project-maha

May 25, 2024

1 Car Price Predication

[3]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
df = pd.read_csv('C:\\Users\\djerb\\Downloads\\Car Price Prediction\\Car Price␣
↪Prediction\\Car Price Prediction.csv')
```

2 1. Explore the data

[5]:
```
df.shape
```

[5]:
```
(205, 26)
```

[6]:
```
df.head()
```

[6]:
```
   car_ID  symboling                   CarName fueltype aspiration doornumber  \
0       1          3        alfa-romero giulia      gas        std        two
1       2          3       alfa-romero stelvio      gas        std        two
2       3          1  alfa-romero Quadrifoglio      gas        std        two
3       4          2               audi 100 ls      gas        std       four
4       5          2                audi 100ls      gas        std       four

       carbody drivewheel enginelocation  wheelbase  …  enginesize  \
0  convertible        rwd          front       88.6  …         130
1  convertible        rwd          front       88.6  …         130
2    hatchback        rwd          front       94.5  …         152
3        sedan        fwd          front       99.8  …         109
4        sedan        4wd          front       99.4  …         136

   fuelsystem  boreratio  stroke compressionratio horsepower  peakrpm citympg  \
0        mpfi       3.47    2.68              9.0        111     5000      21
1        mpfi       3.47    2.68              9.0        111     5000      21
```

```
2        mpfi       2.68    3.47              9.0         154       5000        19
3        mpfi       3.19    3.40             10.0         102       5500        24
4        mpfi       3.19    3.40              8.0         115       5500        18

   highwaympg    price
0          27  13495.0
1          27  16500.0
2          26  16500.0
3          30  13950.0
4          22  17450.0

[5 rows x 26 columns]
```

[7]:
```
df.columns
```

[7]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
```

```
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

[9]:
```
df['CarName'].unique()
```

[9]:
```
array(['alfa-romero giulia', 'alfa-romero stelvio',
       'alfa-romero Quadrifoglio', 'audi 100 ls', 'audi 100ls',
       'audi fox', 'audi 5000', 'audi 4000', 'audi 5000s (diesel)',
       'bmw 320i', 'bmw x1', 'bmw x3', 'bmw z4', 'bmw x4', 'bmw x5',
       'chevrolet impala', 'chevrolet monte carlo', 'chevrolet vega 2300',
       'dodge rampage', 'dodge challenger se', 'dodge d200',
       'dodge monaco (sw)', 'dodge colt hardtop', 'dodge colt (sw)',
       'dodge coronet custom', 'dodge dart custom',
       'dodge coronet custom (sw)', 'honda civic', 'honda civic cvcc',
       'honda accord cvcc', 'honda accord lx', 'honda civic 1500 gl',
       'honda accord', 'honda civic 1300', 'honda prelude',
       'honda civic (auto)', 'isuzu MU-X', 'isuzu D-Max ',
       'isuzu D-Max V-Cross', 'jaguar xj', 'jaguar xf', 'jaguar xk',
       'maxda rx3', 'maxda glc deluxe', 'mazda rx2 coupe', 'mazda rx-4',
       'mazda glc deluxe', 'mazda 626', 'mazda glc', 'mazda rx-7 gs',
       'mazda glc 4', 'mazda glc custom l', 'mazda glc custom',
       'buick electra 225 custom', 'buick century luxus (sw)',
       'buick century', 'buick skyhawk', 'buick opel isuzu deluxe',
       'buick skylark', 'buick century special',
       'buick regal sport coupe (turbo)', 'mercury cougar',
       'mitsubishi mirage', 'mitsubishi lancer', 'mitsubishi outlander',
       'mitsubishi g4', 'mitsubishi mirage g4', 'mitsubishi montero',
       'mitsubishi pajero', 'Nissan versa', 'nissan gt-r', 'nissan rogue',
       'nissan latio', 'nissan titan', 'nissan leaf', 'nissan juke',
       'nissan note', 'nissan clipper', 'nissan nv200', 'nissan dayz',
       'nissan fuga', 'nissan otti', 'nissan teana', 'nissan kicks',
       'peugeot 504', 'peugeot 304', 'peugeot 504 (sw)', 'peugeot 604sl',
       'peugeot 505s turbo diesel', 'plymouth fury iii',
       'plymouth cricket', 'plymouth satellite custom (sw)',
       'plymouth fury gran sedan', 'plymouth valiant', 'plymouth duster',
       'porsche macan', 'porcshce panamera', 'porsche cayenne',
       'porsche boxter', 'renault 12tl', 'renault 5 gtl', 'saab 99e',
       'saab 99le', 'saab 99gle', 'subaru', 'subaru dl', 'subaru brz',
```

```
       'subaru baja', 'subaru r1', 'subaru r2', 'subaru trezia',
       'subaru tribeca', 'toyota corona mark ii', 'toyota corona',
       'toyota corolla 1200', 'toyota corona hardtop',
       'toyota corolla 1600 (sw)', 'toyota carina', 'toyota mark ii',
       'toyota corolla', 'toyota corolla liftback',
       'toyota celica gt liftback', 'toyota corolla tercel',
       'toyota corona liftback', 'toyota starlet', 'toyota tercel',
       'toyota cressida', 'toyota celica gt', 'toyouta tercel',
       'vokswagen rabbit', 'volkswagen 1131 deluxe sedan',
       'volkswagen model 111', 'volkswagen type 3', 'volkswagen 411 (sw)',
       'volkswagen super beetle', 'volkswagen dasher', 'vw dasher',
       'vw rabbit', 'volkswagen rabbit', 'volkswagen rabbit custom',
       'volvo 145e (sw)', 'volvo 144ea', 'volvo 244dl', 'volvo 245',
       'volvo 264gl', 'volvo diesel', 'volvo 246'], dtype=object)
```

[10]:
```
df['fueltype'].unique()
```

[10]:
```
array(['gas', 'diesel'], dtype=object)
```

[11]:
```
df['aspiration'].unique()
```

[11]:
```
array(['std', 'turbo'], dtype=object)
```

[12]:
```
df['doornumber'].unique()
```

[12]:
```
array(['two', 'four'], dtype=object)
```

[13]:
```
#replace the text values with numbers
df['doornumber'].replace('two',2,inplace=True)
df['doornumber'].replace('four',4,inplace=True)
df['doornumber'].unique()
```

[13]:
```
array([2, 4], dtype=int64)
```

[14]:
```
df['carbody'].unique()
df['drivewheel'].unique()
df['enginelocation'].unique()
df['wheelbase'].unique()
df['carlength'].unique()
df['carwidth'].unique()
df['carheight'].unique()
```

[14]:
```
array([48.8, 52.4, 54.3, 53.1, 55.7, 55.9, 52. , 53.7, 56.3, 53.2, 50.8,
       50.6, 59.8, 50.2, 52.6, 54.5, 58.3, 53.3, 54.1, 51. , 53.5, 51.4,
       52.8, 47.8, 49.6, 55.5, 54.4, 56.5, 58.7, 54.9, 56.7, 55.4, 54.8,
       49.4, 51.6, 54.7, 55.1, 56.1, 49.7, 56. , 50.5, 55.2, 52.5, 53. ,
       59.1, 53.9, 55.6, 56.2, 57.5])
```

[15]:
```
def unique(x):
    return df[x].unique()
```

[16]:
```
unique('curbweight')
unique('cylindernumber')
```

[16]:
```
array(['four', 'six', 'five', 'three', 'twelve', 'two', 'eight'],
      dtype=object)
```

[17]:
```
df['cylindernumber'].replace('four',4,inplace=True)
df['cylindernumber'].replace('six',6,inplace=True)
df['cylindernumber'].replace('five',5,inplace=True)
df['cylindernumber'].replace('three',3,inplace=True)
df['cylindernumber'].replace('twelve',12,inplace=True)
df['cylindernumber'].replace('two',2,inplace=True)
df['cylindernumber'].replace('eight',8,inplace=True)
df['cylindernumber'].unique()
```

[17]:
```
array([ 4,  6,  5,  3, 12,  2,  8], dtype=int64)
```

[18]:
```
unique('enginesize')
unique('fuelsystem')
unique('boreratio')
unique('stroke')
unique('compressionratio')
unique('horsepower')
unique('peakrpm')
unique('citympg')
unique('highwaympg')
unique('price')
```

[18]:
```
array([13495.   , 16500.   , 13950.   , 17450.   , 15250.   , 17710.   ,
       18920.   , 23875.   , 17859.167, 16430.   , 16925.   , 20970.   ,
       21105.   , 24565.   , 30760.   , 41315.   , 36880.   ,  5151.   ,
        6295.   ,  6575.   ,  5572.   ,  6377.   ,  7957.   ,  6229.   ,
        6692.   ,  7609.   ,  8558.   ,  8921.   , 12964.   ,  6479.   ,
        6855.   ,  5399.   ,  6529.   ,  7129.   ,  7295.   ,  7895.   ,
        9095.   ,  8845.   , 10295.   , 12945.   , 10345.   ,  6785.   ,
        8916.5  , 11048.   , 32250.   , 35550.   , 36000.   ,  5195.   ,
        6095.   ,  6795.   ,  6695.   ,  7395.   , 10945.   , 11845.   ,
       13645.   , 15645.   ,  8495.   , 10595.   , 10245.   , 10795.   ,
       11245.   , 18280.   , 18344.   , 25552.   , 28248.   , 28176.   ,
       31600.   , 34184.   , 35056.   , 40960.   , 45400.   , 16503.   ,
        5389.   ,  6189.   ,  6669.   ,  7689.   ,  9959.   ,  8499.   ,
       12629.   , 14869.   , 14489.   ,  6989.   ,  8189.   ,  9279.   ,
        5499.   ,  7099.   ,  6649.   ,  6849.   ,  7349.   ,  7299.   ,
        7799.   ,  7499.   ,  7999.   ,  8249.   ,  8949.   ,  9549.   ,
```

```
       13499.    , 14399.    , 17199.    , 19699.    , 18399.    , 11900.    ,
       13200.    , 12440.    , 13860.    , 15580.    , 16900.    , 16695.    ,
       17075.    , 16630.    , 17950.    , 18150.    , 12764.    , 22018.    ,
       32528.    , 34028.    , 37028.    , 31400.5   ,  9295.    ,  9895.    ,
       11850.    , 12170.    , 15040.    , 15510.    , 18620.    ,  5118.    ,
        7053.    ,  7603.    ,  7126.    ,  7775.    ,  9960.    ,  9233.    ,
       11259.    ,  7463.    , 10198.    ,  8013.    , 11694.    ,  5348.    ,
        6338.    ,  6488.    ,  6918.    ,  7898.    ,  8778.    ,  6938.    ,
        7198.    ,  7788.    ,  7738.    ,  8358.    ,  9258.    ,  8058.    ,
        8238.    ,  9298.    ,  9538.    ,  8449.    ,  9639.    ,  9989.    ,
       11199.    , 11549.    , 17669.    ,  8948.    , 10698.    ,  9988.    ,
       10898.    , 11248.    , 16558.    , 15998.    , 15690.    , 15750.    ,
        7975.    ,  7995.    ,  8195.    ,  9495.    ,  9995.    , 11595.    ,
        9980.    , 13295.    , 13845.    , 12290.    , 12940.    , 13415.    ,
       15985.    , 16515.    , 18420.    , 18950.    , 16845.    , 19045.    ,
       21485.    , 22470.    , 22625.    ])
```

3 2. Data Preprocessing.

[19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    int64
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    int64
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
```

```
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(10), object(8)
memory usage: 41.8+ KB
```

[20]:
```
# check for null values
df.isna().sum()
```

[20]:
```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

[21]:
```
# check for duplicates
df.duplicated().sum()
```

[21]: 0

[22]:
```
df.columns
```

[22]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

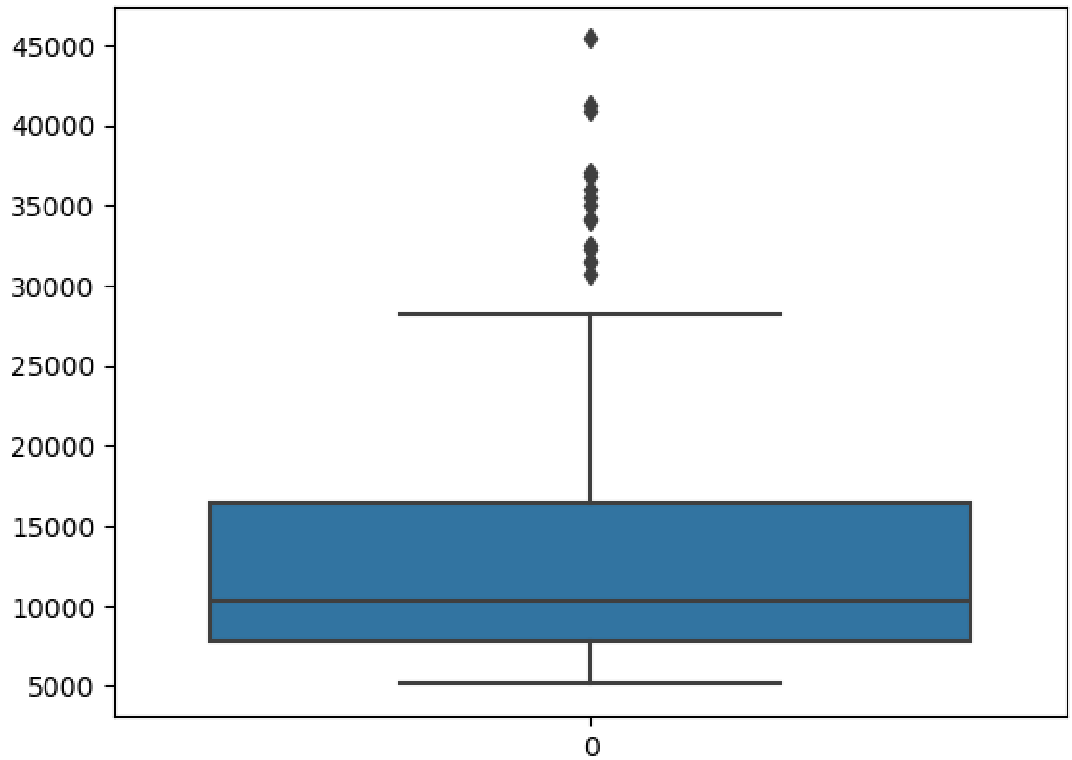
[23]:
```
# statistical analysis
df.describe()
```

[23]:
```
           car_ID   symboling  doornumber   wheelbase   carlength    carwidth  \
count  205.000000  205.000000  205.000000  205.000000  205.000000  205.000000
mean   103.000000    0.834146    3.121951   98.756585  174.049268   65.907805
std     59.322565    1.245307    0.994966    6.021776   12.337289    2.145204
min      1.000000   -2.000000    2.000000   86.600000  141.100000   60.300000
25%     52.000000    0.000000    2.000000   94.500000  166.300000   64.100000
50%    103.000000    1.000000    4.000000   97.000000  173.200000   65.500000
75%    154.000000    2.000000    4.000000  102.400000  183.100000   66.900000
max    205.000000    3.000000    4.000000  120.900000  208.100000   72.300000

        carheight   curbweight  cylindernumber  enginesize   boreratio  \
count  205.000000   205.000000      205.000000  205.000000  205.000000
mean    53.724878  2555.565854        4.380488  126.907317    3.329756
std      2.443522   520.680204        1.080854   41.642693    0.270844
min     47.800000  1488.000000        2.000000   61.000000    2.540000
25%     52.000000  2145.000000        4.000000   97.000000    3.150000
50%     54.100000  2414.000000        4.000000  120.000000    3.310000
75%     55.500000  2935.000000        4.000000  141.000000    3.580000
max     59.800000  4066.000000       12.000000  326.000000    3.940000

           stroke  compressionratio  horsepower      peakrpm     citympg  \
count  205.000000        205.000000  205.000000   205.000000  205.000000
mean     3.255415         10.142537  104.117073  5125.121951   25.219512
std      0.313597          3.972040   39.544167   476.985643    6.542142
min      2.070000          7.000000   48.000000  4150.000000   13.000000
25%      3.110000          8.600000   70.000000  4800.000000   19.000000
50%      3.290000          9.000000   95.000000  5200.000000   24.000000
75%      3.410000          9.400000  116.000000  5500.000000   30.000000
max      4.170000         23.000000  288.000000  6600.000000   49.000000

       highwaympg         price
count  205.000000    205.000000
mean    30.751220  13276.710571
std      6.886443   7988.852332
min     16.000000   5118.000000
25%     25.000000   7788.000000
```

```
50%      30.000000   10295.000000
75%      34.000000   16503.000000
max      54.000000   45400.000000
```

[24]:
```
#check for outliers
sns.boxplot(data=df['price'])
# ouliers are there, but not removing it.
```

[24]:
```
<Axes: >
```



[25]:
```
df.columns
df.head()
```

[25]:
```
   car_ID  symboling                   CarName fueltype aspiration  \
0       1          3        alfa-romero giulia      gas        std
1       2          3       alfa-romero stelvio      gas        std
2       3          1  alfa-romero Quadrifoglio      gas        std
3       4          2               audi 100 ls      gas        std
4       5          2                audi 100ls      gas        std

   doornumber      carbody drivewheel enginelocation  wheelbase  …  \
0           2  convertible        rwd          front       88.6  …
```

```
1           2  convertible        rwd         front        88.6  …
2           2    hatchback        rwd         front        94.5  …
3           4        sedan        fwd         front        99.8  …
4           4        sedan        4wd         front        99.4  …

   enginesize  fuelsystem  boreratio  stroke compressionratio  horsepower  \
0         130        mpfi       3.47    2.68              9.0         111
1         130        mpfi       3.47    2.68              9.0         111
2         152        mpfi       2.68    3.47              9.0         154
3         109        mpfi       3.19    3.40             10.0         102
4         136        mpfi       3.19    3.40              8.0         115

   peakrpm citympg  highwaympg    price
0     5000      21          27  13495.0
1     5000      21          27  16500.0
2     5000      19          26  16500.0
3     5500      24          30  13950.0
4     5500      18          22  17450.0

[5 rows x 26 columns]
```

[26]:
```
#Encode the categorical datas from the dataset. For Ml Model generation
```

[27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    int64
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    int64
 16  enginesize        205 non-null    int64
```

```
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(10), object(8)
memory usage: 41.8+ KB
```

[28]:
```python
#Encoding using Label Encoder
from sklearn import preprocessing
label=preprocessing.LabelEncoder()
```

[29]:
```python
label.fit(df.fueltype)
df.fueltype=label.transform(df.fueltype)
```

[30]:
```python
label.fit(df.aspiration)
df.aspiration=label.transform(df.aspiration)
```

[31]:
```python
label.fit(df.carbody)
df.carbody=label.transform(df.carbody)
```

[32]:
```python
label.fit(df.drivewheel)
df.drivewheel=label.transform(df.drivewheel)
```

[33]:
```python
label.fit(df.enginelocation)
df.enginelocation=label.transform(df.enginelocation)
```

[34]:
```python
label.fit(df.enginetype)
df.enginetype=label.transform(df.enginetype)
```

[35]:
```python
label.fit(df.fuelsystem)
df.fuelsystem=label.transform(df.fuelsystem)
```

[36]:
```python
df.head()
```

[36]:
```
   car_ID  symboling                   CarName  fueltype  aspiration  \
0       1          3        alfa-romero giulia         1           0
1       2          3       alfa-romero stelvio         1           0
2       3          1  alfa-romero Quadrifoglio         1           0
3       4          2               audi 100 ls         1           0
4       5          2                audi 100ls         1           0

   doornumber  carbody  drivewheel  enginelocation  wheelbase  …  \
```

```
0          2       0           2               0       88.6  …
1          2       0           2               0       88.6  …
2          2       2           2               0       94.5  …
3          4       3           1               0       99.8  …
4          4       3           0               0       99.4  …

   enginesize  fuelsystem  boreratio  stroke  compressionratio  horsepower  \
0         130           5       3.47    2.68               9.0         111
1         130           5       3.47    2.68               9.0         111
2         152           5       2.68    3.47               9.0         154
3         109           5       3.19    3.40              10.0         102
4         136           5       3.19    3.40               8.0         115

   peakrpm  citympg  highwaympg    price
0     5000       21          27  13495.0
1     5000       21          27  16500.0
2     5000       19          26  16500.0
3     5500       24          30  13950.0
4     5500       18          22  17450.0

[5 rows x 26 columns]
```

