Ue01_Diamonds

November 14, 2022

[ ]:
```
import pandas as pd
import math

df = pd.read_csv('diamonds.csv')
df.describe()
```

1 Implementation Part 1 (50%): Diamond Prices

1.1 1. Give an overview of the dataset structure by answering those questions:

1.1.1 How many samples and features are in the dataset?

[2]:
```
print('Samples:', len(df))
print('Features:',len(df.columns))
```

```
Samples: 53940
Features: 10
```

1.1.2 What are the feature data types?

[3]:
```
df.dtypes
```

[3]:
```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price        int64
x          float64
y          float64
z          float64
dtype: object
```

1.1.3 Are diamonds balanced across color, cut and clarity? (Hint: roughly 1:1 means balanced, e.g. 1:2 is a "1:2 imbalance")

[4]:
```
color_df = df.groupby('color').size().to_frame('count')
color_df['balance'] = color_df['count'] / color_df['count'].min()
color_df = color_df.sort_values('balance')
print(color_df)
```

```
       count    balance
color
J       2808   1.000000
I       5422   1.930912
D       6775   2.412749
H       8304   2.957265
F       9542   3.398148
E       9797   3.488960
G      11292   4.021368
```

Compared to the lowest count J, every class has at least an imbalance of 1:2 and most of them 1:3. E and F for example are balanced if viewed separately without the other colors.

[5]:
```
cut_df = df.groupby('cut').size().to_frame('count')
cut_df['balance'] = cut_df['count'] / cut_df['count'].min()
cut_df = cut_df.sort_values('balance')
print(cut_df)
```

```
           count    balance
cut
Fair        1610   1.000000
Good        4906   3.047205
Very Good  12082   7.504348
Premium    13791   8.565839
Ideal      21551  13.385714
```

Compared to the fair cut every class is imbalanced and ideal has an imbalance of 1:13

[6]:
```
clarity_df = df.groupby('clarity').size().to_frame('count')
clarity_df['balance'] = clarity_df['count'] / clarity_df['count'].min()
clarity_df = clarity_df.sort_values('balance')
print(clarity_df)
```

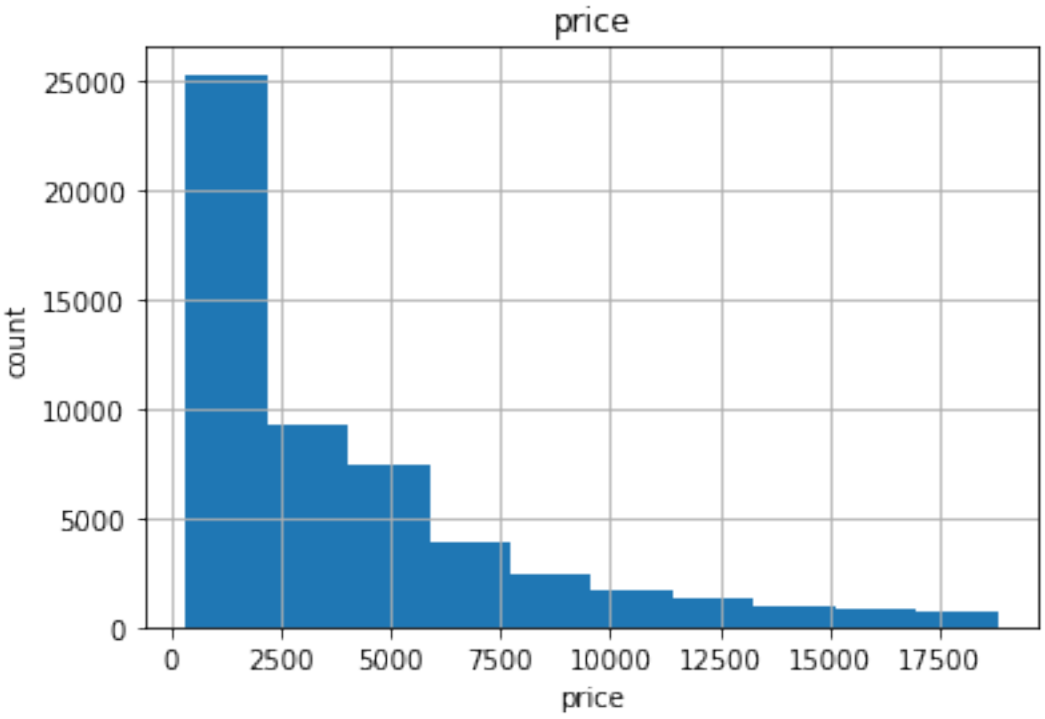
```
         count    balance
clarity
I1         741   1.000000
IF        1790   2.415655
VVS1      3655   4.932524
VVS2      5066   6.836707
VS1       8171  11.026991
SI2       9194  12.407557
VS2      12258  16.542510
```

```
SI1      13065  17.631579
```

