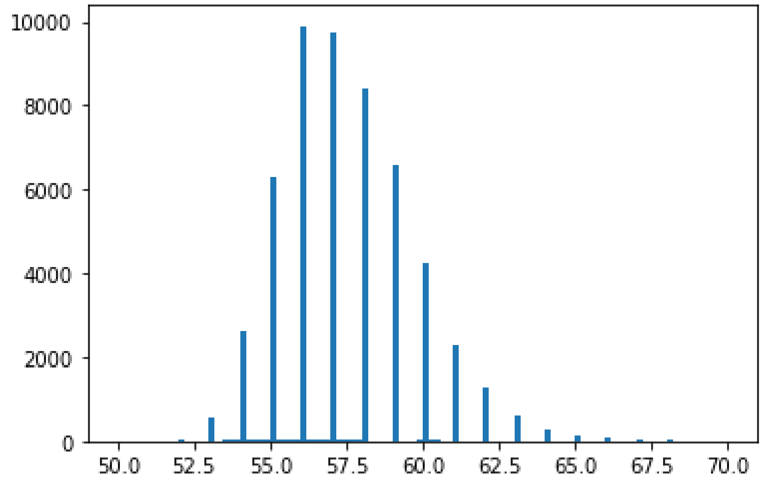


In [44]:
```
sns.kdeplot(data['carat'], shade=True)
```

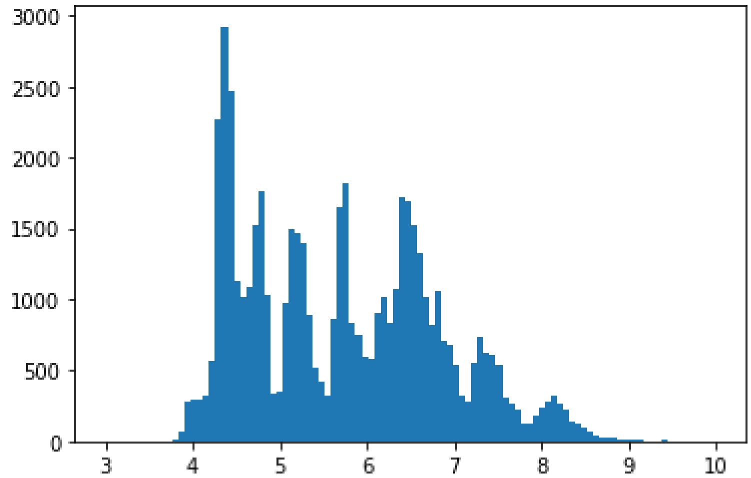
Out[44]: `<AxesSubplot:xlabel='carat', ylabel='Density'>`



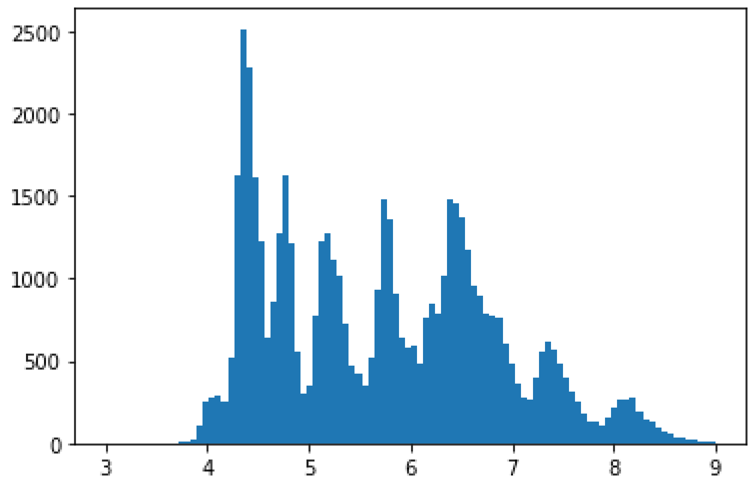
In [45]:
```
plt.hist(data['table'], bins=100, range=(50,70))
plt.show()
```



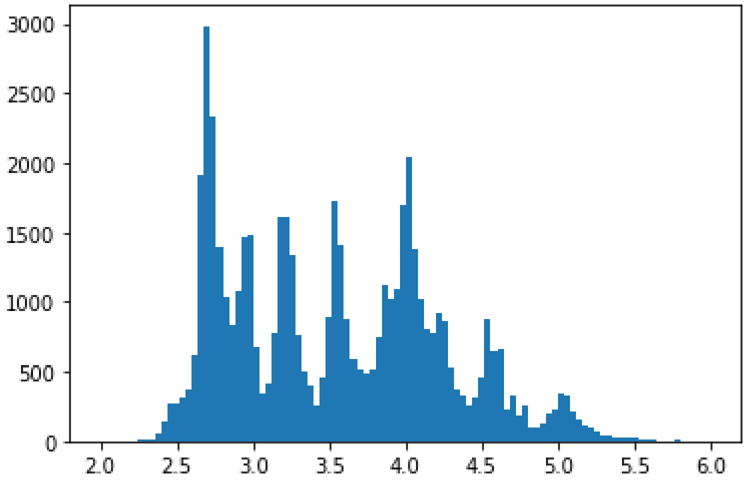
In [46]:
```
plt.hist(data['x'], bins=100, range=(3,10))
plt.show()
```



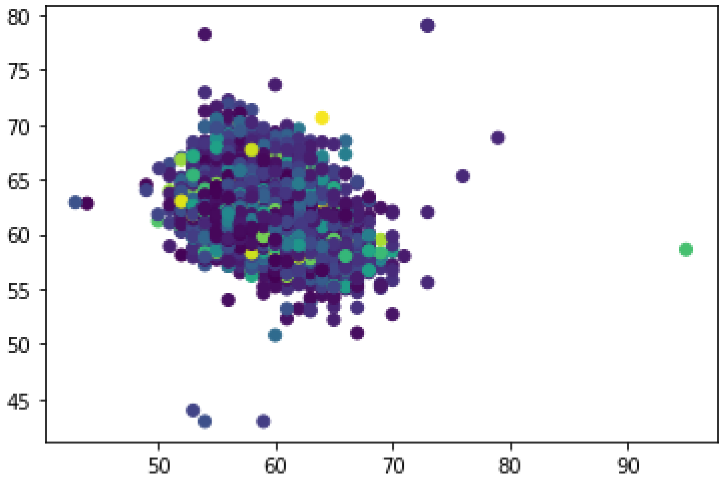
In [47]:
```
plt.hist(data['y'], bins=100, range=(3,9))
plt.show()
```



In [48]:
```
plt.hist(data['z'], bins=100, range=(2,6))
plt.show()
```

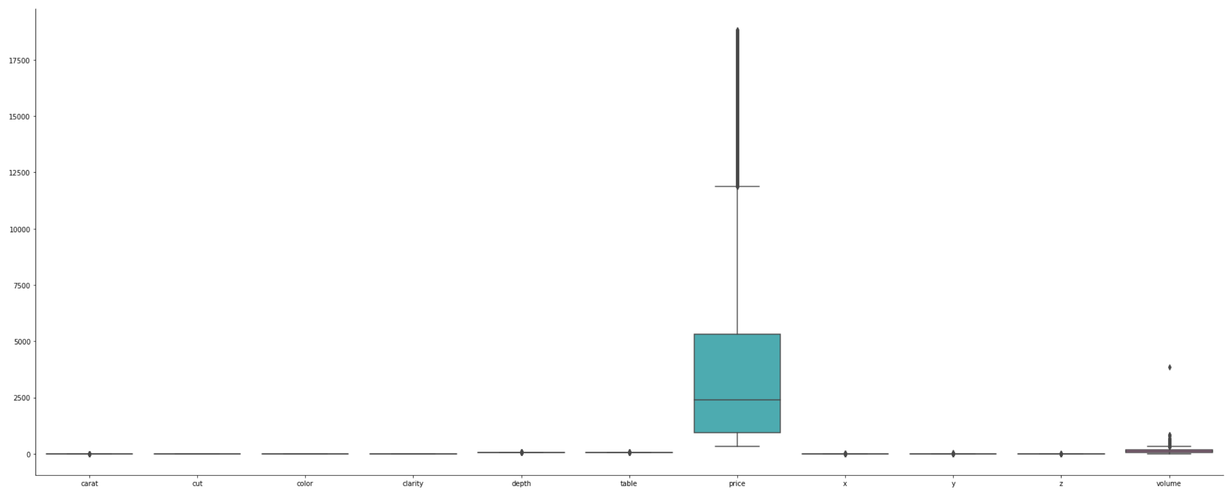


In [49]:
```
plt.scatter(data['table'], data['depth'], c=data['price'], cmap='viridis')
plt.show()
```



