

EDA on used cars

November 25, 2022

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

```
[2]: # Getting the current working directory
os.getcwd()
```

```
[2]: 'C:\\Users\\shoaib\\Desktop\\Revise Py-2022'
```

```
[3]: #Reading the data from csv
cars_data=pd.read_csv("cars_sampled.csv")
```

```
[4]: #Creating copy of cars_data
cars=cars_data.copy() # deep copy
#Whatever changes made in cars will not be reflected in cars_data
```

```
[5]: cars.head()
```

```
[5]:      dateCrawled      name \
0  30/03/2016 13:51      Zu_verkaufen
1    7/3/2016 9:54  Volvo_XC90_2.4D_Summum
2   1/4/2016 0:57  Volkswagen_Touran
3  19/03/2016 17:50  Seat_Ibiza_1.4_16V_Reference
4  16/03/2016 14:51  Volvo_XC90_D5_Aut._RDesign_R_Design_AWD_GSHD_S...

      seller offerType  price  abtest vehicleType  yearOfRegistration \
0  private      offer   4450   test   limousine              2003
1  private      offer  13299  control         suv              2005
2  private      offer   3200   test         bus              2003
3  private      offer   4500  control   small car              2006
4  private      offer  18750   test         suv              2008

      gearbox  powerPS      model  kilometer  monthOfRegistration  fuelType \
0    manual     150        3er    150000              3    diesel
1    manual     163  xc_reihe    150000              6    diesel
2    manual     101    touran    150000             11    diesel
3    manual      86    ibiza     60000             12    petrol
4  automatic     185  xc_reihe    150000             11    diesel
```

	brand	notRepairedDamage	dateCreated	postalCode	lastSeen
0	bmw	NaN	30/03/2016 0:00	20257	7/4/2016 4:44
1	volvo	no	7/3/2016 0:00	88045	26/03/2016 13:17
2	volkswagen	NaN	31/03/2016 0:00	27449	1/4/2016 8:40
3	seat	no	19/03/2016 0:00	34537	7/4/2016 4:44
4	volvo	no	16/03/2016 0:00	55270	1/4/2016 23:18

```
[6]: cars.info()
#This gives the total entries in a column and its data type
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50001 entries, 0 to 50000
Data columns (total 19 columns):
dateCrawled      50001 non-null object
name            50001 non-null object
seller          50001 non-null object
offerType       50001 non-null object
price           50001 non-null int64
abtest         50001 non-null object
vehicleType     44813 non-null object
yearOfRegistration 50001 non-null int64
gearbox         47177 non-null object
powerPS         50001 non-null int64
model           47243 non-null object
kilometer       50001 non-null int64
monthOfRegistration 50001 non-null int64
fuelType        45498 non-null object
brand           50001 non-null object
notRepairedDamage 40285 non-null object
dateCreated     50001 non-null object
postalCode      50001 non-null int64
lastSeen        50001 non-null object
dtypes: int64(6), object(13)
memory usage: 7.2+ MB
```

```
[7]: pd.set_option('display.float_format', lambda x: '%.3f'%x) # Get the floating_
      ↪point to 3 decimal places
pd.set_option('display.max_columns', 500) # To display maximum set of columns
cars.describe() #This gives the statistical info about the data- https://www.
      ↪mathsisfun.com/data/standard-deviation.html
# https://statisticsbyjim.com/basics/coefficient-variation/
```

```
[7]:
```

	price	yearOfRegistration	powerPS	kilometer	\
count	50001.000	50001.000	50001.000	50001.000	
mean	6559.865	2005.544	116.496	125613.688	
std	85818.470	122.992	230.568	40205.234	
min	0.000	1000.000	0.000	5000.000	

25%	1150.000	1999.000	69.000	125000.000
50%	2950.000	2003.000	105.000	150000.000
75%	7190.000	2008.000	150.000	150000.000
max	12345678.000	9999.000	19312.000	150000.000

	monthOfRegistration	postalCode
count	50001.000	50001.000
mean	5.744	50775.217
std	3.711	25743.702
min	0.000	1067.000
25%	3.000	30559.000
50%	6.000	49504.000
75%	9.000	71404.000
max	12.000	99998.000

```
[8]: print('-----')
      print('          Data Cleaning          ')
      print('-----')
```

```
-----
          Data Cleaning
-----
```

```
[9]: #Dropping unwanted columns as these variables are related ads
      cars=cars.drop(['dateCrawled','name','dateCreated','lastSeen'],axis=1)
```

```
[10]: #Removing the duplicates
       cars.drop_duplicates(keep='first',inplace=True)
       cars.info()
       #16 duplicates removed from the cars dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49666 entries, 0 to 50000
Data columns (total 15 columns):
seller                49666 non-null object
offerType             49666 non-null object
price                49666 non-null int64
abtest               49666 non-null object
vehicleType          44491 non-null object
yearOfRegistration   49666 non-null int64
gearbox              46855 non-null object
powerPS              49666 non-null int64
model                46919 non-null object
kilometer            49666 non-null int64
monthOfRegistration   49666 non-null int64
fuelType             45177 non-null object
brand                49666 non-null object
notRepairedDamage    39980 non-null object
```

```
postalCode          49666 non-null int64
dtypes: int64(6), object(9)
memory usage: 6.1+ MB
```

```
[11]: #Checking the missing values from each variable
cars.isnull().sum()
```

```
[11]: seller          0
offerType          0
price             0
abtest            0
vehicleType       5175
yearOfRegistration 0
gearbox          2811
powerPS           0
model            2747
kilometer         0
monthOfRegistration 0
fuelType         4489
brand             0
notRepairedDamage 9686
postalCode        0
dtype: int64
```

```
[12]: #variable- year of registration
print(cars['yearOfRegistration'].value_counts().sort_index())
#There are some cars which are registered before 1900 and there are also cars
→after 2019
```

```
1000      6
1255      1
1500      2
1910     15
1928      1
1929      1
1933      1
1934      1
1936      2
1938      1
1940      1
1941      1
1943      2
1945      2
1947      2
1950      4
1951      4
1952      3
1953      2
```

1954	1
1955	6
1956	7
1957	5
1958	4
1959	5
1960	33
1961	7
1962	6
1963	11
1964	16
...	
2002	2560
2003	2742
2004	2614
2005	3109
2006	2668
2007	2357
2008	2190
2009	2018
2010	1645
2011	1548
2012	1235
2013	821
2014	624
2015	406
2016	1351
2017	1376
2018	528
2019	2
2222	1
2900	1
3000	1
3500	1
3800	1
5000	3
6000	4
7500	1
7800	1
8500	1
8888	2
9999	7

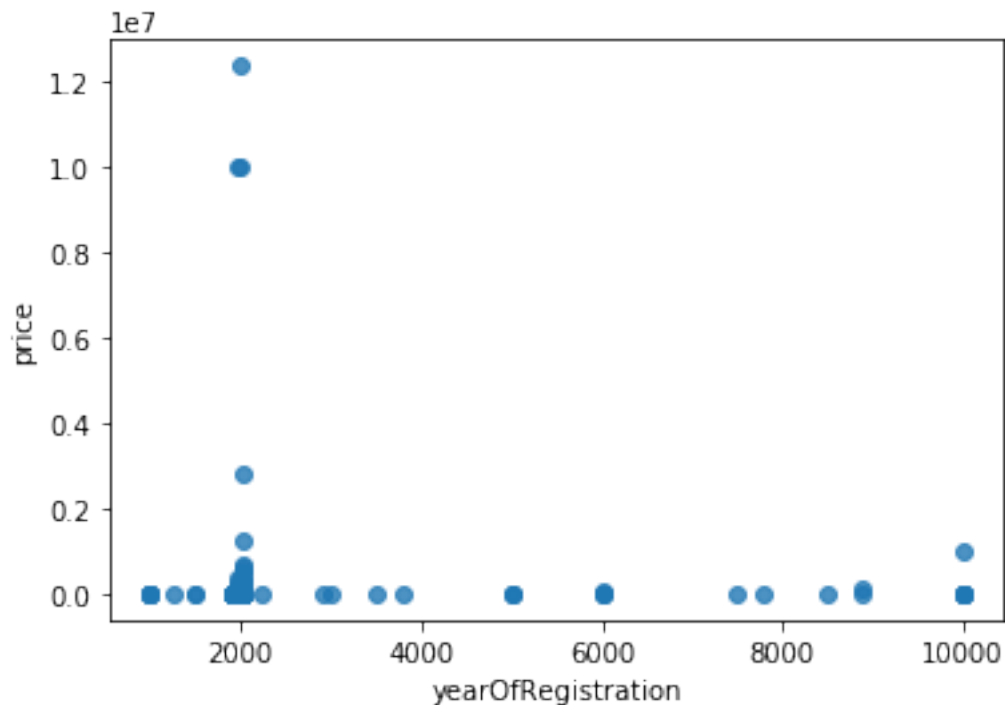
Name: yearOfRegistration, Length: 97, dtype: int64

```
[13]: print(cars['yearOfRegistration'].describe())
      #There is no much difference b/w mean and median but std dev is differing from
      ↳mean by 123 years
```

```
sns.
→regplot(x='yearOfRegistration',y='price',scatter=True,fit_reg=False,data=cars)
#Here we can't get any clear picture as there lot of outliers. so data is
→concentrated at one point
```

```
count    49666.000
mean      2005.551
std       123.405
min       1000.000
25%       1999.000
50%       2003.000
75%       2008.000
max       9999.000
Name: yearOfRegistration, dtype: float64
```

[13]: <matplotlib.axes._subplots.AxesSubplot at 0x15fda7a3940>



```
[14]: #After trial error, getting working range for year of registration
# Working range- 1950 and 2018
print(sum(cars['yearOfRegistration'] > 2018))
print(sum(cars['yearOfRegistration'] < 1950))
```

26
39

```
[15]: #Variable - Price
cars['price'].value_counts().sort_index()
#There are 1437 cars at 0 dollar
```

```
[15]: 0          1437
      1          172
      2           1
      3           1
      5           4
      7           1
      8           2
     10           5
     11           1
     12           1
     14           1
     15           8
     20           6
     21           1
     25           5
     26           1
     30           7
     35           4
     39           1
     40           3
     45           6
     50          41
     55           3
     60           7
     65           1
     70           2
     75           9
     77           1
     80          12
     85           3

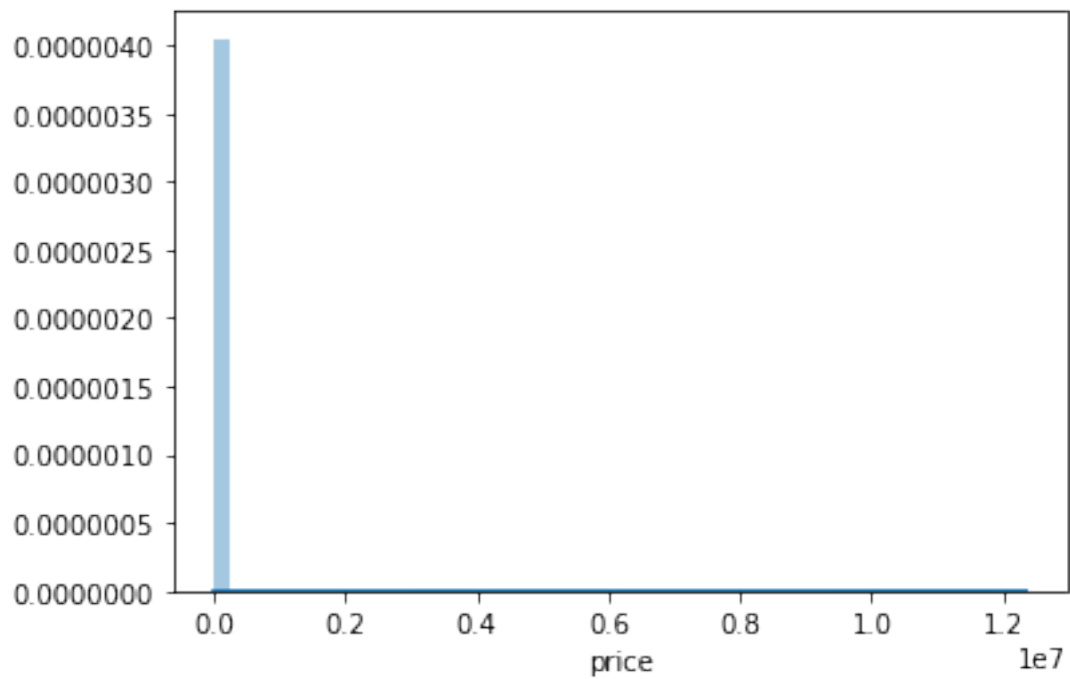
      ...
163991          1
165000          1
169999          1
171000          1
175000          1
179999          1
189981          1
205000          1
214800          1
225000          1
230000          2
239000          1
```

249000	1
250000	1
257500	1
260000	1
270000	1
300000	1
370000	1
395000	1
485000	1
487000	1
619000	1
700000	1
999999	1
1250000	1
2795000	1
9999999	1
10010011	1
12345678	1

Name: price, Length: 2393, dtype: int64

```
[16]: print(sns.distplot(cars['price']))
      #Here the values are cluttered at zero
```

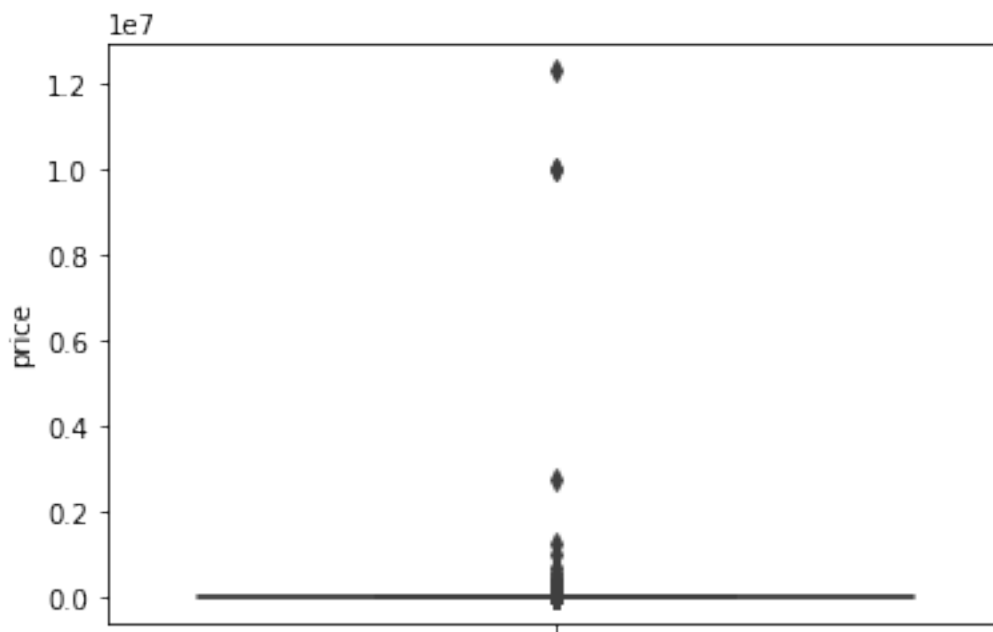
AxesSubplot(0.125,0.125;0.775x0.755)




```
[17]: print(sns.boxplot(y=cars['price']))
      # similarly boxplot is not clearly visible due to outliers
      cars['price'].describe()
      # Std dev is 1300% of mean
```

AxesSubplot(0.125,0.125;0.775x0.755)

```
[17]: count      49666.000
      mean        6559.299
      std         86105.705
      min           0.000
      25%         1150.000
      50%         2950.000
      75%         7100.000
      max        12345678.000
      Name: price, dtype: float64
```



```
[18]: #count
      print(sum(cars['price'] > 150000))
      print(sum(cars['price'] < 100))
      # Working range- 100 and 150000 after trial and error
```

34
1770

```
[19]: # Variable powerPS
cars['powerPS'].value_counts().sort_index()
#5581 cars have zero horse power
```

