# 1. import important libraries

In [73]:

```
pip install xgboost
```

Note: you may need to restart the kernel to use updated packages.Collecting xgboost

In [74]:

```
import numpy as np
import warnings
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor

sns.set_style('darkgrid')
warnings.filterwarnings('ignore')
```

```
  Downloading xgboost-1.7.2-py3-none-win_amd64.whl (89.1 MB)
Requirement already satisfied: numpy in c:\users\lucky computers\anaconda3\lib\site-packages (from xgboost) (1.20.3)
Requirement already satisfied: scipy in c:\users\lucky computers\anaconda3\lib\site-packages (from xgboost) (1.7.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.2
```

# 2. Reading dataset

In [19]:

```
df = pd.read_csv("C:/Users/Lucky Computers/Downloads/diamonds.csv")
```

In [20]:

```
df
```

Out[20]:

| | Unnamed: 0 | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55.0 | 326 | 3.95 | 3.98 | 2.43 |
| **1** | 2 | 0.21 | Premium | E | SI1 | 59.8 | 61.0 | 326 | 3.89 | 3.84 | 2.31 |
| **2** | 3 | 0.23 | Good | E | VS1 | 56.9 | 65.0 | 327 | 4.05 | 4.07 | 2.31 |
| **3** | 4 | 0.29 | Premium | I | VS2 | 62.4 | 58.0 | 334 | 4.20 | 4.23 | 2.63 |
| **4** | 5 | 0.31 | Good | J | SI2 | 63.3 | 58.0 | 335 | 4.34 | 4.35 | 2.75 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **53935** | 53936 | 0.72 | Ideal | D | SI1 | 60.8 | 57.0 | 2757 | 5.75 | 5.76 | 3.50 |
| **53936** | 53937 | 0.72 | Good | D | SI1 | 63.1 | 55.0 | 2757 | 5.69 | 5.75 | 3.61 |
| **53937** | 53938 | 0.70 | Very Good | D | SI1 | 62.8 | 60.0 | 2757 | 5.66 | 5.68 | 3.56 |
| **53938** | 53939 | 0.86 | Premium | H | SI2 | 61.0 | 58.0 | 2757 | 6.15 | 6.12 | 3.74 |
| **53939** | 53940 | 0.75 | Ideal | D | SI2 | 62.2 | 55.0 | 2757 | 5.83 | 5.87 | 3.64 |

53940 rows × 11 columns

In [21]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  53940 non-null  int64  
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object 
 3   color       53940 non-null  object 
 4   clarity     53940 non-null  object 
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64  
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

# 3. Statistical Summary

In [22]:

```
df.describe()
```

Out[22]:

| | Unnamed: 0 | carat | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|
| **count** | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 |
| **mean** | 26970.500000 | 0.797940 | 61.749405 | 57.457184 | 3932.799722 | 5.731157 | 5.734526 | 3.538734 |
| **std** | 15571.281097 | 0.474011 | 1.432621 | 2.234491 | 3989.439738 | 1.121761 | 1.142135 | 0.705699 |
| **min** | 1.000000 | 0.200000 | 43.000000 | 43.000000 | 326.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 13485.750000 | 0.400000 | 61.000000 | 56.000000 | 950.000000 | 4.710000 | 4.720000 | 2.910000 |
| **50%** | 26970.500000 | 0.700000 | 61.800000 | 57.000000 | 2401.000000 | 5.700000 | 5.710000 | 3.530000 |
| **75%** | 40455.250000 | 1.040000 | 62.500000 | 59.000000 | 5324.250000 | 6.540000 | 6.540000 | 4.040000 |
| **max** | 53940.000000 | 5.010000 | 79.000000 | 95.000000 | 18823.000000 | 10.740000 | 58.900000 | 31.800000 |

### Displaying first 5 rows

In [23]:

```
df.head()
```

Out[23]:

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

### shape of data

In [24]:

```
df.shape
```

Out[24]:

(53940, 11)

# 4. Checking duplicate rows

In [25]:

```
df[df.duplicated()]
```

Out[25]:

| Unnamed: 0 | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|

In [26]:

```
len(df[df.duplicated()])
```

Out[26]:

0

## dropping duplicate rows

In [27]:

```
df.drop_duplicates(inplace=True)
```

# 5. Checking null values

In [28]:

```
df.isnull().sum()
```

Out[28]:

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

# 6. Dropingg Unnamed column from data

In [29]:

```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
```

# describe data

In [30]:

```
df.describe()
```

Out[30]:

| | carat | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|
| **count** | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 | 53940.000000 |
| **mean** | 0.797940 | 61.749405 | 57.457184 | 3932.799722 | 5.731157 | 5.734526 | 3.538734 |
| **std** | 0.474011 | 1.432621 | 2.234491 | 3989.439738 | 1.121761 | 1.142135 | 0.705699 |
| **min** | 0.200000 | 43.000000 | 43.000000 | 326.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.400000 | 61.000000 | 56.000000 | 950.000000 | 4.710000 | 4.720000 | 2.910000 |
| **50%** | 0.700000 | 61.800000 | 57.000000 | 2401.000000 | 5.700000 | 5.710000 | 3.530000 |
| **75%** | 1.040000 | 62.500000 | 59.000000 | 5324.250000 | 6.540000 | 6.540000 | 4.040000 |
| **max** | 5.010000 | 79.000000 | 95.000000 | 18823.000000 | 10.740000 | 58.900000 | 31.800000 |

# 7. After use df.describe(), we could see that the minimun value of x, y and z are equal to zero. Let's take a look at them.

In [31]:

```
dictn = {"carat" : "{:.2f}", "depth" : "{:.1f}", "table" : "{:.1f}", "x" : "{:.2f}", "y" : "{:.2f}", "z" : "{:.2f}"}
df0 = df.loc[(df["x"] == 0) | (df["y"] == 0) | (df["z"] == 0)]
df0.style.apply(lambda x: ["background: yellow" if n == 0 else "" for n in x], axis = 1).format(dictn)
```

Out[31]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **2207** | 1.00 | Premium | G | SI2 | 59.1 | 59.0 | 3142 | 6.55 | 6.48 | 0.00 |
| **2314** | 1.01 | Premium | H | I1 | 58.1 | 59.0 | 3167 | 6.66 | 6.60 | 0.00 |
| **4791** | 1.10 | Premium | G | SI2 | 63.0 | 59.0 | 3696 | 6.50 | 6.47 | 0.00 |
| **5471** | 1.01 | Premium | F | SI2 | 59.2 | 58.0 | 3837 | 6.50 | 6.47 | 0.00 |
| **10167** | 1.50 | Good | G | I1 | 64.0 | 61.0 | 4731 | 7.15 | 7.04 | 0.00 |
| **11182** | 1.07 | Ideal | F | SI2 | 61.6 | 56.0 | 4954 | 0.00 | 6.62 | 0.00 |
| **11963** | 1.00 | Very Good | H | VS2 | 63.3 | 53.0 | 5139 | 0.00 | 0.00 | 0.00 |
| **13601** | 1.15 | Ideal | G | VS2 | 59.2 | 56.0 | 5564 | 6.88 | 6.83 | 0.00 |
| **15951** | 1.14 | Fair | G | VS1 | 57.5 | 67.0 | 6381 | 0.00 | 0.00 | 0.00 |
| **24394** | 2.18 | Premium | H | SI2 | 59.4 | 61.0 | 12631 | 8.49 | 8.45 | 0.00 |
| **24520** | 1.56 | Ideal | G | VS2 | 62.2 | 54.0 | 12800 | 0.00 | 0.00 | 0.00 |
| **26123** | 2.25 | Premium | I | SI1 | 61.3 | 58.0 | 15397 | 8.52 | 8.42 | 0.00 |
| **26243** | 1.20 | Premium | D | VVS1 | 62.1 | 59.0 | 15686 | 0.00 | 0.00 | 0.00 |
| **27112** | 2.20 | Premium | H | SI1 | 61.2 | 59.0 | 17265 | 8.42 | 8.37 | 0.00 |
| **27429** | 2.25 | Premium | H | SI2 | 62.8 | 59.0 | 18034 | 0.00 | 0.00 | 0.00 |
| **27503** | 2.02 | Premium | H | VS2 | 62.7 | 53.0 | 18207 | 8.02 | 7.95 | 0.00 |
| **27739** | 2.80 | Good | G | SI2 | 63.8 | 58.0 | 18788 | 8.90 | 8.85 | 0.00 |
| **49556** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | 0.00 | 0.00 | 0.00 |
| **49557** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | 0.00 | 0.00 | 0.00 |
| **51506** | 1.12 | Premium | G | I1 | 60.4 | 59.0 | 2383 | 6.71 | 6.67 | 0.00 |

# Transforming them into NaN values

In [32]:

```
df.loc[df["x"] == 0, "x"] = np.nan
df.loc[df["y"] == 0, "y"] = np.nan
df.loc[df["z"] == 0, "z"] = np.nan
```

# Seeing the number of the new missing values

In [35]:

```
df[["x", "y", "z"]].isnull().sum()
```

Out[35]:

```
x     8
y     7
z    20
dtype: int64
```

# 8. filling null values

In [36]:

```
def filling_null_values(col):
    c = df.groupby(["carat"])[col].median()
    idx = list(df.loc[df[col].isnull() == True].sort_values(by = "carat", ascending = False).index)
    for i in idx:
        cv = df.loc[i, "carat"]
        val = c[cv]
        df.loc[i, col] = val
        print("carat: {} / median {} value: {}".format(cv, col, val))
    return df.iloc[idx].style.applymap(lambda x: "background-color: limegreen", subset = col).format(dictn)
```

## filling null valus in y

In [37]:

```
filling_null_values("y")
```

carat: 2.25 / median y value: 8.39
carat: 1.56 / median y value: 7.46
carat: 1.2 / median y value: 6.79
carat: 1.14 / median y value: 6.72
carat: 1.0 / median y value: 6.38
carat: 0.71 / median y value: 5.73
carat: 0.71 / median y value: 5.73

Out[37]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **27429** | 2.25 | Premium | H | SI2 | 62.8 | 59.0 | 18034 | nan | 8.39 | nan |
| **24520** | 1.56 | Ideal | G | VS2 | 62.2 | 54.0 | 12800 | nan | 7.46 | nan |
| **26243** | 1.20 | Premium | D | VVS1 | 62.1 | 59.0 | 15686 | nan | 6.79 | nan |
| **15951** | 1.14 | Fair | G | VS1 | 57.5 | 67.0 | 6381 | nan | 6.72 | nan |
| **11963** | 1.00 | Very Good | H | VS2 | 63.3 | 53.0 | 5139 | nan | 6.38 | nan |
| **49556** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | nan | 5.73 | nan |
| **49557** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | nan | 5.73 | nan |