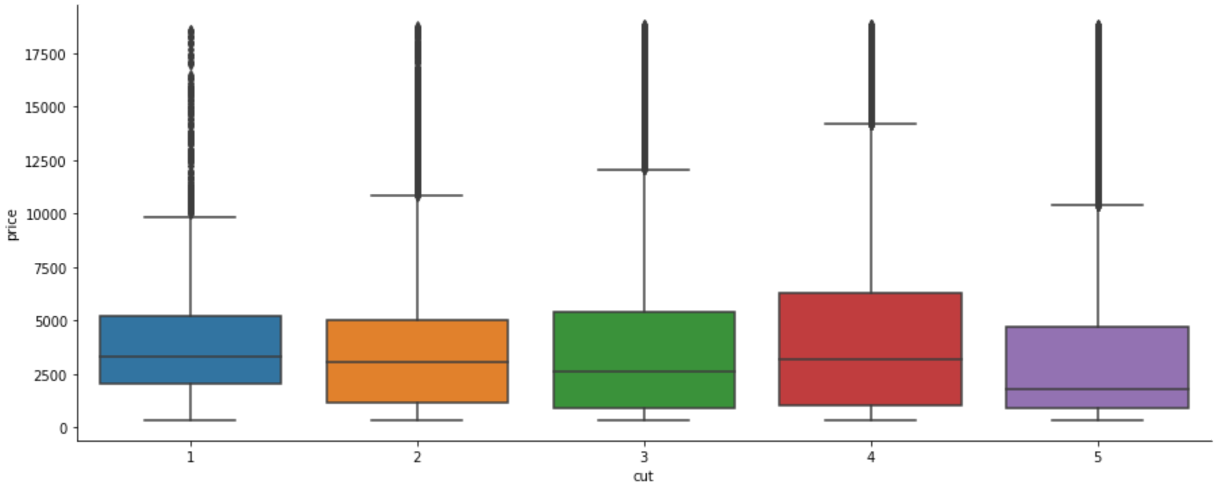
In [50]:
```
sns.catplot(data=data , kind='box' , height=10, aspect=2.5)
```

Out[50]: <seaborn.axisgrid.FacetGrid at 0x221c201a3a0>



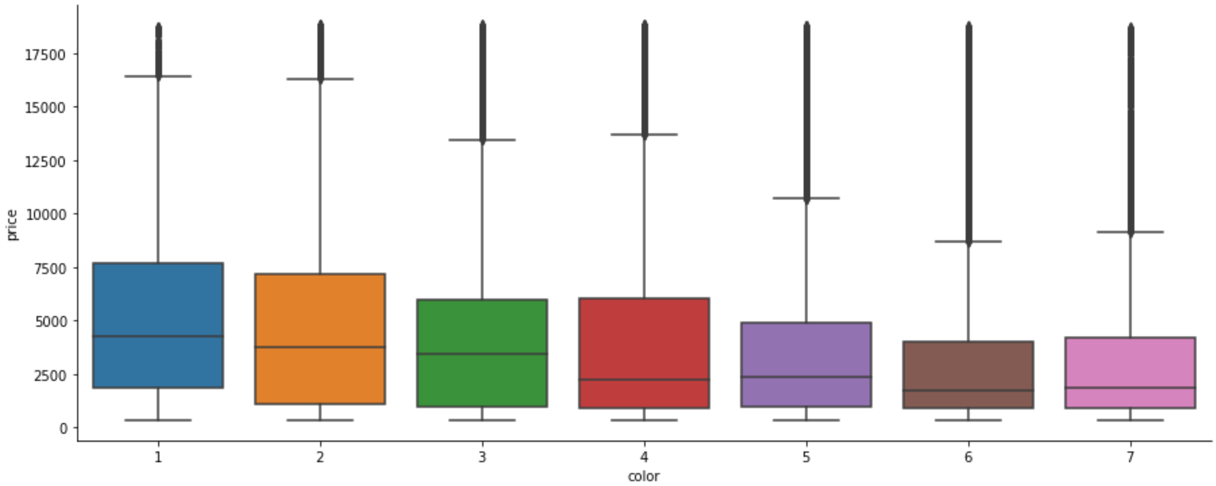
In [51]:
```
sns.catplot(x='cut', y='price', data=data, kind='box' ,aspect=2.5 )
```

Out[51]: <seaborn.axisgrid.FacetGrid at 0x221c68f96a0>



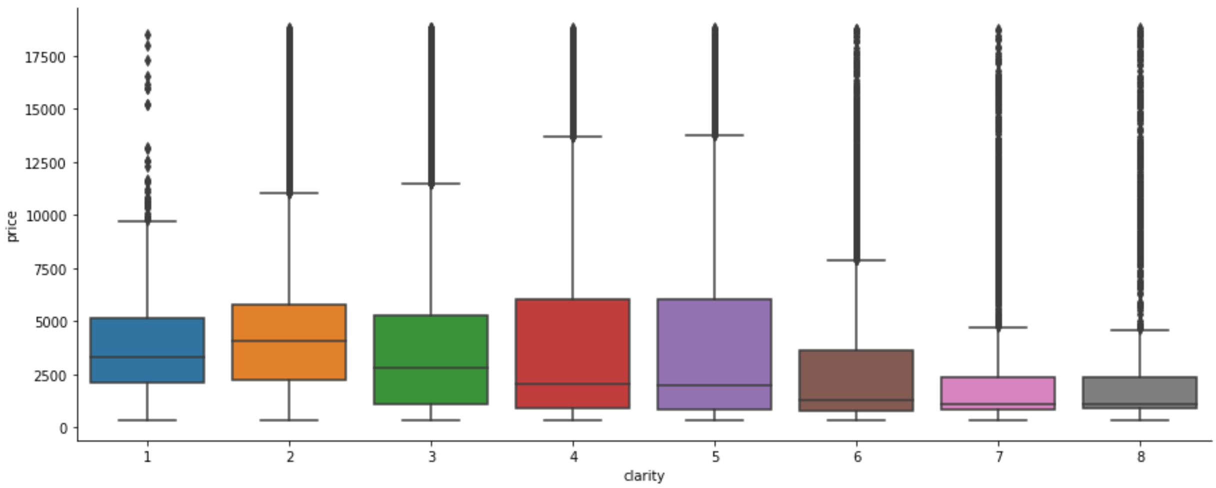
In [52]:
```
sns.catplot(x='color', y='price', data=data, kind='box' ,aspect=2.5 )
```

Out[52]: <seaborn.axisgrid.FacetGrid at 0x221c6a63550>



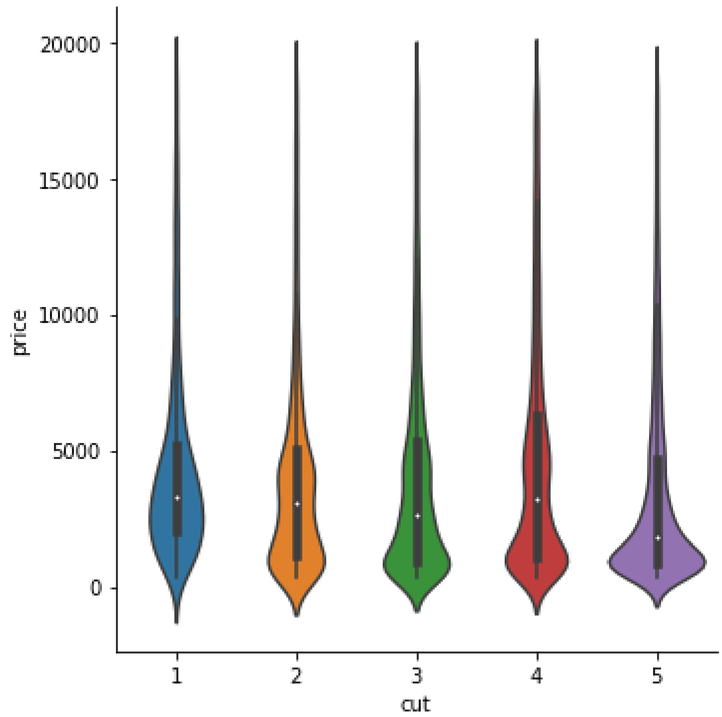
In [53]:
```
sns.catplot(x='clarity', y='price', data=data, kind='box' ,aspect=2.5 )
```

Out[53]: <seaborn.axisgrid.FacetGrid at 0x221ca27aa30>



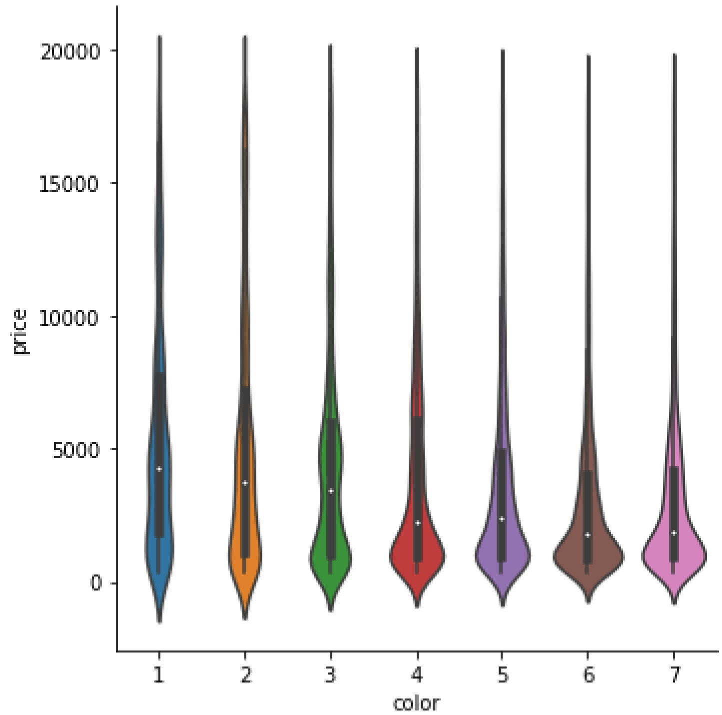
In [100…
```
sns.catplot(x='cut', y='price' , data=data , kind='violin', aspect=1)
```

