Predicting Prices of Pre-owned Cars - Building Linear Regression and Random Forest Models

July 15, 2024

Harsh Mittal
Predicting Price of Pre-owned Cars

[445]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Setting dimensions for plot

[446]:
```
sns.set(rc={'figure.figsize':(11.7, 8.27)})
```

Reading CSV file

[447]:
```
cars_data = pd.read_csv('C:/Users/harsh.hm.mittal/Downloads/cars_sampled.csv')
```

Creating copy

[448]:
```
cars = cars_data.copy()
```

Structure of the dataset

[449]:
```
cars.info()
```

```
<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
```

Summarizing data

[450]:
```
cars.describe()
pd.set_option('display.float_format', lambda x: '%3.3f' % x)
cars.describe()
```

[450]:
```
             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
```

Dropping unwanted columns

[451]:
```
col = ['name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen']
cars = cars.drop(columns = col, axis = 1)
```

Rmoving duplicate records

[452]:
```
cars.drop_duplicates(keep = 'first', inplace = True)
```

Data cleaning

No. of missing values in each column

[453]:
```
cars.isnull().sum()
```

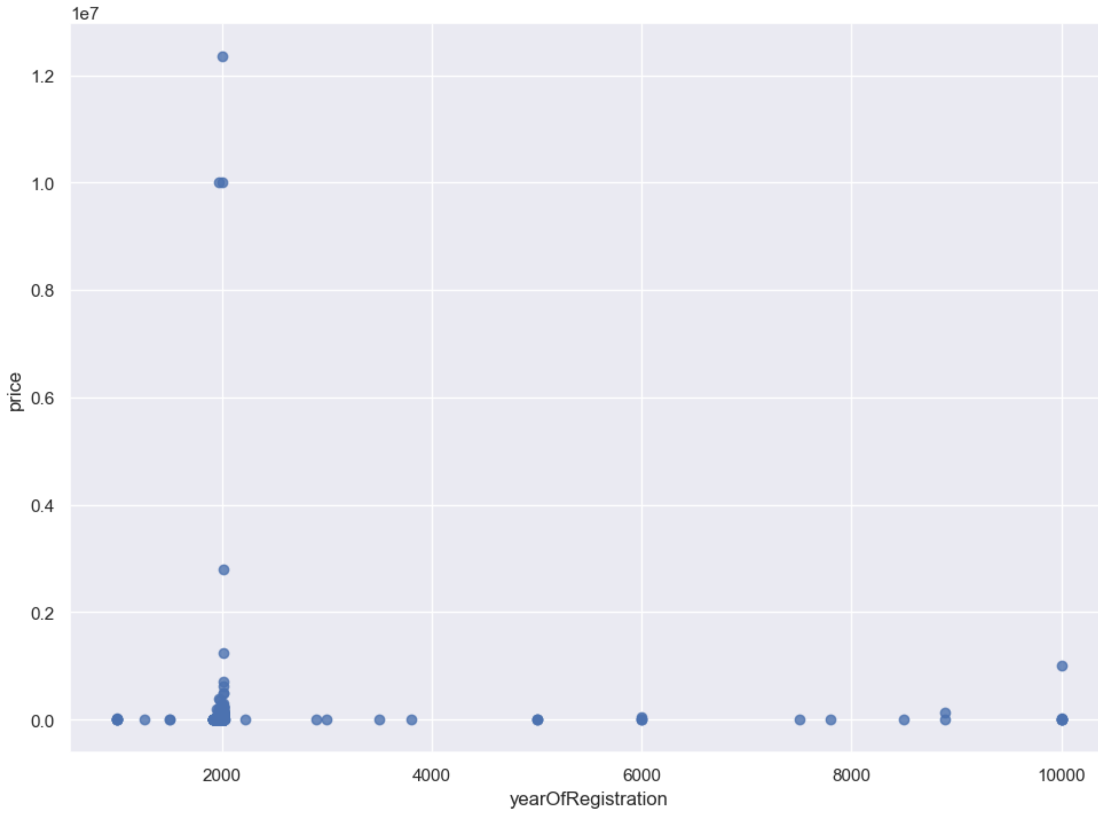
[453]:
```
seller                  0
offerType               0
price                   0
abtest                  0
vehicleType          5152
yearOfRegistration      0
gearbox              2765
powerPS                 0
model                2730
kilometer               0
monthOfRegistration     0
fuelType             4467
brand                   0
notRepairedDamage    9640
dtype: int64
```

Vairable yearOfRegistration

[454]:
```
yearwise_count = cars['yearOfRegistration'].value_counts().sort_index()
print(sum(cars['yearOfRegistration'] > 2018))
print(sum(cars['yearOfRegistration'] < 1950))
sns.regplot(x='yearOfRegistration', y='price', scatter=True, fit_reg=False,␣
 ↪data=cars)
```

```
26
38
```

[454]:
```
<AxesSubplot:xlabel='yearOfRegistration', ylabel='price'>
```



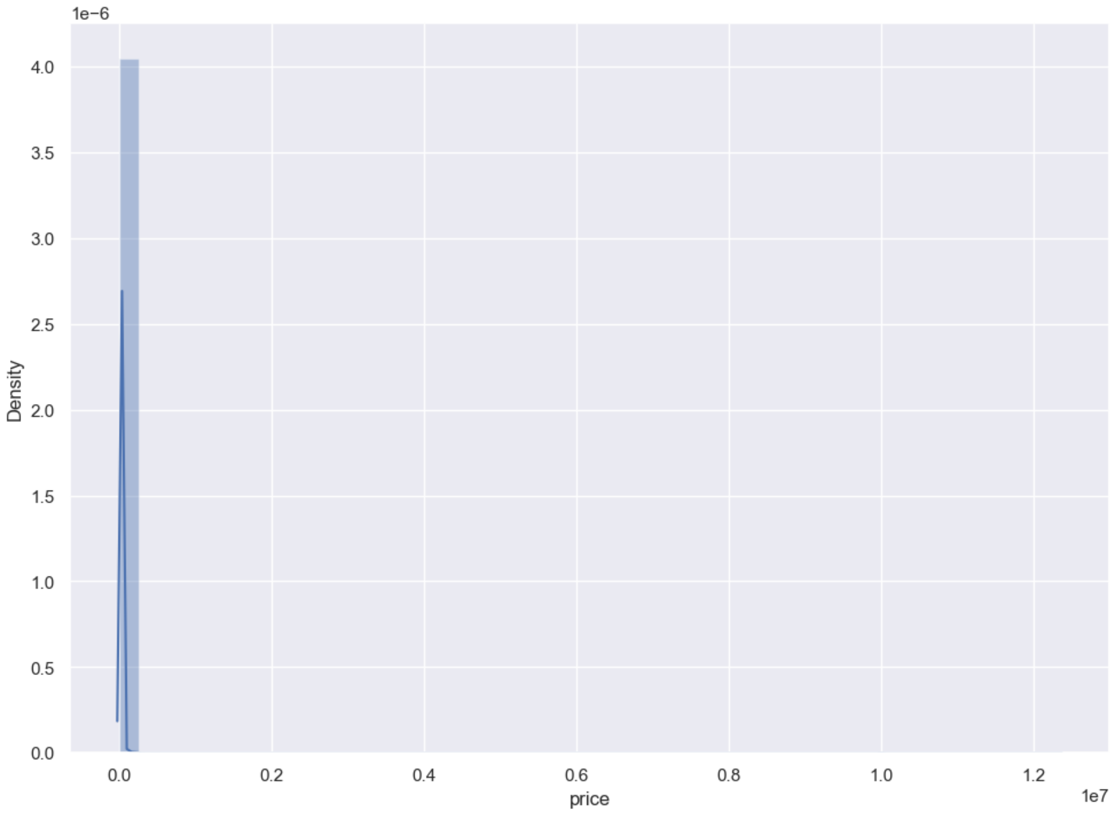
Working range - 1950 and 2018

Variable price

[455]:
```
price_count = cars['price'].value_counts().sort_index()
print(price_count)
sns.distplot(cars['price'])
```

```
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
```

[455]: `<AxesSubplot:xlabel='price', ylabel='Density'>`

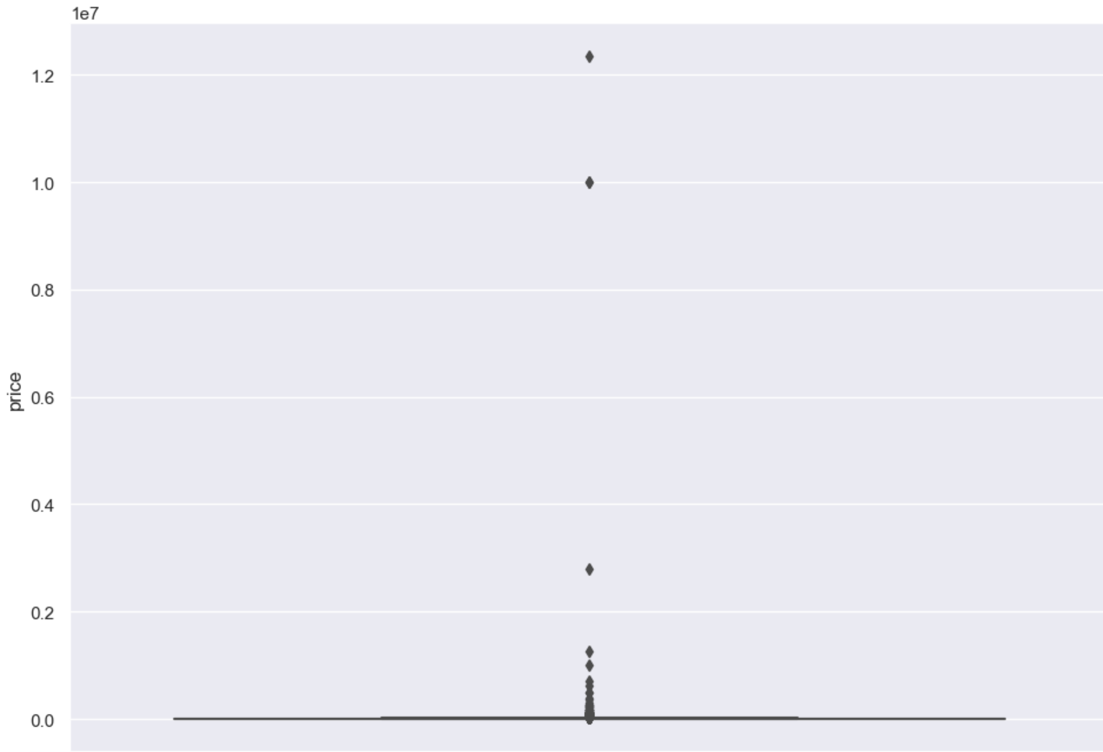


[456]:
```
cars['price'].describe()
```

[456]:
```
count       49531.000
mean         6567.220
std         86222.378
min             0.000
25%          1150.000
50%          2950.000
75%          7100.000
max      12345678.000
Name: price, dtype: float64
```

[457]:
```
sns.boxplot(y=cars['price'])
print(sum(cars['price'] > 150000))
print(sum(cars['price'] < 100))
```

```
34
1748
```