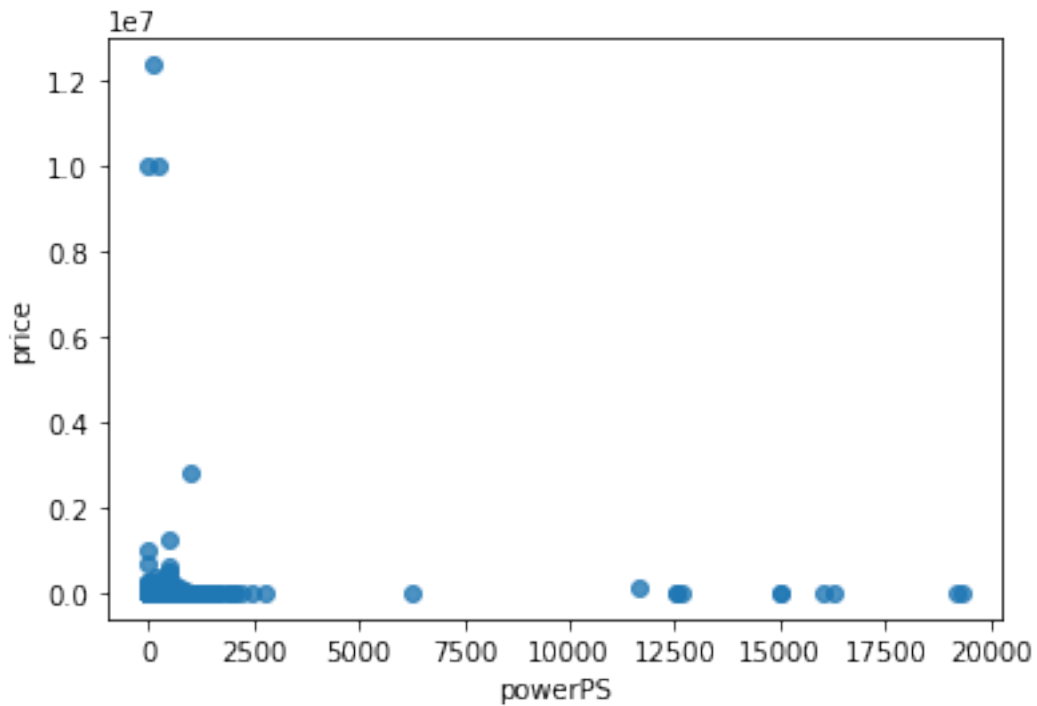
```
[19]: 0          5581
      1           3
      2           2
      3           2
      4           4
      5          17
      6           2
      7           1
      8           1
      9           2
     10           4
     11           6
     12           1
     13           6
     14           3
     15           1
     16           9
     17           2
     18           1
     19           1
     20           3
     21           5
     22          46
     23           5
     24           2
     25           7
     26           1
     27           2
     28           5
     29           2
     30           29
     ...
    1223           1
    1256           1
    1363           1
    1416           1
    1502           1
    1595           1
    1598           1
    1625           1
    1653           1
    1799           1
    1910           1
    1968           1
```

1992	1
1998	1
2004	1
2017	1
2172	1
2461	1
2789	1
6226	1
11620	1
12510	1
12512	1
12684	1
15017	1
15033	1
16011	1
16312	1
19211	1
19312	1

Name: powerPS, Length: 460, dtype: int64

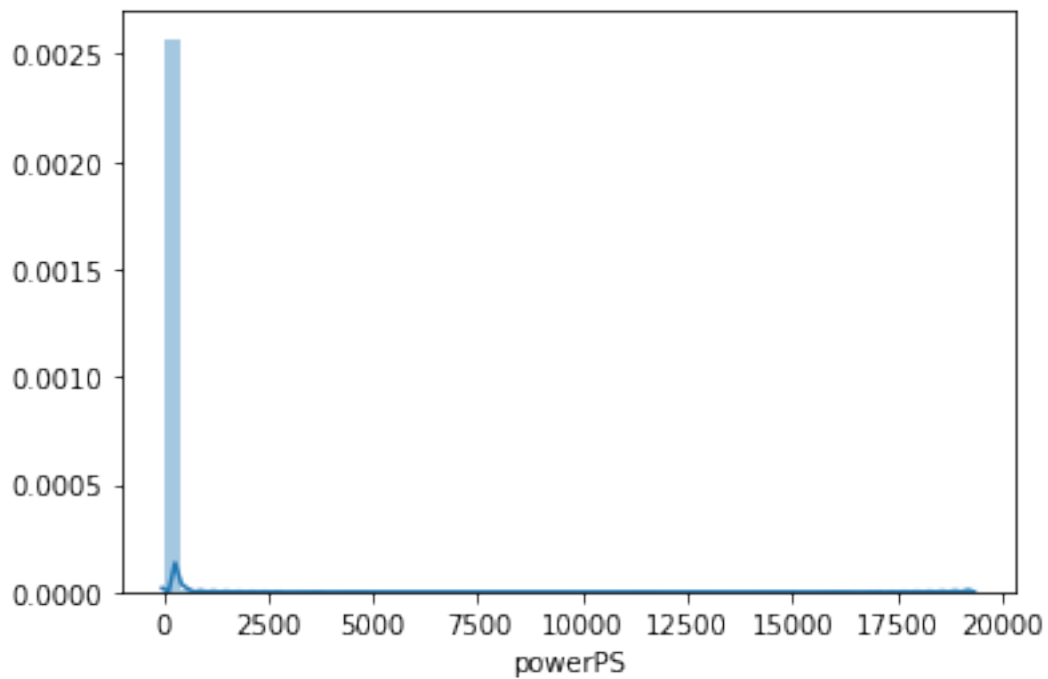
```
[20]: sns.regplot(x='powerPS',y='price',scatter=True,fit_reg=False,data=cars)
      #data is concentrated at zero power ps
```

```
[20]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdbaf9d68>
```



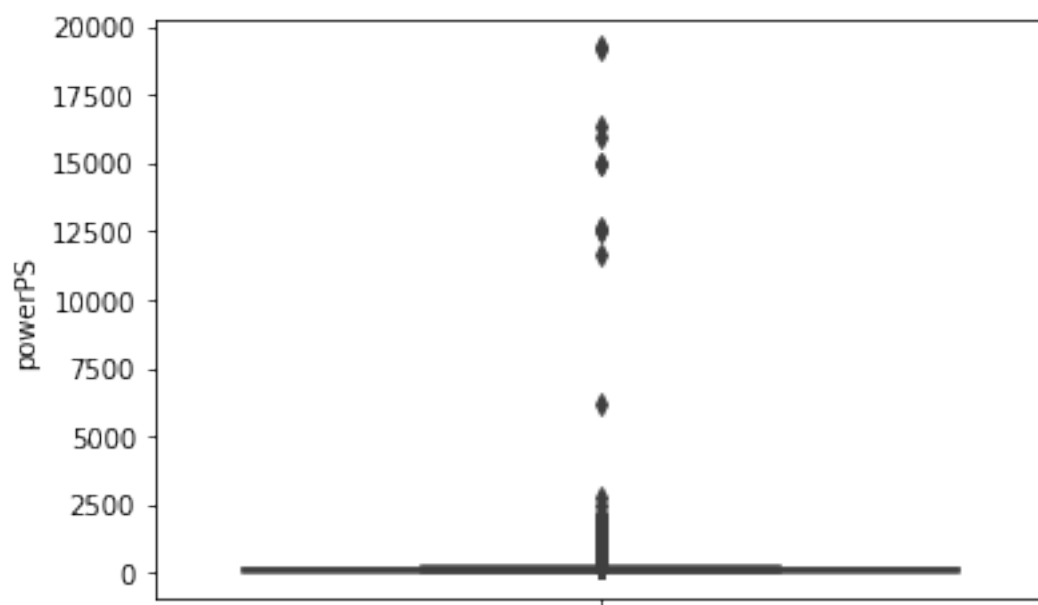
```
[21]: sns.distplot(cars['powerPS'])
```

```
[21]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdbbbe940>
```



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

```
[22]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdbcc5f60>
```



```
[23]: cars['powerPS'].describe()
      #std dev is 200% of mean
      #minimum is zero but engine cannot start at that HP
```

```
[23]: count    49666.000
      mean      116.404
      std       231.262
      min        0.000
      25%       69.000
      50%      105.000
      75%      150.000
      max     19312.000
      Name: powerPS, dtype: float64
```

```
[24]: # Working range- 10 and 500
      print(sum(cars['powerPS'] > 500))
      print(sum(cars['powerPS'] < 10))
```

```
115
5613
```

```
[25]: # =====
      # Working range of data
      # =====
      cars=cars[
          (cars.yearOfRegistration <= 2018)
          & (cars.yearOfRegistration >= 1950)
          & (cars.price >= 100)
          & (cars.price <= 150000)
          & (cars.powerPS >= 10)
          & (cars.powerPS <= 500)]
      cars.shape
      #~7300 records are dropped
```

```
[25]: (42855, 15)
```

```
[26]: #Variable reduction
      #Combining year of registration and month of registration to form new variable_
      ↪Age
      cars['monthOfRegistration']/=12
      cars.head()
```

```
[26]:   seller offerType  price  abtest vehicleType  yearOfRegistration  \
0  private    offer   4450    test   limousine           2003
1  private    offer  13299  control         suv           2005
2  private    offer   3200    test         bus           2003
3  private    offer   4500  control   small car           2006
4  private    offer  18750    test         suv           2008
```

	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	\
0	manual	150	3er	150000	0.250	diesel	
1	manual	163	xc_reihe	150000	0.500	diesel	
2	manual	101	touran	150000	0.917	diesel	
3	manual	86	ibiza	60000	1.000	petrol	
4	automatic	185	xc_reihe	150000	0.917	diesel	

	brand	notRepairedDamage	postalCode
0	bmw	NaN	20257
1	volvo	no	88045
2	volkswagen	NaN	27449
3	seat	no	34537
4	volvo	no	55270

```
[27]: #new variable Age
cars['Age']=(2018-cars['yearOfRegistration']) + cars['monthOfRegistration']
cars['Age']= round(cars['Age'],2)
```

```
[28]: cars['Age'].describe()
```

```
[28]: count    42855.000
mean         14.872
std           7.090
min           0.000
25%          10.330
50%          14.830
75%          19.170
max           67.750
Name: Age, dtype: float64
```

```
[29]: #dropping year of registration and month of registration
cars=cars.drop(['yearOfRegistration','monthOfRegistration'],axis=1)
```

```
[30]: cars.head()
```

	seller	offerType	price	abtest	vehicleType	gearbox	powerPS	\
0	private	offer	4450	test	limousine	manual	150	
1	private	offer	13299	control	suv	manual	163	
2	private	offer	3200	test	bus	manual	101	
3	private	offer	4500	control	small car	manual	86	
4	private	offer	18750	test	suv	automatic	185	

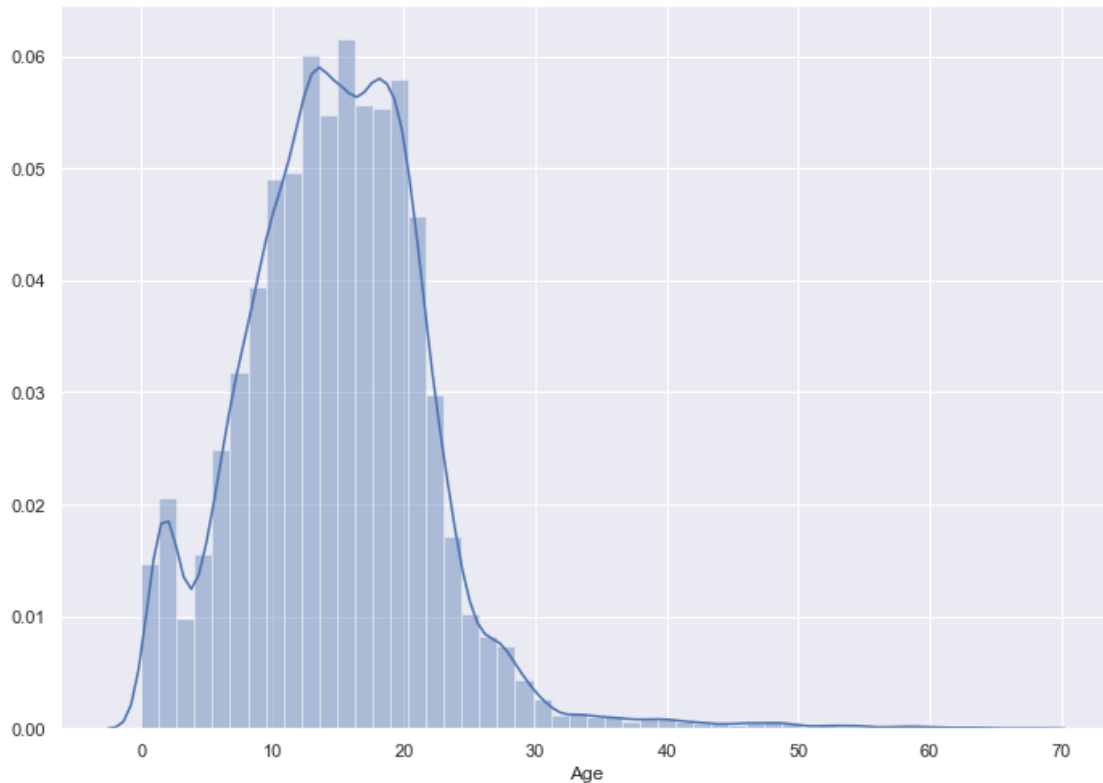
	model	kilometer	fuelType	brand	notRepairedDamage	postalCode	\
0	3er	150000	diesel	bmw	NaN	20257	
1	xc_reihe	150000	diesel	volvo	no	88045	
2	touran	150000	diesel	volkswagen	NaN	27449	
3	ibiza	60000	petrol	seat	no	34537	
4	xc_reihe	150000	diesel	volvo	no	55270	

```
Age
0 15.250
1 13.500
2 15.920
3 13.000
4 10.920
```

```
[31]: # =====
# Visualizing Parameters
# =====
sns.set(rc={'figure.figsize':(11.7,8.27)}) #A4 size dimension
```

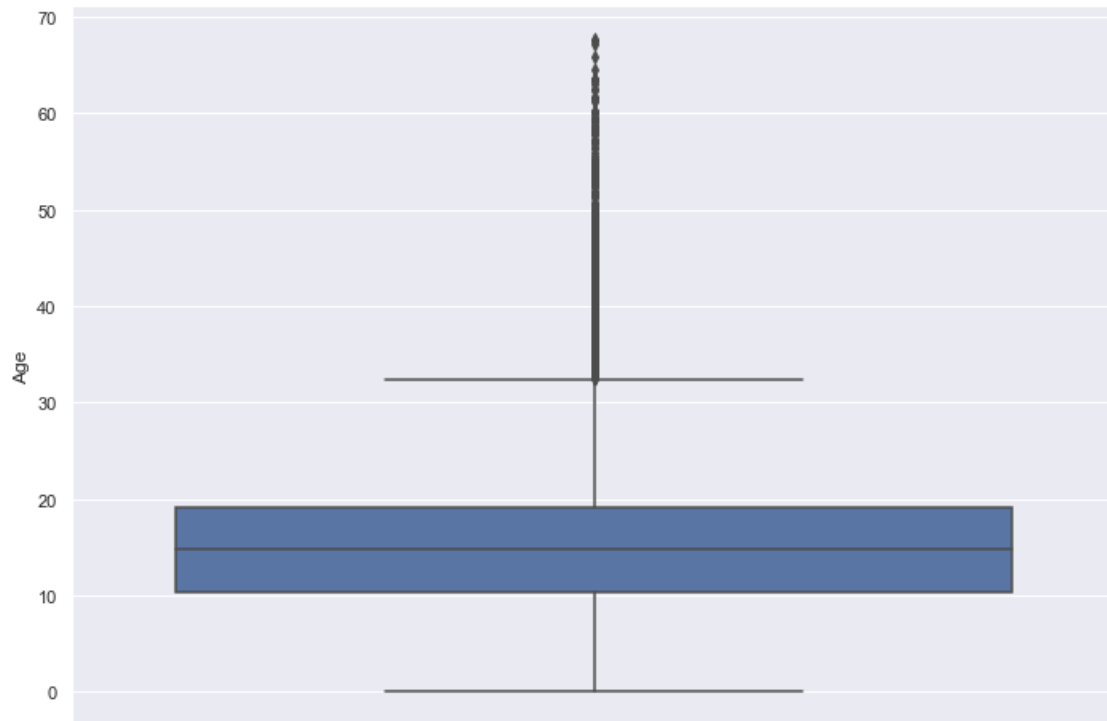
```
[32]: #Age
sns.distplot(cars['Age'])
#data is slightly skewed towards right
```

```
[32]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdbb5d2e8>
```



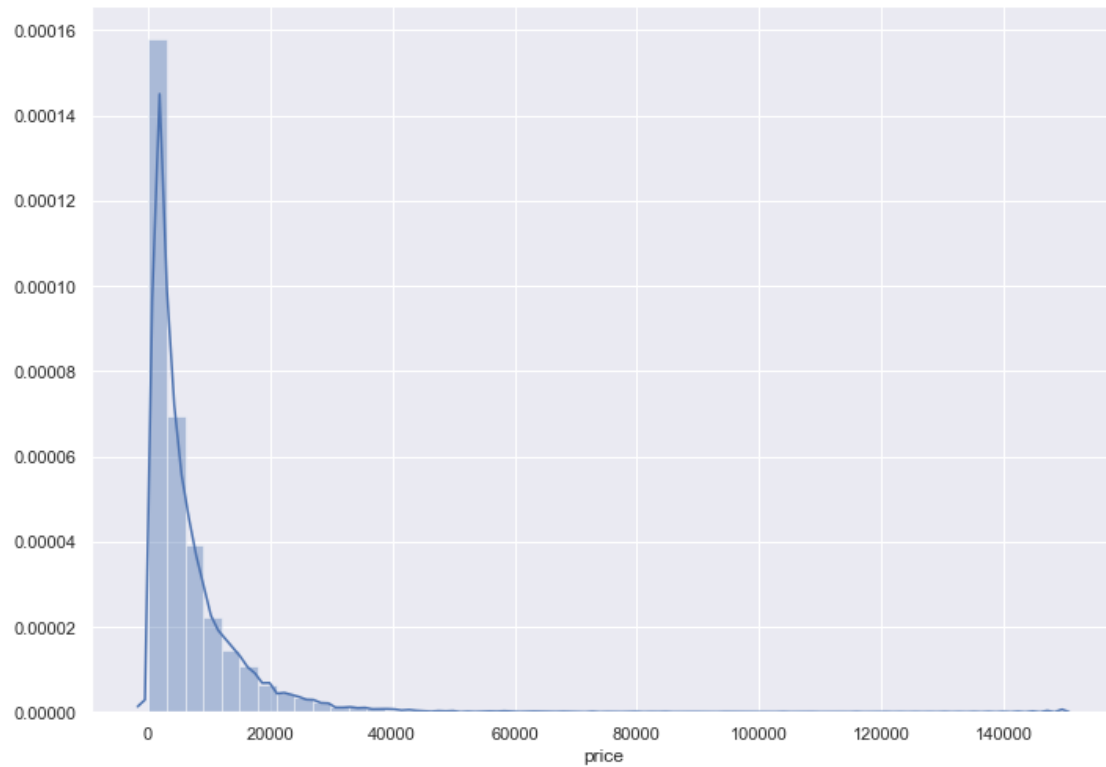
```
[33]: sns.boxplot(y=cars['Age'])
# Here it can be see that plots evenly distributed but still there are few
→outliers
```

```
[33]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdb0860b8>
```



```
[34]: #price
sns.distplot(cars['price'])
#plot is right skewed because of high end luxurious cars
```

```
[34]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdb048cc0>
```

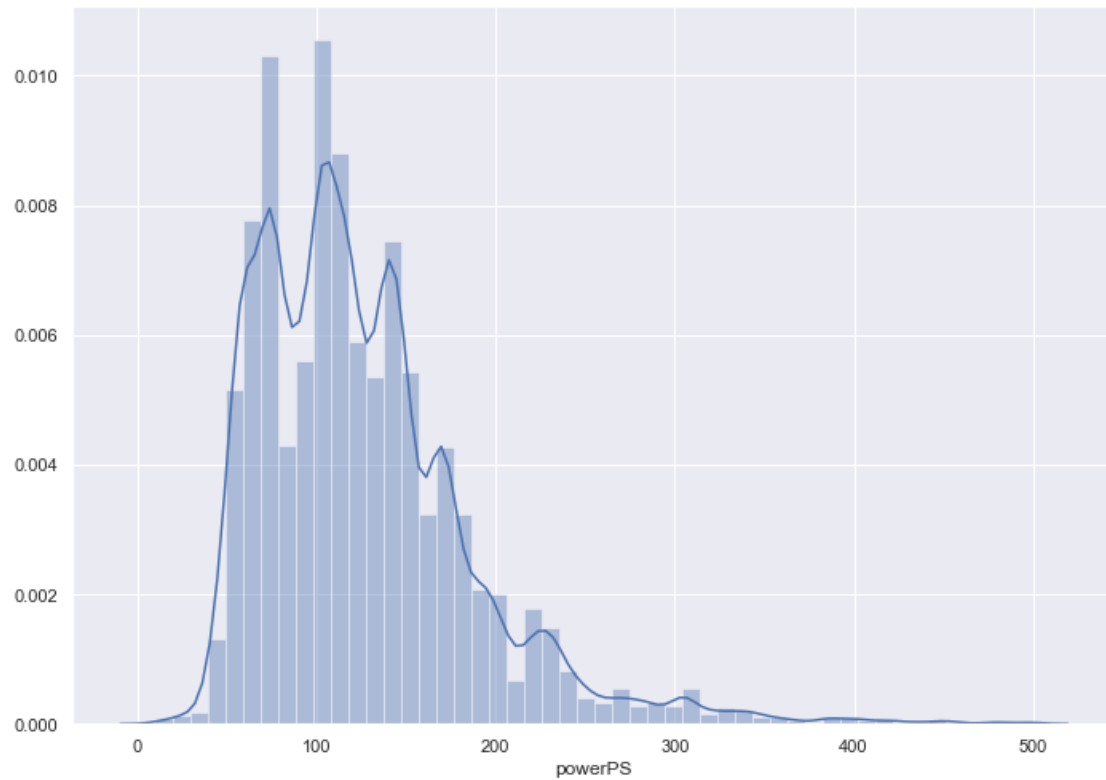
```
[35]: sns.boxplot(y=cars['price'])  
cars['price'].describe()  
#std dev is still high due to luxurious cars
```

```
[35]: count    42855.000  
      mean      6132.950  
      std      7944.135  
      min       100.000  
      25%     1450.000  
      50%     3499.000  
      75%     7900.000  
      max    149000.000  
      Name: price, dtype: float64
```



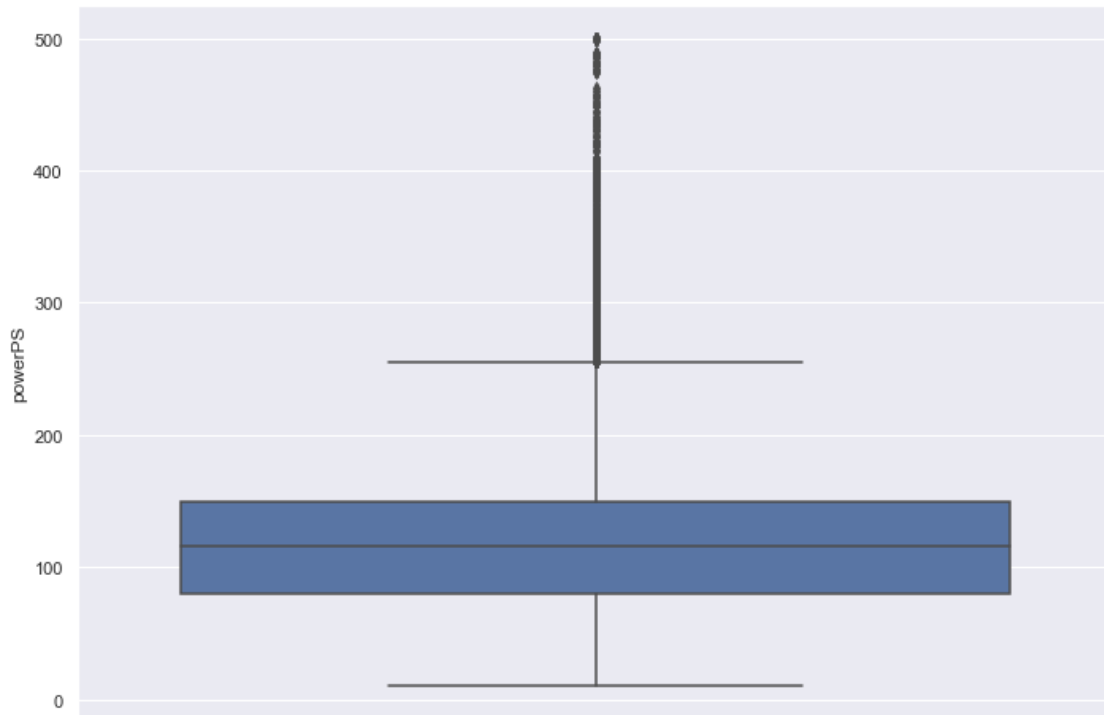
```
[36]: #powerPS  
sns.distplot(cars['powerPS'])
```

```
[36]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdca838d0>
```



```
[37]: sns.boxplot(y=cars['powerPS'])  
cars['powerPS'].describe()
```

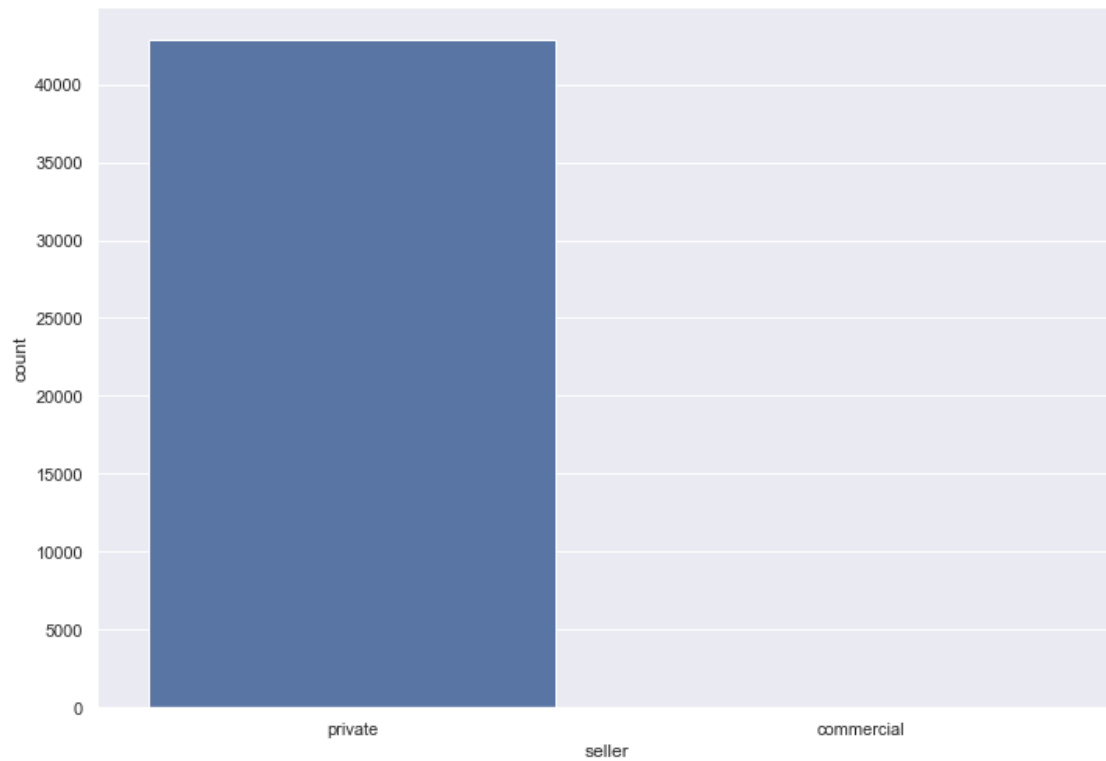
```
[37]: count    42855.000  
      mean      126.054  
      std       60.525  
      min       10.000  
      25%       80.000  
      50%      116.000  
      75%      150.000  
      max      500.000  
      Name: powerPS, dtype: float64
```



```
[38]: #Variable Age
print(cars['seller'].value_counts())
print(sns.countplot(cars['seller']))
pd.crosstab(cars['seller'], columns='count', normalize=True)
#almost all cars from private players; 99.99% of sellers at storm motors are
→private persons not commercial businessess
```

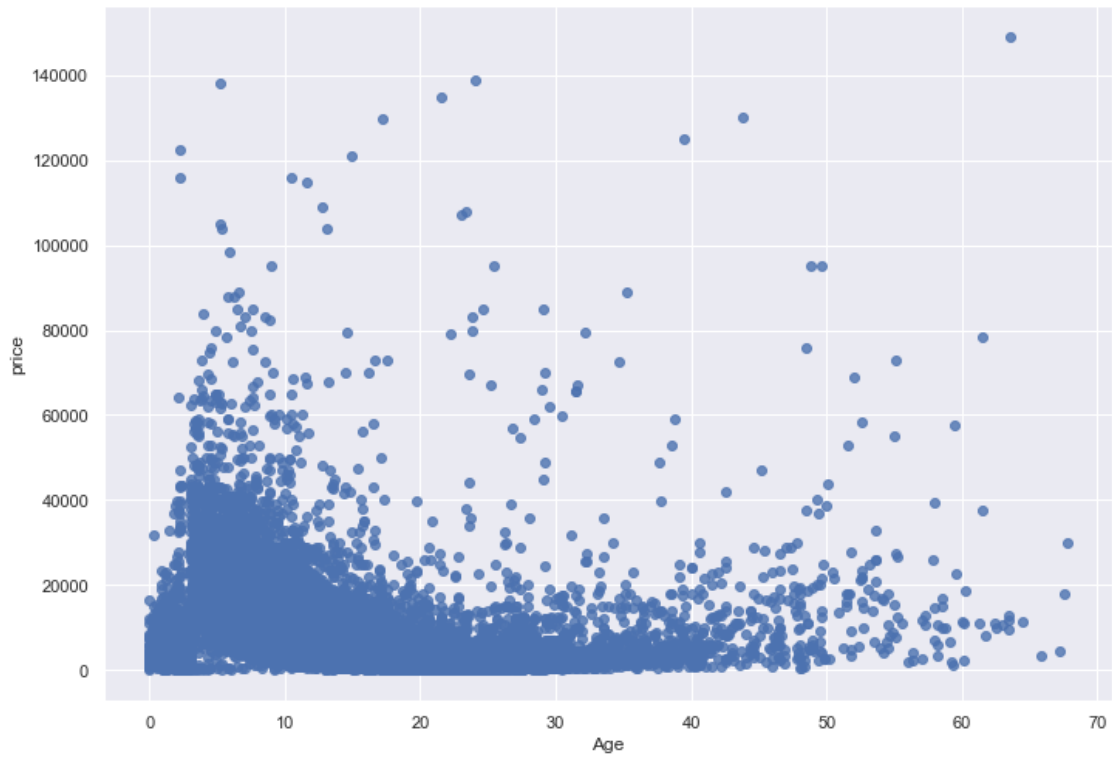
```
private      42854
commercial      1
Name: seller, dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
```

```
[38]: col_0      count
seller
commercial  0.000
private    1.000
```



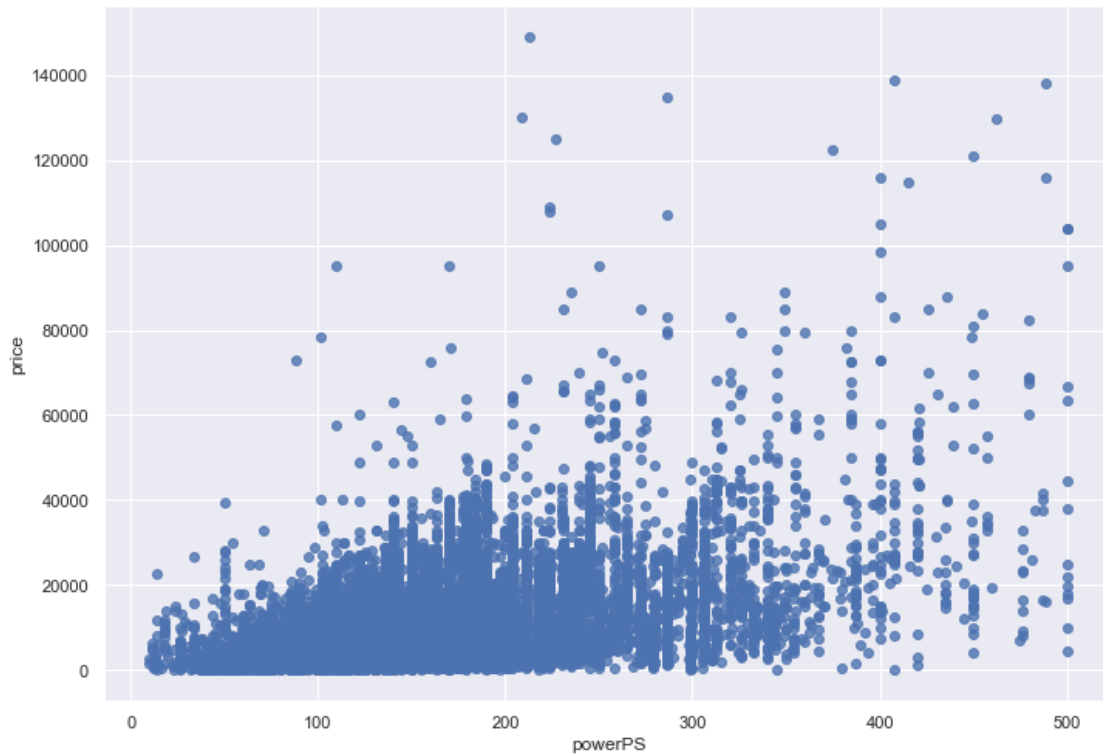
```
[39]: # Age Vs Price
sns.regplot(x='Age',y='price',scatter=True,fit_reg=False,data=cars)
#As the Age of a vehicle increases, price starts to decrease
#but there are some cars whose price is higher despite increase in age
```

```
[39]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdd22c1d0>
```



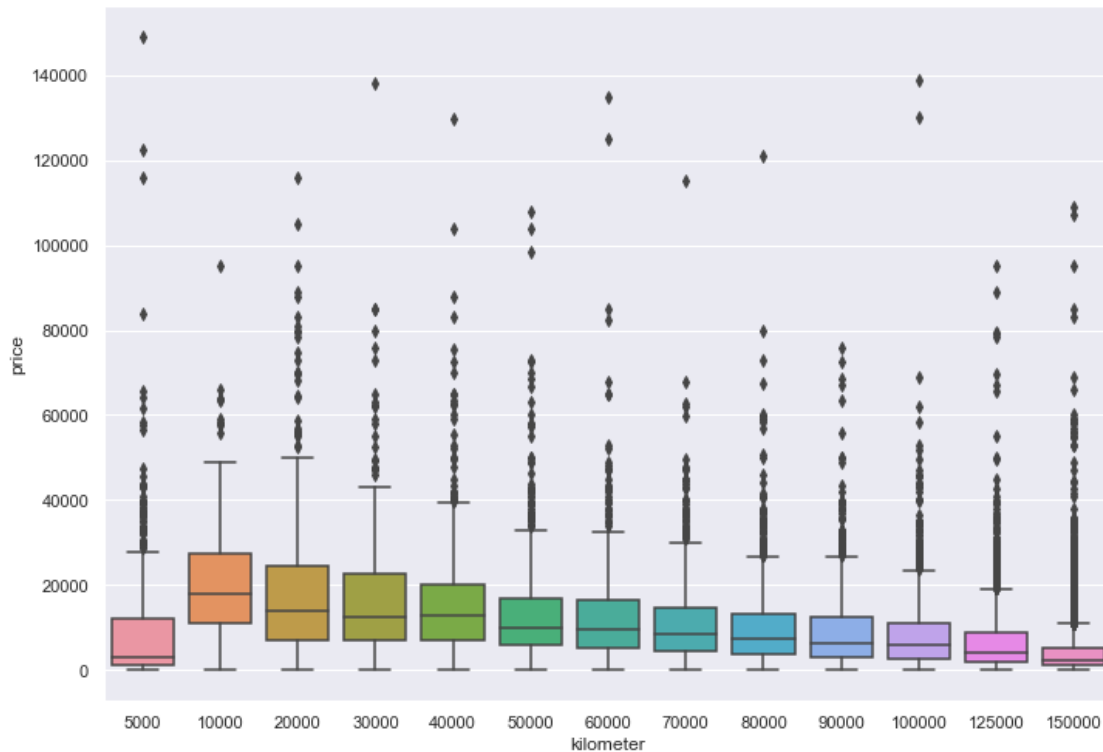
```
[40]: # powerPS vs price
sns.regplot(x='powerPS', y='price', scatter=True,
            fit_reg=False, data=cars)
#as the power PS increases, price also increases
```

```
[40]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdd296fd0>
```



```
[41]: #Price Vs Kilometer
sns.boxplot(x='kilometer',y='price',data=cars)
#As running of a vehicle increases, price gets reduce except for vehicles below
→5000 km running
#cars within 5000km running having low price because of high age
```

```
[41]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdce94f60>
```



```
[42]: cars_age=cars[(cars['kilometer']<= 5000)]
print(cars_age.shape)
cars_age.describe()
# The Average Age of 479 cars is same as the average age of 42000 cars
#which means cars which have running below 5000km have high aged cars due to
→which price is lower for these cars
```

(479, 14)

```
[42]:
```

	price	powerPS	kilometer	postalCode	Age
count	479.000	479.000	479.000	479.000	479.000
mean	9950.616	124.649	5000.000	49691.161	14.844
std	15939.343	72.072	0.000	25925.783	12.320
min	101.000	14.000	5000.000	1069.000	0.170
25%	1100.000	75.000	5000.000	28540.500	3.750
50%	3000.000	110.000	5000.000	47269.000	15.170
75%	11949.500	150.000	5000.000	70702.000	20.500
max	149000.000	500.000	5000.000	99817.000	63.580

```
[43]: cars_age1=cars[(cars['kilometer'] > 5000)]
print(cars_age1.shape)
cars_age1.describe()
```

(42376, 14)

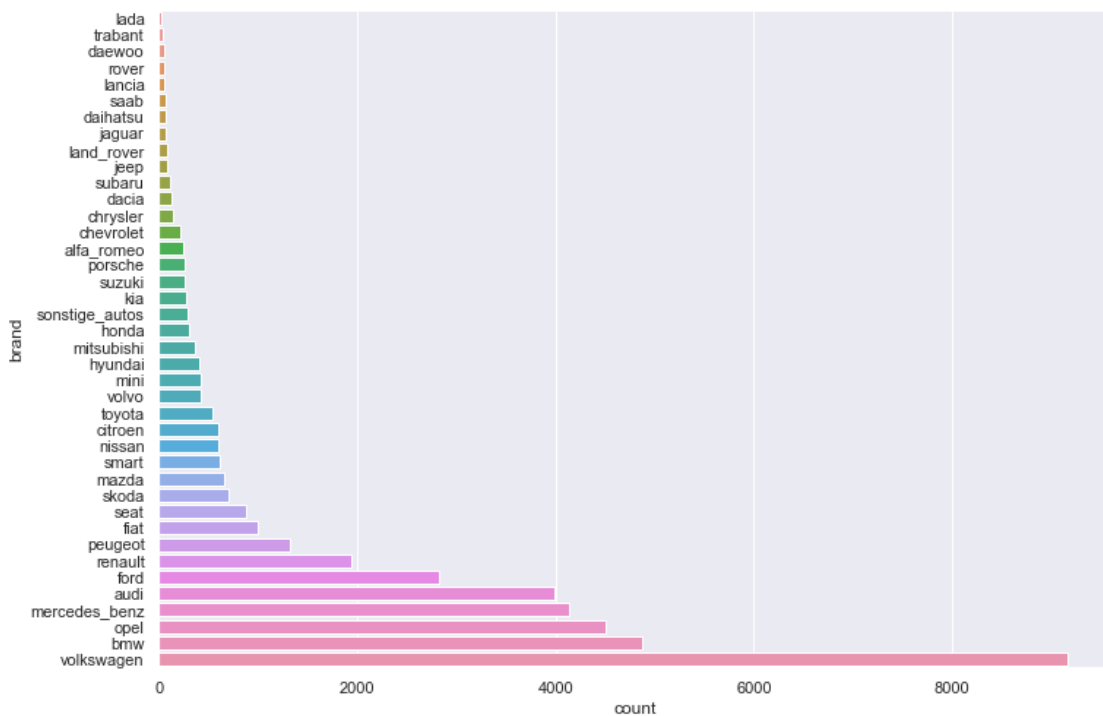

```
[43]:
```

	price	powerPS	kilometer	postalCode	Age
count	42376.000	42376.000	42376.000	42376.000	42376.000
mean	6089.797	126.070	127188.149	51576.149	14.872
std	7796.804	60.382	37106.044	25717.238	7.008
min	100.000	10.000	10000.000	1067.000	0.000
25%	1450.000	80.000	125000.000	31275.000	10.330
50%	3499.000	116.000	150000.000	50674.000	14.830
75%	7899.000	150.000	150000.000	72414.000	19.170
max	139000.000	500.000	150000.000	99998.000	67.750

```
[44]: #cars[(cars['kilometer'] <= 5000) & (cars['Age'] > 10)]
# 293 cars among 500 cars have low price despite running lower than 5000 km but
→age is high due to which price is low
```

```
[45]: sns.countplot(y='brand',data=cars,order=cars['brand'] .
→value_counts(ascending=True).index)
# Volkswagen is highest sold brand and lada is least sold brand
```

```
[45]: <matplotlib.axes._subplots.AxesSubplot at 0x15fdcacce80>
```



```
[46]: pd.crosstab(cars['brand'],columns='count',normalize=True,colnames=['%']) .
→sort_values(by='count',ascending=False)*100
```

```
[46]: %          count
brand
volkswagen    21.379
```

bmw	11.383
opel	10.498
mercedes_benz	9.665
audi	9.317
ford	6.583
renault	4.539
peugeot	3.089
fiat	2.324
seat	2.070
skoda	1.631
mazda	1.549
smart	1.454
nissan	1.402
citroen	1.395
toyota	1.279
mini	1.001
volvo	1.001
hyundai	0.947
mitsubishi	0.838
honda	0.700
sonstige_autos	0.698
kia	0.644
suzuki	0.616
porsche	0.607
alfa_romeo	0.572
chevrolet	0.497
chrysler	0.352
dacia	0.287
subaru	0.261
jeep	0.212
land_rover	0.189
jaguar	0.182
daihatsu	0.156
saab	0.152
lancia	0.131
daewoo	0.124
rover	0.124
trabant	0.100
lada	0.051

```
[47]: print("Total Brands:", cars['brand'].unique().size)
      print("Top 10 brands:")
      cars.groupby(['brand',]).size().sort_values(ascending=False).nlargest(10)
      #The top 10 most sought brands among 40 brands in the market
```

Total Brands: 40

Top 10 brands:

```
[47]: brand
      volkswagen      9162
      bmw             4878
      opel            4499
      mercedes_benz   4142
      audi            3993
      ford            2821
      renault         1945
      peugeot         1324
      fiat            996
      seat            887
      dtype: int64
```

```
[48]: #Variable Model
      print("Total Brands:",cars['model'].unique().size)
      cars.groupby(['brand','model']).size().sort_values(ascending=False).nlargest(10)
      #There are 248 models among them 10 most sought model are given below
```

Total Brands: 248

```
[48]: brand      model
      volkswagen  golf      3494
      bmw         3er      2489
      volkswagen  polo      1500
      opel        corsa     1393
               astra      1278
      audi        a4       1235
      volkswagen  passat    1207
      mercedes_benz c_klasse 1046
      bmw         5er      1013
      mercedes_benz e_klasse  908
      dtype: int64
```

```
[49]: luxury_cars=cars[(cars['price']>=40000) & (cars['Age'] >= 20)]
      #working range based on observation
```

```
[50]: summary=luxury_cars.groupby(['brand','model']).size().
      →sort_values(ascending=False)
      summary.loc['Grand Total'] = summary.sum()
      lux_cars=summary.to_frame('count')
      lux_cars
      #Luxury cars with age above 20 years and price above 40000 dollars
```

```
[50]:
      brand      model      count
      porsche     911          23
      mercedes_benz sl           3
      jaguar       others         3
      mercedes_benz others         2
```

ford	mustang	2
alfa_romeo	spider	2
volkswagen	transporter	1
renault	others	1
mercedes_benz	g_klasse	1
	c_klasse	1
fiat	others	1
bmw	m_reihe	1
Grand Total		41

```
[51]: porsche_car=cars[(cars['brand']=='porsche') & (cars['price']>=40000) &
      ↪(cars['Age'] >= 20)]
porsche_car.head(5)
#Porsche 911, a luxury cars, is main vehicle which is having high price despite
↪its increasing age
```

```
[51]:      seller offerType  price  abtest vehicleType  gearbox  powerPS  \
3286  private      offer  139000      test      coupe      manual      408
4190  private      offer   80000  control      cabrio  automatic      286
5092  private      offer   69800      test      coupe      manual      272
5283  private      offer   52890      test      cabrio      manual      131
8650  private      offer   44300      test      coupe      manual      272

      model  kilometer  fuelType  brand  notRepairedDamage  postalCode  Age
3286   911      100000    petrol  porsche                  no      10629  24.000
4190   911       80000    petrol  porsche                  no      50858  23.830
5092   911      125000    petrol  porsche                  no      80802  23.580
5283   911      150000    petrol  porsche                  no      41812  38.580
8650   911      150000    petrol  porsche                  no       9117  23.580
```

```
[52]: luxury_cars.groupby(['vehicleType']).size()
#Coupe and Cabrio are the two most sought vehicle among luxury cars
```

```
[52]: vehicleType
bus      1
cabrio   20
coupe    24
limousine 1
suv      1
dtype: int64
```

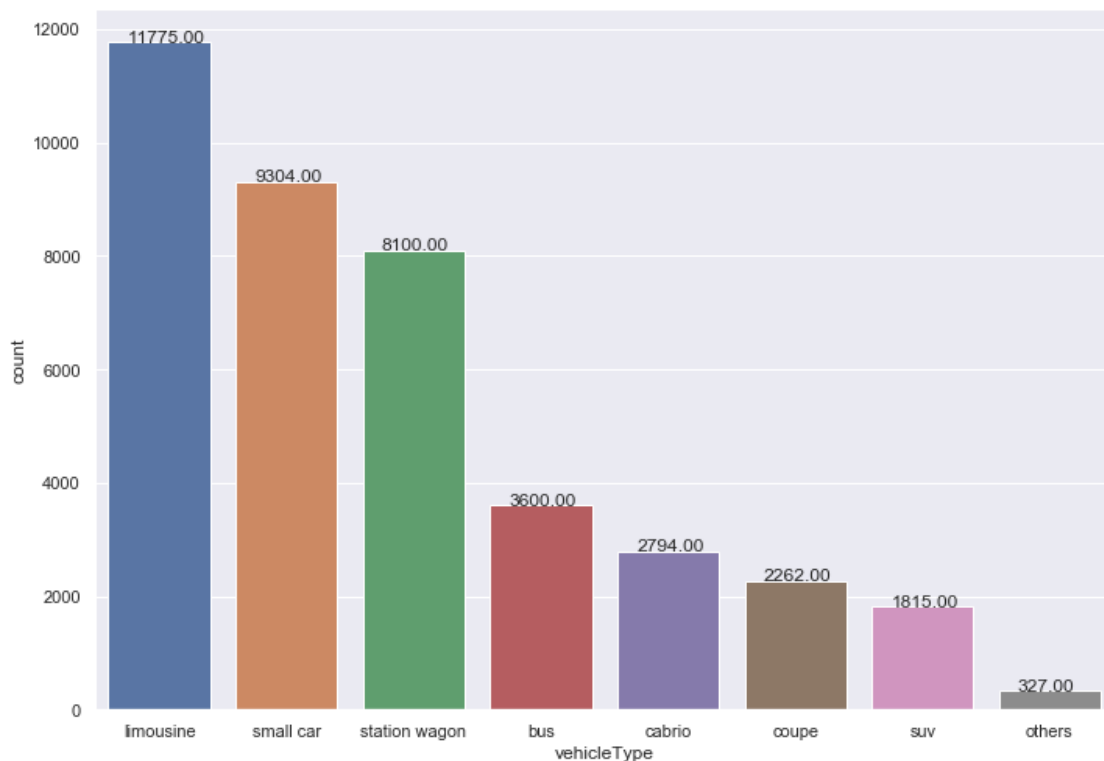
```
[ ]:
```

```
[53]: cars['vehicleType'].value_counts()
```

```
[53]: limousine      11775
small car      9304
station wagon   8100
bus            3600
cabrio         2794
```

```
coupe          2262
suv            1815
others         327
Name: vehicleType, dtype: int64
```

```
[54]: ax=sns.countplot(x='vehicleType',data=cars,order=cars['vehicleType'].
      ↳value_counts().index)
      for p in ax.patches:
          ax.annotate('{:.2f}'.format(p.get_height()), (p.get_x()+0.15, p.
      ↳get_height()+1))
      #Limusine is the highest sold vehicle type and least sold are SUV and others
```



```
[55]: # Variable notRepairedDamage
      # yes- car is damaged but not rectified
      # no- car was damaged but has been rectified
      print(cars['notRepairedDamage'].value_counts())
      print("")
      print(pd.
      ↳crosstab(cars['notRepairedDamage'],columns='count',normalize=True,colnames=['%'])*100)
      # 90% of cars are damage rectified
```

```
no      32574
yes      3994
```

Name: notRepairedDamage, dtype: int64

%	count
notRepairedDamage	
no	89.078
yes	10.922

```
[56]: print(cars['fuelType'].value_counts())
print("")
print(pd.
      ↪crosstab(cars['fuelType'],columns='count',normalize=True,colnames=['%'])*100)
# majority of cars are petrol engine followed by diesel engine cars
```

petrol	26549
diesel	12896
lpg	690
cng	70
hybrid	36
electro	10
other	6

Name: fuelType, dtype: int64

%	count
fuelType	
cng	0.174
diesel	32.034
electro	0.025
hybrid	0.089
lpg	1.714
other	0.015
petrol	65.949

```
[57]: print(cars['gearbox'].value_counts())
print("")
print(pd.
      ↪crosstab(cars['gearbox'],columns='count',normalize=True,colnames=['%'])*100)
# 77% of cars have manual gear box
```

manual	32650
automatic	9410

Name: gearbox, dtype: int64

%	count
gearbox	
automatic	22.373
manual	77.627