Out[100… <seaborn.axisgrid.FacetGrid at 0x221de1daa30>



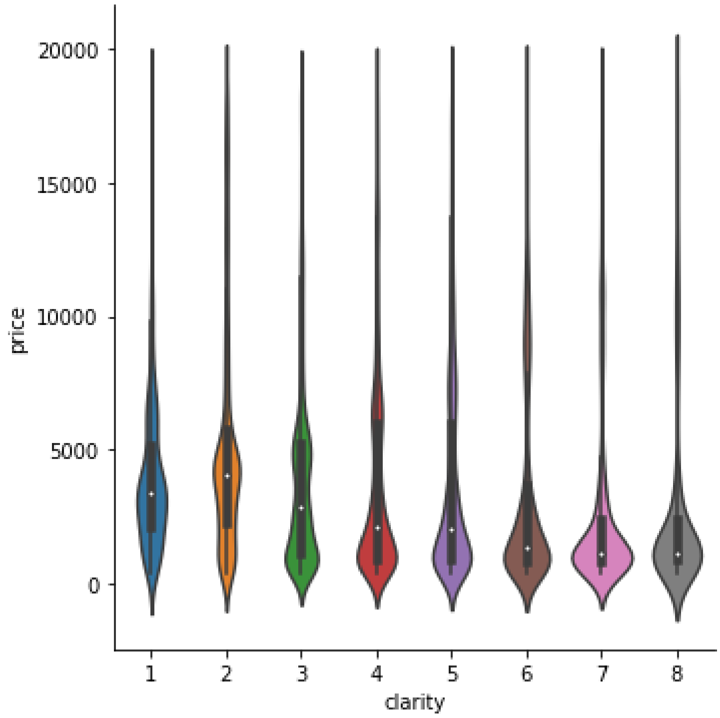
In [101…
```
sns.catplot(x='color', y='price' , data=data , kind='violin', aspect=1)
```

Out[101… <seaborn.axisgrid.FacetGrid at 0x221ce49b730>



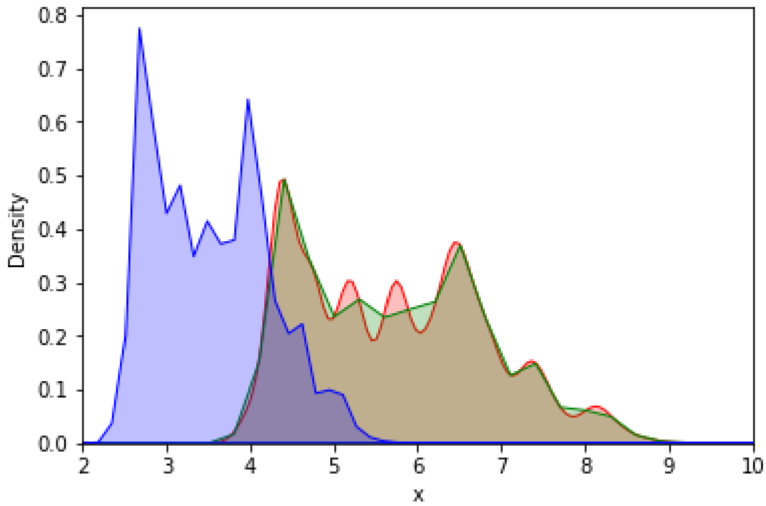
In [102…
```
sns.catplot(x='clarity', y='price' , data=data , kind='violin', aspect=1)
```

Out[102… <seaborn.axisgrid.FacetGrid at 0x221dd444850>



In [57]:
```
sns.kdeplot(data['x'] ,shade=True , color='r' )
sns.kdeplot(data['y'] , shade=True , color='g' )
sns.kdeplot(data['z'] , shade= True , color='b')
plt.xlim(2,10)
```

Out[57]: (2.0, 10.0)

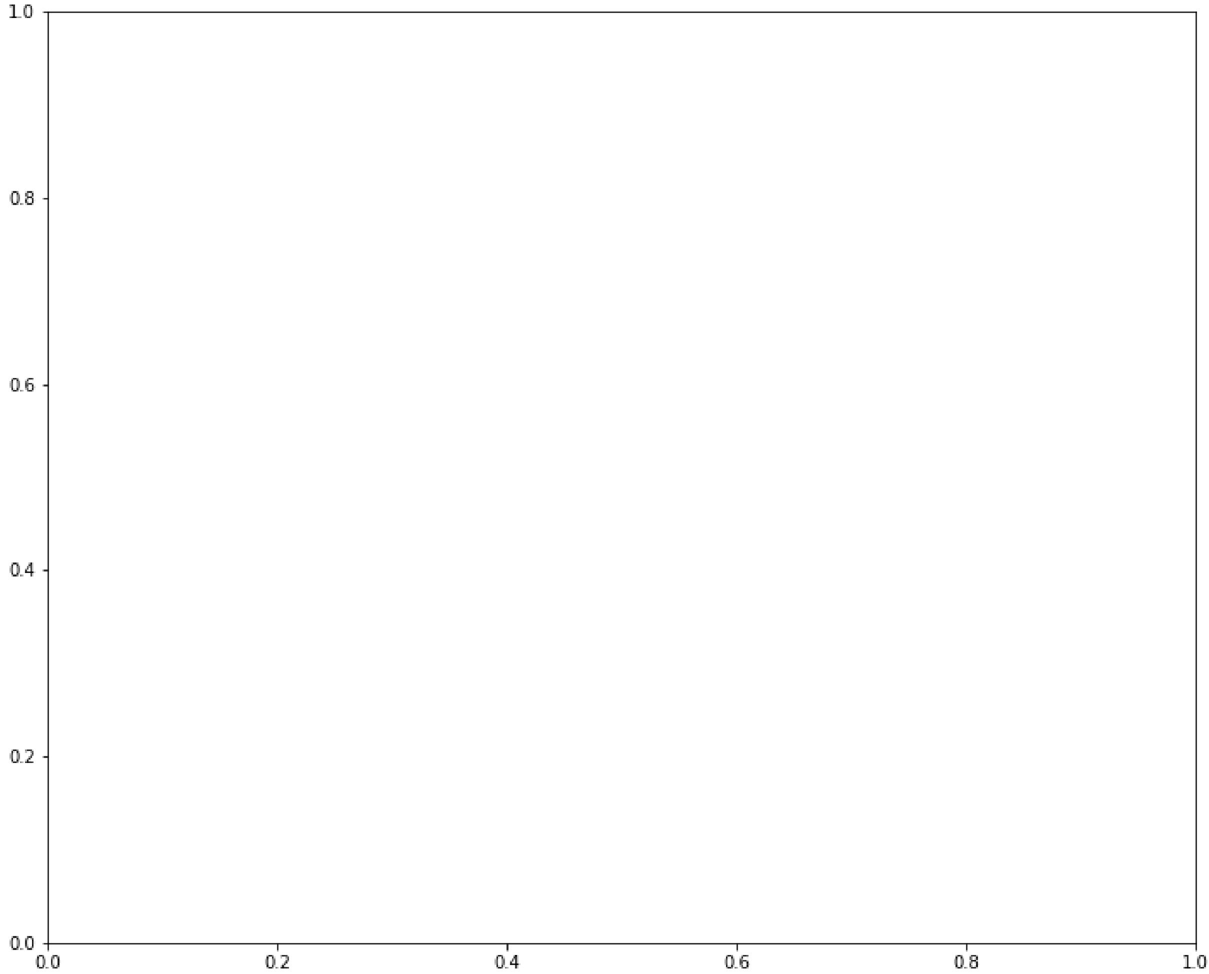


In [97]:
```
plt.scatter(data['table'], data['price'], c=clarity, cmap='viridis')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Roaming\Python\Python38\site-packages\matplotlib\axes\_axes.py in _parse_scatter_color_args(c, edgecolors, kwargs, xsize, get_next_color_func)
   4290             try:  # Is 'c' acceptable as PathCollection facecolors?
-> 4291                 colors = mcolors.to_rgba_array(c)
   4292             except (TypeError, ValueError) as err:

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in to_rgba_array(c, alpha)
    340     else:
--> 341         return np.array([to_rgba(cc, alpha) for cc in c])
    342
```

```
~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in <listcomp>(.
0)
    340     else:
--> 341         return np.array([to_rgba(cc, alpha) for cc in c])
    342 

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in to_rgba(c, a
lpha)
    188     if rgba is None:  # Suppress exception chaining of cache lookup failure.
--> 189         rgba = _to_rgba_no_colorcycle(c, alpha)
    190         try:

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in _to_rgba_no_
colorcycle(c, alpha)
    262     if not np.iterable(c):
--> 263         raise ValueError(f"Invalid RGBA argument: {orig_c!r}")
    264     if len(c) not in [3, 4]:

ValueError: Invalid RGBA argument: 6.0

The above exception was the direct cause of the following exception:

ValueError                                Traceback (most recent call last)
<ipython-input-97-c89b70bf7ae4> in <module>
----> 1 plt.scatter(data['table'], data['price'], c=clarity, cmap='viridis')
      2 plt.show()

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\pyplot.py in scatter(x,
 y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts, edgecolors, plot
nonfinite, data, **kwargs)
   2888         verts=cbook.deprecation._deprecated_parameter,
   2889         edgecolors=None, *, plotnonfinite=False, data=None, **kwargs):
-> 2890     __ret = gca().scatter(
   2891         x, y, s=s, c=c, marker=marker, cmap=cmap, norm=norm,
   2892         vmin=vmin, vmax=vmax, alpha=alpha, linewidths=linewidths,

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\__init__.py in inner(ax,
 data, *args, **kwargs)
   1436     def inner(ax, *args, data=None, **kwargs):
   1437         if data is None:
-> 1438             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1439 
   1440         bound = new_sig.bind(ax, *args, **kwargs)

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\cbook\deprecation.py in w
rapper(*inner_args, **inner_kwargs)
    409                  else deprecation_addendum,
    410             **kwargs)
--> 411         return func(*inner_args, **inner_kwargs)
    412 
    413     return wrapper

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\axes\_axes.py in scatter
(self, x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts, edgecol
ors, plotnonfinite, **kwargs)
   4449 
   4450         c, colors, edgecolors = \
-> 4451             self._parse_scatter_color_args(
   4452                 c, edgecolors, kwargs, x.size,
   4453                 get_next_color_func=self._get_patches_for_fill.get_next_colo
r)

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\axes\_axes.py in _parse_s
catter_color_args(c, edgecolors, kwargs, xsize, get_next_color_func)
   4295             else:
   4296                 if not valid_shape:
-> 4297                     raise invalid_shape_exception(c.size, xsize) from er
r
   4298                 # Both the mapping *and* the RGBA conversion failed: pre
```

```
tty
   4299                     # severe failure => one may appreciate a verbose feedbac
k.

ValueError: 'c' argument has 53940 elements, which is inconsistent with 'x' and 'y'
 with size 53917.
```