[37]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    int32
 4   aspiration        205 non-null    int32
 5   doornumber        205 non-null    int64
 6   carbody           205 non-null    int32
 7   drivewheel        205 non-null    int32
 8   enginelocation    205 non-null    int32
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    int32
 15  cylindernumber    205 non-null    int64
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    int32
```

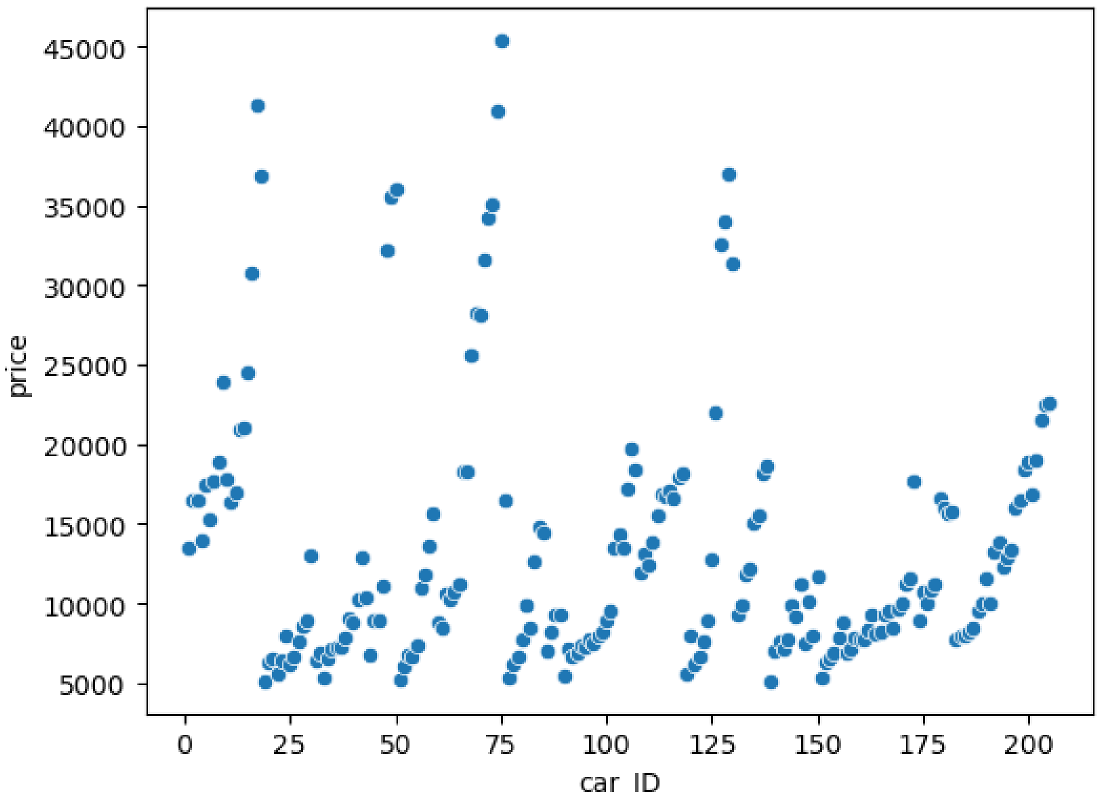
```
 18   boreratio         205 non-null    float64
 19   stroke            205 non-null    float64
 20   compressionratio  205 non-null    float64
 21   horsepower        205 non-null    int64
 22   peakrpm           205 non-null    int64
 23   citympg           205 non-null    int64
 24   highwaympg        205 non-null    int64
 25   price             205 non-null    float64
dtypes: float64(8), int32(7), int64(10), object(1)
memory usage: 36.2+ KB
```

[38]:
```
# check the correlation between 'price' and independent variables
```

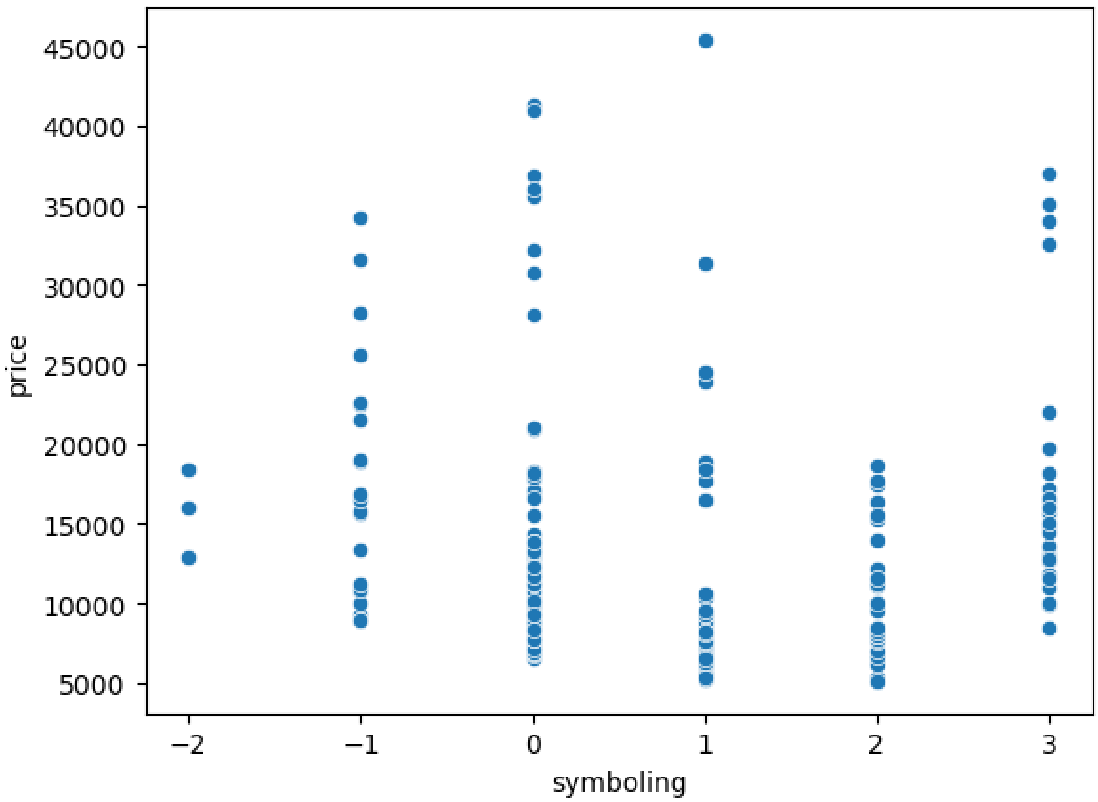
[39]:
```
df.columns
```

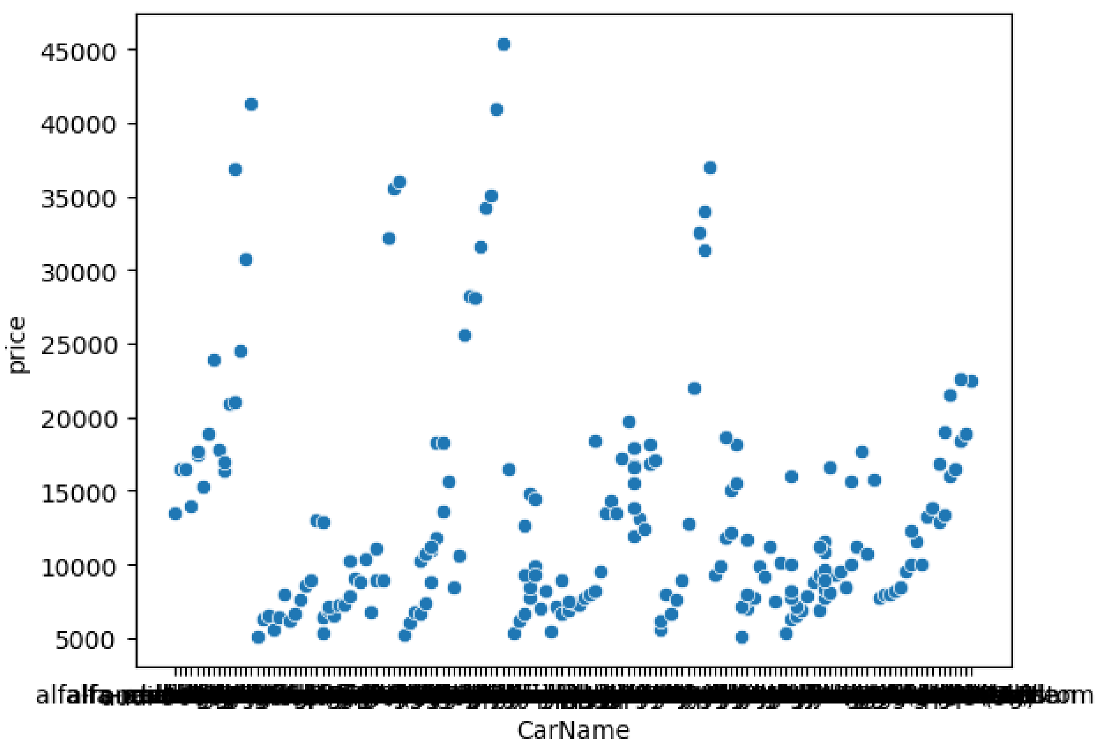
[39]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

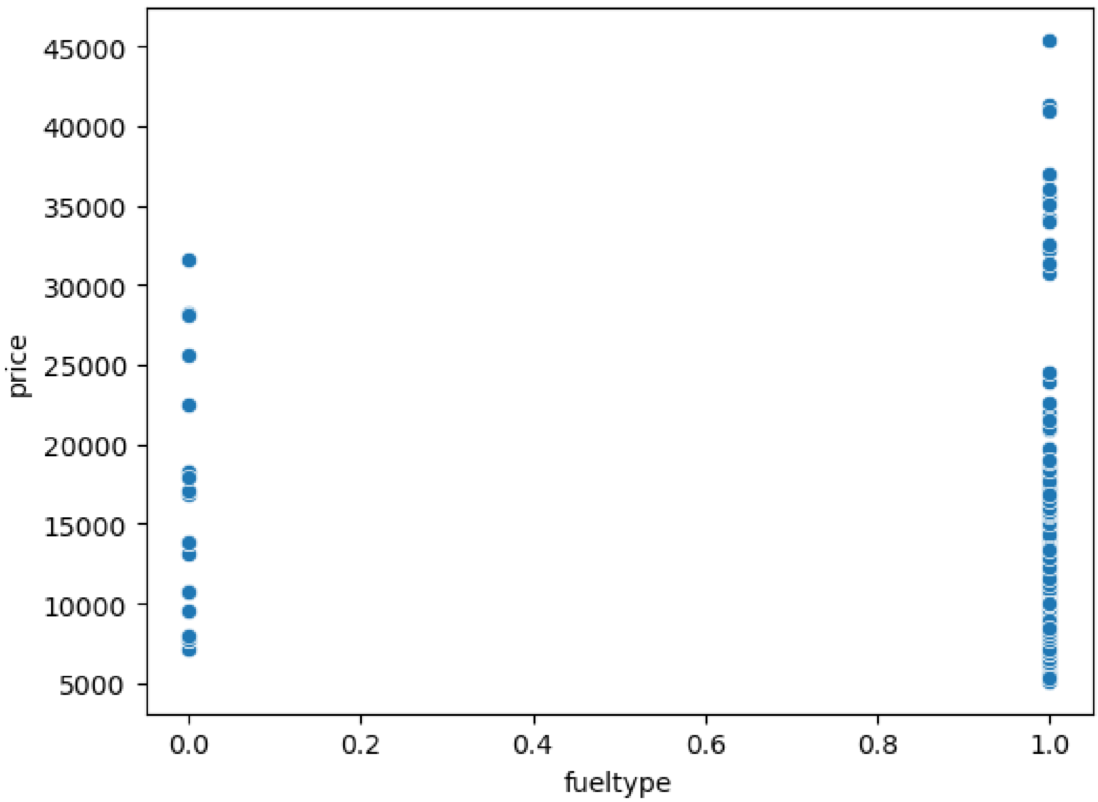
[40]:
```
# Creating scatter plots for each numerical column against 'price'
for col in df:
    sns.scatterplot(data=df, x=col, y='price')
    plt.show()
```

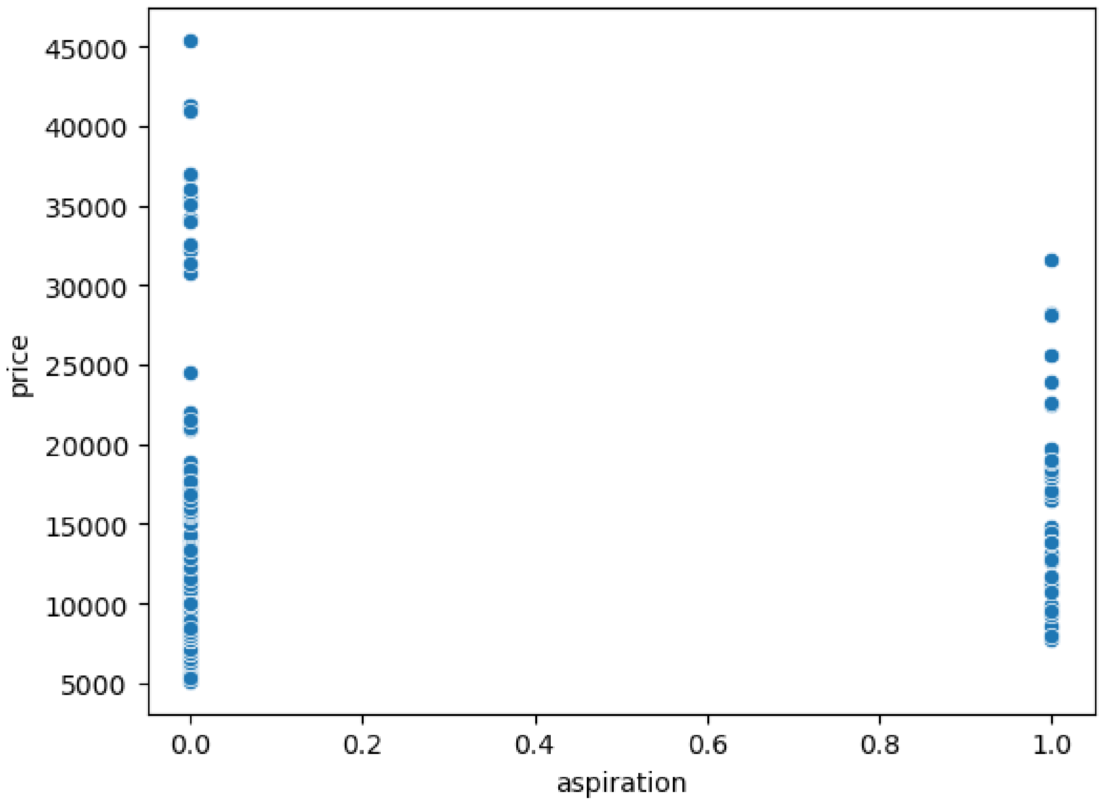


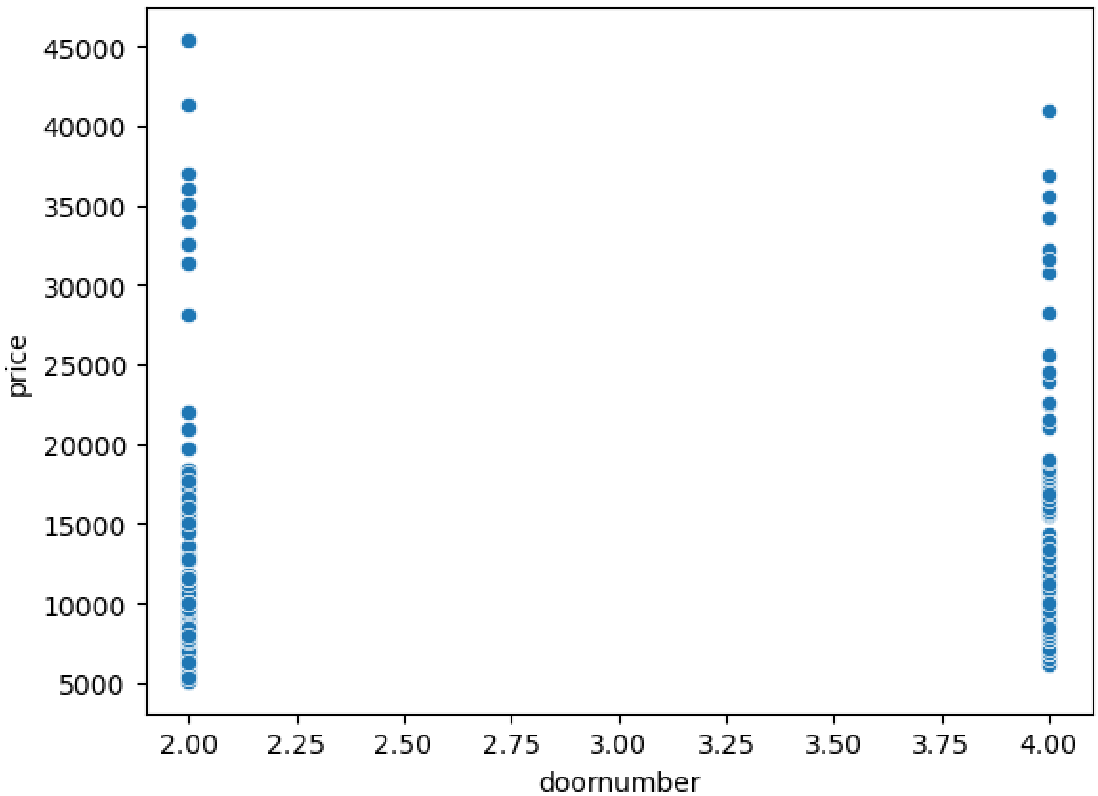
45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0
25
50
75
100
125
150
175
200
car_ID


45000
40000
35000
30000
25000
20000
15000
10000
5000
price
−2
−1
0
1
2
3
symboling