The classes are extremely imbalanced, I1 vs VS2 has even an imbalance of 1:17.

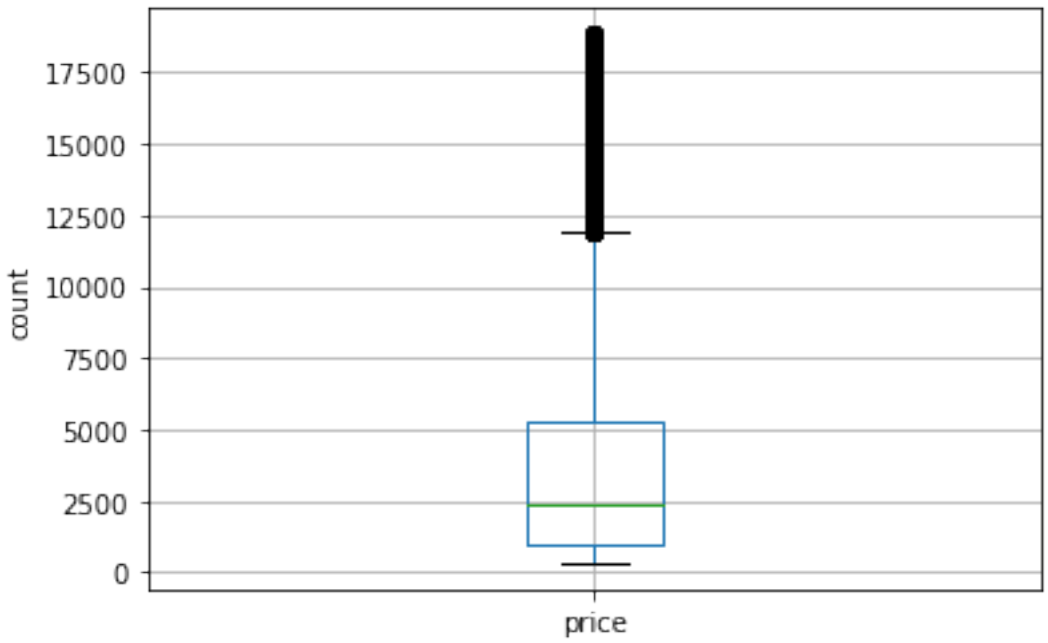
1.2 2. Visualize diamond prices using a histogram, boxplot and density plot.

[7]:
```
axarr = df.hist('price')
for ax in axarr.flatten():
    ax.set_xlabel("price")
    ax.set_ylabel("count")
```