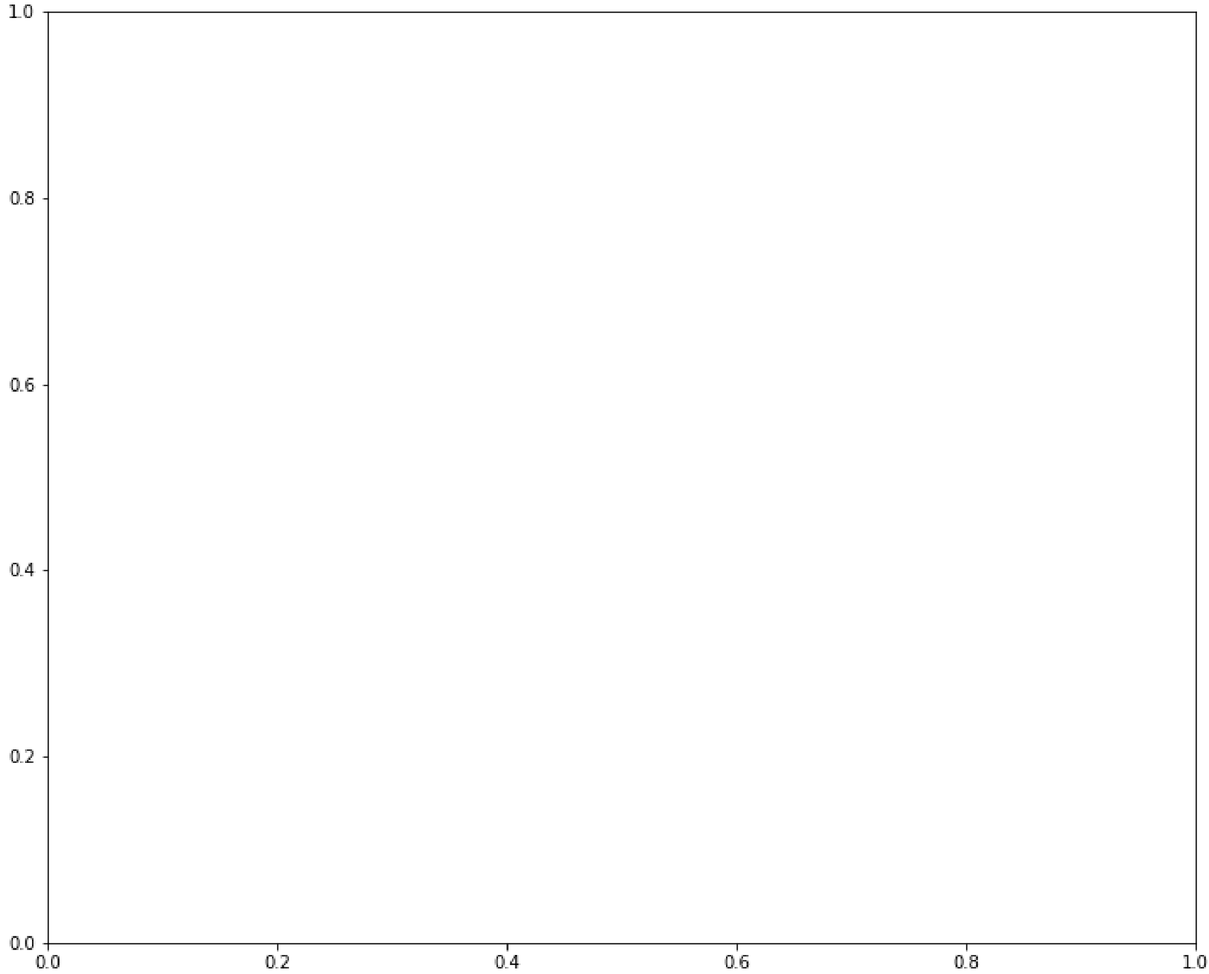


In [98]:

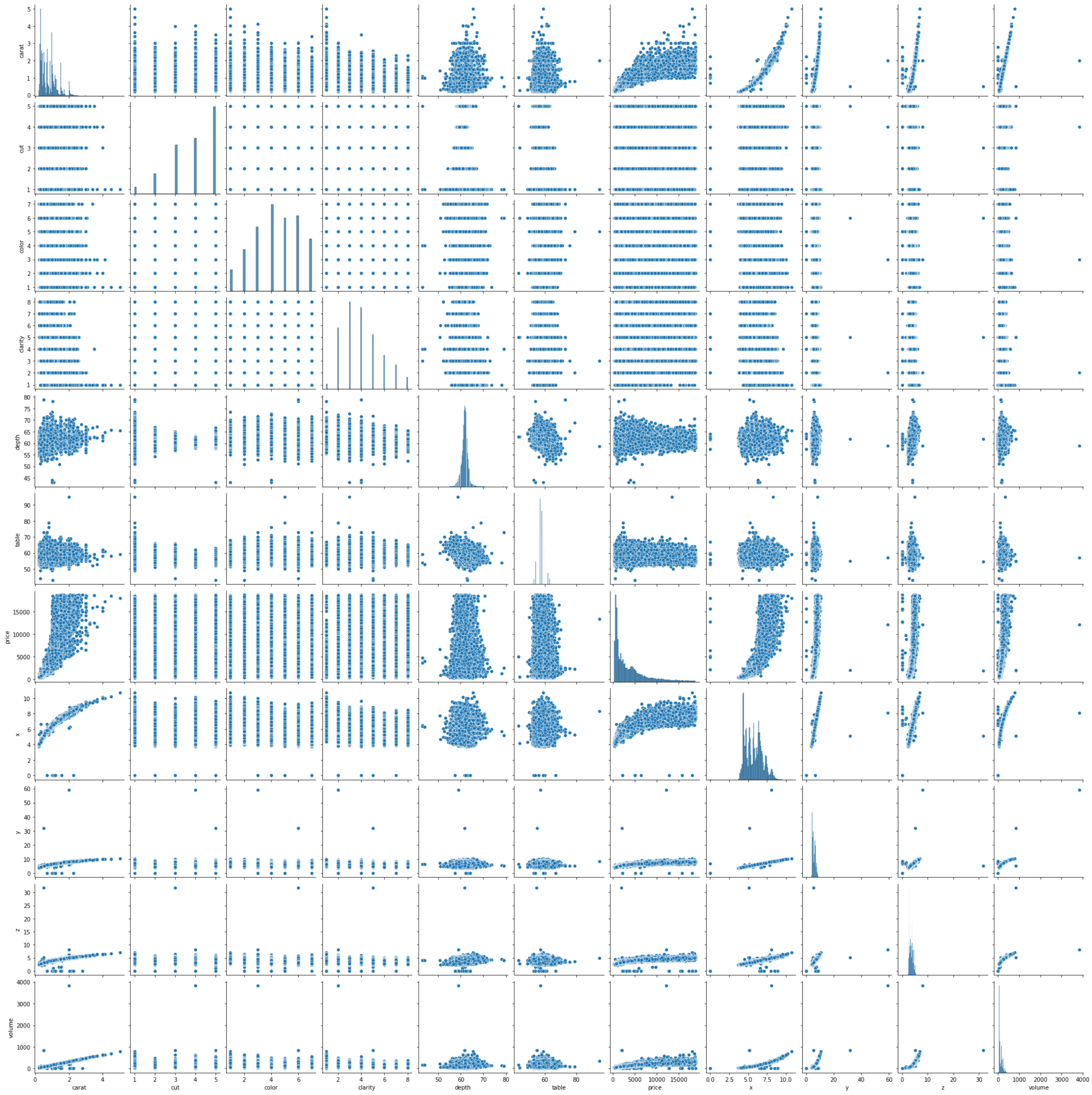
```
plt.scatter(data['depth'], data['price'], c=clarity, cmap='viridis')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Roaming\Python\Python38\site-packages\matplotlib\axes\_axes.py in _parse_s
catter_color_args(c, edgecolors, kwargs, xsize, get_next_color_func)
   4290             try:  # Is 'c' acceptable as PathCollection facecolors?
-> 4291                 colors = mcolors.to_rgba_array(c)
   4292             except (TypeError, ValueError) as err:

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in to_rgba_arra
y(c, alpha)
    340     else:
--> 341         return np.array([to_rgba(cc, alpha) for cc in c])
    342

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in <listcomp>(.
0)
    340     else:
--> 341         return np.array([to_rgba(cc, alpha) for cc in c])
    342

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in to_rgba(c, a
lpha)
    188     if rgba is None:  # Suppress exception chaining of cache lookup failure.
--> 189         rgba = _to_rgba_no_colorcycle(c, alpha)
    190         try:

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\colors.py in _to_rgba_no_
```

```
colorcycle(c, alpha)
    262     if not np.iterable(c):
--> 263         raise ValueError(f"Invalid RGBA argument: {orig_c!r}")
    264     if len(c) not in [3, 4]:

ValueError: Invalid RGBA argument: 6.0

The above exception was the direct cause of the following exception:

ValueError                                Traceback (most recent call last)
<ipython-input-98-cb842045cb00> in <module>
----> 1 plt.scatter(data['depth'], data['price'], c=clarity, cmap='viridis')

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\pyplot.py in scatter(x,
 y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts, edgecolors, plot
nonfinite, data, **kwargs)
   2888         verts=cbook.deprecation._deprecated_parameter,
   2889         edgecolors=None, *, plotnonfinite=False, data=None, **kwargs):
-> 2890     __ret = gca().scatter(
   2891         x, y, s=s, c=c, marker=marker, cmap=cmap, norm=norm,
   2892         vmin=vmin, vmax=vmax, alpha=alpha, linewidths=linewidths,

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\__init__.py in inner(ax,
 data, *args, **kwargs)
   1436     def inner(ax, *args, data=None, **kwargs):
   1437         if data is None:
-> 1438             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1439 
   1440         bound = new_sig.bind(ax, *args, **kwargs)

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\cbook\deprecation.py in w
rapper(*inner_args, **inner_kwargs)
    409                  else deprecation_addendum,
    410             **kwargs)
--> 411         return func(*inner_args, **inner_kwargs)
    412 
    413     return wrapper

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\axes\_axes.py in scatter
(self, x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts, edgecol
ors, plotnonfinite, **kwargs)
   4449 
   4450         c, colors, edgecolors = \
-> 4451             self._parse_scatter_color_args(
   4452                 c, edgecolors, kwargs, x.size,
   4453                 get_next_color_func=self._get_patches_for_fill.get_next_colo
r)

~\AppData\Roaming\Python\Python38\site-packages\matplotlib\axes\_axes.py in _parse_s
catter_color_args(c, edgecolors, kwargs, xsize, get_next_color_func)
   4295                 else:
   4296                     if not valid_shape:
-> 4297                         raise invalid_shape_exception(c.size, xsize) from er
r
   4298                     # Both the mapping *and* the RGBA conversion failed: pre
tty
   4299                     # severe failure => one may appreciate a verbose feedbac
k.

ValueError: 'c' argument has 53940 elements, which is inconsistent with 'x' and 'y'
 with size 53917.
```