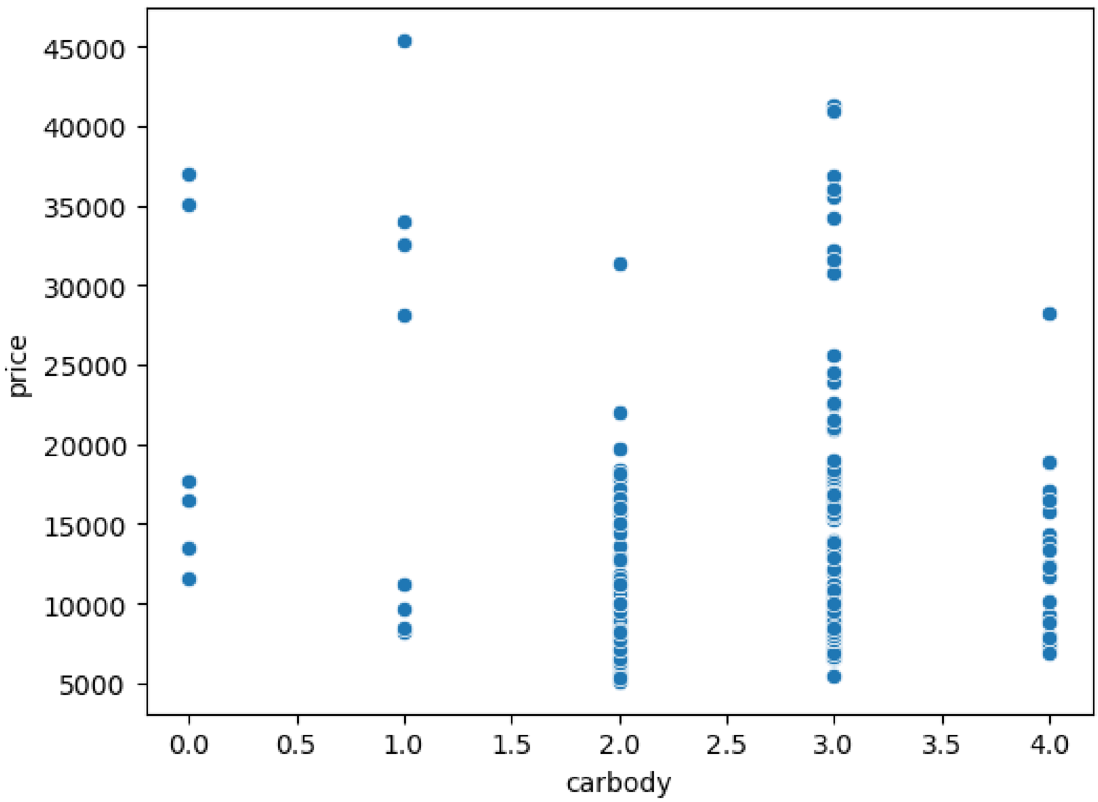

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
CarName


45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0.0
0.2
0.4
0.6
0.8
1.0
fueltype

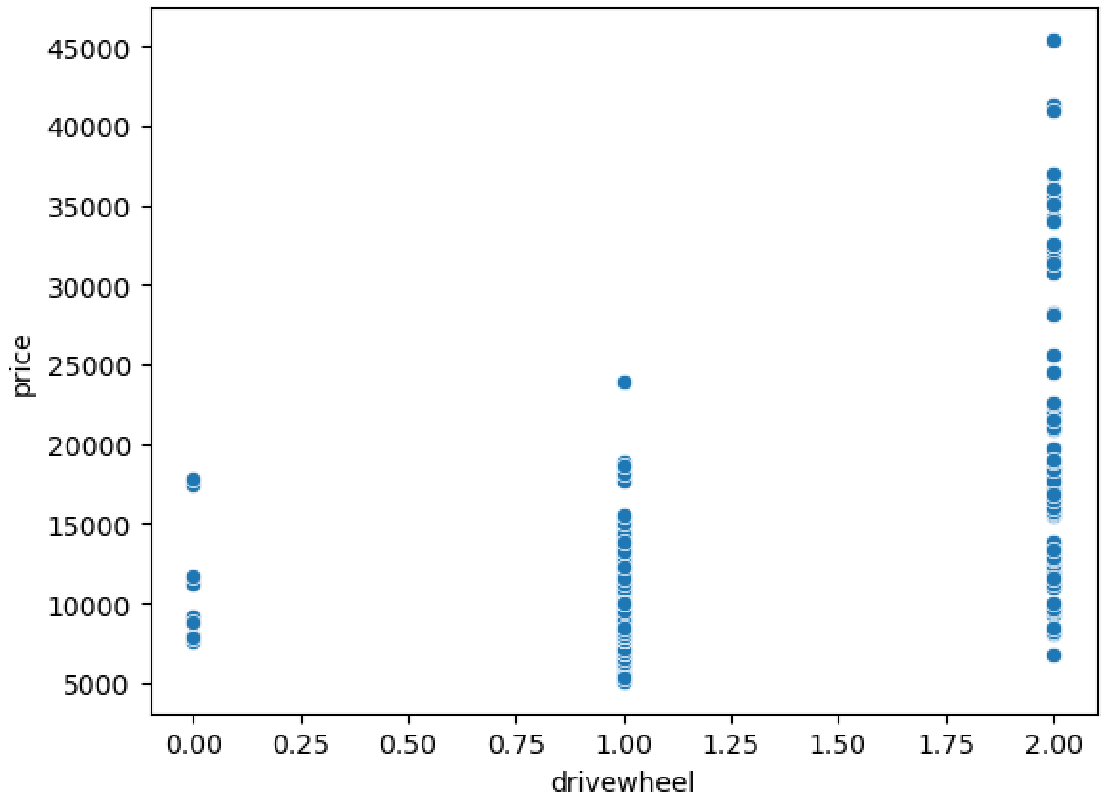

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0.0
0.2
0.4
0.6
0.8
1.0
aspiration

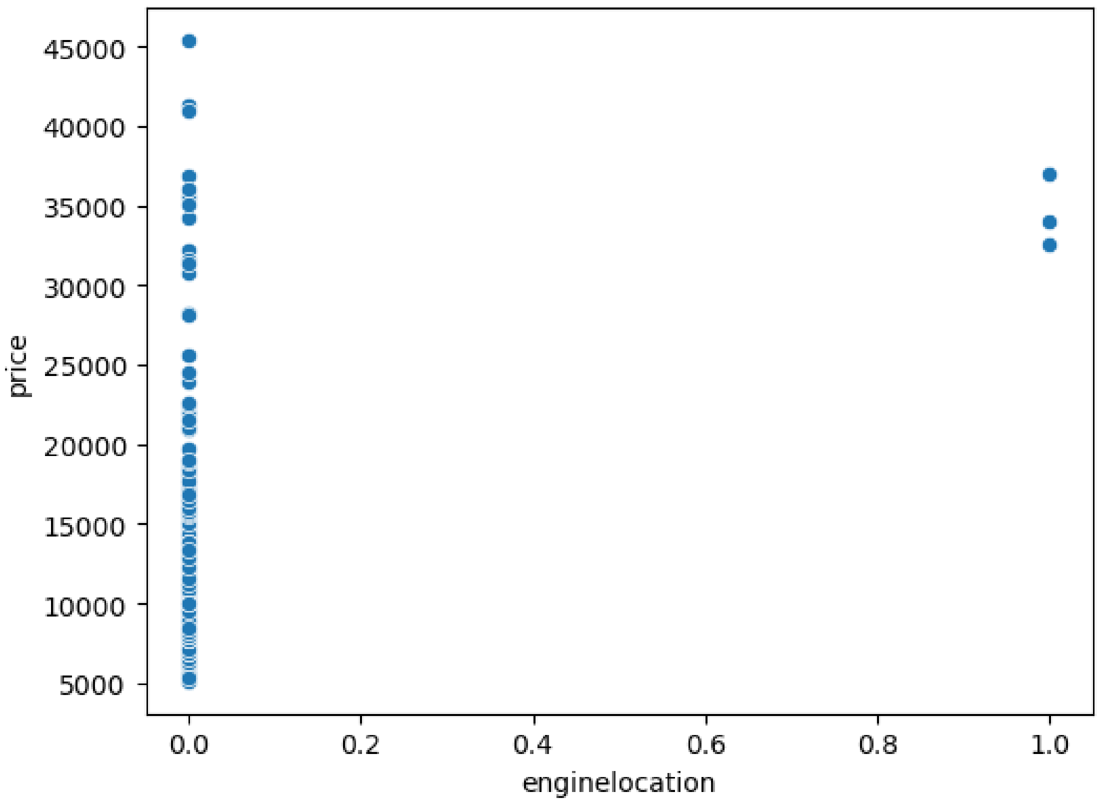


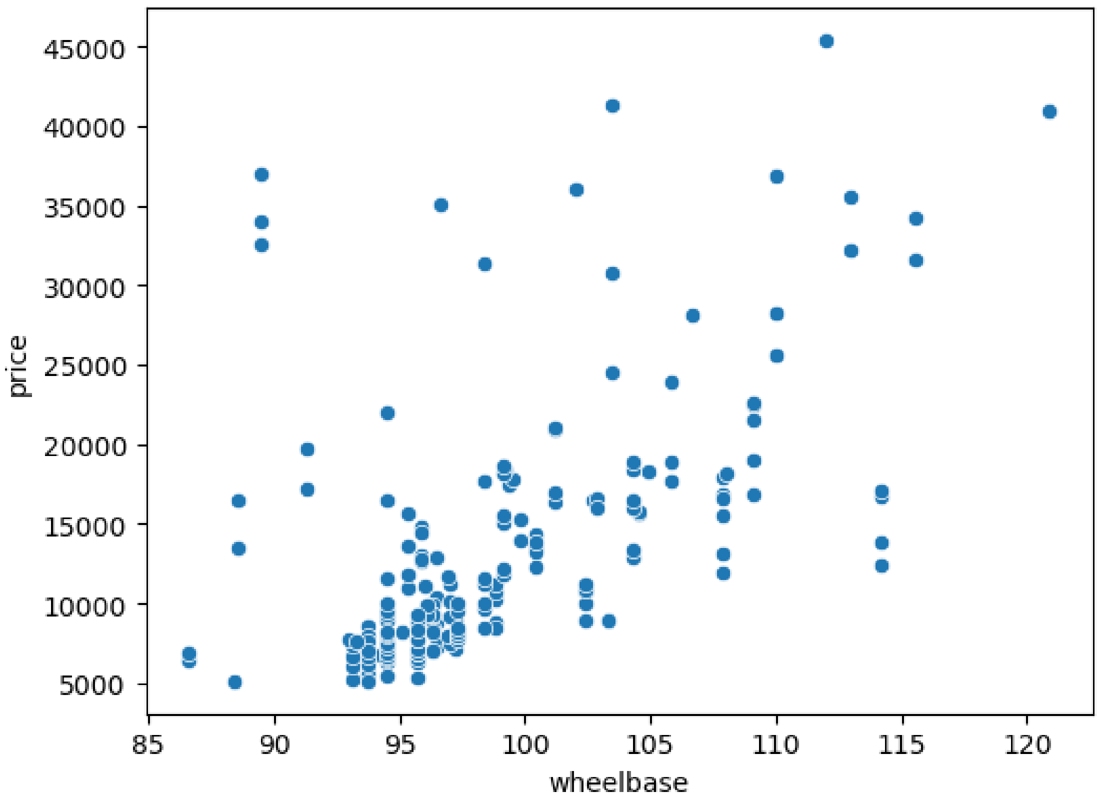
45000
40000
35000
30000
25000
20000
15000
10000
5000
price
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
doornumber

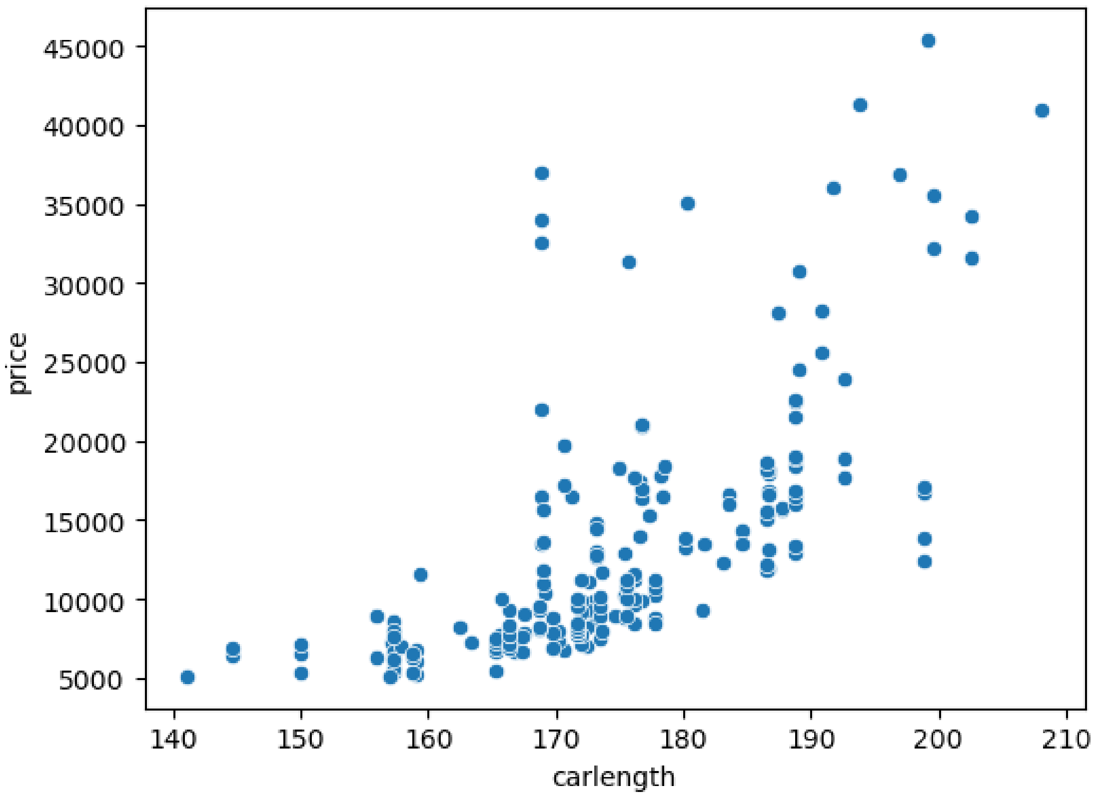


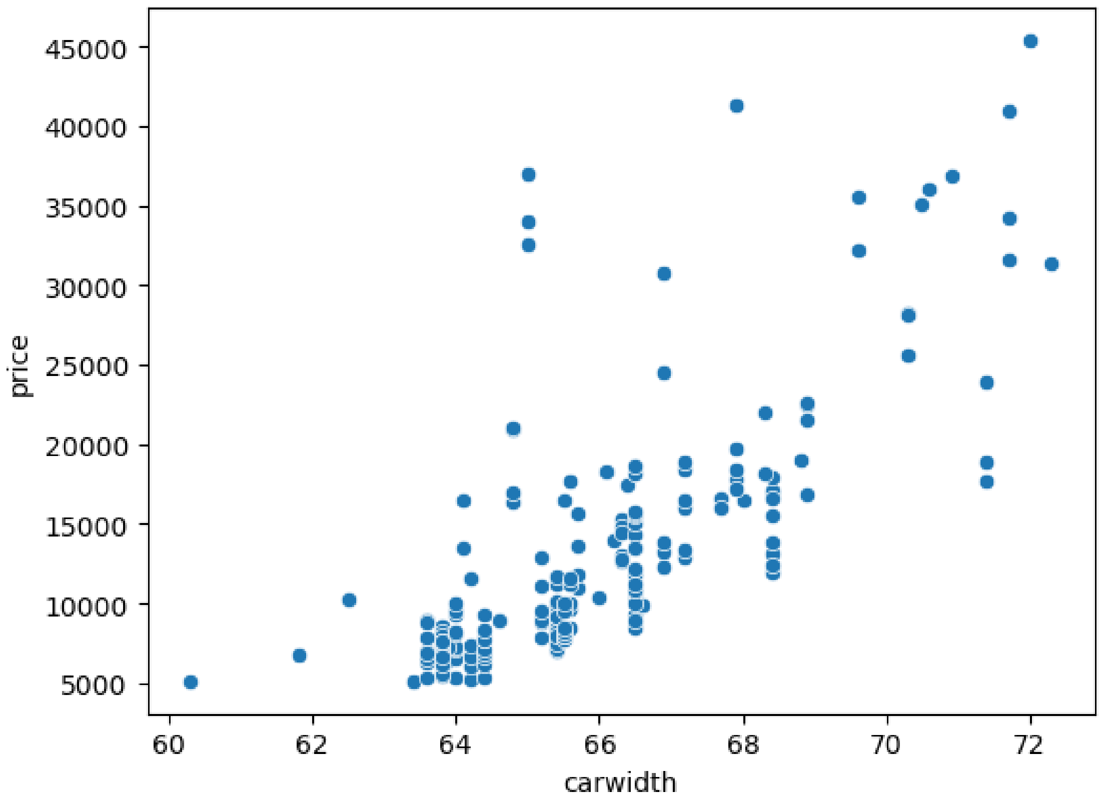
45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
carbody

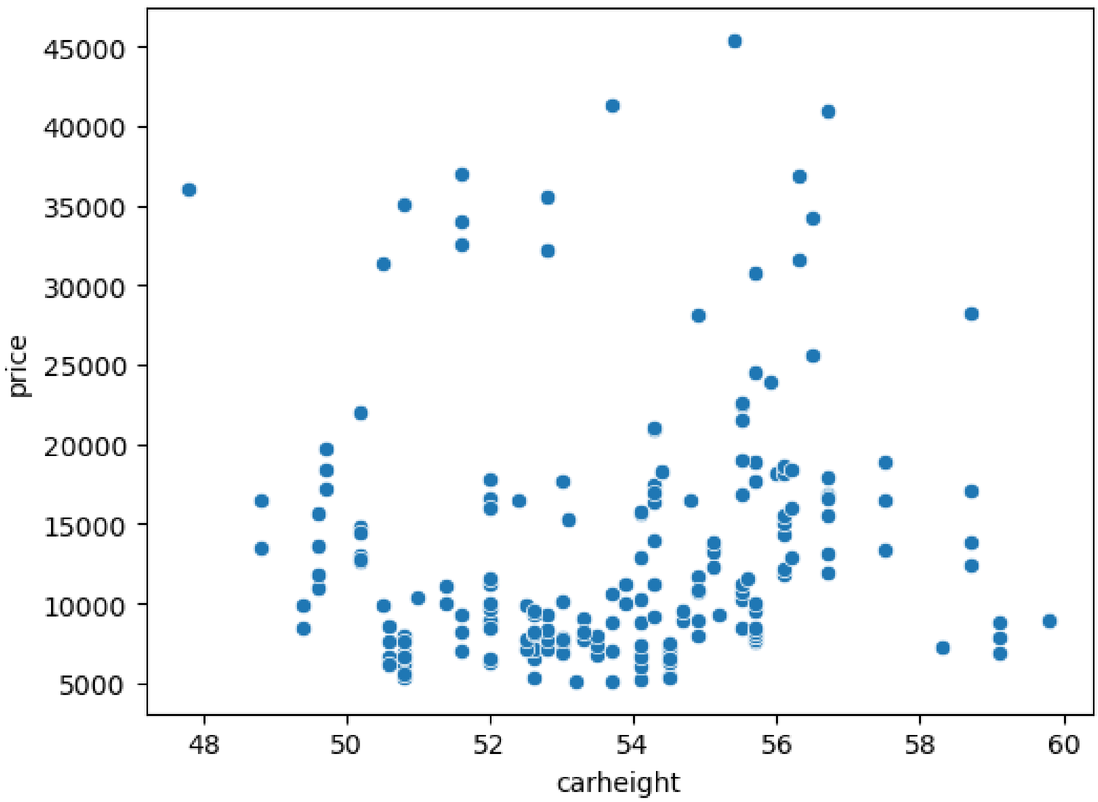

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
drivewheel

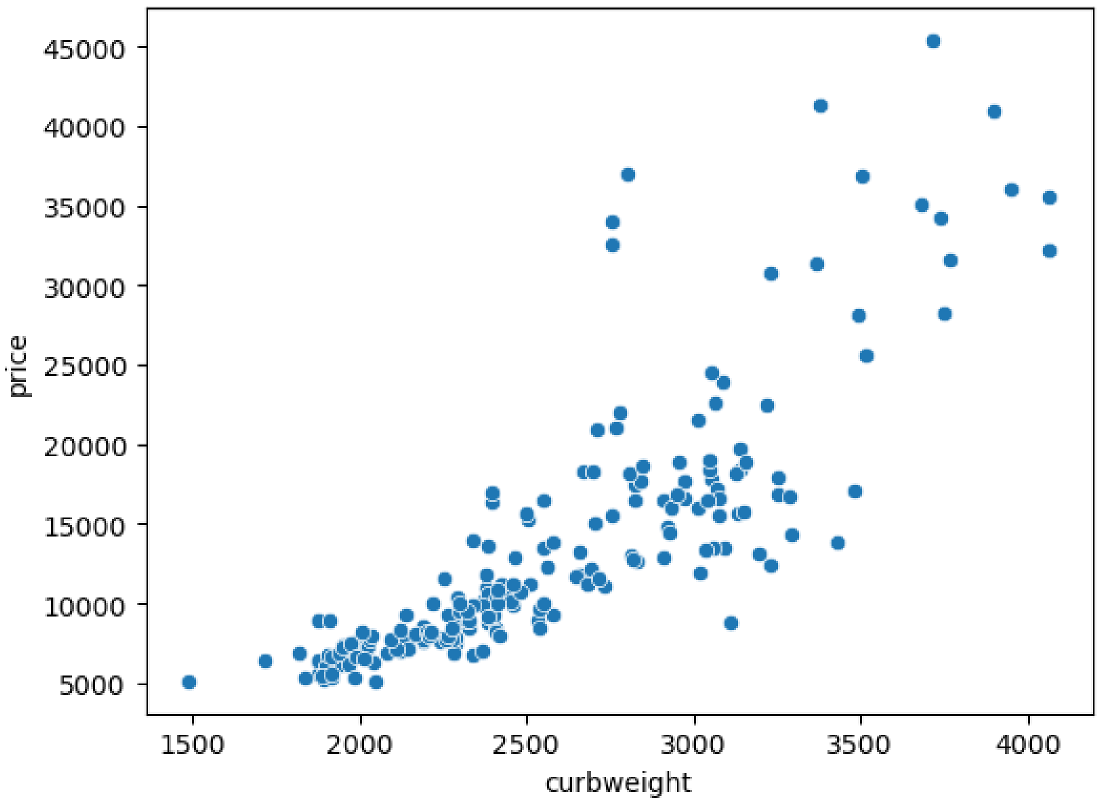

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0.0
0.2
0.4
0.6
0.8
1.0
enginelocation

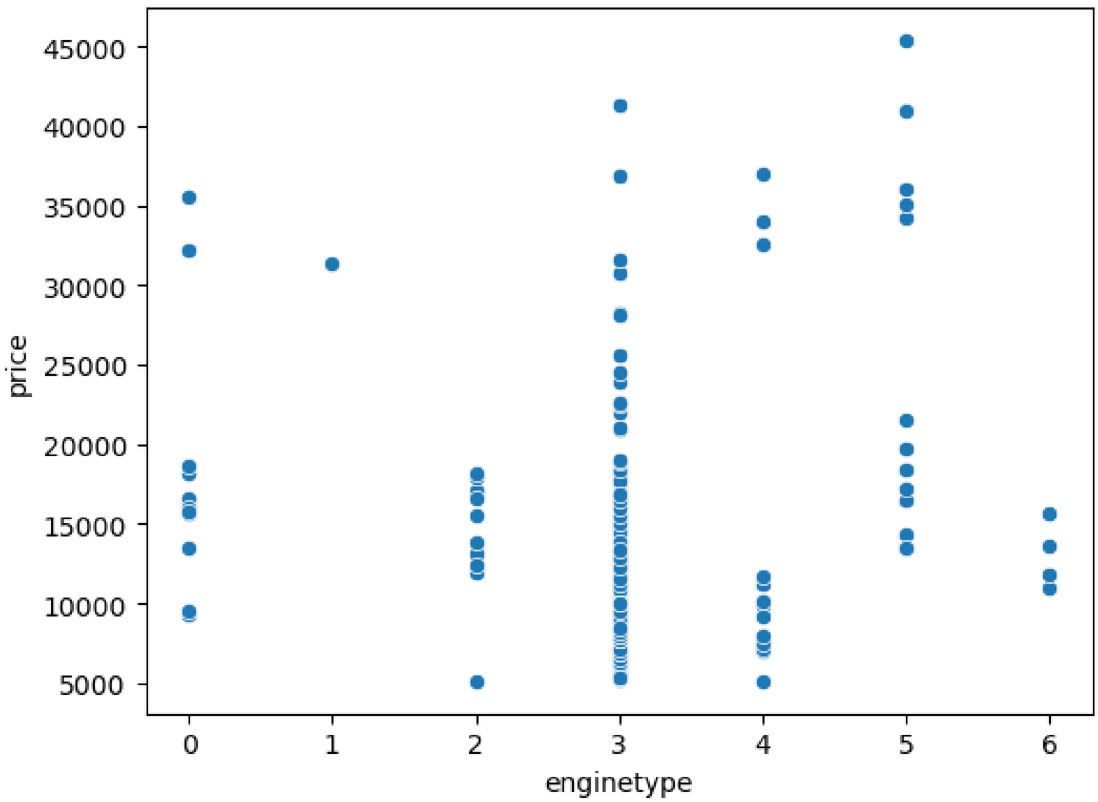

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
85
90
95
100
105
110
115
120
wheelbase

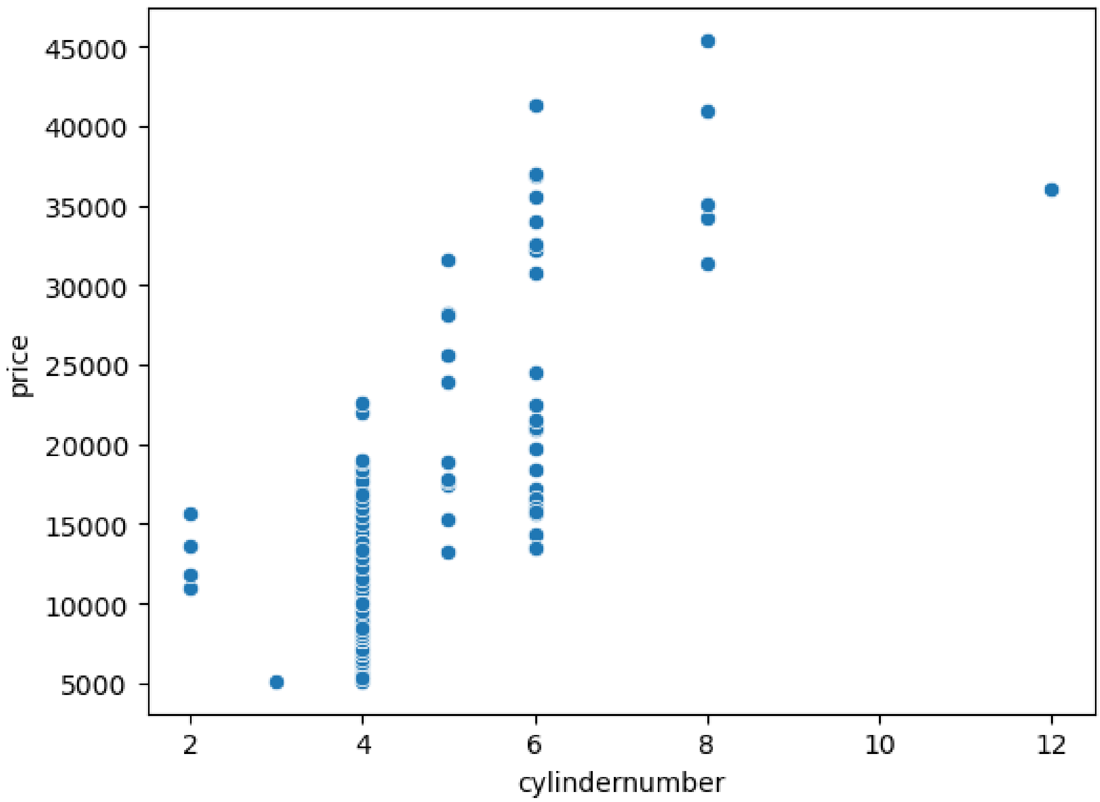

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
140
150
160
170
180
190
200
210
carlength

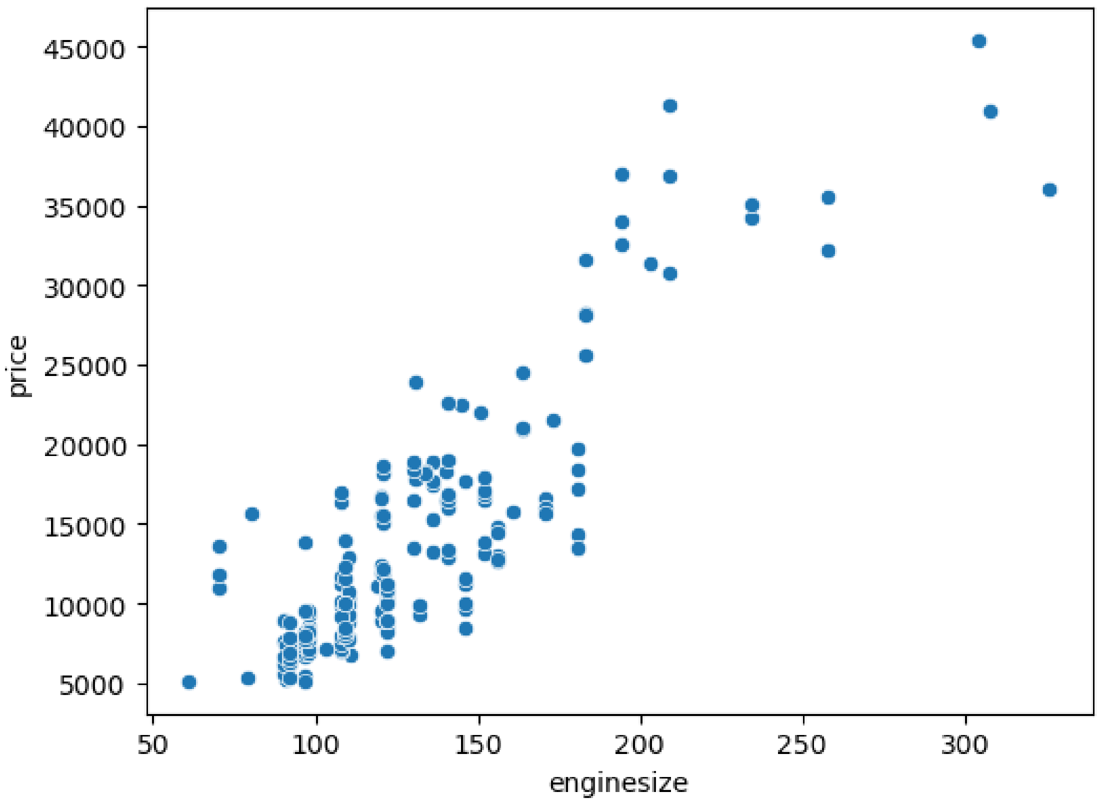

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
60
62
64
66
68
70
72
carwidth

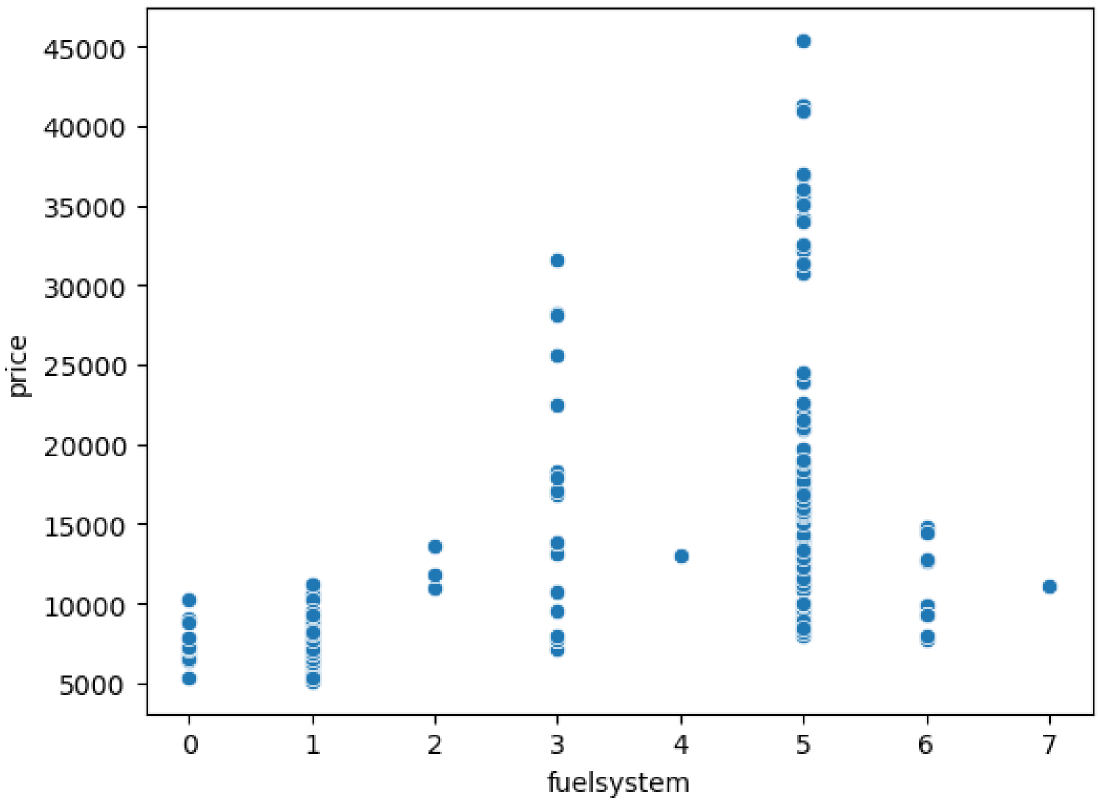

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
48
50
52
54
56
58
60
carheight

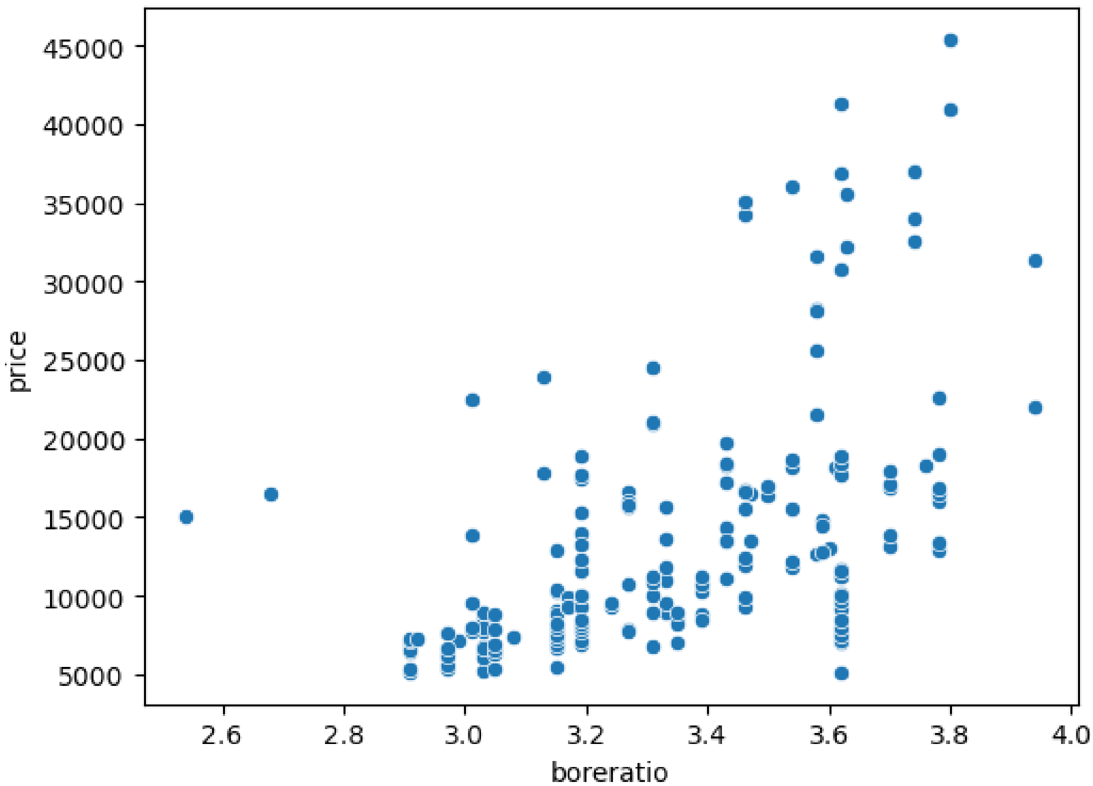

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
1500
2000
2500
3000
3500
4000
curbweight

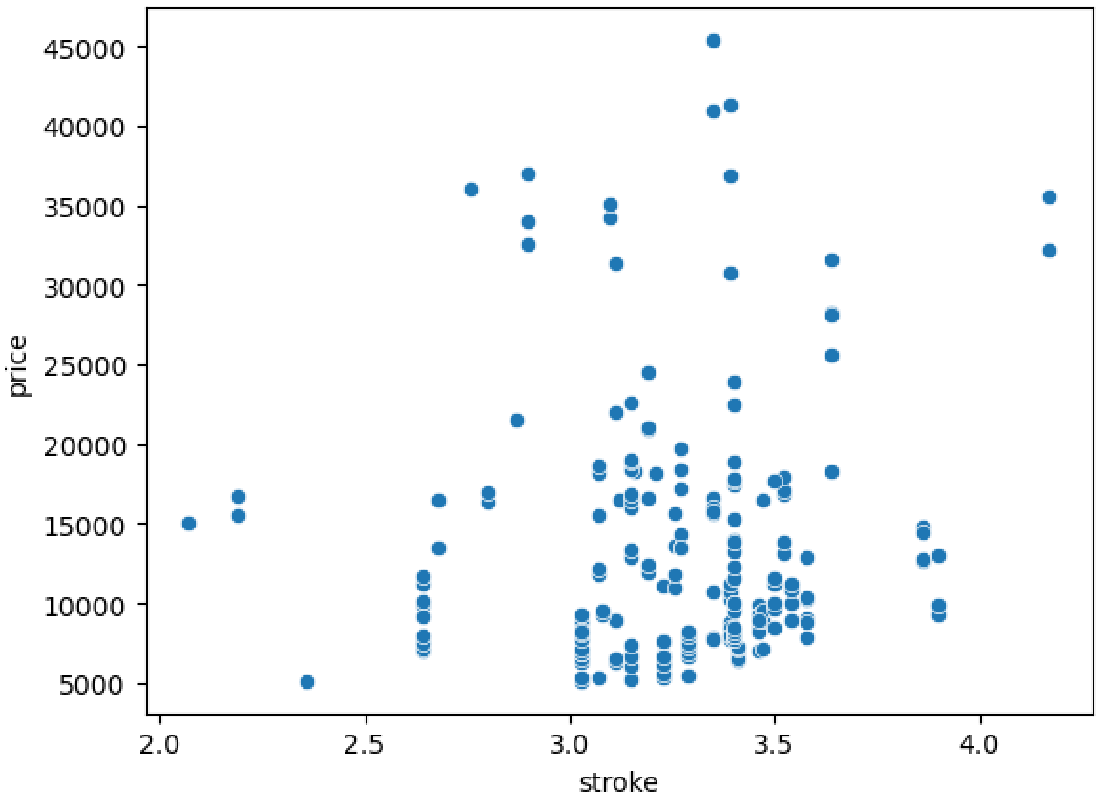

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0