Working rang - 100 and 150000

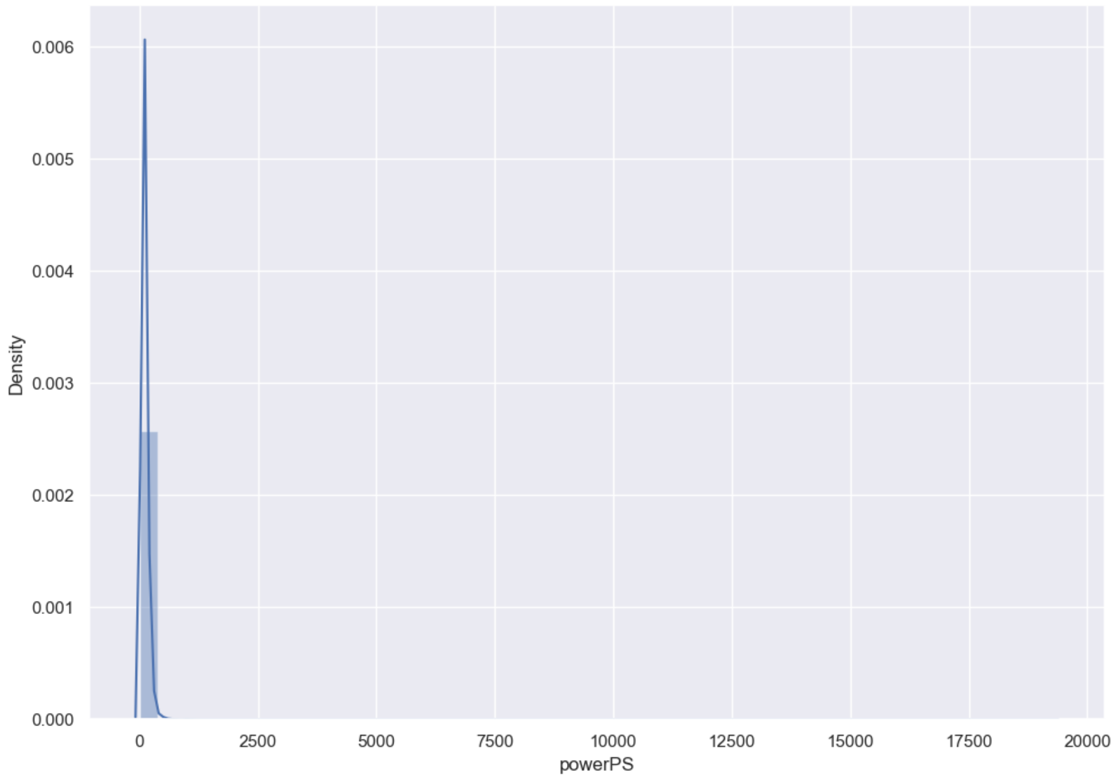
Variable powerPS

[458]:
```
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
```

[459]:
```
sns.distplot(cars['powerPS'])
```

[459]: <AxesSubplot:xlabel='powerPS', ylabel='Density'>

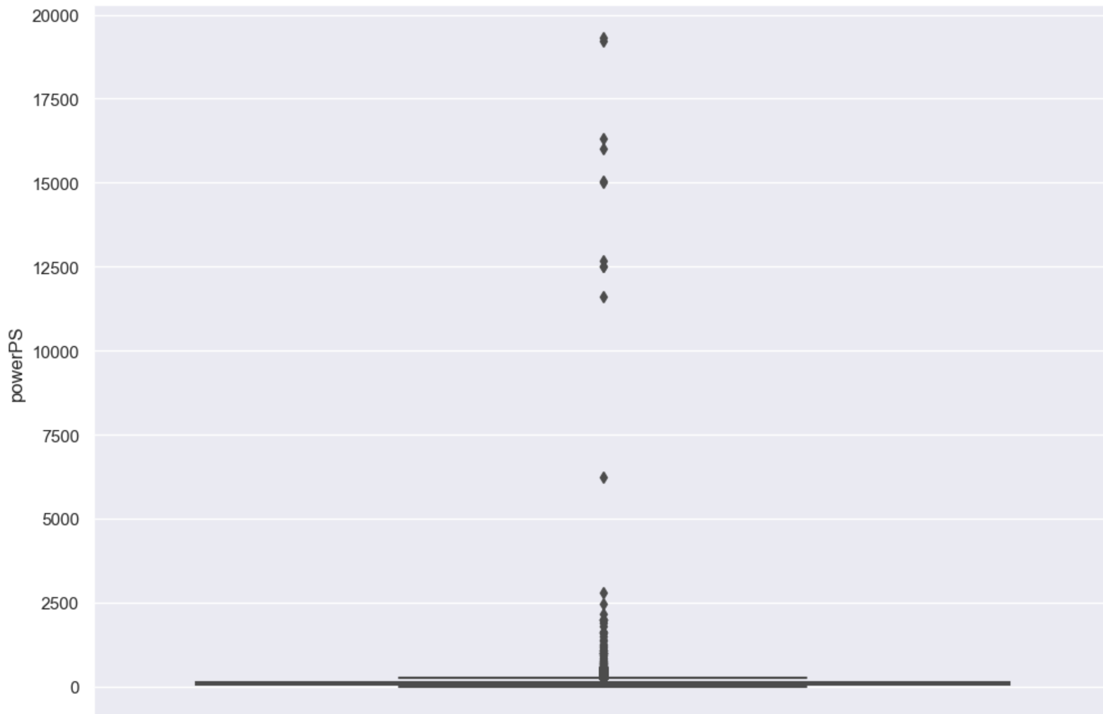


[460]:
```
cars['powerPS'].describe()
```

[460]:
```
count    49531.000
mean       116.501
std        231.536
min          0.000
25%         69.000
50%        105.000
75%        150.000
max      19312.000
Name: powerPS, dtype: float64
```

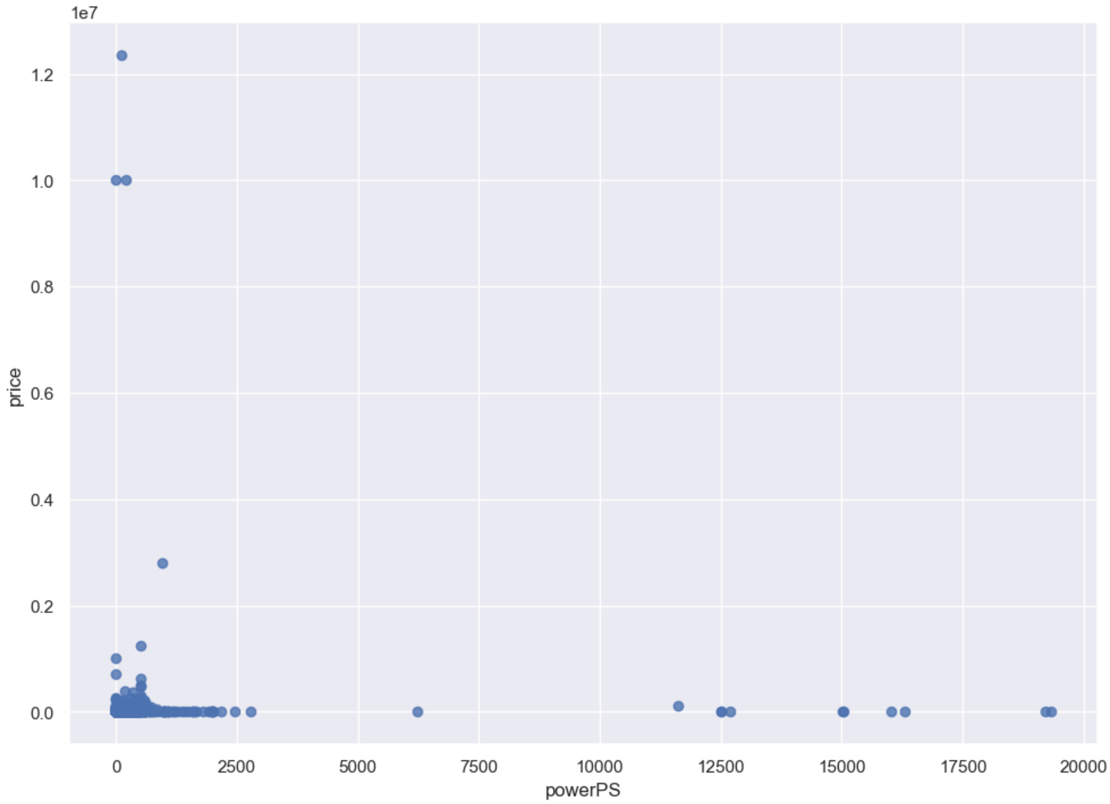
[461]:
```
sns.boxplot(y=cars['powerPS'])
```

[461]: <AxesSubplot:ylabel='powerPS'>



[462]:
```
sns.regplot(x='powerPS', y='price', scatter=True, fit_reg=False, data=cars)
```

[462]: <AxesSubplot:xlabel='powerPS', ylabel='price'>



[463]:
```
print(sum(cars['powerPS'] > 500))
print(sum(cars['powerPS'] < 10))
```

```
115
5565
```

Working range - 10 and 500

Working range for yearOfRegistration - 1950 and 2018
Working range for price - 100 and 150000
Working range for powerPS - 10 and 500

Working range of data

[464]:
```
cars = cars[(cars.yearOfRegistration <= 2018)
            &(cars.yearOfRegistration >= 1950)
            &(cars.price <= 150000)
            &(cars.price >= 100)
            &(cars.powerPS <= 500)
            &(cars.powerPS >= 10)]
```

[465]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 42772 entries, 0 to 50000
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   seller               42772 non-null  object
 1   offerType            42772 non-null  object
 2   price                42772 non-null  int64 
 3   abtest               42772 non-null  object
 4   vehicleType          39896 non-null  object
 5   yearOfRegistration   42772 non-null  int64 
 6   gearbox              41978 non-null  object
 7   powerPS              42772 non-null  int64 
 8   model                41089 non-null  object
 9   kilometer            42772 non-null  int64 
 10  monthOfRegistration  42772 non-null  int64 
 11  fuelType             40175 non-null  object
 12  brand                42772 non-null  object
 13  notRepairedDamage    36495 non-null  object
dtypes: int64(5), object(9)
memory usage: 4.9+ MB
```

Further to simplify - variable reduction
Combining yearOfRegistration and monthOfRegistration

[466]:
```
cars['monthOfRegistration']/=12
```

[467]:
```
cars.head()
```

[467]:
```
    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  
0         bmw               NaN  
1       volvo                no  
2  volkswagen               NaN  
3        seat                no  
4       volvo                no  
```

Creating new variable Age by adding yearOfRegistration and monthOfRegistration

[468]:
```
cars['Age']=(2018-cars['yearOfRegistration'])+cars['monthOfRegistration']
cars['Age'] = round(cars['Age'],2)
cars['Age'].describe()
```

[468]:
```
count    42772.000
mean        14.873
std          7.093
min          0.000
25%         10.330
50%         14.830
75%         19.170
max         67.750
Name: Age, dtype: float64
```

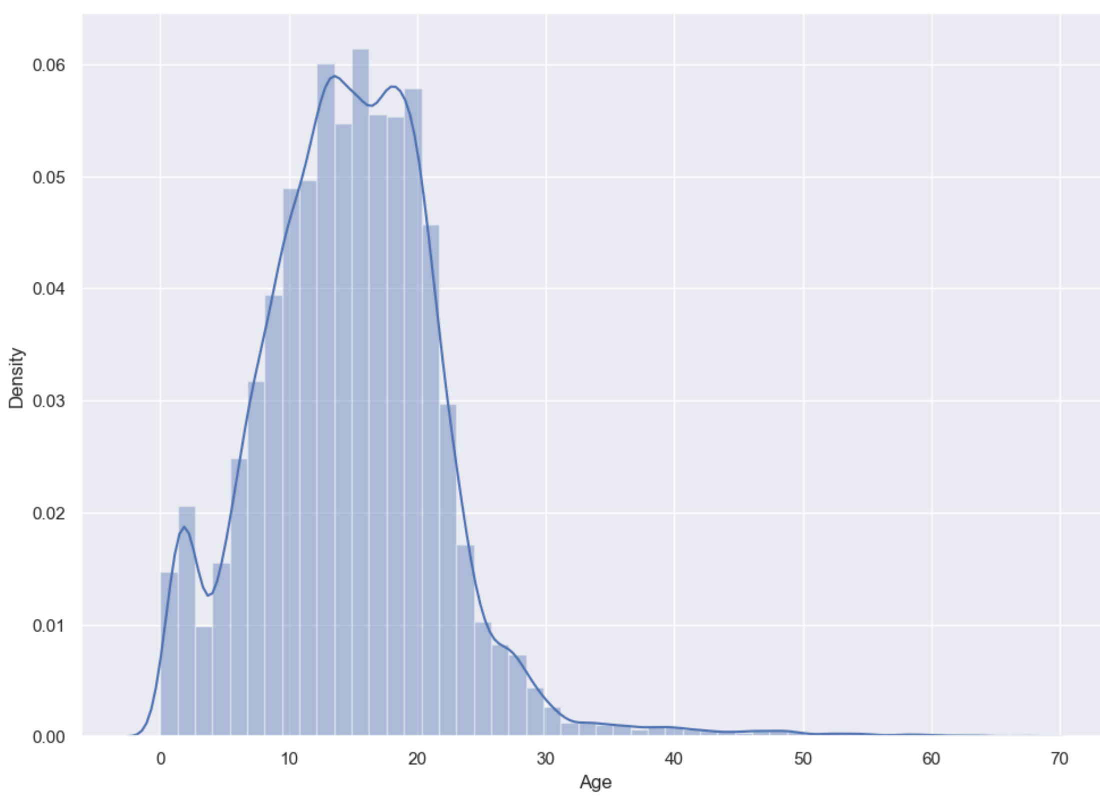
Dropping yearOfRegistration and monthOfRegistration

[469]:
```
cars=cars.drop(columns=['yearOfRegistration', 'monthOfRegistration'], axis = 1)
```

Visualizing parameters
Age

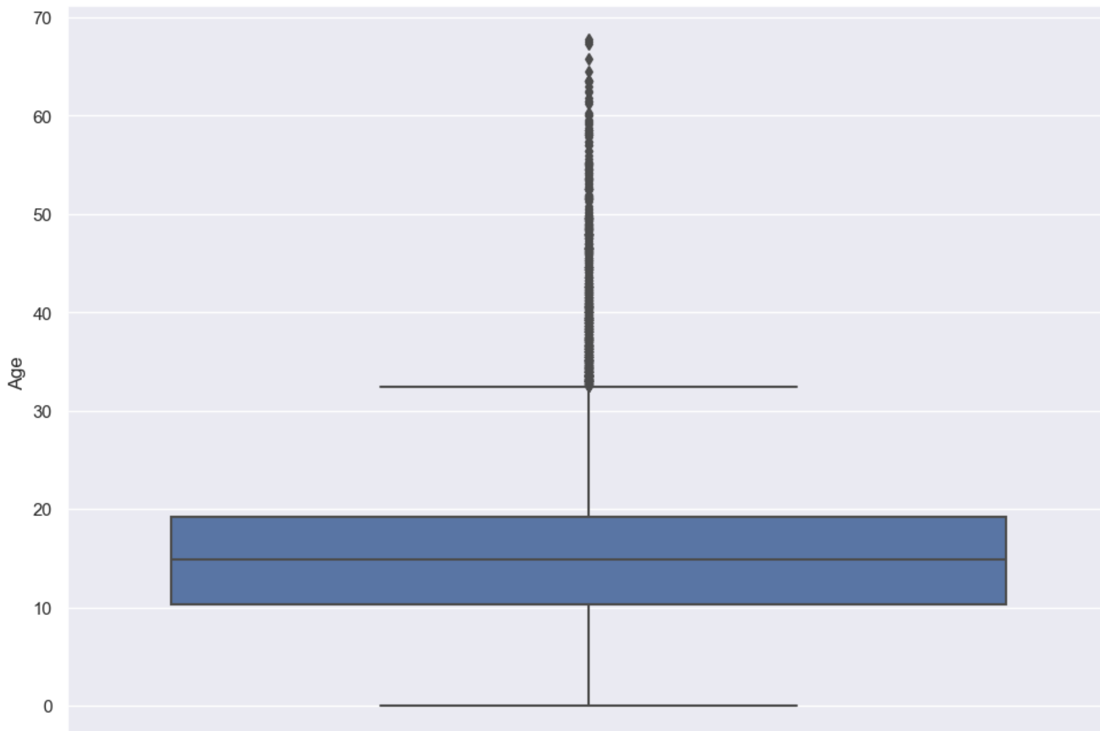
[470]:
```
sns.distplot(cars['Age'])
```

[470]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```



[471]:
```
sns.boxplot(y=cars['Age'])
```

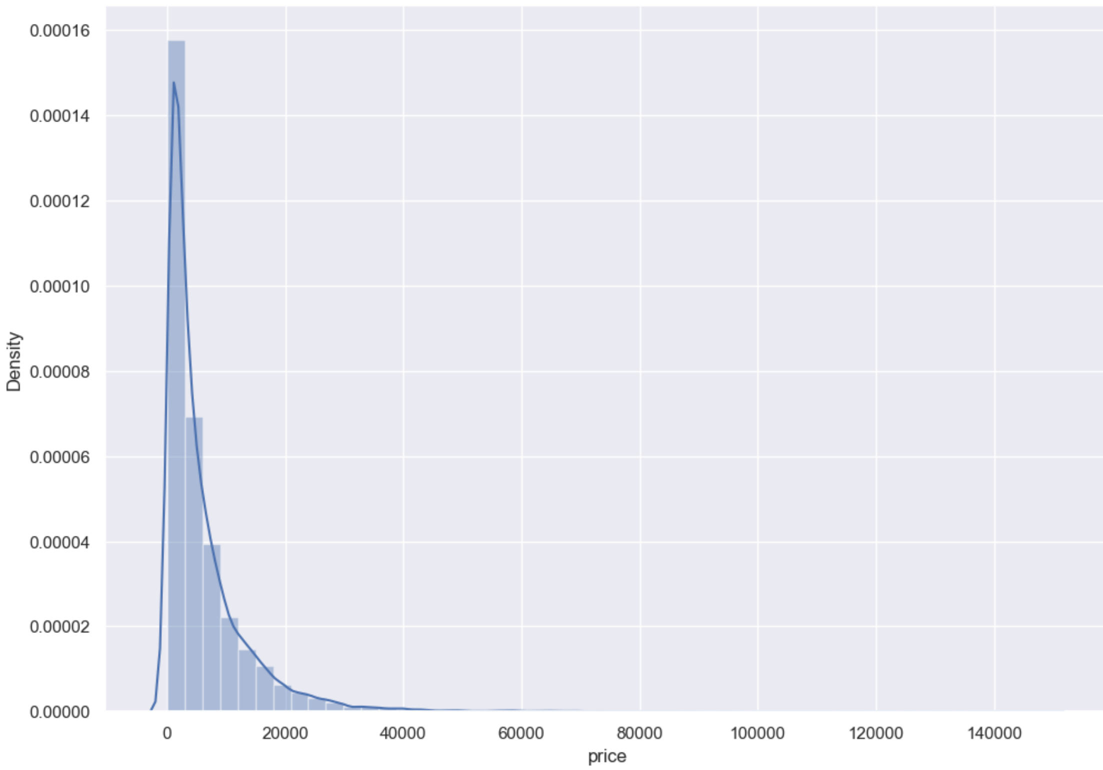
[471]: <AxesSubplot:ylabel='Age'>



Price

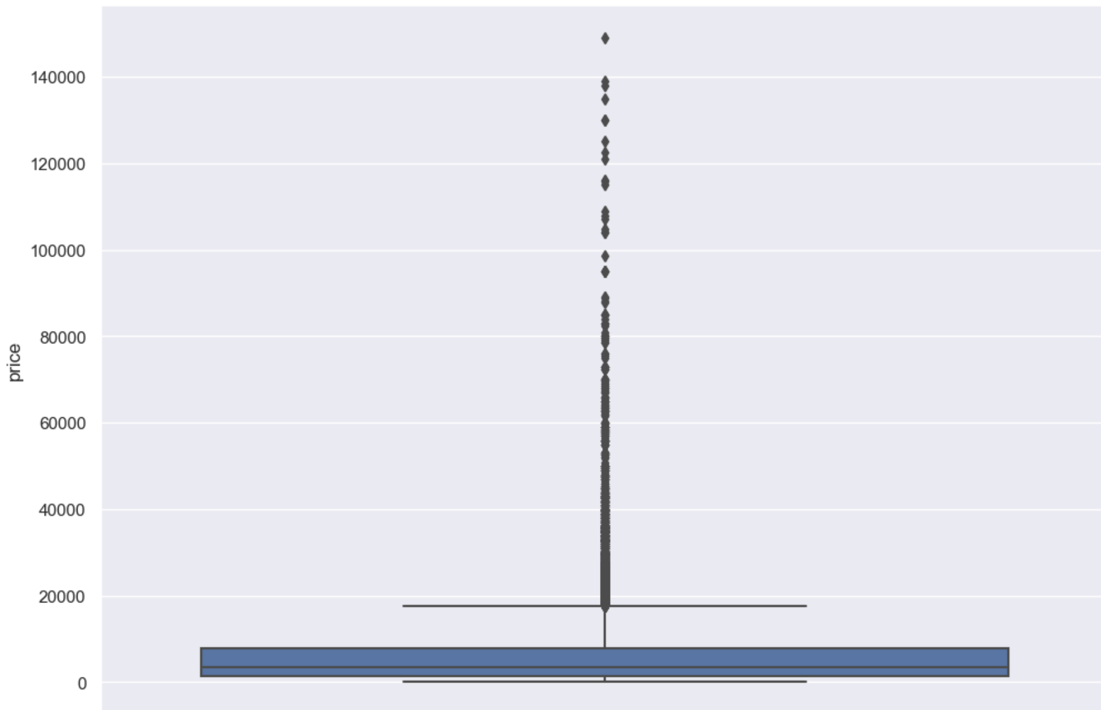
[472]:
```
sns.distplot(cars['price'])
```

[472]: <AxesSubplot:xlabel='price', ylabel='Density'>



[473]:
```
sns.boxplot(y=cars['price'])
```

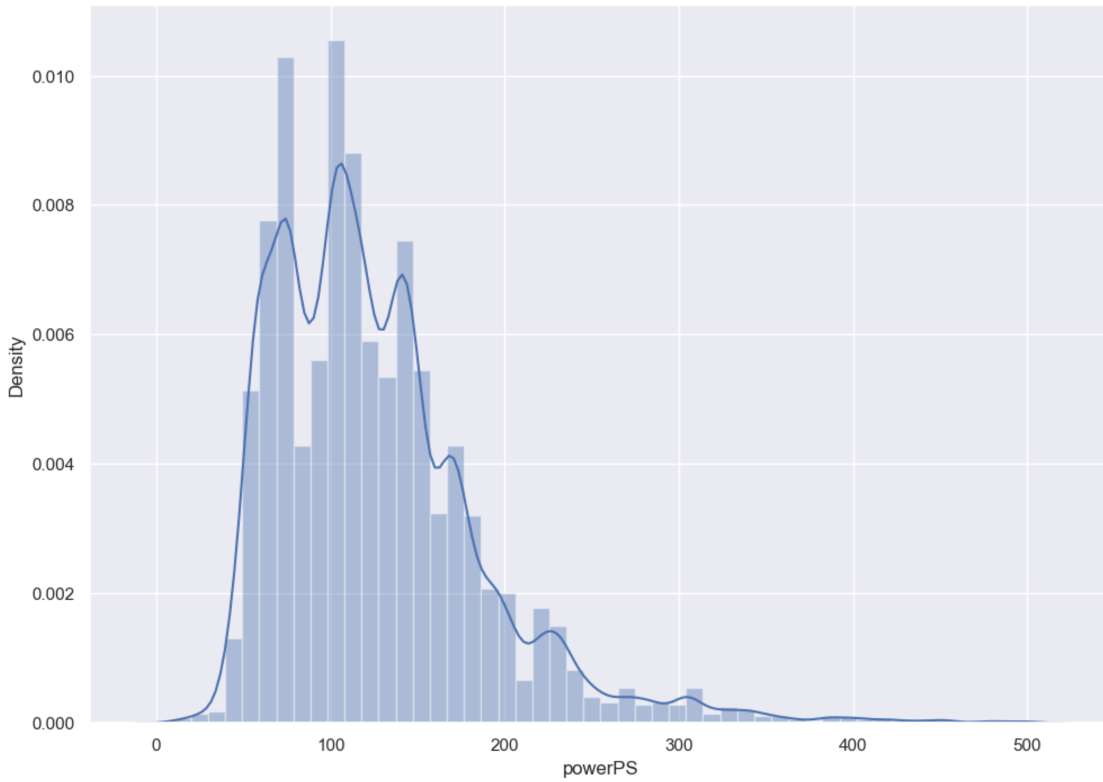
[473]: <AxesSubplot:ylabel='price'>



powerPS

[474]:
```
sns.distplot(cars['powerPS'])
```

[474]: <AxesSubplot:xlabel='powerPS', ylabel='Density'>



[475]:
```
sns.boxplot(y=cars['powerPS'])
```

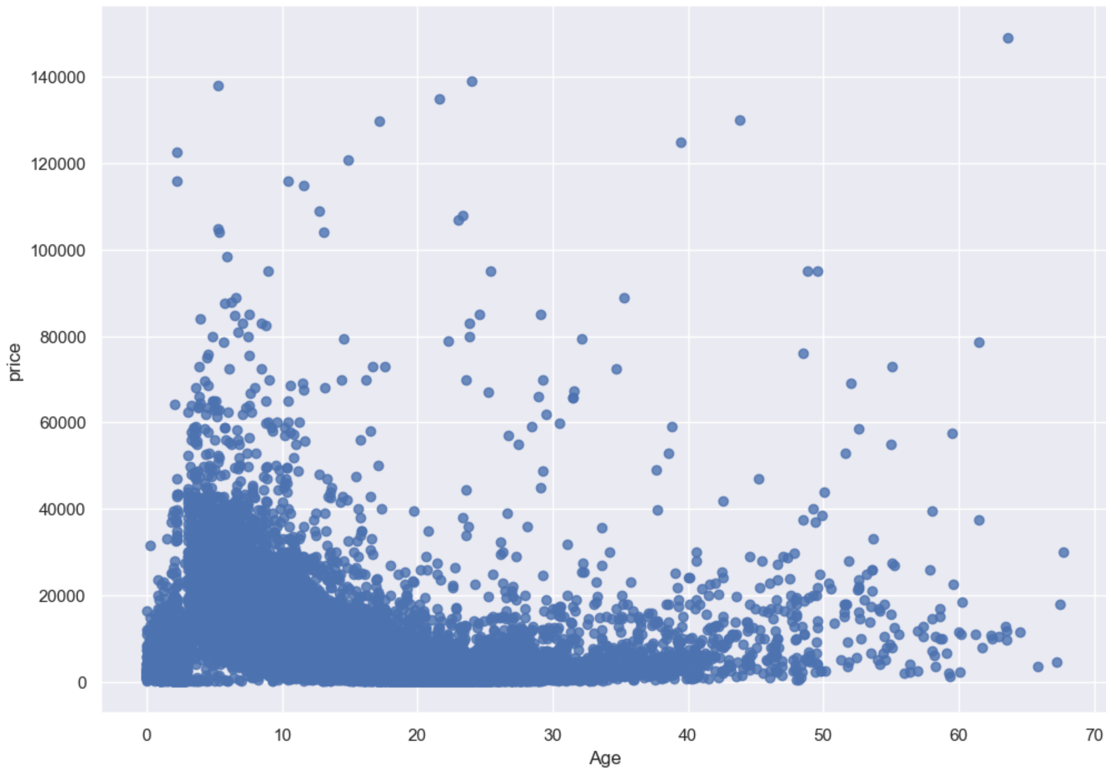
[475]: <AxesSubplot:ylabel='powerPS'>

Visualizing parameters after narrowing working rane

Age vs price

[476]:
```
sns.regplot(x='Age', y='price', scatter=True, fit_reg=False, data=cars)
```

[476]: <AxesSubplot:xlabel='Age', ylabel='price'>

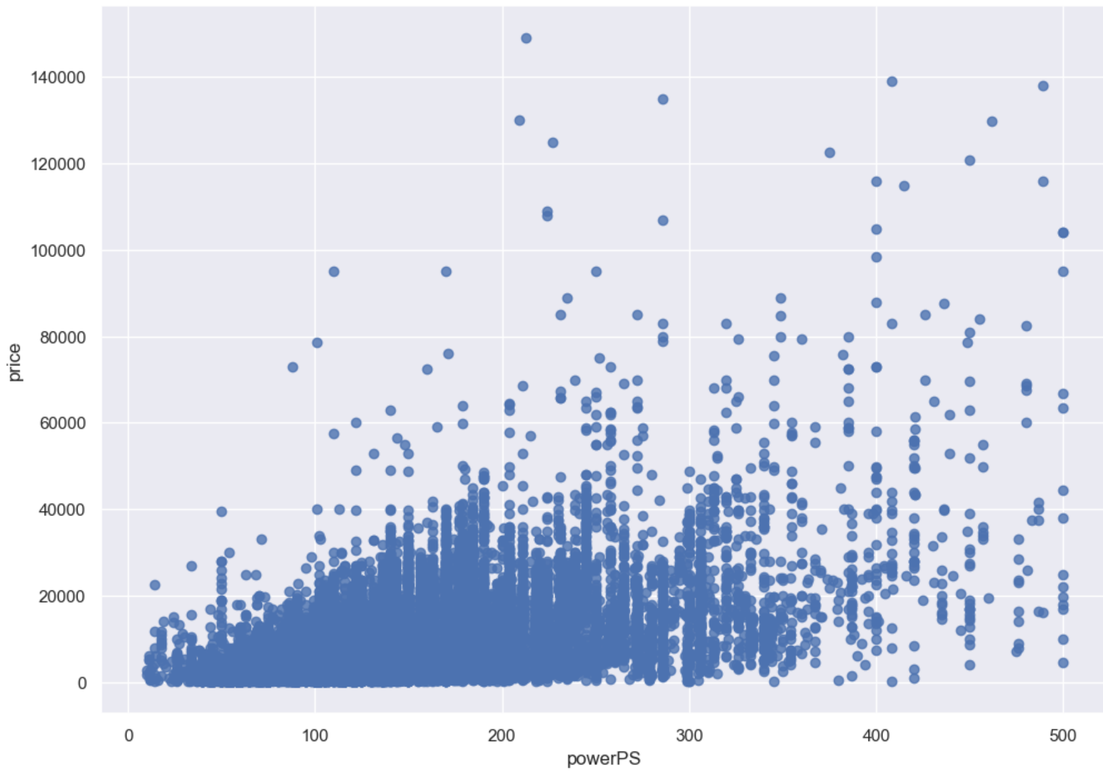


Cars priced higher are newer
WIth increase in age, price decreases
However some cars are priced higher with increase in age

powerPS vs price

[477]:
```
sns.regplot(x='powerPS', y='price', scatter=True, fit_reg=False, data=cars)
```

[477]: <AxesSubplot:xlabel='powerPS', ylabel='price'>



Variable seller

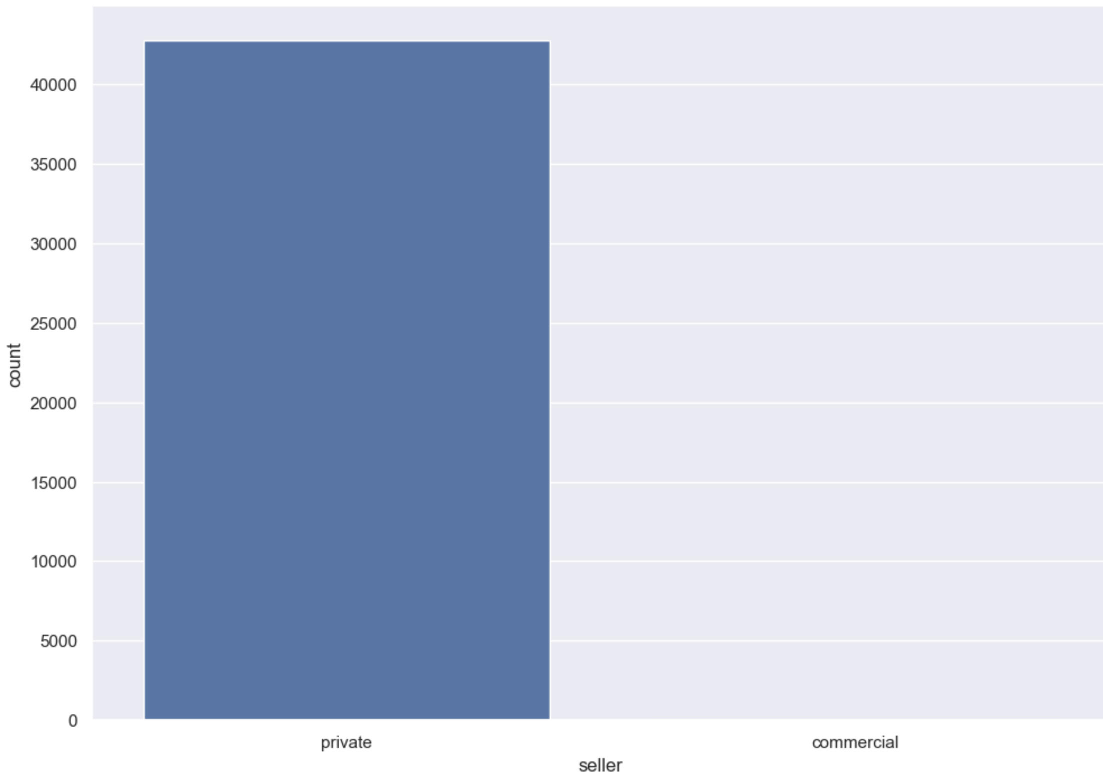
```
[478]: print(cars['seller'].value_counts())
       pd.crosstab(cars['seller'], columns='count',normalize=True)
```

```
private       42771
commercial        1
Name: seller, dtype: int64
```

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

```
[479]: sns.countplot(x= 'seller', data=cars)
```

```
[479]: <AxesSubplot:xlabel='seller', ylabel='count'>
```



Fewer cars have 'commercial' => Insignificant

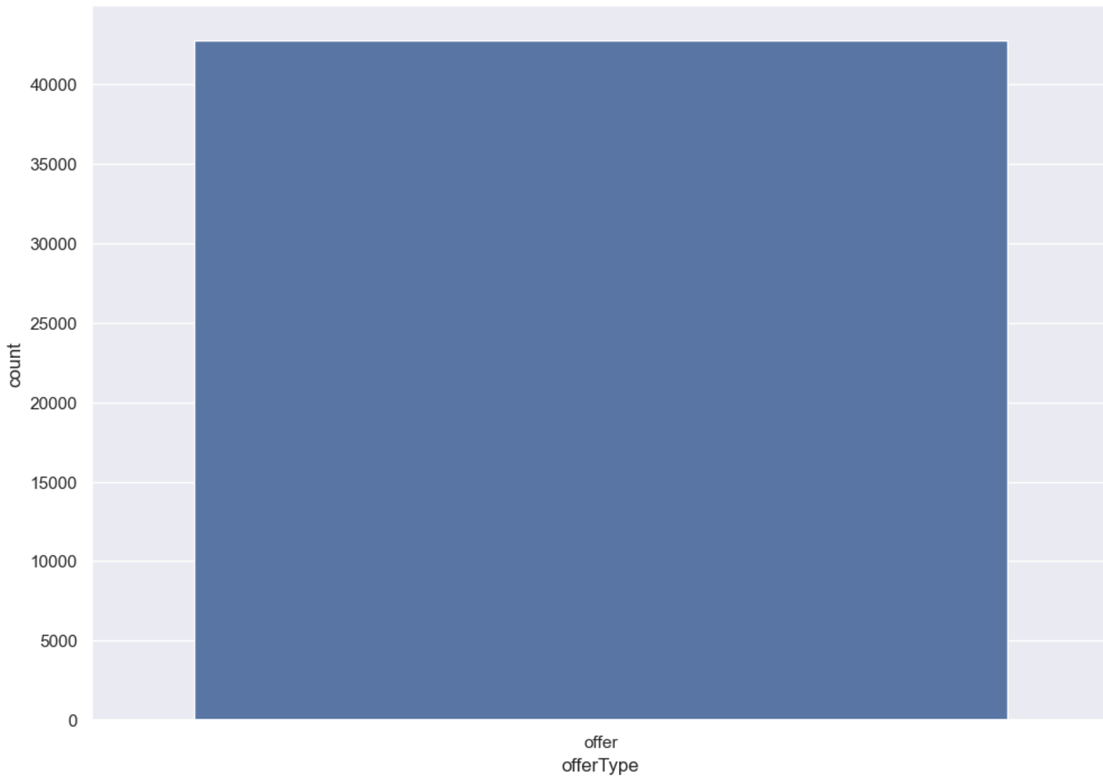
Variable offerType

[480]:
```
cars['offerType'].value_counts()
```

[480]:
```
offer    42772
Name: offerType, dtype: int64
```

[481]:
```
sns.countplot(x='offerType', data=cars)
```

[481]:
```
<AxesSubplot:xlabel='offerType', ylabel='count'>
```



All cars have ‘offer’ => Insignificant

Variable abtest

[482]:
```
cars['abtest'].value_counts()
```

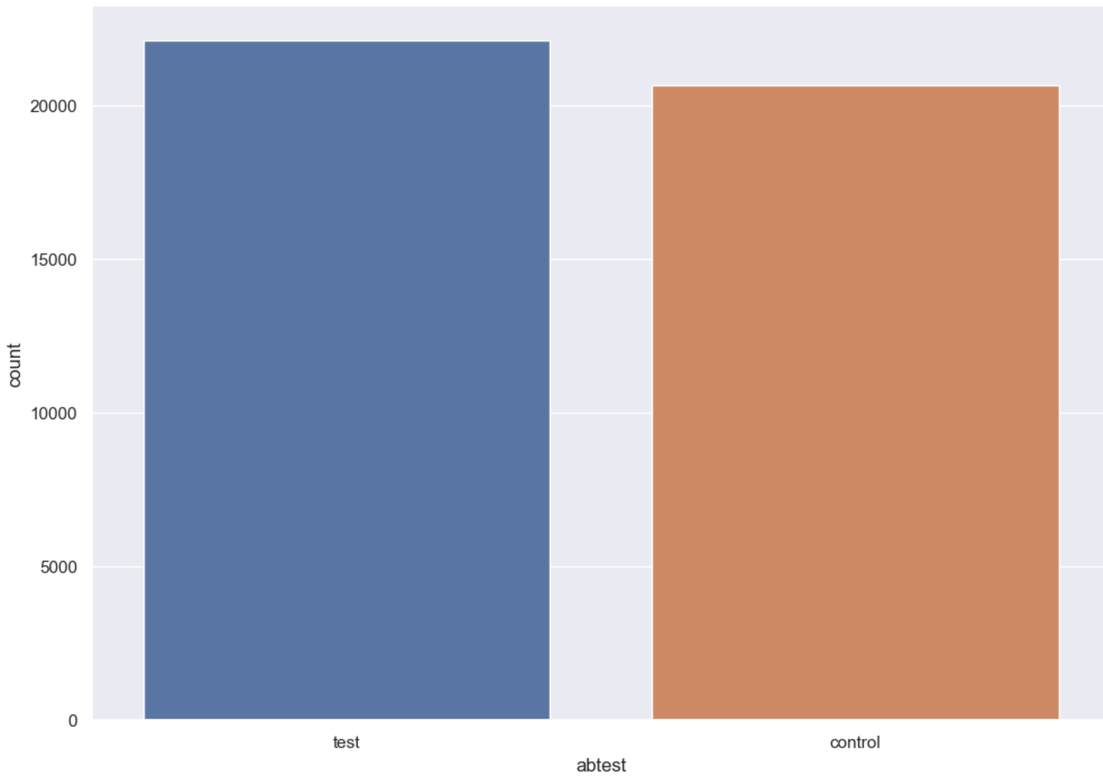
[482]:
```
test       22128
control    20644
Name: abtest, dtype: int64
```

[483]:
```
pd.crosstab(cars['abtest'], columns='count', normalize=True)
```

[483]:
```
col_0    count
abtest
control  0.483
test     0.517
```

[484]:
```
sns.countplot(x='abtest', data=cars)
```

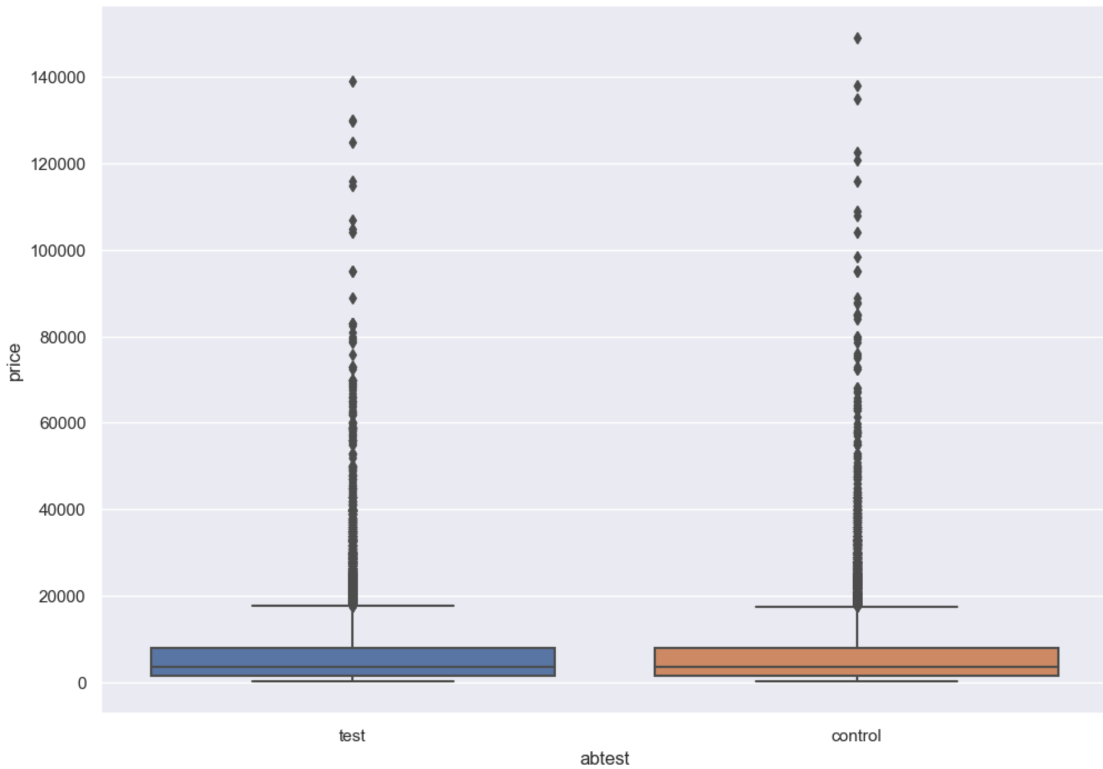
[484]: <AxesSubplot:xlabel='abtest', ylabel='count'>



Equally ditributed

[485]:
```
sns.boxplot(x='abtest', y='price', data=cars)
```

[485]: <AxesSubplot:xlabel='abtest', ylabel='price'>



For every price value there is almost 50-50 distribution
Does not affect price => Insignificant

Variable vehicleType

[486]:
```
cars['vehicleType'].value_counts()
```

[486]:
```
limousine        11746
small car         9285
station wagon     8076
bus               3597
cabrio            2792
coupe             2261
suv               1813
others             326
Name: vehicleType, dtype: int64
```

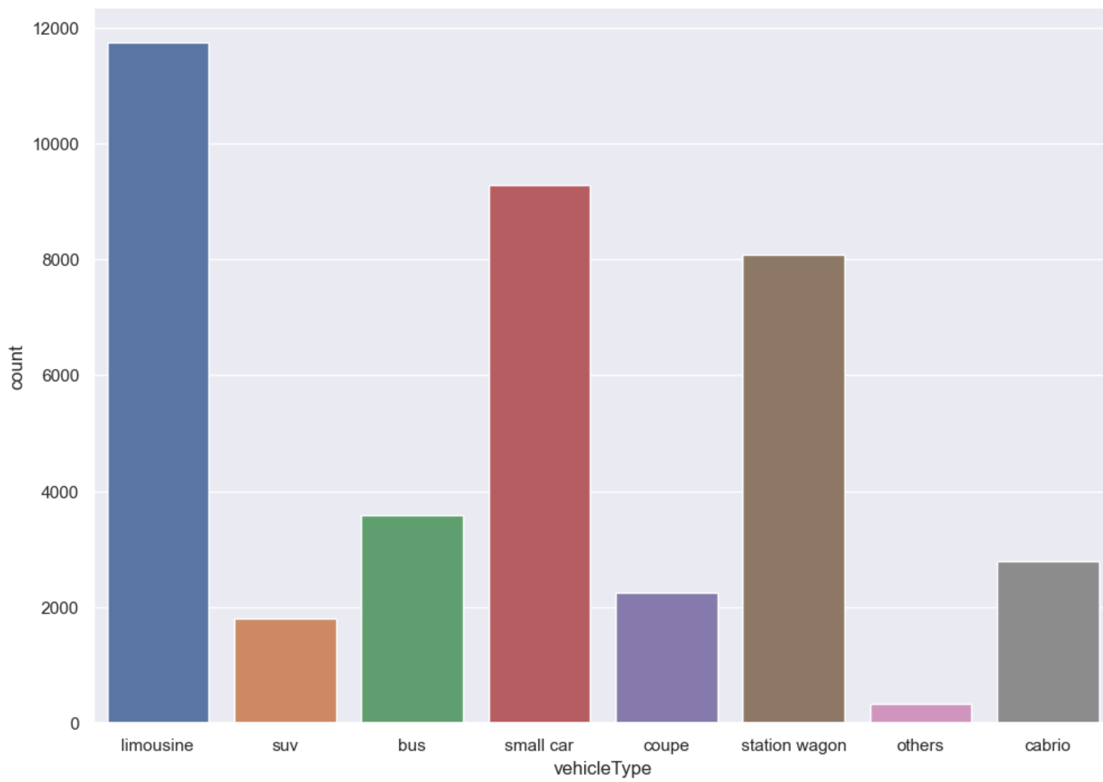
[487]:
```
pd.crosstab(cars['vehicleType'], columns='count', normalize=True)
```

[487]:
```
col_0          count
vehicleType
bus            0.090
cabrio         0.070
```

```
coupe            0.057
limousine        0.294
others           0.008
small car        0.233
station wagon    0.202
suv              0.045
```

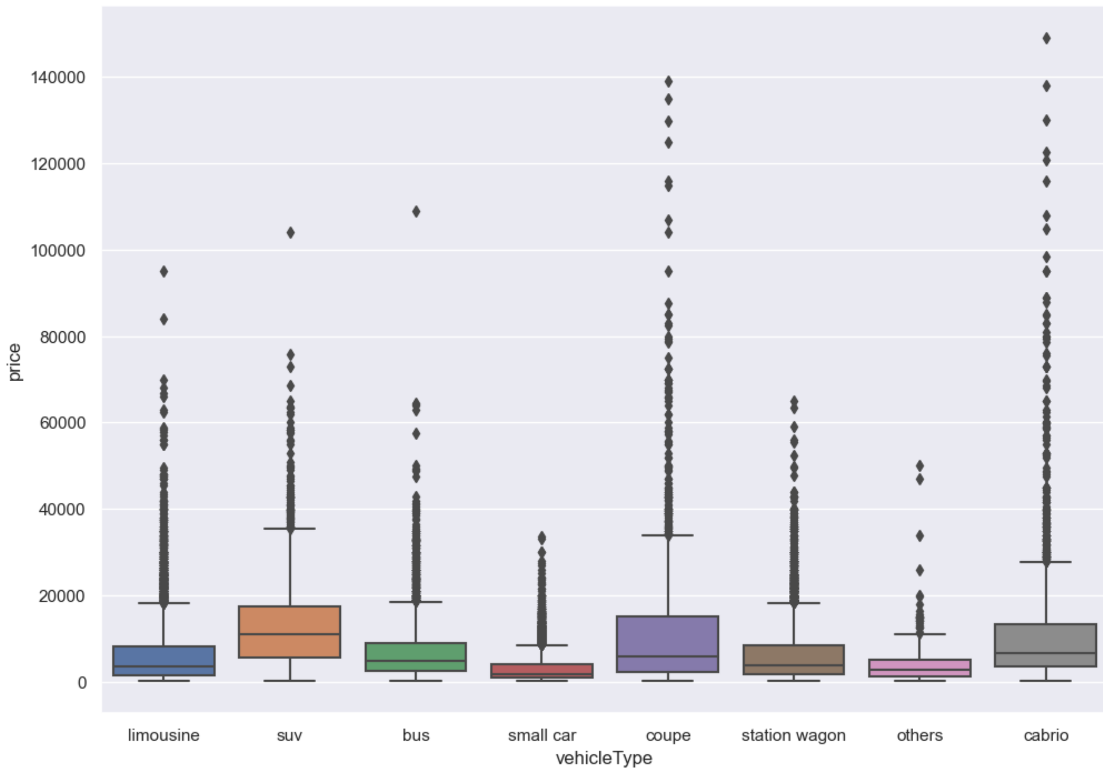
[488]:
```
sns.countplot(x='vehicleType', data=cars)
```

[488]: <AxesSubplot:xlabel='vehicleType', ylabel='count'>



[489]:
```
sns.boxplot(x='vehicleType', y='price', data=cars)
```

[489]: <AxesSubplot:xlabel='vehicleType', ylabel='price'>



8 types - limousine, small cars and station wagons max freq
vehicleType affects price

Variable gearbox

[490]:
```
cars['gearbox'].value_counts()
```

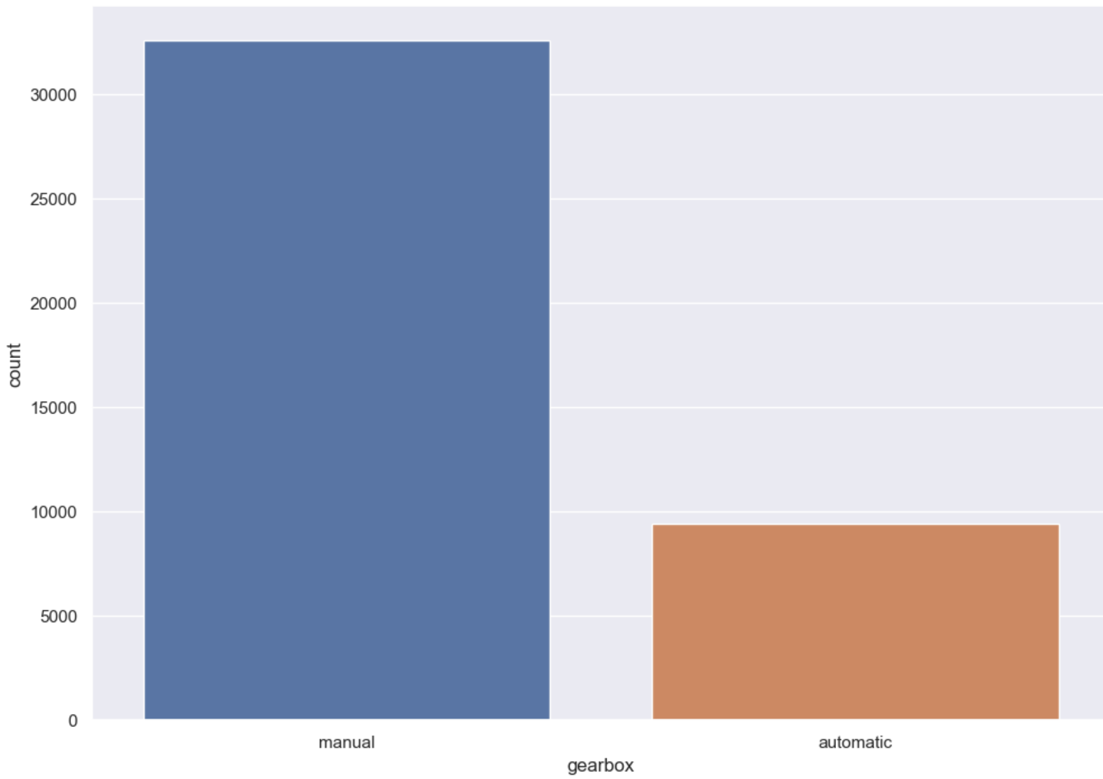
[490]:
```
manual       32582
automatic     9396
Name: gearbox, dtype: int64
```

[491]:
```
pd.crosstab(cars['gearbox'], columns='count', normalize=True)
```

[491]:
```
col_0      count
gearbox
automatic  0.224
manual     0.776
```

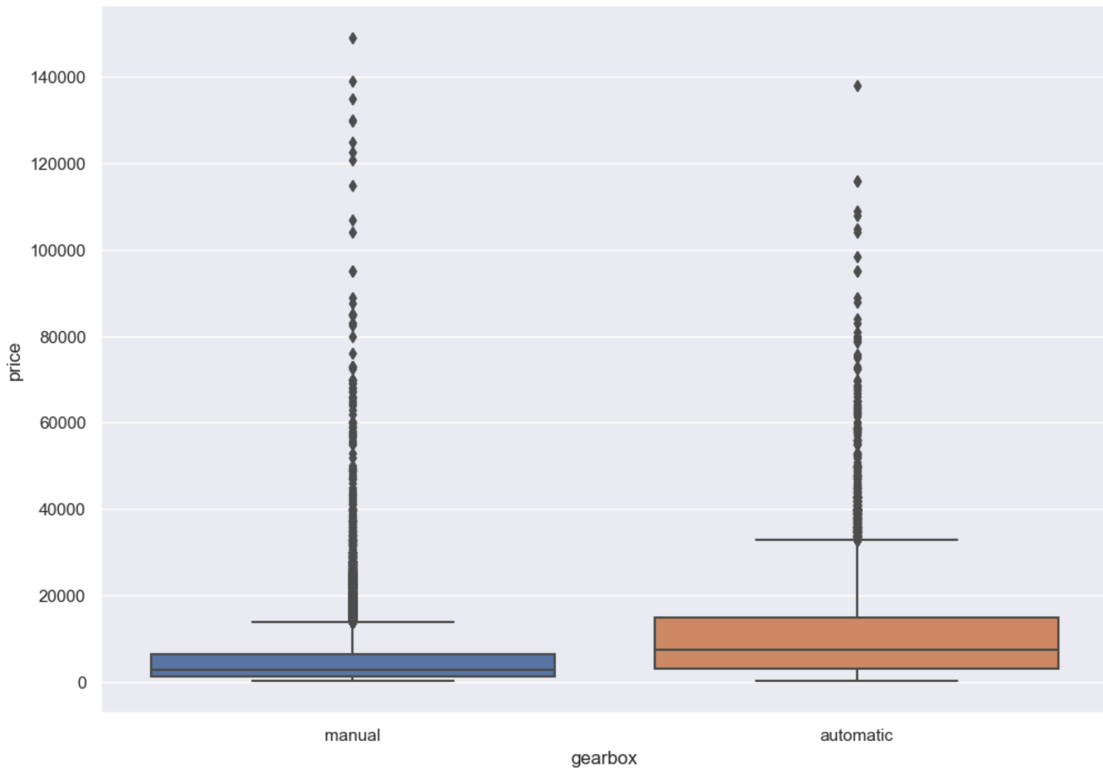
[492]:
```
sns.countplot(x='gearbox', data=cars)
```

[492]:
```
<AxesSubplot:xlabel='gearbox', ylabel='count'>
```



[493]:
```
sns.boxplot(x='gearbox', y='price', data=cars)
```

[493]: <AxesSubplot:xlabel='gearbox', ylabel='price'>



gearbox affects price

Variable model

[494]:
```
cars['model'].value_counts()
```

[494]:
```
golf           3478
others         2900
3er            2482
polo           1500
corsa          1386
               …
b_max             1
serie_3           1
elefantino        1
charade           1
rangerover        1
Name: model, Length: 247, dtype: int64
```

[495]:
```
pd.crosstab(cars['model'], columns='count', normalize=True)
```

[495]:
```
col_0     count
model
```

```
100      0.001
145      0.000
147      0.001
156      0.002
159      0.000
…        …
yaris    0.003
yeti     0.001
ypsilon  0.001
z_reihe  0.003
zafira   0.008

[247 rows x 1 columns]
```

Cars are distributed over many model
Considered in modelling

Variable kilometer

[496]:
```python
cars['kilometer'].value_counts().sort_index()
```

[496]:
```
5000        479
10000       207
20000       651
30000       712
40000       795
50000       932
60000      1101
70000      1182
80000      1378
90000      1484
100000     1824
125000     4597
150000    27430
Name: kilometer, dtype: int64
```

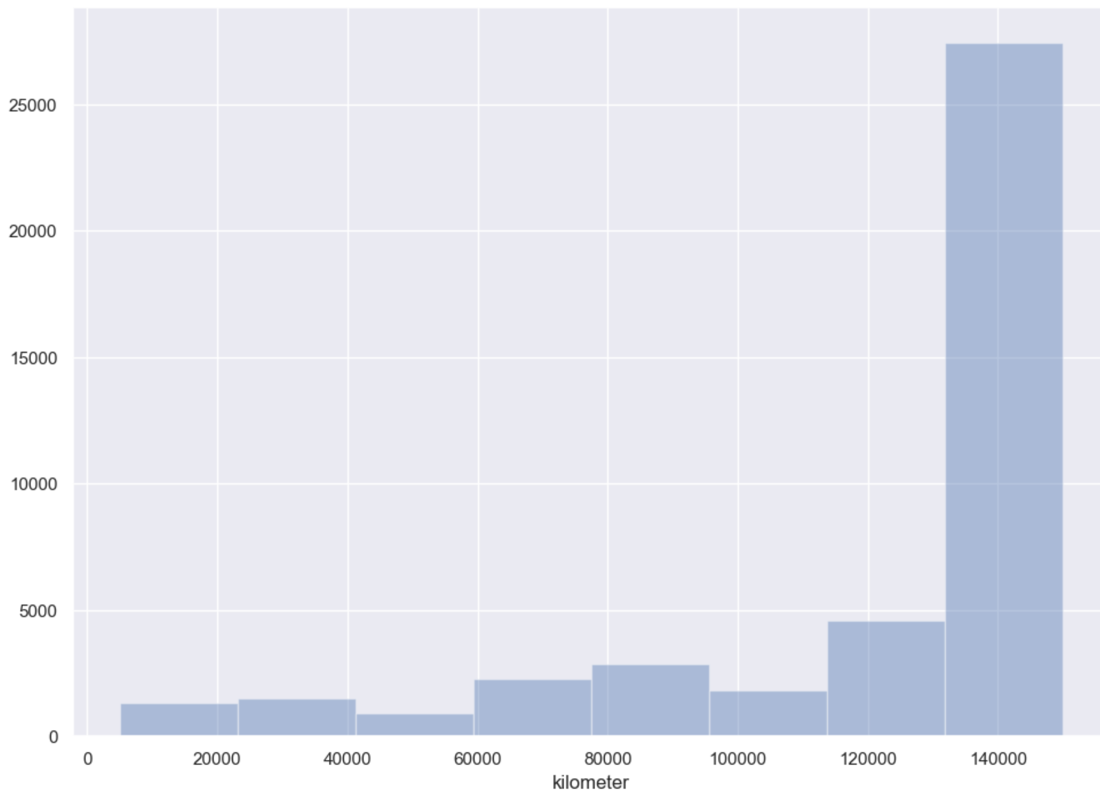
[497]:
```python
pd.crosstab(cars['kilometer'], columns='count', normalize=True)
```

[497]:
```
col_0      count
kilometer
5000       0.011
10000      0.005
20000      0.015
30000      0.017
40000      0.019
50000      0.022
60000      0.026
70000      0.028
```

```
80000      0.032
90000      0.035
100000     0.043
125000     0.107
150000     0.641
```

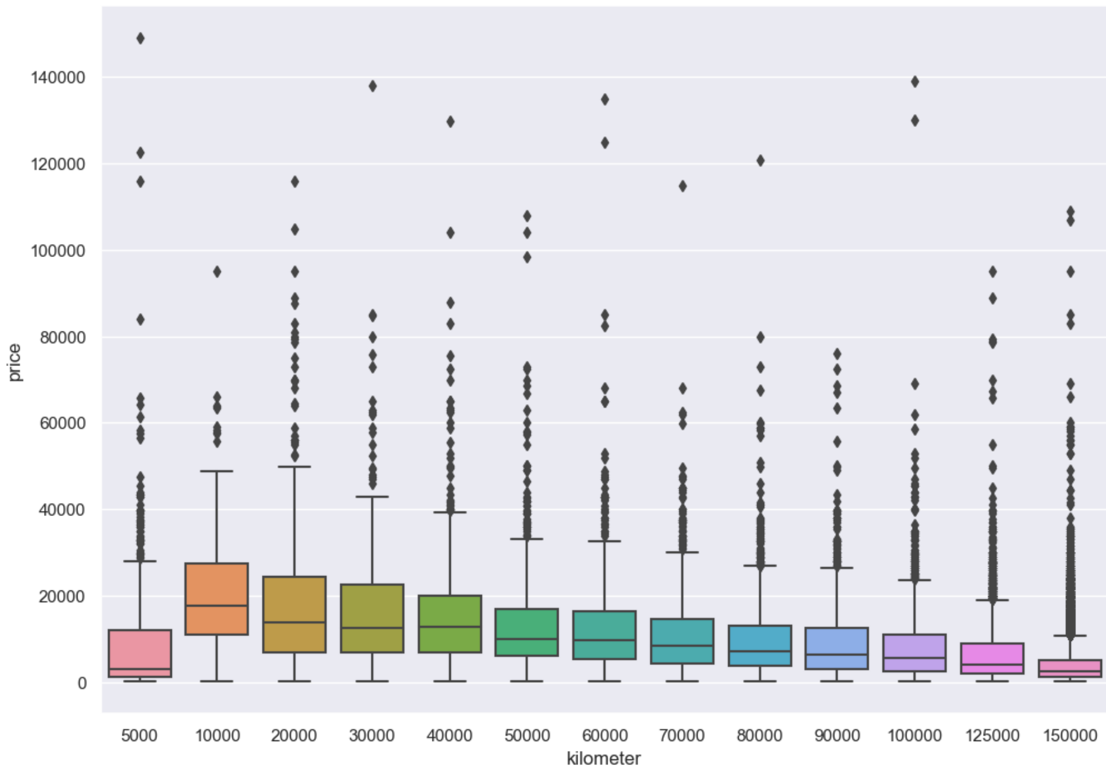
[498]:
```
sns.distplot(cars['kilometer'] , bins=8, kde=False )
```

[498]: <AxesSubplot:xlabel='kilometer'>



[499]:
```
sns.boxplot(x='kilometer', y='price', data=cars)
```

[499]: <AxesSubplot:xlabel='kilometer', ylabel='price'>

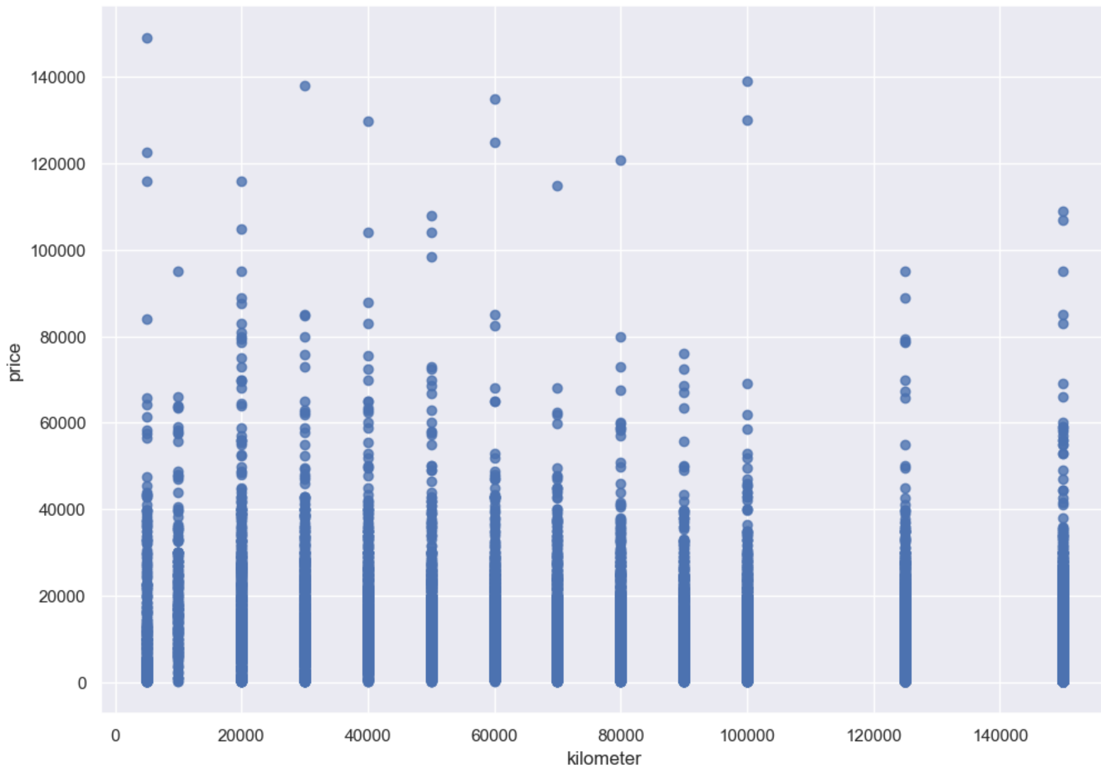


[500]:
```
cars['kilometer'].describe()
```

[500]:
```
count     42772.000
mean     125815.253
std       39078.120
min        5000.000
25%      100000.000
50%      150000.000
75%      150000.000
max      150000.000
Name: kilometer, dtype: float64
```

[501]:
```
sns.regplot(x='kilometer', y='price', scatter=True, fit_reg=False, data=cars)
```

[501]: <AxesSubplot:xlabel='kilometer', ylabel='price'>



Considered in modelling

Variable fuelType

[502]:
```
cars['fuelType'].value_counts()
```

[502]:
```
petrol      26509
diesel      12854
lpg           690
cng            70
hybrid         36
electro        10
other           6
Name: fuelType, dtype: int64
```

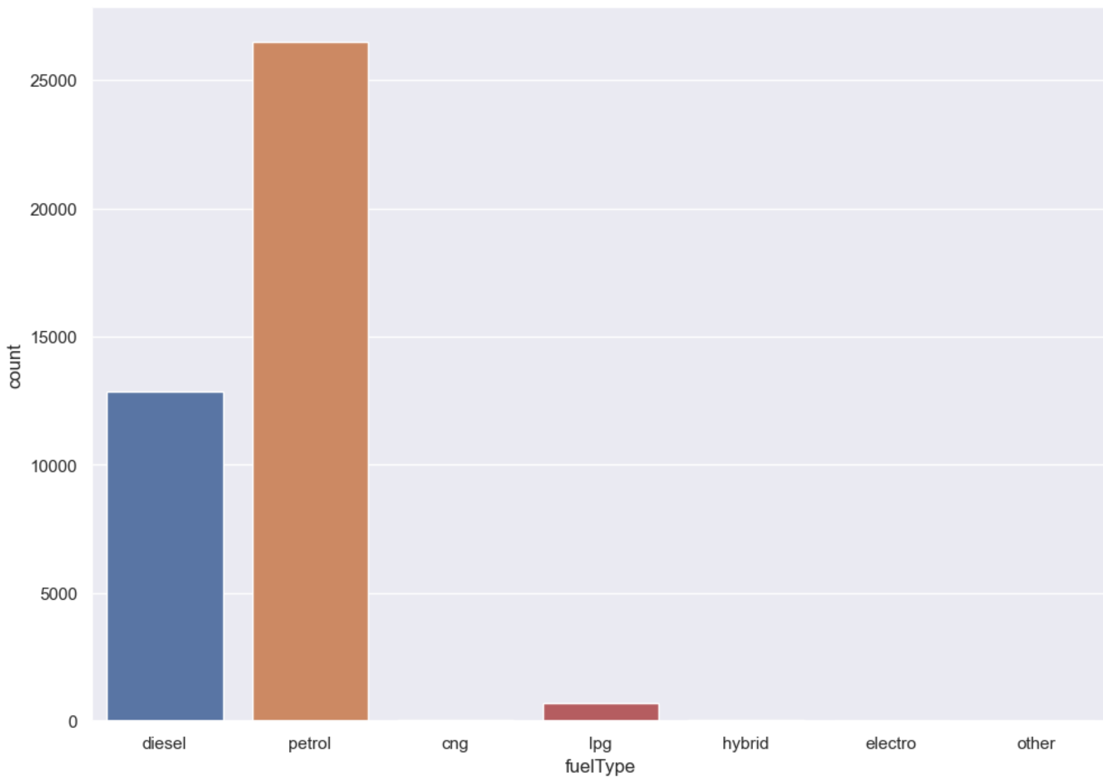
[503]:
```
pd.crosstab(cars['fuelType'], columns='count', normalize=True)
```

[503]:
```
col_0     count
fuelType
cng       0.002
diesel    0.320
electro   0.000
hybrid    0.001
```

```
lpg       0.017
other     0.000
petrol    0.660
```

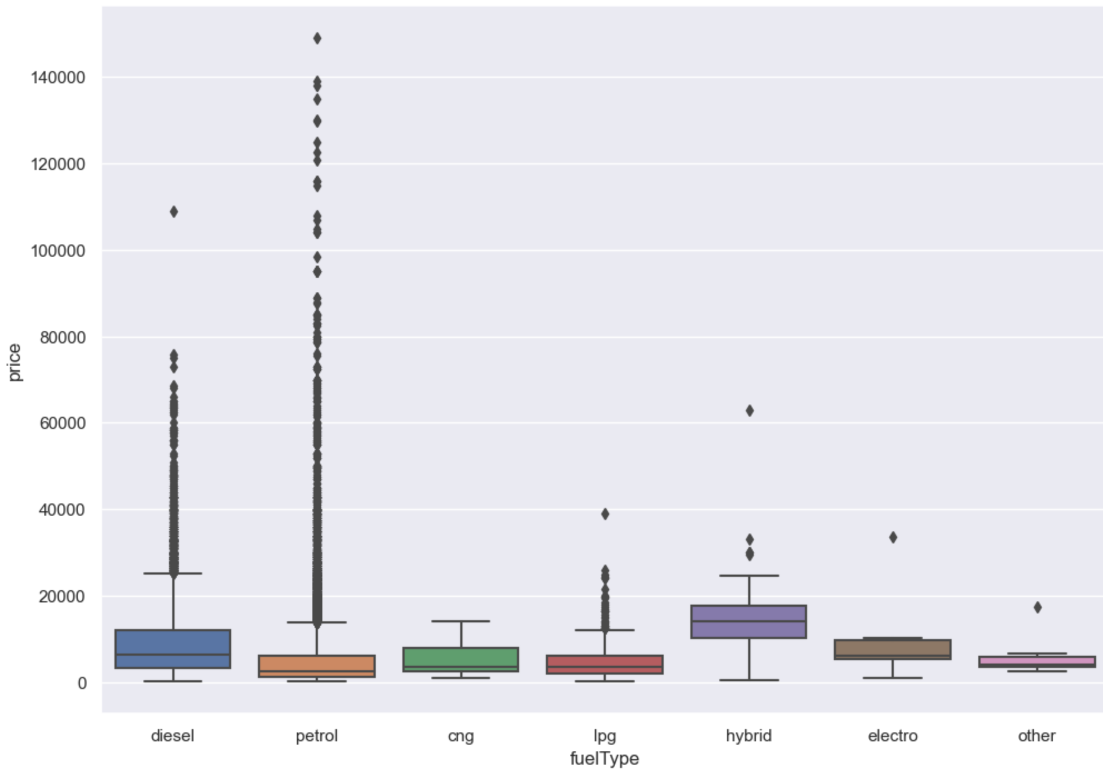
[504]:
```python
sns.countplot(x='fuelType', data=cars)
```

[504]: <AxesSubplot:xlabel='fuelType', ylabel='count'>



[505]:
```python
sns.boxplot(x='fuelType', y='price', data=cars)
```

[505]: <AxesSubplot:xlabel='fuelType', ylabel='price'>



fuelType affects price

Variable brand

[506]:
```
cars['brand'].value_counts()
```

[506]:
```
volkswagen       9134
bmw              4868
opel             4487
mercedes_benz    4134
audi             3984
ford             2815
renault          1941
peugeot          1323
fiat              996
seat              886
skoda             698
mazda             663
smart             623
nissan            601
citroen           598
toyota            547
volvo             429
```

```
mini              428
hyundai           406
mitsubishi        359
honda             300
sonstige_autos    299
kia               276
suzuki            264
porsche           260
alfa_romeo        245
chevrolet         213
chrysler          151
dacia             123
subaru            112
jeep               91
land_rover         81
jaguar             78
daihatsu           67
saab               65
lancia             56
rover              53
daewoo             53
trabant            43
lada               22
Name: brand, dtype: int64
```

[507]:
```python
pd.crosstab(cars['brand'], columns='count', normalize=True)
```

[507]:
```
col_0           count
brand
alfa_romeo      0.006
audi            0.093
bmw             0.114
chevrolet       0.005
chrysler        0.004
citroen         0.014
dacia           0.003
daewoo          0.001
daihatsu        0.002
fiat            0.023
ford            0.066
honda           0.007
hyundai         0.009
jaguar          0.002
jeep            0.002
kia             0.006
lada            0.001
lancia          0.001
```

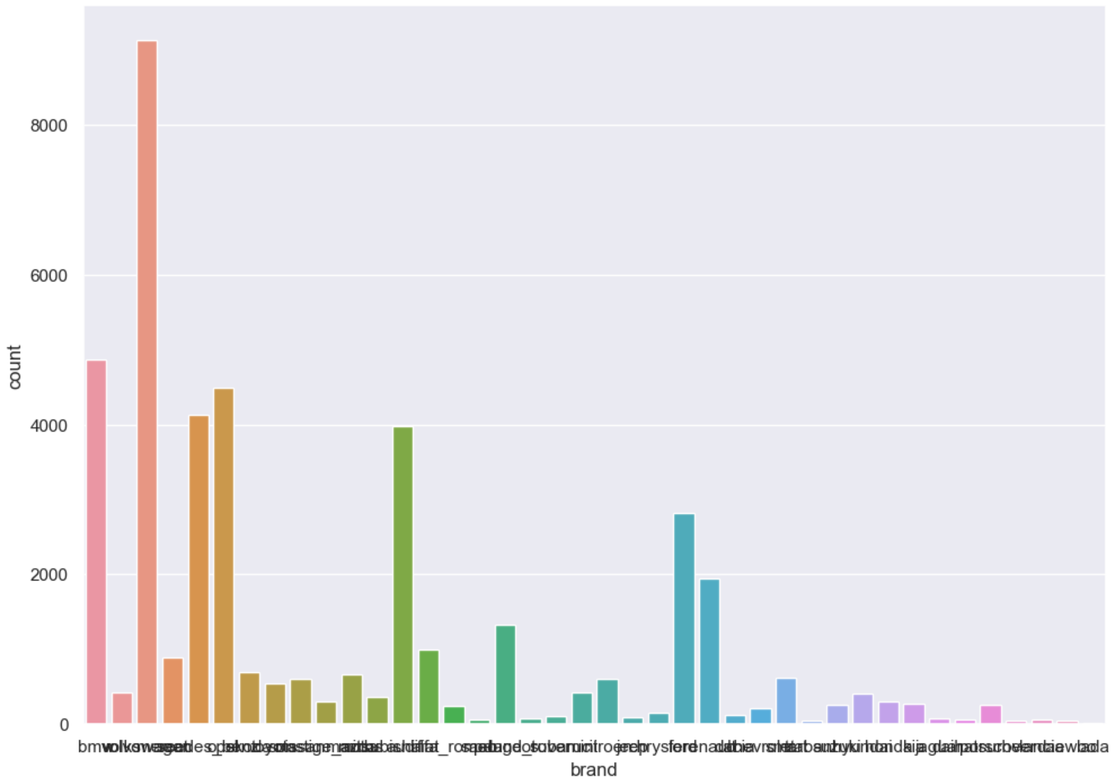
```
land_rover       0.002
mazda            0.016
mercedes_benz    0.097
mini             0.010
mitsubishi       0.008
nissan           0.014
opel             0.105
peugeot          0.031
porsche          0.006
renault          0.045
rover            0.001
saab             0.002
seat             0.021
skoda            0.016
smart            0.015
sonstige_autos   0.007
subaru           0.003
suzuki           0.006
toyota           0.013
trabant          0.001
volkswagen       0.214
volvo            0.010
```

[508]:

```
sns.countplot(x='brand', data=cars)
```

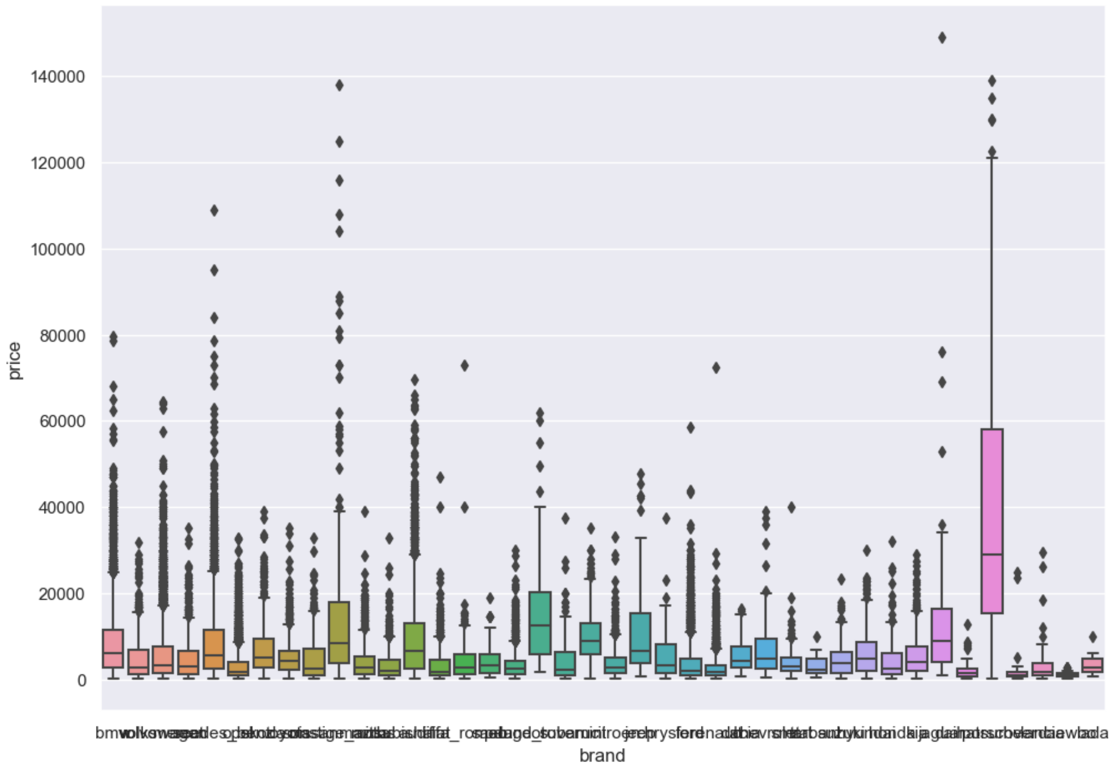
[508]:

```
<AxesSubplot:xlabel='brand', ylabel='count'>
```




```
[509]: sns.boxplot(x='brand', y='price', data=cars)
```

[509]: <AxesSubplot:xlabel='brand', ylabel='price'>



Cars are distributed over many brand
Considered for modelling

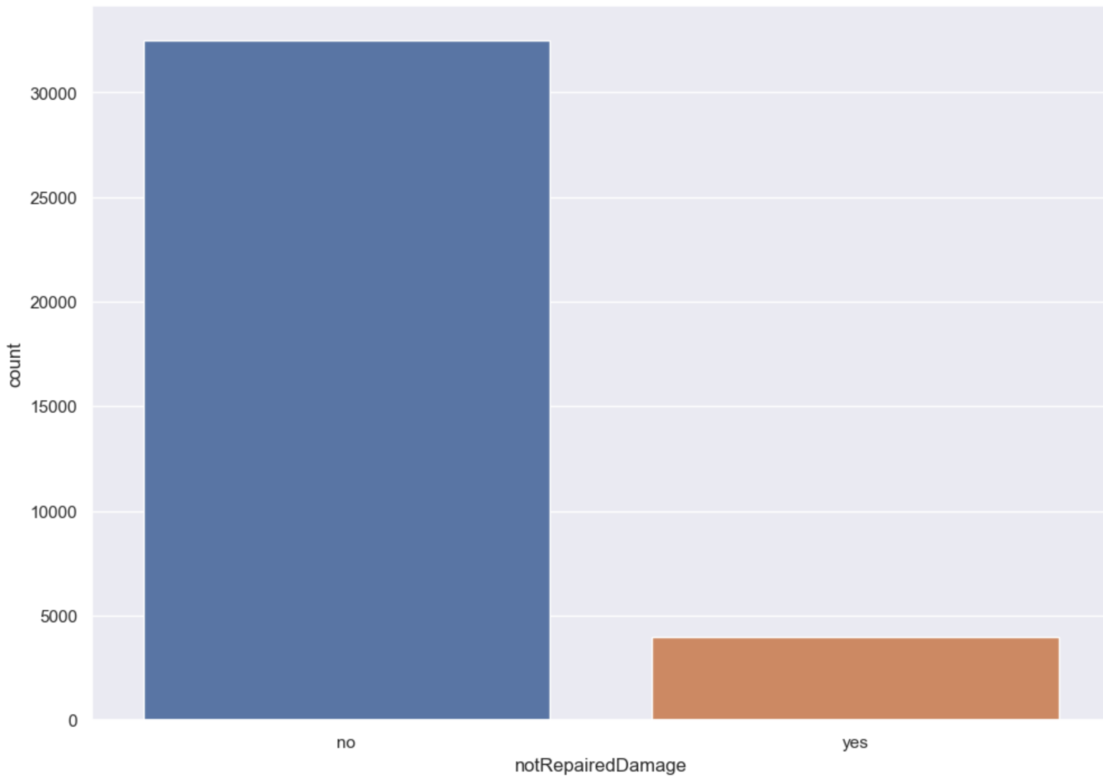
Variable notRepairedDamage
yes - car is damaged but not rectified
no - car was damaged but has been rectified

[510]:
```
cars['notRepairedDamage'].value_counts()
```

[510]:
```
no     32507
yes     3988
Name: notRepairedDamage, dtype: int64
```

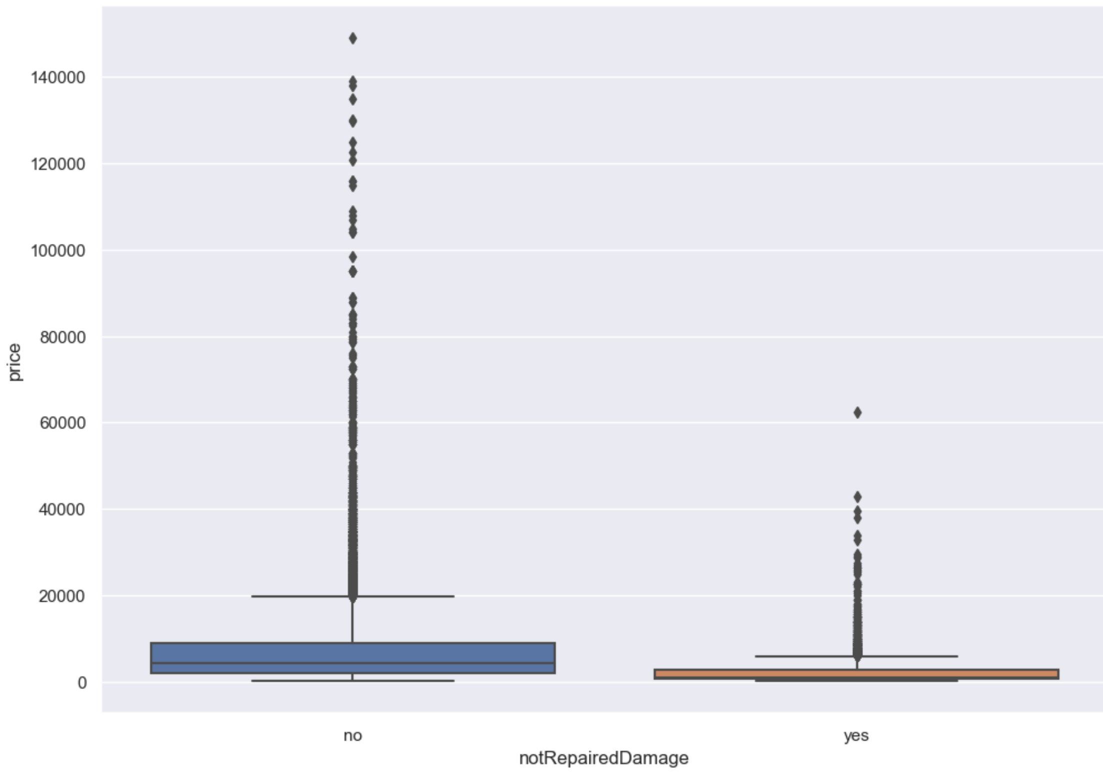
[511]:
```
sns.countplot(x='notRepairedDamage', data=cars)
```

[511]:
```
<AxesSubplot:xlabel='notRepairedDamage', ylabel='count'>
```




```
[512]: sns.boxplot(x='notRepairedDamage', y='price', data=cars)
```

```
[512]: <AxesSubplot:xlabel='notRepairedDamage', ylabel='price'>
```



As expected, the cars that require the damages to be repaired
fall under lower price ranges

Removing insignificant variables

[513]:
```
col = ['seller', 'offerType', 'abtest']
cars = cars.drop(columns=col, axis=1)
```

[514]:
```
cars_copy = cars.copy()
```

[515]:
```
cars_select1 = cars.select_dtypes(exclude=[object])
correlation = cars_select1.corr()
round(correlation, 3)
```

[515]:

	price	powerPS	kilometer	Age
price	1.000	0.575	-0.440	-0.336
powerPS	0.575	1.000	-0.016	-0.151
kilometer	-0.440	-0.016	1.000	0.292
Age	-0.336	-0.151	0.292	1.000

[516]:
```
cars_select1.corr().loc[:, 'price'].abs().sort_values(ascending=False)[1:]
```

[516]:
```
powerPS      0.575
kilometer    0.440
Age          0.336
Name: price, dtype: float64
```

I'll build a Linear Regression and Random Forest model two sets of data.
1. Data obtained by omitting rows with any missing value 2. Data obtained by imputing the missing values

Omitting Missing Values

[517]:
```
cars_omit=cars.dropna(axis=0)
```

Converting categorical variables to dummy variables

[518]:
```
cars_omit = pd.get_dummies(cars_omit, drop_first=True)
```

Importing Necessary Libraries

[519]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Model Building with Omitted Data

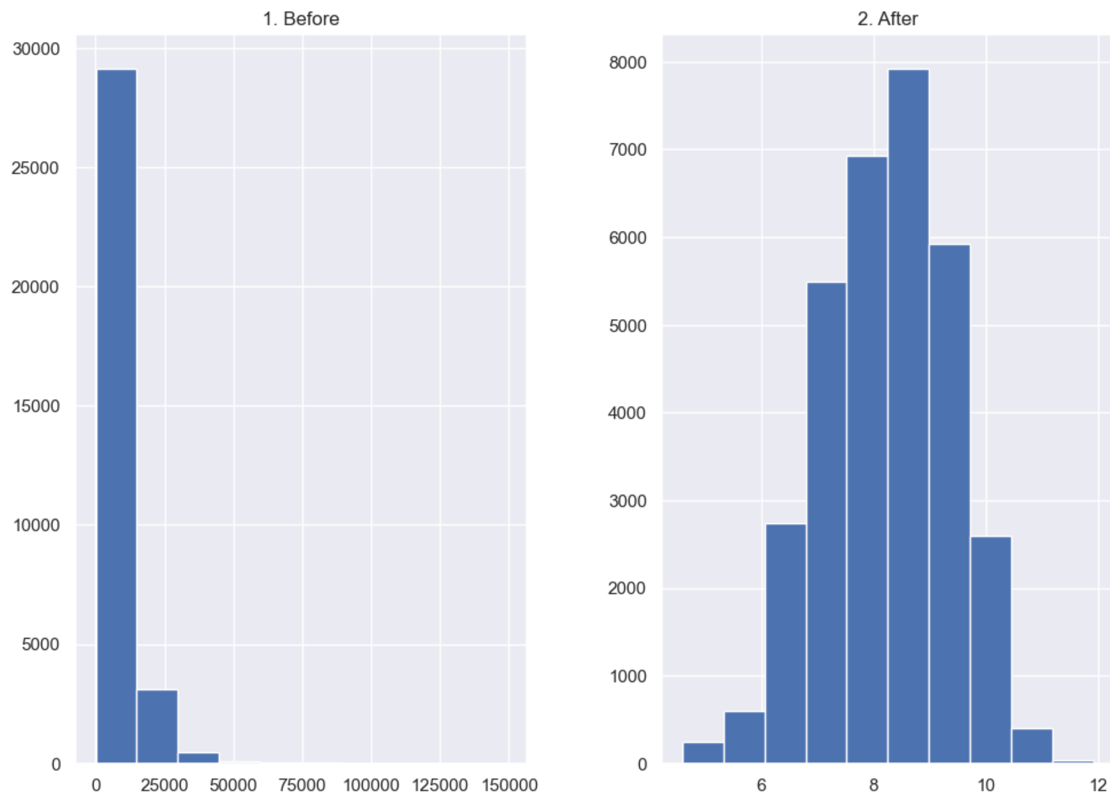
Seperating input and output features

[520]:
```
x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
y1 = cars_omit['price']
```

Plotting the variable price

[521]:
```
prices = pd.DataFrame({"1. Before": y1, "2. After": np.log(y1)})
prices.hist()
```

[521]:
```
array([[<AxesSubplot:title={'center':'1. Before'}>,
        <AxesSubplot:title={'center':'2. After'}>]], dtype=object)
```



Transforming price as a logarithmic value

[522]:
```
y1 = np.log(y1)
```

Splitting data into test and train

[523]:
```
X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_size = 0.3,␣
↪random_state = 3)
```

[524]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(23018, 300) (9866, 300) (23018,) (9866,)
```

Baseline Model for Omitted Data

Base model by using test data mean value
This is to set a benchmark and to compare with the regression model

Finding the mean for test data value

[525]:
```
base_pred = np.mean(y_test)
print(base_pred)
```

```
8.249615787653337
```

Repeating same value till length of test data

```
[526]: base_pred = np.repeat(base_pred, len(y_test))
```

Finding the RMSE

```
[527]: base_root_mean_sqaure_error = np.sqrt(mean_squared_error(y_test, base_pred))
```

```
[528]: base_root_mean_sqaure_error
```

```
[528]: 1.1274483657478247
```

Linear Regression with Omitted Data

Setting intercept as true

```
[529]: lgr = LinearRegression(fit_intercept=True)
```

Model

```
[530]: model_lin1 = lgr.fit(X_train, y_train)
```

Predicting model on test set

```
[531]: cars_predictions_lin1 = lgr.predict(X_test)
```

Computing MSE and RMSE

```
[532]: lin_mse1 = mean_squared_error(y_test, cars_predictions_lin1)
        lin_rmse1 = np.sqrt(lin_mse1)
        print(lin_rmse1)
```

```
0.5455481266513817
```

R squared value

```
[533]: r2_lin_test1 = model_lin1.score(X_test, y_test)
        r2_lin_train1 = model_lin1.score(X_train, y_train)
        print(r2_lin_test1,r2_lin_train1)
```

```
0.7658615091649263 0.7800936978183916
```

Regression diagnostics - Residual plot analysis

```
[534]: residuals1 = y_test-cars_predictions_lin1
```

```
[535]: sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter = True,␣
         ↪fit_reg=False,)
```

```
[535]: <AxesSubplot:ylabel='price'>
```

[536]:
```
residuals1.describe()
```

[536]:
```
count    9866.000
mean        0.003
std         0.546
min        -5.796
25%        -0.261
50%         0.041
75%         0.302
max         4.547
Name: price, dtype: float64
```

Random Forest with Omitted Data

Model Parameters

[537]:
```
rf = RandomForestRegressor(n_estimators = 100, max_features = 'auto',
                           max_depth = 100, min_samples_split = 10,
                           min_samples_leaf = 4, random_state = 1)
```

Model

[538]:
```
model_rf1 = rf.fit(X_train, y_train)
```

Predicting model on test set

[539]:
```
cars_predictions_rf1 = rf.predict(X_test)
```

Computing MSE and RMSE

[540]:
```
rf_mse1 = mean_squared_error(y_test, cars_predictions_rf1)
rf_rmse1 = np.sqrt(rf_mse1)
print(rf_mse1,rf_rmse1)
```

```
0.19016020985430382 0.4360736289370223
```

R squared value

[541]:
```
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)
```

```
0.8504018147750623 0.9202494705146291
```

Model Building with Imputed Data

[542]:
```
cars_imputed = cars.apply(lambda x:x.fillna(x.median()) \
                        if x.dtype == 'float' else \
                         x.fillna(x.value_counts().index[0]))
cars_imputed.isnull().sum()
```

[542]:
```
price                 0
vehicleType           0
gearbox               0
powerPS               0
model                 0
kilometer             0
fuelType              0
brand                 0
notRepairedDamage     0
Age                   0
dtype: int64
```

Converting categorical variables to dummy variables

[543]:
```
cars_imputed = pd.get_dummies(cars_imputed, drop_first=True)
```

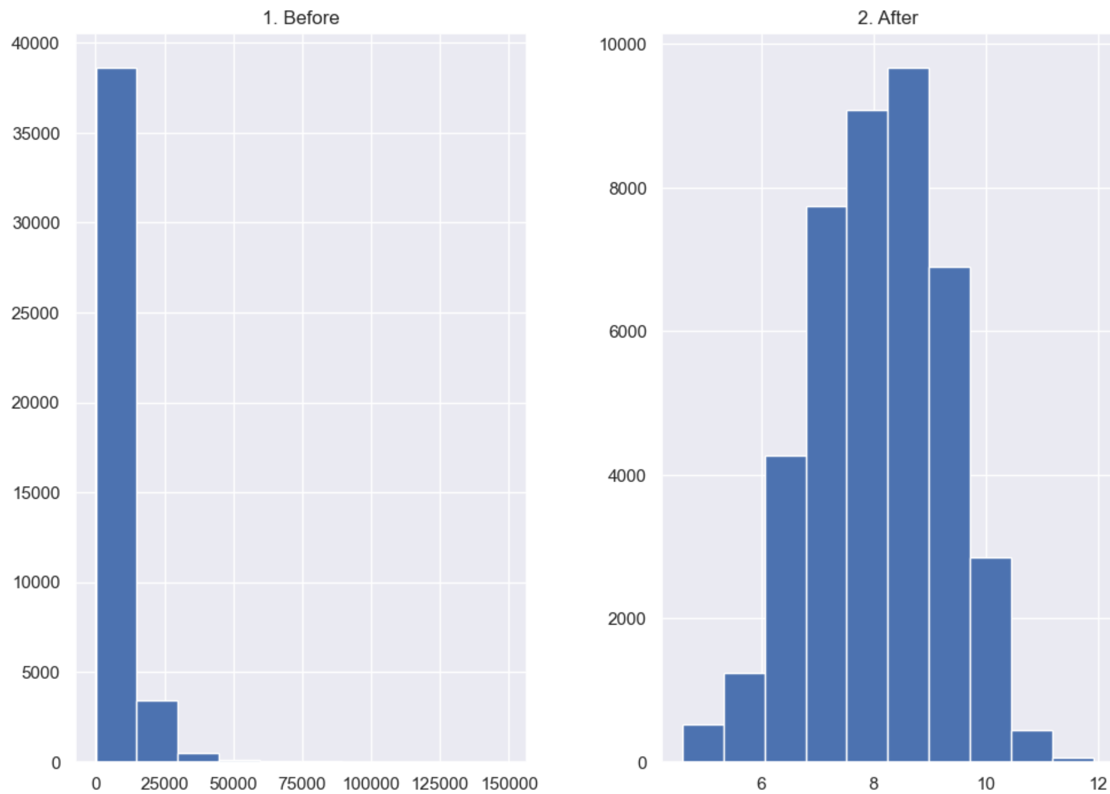
Separating input and output feature

[544]:
```
x2 = cars_imputed.drop(['price'], axis = 'columns', inplace = False)
y2 = cars_imputed['price']
```

Plotting the variable price

[545]:
```
prices = pd.DataFrame({"1. Before": y2, "2. After": np.log(y2)})
prices.hist()
```

[545]: array([[<AxesSubplot:title={'center':'1. Before'}>,
        <AxesSubplot:title={'center':'2. After'}>]], dtype=object)



Transfroming price as a logarithmic value

[546]:
```
y2 = np.log(y2)
```

Splitting data into test and train

[547]:
```
X_train1, X_test1, y_train1, y_test1 = train_test_split(x2, y2, test_size=0.3,␣
↪random_state = 3)
print(X_train1.shape, X_test1.shape, y_train1.shape, y_test1.shape)
```

```
(29940, 303) (12832, 303) (29940,) (12832,)
```

Baseline Model for Imputed Data

We are making a base model by using test data mean value
This is to set a benchmark and to compare with our regression model

Finding the mean for test data value

[548]:
```
base_pred = np.mean(y_test1)
print(base_pred)
```

8.068391740519193

Repeating same value till length of test data

[549]:
```
base_pred = np.repeat(base_pred, len(y_test1))
```

finding the RMSE

[550]:
```
base_root_mean_square_error_imputed = np.sqrt(mean_squared_error(y_test1,␣
 ↪base_pred))
print(base_root_mean_square_error_imputed)
```

1.1884349112889792

Linear Regression with Imputed Data

Setting intercept as true

[551]:
```
lgr2 = LinearRegression(fit_intercept=True)
```

Model

[552]:
```
model_lin2 = lgr2.fit(X_train1, y_train1)
```

Predicting model on test data

[553]:
```
cars_predictions_lin2 = lgr2.predict(X_test1)
```

Computing MSE and RMSE

[554]:
```
lin_mse2 = mean_squared_error(y_test1, cars_predictions_lin2)
lin_rmse2 = np.sqrt(lin_mse2)
print(lin_rmse2)
```

0.6483956449231337

R squared value

[555]:
```
r2_lin_train2 = model_lin2.score(X_train1, y_train1)
r2_lin_test2 = model_lin2.score(X_test1, y_test1)
print(r2_lin_train2, r2_lin_test2)
```

0.7071658736894362 0.7023339008631146

Random Forest with Imputed Data

Model Parameters

[556]:
```
rf2 = RandomForestRegressor(n_estimators = 100, max_features='auto',
                            max_depth=100, min_samples_split=10,
                            min_samples_leaf=4,random_state=1)
```

Model

[557]:
```
model_rf2 = rf2.fit(X_train1, y_train1)
```

Predicting model on test set

[558]:
```
cars_predictions_rf2 = rf2.predict(X_test1)
```

Computing MSE and RMSE

[559]:
```
rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)
rf_rmse2 = np.sqrt(rf2_mse2)
print(rf_rmse2)
```

```
0.0683662371791568
```

R squared value

[560]:
```
r2_rf_train2 = model_rf2.score(X_train1, y_train1)
r2_rf_test2 = model_rf2.score(X_test1, y_test1)
print(r2_rf_train2, r2_rf_test2)
```

```
0.9024289431669166 0.8269964521311131
```

Final output

[561]:
```
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"% base_root_mean_sqaure_error)
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 omitted =␣
 ↪%s"% base_root_mean_square_error_imputed)
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 = 0.7800936978183916
R squared value for test from Linear Regression = 0.7658615091649263
R squared value for train from Random Forest = 0.9202494705146291
R squared value for test from Random Forest = 0.8504018147750623
Base RMSE of model built from data where missing values were omitted =
1.1274483657478247
RMSE value for test from Linear Regression = 0.5455481266513817
RMSE value for test from Random Forest = 0.4360736289370223
```

```
Metrics for models built from data where missing values were imputed
R squared value for train from Linear Regression = 0.7071658736894362
R squared value for test from Linear Regression = 0.7023339008631146
R squared value for train from Random Forest = 0.9024289431669166
R squared value for test from Random Forest = 0.8269964521311131
Base RMSE of model built from data where missing values were omitted =
1.1884349112889792
RMSE value for test from Linear Regression = 0.6483956449231337
RMSE value for test from Random Forest = 0.0683662371791568
```