In [60]:
```
pairplot = sns.pairplot(data)
```



In [61]:
```
# Podemos ver que há 2 pontos outliers. Por inspeção, são os pontos com y > 20. Vamo
```

In [62]:
```
data.describe()
```

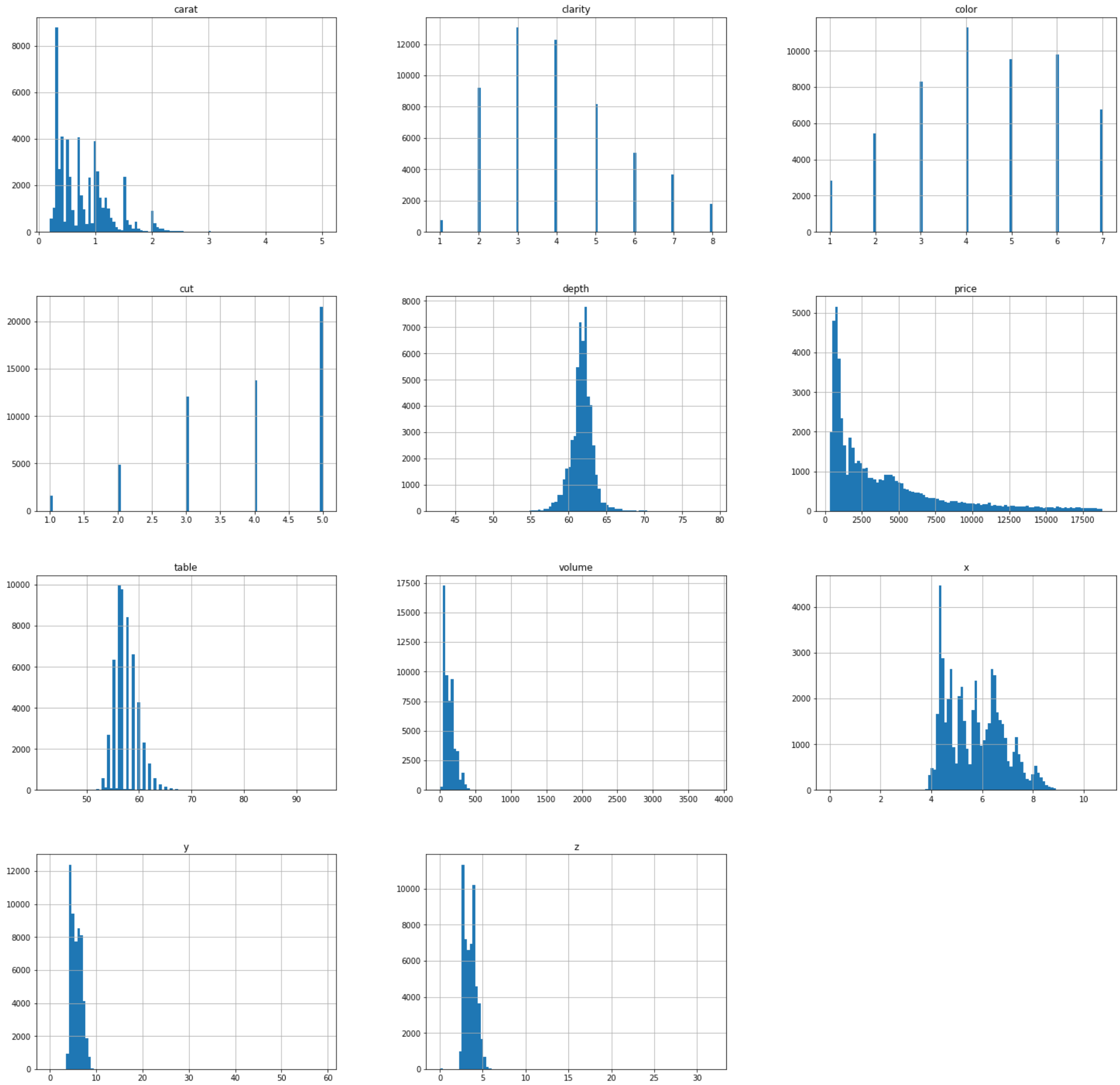
Out[62]:

	carat	cut	color	clarity	depth	table	
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.0(
mean	0.797940	3.904097	4.405803	4.051020	61.749405	57.457184	3932.7!
std	0.474011	1.116600	1.701105	1.647136	1.432621	2.234491	3989.4:
min	0.200000	1.000000	1.000000	1.000000	43.000000	43.000000	326.0(
25%	0.400000	3.000000	3.000000	3.000000	61.000000	56.000000	950.0(
50%	0.700000	4.000000	4.000000	4.000000	61.800000	57.000000	2401.0(
75%	1.040000	5.000000	6.000000	5.000000	62.500000	59.000000	5324.2!
max	5.010000	5.000000	7.000000	8.000000	79.000000	95.000000	18823.0(

In [63]:
```
hist = data.hist(figsize = (25,25), bins=100)
```

```
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:30
```

```
7: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:30
7: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:31
3: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:31
3: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
```



In [64]:

```
## Visualizando entradas com dimensão 'zero'

data.loc[(data['x']==0) | (data['y']==0) | (data['z']==0)]
```

Out[64]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
666	1.00	3	3	4	63.3	53.0	5139	0.00	0.00	0.0	0.0
3832	1.01	4	3	1	58.1	59.0	3167	6.66	6.60	0.0	0.0

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
5506	1.15	5	4	4	59.2	56.0	5564	6.88	6.83	0.0	0.0
7146	2.20	4	3	3	61.2	59.0	17265	8.42	8.37	0.0	0.0
10783	0.71	2	5	2	64.1	60.0	2130	0.00	0.00	0.0	0.0
13551	1.07	5	5	2	61.6	56.0	4954	0.00	6.62	0.0	0.0
14867	2.80	2	4	2	63.8	58.0	18788	8.90	8.85	0.0	0.0
20608	1.00	4	4	2	59.1	59.0	3142	6.55	6.48	0.0	0.0
23201	2.18	4	3	2	59.4	61.0	12631	8.49	8.45	0.0	0.0
31083	2.02	4	3	4	62.7	53.0	18207	8.02	7.95	0.0	0.0
31912	1.01	4	5	2	59.2	58.0	3837	6.50	6.47	0.0	0.0
33198	1.14	1	4	5	57.5	67.0	6381	0.00	0.00	0.0	0.0
34619	1.10	4	4	2	63.0	59.0	3696	6.50	6.47	0.0	0.0
37525	1.56	5	4	4	62.2	54.0	12800	0.00	0.00	0.0	0.0
39382	1.12	4	4	1	60.4	59.0	2383	6.71	6.67	0.0	0.0
42823	1.50	2	4	1	64.0	61.0	4731	7.15	7.04	0.0	0.0
45807	0.71	2	5	2	64.1	60.0	2130	0.00	0.00	0.0	0.0
47582	2.25	4	3	2	62.8	59.0	18034	0.00	0.00	0.0	0.0
51878	2.25	4	2	3	61.3	58.0	15397	8.52	8.42	0.0	0.0
52683	1.20	4	7	7	62.1	59.0	15686	0.00	0.00	0.0	0.0