1
2
3
4
5
6
enginetype

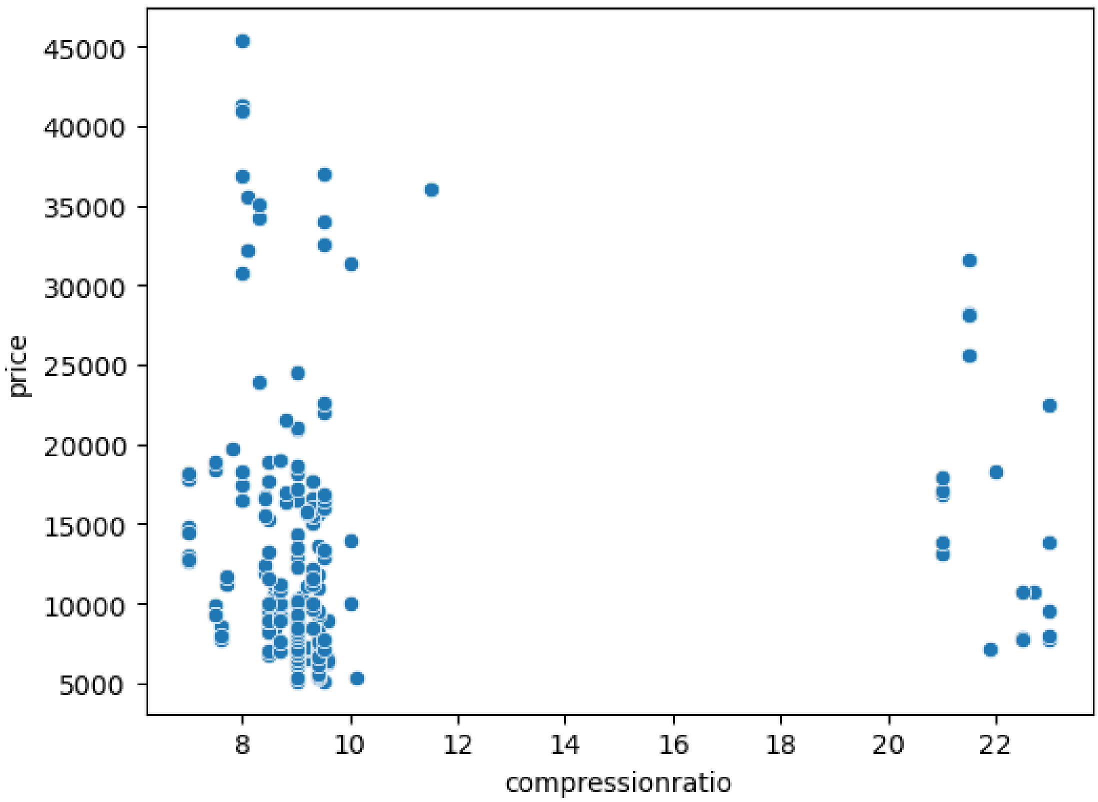

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
2
4
6
8
10
12
cylindernumber


45000
40000
35000
30000
25000
20000
15000
10000
5000
price
50
100
150
200
250
300
enginesize

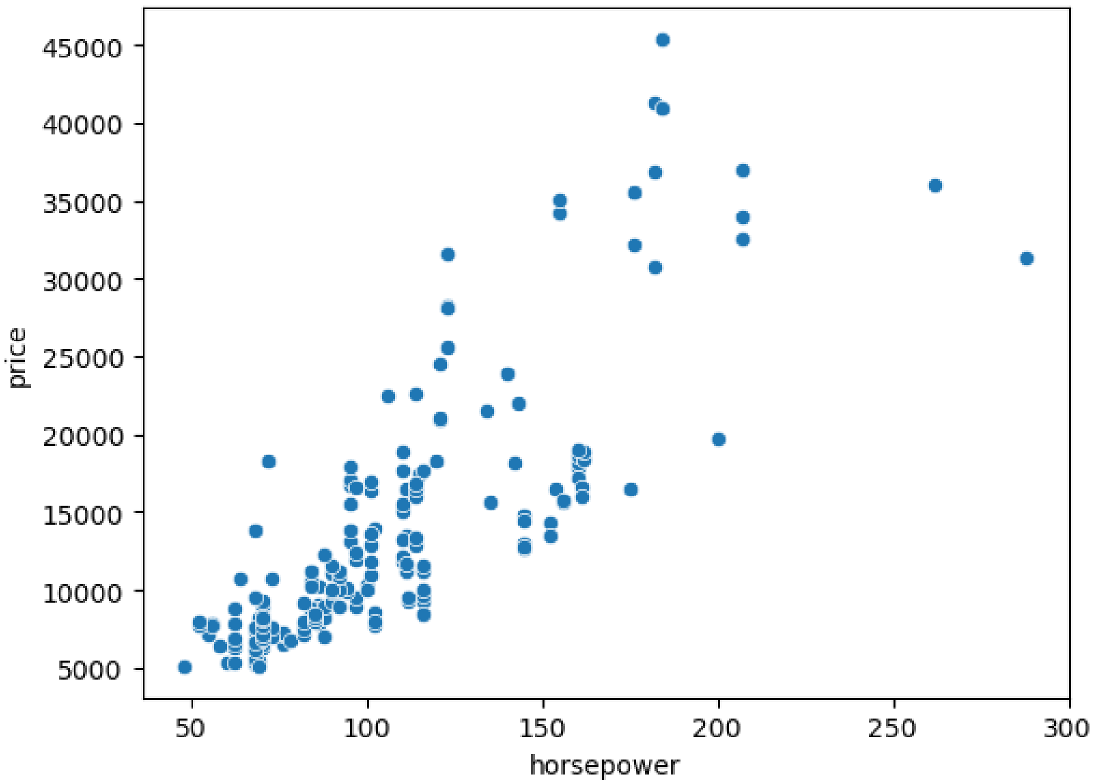

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
0
1
2
3
4
5
6
7
fuelsystem


45000
40000
35000
30000
25000
20000
15000
10000
5000
price
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
boreratio

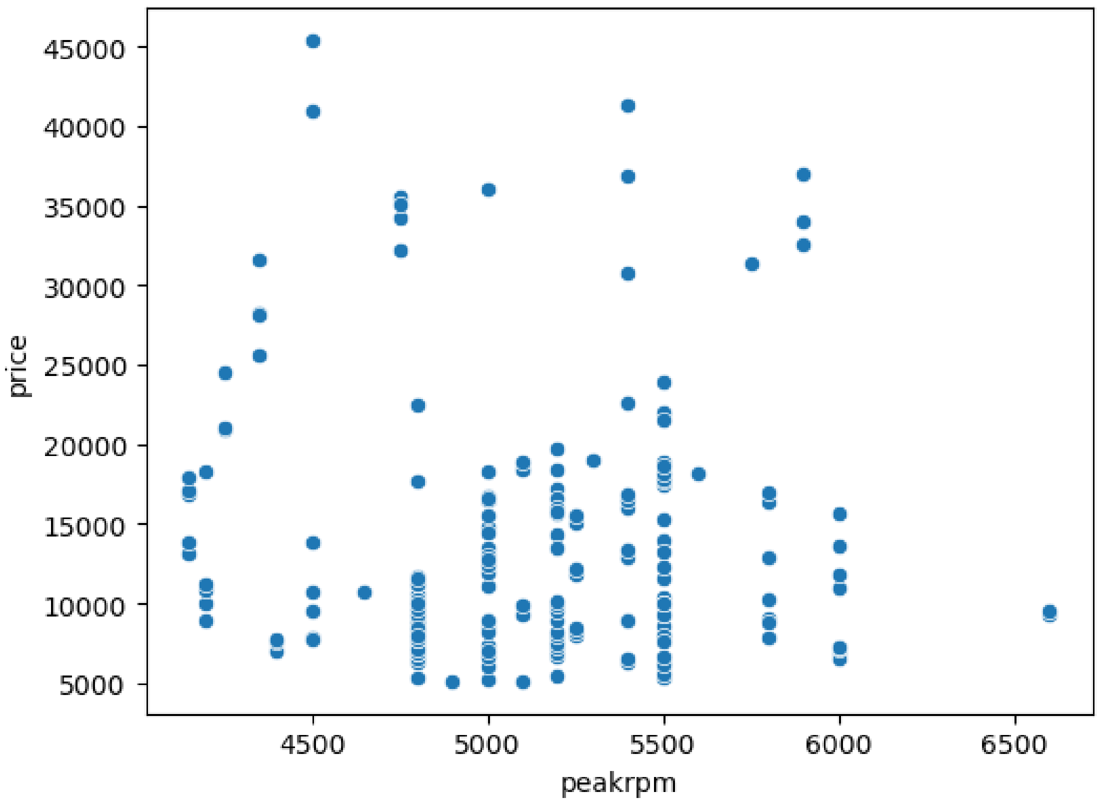

45000
40000
35000
30000
25000
20000
15000
10000
5000
price
2.0
2.5
3.0
3.5
4.0
stroke



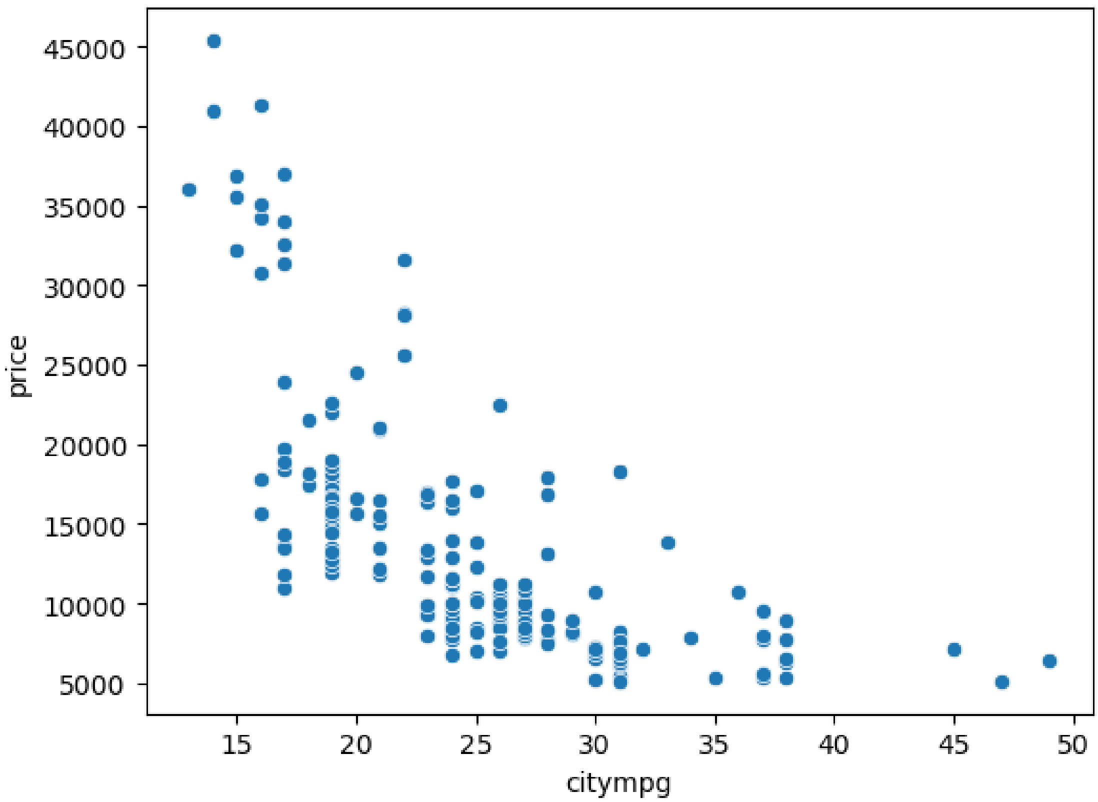
price
compressionratio
45000
40000
35000
30000
25000
20000
15000
10000
5000
8
10
12
14
16
18
20
22

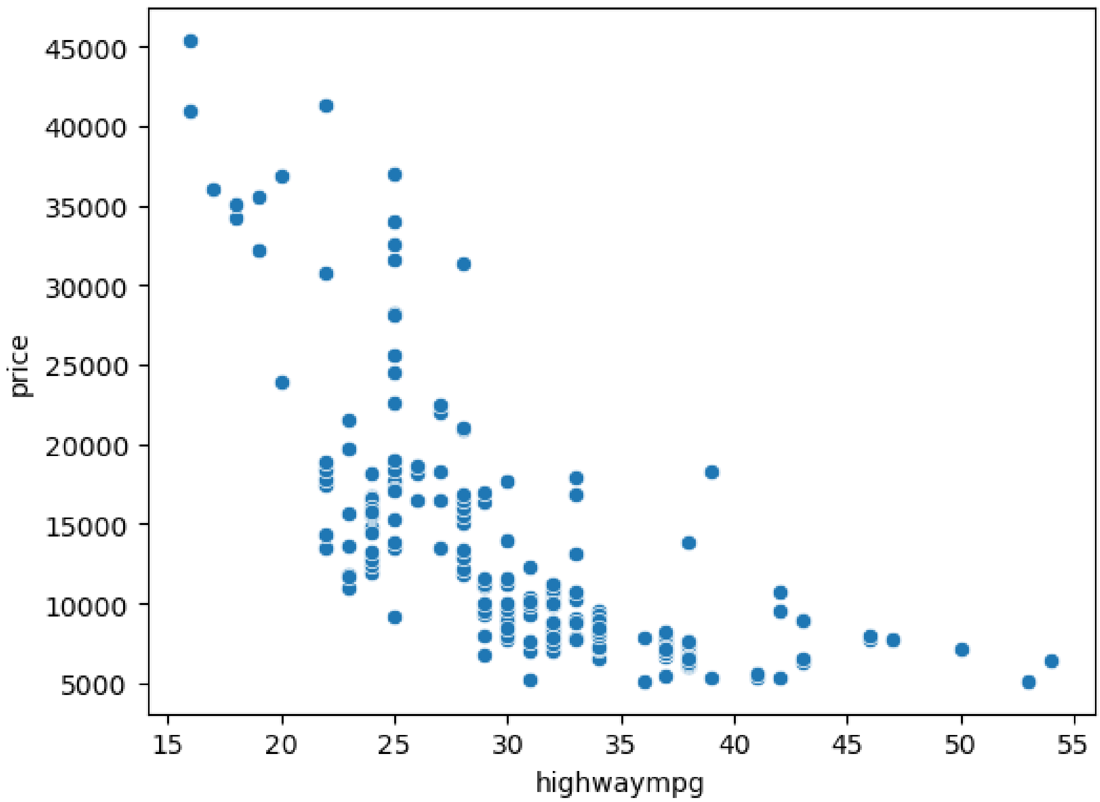


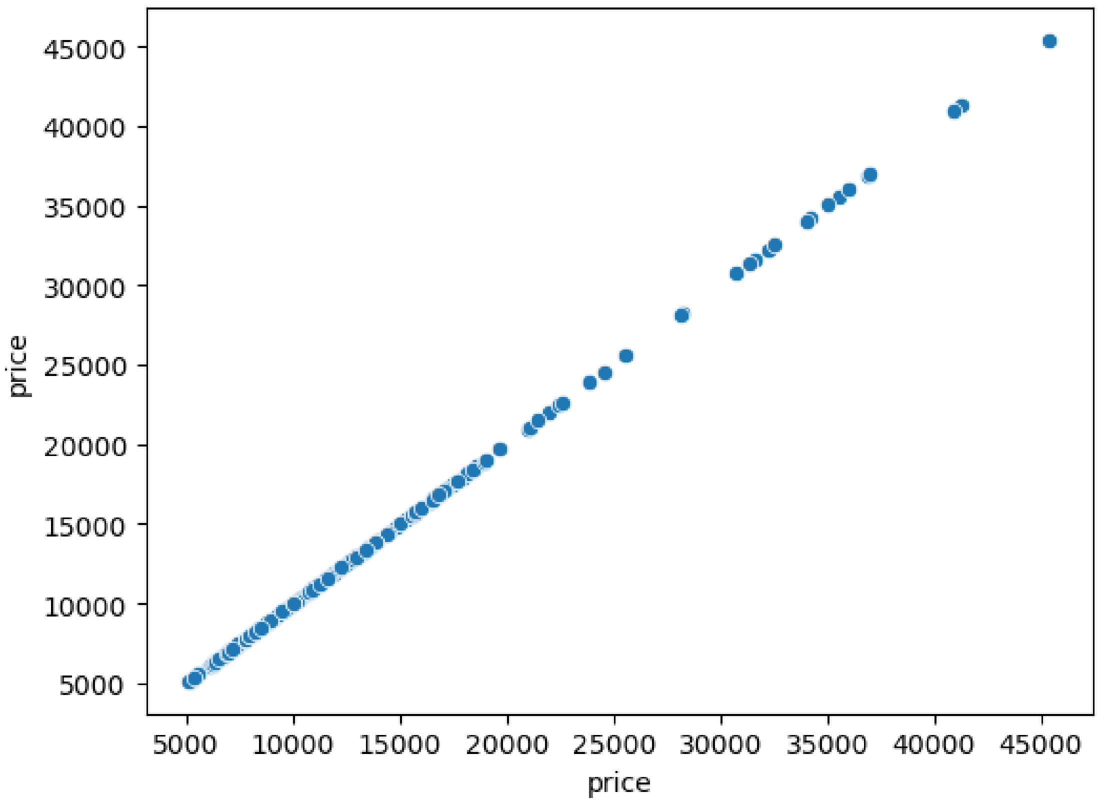
45000
40000
35000
30000
25000
20000
15000
10000
5000
price
50
100
150
200
250
300
horsepower



45000
40000
35000
30000
25000
20000
15000
10000
5000
price
4500
5000
5500
6000
6500
peakrpm


45000
40000
35000
30000
25000
20000
15000
10000
5000
price
15
20
25
30
35
40
45
50
citympg


price
highwaympg
45000
40000
35000
30000
25000
20000
15000
10000
5000
15
20
25
30
35
40
45
50
55



[41]:
```
# plot a heatmap to show the correlation among numerical variables.
```

[43]:
```
# Create a correlation matrix between numerical columns
correlation_matrix = df.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[43], line 2
      1 # Create a correlation matrix between numerical columns
----> 2 correlation_matrix = df.corr()
      4 plt.figure(figsize=(12, 10))
      5 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5)
```

