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

In [2]:
```python
car_data=pd.read_csv("C:/Users/shrey/Desktop/Datasets/cars_sampled.csv")
car_data.head()
```

Out[2]:

	dateCrawled	name	seller	offerType	pri
0	30/03/2016 13:51	Zu_verkaufen	private	offer	44
1	7/3/2016 9:54	Volvo_XC90_2.4D_Summum	private	offer	132
2	1/4/2016 0:57	Volkswagen_Touran	private	offer	32
3	19/03/2016 17:50	Seat_Ibiza_1.4_16V_Reference	private	offer	45
4	16/03/2016 14:51	Volvo_XC90_D5_Aut._RDesign_R_Design_AWD_GSHD_S...	private	offer	187

In [3]:
```python
car_data.shape
```

Out[3]:
```
(50001, 19)
```

In [4]:
```python
car_data.columns
```

Out[4]:
```
Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
       'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model',
       'kilometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage', 'dateCreated', 'postalCode', 'lastSeen'],
      dtype='object')
```

In [5]:
```python
# Setting dimensions for the plot
sns.set(rc={"figure.figsize":(11.7,8.27)})
```

In [6]:
```python
cars=car_data.copy()
```

In [7]:

```python
# structures
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
```

In [8]:

```python
# Summarizing data
cars.describe()
pd.set_option('display.float_format', lambda x: '%.3f' % x)
cars.describe()
```

Out[8]:

	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	postalC
count	50001.000	50001.000	50001.000	50001.000	50001.000	50001.
mean	6559.865	2005.544	116.496	125613.688	5.744	50775.
std	85818.470	122.992	230.568	40205.234	3.711	25743.
min	0.000	1000.000	0.000	5000.000	0.000	1067.
25%	1150.000	1999.000	69.000	125000.000	3.000	30559.
50%	2950.000	2003.000	105.000	150000.000	6.000	49504.
75%	7190.000	2008.000	150.000	150000.000	9.000	71404.
max	12345678.000	9999.000	19312.000	150000.000	12.000	99998.

In [9]:
```python
# To display maximum set of columns
pd.set_option('display.max_columns', 500)
cars.describe()
```

Out[9]:

	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	postalC
count	50001.000	50001.000	50001.000	50001.000	50001.000	50001.
mean	6559.865	2005.544	116.496	125613.688	5.744	50775.
std	85818.470	122.992	230.568	40205.234	3.711	25743.
min	0.000	1000.000	0.000	5000.000	0.000	1067.
25%	1150.000	1999.000	69.000	125000.000	3.000	30559.
50%	2950.000	2003.000	105.000	150000.000	6.000	49504.
75%	7190.000	2008.000	150.000	150000.000	9.000	71404.
max	12345678.000	9999.000	19312.000	150000.000	12.000	99998.

In [10]:
```python
# Dropping unwanted columns
col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
cars=cars.drop(columns=col, axis=1)
```

In [11]:
```python
# Removing duplicate records
cars.drop_duplicates(keep='first',inplace=True)
#470 duplicate records
```

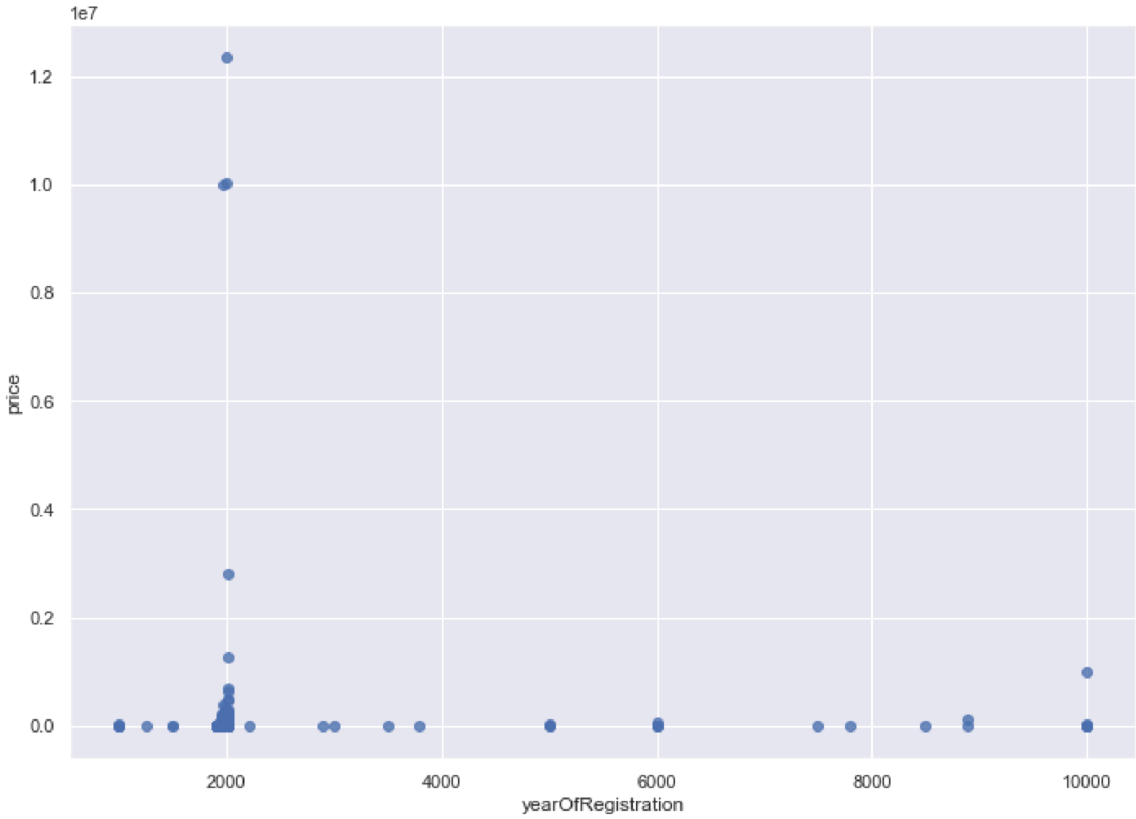
In [12]:
```python
# Data cleaning
# No. of missing values in each column
cars.isnull().sum()
```

Out[12]:
```
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 [13]:

```python
# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018
```

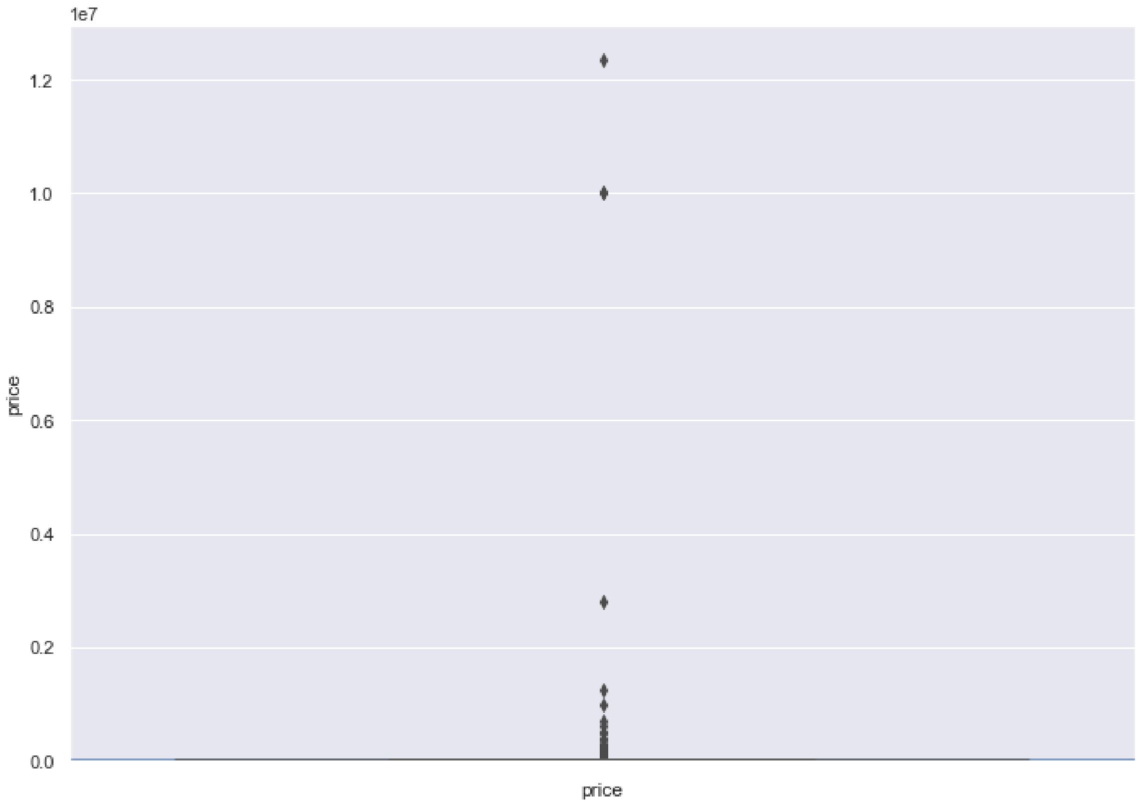
Out[13]: <AxesSubplot:xlabel='yearOfRegistration', ylabel='price'>



In [14]:
```
# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000
```

```
C:\Users\shrey\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

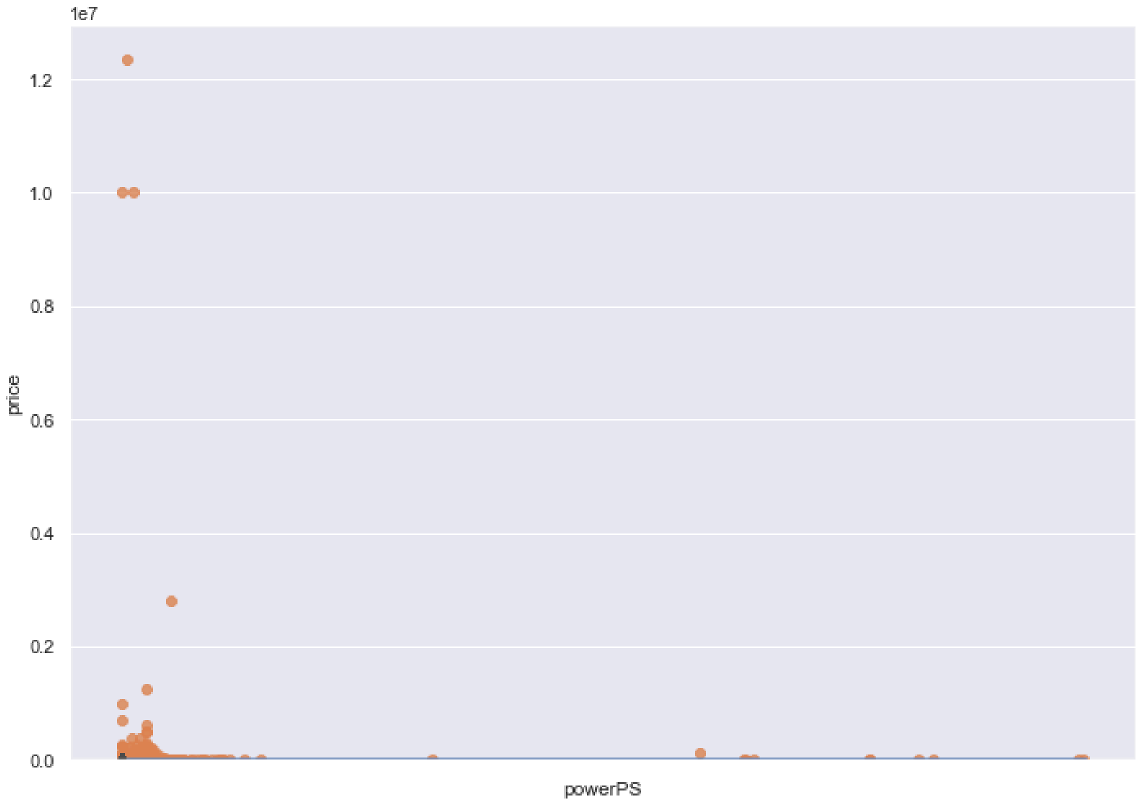
Out[14]: 1748



In [15]:
```python
# Variable powerPS
power_count=cars['powerPS'].value_counts().sort_index()
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(x='powerPS', y='price', scatter=True,
            fit_reg=False, data=cars)
sum(cars['powerPS'] > 500)
sum(cars['powerPS'] < 10)
# Working range- 10 and 500
```

```
C:\Users\shrey\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[15]: 5565



In [16]:
```python
# Working range of data
cars = cars[
        (cars.yearOfRegistration <= 2018)
      & (cars.yearOfRegistration >= 1950)
      & (cars.price >= 100)
      & (cars.price <= 150000)
      & (cars.powerPS >= 10)
      & (cars.powerPS <= 500)]
# ~6700 records are dropped
```

In [17]:
```python
# Further to simplify- variable reduction
# Combining yearOfRegistration and monthOfRegistration
cars['monthOfRegistration']/=12
```

In [18]:
```python
# Creating new varible Age by adding yearOfRegistration and monthOfRegistrat
cars['Age']=(2018-cars['yearOfRegistration'])+cars['monthOfRegistration']
cars['Age']=round(cars['Age'],2)
cars['Age'].describe()
```

Out[18]:
```
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
```

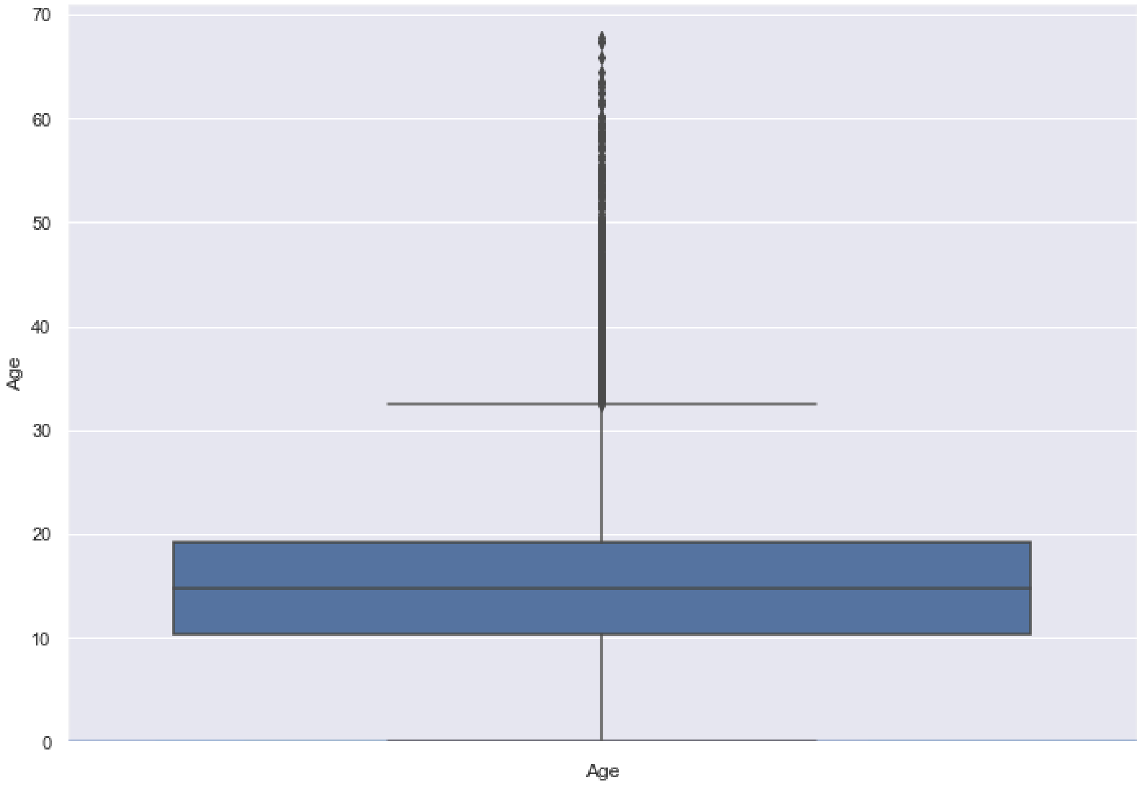
In [19]:
```python
# Dropping yearOfRegistration and monthOfRegistration
cars=cars.drop(columns=['yearOfRegistration','monthOfRegistration'], axis=1)
```

Visualizing parameters

In [20]:
```
# Age
sns.distplot(cars['Age'])
sns.boxplot(y=cars['Age'])
```

```
C:\Users\shrey\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

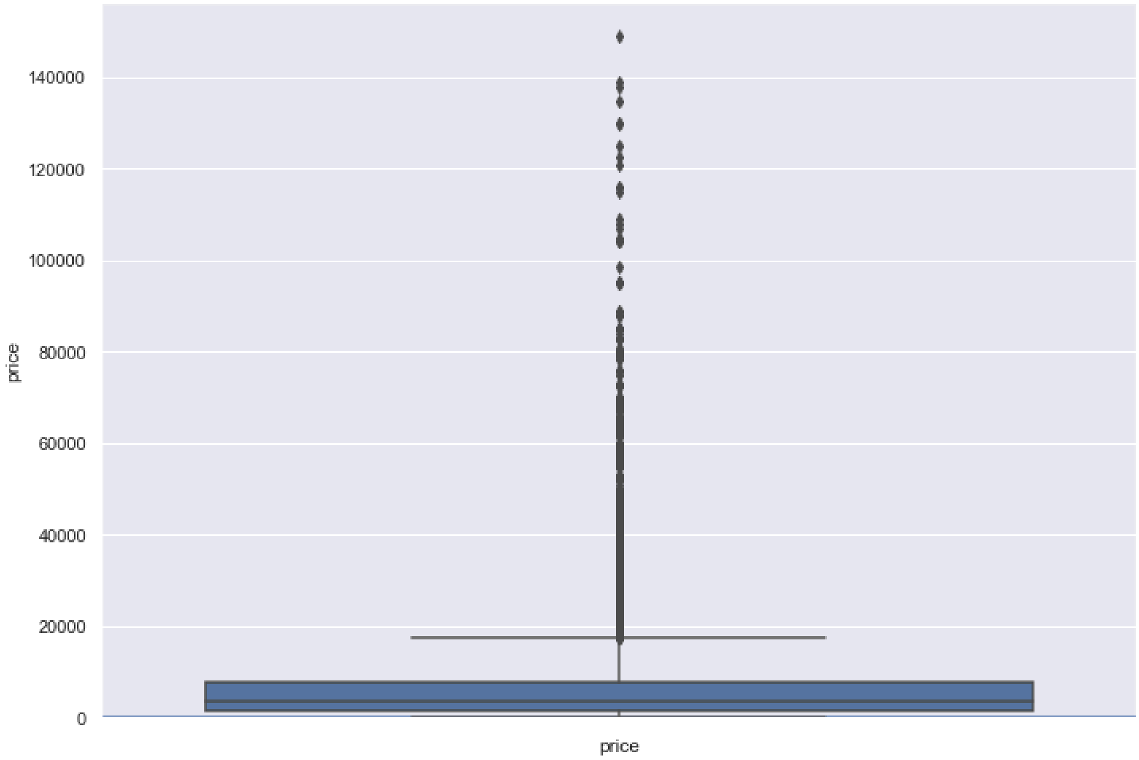
Out[20]:
```
<AxesSubplot:xlabel='Age', ylabel='Age'>
```



In [21]:
```
# price
sns.distplot(cars['price'])
sns.boxplot(y=cars['price'])
```

```
C:\Users\shrey\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

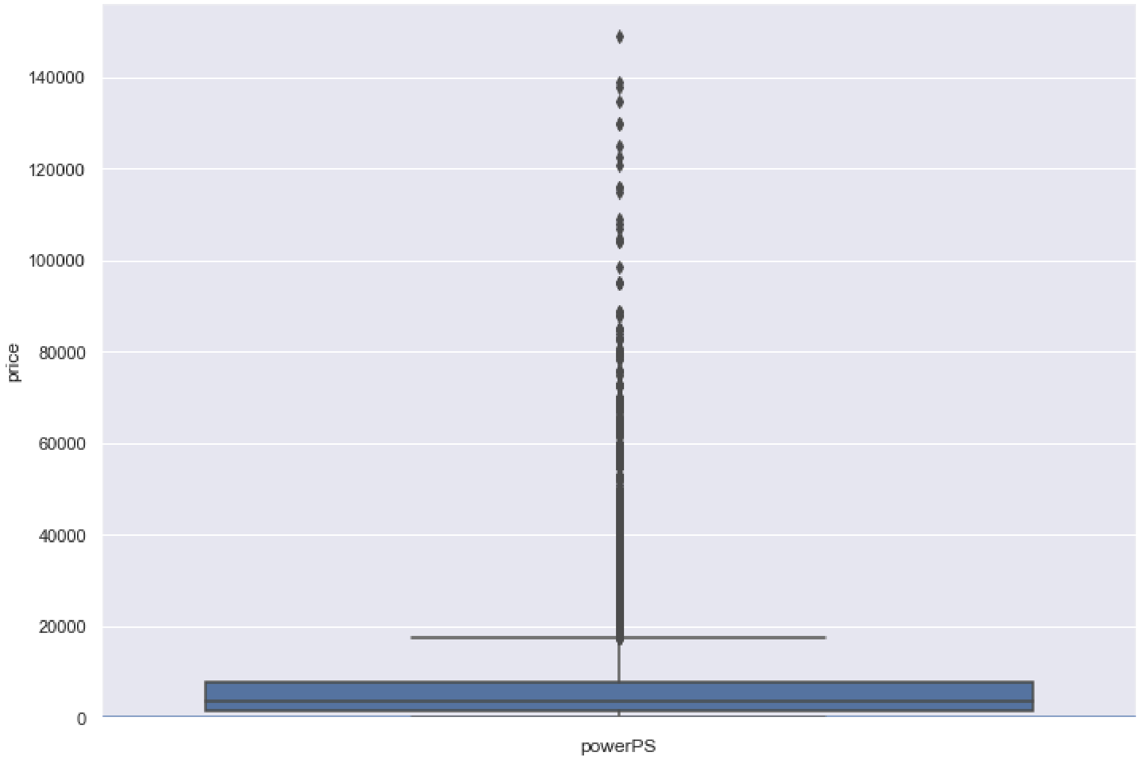
Out[21]:
```
<AxesSubplot:xlabel='price', ylabel='price'>
```



In [22]:
```
# powerPS
sns.distplot(cars['powerPS'])
sns.boxplot(y=cars['price'])
```

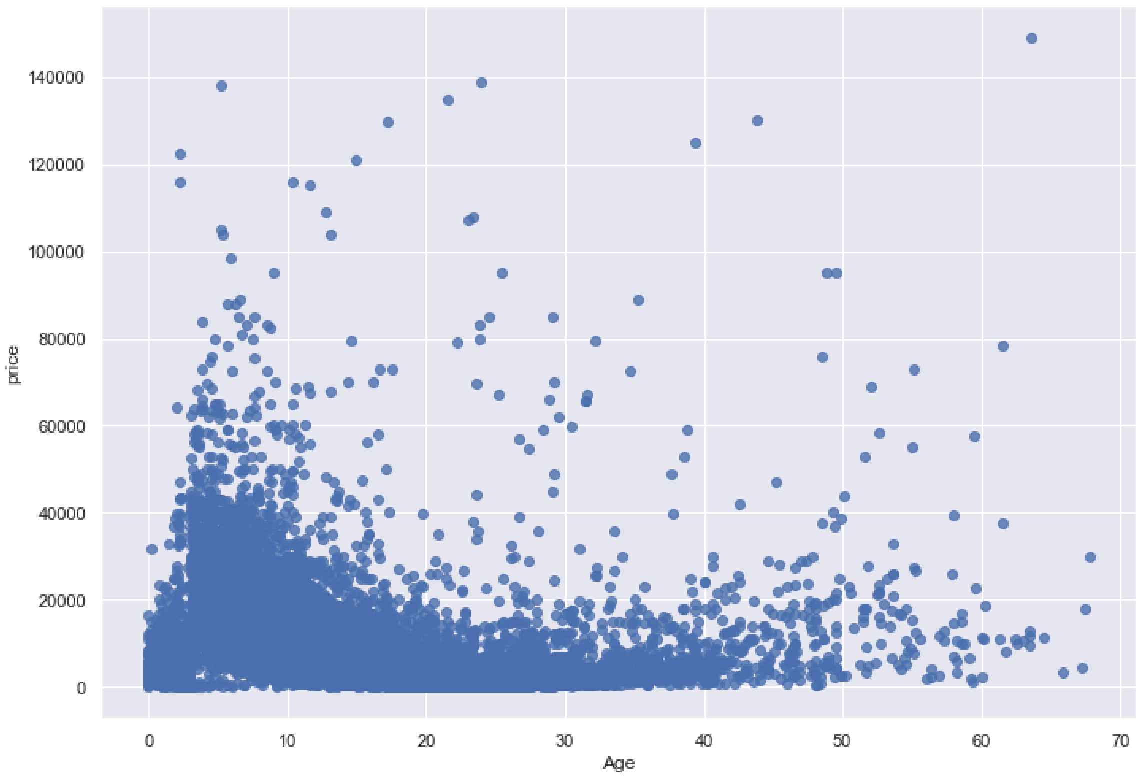
```
C:\Users\shrey\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[22]:
```
<AxesSubplot:xlabel='powerPS', ylabel='price'>
```



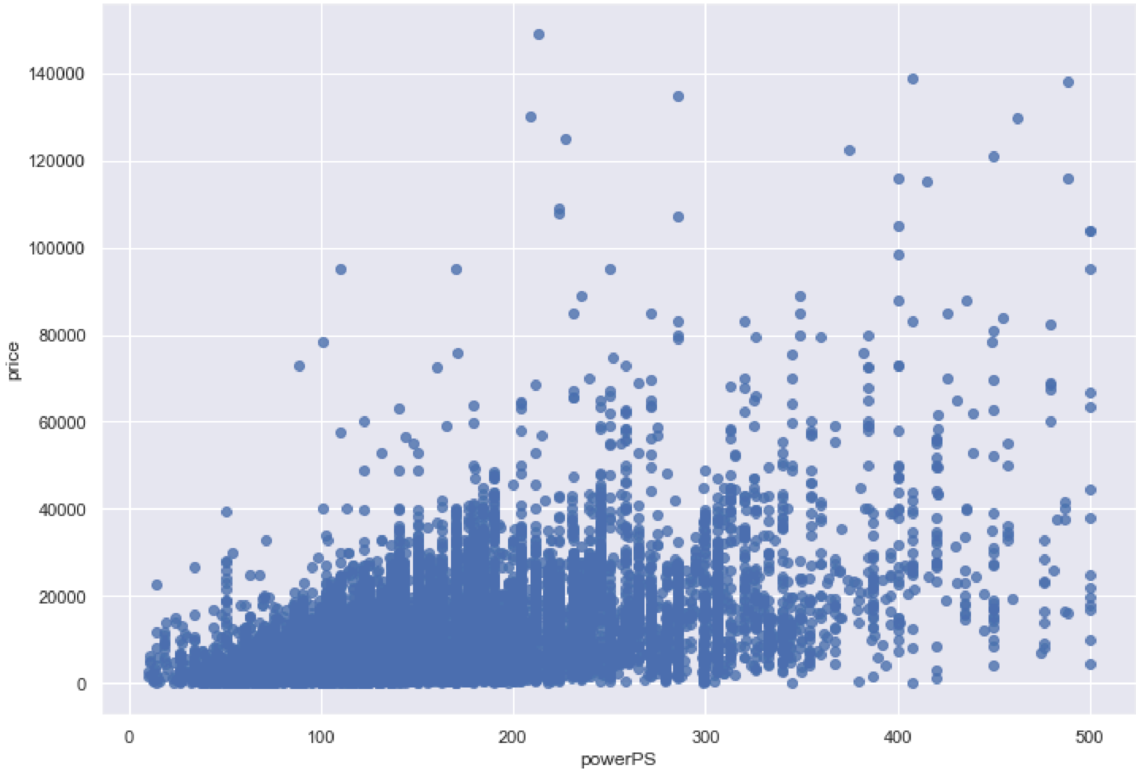
In [23]:
```
# Visualizing parameters after narrowing working range
# Age vs price
sns.regplot(x='Age', y='price', scatter=True,
            fit_reg=False, data=cars)
# Cars priced higher are newer
# With increase in age, price decreases
# However some cars are priced higher with increase in age
```

Out[23]: <AxesSubplot:xlabel='Age', ylabel='price'>



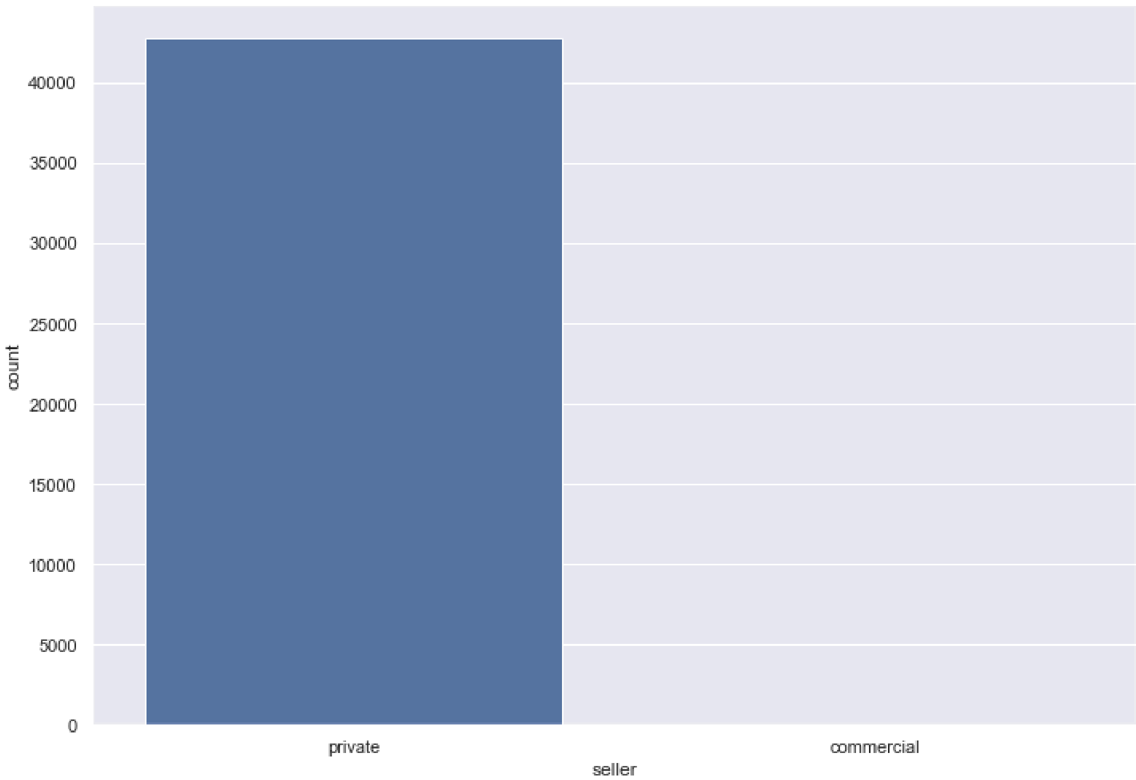
In [24]:
```
# powerPS vs price
sns.regplot(x='powerPS', y='price', scatter=True,fit_reg=False, data=cars)
```

Out[24]: <AxesSubplot:xlabel='powerPS', ylabel='price'>



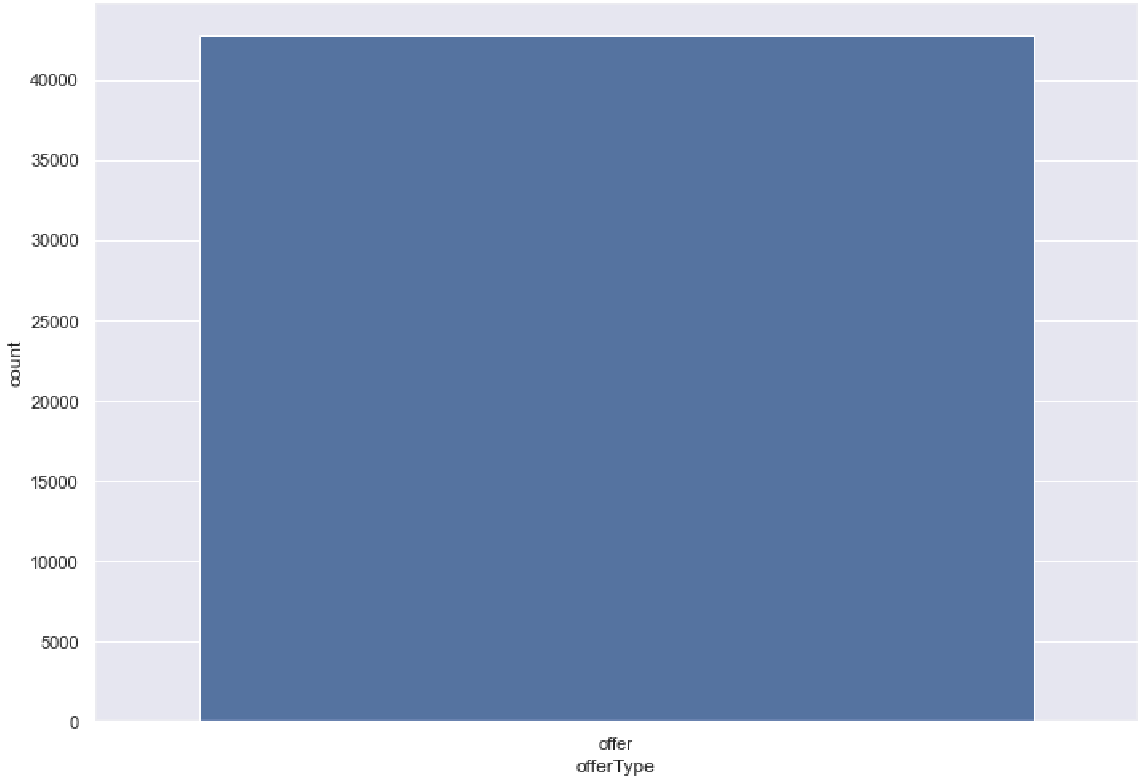
In [25]:
```
# Variable seller
cars['seller'].value_counts()
pd.crosstab(cars['seller'],columns='count',normalize=True)
sns.countplot(x= 'seller',data=cars)
# Fewer cars have 'commercial'=> Insignificant
```

Out[25]: <AxesSubplot:xlabel='seller', ylabel='count'>



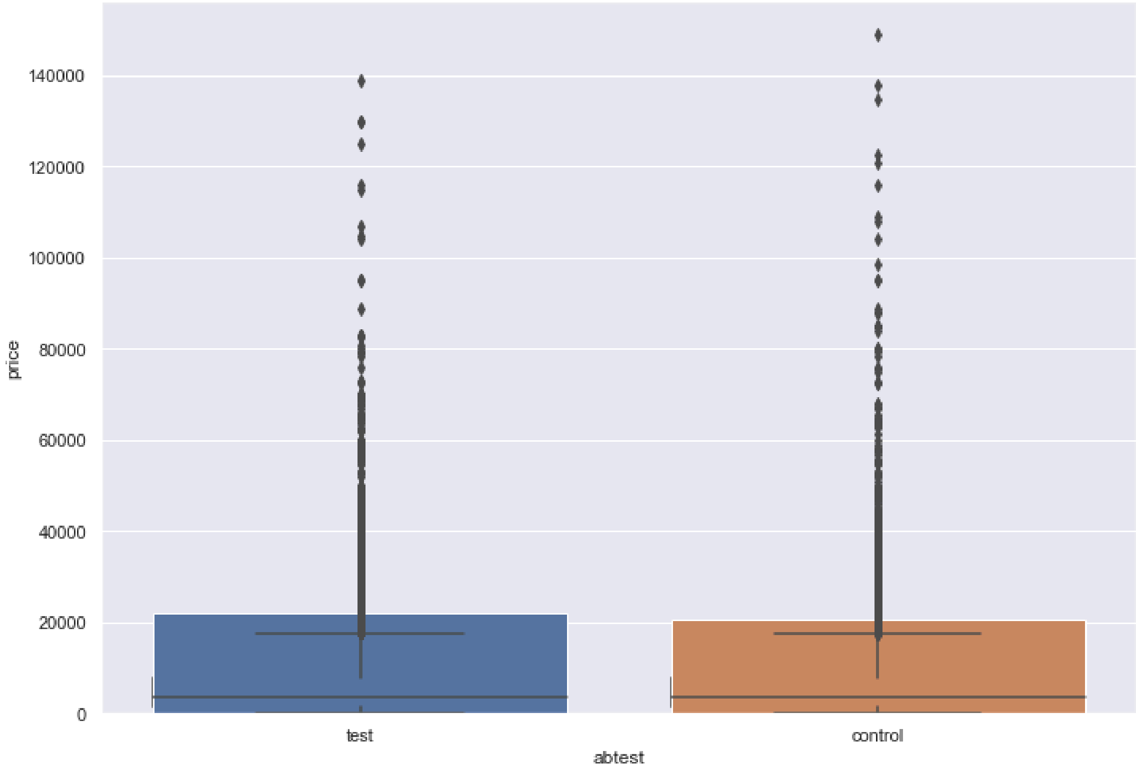
In [26]:
```
# Variable offerType
cars['offerType'].value_counts()
sns.countplot(x= 'offerType',data=cars)
# All cars have 'offer'=> Insignificant
```

Out[26]: <AxesSubplot:xlabel='offerType', ylabel='count'>



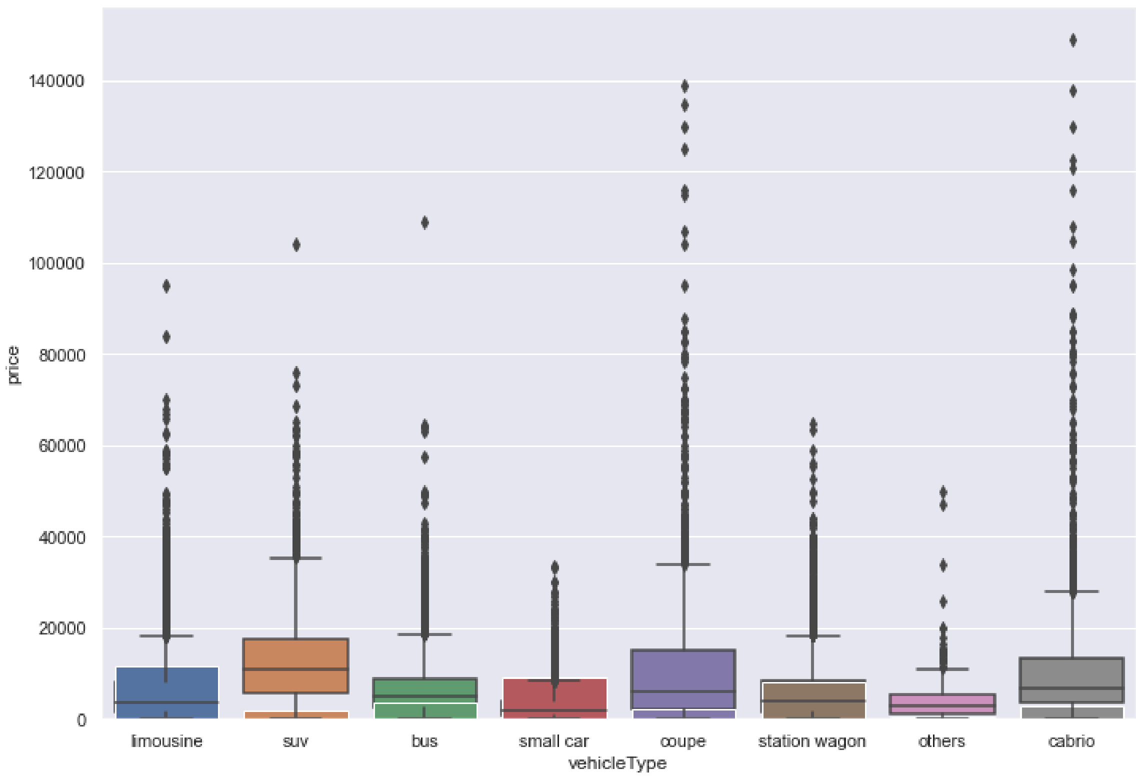
In [27]:
```python
# Variable abtest
cars['abtest'].value_counts()
pd.crosstab(cars['abtest'],columns='count',normalize=True)
sns.countplot(x= 'abtest',data=cars)
# Equally distributed
sns.boxplot(x= 'abtest',y='price',data=cars)
# For every price value there is almost 50-50 distribution
# Does not affect price => Insignificant
```

Out[27]: <AxesSubplot:xlabel='abtest', ylabel='price'>



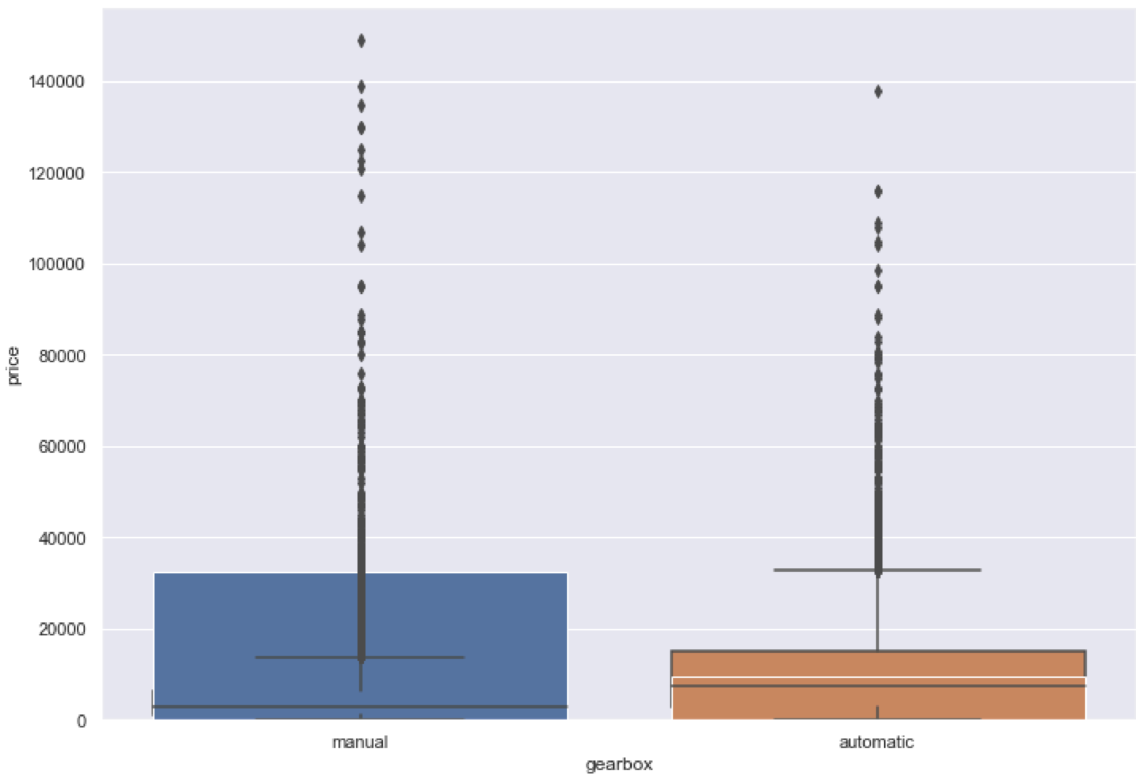
In [28]:
```
# Variable vehicleType
cars['vehicleType'].value_counts()
pd.crosstab(cars['vehicleType'],columns='count',normalize=True)
sns.countplot(x= 'vehicleType',data=cars)
sns.boxplot(x= 'vehicleType',y='price',data=cars)
# 8 types- limousine, small cars and station wagons max freq
# vehicleType affects price
```

Out[28]:
```
<AxesSubplot:xlabel='vehicleType', ylabel='price'>
```



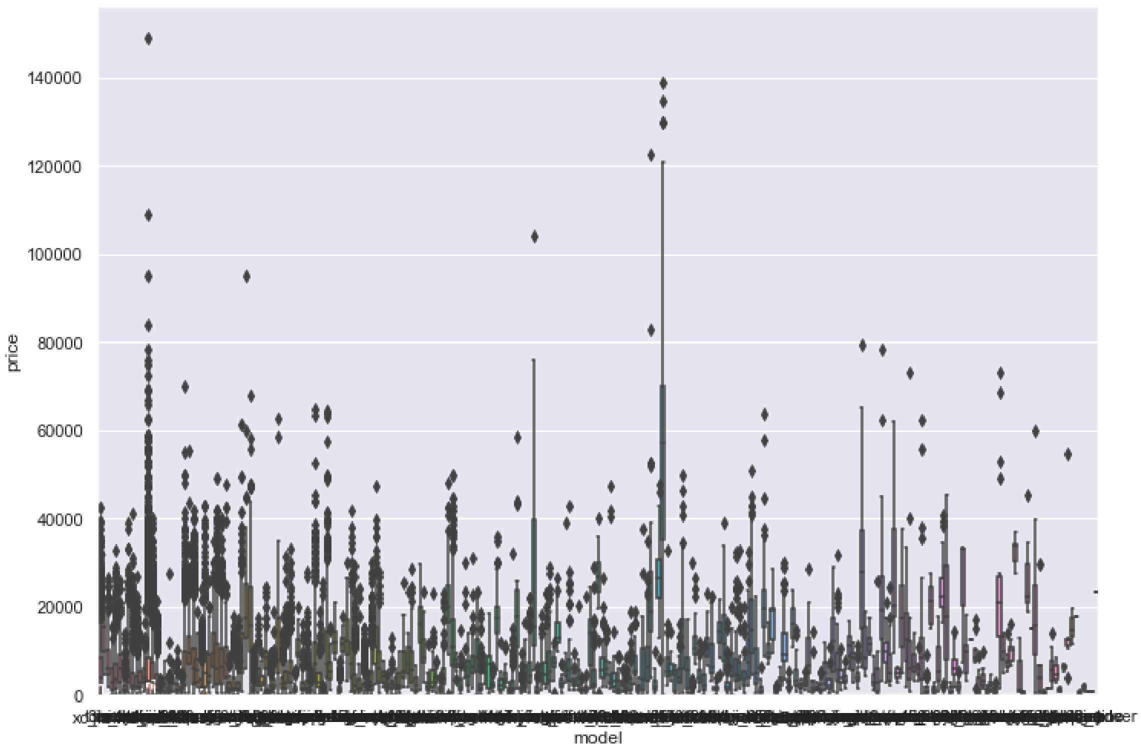
In [29]:
```
# Variable gearbox
cars['gearbox'].value_counts()
pd.crosstab(cars['gearbox'],columns='count',normalize=True)
sns.countplot(x= 'gearbox',data=cars)
sns.boxplot(x= 'gearbox',y='price',data=cars)
# gearbox affects price
```

Out[29]: <AxesSubplot:xlabel='gearbox', ylabel='price'>



```python
In [30]: # Variable model
cars['model'].value_counts()
pd.crosstab(cars['model'],columns='count',normalize=True)
sns.countplot(x= 'model',data=cars)
sns.boxplot(x= 'model',y='price',data=cars)
# Cars are distributed over many models
# Considered in modelling
```

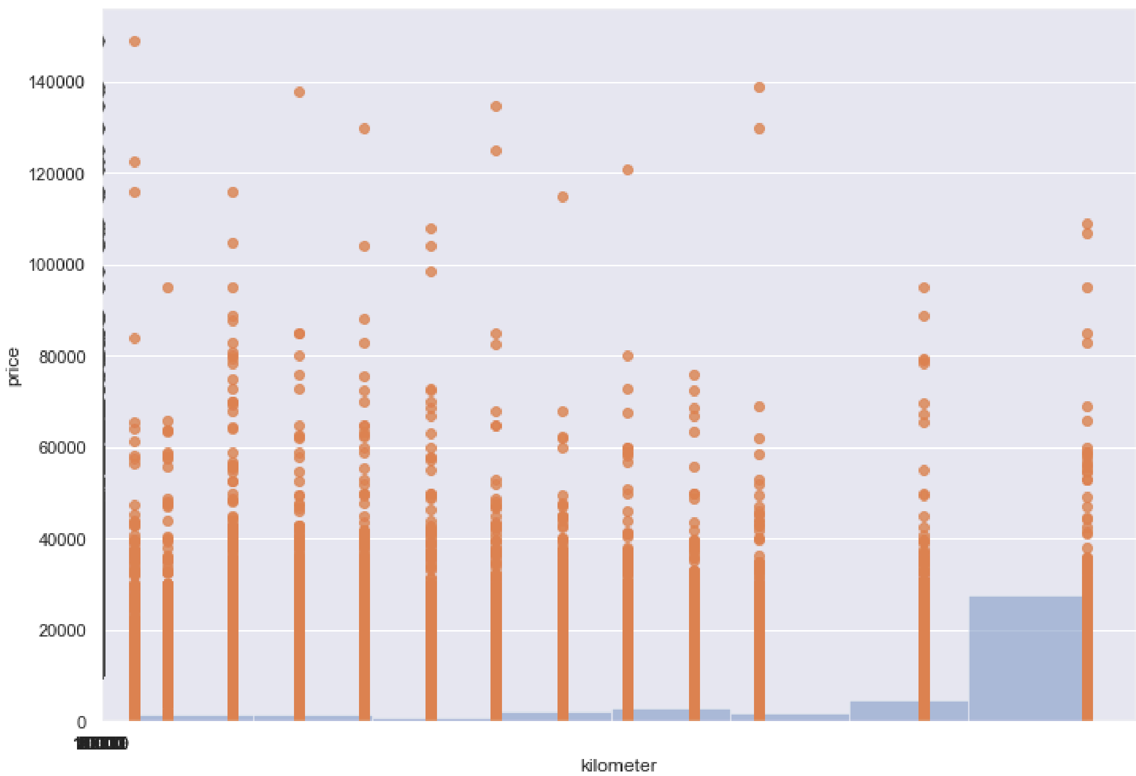
Out[30]: <AxesSubplot:xlabel='model', ylabel='price'>



In [31]:
```
# Variable kilometer
cars['kilometer'].value_counts().sort_index()
pd.crosstab(cars['kilometer'],columns='count',normalize=True)
sns.boxplot(x= 'kilometer',y='price',data=cars)
cars['kilometer'].describe()
sns.distplot(cars['kilometer'],bins=8 ,kde=False)
sns.regplot(x='kilometer', y='price', scatter=True,
            fit_reg=False, data=cars)
# Considered in modelling
```

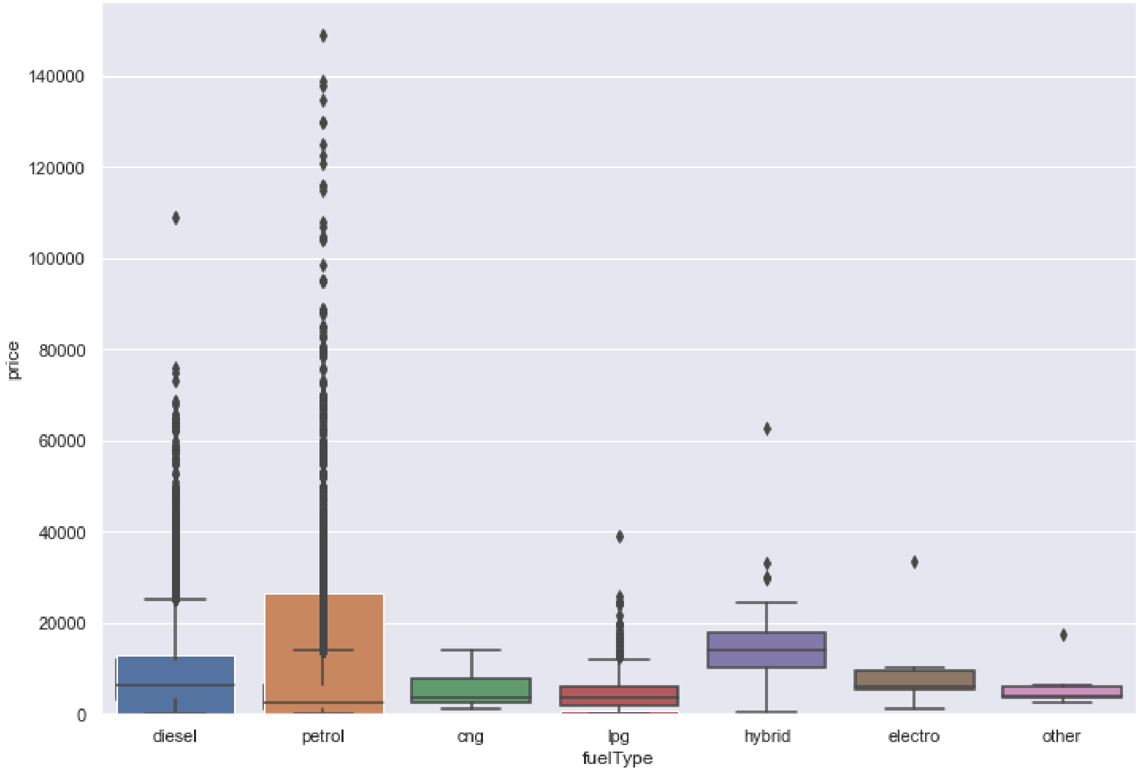
```
C:\Users\shrey\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[31]: <AxesSubplot:xlabel='kilometer', ylabel='price'>



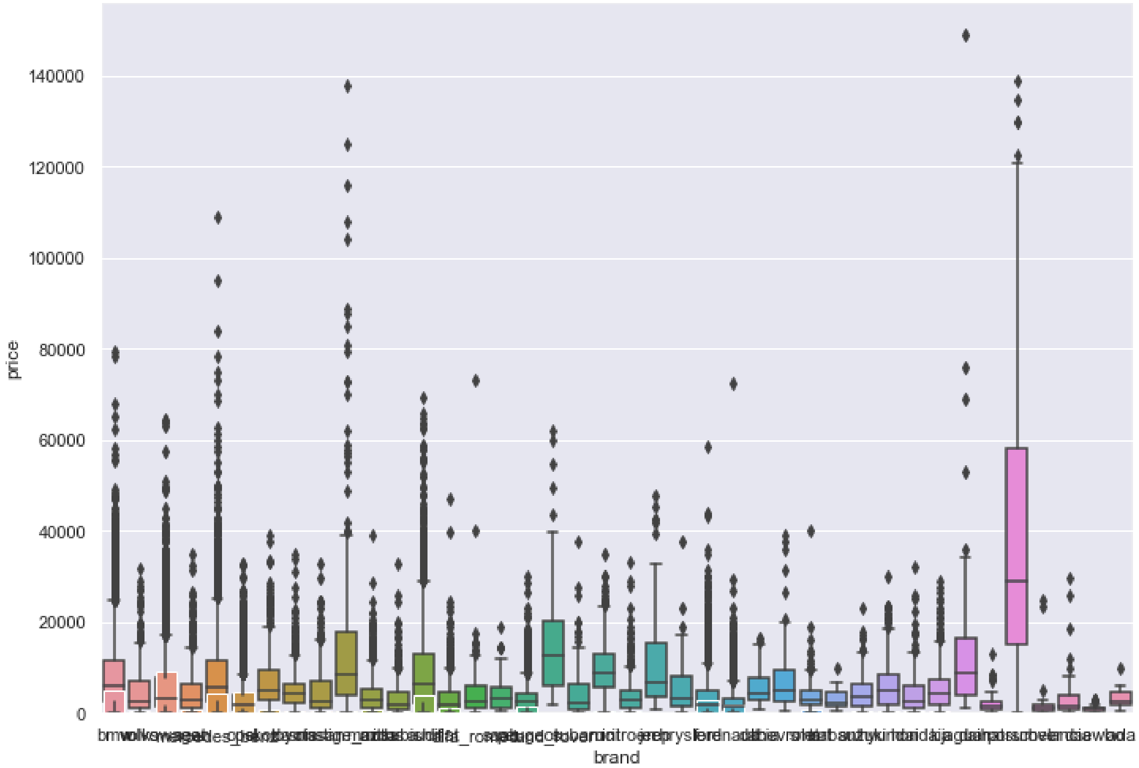
In [32]:
```python
# Variable fuelType
cars['fuelType'].value_counts()
pd.crosstab(cars['fuelType'],columns='count',normalize=True)
sns.countplot(x= 'fuelType',data=cars)
sns.boxplot(x= 'fuelType',y='price',data=cars)
# fuelType affects price
```

Out[32]: <AxesSubplot:xlabel='fuelType', ylabel='price'>



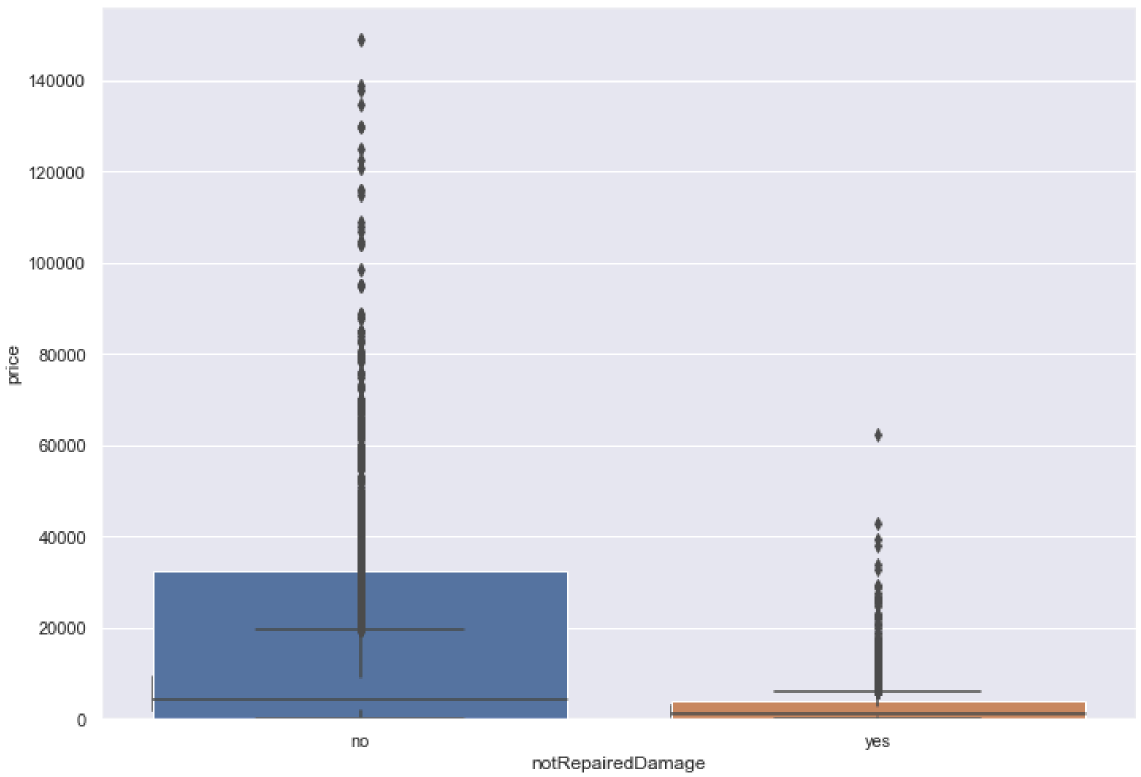
In [33]:
```python
# Variable brand
cars['brand'].value_counts()
pd.crosstab(cars['brand'],columns='count',normalize=True)
sns.countplot(x= 'brand',data=cars)
sns.boxplot(x= 'brand',y='price',data=cars)
# Cars are distributed over many brands
# Considered for modelling
```

Out[33]: <AxesSubplot:xlabel='brand', ylabel='price'>



In [34]:
```
# Variable notRepairedDamage
# yes- car is damaged but not rectified
# no- car was damaged but has been rectified
cars['notRepairedDamage'].value_counts()
pd.crosstab(cars['notRepairedDamage'],columns='count',normalize=True)
sns.countplot(x= 'notRepairedDamage',data=cars)
sns.boxplot(x= 'notRepairedDamage',y='price',data=cars)
# As expected, the cars that require the damages to be repaired
# fall under lower price ranges
```

Out[34]:
```
<AxesSubplot:xlabel='notRepairedDamage', ylabel='price'>
```



In [35]:
```
# Removing insignificant variables
col=['seller','offerType','abtest']
cars=cars.drop(columns=col, axis=1)
cars_copy=cars.copy()
```

In [36]:
```
# Correlation
cars_select1=cars.select_dtypes(exclude=[object])
correlation=cars_select1.corr()
round(correlation,3)
cars_select1.corr().loc[:,'price'].abs().sort_values(ascending=False)[1:]
```

Out[36]:
```
powerPS      0.575
kilometer    0.440
Age          0.336
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 rows with any missing value
2. Data obtained by imputing the missing values
```

OMITTING MISSING VALUES

In [37]:
```python
cars_omit=cars.dropna(axis=0)

# Converting categorical variables to dummy variables
cars_omit=pd.get_dummies(cars_omit,drop_first=True)
```

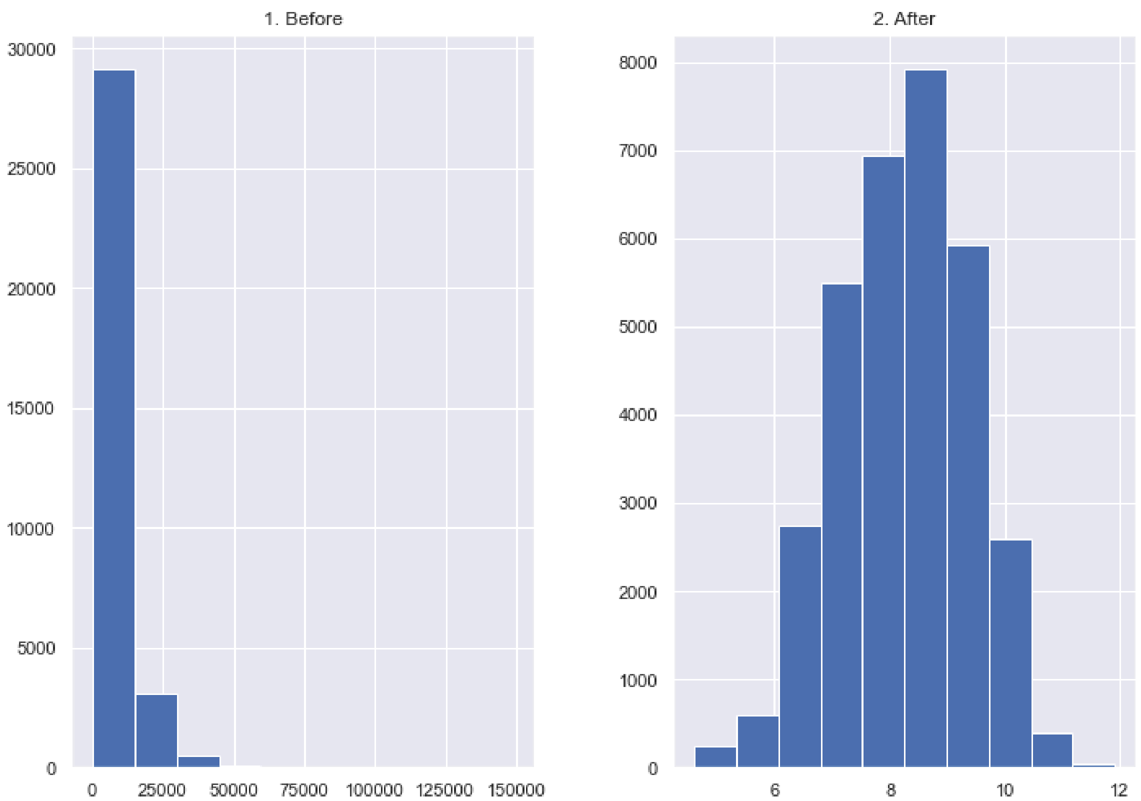
In [38]:
```python
# IMPORTING NECESSARY LIBRARIES
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

MODEL BUILDING WITH OMITTED DATA

In [39]:
```python
# Separating input and output features
x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
y1 = cars_omit['price']
```

In [40]:
```python
# Plotting the variable price
prices = pd.DataFrame({"1. Before":y1, "2. After":np.log(y1)})
prices.hist()
```

Out[40]:
```
array([[<AxesSubplot:title={'center':'1. Before'}>,
        <AxesSubplot:title={'center':'2. After'}>]], dtype=object)
```



In [41]:
```
# Transforming price as a logarithmic value
y1 = np.log(y1)
```

In [42]:
```
# Splitting data into test and train
X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_size=0.3, r
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

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

BASELINE MODEL FOR OMITTED 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
```

In [43]:
```
# finding the mean for test data value
base_pred = np.mean(y_test)
print(base_pred)
```

```
8.249615787653337
```

In [44]:
```
# Repeating same value till length of test data
base_pred = np.repeat(base_pred, len(y_test))
```

In [45]:
```
# finding the RMSE
base_root_mean_square_error = np.sqrt(mean_squared_error(y_test, base_pred))
print(base_root_mean_square_error)
```

```
1.1274483657478247
```

LINEAR REGRESSION WITH OMITTED DATA

In [46]:
```
# Setting intercept as true
lgr=LinearRegression(fit_intercept=True)
```

In [47]:
```
# Model
model_lin1=lgr.fit(X_train,y_train)
```

In [48]:
```
# Predicting model on test set
cars_predictions_lin1 = lgr.predict(X_test)
```

In [49]:
```
# Computing MSE and RMSE
lin_mse1 = mean_squared_error(y_test, cars_predictions_lin1)
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)
```

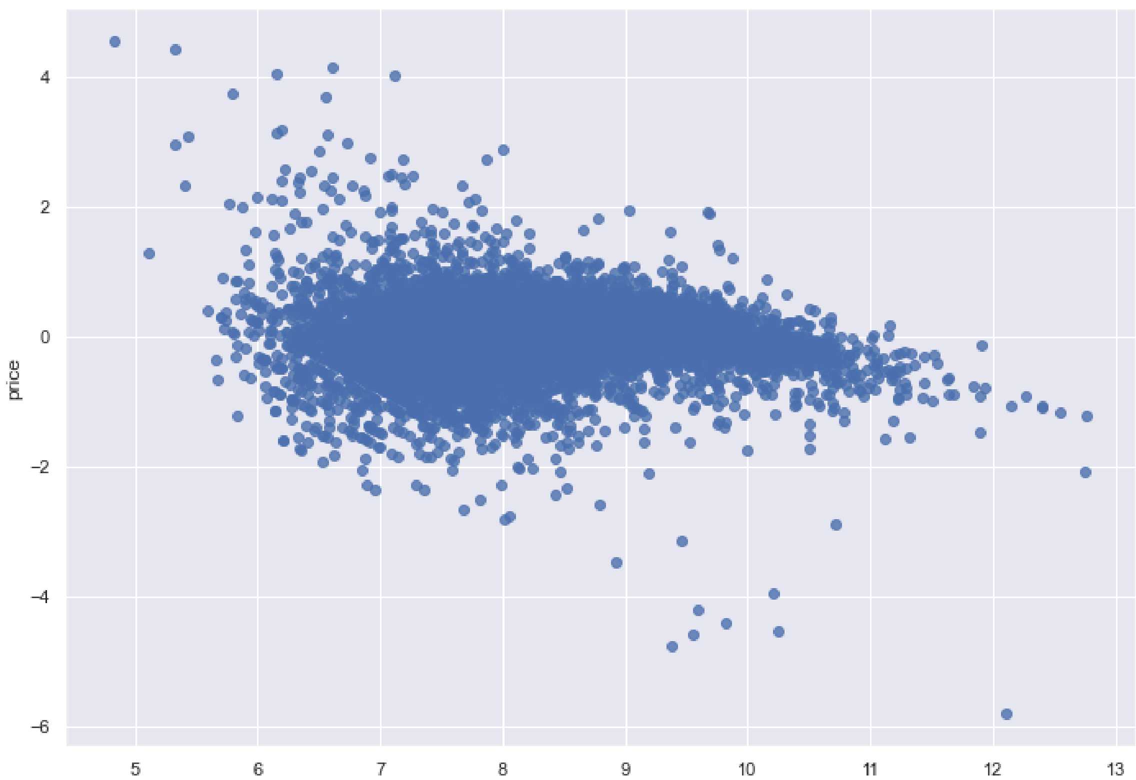
```
0.5455481266513857
```

In [50]:
```
# R squared value
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)
```

```
0.7658615091649229 0.7800936978183916
```

In [51]:
```
# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False)
residuals1.describe()
```

Out[51]:
```
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

In [52]:
```
# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)
```

In [53]:
```
# Model
model_rf1=rf.fit(X_train,y_train)
```

In [54]:
```
# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)
```

In [55]:
```
# Computing MSE and RMSE
rf_mse1 = mean_squared_error(y_test, cars_predictions_rf1)
rf_rmse1 = np.sqrt(rf_mse1)
print(rf_rmse1)
```

```
0.4360736289370223
```

In [56]:
```
# 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)
```

```
0.8504018147750623 0.9202494705146291
```

MODEL BUILDING WITH IMPUTED DATA

In [57]:
```
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()
```

Out[57]:
```
price                0
vehicleType          0
gearbox              0
powerPS              0
model                0
kilometer            0
fuelType             0
brand                0
notRepairedDamage    0
Age                  0
dtype: int64
```

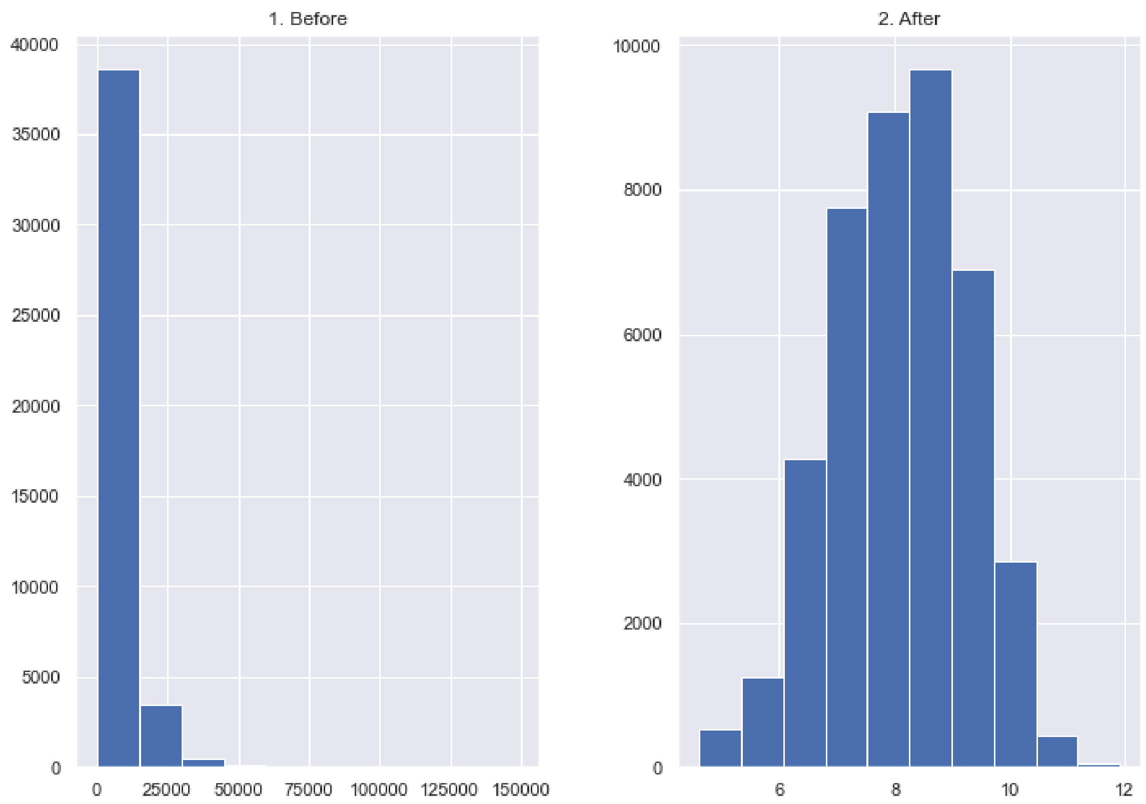
In [58]:
```
# Converting categorical variables to dummy variables
cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)
```

MODEL BUILDING WITH IMPUTED DATA

In [59]:
```
# Separating input and output feature
x2 = cars_imputed.drop(['price'], axis='columns', inplace=False)
y2 = cars_imputed['price']
```

In [60]:
```python
# Plotting the variable price
prices = pd.DataFrame({"1. Before":y2, "2. After":np.log(y2)})
prices.hist()
```

Out[60]:
```
array([[<AxesSubplot:title={'center':'1. Before'}>,
        <AxesSubplot:title={'center':'2. After'}>]], dtype=object)
```



In [61]:
```python
# Transforming price as a logarithmic value
y2 = np.log(y2)
```

In [62]:
```python
# Splitting data into test and train
X_train1, X_test1, y_train1, y_test1 = train_test_split(x2, y2, test_size=0.
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
```

In [63]:
```python
# finding the mean for test data value
base_pred = np.mean(y_test1)
print(base_pred)
```

```
8.068391740519193
```

In [64]:
```python
# Repeating same value till length of test data
base_pred = np.repeat(base_pred, len(y_test1))
```

In [65]:
```
# finding the RMSE
base_root_mean_square_error_imputed = np.sqrt(mean_squared_error(y_test1, ba

print(base_root_mean_square_error_imputed)
```

```
1.1884349112889792
```

LINEAR REGRESSION WITH IMPUTED DATA

In [66]:
```
# Setting intercept as true
lgr2=LinearRegression(fit_intercept=True)
```

In [67]:
```
# Model
model_lin2=lgr2.fit(X_train1,y_train1)
```

In [68]:
```
# Predicting model on test set
cars_predictions_lin2 = lgr2.predict(X_test1)
```

In [69]:
```
# Computing MSE and RMSE
lin_mse2 = mean_squared_error(y_test1, cars_predictions_lin2)
lin_rmse2 = np.sqrt(lin_mse2)
print(lin_rmse2)
```

```
0.6483956449231295
```

In [70]:
```
# R squared value
r2_lin_test2=model_lin2.score(X_test1,y_test1)
r2_lin_train2=model_lin2.score(X_train1,y_train1)
print(r2_lin_test2,r2_lin_train2)
```

```
0.7023339008631185 0.7071658736894363
```

RANDOM FOREST WITH IMPUTED DATA

In [71]:
```
# Model parameters
rf2 = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)
```

In [72]:
```
# Model
model_rf2=rf2.fit(X_train1,y_train1)
```

In [73]:
```
# Predicting model on test set
cars_predictions_rf2 = rf2.predict(X_test1)
```

In [74]:
```python
# Computing MSE and RMSE
rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)
rf_rmse2 = np.sqrt(rf_mse2)
print(rf_rmse2)
```

```
0.494313994408829
```

In [75]:
```python
# 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)
```

```
0.8269964521311131 0.9024289431669166
```

In [76]:
```python
# 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=
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=
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=  0.7658615091649229
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= 1.12
74483657478247
RMSE value for test from Linear Regression=  0.5455481266513857
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=  0.7071658736894363
R squared value for test from Linear Regression=  0.7023339008631185
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 imputed= 1.18
84349112889792
RMSE value for test from Linear Regression=  0.6483956449231295
RMSE value for test from Random Forest=  0.494313994408829
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```