In [65]:
```
# Quantos têm dimensão zero?
len(data.loc[(data['volume']==0)])
```

Out[65]: 20

In [66]:
```
data = data[(data[['x','y','z']] != 0).all(axis=1)]
```

In [67]:
```
# Confirmando

data.loc[(data['x']==0) | (data['y']==0) | (data['z']==0)]
```

Out[67]:

carat	cut	color	clarity	depth	table	price	x	y	z	volume

In [68]:
```
data.describe()
```

Out[68]:

	carat	cut	color	clarity	depth	table	
count	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.0(
mean	0.797698	3.904228	4.405972	4.051502	61.749514	57.456834	3930.9!
std	0.473795	1.116579	1.701272	1.647005	1.432331	2.234064	3987.2!
min	0.200000	1.000000	1.000000	1.000000	43.000000	43.000000	326.0(
25%	0.400000	3.000000	3.000000	3.000000	61.000000	56.000000	949.0(
50%	0.700000	4.000000	4.000000	4.000000	61.800000	57.000000	2401.0(

	carat	cut	color	clarity	depth	table	
75%	1.040000	5.000000	6.000000	5.000000	62.500000	59.000000	5323.2!
max	5.010000	5.000000	7.000000	8.000000	79.000000	95.000000	18823.0(

In [69]:
```
# Visualizando outliers y>20 e z>20
```

In [70]:
```
data.loc[(data['y']>20)]
```

Out[70]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
32797	2.00	4	3	2	58.9	57.0	12210	8.09	58.9	8.06	3840.59806
46063	0.51	5	6	5	61.8	55.0	2075	5.15	31.8	5.12	838.50240

In [71]:
```
data = data[(data[['y']] < 20).all(axis=1)]
```

In [72]:
```
data.loc[(data['y']>20)]
```

Out[72]:

carat	cut	color	clarity	depth	table	price	x	y	z	volume

In [73]:
```
data.loc[(data['z']>20)]
```

Out[73]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
19269	0.51	3	6	5	61.8	54.7	1970	5.12	5.15	31.8	838.5024

In [74]:
```
data = data[(data[['z']] < 20).all(axis=1)]
```

In [75]:
```
data.loc[(data['z']>20)]
```

Out[75]:

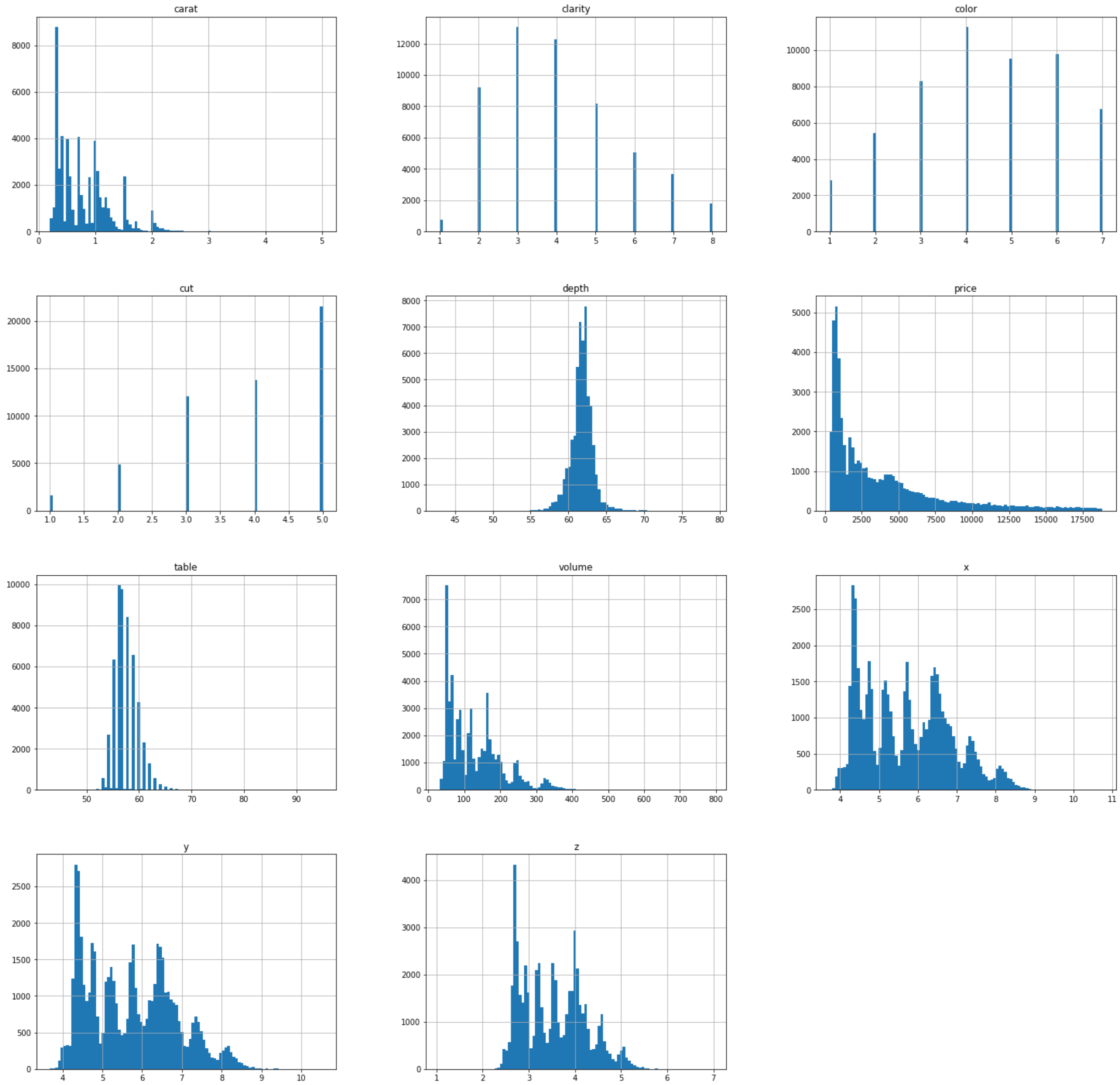
carat	cut	color	clarity	depth	table	price	x	y	z	volume

In [76]:
```
data.describe()
```

Out[76]:

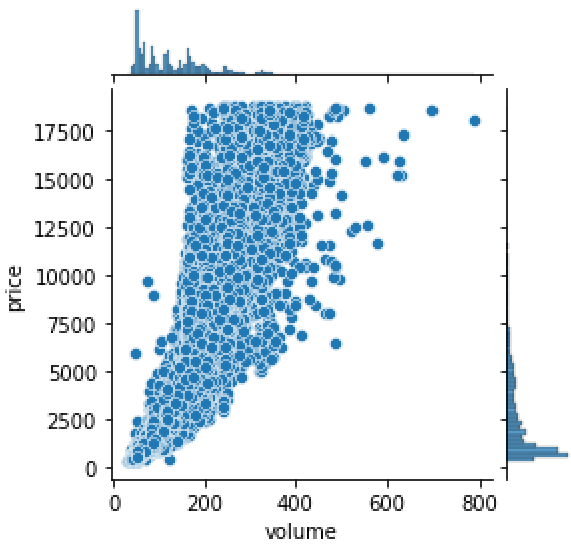
	carat	cut	color	clarity	depth	table	
count	53917.000000	53917.000000	53917.000000	53917.000000	53917.000000	53917.000000	53917.0(
mean	0.797687	3.904223	4.405939	4.051505	61.749565	57.456939	3930.9'
std	0.473777	1.116593	1.701281	1.647017	1.432318	2.234069	3987.2'
min	0.200000	1.000000	1.000000	1.000000	43.000000	43.000000	326.0(
25%	0.400000	3.000000	3.000000	3.000000	61.000000	56.000000	949.0(
50%	0.700000	4.000000	4.000000	4.000000	61.800000	57.000000	2401.0(
75%	1.040000	5.000000	6.000000	5.000000	62.500000	59.000000	5323.0(
max	5.010000	5.000000	7.000000	8.000000	79.000000	95.000000	18823.0(