## filling null values in x

In [38]:

```
filling_null_values("x")
```

carat: 2.25 / median x value: 8.47
carat: 1.56 / median x value: 7.46
carat: 1.2 / median x value: 6.78
carat: 1.14 / median x value: 6.71
carat: 1.07 / median x value: 6.57
carat: 1.0 / median x value: 6.38
carat: 0.71 / median x value: 5.72
carat: 0.71 / median x value: 5.72

Out[38]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **27429** | 2.25 | Premium | H | SI2 | 62.8 | 59.0 | 18034 | 8.47 | 8.39 | nan |
| **24520** | 1.56 | Ideal | G | VS2 | 62.2 | 54.0 | 12800 | 7.46 | 7.46 | nan |
| **26243** | 1.20 | Premium | D | VVS1 | 62.1 | 59.0 | 15686 | 6.78 | 6.79 | nan |
| **15951** | 1.14 | Fair | G | VS1 | 57.5 | 67.0 | 6381 | 6.71 | 6.72 | nan |
| **11182** | 1.07 | Ideal | F | SI2 | 61.6 | 56.0 | 4954 | 6.57 | 6.62 | nan |
| **11963** | 1.00 | Very Good | H | VS2 | 63.3 | 53.0 | 5139 | 6.38 | 6.38 | nan |
| **49556** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | 5.72 | 5.73 | nan |
| **49557** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | 5.72 | 5.73 | nan |

## filling null values in z

In [39]:

```
filling_null_values("z")
```

carat: 2.8 / median z value: 5.5
carat: 2.25 / median z value: 5.19
carat: 2.25 / median z value: 5.19
carat: 2.2 / median z value: 5.17
carat: 2.18 / median z value: 5.16
carat: 2.02 / median z value: 5.0
carat: 1.56 / median z value: 4.59
carat: 1.5 / median z value: 4.53
carat: 1.2 / median z value: 4.21
carat: 1.15 / median z value: 4.16
carat: 1.14 / median z value: 4.14
carat: 1.12 / median z value: 4.11
carat: 1.1 / median z value: 4.09
carat: 1.07 / median z value: 4.05
carat: 1.01 / median z value: 3.98
carat: 1.01 / median z value: 3.98
carat: 1.0 / median z value: 3.96
carat: 1.0 / median z value: 3.96
carat: 0.71 / median z value: 3.54
carat: 0.71 / median z value: 3.54

Out[39]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **27739** | 2.80 | Good | G | SI2 | 63.8 | 58.0 | 18788 | 8.90 | 8.85 | 5.50 |
| **26123** | 2.25 | Premium | I | SI1 | 61.3 | 58.0 | 15397 | 8.52 | 8.42 | 5.19 |
| **27429** | 2.25 | Premium | H | SI2 | 62.8 | 59.0 | 18034 | 8.47 | 8.39 | 5.19 |
| **27112** | 2.20 | Premium | H | SI1 | 61.2 | 59.0 | 17265 | 8.42 | 8.37 | 5.17 |
| **24394** | 2.18 | Premium | H | SI2 | 59.4 | 61.0 | 12631 | 8.49 | 8.45 | 5.16 |
| **27503** | 2.02 | Premium | H | VS2 | 62.7 | 53.0 | 18207 | 8.02 | 7.95 | 5.00 |
| **24520** | 1.56 | Ideal | G | VS2 | 62.2 | 54.0 | 12800 | 7.46 | 7.46 | 4.59 |
| **10167** | 1.50 | Good | G | I1 | 64.0 | 61.0 | 4731 | 7.15 | 7.04 | 4.53 |
| **26243** | 1.20 | Premium | D | VVS1 | 62.1 | 59.0 | 15686 | 6.78 | 6.79 | 4.21 |
| **13601** | 1.15 | Ideal | G | VS2 | 59.2 | 56.0 | 5564 | 6.88 | 6.83 | 4.16 |
| **15951** | 1.14 | Fair | G | VS1 | 57.5 | 67.0 | 6381 | 6.71 | 6.72 | 4.14 |
| **51506** | 1.12 | Premium | G | I1 | 60.4 | 59.0 | 2383 | 6.71 | 6.67 | 4.11 |
| **4791** | 1.10 | Premium | G | SI2 | 63.0 | 59.0 | 3696 | 6.50 | 6.47 | 4.09 |
| **11182** | 1.07 | Ideal | F | SI2 | 61.6 | 56.0 | 4954 | 6.57 | 6.62 | 4.05 |
| **2314** | 1.01 | Premium | H | I1 | 58.1 | 59.0 | 3167 | 6.66 | 6.60 | 3.98 |
| **5471** | 1.01 | Premium | F | SI2 | 59.2 | 58.0 | 3837 | 6.50 | 6.47 | 3.98 |
| **11963** | 1.00 | Very Good | H | VS2 | 63.3 | 53.0 | 5139 | 6.38 | 6.38 | 3.96 |
| **2207** | 1.00 | Premium | G | SI2 | 59.1 | 59.0 | 3142 | 6.55 | 6.48 | 3.96 |
| **49556** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | 5.72 | 5.73 | 3.54 |
| **49557** | 0.71 | Good | F | SI2 | 64.1 | 60.0 | 2130 | 5.72 | 5.73 | 3.54 |

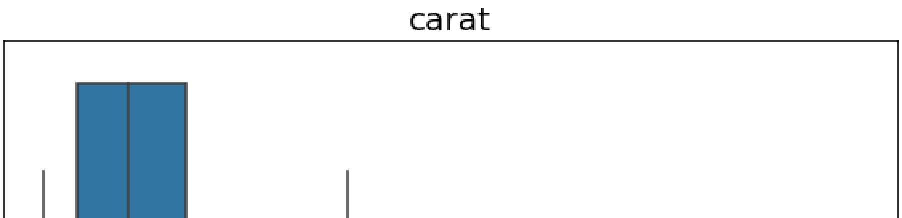
# 9. Visualize the data

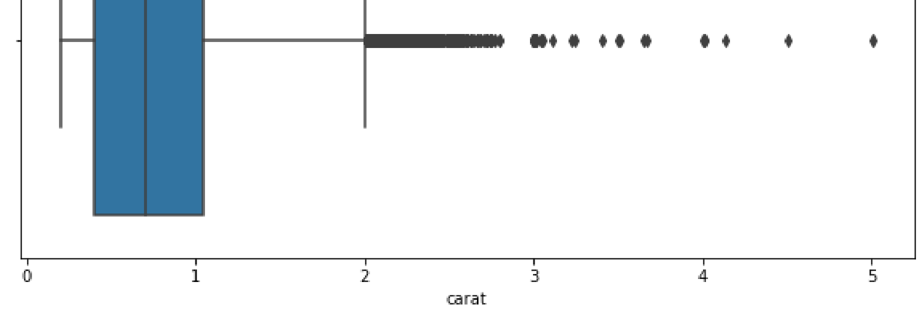
# boxplot

In [41]:

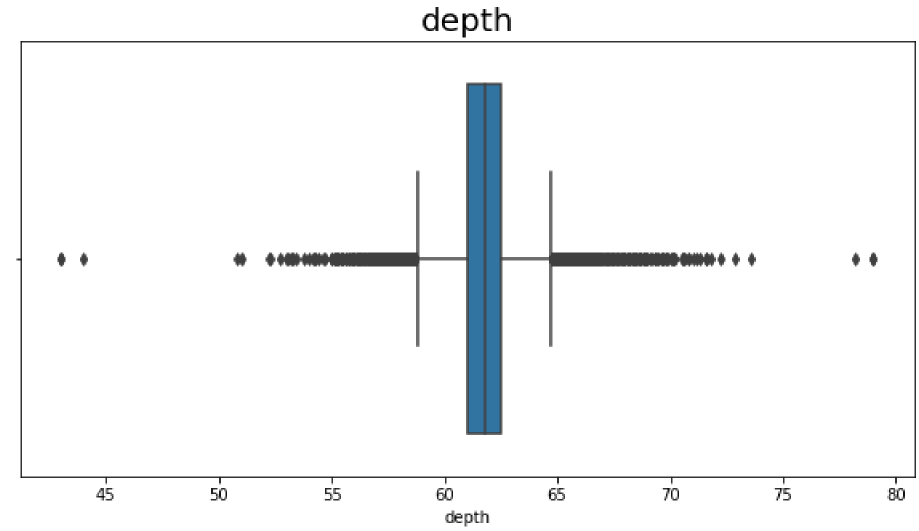
```
for c in ['carat', 'depth', 'table', 'price', 'x', 'y', 'z']:
    plt.figure(figsize=(10, 5))
    sns.boxplot(df[c])
    plt.title(c, fontsize=20)
    plt.show()
```

C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(

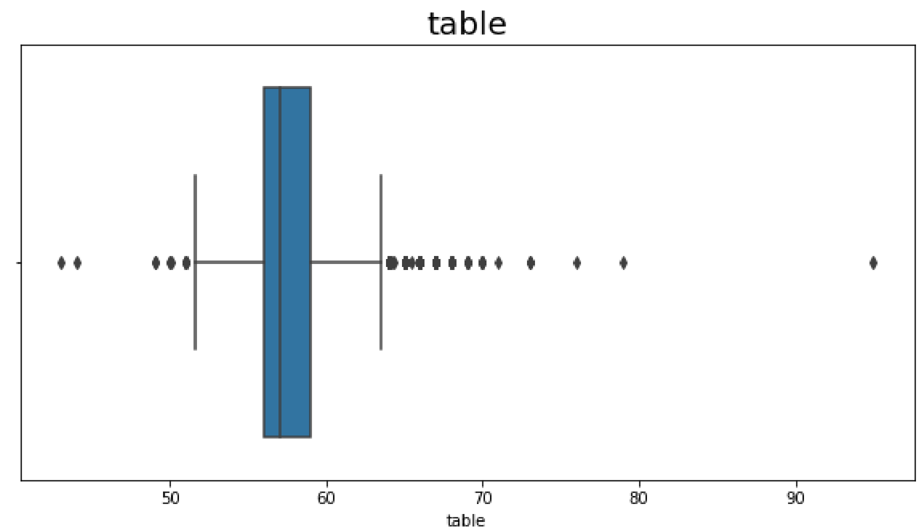




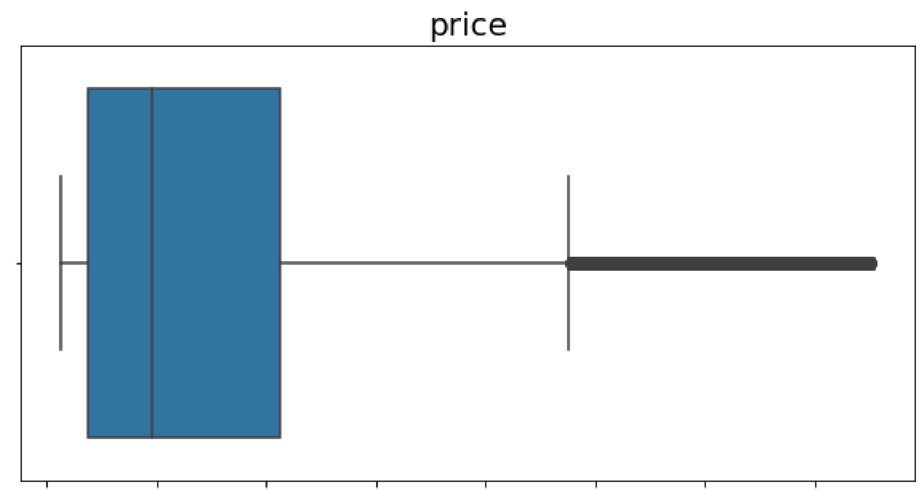

C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(



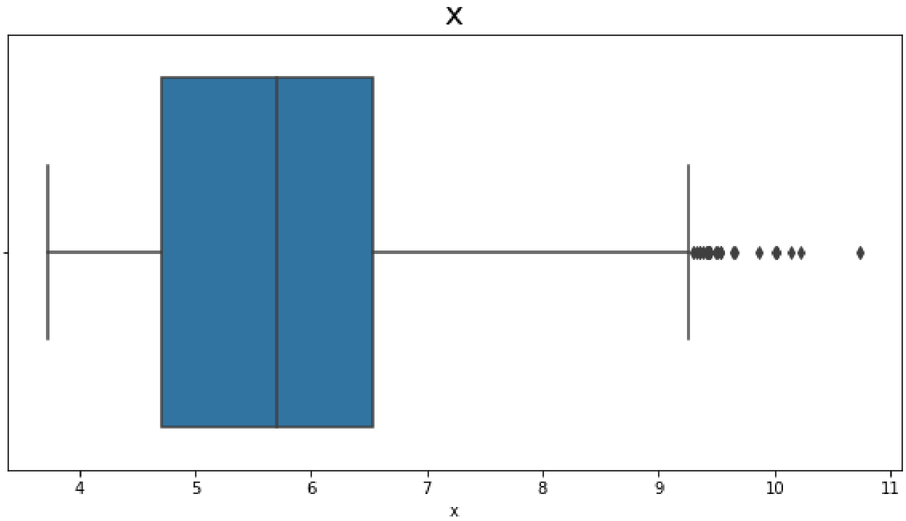

C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(



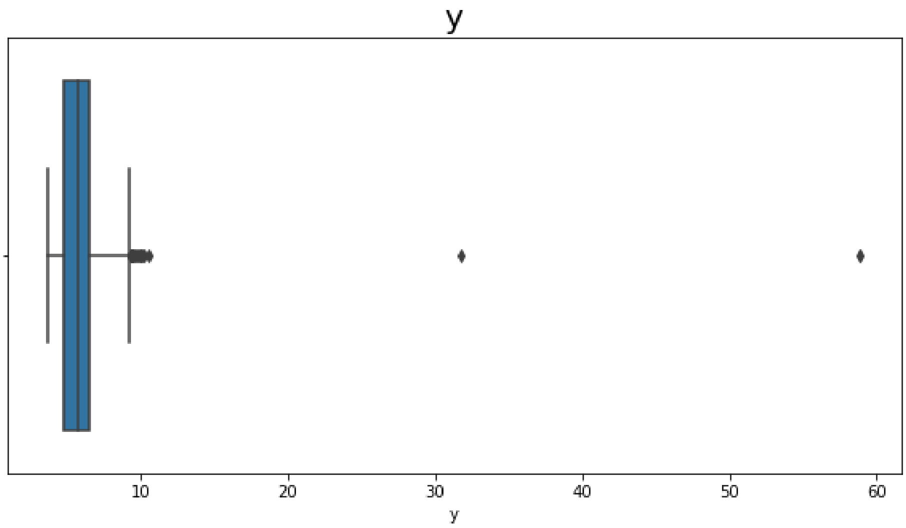

C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(



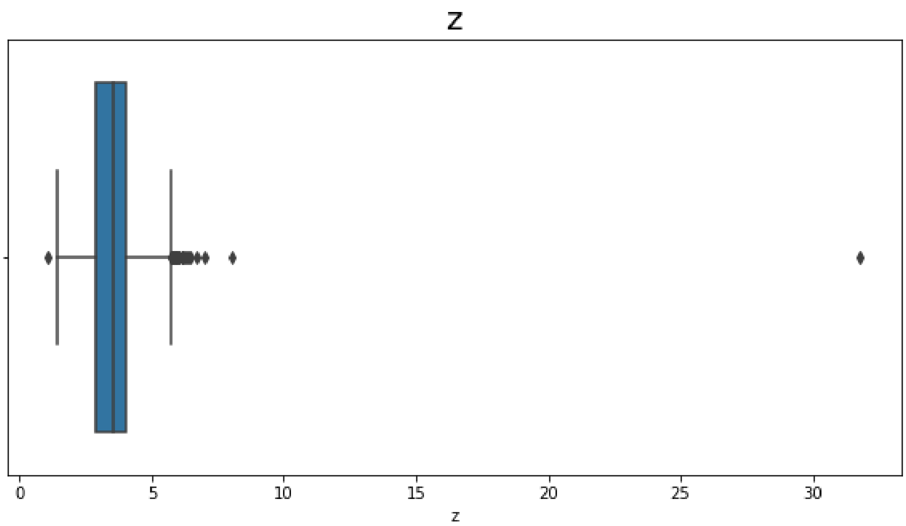

C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(




C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(




C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(




# 10. checking outliers

In [42]:

```
def outliers(out, col):
```

```
    out_idx = out.index
    i = pd.IndexSlice[out_idx, col]
    return out.style.applymap(lambda x: "background-color: red", subset = i).format(dictn)
```