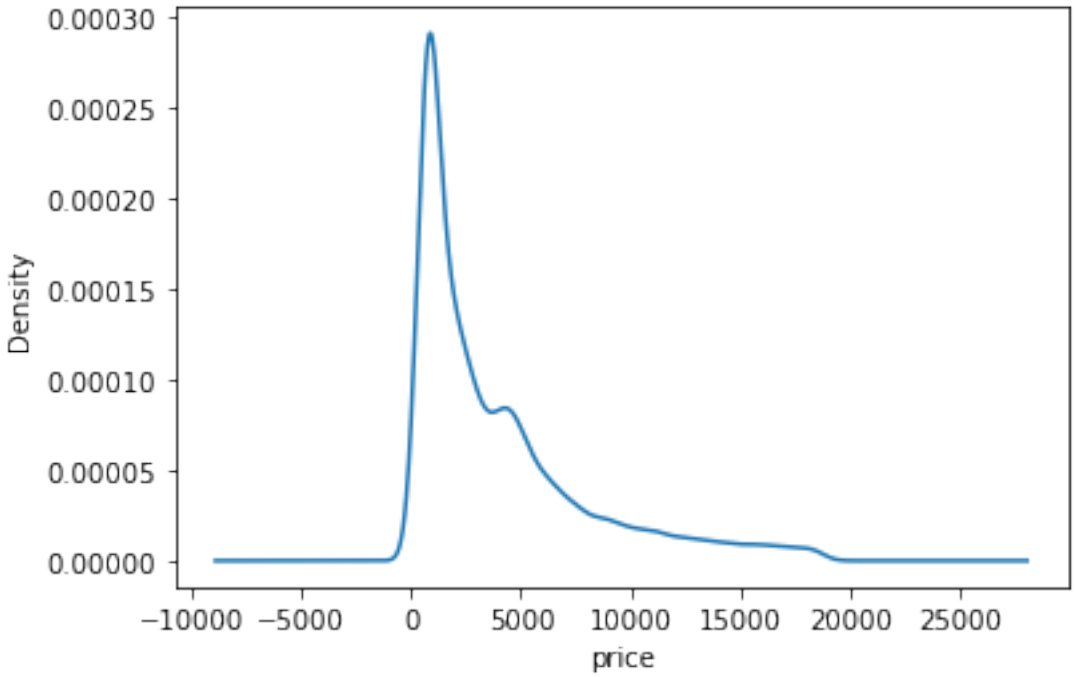
[8]:
```
axarr = df.boxplot('price')
axarr.set_ylabel("count")
```

[8]:
```
Text(0, 0.5, 'count')
```



[9]:
```
axarr = df['price'].plot.density()
axarr.set_xlabel("price")
```

[9]: Text(0.5, 0, 'price')



1.2.1 Answer this question: Is there trend visible in those plots? If yes, which is it and in which plots can you see it?

There exist much more cheap diamonds up to 2500 and the amount decreases exponential which is very good visible in the boxplot and histogram. The boxplot also shows that there are many outliers beyond 12000. 75% of the diamonds are below 5000.

1.3 3. Calculate and state the mean, median, standard deviation, median absolute deviation (MAD), 1st and 3rd quartile (Q1 and Q3), and inner quartile range of the diamond price.

• If you are not familiar with those functions: use Google, Wikipedia, etc. • Required commands are all in the provided script.

[10]:
```
print('Mean:', round(df['price'].mean(), 2))
print('Median:', round(df['price'].median(), 2))
print('STD: ', round(df['price'].std(), 2))
# the mad() function is deprecated
print('MAD: ', round((df['price'] - df['price'].mean()).abs().mean(), 2))
print('Q1:', df['price'].quantile(0.25))
print('Q3:', df['price'].quantile(0.75))
print('Inner quartile: ', df['price'].quantile(0.75) - df['price'].quantile(0.
↪25))
```