In [77]:
```
hist = data.hist(figsize = (25,25), bins=100)
```

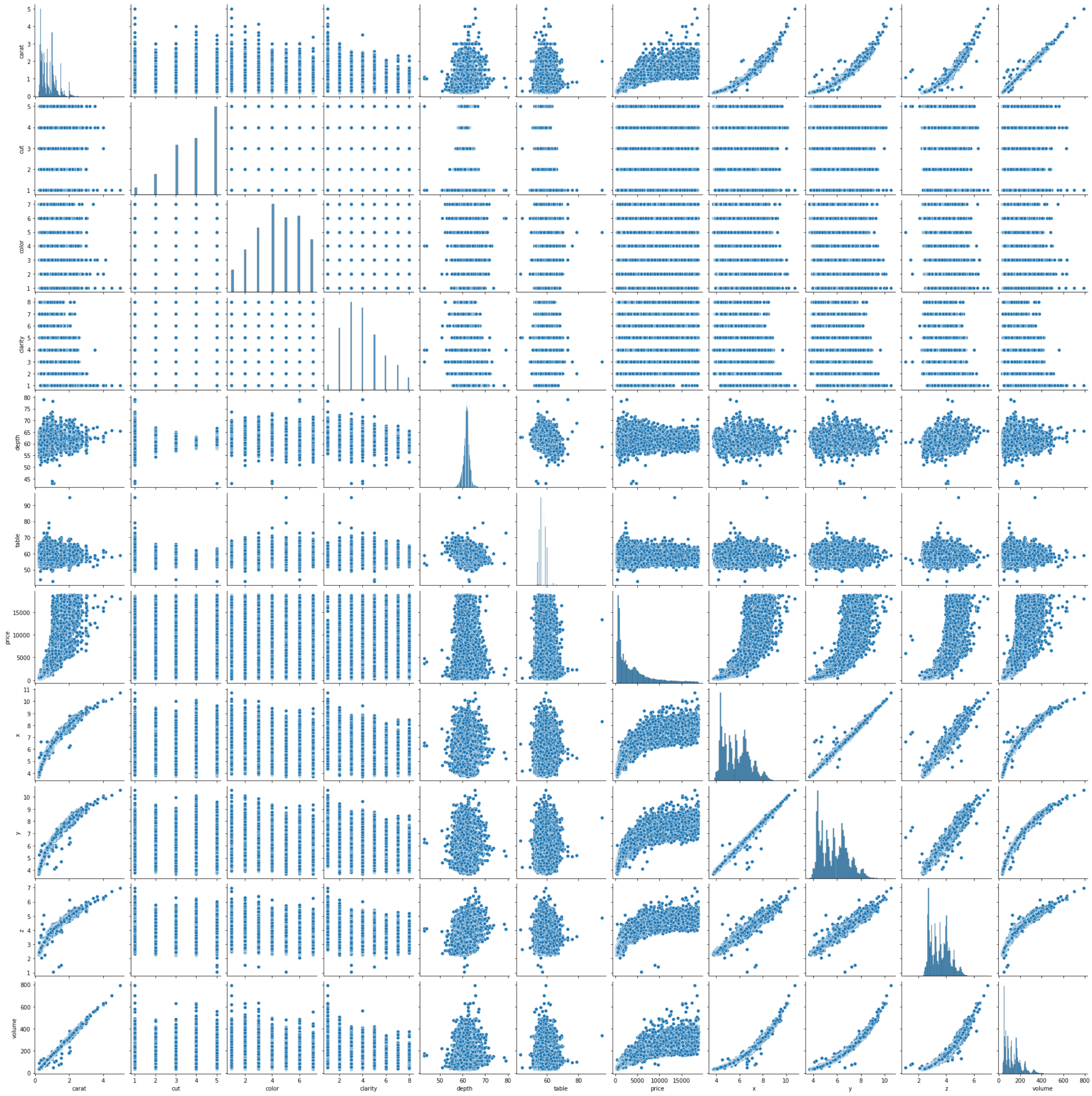


In [99]:
```
sns.jointplot(x='volume', y='price' , data=data, height=4)
```

Out[99]: <seaborn.axisgrid.JointGrid at 0x221dcff0d30>



In [79]:
```
pairplot = sns.pairplot(data)
```



In [80]:
```
from matplotlib import rcParams
rcParams['figure.figsize'] = 12,10

heatmap = sns.heatmap(data.corr(), linewidth=0, annot=True, center=0.5, cmap='Blues'
```

In [81]:
```
dataClean = data.drop(['x', 'y', 'z'], axis=1)
dataClean
```

Out[81]:

	carat	cut	color	clarity	depth	table	price	volume
0	0.31	4	3	6	62.6	60.0	802	50.154390
1	0.58	3	7	4	61.0	58.0	1750	95.756766
2	1.22	5	5	2	62.5	56.0	6518	199.189095
3	0.30	2	3	3	63.4	58.0	421	47.839072
4	0.30	4	7	2	62.3	58.0	447	49.667904
...	...	...	...	...	...	...	...	...
53935	0.71	4	6	3	59.1	60.0	2441	118.611568
53936	1.51	5	3	2	61.9	56.0	8951	244.259264
53937	0.33	4	4	6	59.5	60.0	752	55.528200
53938	0.32	3	6	3	63.3	56.0	720	51.917250
53939	0.52	4	6	4	61.6	59.0	1694	84.076008

53917 rows × 8 columns

In [82]:
```
data = data.sample(frac=1).reset_index(drop=True)
data
```

Out[82]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
0	0.91	3	6	2	62.9	56.0	3763	6.14	6.17	3.87	146.610306
1	0.32	5	5	4	62.4	55.0	645	4.41	4.43	2.76	53.920188
2	1.00	4	4	3	60.1	61.0	3634	6.44	6.40	3.86	159.093760
3	1.50	5	2	1	61.3	57.0	5695	7.39	7.36	4.52	245.844608
4	0.78	5	2	4	60.9	57.0	2422	5.97	5.92	3.62	127.939488
...	...	...	...	...	...	...	...	...	...	...	...
53912	1.21	4	7	2	62.5	57.0	6505	6.79	6.71	4.22	192.266998
53913	0.81	5	2	4	61.4	55.0	3355	6.03	6.06	3.71	135.570078
53914	1.31	5	3	2	62.3	56.0	6225	7.00	6.97	4.35	212.236500
53915	0.31	5	3	8	62.3	55.0	789	4.32	4.35	2.70	50.738400
53916	0.31	2	7	3	63.8	57.0	533	4.29	4.33	2.75	51.083175

53917 rows × 11 columns

In [83]:
```
#compare = pd.DataFrame({'Algorithms' : models , 'R2-Scores' : R2_Scores})
#compare.sort_values(by='R2-Scores' ,ascending=False)
```

In [84]:
```
#sns.barplot(x='R2-Scores' , y='Algorithms' , data=compare)
```

In [85]:
```
#sns.factorplot(x='Algorithms', y='R2-Scores' , data=compare, size=6 , aspect=4)
```

In [86]:
```
#from sklearn.preprocessing import StandardScaler
#dataScaled =
```

In [87]:
```
data
```

Out[87]:

	carat	cut	color	clarity	depth	table	price	x	y	z	volume
0	0.91	3	6	2	62.9	56.0	3763	6.14	6.17	3.87	146.610306
1	0.32	5	5	4	62.4	55.0	645	4.41	4.43	2.76	53.920188
2	1.00	4	4	3	60.1	61.0	3634	6.44	6.40	3.86	159.093760
3	1.50	5	2	1	61.3	57.0	5695	7.39	7.36	4.52	245.844608
4	0.78	5	2	4	60.9	57.0	2422	5.97	5.92	3.62	127.939488
...	...	...	...	...	...	...	...	...	...	...	...
53912	1.21	4	7	2	62.5	57.0	6505	6.79	6.71	4.22	192.266998
53913	0.81	5	2	4	61.4	55.0	3355	6.03	6.06	3.71	135.570078
53914	1.31	5	3	2	62.3	56.0	6225	7.00	6.97	4.35	212.236500
53915	0.31	5	3	8	62.3	55.0	789	4.32	4.35	2.70	50.738400
53916	0.31	2	7	3	63.8	57.0	533	4.29	4.33	2.75	51.083175

53917 rows × 11 columns

In [88]:
```
data.mean()
```

Out[88]:

```
carat          0.797687
cut            3.904223
color          4.405939
clarity        4.051505
depth         61.749565
table         57.456939
price       3930.910474
x              5.731605
y              5.733428
z              3.539409
volume       129.802460
dtype: float64
```

In [89]:

```
data.std()
```

Out[89]:

```
carat          0.473777
cut            1.116593
color          1.701281
clarity        1.647017
depth          1.432318
table          2.234069
price       3987.215003
x              1.119402
y              1.111272
z              0.691620
volume        76.450353
dtype: float64
```

In [90]:

```
dataScaled = (data - data.mean())/data.std()
dataScaled
```

Out[90]:

	carat	cut	color	clarity	depth	table	price	x	
0	0.237060	-0.809805	0.936977	-1.245589	0.803198	-0.652146	-0.042112	0.364833	0.3928
1	-1.008252	0.981357	0.349185	-0.031272	0.454113	-1.099760	-0.824112	-1.180635	-1.1729
2	0.427022	0.085776	-0.238608	-0.638430	-1.151675	1.585923	-0.074466	0.632833	0.5998
3	1.482371	0.981357	-1.414192	-1.852747	-0.313872	-0.204532	0.442437	1.481501	1.4637
4	-0.037331	0.981357	-1.414192	-0.031272	-0.593140	-0.204532	-0.378437	0.212966	0.1678
...	...	...	...	...	...	...	...	...	
53912	0.870269	0.085776	1.524769	-1.245589	0.523930	-0.204532	0.645586	0.945500	0.8787
53913	0.025990	0.981357	-1.414192	-0.031272	-0.244055	-1.099760	-0.144439	0.266566	0.2938
53914	1.081339	0.981357	-0.826400	-1.245589	0.384297	-0.652146	0.575361	1.133100	1.1127
53915	-1.029359	0.981357	-0.826400	2.397362	0.384297	-1.099760	-0.787996	-1.261035	-1.2449
53916	-1.029359	-1.705387	1.524769	-0.638430	1.431550	-0.204532	-0.852201	-1.287835	-1.2629