```
File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:10704, in DataFrame.
 ↪corr(self, method, min_periods, numeric_only)
  10702 cols = data.columns
  10703 idx = cols.copy()
> 10704 mat = data.to_numpy(dtype=float, na_value=np.nan, copy=False)
  10706 if method == "pearson":
  10707     correl = libalgos.nancorr(mat, minp=min_periods)

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:1889, in DataFrame.
 ↪to_numpy(self, dtype, copy, na_value)
   1887 if dtype is not None:
   1888     dtype = np.dtype(dtype)
-> 1889 result = self._mgr.as_array(dtype=dtype, copy=copy, na_value=na_value)
   1890 if result.dtype is not dtype:
   1891     result = np.array(result, dtype=dtype, copy=False)

File ~\anaconda3\Lib\site-packages\pandas\core\internals\managers.py:1656, in␣
 ↪BlockManager.as_array(self, dtype, copy, na_value)
   1654         arr.flags.writeable = False
   1655 else:
-> 1656     arr = self._interleave(dtype=dtype, na_value=na_value)
   1657     # The underlying data was copied within _interleave, so no need
   1658     # to further copy if copy=True or setting na_value
   1660 if na_value is lib.no_default:

File ~\anaconda3\Lib\site-packages\pandas\core\internals\managers.py:1715, in␣
 ↪BlockManager._interleave(self, dtype, na_value)
   1713     else:
   1714         arr = blk.get_values(dtype)
-> 1715     result[rl.indexer] = arr
   1716     itemmask[rl.indexer] = 1
   1718 if not itemmask.all():

ValueError: could not convert string to float: 'alfa-romero giulia'
```

3.1 Observations from scatter plot and heatmap

The independent variables that affects the price of car are, 1. ‘drivewheel’ 2. ‘wheelbase’ 3. ‘carlength’ 4. ‘carwidth’ 5. ‘curbweight’ 6. ‘cylindernumber’ 7. ‘enginesize’ 8. ‘fuelsystem’ 9. ‘boreratio’ 10. ‘horsepower’ 11. ‘citympg’ (negative correlation) 12. ‘highwaympg’ (negative correlation)

[44]:
```
#sort the dataset with the necessary features only.
columns_to_sort=['drivewheel','wheelbase','carlength','carwidth',
                ␣
 ↪'curbweight','cylindernumber','enginesize','fuelsystem','boreratio','horsepower','citympg',
```

[45]:
```
independent_data_for_model=df[columns_to_sort]
independent_data_for_model.head()
```

[45]:
```
   drivewheel  wheelbase  carlength  carwidth  curbweight  cylindernumber  \
0           2       88.6      168.8      64.1        2548               4
1           2       88.6      168.8      64.1        2548               4
2           2       94.5      171.2      65.5        2823               6
3           1       99.8      176.6      66.2        2337               4
4           0       99.4      176.6      66.4        2824               5

   enginesize  fuelsystem  boreratio  horsepower  citympg  highwaympg
0         130           5       3.47         111       21          27
1         130           5       3.47         111       21          27
2         152           5       2.68         154       19          26
3         109           5       3.19         102       24          30
4         136           5       3.19         115       18          22
```

3.2 Create a ML model for Car Price Prediction using Linear Regression-Multiple Variables

The independent variables that affects the price of car are, 1. 'drivewheel' 2. 'wheelbase' 3. 'carlength' 4. 'carwidth' 5. 'curbweight' 6. 'cylindernumber' 7. 'enginesize' 8. 'fuelsystem' 9. 'boreratio' 10. 'horsepower' 11. 'citympg' (negative correlation) 12. 'highwaympg' (negative correlation)

[46]:
```
#Using Linear Regression Model
```

[47]:
```
#spliting the dataset to x(independent variables) and y(dependent␣
↪variable='Price')
x=df[['drivewheel','wheelbase','carlength','carwidth',
                         ␣
↪'curbweight','cylindernumber','enginesize','fuelsystem','boreratio','horsepower','citympg',
y=df.price
```

[48]:
```
#independant variable,x
x
```

[48]:
```
     drivewheel  wheelbase  carlength  carwidth  curbweight  cylindernumber  \
0             2       88.6      168.8      64.1        2548               4
1             2       88.6      168.8      64.1        2548               4
2             2       94.5      171.2      65.5        2823               6
3             1       99.8      176.6      66.2        2337               4
4             0       99.4      176.6      66.4        2824               5
..          …        …        …       …        …              …
200           2      109.1      188.8      68.9        2952               4
201           2      109.1      188.8      68.8        3049               4
202           2      109.1      188.8      68.9        3012               6
```

```
203          2     109.1     188.8     68.9      3217                6
204          2     109.1     188.8     68.9      3062                4

     enginesize  fuelsystem  boreratio  horsepower  citympg  highwaympg
0           130           5       3.47         111       21          27
1           130           5       3.47         111       21          27
2           152           5       2.68         154       19          26
3           109           5       3.19         102       24          30
4           136           5       3.19         115       18          22
..          …           …         …          …         …           …
200         141           5       3.78         114       23          28
201         141           5       3.78         160       19          25
202         173           5       3.58         134       18          23
203         145           3       3.01         106       26          27
204         141           5       3.78         114       19          25

[205 rows x 12 columns]
```

[49]:
```
#dependant variable,y
y
```

[49]:
```
0      13495.0
1      16500.0
2      16500.0
3      13950.0
4      17450.0
        …
200    16845.0
201    19045.0
202    21485.0
203    22470.0
204    22625.0
Name: price, Length: 205, dtype: float64
```

[50]:
```
#import libraries
from sklearn.linear_model import LinearRegression
```

[51]:
```
linre=LinearRegression()
linre
```

[51]:
```
LinearRegression()
```

[52]:
```
linre.fit(x,y)
```

[52]:
```
LinearRegression()
```

[53]:
```
#predict the price
value=[[2,88.6,168.8,64.1,2548,4,130,5,3.47,111,21,27]]
predicted=linre.predict(value)
print(predicted)
```

```
[13039.64047938]
```

```
C:\Users\djerb\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

[54]:
```
#check the Accuracy of created ML model
linre.score(x,y)
```

[54]: 0.830484708905505

[55]:
```
x.head(1)
```

[55]:
```
   drivewheel  wheelbase  carlength  carwidth  curbweight  cylindernumber  \
0           2       88.6      168.8      64.1        2548               4

   enginesize  fuelsystem  boreratio  horsepower  citympg  highwaympg
0         130           5       3.47         111       21          27
```

[56]:
```
y.head(1)
```

[56]:
```
0    13495.0
Name: price, dtype: float64
```

4 Result analysis

1. The Score of created ML model is good.
2. The predicted value lies closer to the actual value, So The created ML model is working good.

5 2- Investigation of the Diamond Dataset

[59]:
```
#Import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
%matplotlib inline
sns.set_style('whitegrid')
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from kneed import KneeLocator
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
from sklearn.model_selection import cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

[61]:
```python
diamonds = pd.read_csv('C:/Users/djerb/Downloads/Diamonds/Diamonds/diamonds.
 ↪csv')
diamonds.head()
```

[61]:
```
   Unnamed: 0  carat      cut color clarity  depth  table  price     x     y  \
0           1   0.23    Ideal     E     SI2   61.5   55.0    326  3.95  3.98
1           2   0.21  Premium     E     SI1   59.8   61.0    326  3.89  3.84
2           3   0.23     Good     E     VS1   56.9   65.0    327  4.05  4.07
3           4   0.29  Premium     I     VS2   62.4   58.0    334  4.20  4.23
4           5   0.31     Good     J     SI2   63.3   58.0    335  4.34  4.35

      z
0  2.43
1  2.31
2  2.31
3  2.63
4  2.75
```

[62]:
```python
diamonds["size"] = diamonds["x"] * diamonds["y"] * diamonds["z"]
col_names = ['price','carat', 'size', 'depth', 'table', 'color', 'clarity',␣
 ↪'cut']
diamonds = diamonds[col_names]
diamonds.head()
```

[62]:
```
   price  carat       size  depth  table color clarity      cut
0    326   0.23  38.202030   61.5   55.0     E     SI2    Ideal
```

	price	carat	size	depth	table	color	clarity	cut
1	326	0.21	34.505856	59.8	61.0	E	SI1	Premium
2	327	0.23	38.076885	56.9	65.0	E	VS1	Good
3	334	0.29	46.724580	62.4	58.0	I	VS2	Premium
4	335	0.31	51.917250	63.3	58.0	J	SI2	Good

[63]:
```
#Encoding categorical features
encoder = LabelEncoder()

encoder.fit(diamonds['cut'])
diamonds['cut'] = encoder.transform(diamonds['cut'])

encoder.fit(diamonds['color'])
diamonds['color'] = encoder.transform(diamonds['color'])

encoder.fit(diamonds['clarity'])
diamonds['clarity'] = encoder.transform(diamonds['clarity'])

diamonds.head()
```

[63]:

	price	carat	size	depth	table	color	clarity	cut
0	326	0.23	38.202030	61.5	55.0	1	3	2
1	326	0.21	34.505856	59.8	61.0	1	2	3
2	327	0.23	38.076885	56.9	65.0	1	4	1
3	334	0.29	46.724580	62.4	58.0	5	5	3
4	335	0.31	51.917250	63.3	58.0	6	3	1

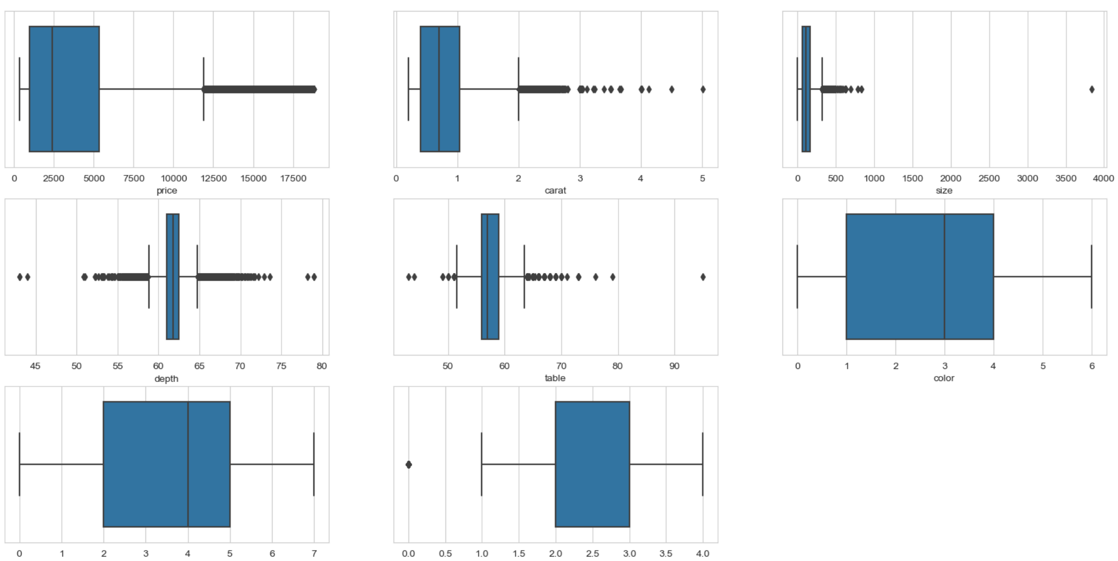
[64]:
```
#Split dataset in train and test Data
df = diamonds.copy()

#"cut" is my predictor variable
X = df.drop('cut',axis=1)

#Label or target variable
y = df['cut']

#Define the split in Training/Test and their proportions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 42)

print(X_train.shape, y_train.shape)
```

```
(43152, 7) (43152,)
```

[65]:
```
df.info()
df.columns
df.shape
df.dtypes
```

```python
df.isnull().sum()
fig = plt.figure(figsize=(20,20))
for col in range(len(df.columns)) :
    fig.add_subplot(6,3,col+1)
    sns.boxplot(x=df.iloc[ : , col])
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   price     53940 non-null  int64
 1   carat     53940 non-null  float64
 2   size      53940 non-null  float64
 3   depth     53940 non-null  float64
 4   table     53940 non-null  float64
 5   color     53940 non-null  int32
 6   clarity   53940 non-null  int32
 7   cut       53940 non-null  int32
dtypes: float64(4), int32(3), int64(1)
memory usage: 2.7 MB
```



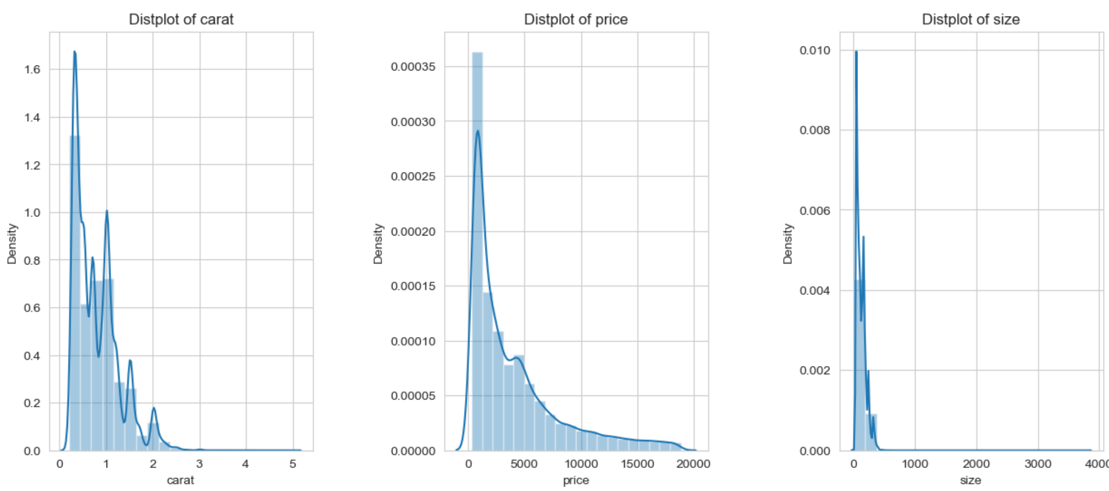
[68]:
```python
df.describe().T
#Check correlation between different features
df.corr()
```

[68]:

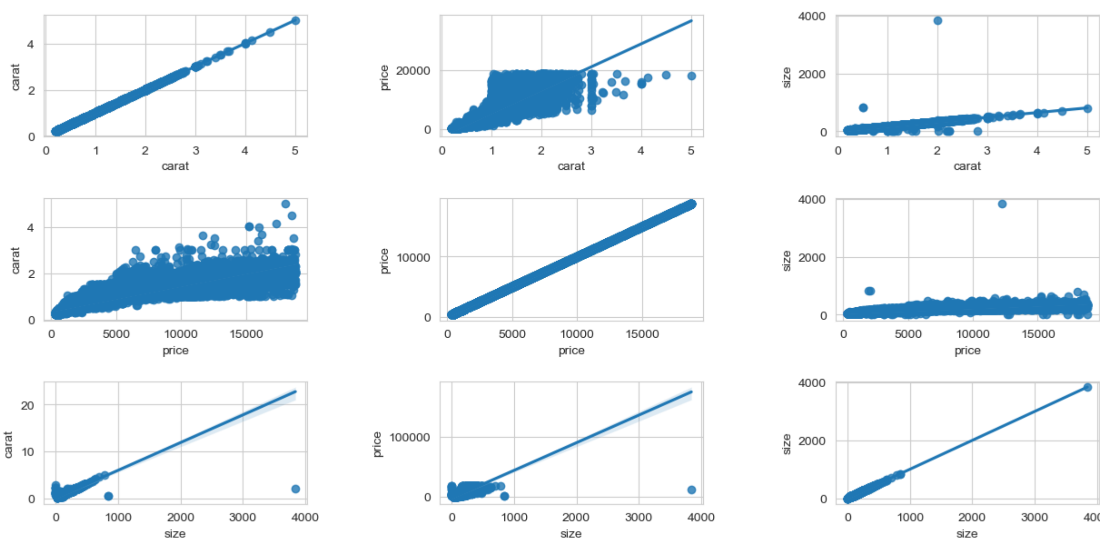
	price	carat	size	depth	table	color	clarity	cut
price	1.000000	0.921591	0.902385	-0.010647	0.127134	0.172511	-0.071535	0.039860
carat	0.921591	1.000000	0.976308	0.028224	0.181618	0.291437	-0.214290	0.017124
size	0.902385	0.976308	1.000000	0.009157	0.167400	0.284267	-0.206632	0.021440
depth	-0.010647	0.028224	0.009157	1.000000	-0.295779	0.047279	-0.053080	-0.194249
table	0.127134	0.181618	0.167400	-0.295779	1.000000	0.026465	-0.088223	0.150327
color	0.172511	0.291437	0.284267	0.047279	0.026465	1.000000	-0.027795	0.000304
clarity	-0.071535	-0.214290	-0.206632	-0.053080	-0.088223	-0.027795	1.000000	0.028235
cut	0.039860	0.017124	0.021440	-0.194249	0.150327	0.000304	0.028235	1.000000

[71]:

```
plt.figure(1 , figsize = (15 , 6))
n = 0
for x in ['carat' , 'price' , 'size']:
    n += 1
    plt.subplot(1 , 3 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.5)
    sns.distplot(df[x] , bins = 20)
    plt.title('Distplot of {}'.format(x))
plt.show()
```



[72]:
```python
#Plot to see relationship between the features 'carat', 'price', 'size'
plt.figure(1 , figsize = (15 , 7))
n = 0
for x in ['carat' , 'price' , 'size']:
    for y in ['carat' , 'price' , 'size']:
        n += 1
        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
        sns.regplot(x = x , y = y , data = df)
        plt.ylabel(y.split()[0]+' '+y.split()[1] if len(y.split()) > 1 else y )
plt.show()
```



[73]:
```python
#Normalize the data and avoid the problem that generates outliers
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(df)
```

[74]:
```python
df1 = pd.DataFrame(X_scaled)
```

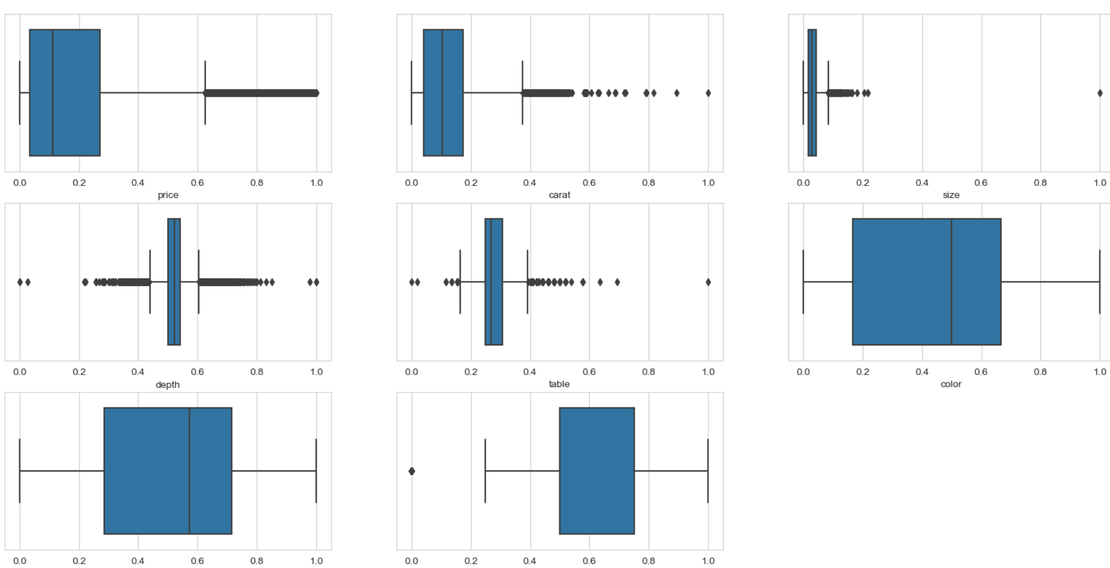
[75]:
```python
df1
```

[75]:

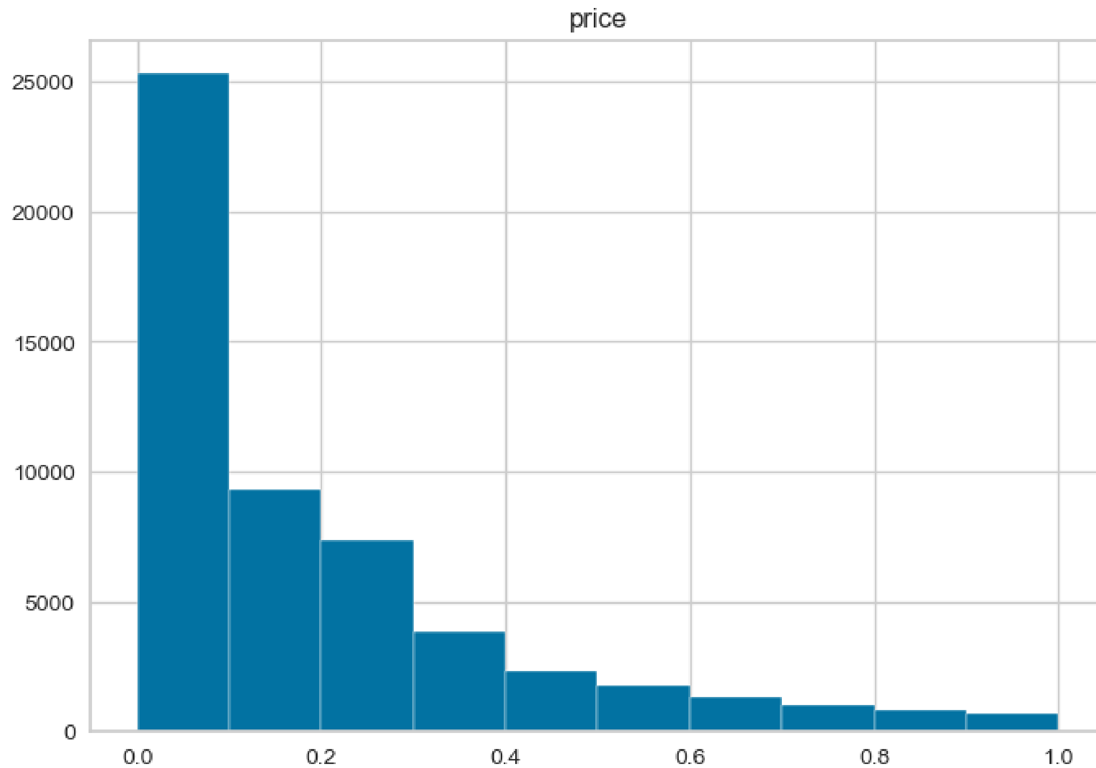
	0	1	2	3	4	5	6	\
0	0.000000	0.006237	0.009947	0.513889	0.230769	0.166667	0.428571	
1	0.000000	0.002079	0.008985	0.466667	0.346154	0.166667	0.285714	
2	0.000054	0.006237	0.009914	0.386111	0.423077	0.166667	0.571429	
3	0.000433	0.018711	0.012166	0.538889	0.288462	0.833333	0.714286	
4	0.000487	0.022869	0.013518	0.563889	0.288462	1.000000	0.428571	
…	…	…	…	…	…	…	…	
53935	0.131427	0.108108	0.030183	0.494444	0.269231	0.000000	0.285714	
53936	0.131427	0.108108	0.030753	0.558333	0.230769	0.000000	0.285714	

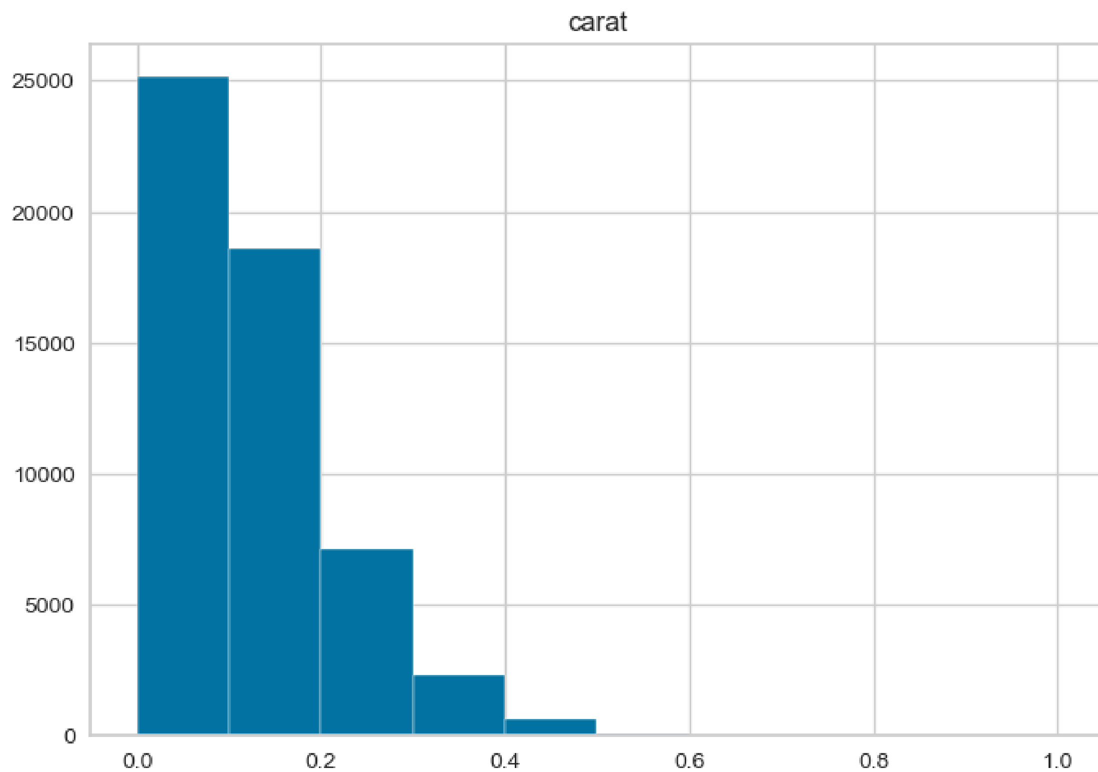
```
53937  0.131427  0.103950  0.029800  0.550000  0.326923  0.000000  0.285714
53938  0.131427  0.137214  0.036652  0.500000  0.288462  0.666667  0.428571
53939  0.131427  0.114345  0.032435  0.533333  0.230769  0.000000  0.428571

          7
0      0.50
1      0.75
2      0.25
3      0.75
4      0.25
…      …
53935  0.50
53936  0.25
53937  1.00
53938  0.75
53939  0.50

[53940 rows x 8 columns]
```

[76]:
```python
df1.columns
df1.columns =['price', 'carat', 'size', 'depth', 'table', 'color', 'clarity',␣
 ↪'cut']
df1.describe()
#Use boxplot to check numeric features again
fig = plt.figure(figsize=(20,20))
for col in range(len(df1.columns)) :
    fig.add_subplot(6,3,col+1)
    sns.boxplot(x=df1.iloc[ : , col])
plt.show()
```

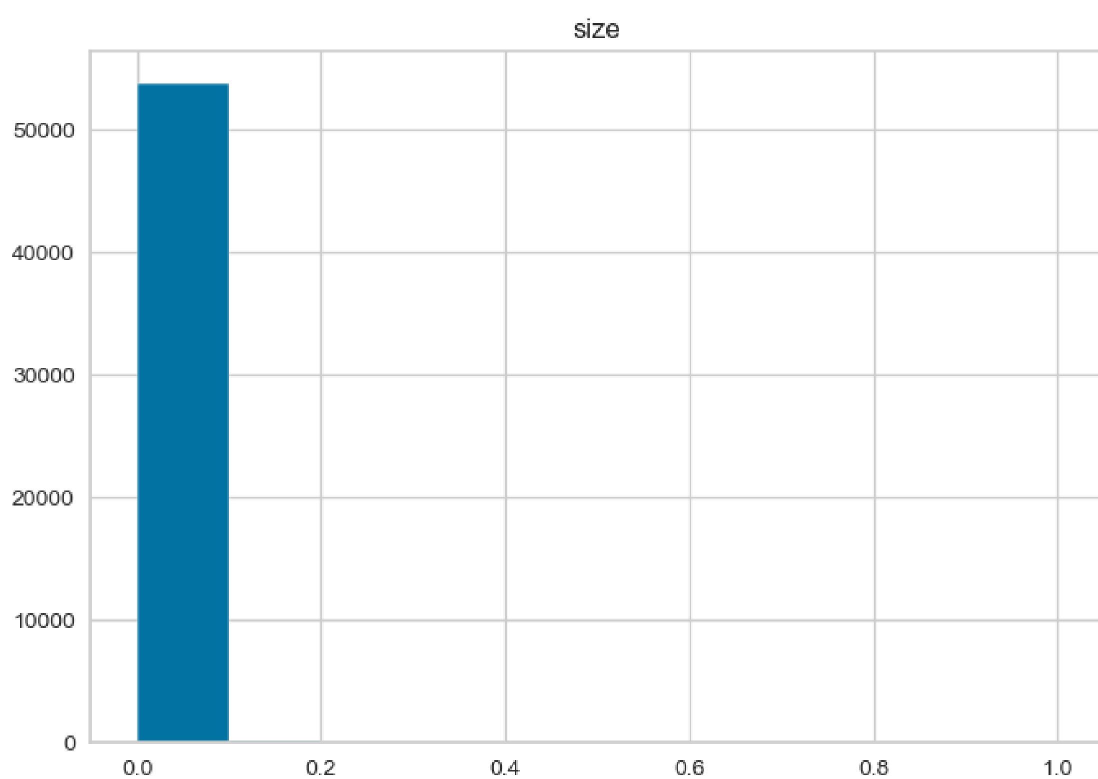


[83]:
```
for col in df1:
    df1[[col]].hist()
```

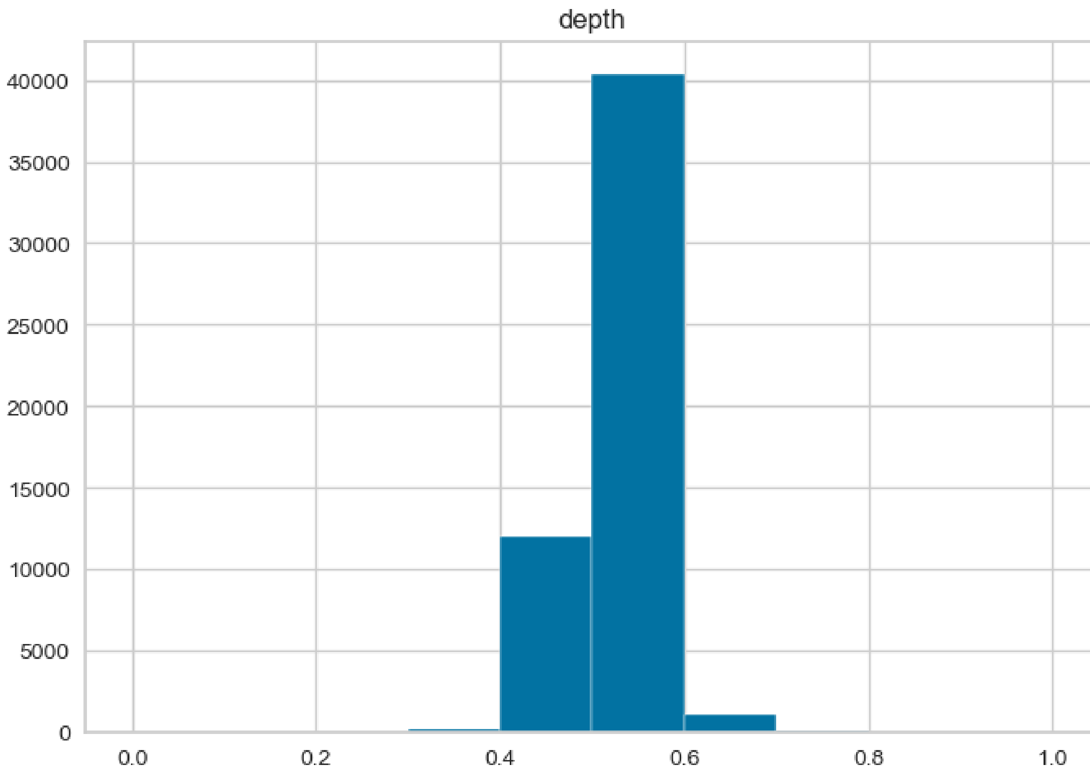




carat
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0



size
50000
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0



depth
0
5000
10000
15000
20000
25000
30000
35000
40000
0.0
0.2
0.4
0.6
0.8
1.0

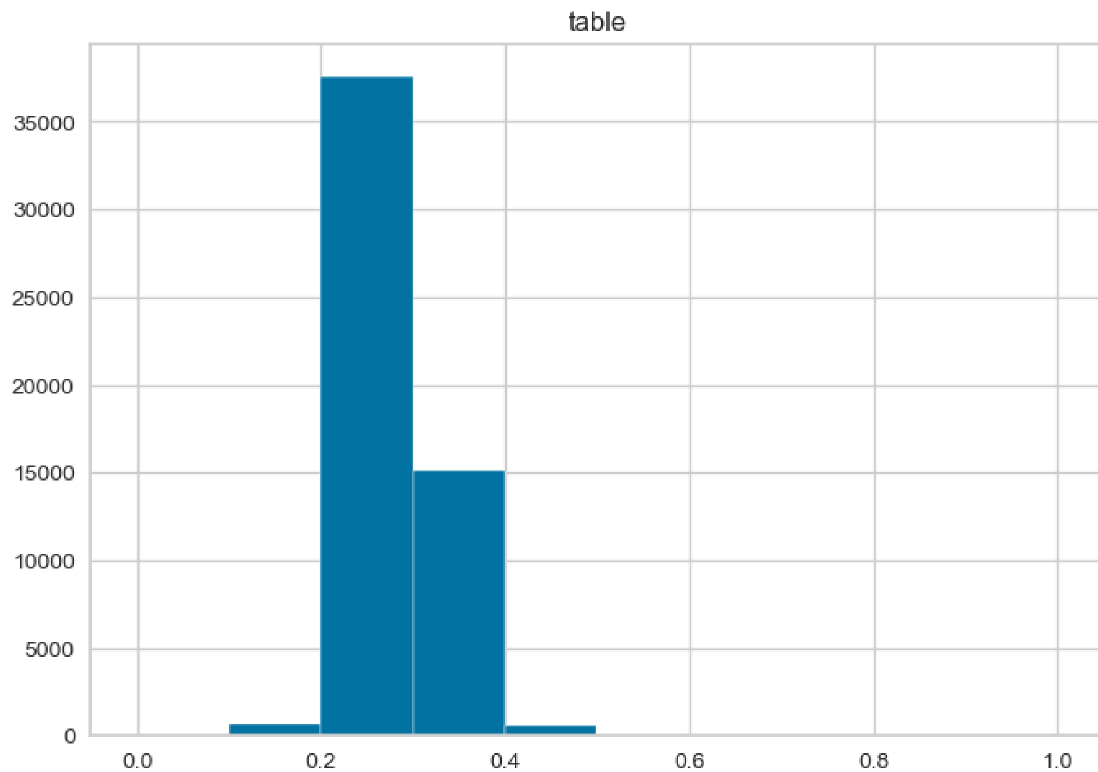
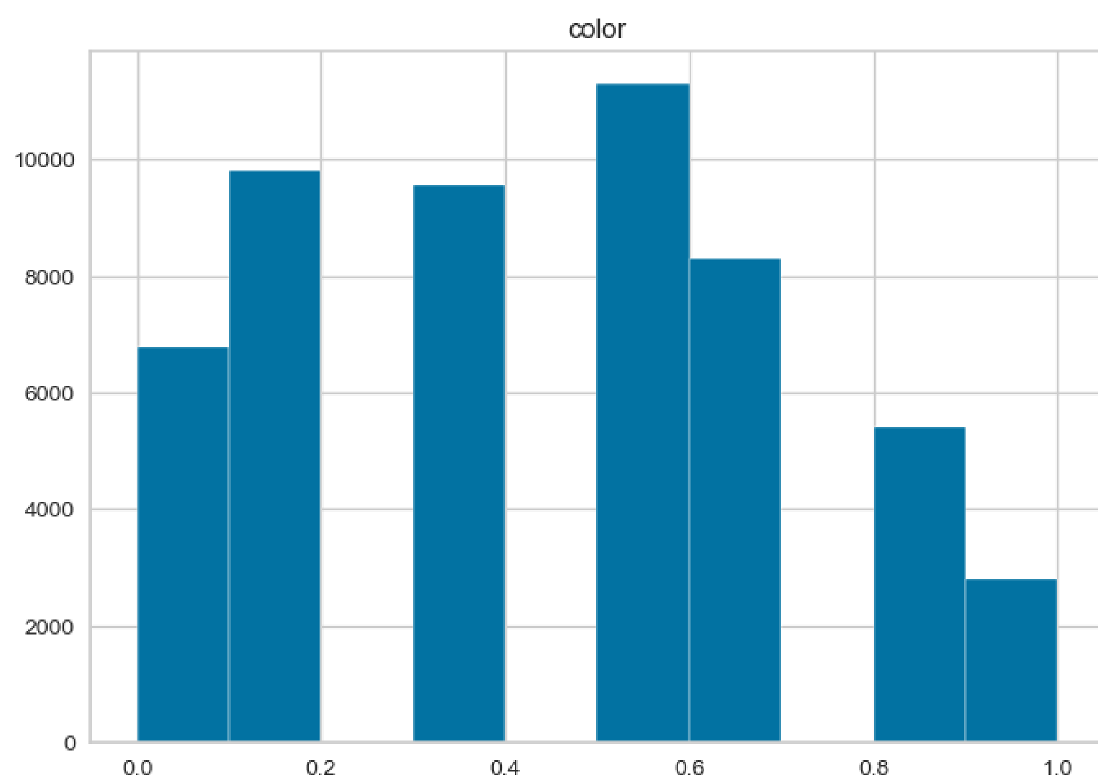
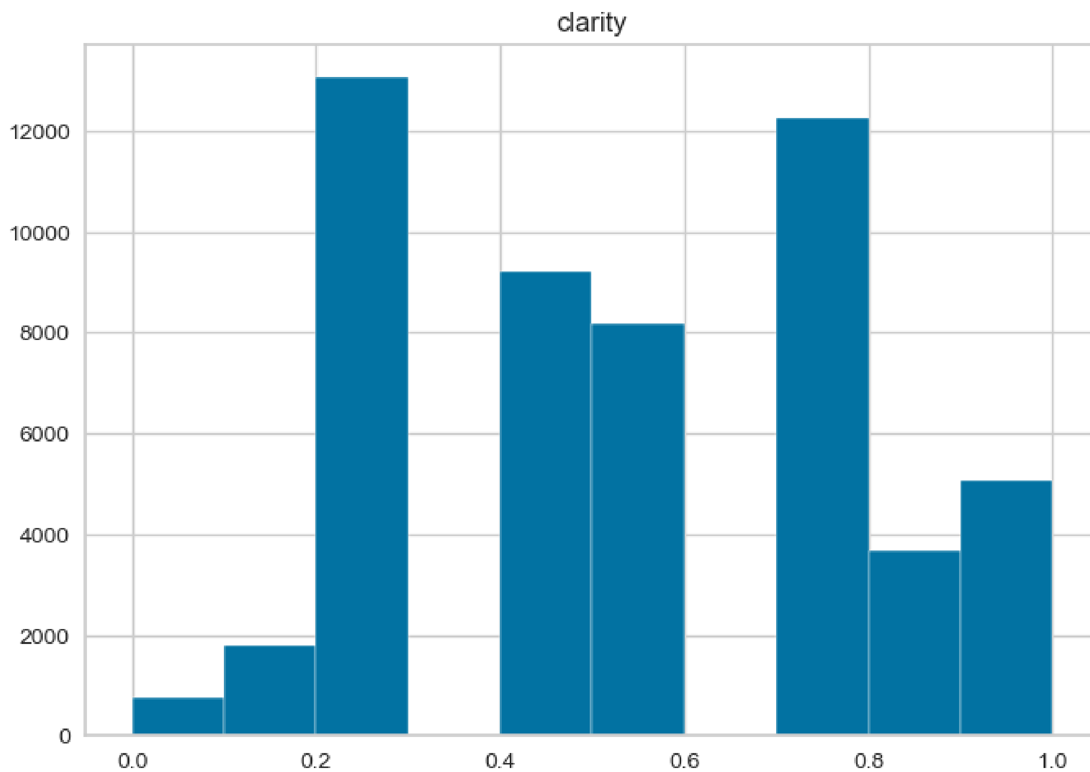
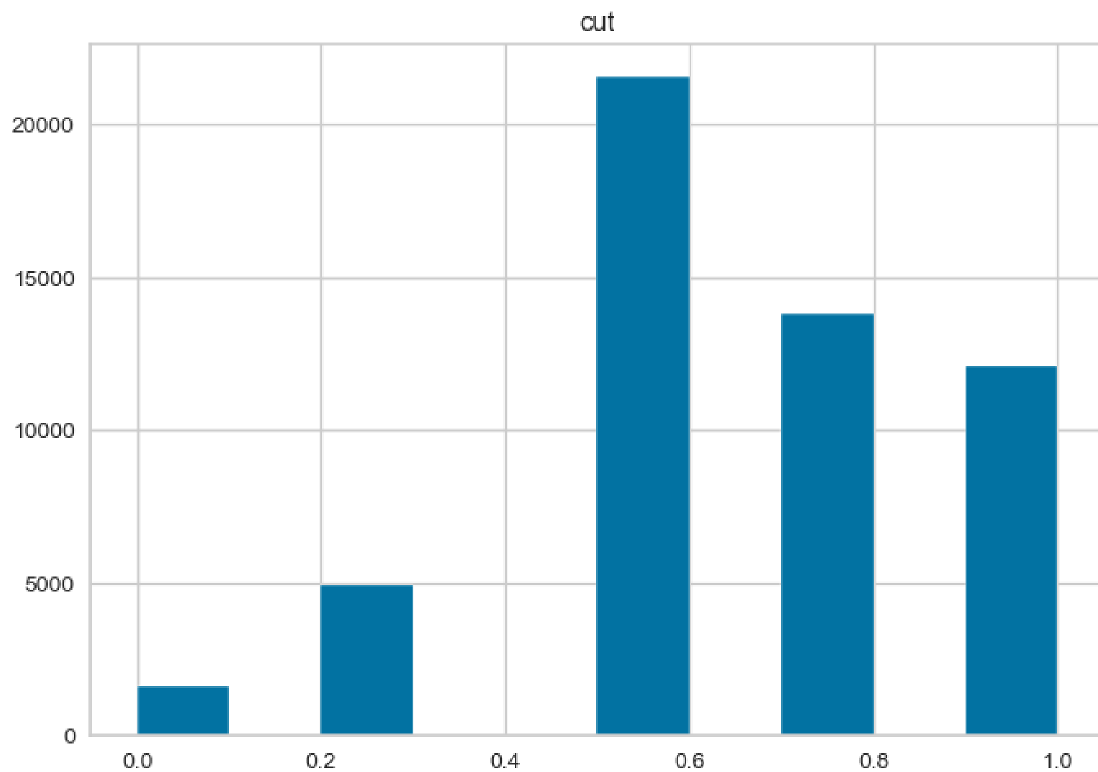


table
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


color
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0


clarity
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0



[90]:
```
# Create an instance of PCA
pca = PCA(n_components = 2) #Define n_components=2

pca.fit(df1)
df2 = pca.transform(df1)
```

[91]:
```
from yellowbrick.cluster import KElbowVisualizer

model = KMeans(random_state=1)
visualizer = KElbowVisualizer(model, k=(2,10))

visualizer.fit(df1)
visualizer.show()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[91], line 6
      3 model = KMeans(random_state=1)
      4 visualizer = KElbowVisualizer(model, k=(2,10))
----> 6 visualizer.fit(df1)
      7 visualizer.show()
```

```
      8 plt.show()

File ~\anaconda3\Lib\site-packages\yellowbrick\cluster\elbow.py:339, in␣
 ↪KElbowVisualizer.fit(self, X, y, **kwargs)
    337 # Set the k value and fit the model
    338 self.estimator.set_params(n_clusters=k)
--> 339 self.estimator.fit(X, **kwargs)
    341 # Append the time and score to our plottable metrics
    342 self.k_timers_.append(time.time() - start)

File ~\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1468, in KMeans.
 ↪fit(self, X, y, sample_weight)
   1465     print("Initialization complete")
   1467 # run a k-means once
-> 1468 labels, inertia, centers, n_iter_ = kmeans_single(
   1469     X,
   1470     sample_weight,
   1471     centers_init,
   1472     max_iter=self.max_iter,
   1473     verbose=self.verbose,
   1474     tol=self._tol,
   1475     n_threads=self._n_threads,
   1476 )
   1478 # determine if these results are the best so far
   1479 # we chose a new run if it has a better inertia and the clustering is
   1480 # different from the best so far (it's possible that the inertia is
   1481 # slightly better even if the clustering is the same with potentially
   1482 # permuted labels, due to rounding errors)
   1483 if best_inertia is None or (
   1484     inertia < best_inertia
   1485     and not _is_same_clustering(labels, best_labels, self.n_clusters)
   1486 ):

File ~\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:679, in␣
 ↪_kmeans_single_lloyd(X, sample_weight, centers_init, max_iter, verbose, tol,␣
 ↪n_threads)
    675 strict_convergence = False
    677 # Threadpoolctl context to limit the number of threads in second level of
    678 # nested parallelism (i.e. BLAS) to avoid oversubscription.
--> 679 with threadpool_limits(limits=1, user_api="blas"):
    680     for i in range(max_iter):
    681         lloyd_iter(
    682             X,
    683             sample_weight,
   (…)
    689             n_threads,
    690         )
```

```
File ~\anaconda3\Lib\site-packages\sklearn\utils\fixes.py:139, in␣
↪threadpool_limits(limits, user_api)
    137     return controller.limit(limits=limits, user_api=user_api)
    138 else:
--> 139     return threadpoolctl.threadpool_limits(limits=limits,␣
↪user_api=user_api)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:171, in threadpool_limits.
↪__init__(self, limits, user_api)
    167 def __init__(self, limits=None, user_api=None):
    168     self._limits, self._user_api, self._prefixes = \
    169         self._check_params(limits, user_api)
--> 171     self._original_info = self._set_threadpool_limits()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:268, in threadpool_limits.
↪_set_threadpool_limits(self)
    265 if self._limits is None:
    266     return None
--> 268 modules = _ThreadpoolInfo(prefixes=self._prefixes,
    269                           user_api=self._user_api)
    270 for module in modules:
    271     # self._limits is a dict {key: num_threads} where key is either
    272     # a prefix or a user_api. If a module matches both, the limit
    273     # corresponding to the prefix is chosed.
    274     if module.prefix in self._limits:

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:340, in _ThreadpoolInfo.
↪__init__(self, user_api, prefixes, modules)
    337     self.user_api = [] if user_api is None else user_api
    339     self.modules = []
--> 340     self._load_modules()
    341     self._warn_if_incompatible_openmp()
    342 else:

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:373, in _ThreadpoolInfo.
↪_load_modules(self)
    371     self._find_modules_with_dyld()
    372 elif sys.platform == "win32":
--> 373     self._find_modules_with_enum_process_module_ex()
    374 else:
    375     self._find_modules_with_dl_iterate_phdr()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:485, in _ThreadpoolInfo.
↪_find_modules_with_enum_process_module_ex(self)
    482         filepath = buf.value
    484         # Store the module if it is supported and selected
--> 485         self._make_module_from_path(filepath)
    486 finally:
```

```
    487     kernel_32.CloseHandle(h_process)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:515, in _ThreadpoolInfo.
↪_make_module_from_path(self, filepath)
    513 if prefix in self.prefixes or user_api in self.user_api:
    514     module_class = globals()[module_class]
--> 515     module = module_class(filepath, prefix, user_api, internal_api)
    516     self.modules.append(module)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:606, in _Module.
↪__init__(self, filepath, prefix, user_api, internal_api)
    604 self.internal_api = internal_api
    605 self._dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
--> 606 self.version = self.get_version()
    607 self.num_threads = self.get_num_threads()
    608 self._get_extra_info()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:646, in _OpenBLASModule.
↪get_version(self)
    643 get_config = getattr(self._dynlib, "openblas_get_config",
    644                      lambda: None)
    645 get_config.restype = ctypes.c_char_p
--> 646 config = get_config().split()
    647 if config[0] == b"OpenBLAS":
    648     return config[1].decode("utf-8")

AttributeError: 'NoneType' object has no attribute 'split'
```

[82]:

```
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300,
    "random_state": 42,
}

# A list holds the SSE values for each k
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(df1)
    sse.append(kmeans.inertia_)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[82], line 12
     10 for k in range(1, 11):
     11     kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
```

```
---> 12     kmeans.fit(df1)
     13     sse.append(kmeans.inertia_)

File ~\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1468, in KMeans.
↪fit(self, X, y, sample_weight)
   1465     print("Initialization complete")
   1467 # run a k-means once
-> 1468 labels, inertia, centers, n_iter_ = kmeans_single(
   1469     X,
   1470     sample_weight,
   1471     centers_init,
   1472     max_iter=self.max_iter,
   1473     verbose=self.verbose,
   1474     tol=self._tol,
   1475     n_threads=self._n_threads,
   1476 )
   1478 # determine if these results are the best so far
   1479 # we chose a new run if it has a better inertia and the clustering is
   1480 # different from the best so far (it's possible that the inertia is
   1481 # slightly better even if the clustering is the same with potentially
   1482 # permuted labels, due to rounding errors)
   1483 if best_inertia is None or (
   1484     inertia < best_inertia
   1485     and not _is_same_clustering(labels, best_labels, self.n_clusters)
   1486 ):

File ~\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:679, in␣
↪_kmeans_single_lloyd(X, sample_weight, centers_init, max_iter, verbose, tol,␣
↪n_threads)
    675 strict_convergence = False
    677 # Threadpoolctl context to limit the number of threads in second level of
    678 # nested parallelism (i.e. BLAS) to avoid oversubscription.
--> 679 with threadpool_limits(limits=1, user_api="blas"):
    680     for i in range(max_iter):
    681         lloyd_iter(
    682             X,
    683             sample_weight,
   (…)
    689             n_threads,
    690         )

File ~\anaconda3\Lib\site-packages\sklearn\utils\fixes.py:139, in␣
↪threadpool_limits(limits, user_api)
    137     return controller.limit(limits=limits, user_api=user_api)
    138 else:
--> 139     return threadpoolctl.threadpool_limits(limits=limits,␣
↪user_api=user_api)
```

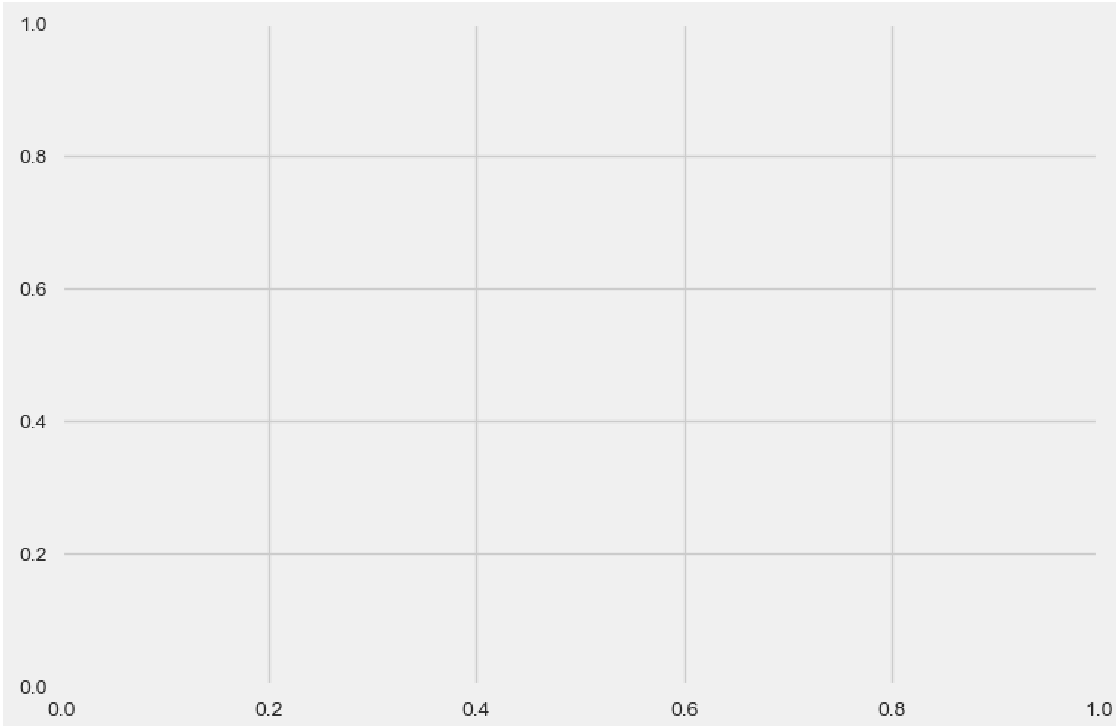
```
File ~\anaconda3\Lib\site-packages\threadpoolctl.py:171, in threadpool_limits.
↪__init__(self, limits, user_api)
    167 def __init__(self, limits=None, user_api=None):
    168     self._limits, self._user_api, self._prefixes = \
    169         self._check_params(limits, user_api)
--> 171     self._original_info = self._set_threadpool_limits()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:268, in threadpool_limits.
↪_set_threadpool_limits(self)
    265 if self._limits is None:
    266     return None
--> 268 modules = _ThreadpoolInfo(prefixes=self._prefixes,
    269                           user_api=self._user_api)
    270 for module in modules:
    271     # self._limits is a dict {key: num_threads} where key is either
    272     # a prefix or a user_api. If a module matches both, the limit
    273     # corresponding to the prefix is chosed.
    274     if module.prefix in self._limits:

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:340, in _ThreadpoolInfo.
↪__init__(self, user_api, prefixes, modules)
    337     self.user_api = [] if user_api is None else user_api
    339     self.modules = []
--> 340     self._load_modules()
    341     self._warn_if_incompatible_openmp()
    342 else:

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:373, in _ThreadpoolInfo.
↪_load_modules(self)
    371     self._find_modules_with_dyld()
    372 elif sys.platform == "win32":
--> 373     self._find_modules_with_enum_process_module_ex()
    374 else:
    375     self._find_modules_with_dl_iterate_phdr()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:485, in _ThreadpoolInfo.
↪_find_modules_with_enum_process_module_ex(self)
    482         filepath = buf.value
    484         # Store the module if it is supported and selected
--> 485         self._make_module_from_path(filepath)
    486 finally:
    487     kernel_32.CloseHandle(h_process)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:515, in _ThreadpoolInfo.
↪_make_module_from_path(self, filepath)
    513 if prefix in self.prefixes or user_api in self.user_api:
    514     module_class = globals()[module_class]
--> 515     module = module_class(filepath, prefix, user_api, internal_api)
```

```
    516     self.modules.append(module)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:606, in _Module.
↪__init__(self, filepath, prefix, user_api, internal_api)
    604 self.internal_api = internal_api
    605 self._dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
--> 606 self.version = self.get_version()
    607 self.num_threads = self.get_num_threads()
    608 self._get_extra_info()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:646, in _OpenBLASModule.
↪get_version(self)
    643 get_config = getattr(self._dynlib, "openblas_get_config",
    644                      lambda: None)
    645 get_config.restype = ctypes.c_char_p
--> 646 config = get_config().split()
    647 if config[0] == b"OpenBLAS":
    648     return config[1].decode("utf-8")

AttributeError: 'NoneType' object has no attribute 'split'
```

[86]:
```
plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[86], line 2
      1 plt.style.use("fivethirtyeight")
----> 2 plt.plot(range(1, 11), sse)
      3 plt.xticks(range(1, 11))
      4 plt.xlabel("Number of Clusters")

File ~\anaconda3\Lib\site-packages\matplotlib\pyplot.py:3578, in plot(scalex,␣
↪scaley, data, *args, **kwargs)
   3570 @_copy_docstring_and_deprecators(Axes.plot)
   3571 def plot(
   3572     *args: float | ArrayLike | str,
   (...)
   3576     **kwargs,
   3577 ) -> list[Line2D]:
-> 3578     return gca().plot(
   3579         *args,
   3580         scalex=scalex,
```

```
   3581         scaley=scaley,
   3582         **({"data": data} if data is not None else {}),
   3583         **kwargs,
   3584     )

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_axes.py:1721, in Axes.plot(self, scalex, scaley, data, *args, **kwargs)
   1478 """
   1479 Plot y versus x as lines and/or markers.
   1480
   (…)
   1718 (``'green'``) or hex strings (``'#008000'``).
   1719 """
   1720 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1721 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1722 for line in lines:
   1723     self.add_line(line)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:303, in _process_plot_var_args.__call__(self, axes, data, *args, **kwargs)
    301     this += args[0],
    302     args = args[1:]
--> 303 yield from self._plot_args(
    304     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:499, in _process_plot_var_args._plot_args(self, axes, tup, kwargs, return_kwargs, ambiguous_fmt_datakey)
    496     axes.yaxis.update_units(y)
    498 if x.shape[0] != y.shape[0]:
--> 499     raise ValueError(f"x and y must have same first dimension, but "
    500                      f"have shapes {x.shape} and {y.shape}")
    501 if x.ndim > 2 or y.ndim > 2:
    502     raise ValueError(f"x and y can be no greater than 2D, but have "
    503                      f"shapes {x.shape} and {y.shape}")

ValueError: x and y must have same first dimension, but have shapes (10,) and (0,)
```



[94]:
```
kl = KneeLocator(
    range(1, 11), sse, curve="convex", direction="decreasing"
)

kl.elbow
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[94], line 1
----> 1 kl = KneeLocator(
      2     range(1, 11), sse, curve="convex", direction="decreasing"
      3 )
      5 kl.elbow

File ~\anaconda3\Lib\site-packages\kneed\knee_locator.py:171, in KneeLocator.
  ↪__init__(self, x, y, S, curve, direction, interp_method, online,␣
  ↪polynomial_degree)
    169 # Step 1: fit a smooth line
    170 if interp_method == "interp1d":
--> 171     uspline = interpolate.interp1d(self.x, self.y)
    172     self.Ds_y = uspline(self.x)
    173 elif interp_method == "polynomial":
```

```
File ~\anaconda3\Lib\site-packages\scipy\interpolate\_interpolate.py:494, in␣
↪interp1d.__init__(self, x, y, kind, axis, copy, bounds_error, fill_value,␣
↪assume_sorted)
    490 def __init__(self, x, y, kind='linear', axis=-1,
    491              copy=True, bounds_error=None, fill_value=np.nan,
    492              assume_sorted=False):
    493     """ Initialize a 1-D linear interpolation class."""
--> 494     _Interpolator1D.__init__(self, x, y, axis=axis)
    496     self.bounds_error = bounds_error  # used by fill_value setter
    497     self.copy = copy

File ~\anaconda3\Lib\site-packages\scipy\interpolate\_polyint.py:56, in␣
↪_Interpolator1D.__init__(self, xi, yi, axis)
     54 self.dtype = None
     55 if yi is not None:
---> 56     self._set_yi(yi, xi=xi, axis=axis)

File ~\anaconda3\Lib\site-packages\scipy\interpolate\_polyint.py:126, in␣
↪_Interpolator1D._set_yi(self, yi, xi, axis)
    124     shape = (1,)
    125 if xi is not None and shape[axis] != len(xi):
--> 126     raise ValueError("x and y arrays must be equal in length along "
    127                      "interpolation axis.")
    129 self._y_axis = (axis % yi.ndim)
    130 self._y_extra_shape = yi.shape[:self._y_axis]+yi.shape[self._y_axis+1:]

ValueError: x and y arrays must be equal in length along interpolation axis.
```

[95]:
```
#Model building using KMeans
kmeans =  KMeans(n_clusters=5, max_iter=600, algorithm = 'auto')
kmeans.fit(df1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[95], line 3
      1 #Model building using KMeans
      2 kmeans =  KMeans(n_clusters=5, max_iter=600, algorithm = 'auto')
----> 3 kmeans.fit(df1)

File ~\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1468, in KMeans.
↪fit(self, X, y, sample_weight)
   1465     print("Initialization complete")
   1467 # run a k-means once
-> 1468 labels, inertia, centers, n_iter_ = kmeans_single(
   1469     X,
   1470     sample_weight,
   1471     centers_init,
```

```
   1472     max_iter=self.max_iter,
   1473     verbose=self.verbose,
   1474     tol=self._tol,
   1475     n_threads=self._n_threads,
   1476 )
   1478 # determine if these results are the best so far
   1479 # we chose a new run if it has a better inertia and the clustering is
   1480 # different from the best so far (it's possible that the inertia is
   1481 # slightly better even if the clustering is the same with potentially
   1482 # permuted labels, due to rounding errors)
   1483 if best_inertia is None or (
   1484     inertia < best_inertia
   1485     and not _is_same_clustering(labels, best_labels, self.n_clusters)
   1486 ):

File ~\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:679, in _kmeans_single_lloyd(X, sample_weight, centers_init, max_iter, verbose, tol, n_threads)
    675 strict_convergence = False
    677 # Threadpoolctl context to limit the number of threads in second level of
    678 # nested parallelism (i.e. BLAS) to avoid oversubscription.
--> 679 with threadpool_limits(limits=1, user_api="blas"):
    680     for i in range(max_iter):
    681         lloyd_iter(
    682             X,
    683             sample_weight,
   (…)
    689             n_threads,
    690         )

File ~\anaconda3\Lib\site-packages\sklearn\utils\fixes.py:139, in threadpool_limits(limits, user_api)
    137     return controller.limit(limits=limits, user_api=user_api)
    138 else:
--> 139     return threadpoolctl.threadpool_limits(limits=limits, user_api=user_api)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:171, in threadpool_limits.__init__(self, limits, user_api)
    167 def __init__(self, limits=None, user_api=None):
    168     self._limits, self._user_api, self._prefixes = \
    169         self._check_params(limits, user_api)
--> 171     self._original_info = self._set_threadpool_limits()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:268, in threadpool_limits._set_threadpool_limits(self)
    265 if self._limits is None:
    266     return None
```

```
--> 268 modules = _ThreadpoolInfo(prefixes=self._prefixes,
    269                           user_api=self._user_api)
    270 for module in modules:
    271     # self._limits is a dict {key: num_threads} where key is either
    272     # a prefix or a user_api. If a module matches both, the limit
    273     # corresponding to the prefix is chosed.
    274     if module.prefix in self._limits:

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:340, in _ThreadpoolInfo.__init__(self, user_api, prefixes, modules)
    337     self.user_api = [] if user_api is None else user_api
    339     self.modules = []
--> 340     self._load_modules()
    341     self._warn_if_incompatible_openmp()
    342 else:

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:373, in _ThreadpoolInfo._load_modules(self)
    371     self._find_modules_with_dyld()
    372 elif sys.platform == "win32":
--> 373     self._find_modules_with_enum_process_module_ex()
    374 else:
    375     self._find_modules_with_dl_iterate_phdr()

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:485, in _ThreadpoolInfo._find_modules_with_enum_process_module_ex(self)
    482         filepath = buf.value
    484         # Store the module if it is supported and selected
--> 485         self._make_module_from_path(filepath)
    486 finally:
    487     kernel_32.CloseHandle(h_process)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:515, in _ThreadpoolInfo._make_module_from_path(self, filepath)
    513 if prefix in self.prefixes or user_api in self.user_api:
    514     module_class = globals()[module_class]
--> 515     module = module_class(filepath, prefix, user_api, internal_api)
    516     self.modules.append(module)

File ~\anaconda3\Lib\site-packages\threadpoolctl.py:606, in _Module.__init__(self, filepath, prefix, user_api, internal_api)
    604 self.internal_api = internal_api
    605 self._dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
--> 606 self.version = self.get_version()
    607 self.num_threads = self.get_num_threads()
    608 self._get_extra_info()
```

```
File ~\anaconda3\Lib\site-packages\threadpoolctl.py:646, in _OpenBLASModule.
↪get_version(self)
    643 get_config = getattr(self._dynlib, "openblas_get_config",
    644                      lambda: None)
    645 get_config.restype = ctypes.c_char_p
--> 646 config = get_config().split()
    647 if config[0] == b"OpenBLAS":
    648     return config[1].decode("utf-8")

AttributeError: 'NoneType' object has no attribute 'split'
```

[96]:

```
#Use scatterplot to show clusters via PCA
plt.scatter(df2[:, 0], df2[:, 1],
            c= kmeans.labels_.astype(float), edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('Spectral', 10))
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.title('Diamonds clusters using KMEANS')
plt.colorbar();
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[96], line 3
      1 #Use scatterplot to show clusters via PCA
      2 plt.scatter(df2[:, 0], df2[:, 1],
----> 3             c= kmeans.labels_.astype(float), edgecolor='none', alpha=0.5,
      4             cmap=plt.cm.get_cmap('Spectral', 10))
      5 plt.xlabel('PCA1')
      6 plt.ylabel('PCA2')

AttributeError: 'KMeans' object has no attribute 'labels_'
```

[98]:

```
df1["cluster"] = kmeans.labels_.astype(float)
df1
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[98], line 1
----> 1 df1["cluster"] = kmeans.labels_.astype(float)
      2 df1

AttributeError: 'KMeans' object has no attribute 'labels_'
```

[99]:

```
# Number of clusters in labels, ignoring noise if present.
labels_kmeans = kmeans.labels_
```

```
n_clusters_ = len(set(labels_kmeans)) - (1 if -1 in labels_kmeans else 0)
n_noise_ = list(labels_kmeans).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[99], line 2
      1 # Number of clusters in labels, ignoring noise if present.
----> 2 labels_kmeans = kmeans.labels_
      3 n_clusters_ = len(set(labels_kmeans)) - (1 if -1 in labels_kmeans else 0)
      4 n_noise_ = list(labels_kmeans).count(-1)

AttributeError: 'KMeans' object has no attribute 'labels_'
```

[101]:
```
for c in df1:
    grid= sns.FacetGrid(df1, col='cluster')
    grid.map(plt.hist, c)
kmeans=KMeans(5)
kmeans.fit_predict(df1)
y_pred = kmeans.predict(df1)
#preds = kmeans.labels_
data = pd.DataFrame(df)
data['cluster'] = y_pred
data.head(10)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3791, in Index.
 ↪get_loc(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'cluster'
```

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

KeyError                                  Traceback (most recent call last)
Cell In[101], line 2
      1 for c in df1:
----> 2     grid= sns.FacetGrid(df1, col='cluster')
      3     grid.map(plt.hist, c)
      4 kmeans=KMeans(5)

File ~\anaconda3\Lib\site-packages\seaborn\axisgrid.py:396, in FacetGrid.
 ↪__init__(self, data, row, col, hue, col_wrap, sharex, sharey, height, aspect,␣
 ↪palette, row_order, col_order, hue_order, hue_kws, dropna, legend_out,␣
 ↪despine, margin_titles, xlim, ylim, subplot_kws, gridspec_kws)
    394     col_names = []
    395 else:
--> 396     col_names = categorical_order(data[col], col_order)
    398 # Additional dict of kwarg -> list of values for mapping the hue var
    399 hue_kws = hue_kws if hue_kws is not None else {}

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:3893, in DataFrame.
 ↪__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3798, in Index.
 ↪get_loc(self, key)
   3793     if isinstance(casted_key, slice) or (
   3794         isinstance(casted_key, abc.Iterable)
   3795         and any(isinstance(x, slice) for x in casted_key)
   3796     ):
   3797         raise InvalidIndexError(key)
-> 3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-raise
   3802     #  the TypeError.
   3803     self._check_indexing_error(key)

KeyError: 'cluster'
```

[102]:
```
#Use scatterplot to show clusters via KMEANS
plt.figure(figsize=(8,8))
sns.scatterplot(x='price',y='carat',hue="cluster",data=data,palette='Paired_r')
```

```
plt.title('Diamonds clusters using KMEANS')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[102], line 3
      1 #Use scatterplot to show clusters via KMEANS
      2 plt.figure(figsize=(8,8))
----> 3 sns.
 ↪scatterplot(x='price',y='carat',hue="cluster",data=data,palette='Paired_r')
      4 plt.title('Diamonds clusters using KMEANS')
      5 plt.show()

NameError: name 'data' is not defined
```

```
<Figure size 800x800 with 0 Axes>
```

6 Using KNN Algorithm to predict if a person will have diabetes or not

[103]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[105]:
```
data = pd.read_csv(r"C:\Users\djerb\Downloads\Diabetes\Diabetes\diabetes.csv")
data.head()
```

[105]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
```

```
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[106]:
```
zero_not_accepted = ['Glucose','BloodPressure','SkinThickness','BMI','Insulin']
# for col in zero_not_accepted:
#     for i in data[col]:
#         if i==0:
#             colSum = sum(data[col])
#             meanCol=colSum/len(data[col])
#             data[col]=meanCol

for col in zero_not_accepted:
    data[col]= data[col].replace(0,np.NaN)
    mean = int(data[col].mean(skipna=True))
    data[col] = data[col].replace(np.NaN,mean)
```

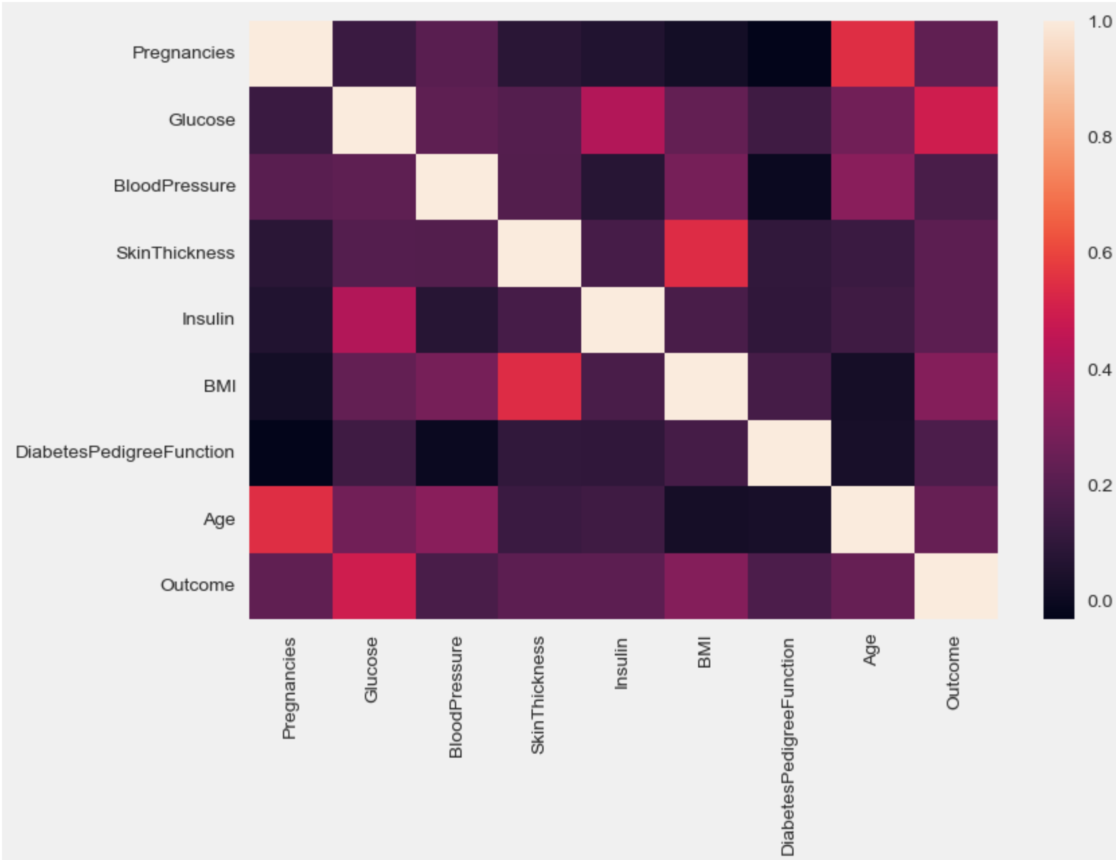
[107]:
```
X = data.iloc[:,0:8]
```

[108]:
```
y = data.iloc[:,8]
```

6.1 Explorning data to know relation before processing

[109]:
```
sns.heatmap(data.corr())
```

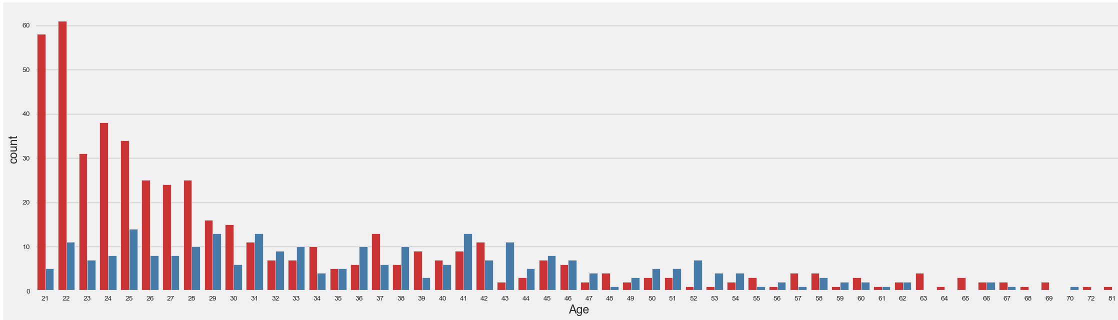
[109]:
```
<Axes: >
```



[110]:
```
plt.figure(figsize=(25,7))
sns.countplot(x='Age',hue='Outcome',data=data,palette='Set1')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[110], line 2
      1 plt.figure(figsize=(25,7))
----> 2 sns.countplot(x='Age',hue='Outcome',data=data,palette='Set1')

File ~\anaconda3\Lib\site-packages\seaborn\categorical.py:2955, in␣
↪countplot(data, x, y, hue, order, hue_order, orient, color, palette,␣
↪saturation, width, dodge, ax, **kwargs)
   2952 if ax is None:
   2953     ax = plt.gca()
-> 2955 plotter.plot(ax, kwargs)
   2956 return ax

File ~\anaconda3\Lib\site-packages\seaborn\categorical.py:1587, in _BarPlotter.
↪plot(self, ax, bar_kws)
   1585 """Make the plot."""
```

```
   1586 self.draw_bars(ax, bar_kws)
-> 1587 self.annotate_axes(ax)
   1588 if self.orient == "h":
   1589     ax.invert_yaxis()

File ~\anaconda3\Lib\site-packages\seaborn\categorical.py:767, in␣
↪_CategoricalPlotter.annotate_axes(self, ax)
    764     ax.set_ylim(-.5, len(self.plot_data) - .5, auto=None)
    766 if self.hue_names is not None:
--> 767     ax.legend(loc="best", title=self.hue_title)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_axes.py:322, in Axes.
↪legend(self, *args, **kwargs)
    204 @_docstring.dedent_interpd
    205 def legend(self, *args, **kwargs):
    206     """
    207     Place a legend on the Axes.
    208
   (…)
    320     .. plot:: gallery/text_labels_and_annotations/legend.py
    321     """
--> 322     handles, labels, kwargs = mlegend._parse_legend_args([self], *args,␣
↪**kwargs)
    323     self.legend_ = mlegend.Legend(self, handles, labels, **kwargs)
    324     self.legend_._remove_method = self._remove_legend

File ~\anaconda3\Lib\site-packages\matplotlib\legend.py:1361, in␣
↪_parse_legend_args(axs, handles, labels, *args, **kwargs)
   1357     handles = [handle for handle, label
   1358                in zip(_get_legend_handles(axs, handlers), labels)]
   1360 elif len(args) == 0:  # 0 args: automatically detect labels and handles
-> 1361     handles, labels = _get_legend_handles_labels(axs, handlers)
   1362     if not handles:
   1363         log.warning(
   1364             "No artists with labels found to put in legend.  Note that
   1365             "artists whose label start with an underscore are ignored "
   1366             "when legend() is called with no argument.")

File ~\anaconda3\Lib\site-packages\matplotlib\legend.py:1291, in␣
↪_get_legend_handles_labels(axs, legend_handler_map)
   1289 for handle in _get_legend_handles(axs, legend_handler_map):
   1290     label = handle.get_label()
-> 1291     if label and not label.startswith('_'):
   1292         handles.append(handle)
   1293         labels.append(label)

AttributeError: 'numpy.int64' object has no attribute 'startswith'
```



[111]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=0)
```

[112]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[113]:
```
classifier = KNeighborsClassifier(n_neighbors=11,p=2,metric='euclidean')
```

[114]:
```
classifier.fit(X_train,y_train)
```

[114]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=11)
```

[115]:
```
y_pred = classifier.predict(X_test)
```

[116]:
```
conf_matrix = confusion_matrix(y_test,y_pred)
print(conf_matrix)
print(f1_score(y_test,y_pred))
```

```
[[94 13]
 [15 32]]
0.6956521739130436
```

[117]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.8181818181818182
```

[ ]:
```

```