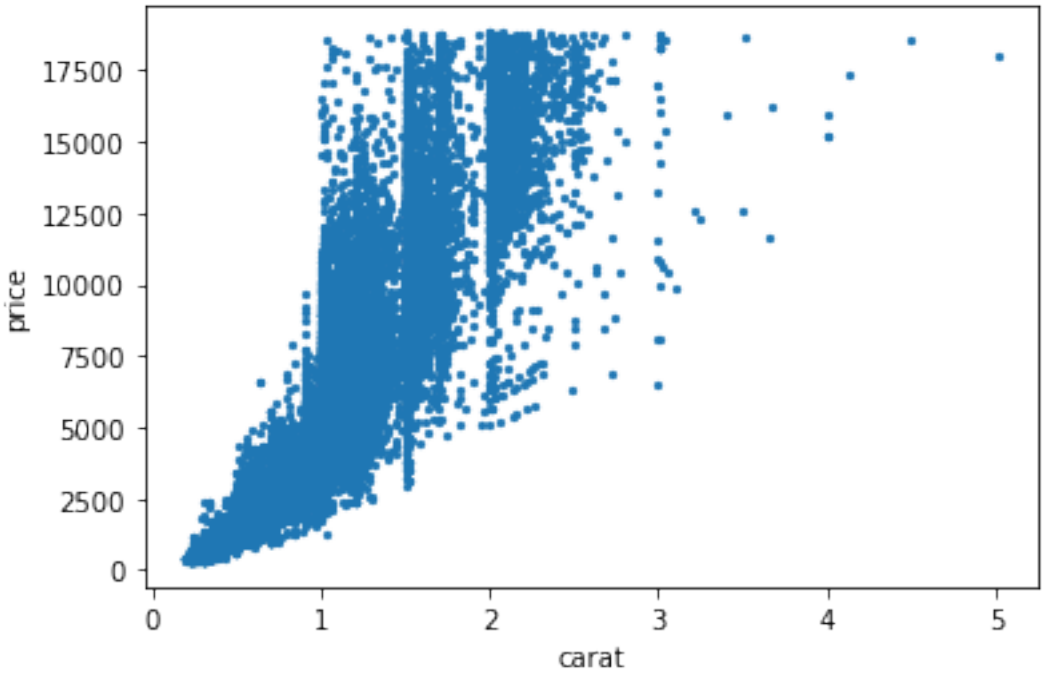
```
Mean: 3932.8
Median: 2401.0
STD:  3989.44
MAD:  3031.6
Q1: 950.0
Q3: 5324.25
Inner quartile:  4374.25
```

1.4 4. Plot the diamond price against the carat values as a scatterplot. Answer this question:

Hint: plotting many samples will be slow. Changing the plot symbol to '.' will cause a speedup.

[11]:
```
df.plot.scatter(x='carat', y='price', marker='.')
```

[11]:
```
<AxesSubplot:xlabel='carat', ylabel='price'>
```



1.4.1 Is there a trend visible in the plot? If yes, which is it?

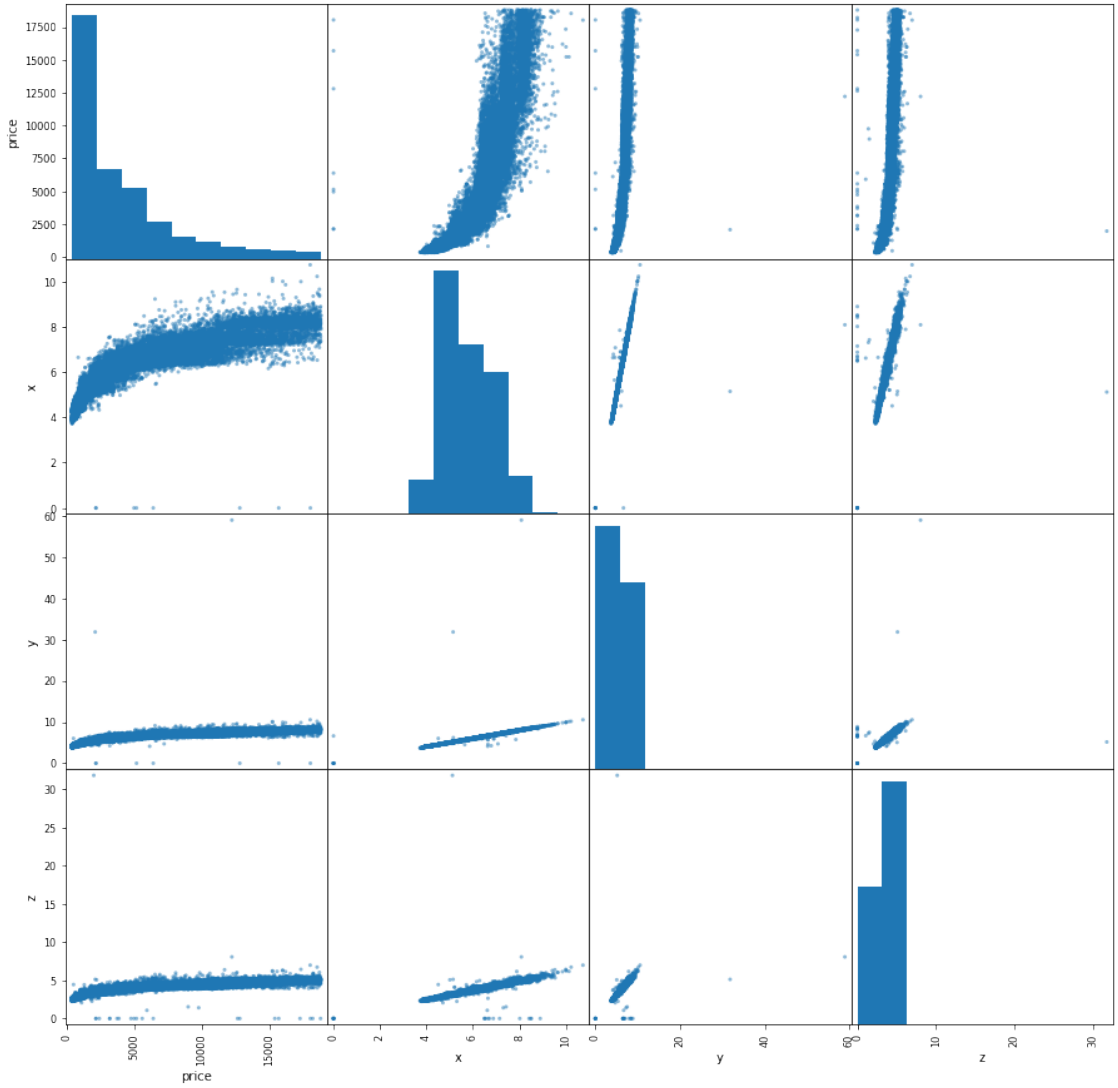
There is no direct connection between price and carat. It depends on more than the carat feature when you look on the spread of the points. Tendencial more carat leads to higher prices.

1.5 5. Analyze the correlation between diamond price and diamond x, y, and z dimensions.

1.5.1 Create pairwise plots for these features.

[12]:
```
pairwise_plot = df[['price', 'x', 'y', 'z']]
pd.plotting.scatter_matrix(pairwise_plot, figsize=(15,15))
print(pairwise_plot.corr())
```

```
           price         x         y         z
price   1.000000  0.884435  0.865421  0.861249
x       0.884435  1.000000  0.974701  0.970772
y       0.865421  0.974701  1.000000  0.952006
z       0.861249  0.970772  0.952006  1.000000
```



1.5.2 Is there a trend visible between x, y, and z? If yes, which is it?

Yes the dimensions all have a very high linear correlation, there is nearly a straight line.

1.5.3 Is there a trend visible between the dimensions and the price? If yes, which is it?

• Hint: if you don't know what a linear relation is (Google it!): – Linear correlation: feature A low –> feature B low, and feature A high –> feature B high. – (Inverse) linear correlation is also a linear correlation: feature A low –> feature B high, and feature A high –> feature B low: inverse linear correlation. Usually also just called linear correlation. – When plotting feature A against feature B and their points form a "straight line", then it's a linear relationship between A and B = linear correlation.

The larger the dimensions the greater the price. X spreads more than y or z. There is also a large

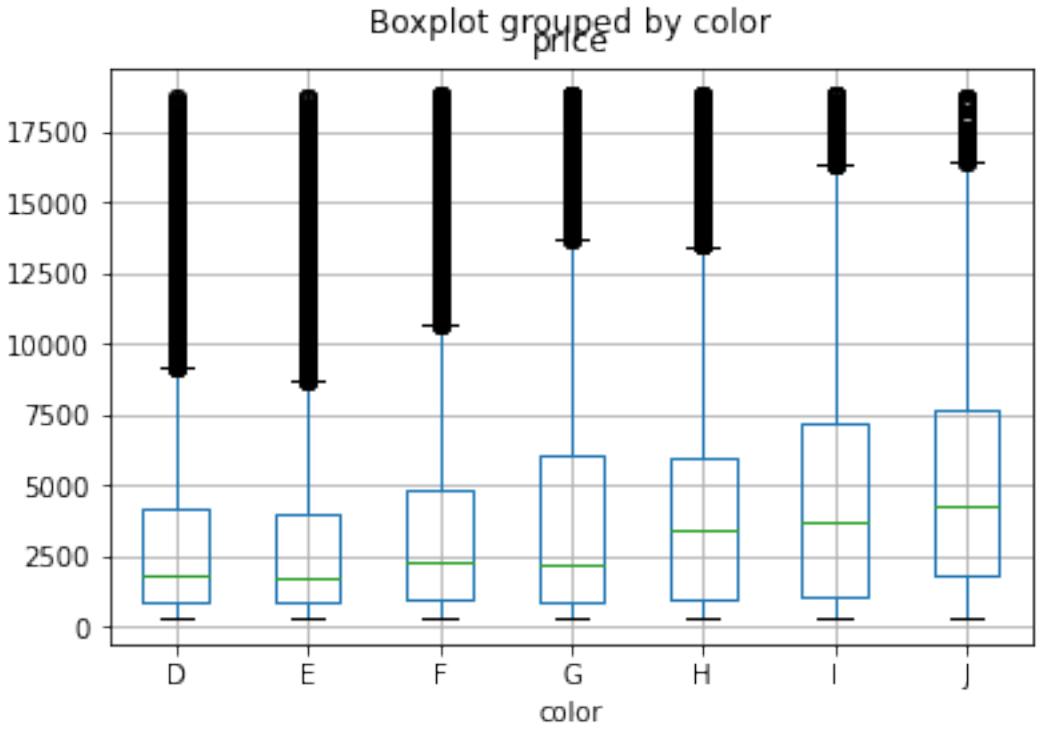
linear correlation between price and dimension, at least 86%.

1.6 6. Analyze diamond prices per diamond color.

1.6.1 Create boxplots showing diamond price boxes for each diamond color (all boxes should be in one figure).

[13]:
```
df.boxplot('price', by='color')
```

[13]:
```
<AxesSubplot:title={'center':'price'}, xlabel='color'>
```

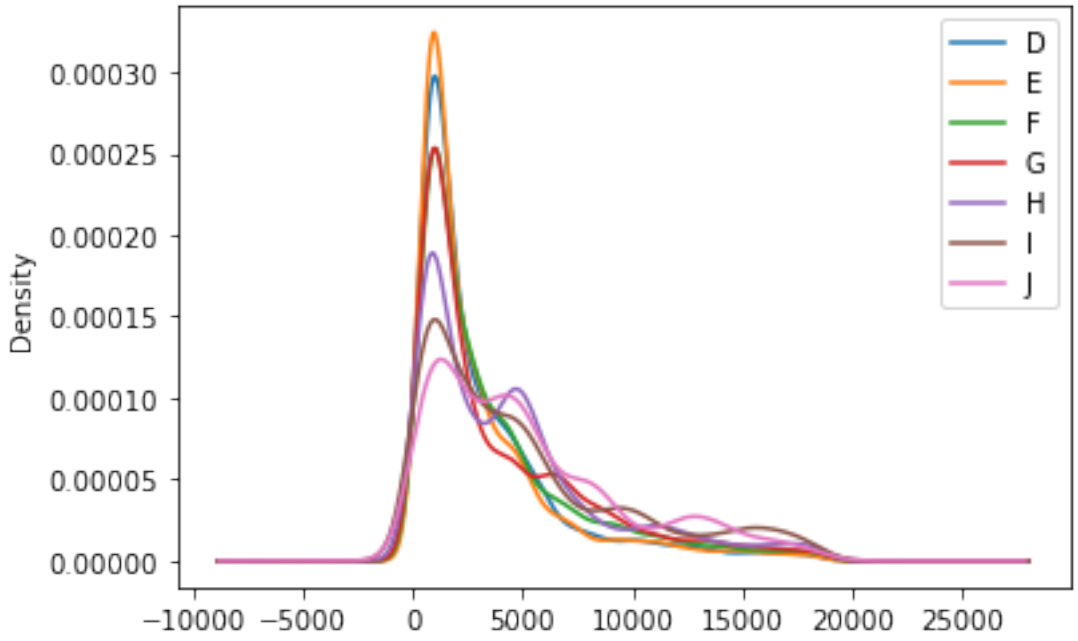


1.6.2 Create densityplots showing diamond prices for each diamond color (all densities should be in one figure).

[14]:
```
df.groupby('color')['price'].plot.density(legend=True)
```

[14]:
```
color
D    AxesSubplot(0.125,0.125;0.775x0.755)
E    AxesSubplot(0.125,0.125;0.775x0.755)
F    AxesSubplot(0.125,0.125;0.775x0.755)
G    AxesSubplot(0.125,0.125;0.775x0.755)
H    AxesSubplot(0.125,0.125;0.775x0.755)
I    AxesSubplot(0.125,0.125;0.775x0.755)
```

```
J    AxesSubplot(0.125,0.125;0.775x0.755)
Name: price, dtype: object
```



1.6.3 Answer this question: is there a trend visible? If yes, which one?

Most of the diamond colors are sold for approximately the same price. Depending on the color there exist more or less diamonds. Type E for example looks very stable and there are no peaks.

1.7 7. Use vectorized commands (= no loops!) to answer these questions:

1.7.1 How many diamonds have a price above 9500?

[15]:
```
df[df['price'] > 9500].count()
```

[15]:
```
carat      5734
cut        5734
color      5734
clarity    5734
depth      5734
table      5734
price      5734
x          5734
y          5734
z          5734
dtype: int64
```

1.7.2 How many diamonds have a price above 9500 and have color "D"?

[16]:
```
df[(df['price'] > 9500) & (df['color'] == 'D')].count()
```

[16]:
```
carat      461
cut        461
color      461
clarity    461
depth      461
table      461
price      461
x          461
y          461
z          461
dtype: int64
```

1.7.3 What is the mean and std of the price of all color "D" diamonds with cut "Fair"?

[17]:
```
df[(df['cut'] == 'Fair') & (df['color'] == 'D')]['price'].mean()
```

[17]:
```
4291.061349693252
```

[18]:
```
df[(df['cut'] == 'Fair') & (df['color'] == 'D')]['price'].std()
```

[18]:
```
3286.114238174996
```

1.7.4 What is the median and mad of the price of all color "J" diamonds with cut "Ideal"?

[19]:
```
dfIdeal = df[(df['cut'] == 'Ideal') & (df['color'] == 'J')]['price']
print('Median: ')
print(dfIdeal.median())
print('Ideal: ')
print((dfIdeal - dfIdeal.mean()).abs().mean())
```

```
Median:
4096.0
Ideal:
3467.586682378051
```

1.7.5 Create two copies of the dataframe that contains only the price and carat feature. Apply a log with base 10 to both features in one of those dataframes, and square (x' = x^2) the features in the other dataframe. What is the mean and std of the transformed features in both dataframes?

[20]:
```
dfSquare = df[['carat', 'price']].applymap(func=lambda x: x ** 2)
print(dfSquare.mean())
print(dfSquare.std())
```

```
carat    8.613903e-01
price    3.138225e+07
dtype: float64
carat    1.056506e+00
price    6.049189e+07
dtype: float64
```

[21]:
```
dfLog = df[['carat', 'price']].applymap(func=math.log10)
print(dfLog.mean())
print(dfLog.std())
```

```
carat   -0.171532
price    3.381751
dtype: float64
carat    0.253987
price    0.440657
dtype: float64
```

[21]:
```

```