53917 rows × 11 columns

In [91]:

```
dataScaled.describe()
```

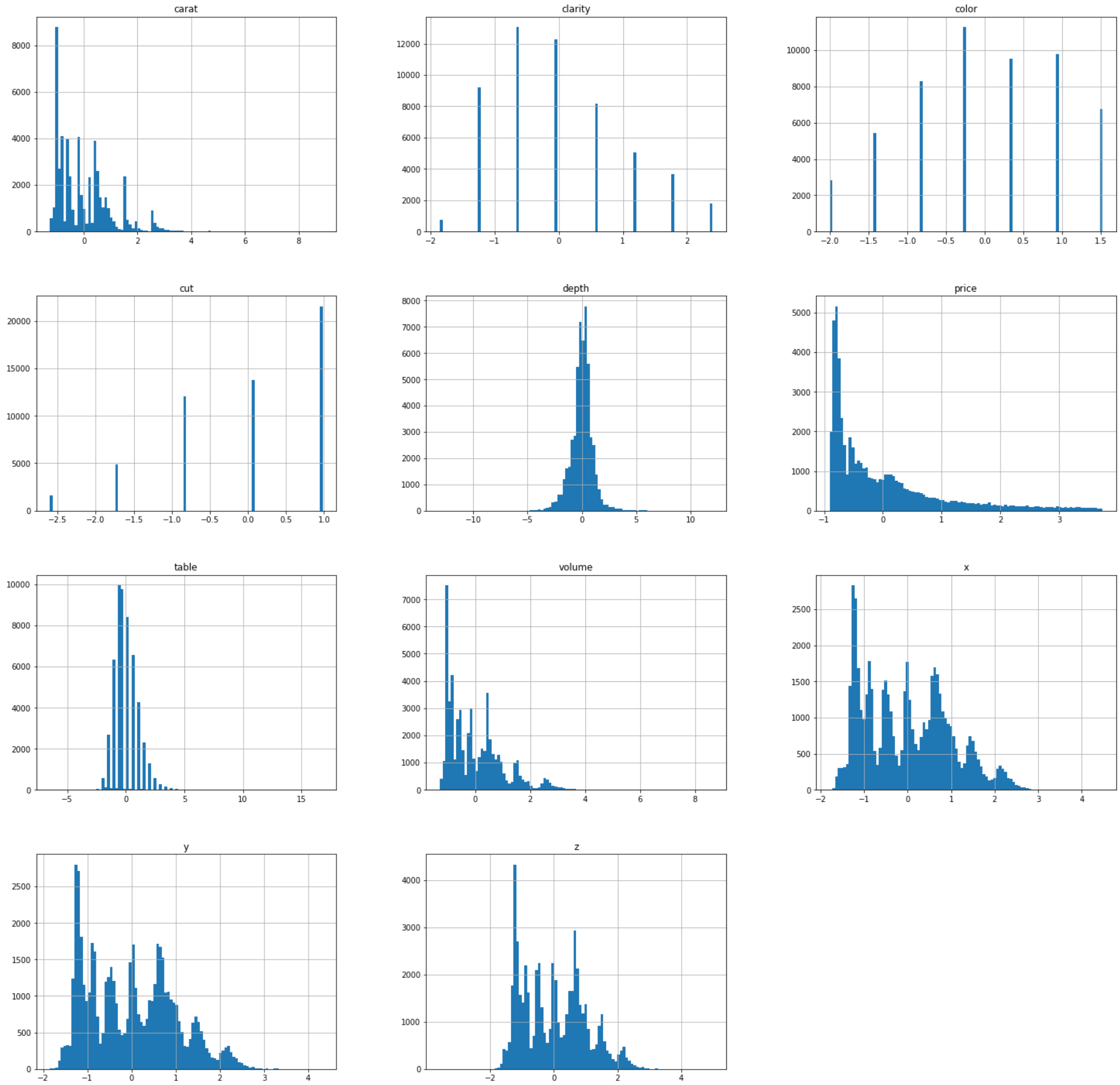
Out[91]:

	carat	cut	color	clarity	depth	table
count	5.391700e+04	5.391700e+04	5.391700e+04	5.391700e+04	5.391700e+04	5.391700e+04
mean	-1.567748e-14	-6.813261e-17	1.072726e-16	2.233748e-16	5.474005e-13	-7.010279e-15

	carat	cut	color	clarity	depth	table
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.261536e+00	-2.600968e+00	-2.001984e+00	-1.852747e+00	-1.309036e+01	-6.471125e+00
25%	-8.393963e-01	-8.098054e-01	-8.263999e-01	-6.384302e-01	-5.233230e-01	-6.521461e-01
50%	-2.061870e-01	8.577595e-02	-2.386077e-01	-3.127175e-02	3.521209e-02	-2.045324e-01
75%	5.114503e-01	9.813573e-01	9.369769e-01	5.758867e-01	5.239303e-01	6.906952e-01
max	8.890921e+00	9.813573e-01	1.524769e+00	2.397362e+00	1.204372e+01	1.680479e+01

In [92]:
```
hist = dataScaled.hist(figsize = (25,25), bins=100)
```

```
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:30
7: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:30
7: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:31
3: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
c:\program files\python38\lib\site-packages\pandas\plotting\_matplotlib\tools.py:31
3: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
releases later. Use ax.get_subplotspec().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
```



Preparing for Regression Learning:

In [93]:
```
y = dataScaled['price']
x = dataScaled.drop(['price', 'x', 'y', 'z'], axis=1)
```

In [94]:
```
x
```

Out[94]:

	carat	cut	color	clarity	depth	table	volume
0	0.237060	-0.809805	0.936977	-1.245589	0.803198	-0.652146	0.219853
1	-1.008252	0.981357	0.349185	-0.031272	0.454113	-1.099760	-0.992569
2	0.427022	0.085776	-0.238608	-0.638430	-1.151675	1.585923	0.383141
3	1.482371	0.981357	-1.414192	-1.852747	-0.313872	-0.204532	1.517876
4	-0.037331	0.981357	-1.414192	-0.031272	-0.593140	-0.204532	-0.024368
...	...	...	...	...	...	...	...
53912	0.870269	0.085776	1.524769	-1.245589	0.523930	-0.204532	0.817060
53913	0.025990	0.981357	-1.414192	-0.031272	-0.244055	-1.099760	0.075443
53914	1.081339	0.981357	-0.826400	-1.245589	0.384297	-0.652146	1.078269
53915	-1.029359	0.981357	-0.826400	2.397362	0.384297	-1.099760	-1.034188

	carat	cut	color	clarity	depth	table	volume
53916	-1.029359	-1.705387	1.524769	-0.638430	1.431550	-0.204532	-1.029678

53917 rows × 7 columns

In [95]:
```
y
```

Out[95]:
```
0       -0.042112
1       -0.824112
2       -0.074466
3        0.442437
4       -0.378437
           ...   
53912    0.645586
53913   -0.144439
53914    0.575361
53915   -0.787996
53916   -0.852201
Name: price, Length: 53917, dtype: float64
```

In [96]:
```
from math import log
y_log = log(y)
y_log
y.describe()
y_log = data['price'].map(log)
y_log.describe()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-96-ec778356e809> in <module>
      1 from math import log
----> 2 y_log = log(y)
      3 y_log
      4 y.describe()
      5 y_log = data['price'].map(log)

c:\program files\python38\lib\site-packages\pandas\core\series.py in wrapper(self)
    129         if len(self) == 1:
    130             return converter(self.iloc[0])
--> 131         raise TypeError("cannot convert the series to " "{0}".format(str(con
verter)))
    132 
    133     wrapper.__name__ = "__{name}__".format(name=converter.__name__)

TypeError: cannot convert the series to <class 'float'>
```

In [ ]:
```
y_log_avg = y_log.mean()
y_log_std = y_log.std()
y_log = (y_log-y_log_avg)/y_log_std
y_log
```

In [ ]:
```

```

Neural Network Model:

In [ ]:
```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
mlp = MLPRegressor()
mlp
```

In [ ]:
```
parameters = {
    'activation'          : ['tanh', 'relu', 'logistic'],
```

```
    #'alpha'              : [0.00001, 0.0001, 0.001, 0.01, 0.1],
    #'hidden_layer_sizes' : [(10,10), (1,1), (5,5), (20,20), (50,50)],
    #'max_iter'           : [500],
    #'verbose'            : [True],
    #'early_stopping'     : [True],
    #'tol'                : [0.000001]
}
```

In [ ]:
```
mlp_model = GridSearchCV(mlp, parameters, cv=3, scoring='r2', n_jobs= 8)
```

In [ ]:
```
#mlp_model.fit(x,y_log)
```

In [ ]:
```
mlp_model.get_params()
```

In [ ]:
```
mlp_model.score(x,y)
```

In [ ]:
```
#pd.DataFrame(mlp_model.loss_curve_).plot()

mlp_model.grid_scores_
```

In [ ]:
```
mlp_model.best_estimator_
```

In [ ]:
```
mlp_model.best_score_
```

In [ ]:
```
mlp_model.best_params_
```

In [ ]:
```
mlp_model.scorer_
```

In [ ]:
```

```

In [ ]:
```

```

Score History:

With XYZ:

0.981798388083403 'activation' : ['tanh'], 'alpha' : [0.0001], 'hidden_layer_sizes' : [(100,100)], 'max_iter' : [1000]

With Volume Only:

0.9786930586486694 'hidden_layer_sizes' : [(100,100)],

0.9785935703077155 'hidden_layer_sizes' : [(20,20)],

0.977229894682953 'hidden_layer_sizes' : [(10,10)],

0.9806389984481934 'hidden_layer_sizes' : [(10,10)], 'tol' : [0.000001],

0.7845885329932074 same as before but using log(y) instead of y for fit.

0.9810045927677672 {'activation': 'tanh', 'alpha': 1e-05, 'early_stopping': True, 'hidden_layer_sizes': (50, 50), 'max_iter': 500, 'tol': 1e-06, 'verbose': True}

In [ ]:
```

```