In [43]:

```
df_out= df.loc[df["y"] > 30].copy()
outliers(df_out, "y")
```

Out[43]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **24067** | 2.00 | Premium | H | SI2 | 58.9 | 57.0 | 12210 | 8.09 | 58.90 | 8.06 |
| **49189** | 0.51 | Ideal | E | VS1 | 61.8 | 55.0 | 2075 | 5.15 | 31.80 | 5.12 |

In [44]:

```
df_out = df.loc[df["z"] > 30].copy()
outliers(df_out, "z")
```

Out[44]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **48410** | 0.51 | Very Good | E | VS1 | 61.8 | 54.7 | 1970 | 5.12 | 5.15 | 31.80 |

# filling outliers values with np.nan in x and y

In [45]:

```
df.loc[df["y"] > 30, "y"] = np.nan
df.loc[df["z"] > 30, "z"] = np.nan
```

## filling null values in y

In [46]:

```
filling_null_values("y")
```

carat: 2.0 / median y value: 8.01
carat: 0.51 / median y value: 5.14

Out[46]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **24067** | 2.00 | Premium | H | SI2 | 58.9 | 57.0 | 12210 | 8.09 | 8.01 | 8.06 |
| **49189** | 0.51 | Ideal | E | VS1 | 61.8 | 55.0 | 2075 | 5.15 | 5.14 | 5.12 |

## filling null values in z

In [47]:

```
filling_null_values('z')
```

carat: 0.51 / median z value: 3.17

Out[47]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **48410** | 0.51 | Very Good | E | VS1 | 61.8 | 54.7 | 1970 | 5.12 | 5.15 | 3.17 |

## checking outiers in depth column

In [48]:

```
df_out = df.loc[(df["depth"] > 75) | (df["depth"] < 45)].copy()
outliers(df_out, "depth")
```

Out[48]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **4518** | 1.00 | Fair | G | SI1 | 43.0 | 59.0 | 3634 | 6.32 | 6.27 | 3.97 |
| **6341** | 1.00 | Fair | G | VS2 | 44.0 | 53.0 | 4032 | 6.31 | 6.24 | 4.12 |
| **10377** | 1.09 | Ideal | J | VS2 | 43.0 | 54.0 | 4778 | 6.53 | 6.55 | 4.12 |
| **41918** | 1.03 | Fair | E | I1 | 78.2 | 54.0 | 1262 | 5.72 | 5.59 | 4.42 |
| **52860** | 0.50 | Fair | E | VS2 | 79.0 | 73.0 | 2579 | 5.21 | 5.18 | 4.09 |
| **52861** | 0.50 | Fair | E | VS2 | 79.0 | 73.0 | 2579 | 5.21 | 5.18 | 4.09 |

## checking outliers in table column

In [50]:

```
df_out = df.loc[(df["table"] > 90) | (df["depth"] < 45)].copy()
outliers(df_out, "table")
```

Out[50]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **4518** | 1.00 | Fair | G | SI1 | 43.0 | 59.0 | 3634 | 6.32 | 6.27 | 3.97 |
| **6341** | 1.00 | Fair | G | VS2 | 44.0 | 53.0 | 4032 | 6.31 | 6.24 | 4.12 |
| **10377** | 1.09 | Ideal | J | VS2 | 43.0 | 54.0 | 4778 | 6.53 | 6.55 | 4.12 |
| **24932** | 2.01 | Fair | F | SI1 | 58.6 | 95.0 | 13387 | 8.32 | 8.31 | 4.87 |

## checking outliers in z column

In [51]:

```
df_out = df.loc[df["z"] < 2].copy()
outliers(df_out, "z")
```

Out[51]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **14635** | 1.07 | Ideal | F | SI1 | 60.6 | 57.0 | 5909 | 6.62 | 6.67 | 1.07 |
| **20694** | 1.53 | Ideal | I | SI1 | 61.9 | 54.0 | 8971 | 7.43 | 7.50 | 1.53 |
| **21654** | 1.41 | Ideal | H | VS1 | 60.7 | 56.0 | 9752 | 7.31 | 7.22 | 1.41 |

In [52]:

```
df.loc[df["carat"] == df["z"], ["carat", "z"]]
```

Out[52]:

| | carat | z |
|---|---|---|
| **14635** | 1.07 | 1.07 |
| **20694** | 1.53 | 1.53 |
| **21654** | 1.41 | 1.41 |

## replacing outliers in z with np.nan

In [53]:

```
df.loc[df["z"] < 2, "z"] = np.nan
```

## filling null value in z

In [54]:

```
filling_null_values('z')
```

carat: 1.53 / median z value: 4.56
carat: 1.41 / median z value: 4.44
carat: 1.07 / median z value: 4.05

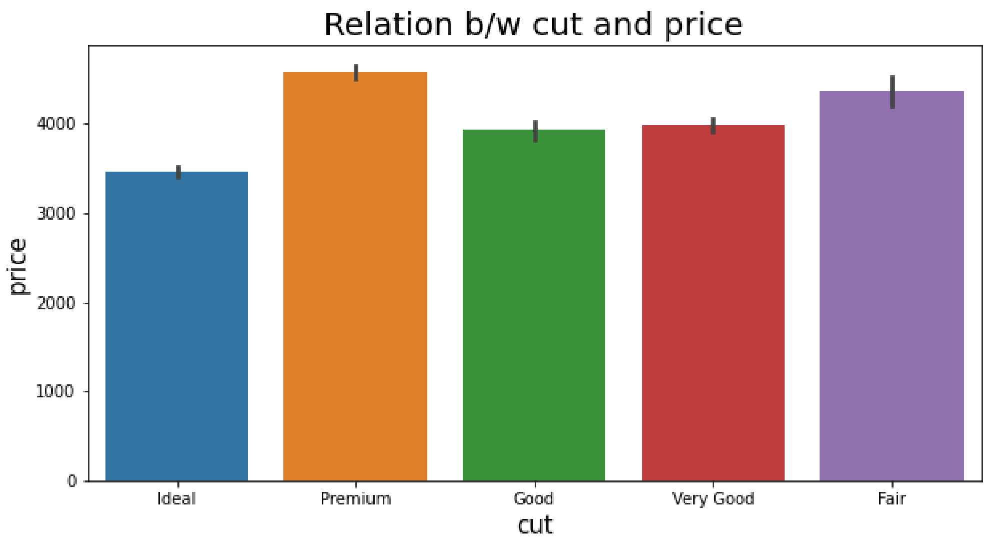
Out[54]:

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| **20694** | 1.53 | Ideal | I | SI1 | 61.9 | 54.0 | 8971 | 7.43 | 7.50 | 4.56 |
| **21654** | 1.41 | Ideal | H | VS1 | 60.7 | 56.0 | 9752 | 7.31 | 7.22 | 4.44 |
| **14635** | 1.07 | Ideal | F | SI1 | 60.6 | 57.0 | 5909 | 6.62 | 6.67 | 4.05 |

# 11. Barplot between cut and price

In [55]:

```
plt.figure(figsize=(10, 5))
sns.barplot(x='cut', y='price', data=df)
plt.title('Relation b/w cut and price', fontsize=20);
plt.xlabel('cut', fontsize=15)
plt.ylabel('price', fontsize=15);
```




# 12. checking correlation of price column with other columns

In [56]:

```
df.corr()['price'].sort_values(ascending=False)[1:]
```

Out[56]:

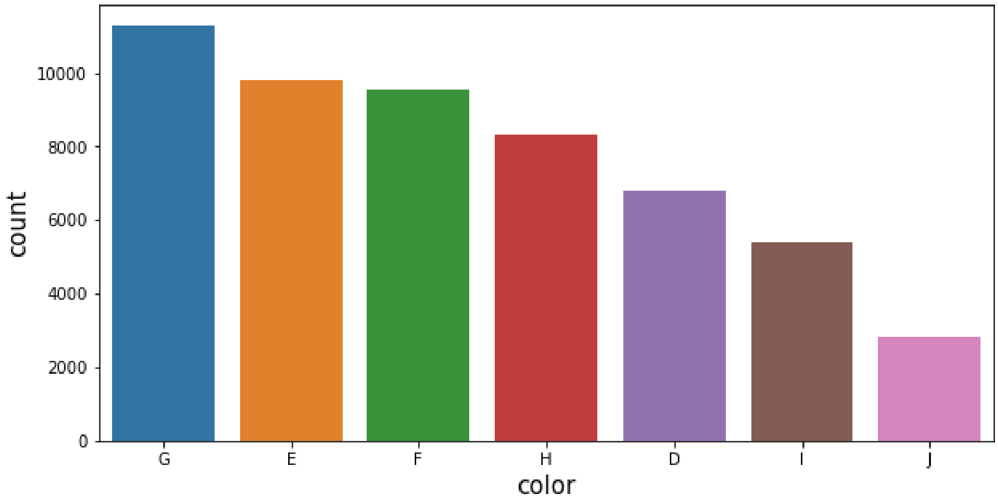
```
carat    0.921591
y        0.888800
x        0.887206
z        0.882368
table    0.127134
depth   -0.010647
Name: price, dtype: float64
```

# color category

In [57]:

```
color_label = df.color.value_counts()
plt.figure(figsize=(10, 5))
sns.barplot(color_label.index, color_label);
plt.ylabel('count', fontsize=15)
plt.xlabel('color', fontsize=15);
```

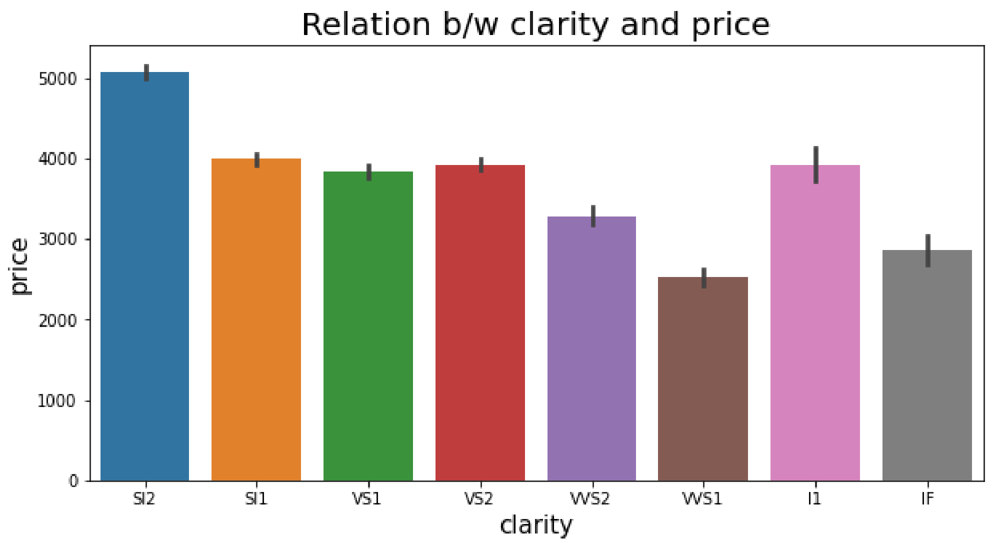
C:\Users\Lucky Computers\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(




# barplot b/w clarity and price

In [58]:

```
plt.figure(figsize=(10, 5))
sns.barplot(x='clarity', y='price', data=df);
plt.title('Relation b/w clarity and price', fontsize=20)
plt.xlabel('clarity', fontsize=15)
plt.ylabel('price', fontsize=15);
```




# 13. Preprocessing of data

## independent and dependent variabels

In [59]:

```
X = df[['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z']]
y = df[['price']]
```

In [60]:

```
X.head()
```

Out[60]:

| | carat | cut | color | clarity | depth | table | x | y | z |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.23 | Ideal | E | SI2 | 61.5 | 55.0 | 3.95 | 3.98 | 2.43 |
| **1** | 0.21 | Premium | E | SI1 | 59.8 | 61.0 | 3.89 | 3.84 | 2.31 |
| **2** | 0.23 | Good | E | VS1 | 56.9 | 65.0 | 4.05 | 4.07 | 2.31 |
| **3** | 0.29 | Premium | I | VS2 | 62.4 | 58.0 | 4.20 | 4.23 | 2.63 |
| **4** | 0.31 | Good | J | SI2 | 63.3 | 58.0 | 4.34 | 4.35 | 2.75 |

In [61]:

```
y.head()
```

Out[61]:

| | price |
|---|---|
| **0** | 326 |
| **1** | 326 |
| **2** | 327 |
| **3** | 334 |
| **4** | 335 |

# 14. "one-hot encoding"

In [62]:

```
X_copy = X.copy()

X_copy = pd.get_dummies(data=X_copy, columns=['clarity', 'color', 'cut'], prefix=['clarity', 'color', 'cut'],drop_first=True).copy()
```

In [63]:

```
X_copy.head()
```

Out[63]:

| | carat | depth | table | x | y | z | clarity_IF | clarity_SI1 | clarity_SI2 | clarity_VS1 | ... | color_E | color_F | color_G | color_H | color_I | color_J | cut_Good | cu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.23 | 61.5 | 55.0 | 3.95 | 3.98 | 2.43 | 0 | 0 | 1 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **1** | 0.21 | 59.8 | 61.0 | 3.89 | 3.84 | 2.31 | 0 | 1 | 0 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **2** | 0.23 | 56.9 | 65.0 | 4.05 | 4.07 | 2.31 | 0 | 0 | 0 | 1 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| **3** | 0.29 | 62.4 | 58.0 | 4.20 | 4.23 | 2.63 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| **4** | 0.31 | 63.3 | 58.0 | 4.34 | 4.35 | 2.75 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 1 | |

5 rows × 23 columns

# 15. scaling

In [64]:

```
sc = StandardScaler()

X_copy = sc.fit_transform(X_copy)
```

# 16. k-fold correlation

In [65]:

```
cv = KFold(n_splits=10, random_state=0, shuffle = True)
```

In [66]:

```
matrix = ["r2", "neg_mean_absolute_error", "neg_mean_squared_error"]
scores = {"train" : [], "test" : [], "mae" : [], "mse" : [], "rmse" : []}
```

In [67]:

```
def result(model, f):
    sc = cross_validate(model, f, y, cv=cv, scoring=matrix, return_train_score=True)

    train_s = sc["train_r2"].mean()
```

```
    scores["train"].append(train_s)

    test_s = sc["test_r2"].mean()
    scores["test"].append(test_s)

    ''' mae '''
    mae = np.absolute(sc["test_neg_mean_absolute_error"]).mean()
    scores["mae"].append(mae)

    ''' scorese '''
    mse = np.absolute(sc["test_neg_mean_squared_error"]).mean()
    scores["mse"].append(mse)

    ''' rscorese '''
    rmse = np.sqrt(mse)
    scores["rmse"].append(rmse)
    print("train score: {0:.4f}\nR2 score: {1:.4f}\nMAE: {2:.2f}\nscoresE: {3:.2f}\nRscoresE: {4:.2f}".format(train_s, test_s,
                                                                            mae, mse, rmse))
```

# 16. linear regression

In [68]:

```
linear_reg = LinearRegression()
result(linear_reg, X_copy)
```

```
train score: 0.9207
R2 score: 0.9205
MAE: 733.52
scoresE: 1264478.94
RscoresE: 1124.49
```

# 17.XGBRegressor

In [75]:

```
xgb = XGBRegressor(learning_rate = 0.1, n_estimators = 200, random_state = 0)
result(xgb, X_copy)
```

```
train score: 0.9850
R2 score: 0.9783
MAE: 301.19
scoresE: 345675.40
RscoresE: 587.94
```

# 18. RandomForestRegressor

In [76]:

```
rf = RandomForestRegressor(max_depth = 8, n_estimators = 40, random_state = 0)
result(rf, X_copy)
```

```
train score: 0.9508
R2 score: 0.9476
MAE: 473.02
scoresE: 834965.60
RscoresE: 913.76
```

# 19. DecisionTreeRegressor

In [78]:

```
dt = DecisionTreeRegressor(max_depth = 8, random_state = 0)
result(dt, X_copy)
```

```
train score: 0.9444
R2 score: 0.9404
MAE: 506.39
scoresE: 949876.92
RscoresE: 974.62
```

In [ ]: