In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:
```
cars_data=pd.read_csv('cars_sampled.csv')
```

In [3]:
```
cars=cars_data.copy()
```

In [4]:
```
cars.info()
#1.it gives you total data and attributes  name and data
#type of each attribute
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50001 entries, 0 to 50000
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   dateCrawled          50001 non-null  object
 1   name                 50001 non-null  object
 2   seller               50001 non-null  object
 3   offerType            50001 non-null  object
 4   price                50001 non-null  int64 
 5   abtest               50001 non-null  object
 6   vehicleType          44813 non-null  object
 7   yearOfRegistration   50001 non-null  int64 
 8   gearbox              47177 non-null  object
 9   powerPS              50001 non-null  int64 
 10  model                47243 non-null  object
 11  kilometer            50001 non-null  int64 
 12  monthOfRegistration  50001 non-null  int64 
 13  fuelType             45498 non-null  object
 14  brand                50001 non-null  object
 15  notRepairedDamage    40285 non-null  object
 16  dateCreated          50001 non-null  object
 17  postalCode           50001 non-null  int64 
 18  lastSeen             50001 non-null  object
dtypes: int64(6), object(13)
memory usage: 7.2+ MB
```

In [5]:
```
cars.describe()#it will give the summary of numerical data
```

Out[5]:

	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	posta
count	5.000100e+04	50001.000000	50001.000000	50001.000000	50001.000000	50001.0
mean	6.559865e+03	2005.544229	116.496130	125613.687726	5.743725	50775.2
std	8.581847e+04	122.991832	230.567709	40205.234483	3.711345	25743.7
min	0.000000e+00	1000.000000	0.000000	5000.000000	0.000000	1067.0
25%	1.150000e+03	1999.000000	69.000000	125000.000000	3.000000	30559.0
50%	2.950000e+03	2003.000000	105.000000	150000.000000	6.000000	49504.0
75%	7.190000e+03	2008.000000	150.000000	150000.000000	9.000000	71404.0
max	1.234568e+07	9999.000000	19312.000000	150000.000000	12.000000	99998.0

In [6]:
```
pd.set_option('display.max_columns',500)#to view the set of column of 500 rows bu
cars.describe()
```

Out[6]:

	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	posta
count	5.000100e+04	50001.000000	50001.000000	50001.000000	50001.000000	50001.0
mean	6.559865e+03	2005.544229	116.496130	125613.687726	5.743725	50775.2
std	8.581847e+04	122.991832	230.567709	40205.234483	3.711345	25743.7
min	0.000000e+00	1000.000000	0.000000	5000.000000	0.000000	1067.0
25%	1.150000e+03	1999.000000	69.000000	125000.000000	3.000000	30559.0
50%	2.950000e+03	2003.000000	105.000000	150000.000000	6.000000	49504.0
75%	7.190000e+03	2008.000000	150.000000	150000.000000	9.000000	71404.0
max	1.234568e+07	9999.000000	19312.000000	150000.000000	12.000000	99998.0

In [7]:
```
cars.columns
#to display all the columns
```

Out[7]:
```
Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
       'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model',
       'kilometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage', 'dateCreated', 'postalCode', 'lastSeen'],
      dtype='object')
```

In [8]:
```
col=['name','postalCode','dateCrawled','dateCreated','lastSeen']
cars=cars.drop(columns=col,axis=1)#here we are droping the unwanted  columns
```

In [9]:
```
cars.drop_duplicates(keep='first',inplace=True)
#470 duplicate records has been removed only the first code line of dupllicates k
```

In [10]:
```
cars.isnull().sum()
#the missing values are shown here
```

Out[10]:
```
seller                   0
offerType                0
price                    0
abtest                   0
vehicleType           5152
yearOfRegistration       0
gearbox               2765
powerPS                  0
model                 2730
kilometer                0
monthOfRegistration      0
fuelType              4467
brand                    0
notRepairedDamage     9640
dtype: int64
```

In [11]:
```
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
print(yearwise_count)#it will calculate the total number of cars produced in a ye
#there is so much unwanted data which are very past years and some are future pre
```

```
1000     6
1255     1
1500     2
1910    14
1928     1
        ..
7500     1
7800     1
8500     1
8888     2
9999     7
Name: yearOfRegistration, Length: 97, dtype: int64
```

In [12]:
```
#yearwise_count=cars['yearOfRegistration'].value_counts()
#print(yearwise_count)
```

In [13]:
```
sum(cars['yearOfRegistration']>2018)#there are 28 cars below which are produced b
```

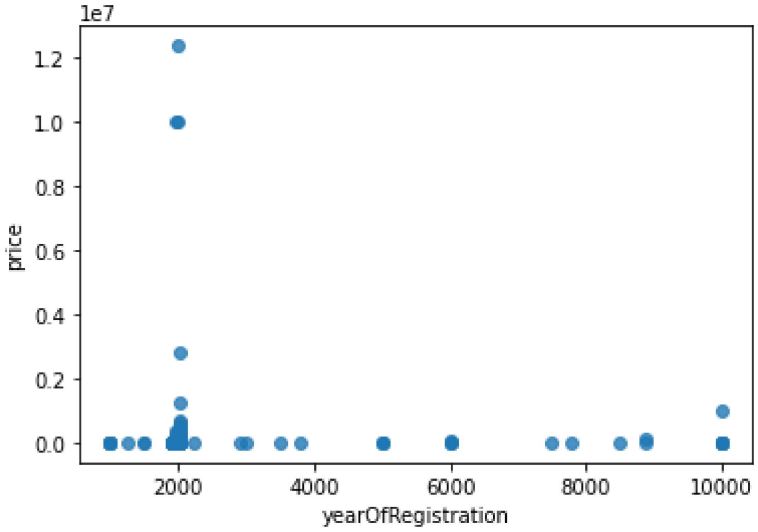
Out[13]:
```
26
```

In [14]:
```
sum(cars['yearOfRegistration']<1950)#there are 38 cars which is above 38 cars
```

Out[14]:
```
38
```

In [15]:
```
sns.regplot(x='yearOfRegistration',y='price',scatter=True,fit_reg=False,data=cars
#we do not get any thing from this data
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26bff4a30>
```

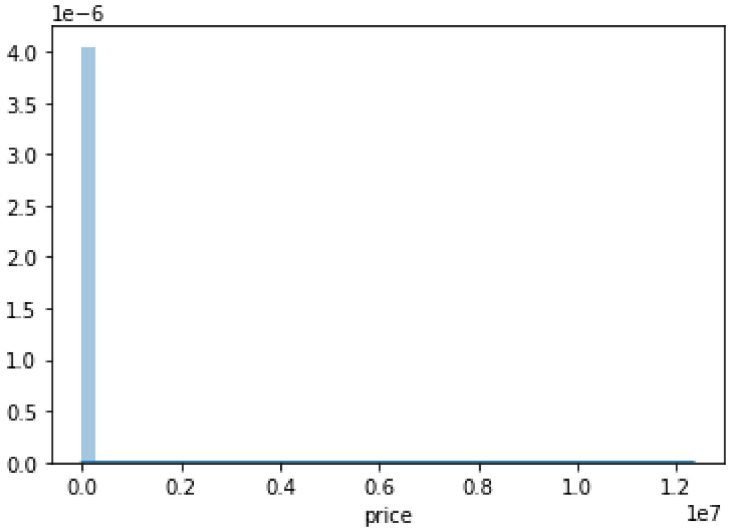


In [16]:
```
price_count=cars['price'].value_counts().sort_index()
print(price_count)
```

```
0           1415
1            172
2              1
3              1
5              4
            ...
1250000        1
2795000        1
9999999        1
10010011       1
12345678       1
Name: price, Length: 2393, dtype: int64
```

In [17]:
```
sns.distplot(cars['price'])
#there is so much 0 category where we have to work on
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26cf96f10>
```

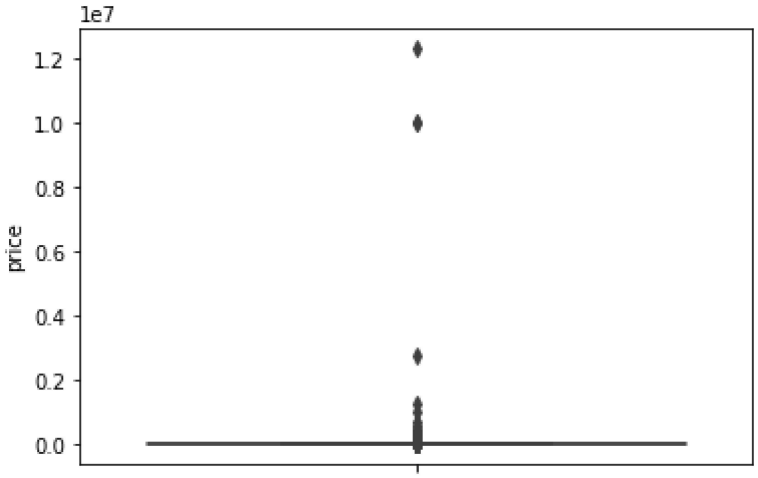


In [18]:
```
cars['price'].describe()
#there is so much difference  between mean and median
```

Out[18]:
```
count    4.953100e+04
mean     6.567220e+03
std      8.622238e+04
min      0.000000e+00
25%      1.150000e+03
50%      2.950000e+03
75%      7.100000e+03
max      1.234568e+07
Name: price, dtype: float64
```

In [19]:
```
sns.boxplot(y=cars['price'])
#there are many outliers shown in figure even we cant see the box and they were v
#able to see the behaviour of variable
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26d09f910>
```



In [20]:
```
sum(cars['price']>150000)#printing the sum greater 150000
```

Out[20]:
```
34
```

In [21]:
```
sum(cars['price']<100)#printing the sum less than 100
```

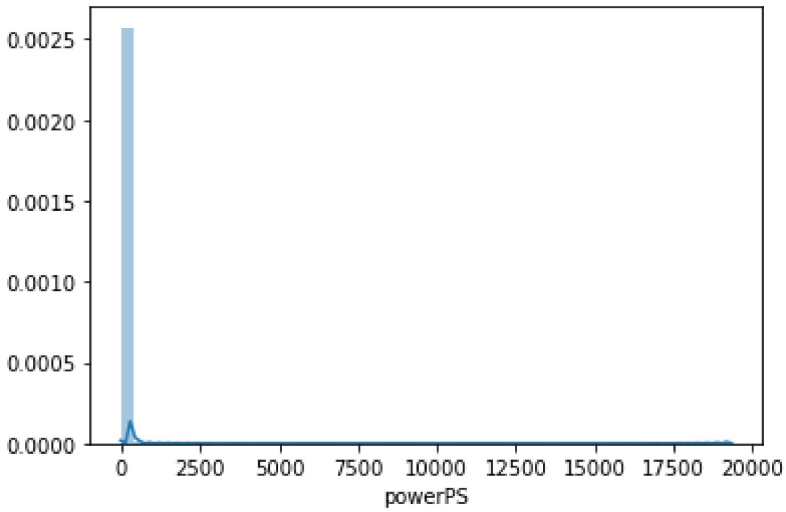
Out[21]:
```
1748
```

In [22]:
```
power_count=cars['powerPS'].value_counts().sort_index()
print(power_count)
```

```
0        5533
1           3
2           2
3           2
4           4
         ...
15033       1
16011       1
16312       1
19211       1
19312       1
Name: powerPS, Length: 460, dtype: int64
```

In [23]:
```
sns.distplot(cars['powerPS'])
#yes there is skewness in 0
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26d0e04c0>
```

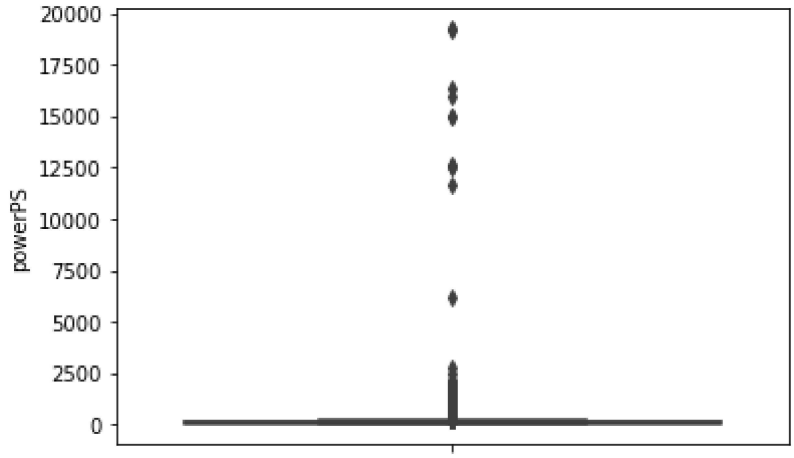


In [24]:
```
cars['powerPS'].describe()
#the mean and median is far and std is very high
```

Out[24]:
```
count    49531.000000
mean       116.500515
std        231.535603
min          0.000000
25%         69.000000
50%        105.000000
75%        150.000000
max      19312.000000
Name: powerPS, dtype: float64
```

In [25]:
```
sns.boxplot(y=cars['powerPS'])
#there is so much outliers we can't see the figure
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26d1f26d0>
```



In [26]:
```
sum(cars['powerPS']>500)
```

Out[26]:
```
115
```

In [27]:
```
sum(cars['powerPS']<10)
```

Out[27]:
```
5565
```

Workking with ranges

In [28]:
```
#here the range of registration is 2018 to 1950
#here the price is 100 and 150000
#here the powerPS is 10 and 500
```

In [29]:
```
cars.columns
```

Out[29]:
```
Index(['seller', 'offerType', 'price', 'abtest', 'vehicleType',
       'yearOfRegistration', 'gearbox', 'powerPS', 'model', 'kilometer',
       'monthOfRegistration', 'fuelType', 'brand', 'notRepairedDamage'],
      dtype='object')
```

In [30]:
```
cars=cars[(cars.yearOfRegistration<=2018)&(cars.yearOfRegistration>=1950)&(cars.p
#here the range of registration is 2018 to 1950
#here the price is 100 and 150000
#here the powerPS is 10 and 500
print(cars)
```

```
           seller offerType  price   abtest    vehicleType  \
0         private     offer   4450     test      limousine
1         private     offer  13299  control            suv
2         private     offer   3200     test            bus
3         private     offer   4500  control      small car
5         private     offer    988     test      limousine
...           ...       ...    ...      ...            ...
49991     private     offer  10900     test      limousine
49992     private     offer    790     test      limousine
49993     private     offer    830     test      small car
49995     private     offer   2290     test  station wagon
50000  commercial     offer   1100     test      small car

       yearOfRegistration gearbox  powerPS     model  kilometer  \
0                    2003  manual      150       3er     150000
1                    2005  manual      163  xc_reihe     150000
2                    2003  manual      101    touran     150000
3                    2006  manual       86     ibiza      60000
5                    1995  manual       90    passat     150000
...                   ...     ...      ...       ...        ...
49991                2004  manual      231   x_reihe     150000
49992                1998  manual       82     astra     150000
49993                1999  manual       60      clio     125000
49995                2001  manual      101     astra     150000
50000                2006  manual       38     matiz     150000

       monthOfRegistration fuelType       brand notRepairedDamage
0                        3   diesel         bmw               NaN
1                        6   diesel       volvo                no
2                       11   diesel  volkswagen               NaN
3                       12   petrol        seat                no
5                        2   petrol  volkswagen                no
...                    ...      ...         ...               ...
49991                    8   petrol         bmw                no
49992                    9   diesel        opel                no
49993                    8   petrol     renault                no
49995                    3   petrol        opel                no
50000                   10   petrol   chevrolet                no

[38760 rows x 14 columns]
```

In [31]:
```
#we are simplifying the variable year of registration and month of registration y
#THAT VALUE
cars['monthOfRegistration']/=12
```

In [32]:
```
#here we are creating the new variable cars age
#we are getting the cars age which is older than 2018
cars['Age']=(2018-cars['yearOfRegistration'])
```

In [33]:
```
#here we are adding month of registration in cars Age the decimal value will be s
cars['Age']=(cars['Age']+cars['monthOfRegistration'])
print(cars['Age'])#here the decimal values tell about the month
```

```
0        15.250000
1        13.500000
2        15.916667
3        13.000000
5        23.166667
           ...
49991    14.666667
49992    20.750000
49993    19.666667
49995    17.250000
50000    12.833333
Name: Age, Length: 38760, dtype: float64
```

In [34]:
```
cars['Age']=round(cars['Age'],2)
#here we are round of the value upto 2 decimal point
print(cars['Age'])
```

```
0        15.25
1        13.50
2        15.92
3        13.00
5        23.17
         ...
49991    14.67
49992    20.75
49993    19.67
49995    17.25
50000    12.83
Name: Age, Length: 38760, dtype: float64
```

In [35]:
```
cars['Age'].describe()
#here the data is not very skewed
```

Out[35]:
```
count    38760.000000
mean        15.448101
std          6.732632
min          0.000000
25%         11.420000
50%         15.500000
75%         19.500000
max         67.250000
Name: Age, dtype: float64
```

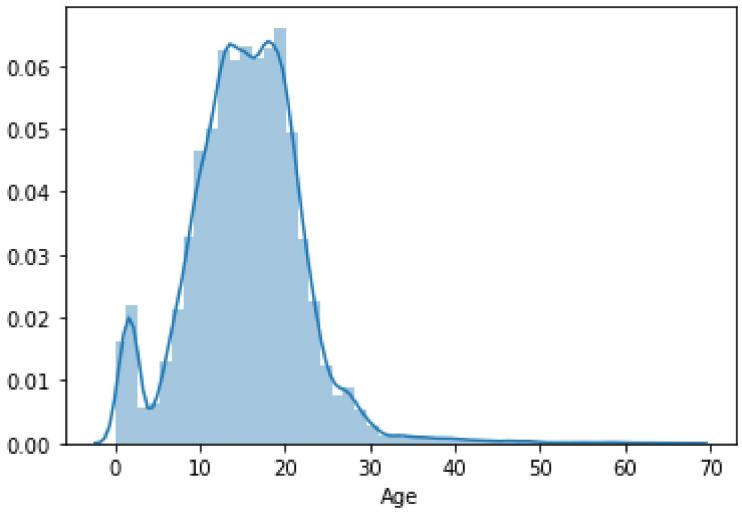
In [36]:
```
cars=cars.drop(columns=['yearOfRegistration','monthOfRegistration'],axis=1)
#here we drop the whole these two columns
```

Visualising age

Age

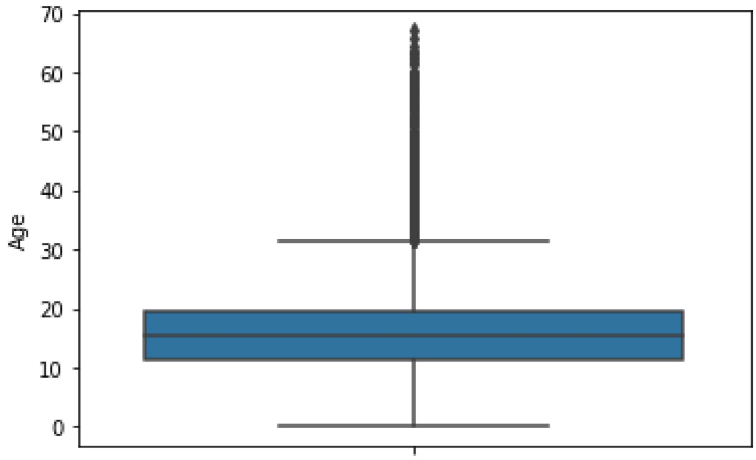
In [37]:
```
sns.distplot(cars['Age'])
```

Out[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26be904c0>
```



In [38]:
```
sns.boxplot(y=cars['Age'])
#here we can see that this is not extremely skewed because we can see the box cle
```

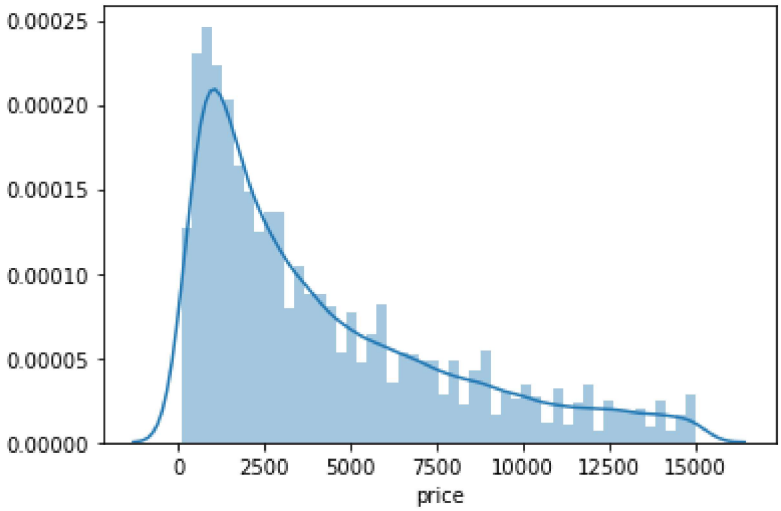
Out[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26bdf5340>
```



Price

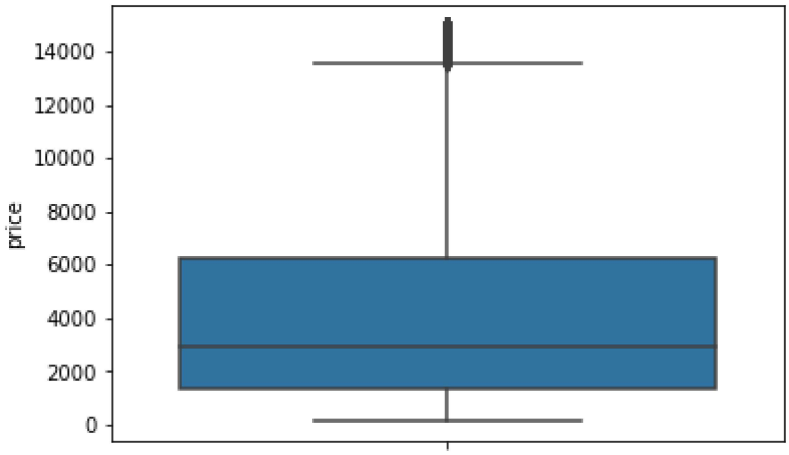
In [39]:
```
sns.distplot(cars['price'])
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26bdea6d0>
```



In [40]:
```
sns.boxplot(y=cars['price'])
#it is better we can see viscous directly
```

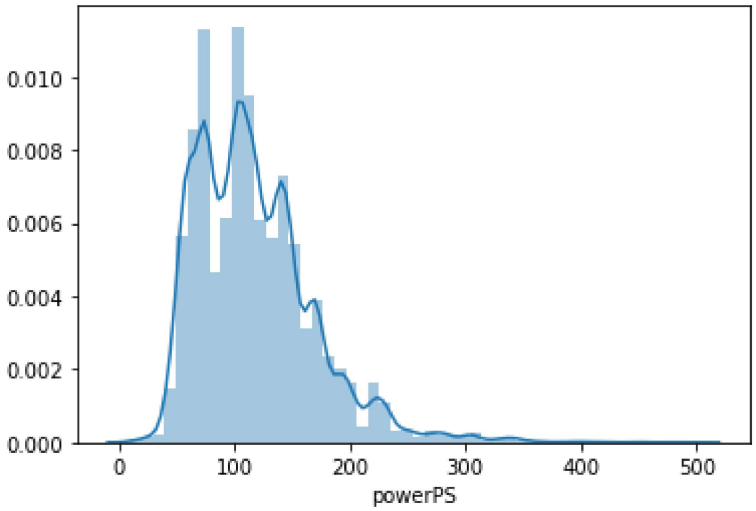
Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26cbf59d0>
```



powerPS

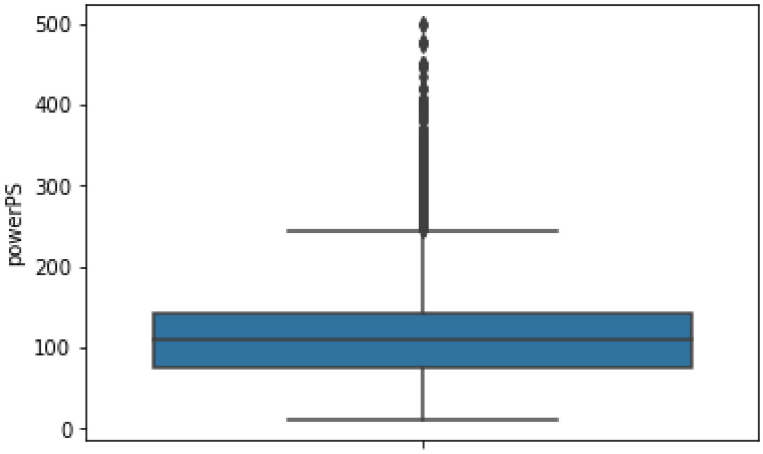
In [41]:
```
sns.distplot(cars['powerPS'])
```

Out[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26d080a30>
```



In [42]:
```
sns.boxplot(y=cars['powerPS'])
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26d37bc40>
```

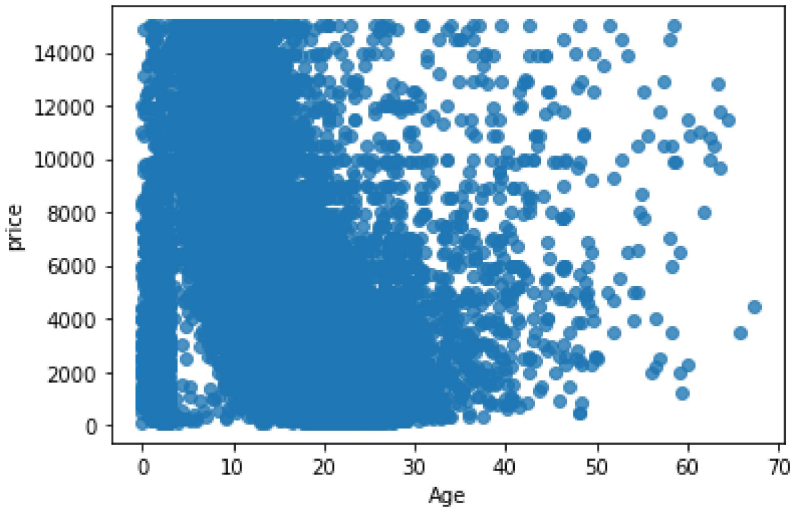


Visualising the parameter after narrowing work space

Age vs price

In [43]:
```
#now we can see the effect of age on price earlier we cant see it
sns.regplot(x='Age',y='price',scatter=True,fit_reg=False,data=cars)
#cars price are higher are newer
#with increase in age price decreases
#some price are higher even there their age are also higher
```

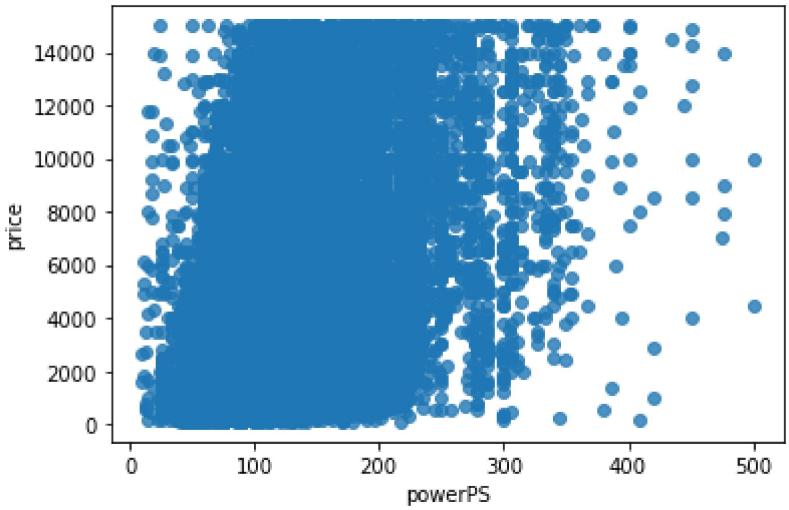
Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26d3c7ac0>
```



powerPS vs price

In [44]:
```
sns.regplot(x='powerPS',y='price',scatter=True,fit_reg=False,data=cars)
#increse in power price also increses
```

Out[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e3faa30>
```



In [45]:
```
#we are checking the other variables on price like categirical variable
```

Seller

In [46]:
```
#we are checking how many rows are present are under columns
cars['seller'].value_counts()
```

Out[46]:
```
private       38759
commercial        1
Name: seller, dtype: int64
```

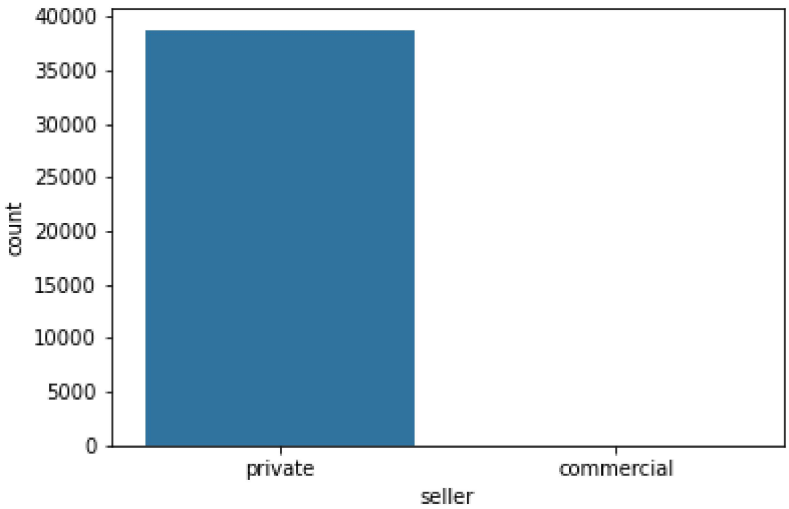
In [47]:
```
pd.crosstab(cars['seller'],columns='count',normalize=True)
#here we are checking the share of each when variable is one
```

Out[47]:

col_0	count
seller	
commercial	0.000026
private	0.999974

In [48]:
```
sns.countplot(x='seller',data=cars)
#here onle one value are present under columns hence this is insignificant colum
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e44c8e0>
```



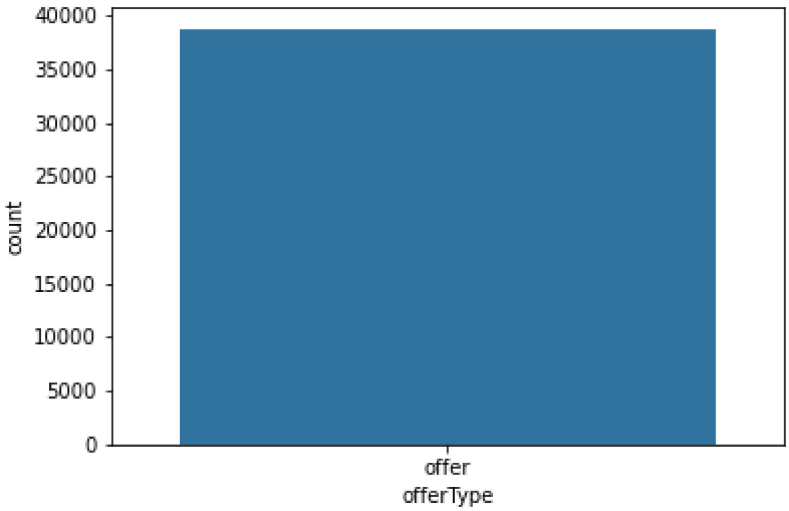
offertype

In [49]:
```
cars['offerType'].value_counts()
```

Out[49]:
```
offer    38760
Name: offerType, dtype: int64
```

In [50]:
```
sns.countplot(x='offerType',data=cars)
#only offer is present so this data is insignificant
```

Out[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e4b9b50>
```



abtest

In [51]:
```
cars['abtest'].value_counts()
```

Out[51]:
```
test       20011
control    18749
Name: abtest, dtype: int64
```

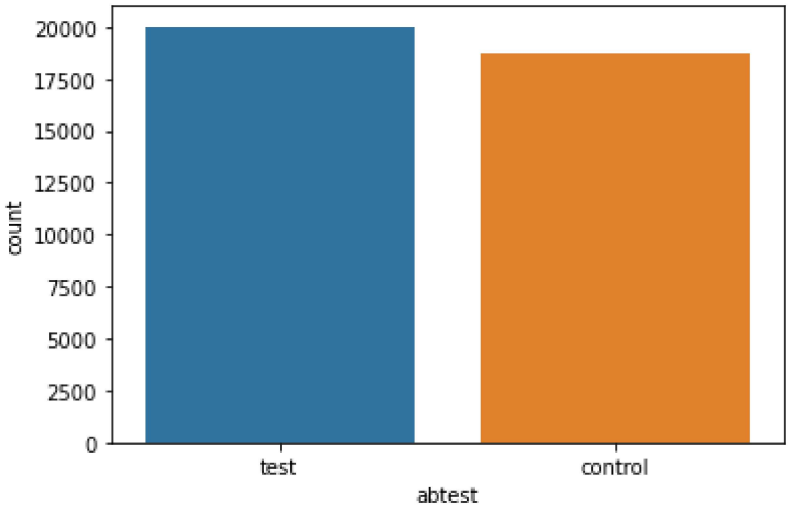
In [52]:
```
pd.crosstab(cars['abtest'],columns='count',normalize=True)
```

Out[52]:

col_0	count
abtest	
control	0.48372
test	0.51628

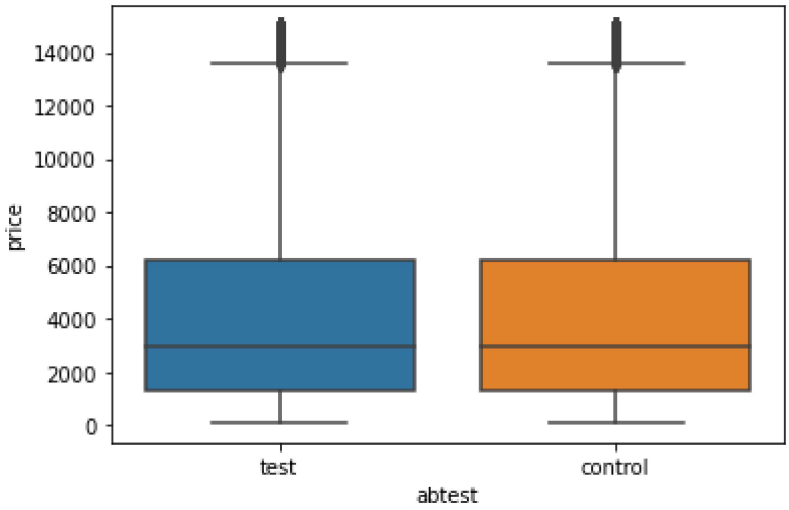
In [53]:
```
sns.countplot(x='abtest',data=cars)
#equally distributed
```

Out[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e4fa280>
```



In [54]:
```
sns.boxplot(x='abtest',y='price',data=cars)
#they have the same effect on price we cant get any significance relation of test
#they have the same median on price so we cant get any relevant option hence it i
```

Out[54]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e55cc10>
```



Vehicle Type

In [55]:
```
cars['vehicleType'].value_counts()
```

Out[55]:
```
limousine        10650
small car         9189
station wagon     7375
bus               3238
cabrio            2239
coupe             1708
suv               1223
others             317
Name: vehicleType, dtype: int64
```

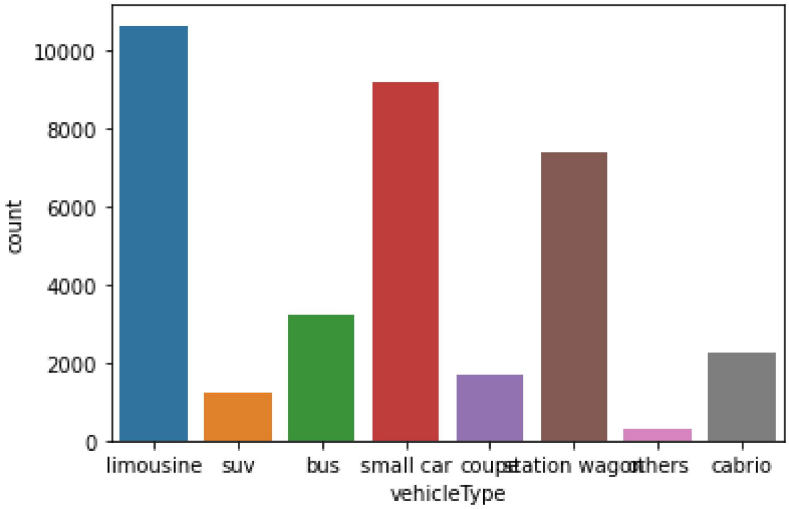
In [56]:
```
#for proportion based count
pd.crosstab(cars['vehicleType'],columns='count',normalize=True)
```

Out[56]:

col_0	count
vehicleType	
bus	0.090097
cabrio	0.062300
coupe	0.047525
limousine	0.296335
others	0.008821
small car	0.255683
station wagon	0.205209
suv	0.034030

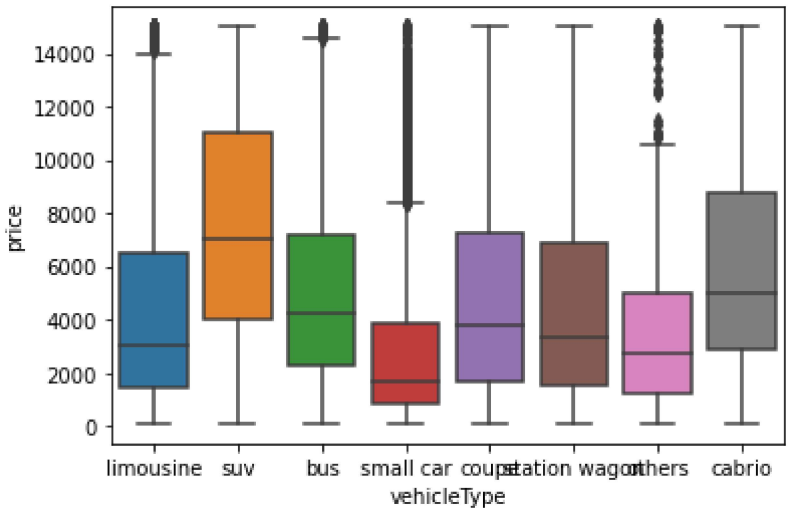
In [57]:
```
sns.countplot(x='vehicleType',data=cars)
#here we can see the contribution of each in vechicle type
```

Out[57]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e5b6610>
```



In [58]:
```
sns.boxplot(x='vehicleType',y='price',data=cars)
#for different data there are different price ranges for each category
#vechicle type affect the prize so we are retain it in our model
```

Out[58]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e6362b0>
```



Gearbox

In [59]:
```
cars['gearbox'].value_counts()
```

Out[59]:
```
manual       30840
automatic     7152
Name: gearbox, dtype: int64
```

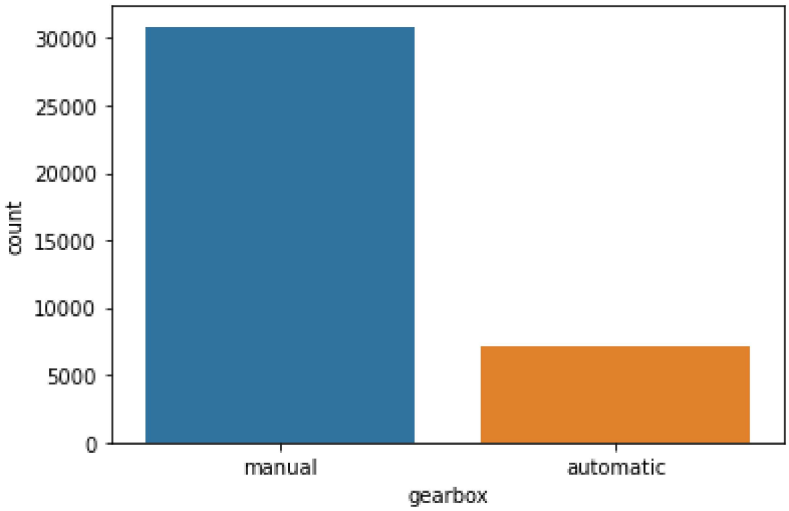
In [60]:
```
pd.crosstab(cars['gearbox'],columns='count',normalize=True)
```

Out[60]:

col_0	count
gearbox	
automatic	0.18825
manual	0.81175

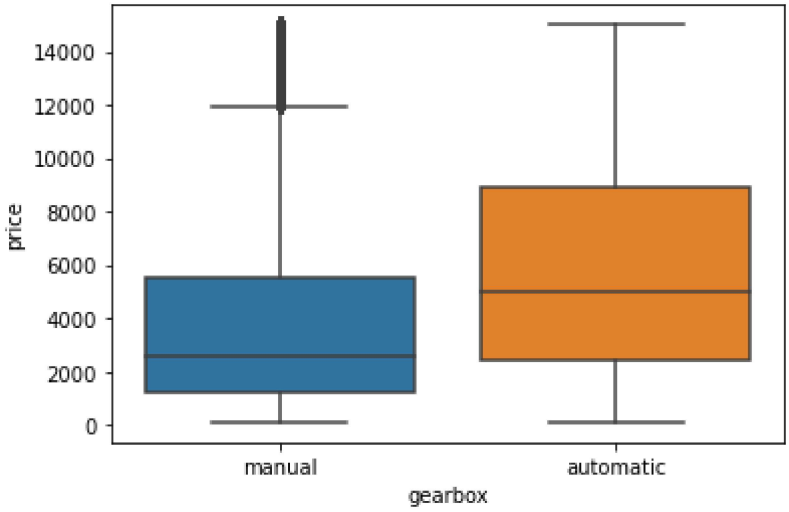
In [61]:
```
sns.countplot(x='gearbox',data=cars)
#manual gearbox are more frequent
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1d26e723190>



In [62]:
```
sns.boxplot(x='gearbox',y='price',data=cars)
#manual price is lower as compare to atomatic
#gearbox affectt] price we can retain it in our final module
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1d26e76f2e0>



Variable model

In [63]:
```
cars['model'].value_counts().sort_index()
```

Out[63]:
```
100         47
145          4
147         58
156         65
159         19
           ...
yaris      132
yeti        17
ypsilon     22
z_reihe     93
zafira     324
Name: model, Length: 243, dtype: int64
```

In [64]:
```
pd.crosstab(cars['model'],columns='count',normalize=True)
```

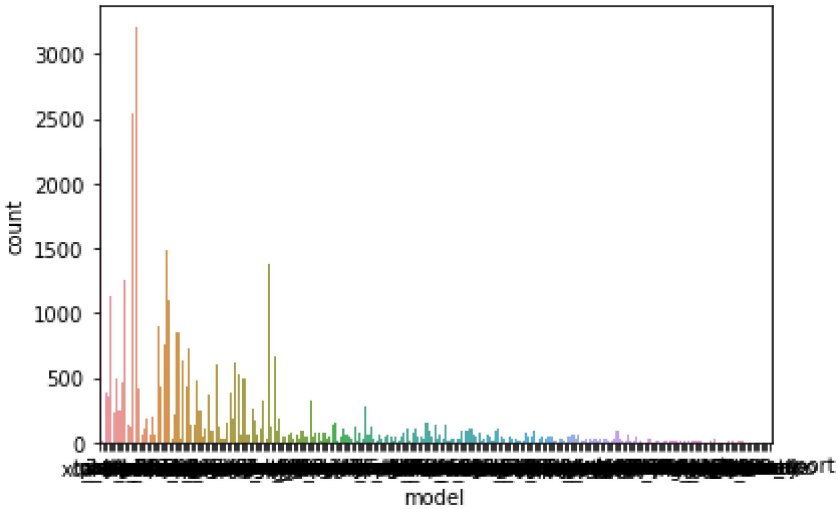
Out[64]:

col_0	count
model	
100	0.001263
145	0.000107
147	0.001559
156	0.001747
159	0.000511
...	...
yaris	0.003547
yeti	0.000457
ypsilon	0.000591
z_reihe	0.002499
zafira	0.008707

243 rows × 1 columns

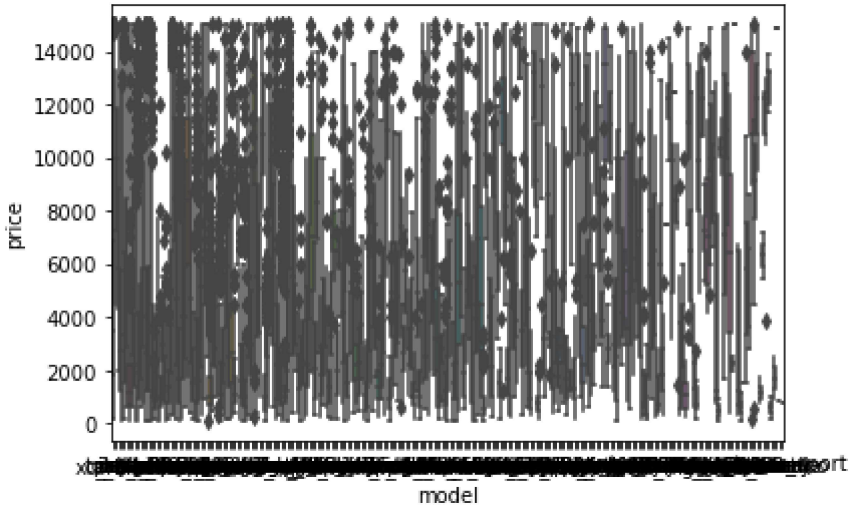
In [65]:
```
sns.countplot(x='model',data=cars)
```

Out[65]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26e5583a0>
```



In [66]:
```
sns.boxplot(x='model',y='price',data=cars)
#we are taking the value because model varies with price
#hence we are taking it
```

Out[66]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26ed17910>
```



kilometer

In [67]:
```
cars['kilometer'].value_counts().sort_index()
```

Out[67]:
```
5000        378
10000        80
20000       351
30000       413
40000       478
50000       656
60000       783
70000       920
80000      1108
90000      1230
100000     1580
125000     4184
150000    26599
Name: kilometer, dtype: int64
```

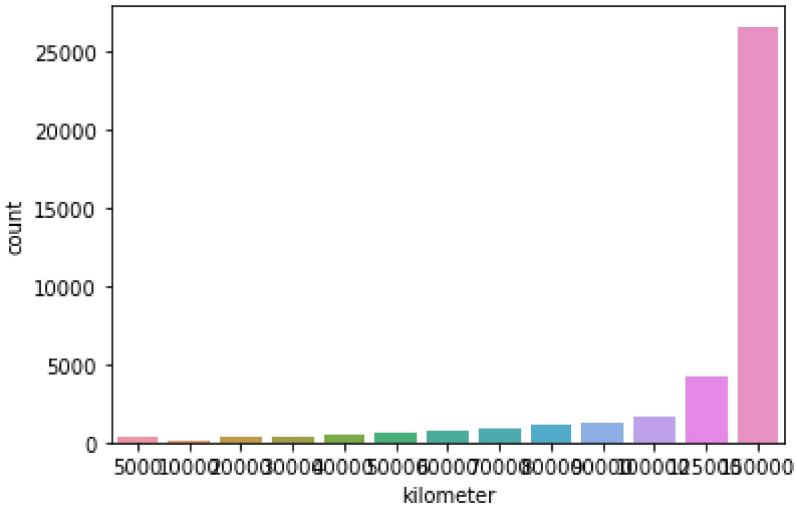
In [68]:
```
pd.crosstab(cars['kilometer'],columns='count',normalize=True)
```

Out[68]:

col_0	count
kilometer	
5000	0.009752
10000	0.002064
20000	0.009056
30000	0.010655
40000	0.012332
50000	0.016925
60000	0.020201
70000	0.023736
80000	0.028586
90000	0.031734
100000	0.040764
125000	0.107946
150000	0.686249

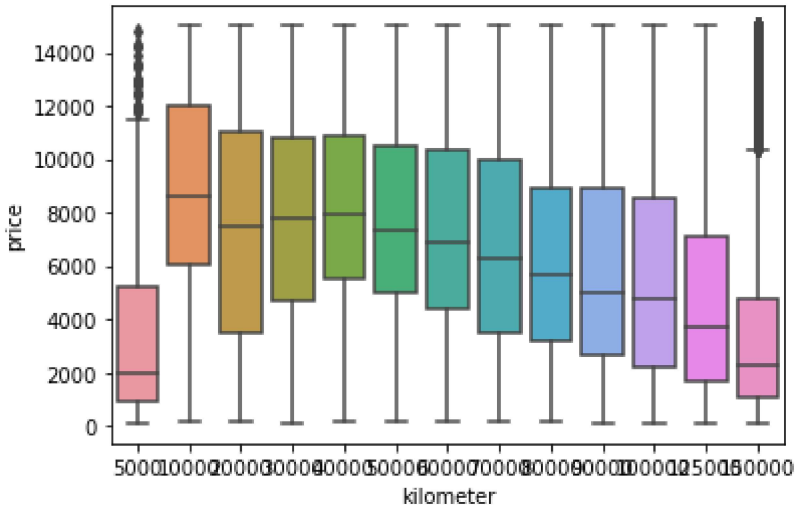
In [69]:
```
sns.countplot(x='kilometer',data=cars)
```

Out[69]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26673d160>
```



In [70]:
```
sns.boxplot(x='kilometer',y='price',data=cars)
#different kilometer affect the different price
#cars which are has less kilometer has high price
```

Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d26ecc77f0>
```



In [71]:
```
cars['kilometer'].describe()
#hence we are retaining the variable
#because kilometer varies with time
```

Out[71]:
```
count     38760.000000
mean     130433.178535
std       35026.307294
min        5000.000000
25%      125000.000000
50%      150000.000000
75%      150000.000000
max      150000.000000
Name: kilometer, dtype: float64
```

fueltype

In [72]:
```
cars['fuelType'].value_counts().sort_index()
```

Out[72]:
```
cng           70
diesel     10705
electro        9
hybrid        20
lpg          670
other          5
petrol     24754
Name: fuelType, dtype: int64
```

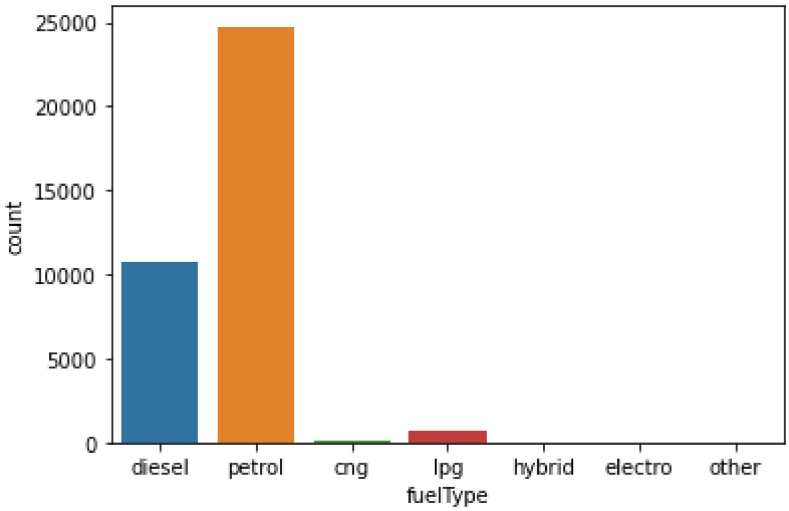
In [73]:
```
pd.crosstab(cars['fuelType'],columns='count',normalize=True)
```

Out[73]:

col_0	count
fuelType	
cng	0.001932
diesel	0.295449
electro	0.000248
hybrid	0.000552
lpg	0.018491
other	0.000138
petrol	0.683189

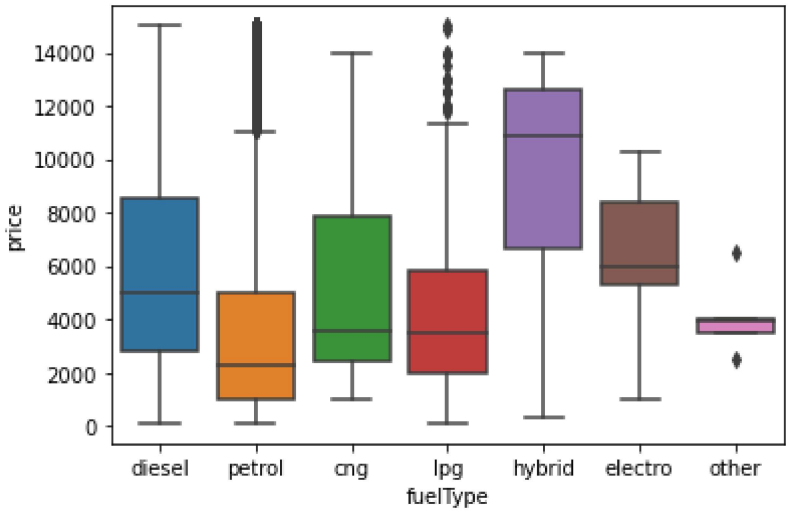
In [74]:
```
sns.countplot(x='fuelType',data=cars)
```

Out[74]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2711a6e50>
```



In [75]:
```
sns.boxplot(x='fuelType',y='price',data=cars)
#fuelType affect the price
#hence we are retaining it
```

Out[75]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d27120a700>
```



Brand

In [76]:
```
cars['brand'].value_counts()
```

Out[76]:
```
volkswagen        8395
opel              4402
bmw               4132
mercedes_benz     3455
audi              3212
ford              2673
renault           1911
peugeot           1302
fiat               981
seat               839
skoda              637
mazda              632
smart              620
citroen            586
nissan             571
toyota             520
volvo              392
hyundai            378
mini               366
mitsubishi         350
honda              292
suzuki             259
kia                251
alfa_romeo         239
sonstige_autos     209
chevrolet          197
chrysler           145
dacia              121
subaru             103
jeep                68
daihatsu            67
porsche             65
saab                64
jaguar              55
daewoo              53
lancia              53
rover               51
land_rover          49
trabant             43
lada                22
Name: brand, dtype: int64
```

In [77]:
```
pd.crosstab(cars['brand'],columns='count',normalize=True)
```

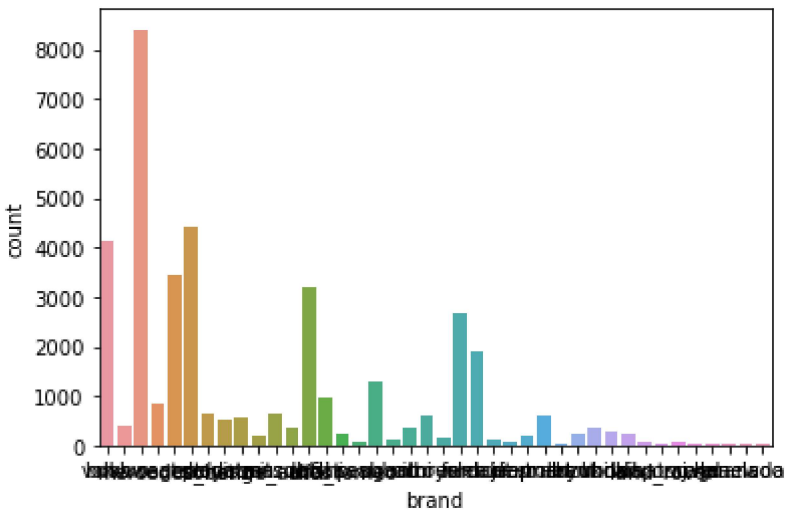
Out[77]:

col_0	count
brand	
alfa_romeo	0.006166
audi	0.082869
bmw	0.106605
chevrolet	0.005083
chrysler	0.003741
citroen	0.015119
dacia	0.003122
daewoo	0.001367
daihatsu	0.001729
fiat	0.025310
ford	0.068963
honda	0.007534
hyundai	0.009752
jaguar	0.001419
jeep	0.001754
kia	0.006476
lada	0.000568
lancia	0.001367
land_rover	0.001264
mazda	0.016305
mercedes_benz	0.089138
mini	0.009443
mitsubishi	0.009030
nissan	0.014732
opel	0.113571
peugeot	0.033591
porsche	0.001677
renault	0.049303
rover	0.001316
saab	0.001651
seat	0.021646
skoda	0.016434
smart	0.015996
sonstige_autos	0.005392

col_0	count
brand	
subaru	0.002657
suzuki	0.006682
toyota	0.013416
trabant	0.001109
volkswagen	0.216589
volvo	0.010114

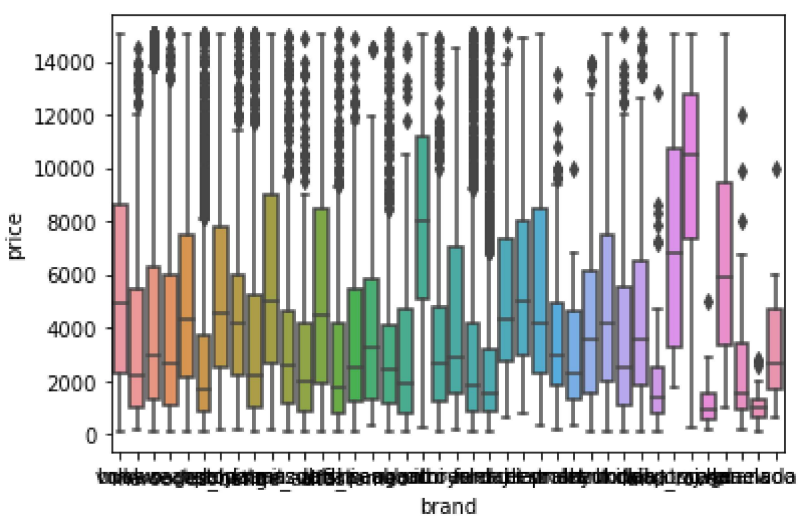
In [78]:
```
sns.countplot(x='brand',data=cars)
```

Out[78]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2712f75e0>
```



In [79]:
```
sns.boxplot(x='brand',y='price',data=cars)#each brand affect the price differentl
```

Out[79]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2713ead60>
```



Variable not repair

yes-car is damaged but not rectified

no-car was damaged but has been rectified

In [80]:
```
cars['notRepairedDamage'].value_counts()
```

Out[80]:
```
no     28736
yes     3941
Name: notRepairedDamage, dtype: int64
```

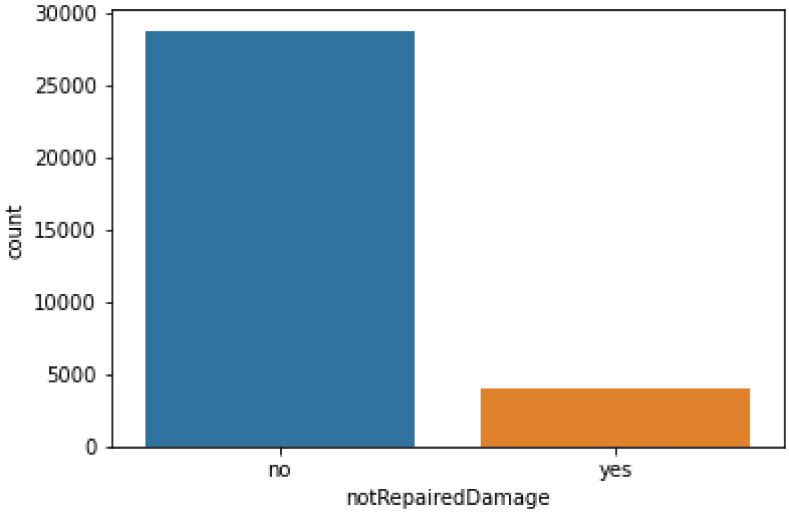
In [81]:
```
pd.crosstab(cars['notRepairedDamage'],columns='count',normalize=True)
```

Out[81]:

col_0	count
notRepairedDamage	
no	0.879395
yes	0.120605

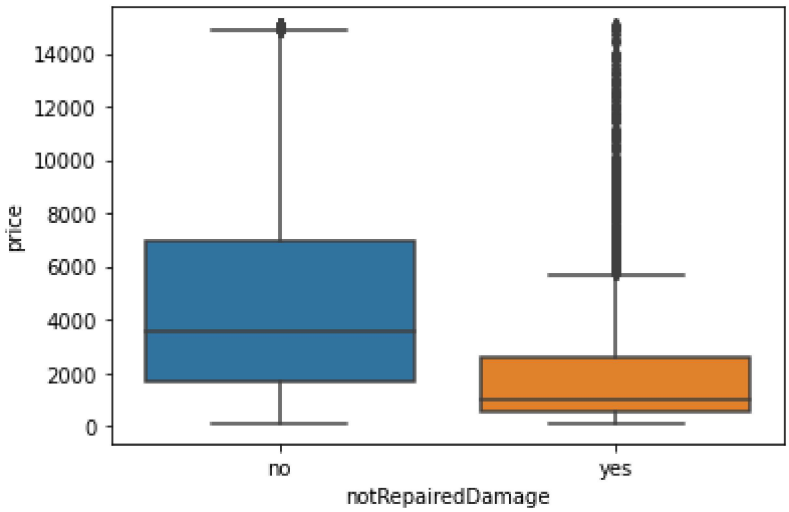
In [82]:
```
sns.countplot(x='notRepairedDamage',data=cars)
```

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d271768c40>
```



In [83]:
```
sns.boxplot(x='notRepairedDamage',y='price',data=cars)
#no has higher price as compare to the yes
#hence we retain it
```

Out[83]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2713ea0d0>
```



Removing the insignificant variable

In [84]:
```
col=['seller','offerType','abtest']
cars=cars.drop(columns=col,axis=1)
#here we are deleting these three columns from the data set
```

In [85]:
```
cars_copy=cars.copy()
#inly 10 columns left out of 13
print(cars_copy)
```

```
       price    vehicleType gearbox  powerPS     model  kilometer fuelType  \
0       4450      limousine  manual      150       3er     150000   diesel
1      13299            suv  manual      163  xc_reihe     150000   diesel
2       3200            bus  manual      101    touran     150000   diesel
3       4500      small car  manual       86     ibiza      60000   petrol
5        988      limousine  manual       90    passat     150000   petrol
...      ...            ...     ...      ...       ...        ...      ...
49991  10900      limousine  manual      231   x_reihe     150000   petrol
49992    790      limousine  manual       82     astra     150000   diesel
49993    830      small car  manual       60      clio     125000   petrol
49995   2290  station wagon  manual      101     astra     150000   petrol
50000   1100      small car  manual       38     matiz     150000   petrol

            brand notRepairedDamage    Age
0             bmw               NaN  15.25
1           volvo                no  13.50
2      volkswagen               NaN  15.92
3            seat                no  13.00
5      volkswagen                no  23.17
...           ...               ...    ...
49991         bmw                no  14.67
49992        opel                no  20.75
49993     renault                no  19.67
49995        opel                no  17.25
50000   chevrolet                no  12.83

[38760 rows x 10 columns]
```

Correlation

In [86]:
```
cars_select1=cars.select_dtypes(exclude=[object])#here we are selecting the datat
```

In [87]:
```
print(cars_select1)
#we have only numerical value
```

```
       price  powerPS  kilometer    Age
0       4450      150     150000  15.25
1      13299      163     150000  13.50
2       3200      101     150000  15.92
3       4500       86      60000  13.00
5        988       90     150000  23.17
...      ...      ...        ...    ...
49991  10900      231     150000  14.67
49992    790       82     150000  20.75
49993    830       60     125000  19.67
49995   2290      101     150000  17.25
50000   1100       38     150000  12.83

[38760 rows x 4 columns]
```

In [88]:
```
correlation=cars_select1.corr()
print(correlation)
```

```
              price   powerPS  kilometer       Age
price      1.000000  0.475581  -0.330323 -0.403710
powerPS    0.475581  1.000000   0.149311 -0.073302
kilometer -0.330323  0.149311   1.000000  0.218996
Age       -0.403710 -0.073302   0.218996  1.000000
```

In [89]:
```
round(correlation,3)
#here we are round up correlation value upto 3
```

Out[89]:

	price	powerPS	kilometer	Age
price	1.000	0.476	-0.330	-0.404
powerPS	0.476	1.000	0.149	-0.073
kilometer	-0.330	0.149	1.000	0.219
Age	-0.404	-0.073	0.219	1.000

In [90]:
```
cars_select1.corr().loc[:,'price'].abs().sort_values(ascending=False)[1:]
#here we are getting the value of r correlation
#only powerPS has some correlation weth price by there  way none of them depender
```

Out[90]:
```
powerPS      0.475581
Age          0.403710
kilometer    0.330323
Name: price, dtype: float64
```

We are going to build a linear regression and random forest model on two sets of data
1)data obtained by omitting the two missing values 2)data obtained by imputing the missing values

Omitting the missing values

In [91]:
```
cars_omit=cars.dropna(axis=0)
#here the any row have missing values the entire row will be deleted
```

In [92]:
```
#here cars_omit contain categorical data type column but machine learning do not
#converting the string values as 0 and 1 this is called encoding
```

In [93]:
```
cars_omit=pd.get_dummies(cars_omit,drop_first=True)
#here the number of columns increases
```

In [94]:
```
from sklearn.model_selection import train_test_split
#on train where we build a model and test where we test the data
#on test we generally test the data
```

In [95]:
```
from sklearn.linear_model import LinearRegression
```

In [96]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [97]:
```
from sklearn.metrics import mean_squared_error
```

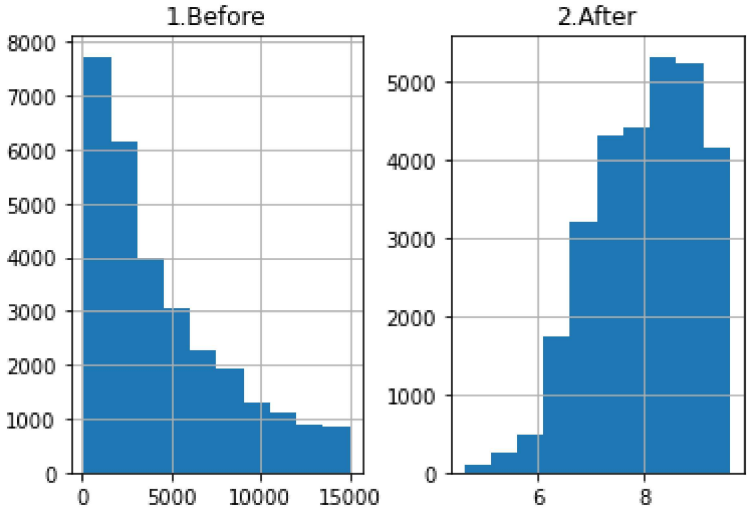
seperating input and output variable

In [98]:
```
x1=cars_omit.drop(['price'],axis='columns',inplace=False)
#here x1 represent the input value and we are droping column name price
#axis=column means droping the whole columns
#inplace =False MEANS THE changed value should not be reflected in cars_omit
#the result should store in X
```

In [99]:
```
y1=cars_omit['price']
#y1 has only column price
```

In [100]:
```
#Plotting the variable price
prices=pd.DataFrame({"1.Before":y1,"2.After":np.log(y1)})
prices.hist()
#here nice shaped curve so log y graph is good
#they are not not shewed while normal price was skewed
#so now we log of y1 as input feature
```

Out[100]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001D2735EC850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D273614D00>]],
      dtype=object)
```



In [101]:
```
#tranforming price as lograthmic value
y1=np.log(y1)
print(y1)
#the range of price is very huge so we are taking log
```

```
1        9.495444
3        8.411833
5        6.895683
7        7.243513
8        8.451053
           ...
49991    9.296518
49992    6.672033
49993    6.721426
49995    7.736307
50000    7.003065
Name: price, Length: 29250, dtype: float64
```

In [102]:
```
#Splitting data into test and train
```

In [103]:
```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=.3,random_state=3)
#here we are fitting data into train and test and 70 percent data go into train 3
#random_state 3 means same number will be genrated
```

In [104]:
```
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

```
(20475, 296) (8775, 296) (20475,) (8775,)
```

Baseline model for omitted data

The predictated value is replaced by mean value of the test data 2)This is to set a benchmark and to compare with our regression model

In [105]:
```
#finding the mean value for test data value
base_pred=np.mean(y_test)
print(base_pred)
```

```
8.033476402647011
```

In [106]:
```
#repeating the same value till length of test data
base_pred=np.repeat(base_pred,len(y_test))
```

In [107]:
```
#finding the rmse (root mean square error)
base_root_mean_square_error=np.sqrt(mean_squared_error(y_test,base_pred))
print(base_root_mean_square_error)
#this is bench mark for prediction
#any other model we make should give Rmse  which should be less than this
```

```
0.9807382237273811
```

Linear regression with Omitted data

In [108]:
```
#setting intercept as true
lgr=LinearRegression(fit_intercept=True)
```

In [109]:
```
#Model
model_lin1=lgr.fit(x_train,y_train)
#this is to be done because we are fitting the modele x_train for input and y_tai
```

In [110]:
```
#prediction model of the test data
cars_prediction_lin1=lgr.predict(x_test)
```

In [111]:
```
#computing MSE AND RMSE
#we are not giving base we are giving the test set of value
lin_mse1=mean_squared_error(y_test,cars_prediction_lin1)
lin_mse1=np.sqrt(lin_mse1)
print(lin_mse1)
```

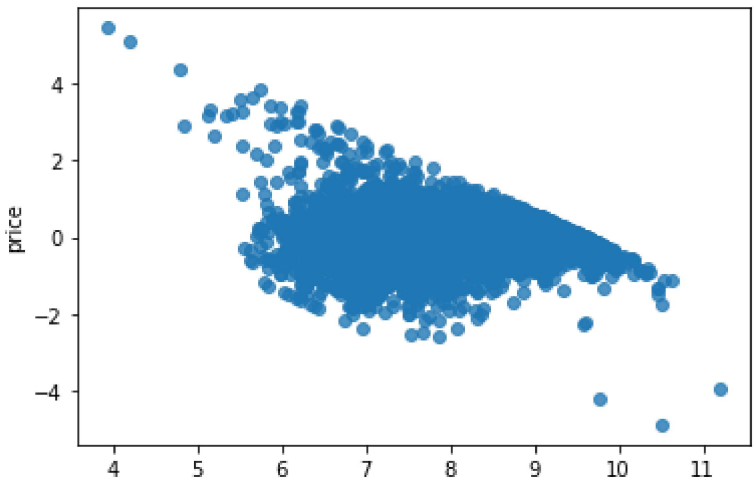
```
0.5336136553204845
```

In [112]:
```
#computing r squared value
#it gives you how good your module can explain variablity in y
r2_lin_test1=model_lin1.score(x_test,y_test)
r2_lin_train1=model_lin1.score(x_train,y_train)
print(r2_lin_test1)
print(r2_lin_train1)
#there is not somuch difference in test and train so this module is good
#it can catch more variability in test data
```

```
0.70396186094384
0.7269298589745459
```

In [114]:
```
#regression diagnostics-Residual plot analysis
residuals1=y_test-cars_prediction_lin1
#residual1 is difference between actual-predicted value this is the new columns
sns.regplot(x=cars_prediction_lin1,y=residuals1,scatter=True,fit_reg=False,data=c
residuals1.describe()
#residual value is equal to zero is good because residual value contain so much e
#if mean is equal to zero that means we are very close to find data
```

Out[114]:
```
count    8775.000000
mean        0.018550
std         0.533322
min        -4.857371
25%        -0.238427
50%         0.052348
75%         0.288358
max         5.436991
Name: price, dtype: float64
```



Regression diagnostics-Residual plot analysis

Random forest with omitted data

In [154]:
```
#model parameter
rf=RandomForestRegressor(n_estimators=100,max_features='auto',max_depth=100,min_s
#n_estimator tells you the number of tress in forest
#max_depth telss us the maximum depth of tress
#min_samples_split minimum number of samples  that we want to split
#min_sample_leaf it meas minimum of samples present at leaf node
#max_features maximum number of feature that we want to consider
#if we select auto it automatically choses
#sqrt it is sqrt of number of feature
#random_state =1 is uses ehen ever we run it gives us the same result
```

In [118]:
```
#Model
model_rf1=rf.fit(x_train,y_train)
```

In [119]:
```
#predicting the model on test set
cars_prediction_rf1=rf.predict(x_test)
```

In [122]:
```
#computing mse and rmse
rf_mse1=mean_squared_error(y_test,cars_prediction_rf1)
rf_rmse1=np.sqrt(rf_mse1)
print(rf_rmse1)
#this is better than linear regression beccuse rmse vallue is less here as compar
```

```
0.42406271125280426
```

In [124]:
```
#r squared value
r2_rf_test1=model_rf1.score(x_test,y_test)
r2_rf_train1=model_rf1.score(x_train,y_train)
print(r2_rf_test1,r2_rf_train1)
#here is the prove of it
```

```
0.8130377321738085 0.8924365224110377
```

Modelling with imputed data

In [125]:
```
cars_imputed=cars.apply(lambda x:x.fillna(x.median())
                        if x.dtypes=='float' else
                        x.fillna(x.value_counts().index[0]))
#here we are filling it with median values if data type is of float otherwise we
#different from the float type
```

In [128]:
```
cars_imputed.isnull().sum()
#none of the column has missing values
```

Out[128]:
```
price                    0
powerPS                  0
kilometer                0
Age                      0
vehicleType_cabrio       0
                        ..
brand_toyota             0
brand_trabant            0
brand_volkswagen         0
brand_volvo              0
notRepairedDamage_yes    0
Length: 300, dtype: int64
```

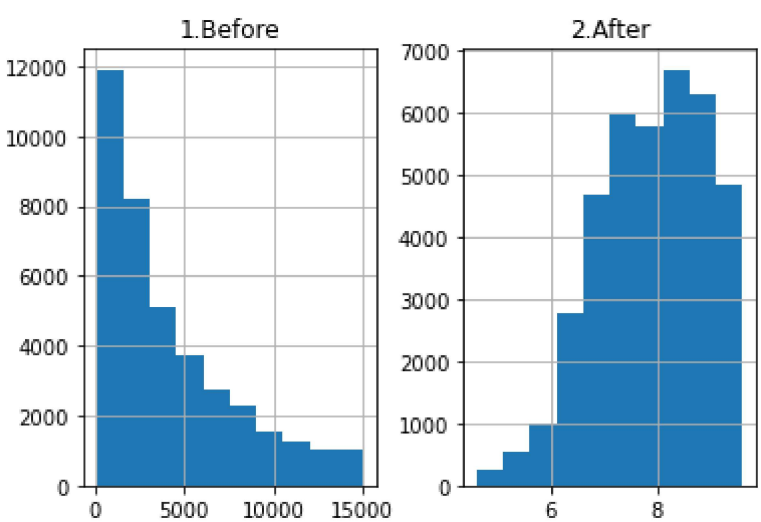
In [127]:
```
#converting the categorical data to dummy variable
cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)
#by this number of rows get increases
```

Model Building with imputed data

In [132]:
```
#seperating the input and output feature
x2=cars_imputed.drop(['price'],axis='columns',inplace=False)
y2=cars_imputed['price']
```

In [135]:
```
#plotting the variable price
prices=pd.DataFrame({"1.Before":y2,"2.After":np.log(y2)})
prices.hist()
#hence they are symmetric all te symmetric are log transform
```

Out[135]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001D276CD2AF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D276CDE520>]],
      dtype=object)
```



In [136]:
```
#transforming data into test and train
x_train1,x_test1,y_train1,y_test1=train_test_split(x2,y2,test_size=.3,random_stat
print(x_train1.shape,x_test1.shape,y_train.shape,y_test)
```

```
(27132, 299) (11628, 299) (20475,) 46318    6.214608
1081     7.162397
3367     9.179778
27288    8.779557
36568    8.895630
           ...
35983    8.968269
35603    7.522941
16651    6.856462
11930    9.609116
48605    9.615139
Name: price, Length: 8775, dtype: float64
```

Baseline model for imputed data

In [139]:
```
#finding mean of test data
base_pred=np.mean(y_test1)
print(base_pred)
```

```
4191.538441692466
```

In [140]:
```
#repeating same value till length of test data
base_pred=np.repeat(base_pred,len(y_test1))
```

In [145]:
```
#finding the rmse
base_root_mean_square_error_imputed=np.sqrt(mean_squared_error(y_test1,base_pred)
print(base_root_mean_square_error_imputed)
```

```
3709.940346130278
```

Linear regression with imputed data

In [146]:
```
#setting intercept as true
lgr2=LinearRegression(fit_intercept=True)
```

In [148]:
```
#model
model_lin2=lgr2.fit(x_train1,y_train1)
```

In [151]:
```
#predicting model on test set
cars_predictions_lin2=lgr2.predict(x_test1)
```

In [153]:
```
#computing mse and rmse
lin_mse2=mean_squared_error(y_test1,cars_predictions_lin2)
lin_rmse2=np.sqrt(lin_mse2)
print(lin_rmse2)
#rmse for linear regression model gone up and hence error gone up
```

```
2265.828715977272
```

Random Forest with imputed data

In [155]:
```
#model parameter
rf2=RandomForestRegressor(n_estimators=100,max_features='auto',max_depth=100,min_
#n_estimator tells you the number of tress in forest
#max_depth telss us the maximum depth of tress
#min_samples_split minimum number of samples  that we want to split
#min_sample_leaf it meas minimum of samples present at leaf node
#max_features maximum number of feature that we want to consider
#if we select auto it automatically choses
#sqrt it is sqrt of number of feature
#random_state =1 is uses ehen ever we run it gives us the same result
```

In [159]:
```
#Model
model_rf2=rf2.fit(x_train1,y_train1)
```

In [161]:
```
#predicting the model on test set
cars_prediction_rf2=rf2.predict(x_test1)
```

In [162]:
```
#computing mse and rmse
rf_mse2=mean_squared_error(y_test1,cars_prediction_rf2)
rf_rmse2=np.sqrt(rf_mse2)
print(rf_rmse2)
#this is better than linear regression with imputed data
```

```
1570.9879567701553
```

In [164]:
```
#r squared value
r2_rf_test2=model_rf2.score(x_test1,y_test1)
r2_rf_train2=model_rf2.score(x_train1,y_train1)
print(r2_rf_test2,r2_rf_train2)
#here is the prove of it
#here is the prove of it
#this is good r square value as compare linear regression module
```

```
0.8206869661427797 0.8939566339778875
```

In [167]:
```
# Final output

print("Metrics for models built from data where missing values were omitted")
print("R squared value for train from Linear Regression=  %s"% r2_lin_train1)
print("R squared value for test from Linear Regression=  %s"% r2_lin_test1)
print("R squared value for train from Random Forest=  %s"% r2_rf_train1)
print("R squared value for test from Random Forest=  %s"% r2_rf_test1)
print("Base RMSE of model built from data where missing values were omitted= %s"%
#print("RMSE value for test from Linear Regression=  %s"% lin_rmse1)
print("RMSE value for test from Random Forest=  %s"% rf_rmse1)
print("\n\n")
print("Metrics for models built from data where missing values were imputed")
#print("R squared value for train from Linear Regression=  %s"% r2_lin_train2)
print("R squared value for test from Linear Regression=  %s"% r2_lin_test2)
print("R squared value for train from Random Forest=  %s"% r2_rf_train2)
print("R squared value for test from Random Forest=  %s"% r2_rf_test2)
print("Base RMSE of model built from data where missing values were imputed= %s"%
print("RMSE value for test from Linear Regression=  %s"% lin_rmse2)
print("RMSE value for test from Random Forest=  %s"% rf_rmse2)
```

```
Metrics for models built from data where missing values were omitted
R squared value for train from Linear Regression=  0.7269298589745459
R squared value for test from Linear Regression=  0.70396186094384
R squared value for train from Random Forest=  0.8924365224110377
R squared value for test from Random Forest=  0.8130377321738085
Base RMSE of model built from data where missing values were omitted= 0.9807382
237273811
RMSE value for test from Random Forest=  0.42406271125280426



Metrics for models built from data where missing values were imputed

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-167-46a05ed07777> in <module>
     12 print("Metrics for models built from data where missing values were imp
uted")
     13 #print("R squared value for train from Linear Regression=  %s"% r2_lin_
train2)
---> 14 print("R squared value for test from Linear Regression=  %s"% r2_lin_te
st2)
     15 print("R squared value for train from Random Forest=  %s"% r2_rf_train2
)
     16 print("R squared value for test from Random Forest=  %s"% r2_rf_test2)

NameError: name 'r2_lin_test2' is not defined
```

In [ ]: