Car Sale EDA

January 14, 2023

[1]:
```
print ('Hello World')
```

```
Hello World
```

0.1 CAR SALES EDA

[2]:
```
import pandas as pd                          # Implements milti_dimensional␣
 ↪array and matrics
import numpy as np                           # For data manipultion
import matplotlib.pyplot as plt              # Plotting library for Python␣
 ↪programming language.
from pandas_profiling import ProfileReport   # Importing pandas_profiling
import seaborn as sns                        # Provides a high level␣
 ↪interface for drawing attractive analysis
%matplotlib inline
sns.set()
from subprocess import check_output
```

[1]:
```
pip install pandas_profiling    ##Installing pandas_profiling packages
```

```
Collecting pandas_profiling
  Downloading pandas_profiling-3.6.2-py2.py3-none-any.whl (328 kB)
                          328.7/328.7 kB
25.7 MB/s eta 0:00:00
Collecting htmlmin==0.1.12
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (6.0)
Collecting phik<0.13,>=0.11.1
  Downloading
phik-0.12.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (679 kB)
                          679.5/679.5 kB
57.6 MB/s eta 0:00:00
Collecting typeguard<2.14,>=2.13.2
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.5.1)
```

```
Collecting pydantic<1.11,>=1.8.1
  Downloading
pydantic-1.10.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1
MB)
                           3.1/3.1 MB
83.7 MB/s eta 0:00:00:00:01
Requirement already satisfied: matplotlib<3.7,>=3.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (3.6.1)
Requirement already satisfied: numpy<1.24,>=1.16.0 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.23.4)
Requirement already satisfied: requests<2.29,>=2.24.0 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (2.28.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (3.1.2)
Collecting visions[type_image_path]==0.7.5
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
                          102.7/102.7 kB
18.3 MB/s eta 0:00:00
Requirement already satisfied: tqdm<4.65,>=4.48.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (4.64.1)
Collecting multimethod<1.10,>=1.4
  Downloading multimethod-1.9.1-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<0.14,>=0.13.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.13.2)
Requirement already satisfied: scipy<1.10,>=1.4.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.9.3)
Requirement already satisfied: seaborn<0.13,>=0.10.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.12.0)
Requirement already satisfied: attrs>=19.3.0 in /opt/conda/lib/python3.10/site-
packages (from visions[type_image_path]==0.7.5->pandas_profiling) (22.1.0)
Collecting tangled-up-in-unicode>=0.0.4
  Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
                           4.7/4.7 MB
84.3 MB/s eta 0:00:00ta 0:00:01
Requirement already satisfied: networkx>=2.4 in
/opt/conda/lib/python3.10/site-packages (from
visions[type_image_path]==0.7.5->pandas_profiling) (2.8.7)
Requirement already satisfied: Pillow in /opt/conda/lib/python3.10/site-packages
(from visions[type_image_path]==0.7.5->pandas_profiling) (9.2.0)
Collecting imagehash
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
                          296.5/296.5 kB
42.9 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.10/site-packages (from
jinja2<3.2,>=2.11.1->pandas_profiling) (2.1.1)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from
```

```
matplotlib<3.7,>=3.2->pandas_profiling) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (1.4.4)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (4.38.0)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib<3.7,>=3.2->pandas_profiling) (0.11.0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-
packages (from pandas!=1.4.0,<1.6,>1.1->pandas_profiling) (2022.5)
Requirement already satisfied: joblib>=0.14.1 in /opt/conda/lib/python3.10/site-
packages (from phik<0.13,>=0.11.1->pandas_profiling) (1.2.0)
Requirement already satisfied: typing-extensions>=4.2.0 in
/opt/conda/lib/python3.10/site-packages (from
pydantic<1.11,>=1.8.1->pandas_profiling) (4.4.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from
requests<2.29,>=2.24.0->pandas_profiling) (1.26.11)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-
packages (from requests<2.29,>=2.24.0->pandas_profiling) (3.4)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.10/site-packages (from
requests<2.29,>=2.24.0->pandas_profiling) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from
requests<2.29,>=2.24.0->pandas_profiling) (2022.9.24)
Requirement already satisfied: patsy>=0.5.2 in /opt/conda/lib/python3.10/site-
packages (from statsmodels<0.14,>=0.13.2->pandas_profiling) (0.5.3)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages
(from patsy>=0.5.2->statsmodels<0.14,>=0.13.2->pandas_profiling) (1.16.0)
Requirement already satisfied: PyWavelets in /opt/conda/lib/python3.10/site-
packages (from imagehash->visions[type_image_path]==0.7.5->pandas_profiling)
(1.3.0)
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) … done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl
size=27082
sha256=76d7baf171ec0204fafbaddd83d27e80ac59b5bccaea8585d230f36c9f42c85c
```

```
  Stored in directory: /home/jovyan/.cache/pip/wheels/ea/1c/a8/5cec3479cd45136a7
111e2d96aac299b219b199c411665250b
Successfully built htmlmin
Installing collected packages: htmlmin, typeguard, tangled-up-in-unicode,
pydantic, multimethod, imagehash, visions, phik, pandas_profiling
Successfully installed htmlmin-0.1.12 imagehash-4.3.1 multimethod-1.9.1
pandas_profiling-3.6.2 phik-0.12.3 pydantic-1.10.4 tangled-up-in-unicode-0.2.0
typeguard-2.13.3 visions-0.7.5
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
# Loading dataset
carsales_data = pd.read_excel(r"Car_Sales.xlsx")
```

[4]:
```
# Viewing start 5 rows from dataset
carsales_data.head()
```

[4]:
```
             car    price       body  mileage  engV engType registration  \
0           Ford  15500.0  crossover       68   2.5     Gas          yes
1  Mercedes-Benz  20500.0      sedan      173   1.8     Gas          yes
2  Mercedes-Benz  35000.0      other      135   5.5  Petrol          yes
3  Mercedes-Benz  17800.0        van      162   1.8  Diesel          yes
4  Mercedes-Benz  33000.0      vagon       91   NaN   Other          yes

   year    model  drive
0  2010     Kuga   full
1  2011  E-Class   rear
2  2008   CL 550   rear
3  2012    B 180  front
4  2013  E-Class    NaN
```

[5]:
```
# Viewing last 5 rows from dataset
carsales_data.tail()
```

[5]:
```
                car    price       body  mileage  engV engType registration  \
9571        Hyundai  14500.0  crossover      140   2.0     Gas          yes
9572     Volkswagen   2200.0      vagon      150   1.6  Petrol          yes
9573  Mercedes-Benz  18500.0  crossover      180   3.5  Petrol          yes
9574          Lexus  16999.0      sedan      150   3.5     Gas          yes
9575           Audi  22500.0      other       71   3.6  Petrol          yes

      year       model  drive
9571  2011      Tucson  front
9572  1986  Passat B2  front
9573  2008      ML 350   full
9574  2008      ES 350  front
9575  2007          Q7   full
```

[6]:
```
## Viewing Shape of dataset
carsales_data.shape
```

[6]:
```
(9576, 10)
```

CarSales_Data has 9576 rows and 10 Columns.

[7]:
```
# Viewing columns name from dataset
carsales_data.columns
```

[7]:
```
Index(['car', 'price', 'body', 'mileage', 'engV', 'engType', 'registration',
       'year', 'model', 'drive'],
      dtype='object')
```

[8]:
```
# Viewing Statical Values from Dataset
carsales_data.describe()
```

[8]:

	price	mileage	engV	year
count	9576.000000	9576.000000	9142.000000	9576.000000
mean	15633.317316	138.862364	2.646344	2006.605994
std	24106.523436	98.629754	5.927699	7.067924
min	0.000000	0.000000	0.100000	1953.000000
25%	4999.000000	70.000000	1.600000	2004.000000
50%	9200.000000	128.000000	2.000000	2008.000000
75%	16700.000000	194.000000	2.500000	2012.000000
max	547800.000000	999.000000	99.990000	2016.000000

[9]:
```
# Viewing All Values from Dataset
carsales_data.describe(include="all")
```

[9]:

	car	price	body	mileage	engV	engType \
count	9576	9576.000000	9576	9576.000000	9142.000000	9576
unique	87	NaN	6	NaN	NaN	4
top	Volkswagen	NaN	sedan	NaN	NaN	Petrol
freq	936	NaN	3646	NaN	NaN	4379
mean	NaN	15633.317316	NaN	138.862364	2.646344	NaN
std	NaN	24106.523436	NaN	98.629754	5.927699	NaN
min	NaN	0.000000	NaN	0.000000	0.100000	NaN
25%	NaN	4999.000000	NaN	70.000000	1.600000	NaN
50%	NaN	9200.000000	NaN	128.000000	2.000000	NaN
75%	NaN	16700.000000	NaN	194.000000	2.500000	NaN
max	NaN	547800.000000	NaN	999.000000	99.990000	NaN

	registration	year	model	drive
count	9576	9576.000000	9576	9065
unique	2	NaN	863	3
top	yes	NaN	E-Class	front

```
freq            9015          NaN       199   5188
mean             NaN  2006.605994       NaN    NaN
std              NaN     7.067924       NaN    NaN
min              NaN  1953.000000       NaN    NaN
25%              NaN  2004.000000       NaN    NaN
50%              NaN  2008.000000       NaN    NaN
75%              NaN  2012.000000       NaN    NaN
max              NaN  2016.000000       NaN    NaN
```

[10]:
```
# Viewing dataset sort by price in descending order.
carsales_data.sort_values(by=['price'],ascending= False)
```

[10]:
```
                car     price       body  mileage  engV engType registration  \
7621        Bentley  547800.0      sedan        0  6.75  Petrol          yes
7914        Bentley  499999.0  crossover        0  6.00  Petrol          yes
1611        Bentley  499999.0  crossover        0  6.00  Petrol          yes
4134        Bentley  449999.0  crossover        0  6.00  Petrol          yes
4325  Mercedes-Benz  300000.0      sedan       68  6.00  Petrol          yes
…               …         …         …       …     …     …        …
70    Mercedes-Benz       0.0  crossover        0  3.00  Diesel          yes
158      Land Rover       0.0  crossover       45  3.00  Petrol          yes
4786  Mercedes-Benz       0.0  crossover       27  3.00  Diesel          yes
4784    Rolls-Royce       0.0      sedan       22  6.75   Other          yes
215   Mercedes-Benz       0.0      sedan       62  3.00  Diesel          yes

      year              model drive
7621  2016           Mulsanne  rear
7914  2016           Bentayga  full
1611  2016           Bentayga  full
4134  2016           Bentayga  full
4325  2011              S 600   NaN
…      …                  …    …
70    2016          GLE-Class  full
158   2014  Range Rover Sport  full
4786  2015              G 350  full
4784  2008            Phantom  rear
215   2013              S 350  rear

[9576 rows x 10 columns]
```

[11]:
```
# Viewing dataset sort by price in descending order starting 10 record .
carsales_data.sort_values(by=['price'],ascending= False).head(10)
```

[11]:
```
                car     price       body  mileage  engV engType registration  \
7621        Bentley  547800.0      sedan        0  6.75  Petrol          yes
7914        Bentley  499999.0  crossover        0  6.00  Petrol          yes
1611        Bentley  499999.0  crossover        0  6.00  Petrol          yes
```

```
4134        Bentley  449999.0  crossover        0  6.00  Petrol           yes
4325  Mercedes-Benz  300000.0      sedan       68  6.00  Petrol           yes
5849  Mercedes-Benz  300000.0      other       37  5.00  Petrol           yes
1891  Mercedes-Benz  295000.0      sedan       29  6.00  Petrol           yes
2165  Mercedes-Benz  295000.0      sedan       29  6.00  Petrol           yes
8205     Land Rover  285000.0  crossover        0  5.00  Petrol           yes
1478        Bentley  259000.0      sedan        0  6.00  Petrol           yes

      year        model drive
7621  2016     Mulsanne  rear
7914  2016     Bentayga  full
1611  2016     Bentayga  full
4134  2016     Bentayga  full
4325  2011        S 600   NaN
5849  2012        G 500  full
1891  2011        S 600  rear
2165  2011      S-Guard  rear
8205  2016  Range Rover  full
1478  2014  Flying Spur  full
```

[12]:
```python
# Viewing dataset sort by price in ascending order first 10 record .
carsales_data.sort_values(by=['price'],ascending= True).head(10)
```

[12]:
```
                car  price       body  mileage  engV engType registration  \
4864            BMW    0.0  crossover       16   NaN   Other          yes
7496         Subaru    0.0  crossover        1   2.0  Diesel          yes
3625         Daewoo    0.0      sedan       39   1.6     Gas          yes
7497  Mercedes-Benz    0.0      sedan       42   3.0  Diesel          yes
8512            GAZ    0.0      sedan        1   2.4  Petrol          yes
8824         Fisker    0.0      other      100   NaN   Other          yes
8026         Toyota    0.0  crossover        1   2.2  Diesel          yes
7580  Mercedes-Benz    0.0      sedan        0   3.0  Petrol          yes
1386        Hyundai    0.0  crossover       39   2.2  Diesel          yes
8015  Mercedes-Benz    0.0      sedan       16   3.0  Diesel          yes

      year     model  drive
4864  2016     X5 M    NaN
7496  2016  Forester   full
3625  2012     Nexia  front
7497  2014     S 350   full
8512  1969        21  front
8824  2001     Karma    NaN
8026  2016     Rav 4   full
7580  2016     S 400   full
1386  2012  Santa FE   full
8015  2014     S 350   full
```

[13]:
```
# creating a group for most selling cars form data sets.
carsales_data.groupby('car')['price'].count().sort_values(ascending=False).
 ↪head(5)
```

[13]:
```
car
Volkswagen       936
Mercedes-Benz    921
BMW              694
Toyota           541
VAZ              489
Name: price, dtype: int64
```

It has been observed that top 3 selling cars are Volkswagen, Mercedes-Benz, BMW .

[14]:
```
# which cars has most sales parcentile.
carsales_data['car'].value_counts(normalize=True) * 100
```

[14]:
```
Volkswagen       9.774436
Mercedes-Benz    9.617794
BMW              7.247285
Toyota           5.649541
VAZ              5.106516
                   …
ZX               0.010443
Other-Retro      0.010443
Mercury          0.010443
Maserati         0.010443
Buick            0.010443
Name: car, Length: 87, dtype: float64
```

[15]:
```
# creating a group for most selling cars by body type.
carsales_data.groupby('car')['body'].value_counts(normalize=True).head()
```

[15]:
```
car         body
Acura       sedan        0.538462
            crossover    0.461538
Alfa Romeo  sedan        0.545455
            hatch        0.363636
            vagon        0.090909
Name: body, dtype: float64
```

[16]:
```
# this will return us correlation between data
carsales_data.corr()
```

```
/tmp/ipykernel_84/3118432294.py:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
```

```
to silence this warning.
  carsales_data.corr()
```

[16]:

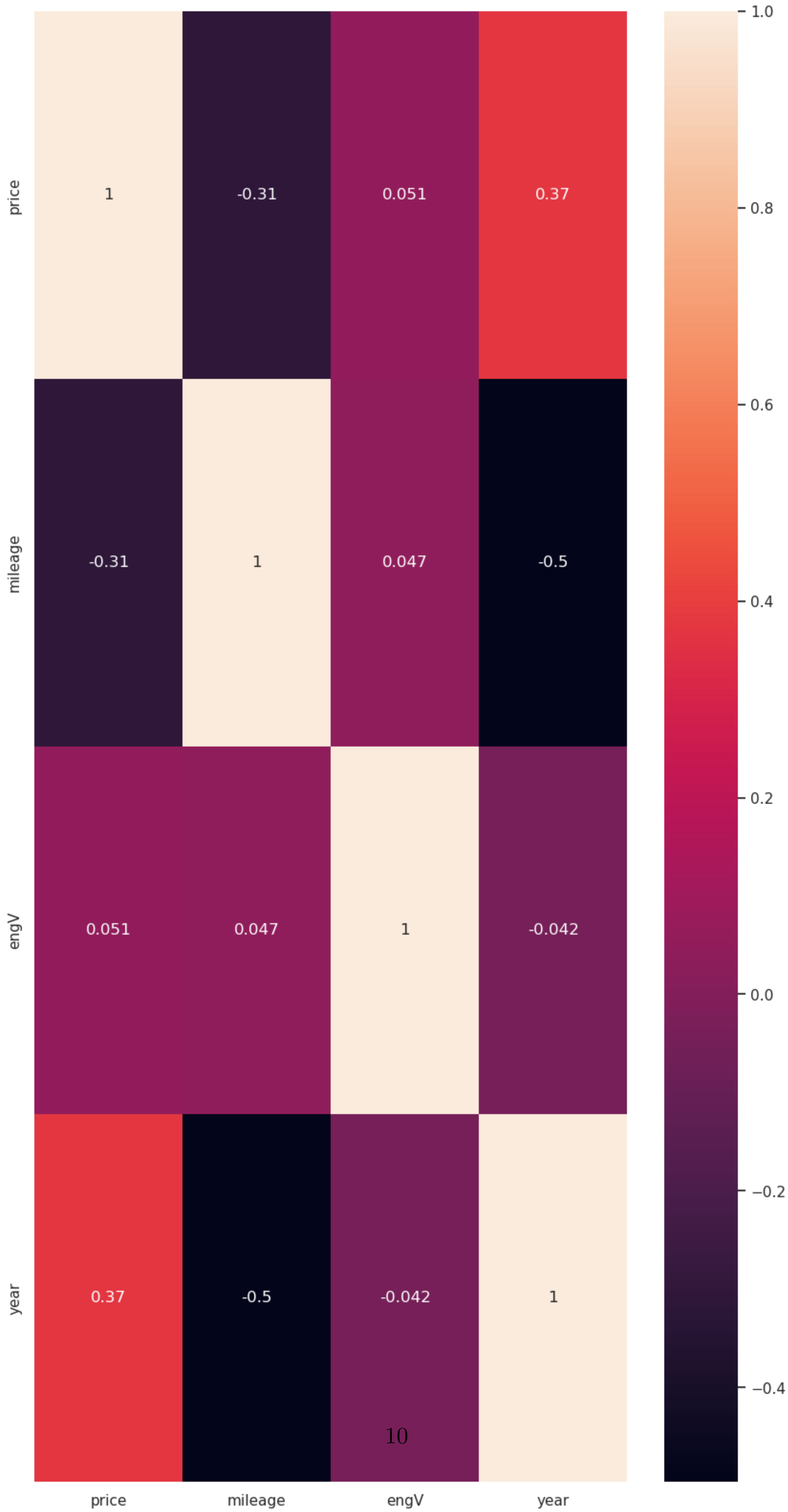
	price	mileage	engV	year
price	1.000000	-0.312415	0.051070	0.370379
mileage	-0.312415	1.000000	0.047070	-0.495599
engV	0.051070	0.047070	1.000000	-0.042251
year	0.370379	-0.495599	-0.042251	1.000000

[17]:

```
# using seaborn for a high level interface for drawing attractive and␣
↪informative statical graph using heatmap.
import seaborn as sns
sns.set()
plt.subplots(figsize=(10,20))
sns.heatmap(carsales_data.corr(),annot=True)
```

```
/tmp/ipykernel_84/973853901.py:5: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(carsales_data.corr(),annot=True)
```

[17]: <AxesSubplot: >


price
mileage
engV
year
1
-0.31
0.051
0.37
-0.31
1
0.047
-0.5
0.051
0.047
1
-0.042
0.37
-0.5
-0.042
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

[18]:
```
# viewing data Dtype.
carsales_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9576 entries, 0 to 9575
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   car           9576 non-null   object
 1   price         9576 non-null   float64
 2   body          9576 non-null   object
 3   mileage       9576 non-null   int64
 4   engV          9142 non-null   float64
 5   engType       9576 non-null   object
 6   registration  9576 non-null   object
 7   year          9576 non-null   int64
 8   model         9576 non-null   object
 9   drive         9065 non-null   object
dtypes: float64(2), int64(2), object(6)
memory usage: 748.2+ KB
```

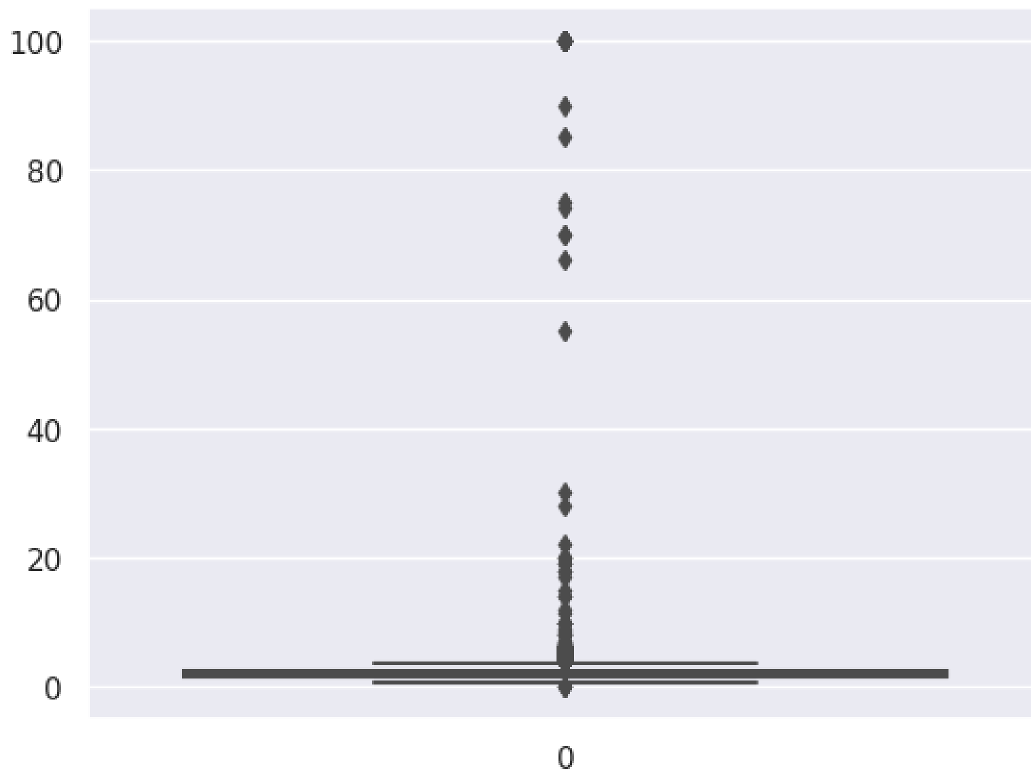
[19]:
```
# viewing how much null data in dataset.
carsales_data.isnull().sum()
```

[19]:
```
car               0
price             0
body              0
mileage           0
engV            434
engType           0
registration      0
year              0
model             0
drive           511
dtype: int64
```

From the above output we can see that engV and drive columns contains maximum null values. We will see how to deal with them.

[20]:
```
# we are seeing outliers from engV.
sns.boxplot(data=carsales_data.engV)
```

[20]:
```
<AxesSubplot: >
```



1. Fill Missing
2. sort().according to price (Asending)
3. Group by drive
4. Dummy

3.2 Pre Profiling

[21]:
```
pip install pandas_profiling    ##Installing pandas_profiling packages
```

```
Requirement already satisfied: pandas_profiling in
/opt/conda/lib/python3.10/site-packages (3.6.2)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (3.1.2)
Requirement already satisfied: tqdm<4.65,>=4.48.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (4.64.1)
Requirement already satisfied: typeguard<2.14,>=2.13.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (2.13.3)
Requirement already satisfied: statsmodels<0.14,>=0.13.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.13.2)
Requirement already satisfied: visions[type_image_path]==0.7.5 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.7.5)
Requirement already satisfied: phik<0.13,>=0.11.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.12.3)
```

```
Requirement already satisfied: requests<2.29,>=2.24.0 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (2.28.1)
Requirement already satisfied: multimethod<1.10,>=1.4 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.9.1)
Requirement already satisfied: pydantic<1.11,>=1.8.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.10.4)
Requirement already satisfied: scipy<1.10,>=1.4.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.9.3)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (6.0)
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.5.1)
Requirement already satisfied: seaborn<0.13,>=0.10.1 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.12.0)
Requirement already satisfied: numpy<1.24,>=1.16.0 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (1.23.4)
Requirement already satisfied: matplotlib<3.7,>=3.2 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (3.6.1)
Requirement already satisfied: htmlmin==0.1.12 in
/opt/conda/lib/python3.10/site-packages (from pandas_profiling) (0.1.12)
Requirement already satisfied: tangled-up-in-unicode>=0.0.4 in
/opt/conda/lib/python3.10/site-packages (from
visions[type_image_path]==0.7.5->pandas_profiling) (0.2.0)
Requirement already satisfied: attrs>=19.3.0 in /opt/conda/lib/python3.10/site-
packages (from visions[type_image_path]==0.7.5->pandas_profiling) (22.1.0)
Requirement already satisfied: networkx>=2.4 in /opt/conda/lib/python3.10/site-
packages (from visions[type_image_path]==0.7.5->pandas_profiling) (2.8.7)
Requirement already satisfied: Pillow in /opt/conda/lib/python3.10/site-packages
(from visions[type_image_path]==0.7.5->pandas_profiling) (9.2.0)
Requirement already satisfied: imagehash in /opt/conda/lib/python3.10/site-
packages (from visions[type_image_path]==0.7.5->pandas_profiling) (4.3.1)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.10/site-packages (from
jinja2<3.2,>=2.11.1->pandas_profiling) (2.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (1.4.4)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (3.0.9)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from
```

```
matplotlib<3.7,>=3.2->pandas_profiling) (21.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib<3.7,>=3.2->pandas_profiling) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib<3.7,>=3.2->pandas_profiling) (1.0.5)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-
packages (from pandas!=1.4.0,<1.6,>1.1->pandas_profiling) (2022.5)
Requirement already satisfied: joblib>=0.14.1 in /opt/conda/lib/python3.10/site-
packages (from phik<0.13,>=0.11.1->pandas_profiling) (1.2.0)
Requirement already satisfied: typing-extensions>=4.2.0 in
/opt/conda/lib/python3.10/site-packages (from
pydantic<1.11,>=1.8.1->pandas_profiling) (4.4.0)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.10/site-packages (from
requests<2.29,>=2.24.0->pandas_profiling) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from
requests<2.29,>=2.24.0->pandas_profiling) (2022.9.24)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-
packages (from requests<2.29,>=2.24.0->pandas_profiling) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from
requests<2.29,>=2.24.0->pandas_profiling) (1.26.11)
Requirement already satisfied: patsy>=0.5.2 in /opt/conda/lib/python3.10/site-
packages (from statsmodels<0.14,>=0.13.2->pandas_profiling) (0.5.3)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages
(from patsy>=0.5.2->statsmodels<0.14,>=0.13.2->pandas_profiling) (1.16.0)
Requirement already satisfied: PyWavelets in /opt/conda/lib/python3.10/site-
packages (from imagehash->visions[type_image_path]==0.7.5->pandas_profiling)
(1.3.0)
Note: you may need to restart the kernel to use updated packages.
```

[22]:
```
import pandas as pd                                  # Implements milti_dimensional␣
 ↪array and matrics
import numpy as np                                   # For data manipultion
import matplotlib.pyplot as plt                      # Plotting library for Python␣
 ↪programming language.
import pandas_profiling         # Importing pandas_profiling
import seaborn as sns                                # Provides a high level␣
 ↪interface for drawing attractive analysis
%matplotlib inline
sns.set()
from subprocess import check_output
```

[23]:
```
# Now performing pandas profiling to understand data better.
#profile = pandas_profiling.ProfileReport(carsales_data)
```

```
#profile
```

[24]:
```
# Now saving a copy of profile report in html format.
#profile.to_file(output_file="CarSales_before_preprocessing.html")
```

3.3 Preprocessing

. Dealing With duplicate rows

. Find number of duplicate rows in the dataset.

. Print the duplicate entries and analyze.

. Drop the duplicate entries from the dataset.

[25]:
```
## creating a duplicate dataset for doing Preprocessing
carsales_data_prepro = carsales_data
```

[26]:
```
## # This will print the first n rows of the Dataset
carsales_data_prepro.head()
```

[26]:
```
             car    price       body  mileage  engV engType registration  \
0           Ford  15500.0  crossover       68   2.5     Gas          yes
1  Mercedes-Benz  20500.0      sedan      173   1.8     Gas          yes
2  Mercedes-Benz  35000.0      other      135   5.5  Petrol          yes
3  Mercedes-Benz  17800.0        van      162   1.8  Diesel          yes
4  Mercedes-Benz  33000.0      vagon       91   NaN   Other          yes

   year    model  drive
0  2010     Kuga   full
1  2011  E-Class   rear
2  2008   CL 550   rear
3  2012    B 180  front
4  2013  E-Class    NaN
```

[27]:
```
# This will print the last n rows of the Dataset
carsales_data_prepro.tail()
```

[27]:
```
                car    price       body  mileage  engV engType registration  \
9571        Hyundai  14500.0  crossover      140   2.0     Gas          yes
9572     Volkswagen   2200.0      vagon      150   1.6  Petrol          yes
9573  Mercedes-Benz  18500.0  crossover      180   3.5  Petrol          yes
9574          Lexus  16999.0      sedan      150   3.5     Gas          yes
9575           Audi  22500.0      other       71   3.6  Petrol          yes

      year      model  drive
9571  2011     Tucson  front
9572  1986  Passat B2  front
9573  2008     ML 350   full
```

```
9574  2008     ES 350  front
9575  2007         Q7   full
```

[28]:
```
## This will print the missing value parcentage from every column.
missing1 = carsales_data_prepro.isnull().sum()
missing = (carsales_data_prepro.isnull().sum()/len(carsales_data_prepro))*100
miss_data = pd.concat([missing1,missing],axis=1, keys=['Total', '%'])
print(miss_data)
```

```
             Total         %
car              0  0.000000
price            0  0.000000
body             0  0.000000
mileage          0  0.000000
engV           434  4.532164
engType          0  0.000000
registration     0  0.000000
year             0  0.000000
model            0  0.000000
drive          511  5.336257
```

[29]:
```
# This will return us total duplicate rows from data frame
carsales_data_prepro.duplicated().sum()
```

[29]: 113

113 Rows are duplicate in data frame .

[30]:
```
## This will return us duplicate rows from data
carsales_data_prepro.loc[carsales_data_prepro.duplicated(), :]       #.head()␣
 ↪for viem n numbers of rows
```

[30]:
```
                car     price       body  mileage  engV engType registration  \
18           Nissan   16600.0  crossover       83   2.0  Petrol          yes
42    Mercedes-Benz   20400.0      sedan      190   1.8     Gas          yes
70    Mercedes-Benz       0.0  crossover        0   3.0  Diesel          yes
86           Toyota  103999.0  crossover        0   4.5  Diesel          yes
98    Mercedes-Benz   20400.0      sedan      190   1.8     Gas          yes
…               …        …         …      …    …       …
9156     Volkswagen   15700.0      sedan      110   1.8  Petrol          yes
9163  Mercedes-Benz   20500.0      sedan      222   5.5  Petrol          yes
9164            VAZ    3900.0      hatch      121   1.4  Petrol          yes
9169        Hyundai   12900.0  crossover       49   2.7  Petrol          yes
9477            BMW   77777.0      sedan        8   4.4  Petrol          yes

      year            model  drive
18    2013          X-Trail   full
```

```
42    2011          E-Class   rear
70    2016        GLE-Class   full
86    2016  Land Cruiser 200  full
98    2011          E-Class   rear
…     …                   …     …
9156  2011        Passat B7  front
9163  2006            S 500   rear
9164  2008             1119  front
9169  2008           Tucson   full
9477  2014              750   full

[113 rows x 10 columns]
```

[31]:
```python
## Droping duplicates rows from data frame
carsales_data_prepro.drop_duplicates(inplace=True)
```

[32]:
```python
## checking for duplicates rows from datasets.
carsales_data_prepro.loc[carsales_data.duplicated(), :]
```

[32]:
```
Empty DataFrame
Columns: [car, price, body, mileage, engV, engType, registration, year, model,
drive]
Index: []
```

[33]:
```python
# Viewing Data Frame
carsales_data_prepro
```

[33]:
```
                car    price       body  mileage  engV engType registration  \
0              Ford  15500.0  crossover       68   2.5     Gas          yes
1     Mercedes-Benz  20500.0      sedan      173   1.8     Gas          yes
2     Mercedes-Benz  35000.0      other      135   5.5  Petrol          yes
3     Mercedes-Benz  17800.0        van      162   1.8  Diesel          yes
4     Mercedes-Benz  33000.0      vagon       91   NaN   Other          yes
…               …        …          …        …     …     …            …
9571        Hyundai  14500.0  crossover      140   2.0     Gas          yes
9572     Volkswagen   2200.0      vagon      150   1.6  Petrol          yes
9573  Mercedes-Benz  18500.0  crossover      180   3.5  Petrol          yes
9574          Lexus  16999.0      sedan      150   3.5     Gas          yes
9575           Audi  22500.0      other       71   3.6  Petrol          yes

      year     model  drive
0     2010      Kuga   full
1     2011   E-Class   rear
2     2008    CL 550   rear
3     2012     B 180  front
4     2013   E-Class    NaN
…     …          …     …
```

```
9571  2011     Tucson  front
9572  1986  Passat B2  front
9573  2008     ML 350   full
9574  2008     ES 350  front
9575  2007         Q7   full

[9463 rows x 10 columns]
```

[34]:
```
# This will return mode of Drive
b = carsales_data_prepro["drive"].mode()
b
```

[34]:
```
0    front
Name: drive, dtype: object
```

[35]:
```
# this will fill null value of drive fill na.
carsales_data_prepro["drive"]=carsales_data_prepro["drive"].fillna("front")
carsales_data_prepro.isnull().sum()
```

[35]:
```
car               0
price             0
body              0
mileage           0
engV            434
engType           0
registration      0
year              0
model             0
drive             0
dtype: int64
```

[36]:
```
## this will return us duplicate rows .
print(carsales_data_prepro.duplicated().sum())
```

```
0
```

[37]:
```
# This will keep duplicate values.
carsales_data.loc[carsales_data.duplicated(keep=False), :]
```

[37]:
```
Empty DataFrame
Columns: [car, price, body, mileage, engV, engType, registration, year, model,
drive]
Index: []
```

Duplicate entries are removed now.

.Dealing with missing values

`.434 missing entries of engV. Replace it with median value of engV from the same Car and body g`

.511 missing entries of drive. Replace it with most common value of drive from the same Car anc

.Drop entries having price is 0 or less than 0.

.CarSales_Data_copy.groupby(['car', 'body'])['engV']

[38]:
```
# using group by in car and body type for engV columns NAN values.
carsales_data_prepro.groupby(['car','body'])['engV'].head()
```

[38]:
```
0        2.5
1        1.8
2        5.5
3        1.8
4        NaN
        …
9499     3.7
9501     1.2
9508    19.0
9539     1.5
9566     NaN
Name: engV, Length: 1018, dtype: float64
```

[39]:
```
# this will return us engV columns median value fill it in every NAN values on␣
 ↪engV columns
carsales_data_prepro['engV'] = carsales_data_prepro.groupby(['car',␣
 ↪'body'])['engV'].transform(lambda x: x.fillna(x.median()))
```

Now let's check if the missing values of engV has been replaced.

[40]:
```
# This will return us null values of every columns.
carsales_data_prepro.isnull().sum()
```

[40]:
```
car              0
price            0
body             0
mileage          0
engV            10
engType          0
registration     0
year             0
model            0
drive            0
dtype: int64
```

424 missing values of engV has been replaced however, still 10 entries are left as missing. Let's see the missing value data.

[41]:
```
# This will return us which 10 value is null engV  column.
carsales_data_prepro[carsales_data_prepro.engV.isnull()]
```

[41]:
```
          car     price       body  mileage  engV engType registration  year  \
319     Tesla   58000.0      hatch       52   NaN   Other          yes  2013
1437    Tesla  178500.0  crossover        0   NaN   Other          yes  2016
2486    Tesla  185000.0  crossover        1   NaN   Other          yes  2016
5084      GAZ       0.0  crossover        1   NaN  Petrol          yes  1963
6773      UAZ    3000.0      other        1   NaN   Other          yes  1985
8569    Tesla  176900.0  crossover        0   NaN   Other          yes  2016
8824   Fisker       0.0      other      100   NaN   Other          yes  2001
8905  Changan    6028.0  crossover      101   NaN   Other          yes  2005
9360   Barkas    5500.0        van       80   NaN  Petrol          yes  2015
9566      UAZ     850.0        van      255   NaN   Other          yes  1981

        model  drive
319   Model S  front
1437  Model X   full
2486  Model X   full
5084       69   full
6773     3303   full
8569  Model X   full
8824    Karma  front
8905    Ideal  front
9360    B1000  front
9566     3962  front
```

Replacing NaN values of drive with most common values of drive from Car and body group.

[42]:
```
#Creating a function for filling drive null values
def f(x):
    if x.count()<=0:
        return np.nan
    return x.value_counts().index[0]

carsales_data_prepro['drive'] = carsales_data_prepro['drive'].
 ↪fillna(carsales_data_prepro.groupby(['car','body'])['drive'].transform(f))
```

[43]:
```
# finding null values from drive columns
carsales_data_prepro[carsales_data_prepro.drive.isnull()]
```

[43]:
```
Empty DataFrame
Columns: [car, price, body, mileage, engV, engType, registration, year, model,
drive]
Index: []
```

Let's check the count of NaN values of engV and drive.

[44]:
```
# finding null values from data frame
carsales_data_prepro.isnull().sum()
```

[44]:
```
car              0
price            0
body             0
mileage          0
engV            10
engType          0
registration     0
year             0
model            0
drive            0
dtype: int64
```

Dropping remaining NaN values of engV and Drive.

[45]:
```
# creating a copy of data frame
carsales_data_droping = carsales_data_prepro
```

[46]:
```
## droping NAN value from engV columns and drive columns
carsales_data_droping.dropna(subset=['engV'],inplace=True)
carsales_data_droping.dropna(subset=['drive'],inplace=True)
carsales_data_droping.isnull().sum()
```

[46]:
```
car             0
price           0
body            0
mileage         0
engV            0
engType         0
registration    0
year            0
model           0
drive           0
dtype: int64
```

Droping entries with price <= 0

[47]:
```
# droping all values where price <= 0
carsales_data_droping = carsales_data_droping.
↪drop(carsales_data_droping[carsales_data_droping.price <= 0 ].index)
```

[48]:
```
# using count function to see price columns has don't have any 0 values
carsales_data_droping.price[carsales_data_droping.price == 0].count()
```

[48]:
```
0
```

[49]:
```
# this replace all milage columns 0 value with mileage column medain values
b = carsales_data_droping["mileage"].median()
carsales_data_droping["mileage"] = carsales_data_droping["mileage"].replace(0,b)
```

[50]:
```
# this will return us all values where mileage columns has value == 0
carsales_data_droping[carsales_data_droping.mileage == 0]
```

[50]:
```
Empty DataFrame
Columns: [car, price, body, mileage, engV, engType, registration, year, model,
drive]
Index: []
```

[51]:
```
## viewing all cleaned data
carsales_data_droping
```

[51]:
```
                car    price       body  mileage  engV engType registration  \
0              Ford  15500.0  crossover       68   2.5     Gas          yes
1     Mercedes-Benz  20500.0      sedan      173   1.8     Gas          yes
2     Mercedes-Benz  35000.0      other      135   5.5  Petrol          yes
3     Mercedes-Benz  17800.0        van      162   1.8  Diesel          yes
4     Mercedes-Benz  33000.0      vagon       91   2.3   Other          yes
…               …       …        …       …    …     …              …
9571        Hyundai  14500.0  crossover      140   2.0     Gas          yes
9572     Volkswagen   2200.0      vagon      150   1.6  Petrol          yes
9573  Mercedes-Benz  18500.0  crossover      180   3.5  Petrol          yes
9574          Lexus  16999.0      sedan      150   3.5     Gas          yes
9575           Audi  22500.0      other       71   3.6  Petrol          yes

      year      model  drive
0     2010       Kuga   full
1     2011    E-Class   rear
2     2008     CL 550   rear
3     2012      B 180  front
4     2013    E-Class  front
…     …          …     …
9571  2011     Tucson  front
9572  1986  Passat B2  front
9573  2008     ML 350   full
9574  2008     ES 350  front
9575  2007         Q7   full

[9215 rows x 10 columns]
```

[52]:
```
# creating a Profile report for understand data frame and gets insights.
profile_cleaned = pandas_profiling.ProfileReport(carsales_data_droping)
profile_cleaned
```

[53]:
```
# saving a copy of pandas profiling copy for understand it in a better way
profile_cleaned.to_file(output_file="CarSales_post_preprocessing.html")
```

4 . Questions

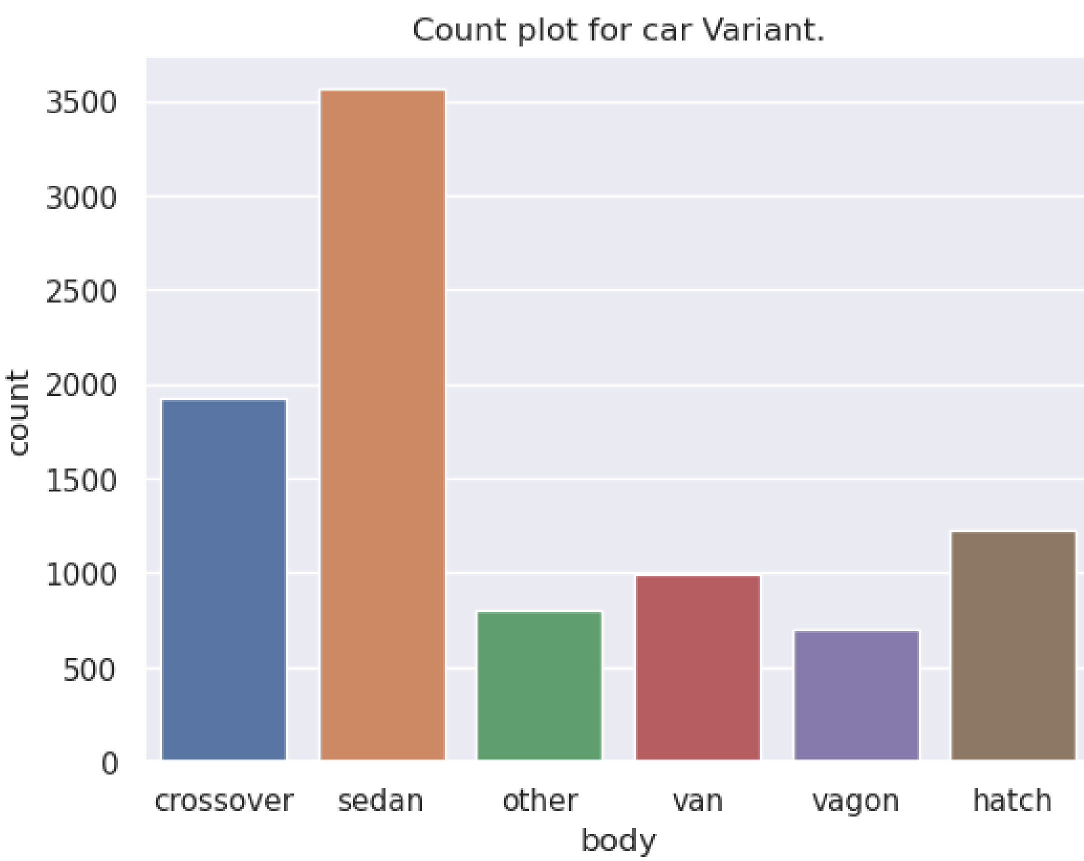
Try to do all the questions along with the neccessary graphs and mention insights too

4.1 Which type of cars are sold maximum?

[54]:
```
###Creating a copy of data for visualisation using matplotlib and seaborn
carsales_ans = carsales_data_droping
```

[55]:
```
# This will return body type of car which is selling maximum.
sns.countplot(x='body',data=carsales_ans).set_title('Count plot for car Variant.
↪')
```

[55]:
```
Text(0.5, 1.0, 'Count plot for car Variant.')
```

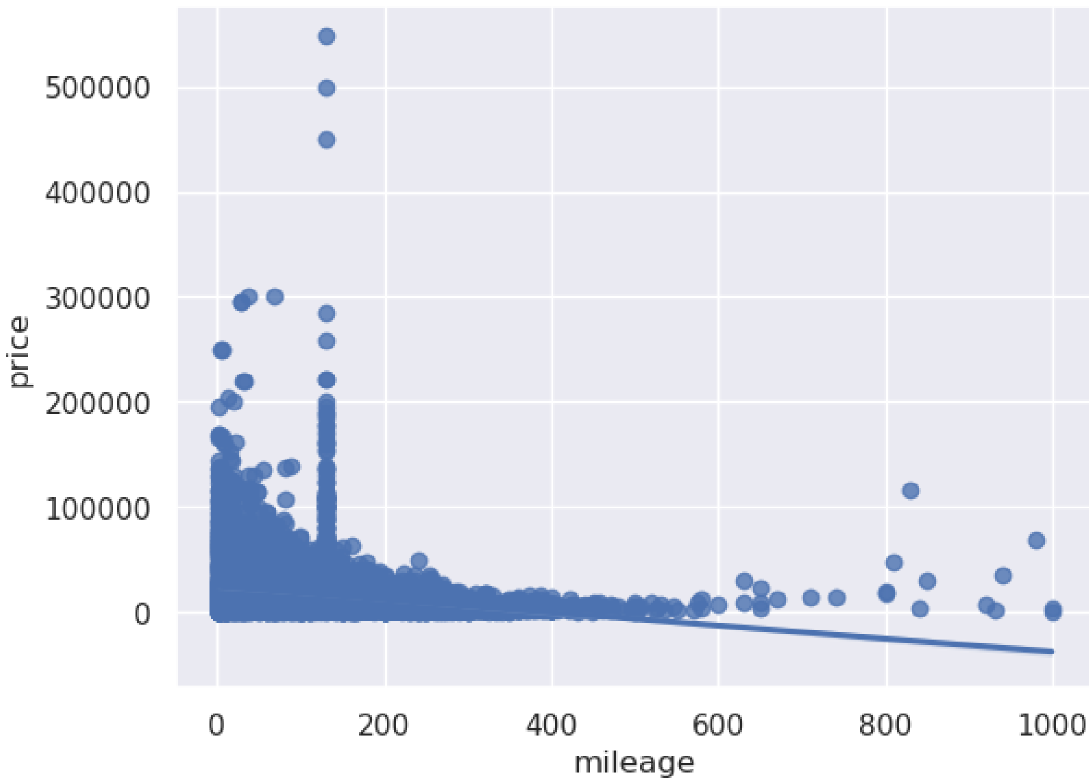


You can see sedan cars are sold maximum and followed that crossover,hatch,van,other and vagon

0.1.1 4.2 What is the co-relation between price and mileage?

[56]:
```
sns.regplot(x='mileage',y= 'price',data = carsales_ans)
```

[56]:
```
<AxesSubplot: xlabel='mileage', ylabel='price'>
```



You can see there are some outliers here. Excluding those,it seems that majority of car price is below 150000 and gives mileage in the range of 0 to 400.

0.1.2 4.3. How many cars are registered?

[57]:
```
carsales_ans['registration'].value_counts()
```

[57]:
```
yes    8661
no      554
Name: registration, dtype: int64
```

[58]:
```
sns.countplot('registration',data = carsales_ans).set_title('Car Registration␣
↪status')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
Cell In [58], line 1
----> 1 sns.countplot('registration',data = carsales_ans).set_title('Car␣
↪Registration status')

TypeError: countplot() got multiple values for argument 'data'
```

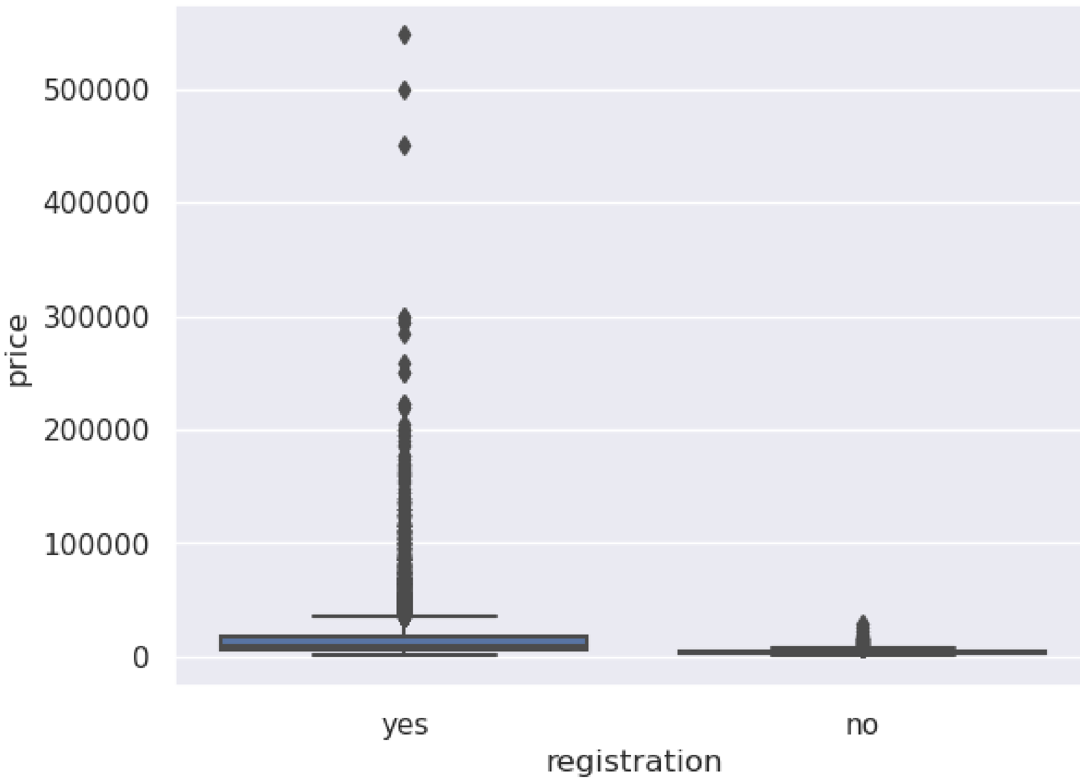
4.4 Price distribution between registered and non-registered cars .

[59]:
```
carsales_ans.groupby(['registration','body'])['price'].mean()
```

[59]:
```
registration  body
no            crossover     7951.310345
              hatch         2563.750000
              other         3936.687500
              sedan         3938.627049
              vagon         3124.428571
              van           3488.457143
yes           crossover    30597.282810
              hatch         8723.459297
              other        20329.349533
              sedan        12826.420380
              vagon        10279.830563
              van          10970.113397
Name: price, dtype: float64
```

[60]:
```
sns.boxplot(x='registration',y='price',data = carsales_ans)
```

[60]:
```
<AxesSubplot: xlabel='registration', ylabel='price'>
```



[61]:
```
sns.displot(x='registration',y='price',data = carsales_ans)
```

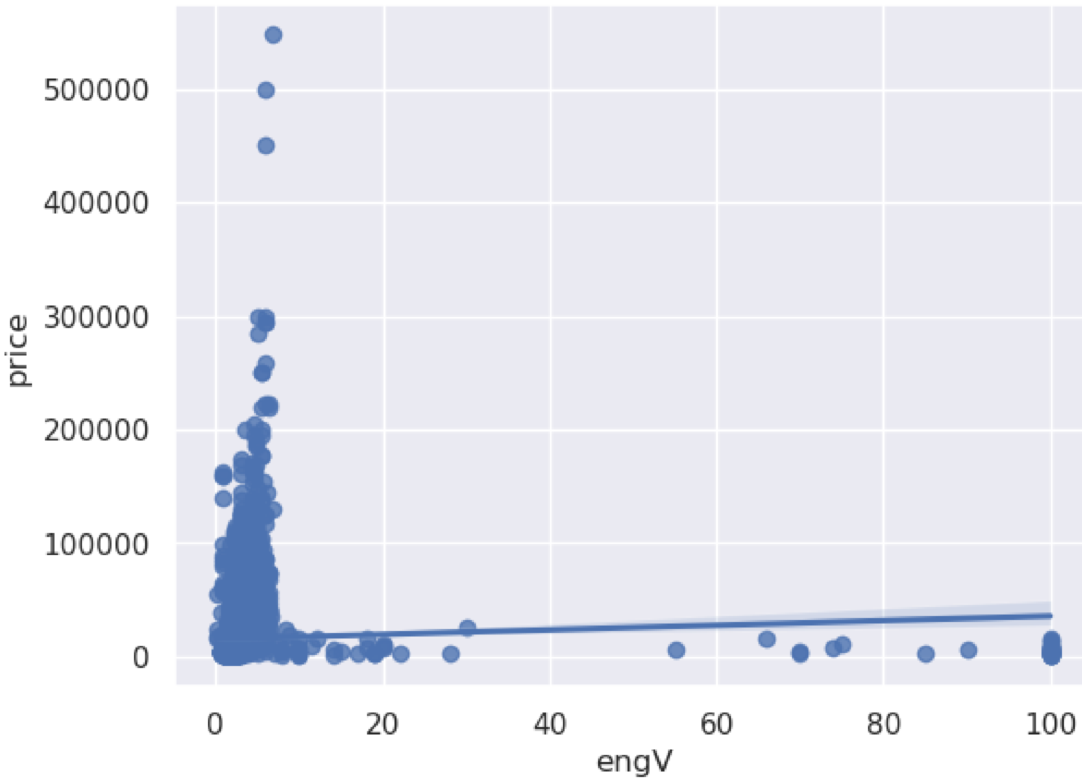
[61]: <seaborn.axisgrid.FacetGrid at 0x7f63c661b310>

Majority of the cars are registered and the price of those cars are below 300000. Non-registered cars are cheaper in cost.

4.5 What is the car price distribution based on Engine Value?

[63]:
```
sns.regplot(x='engV', y = 'price',data = carsales_ans)
```

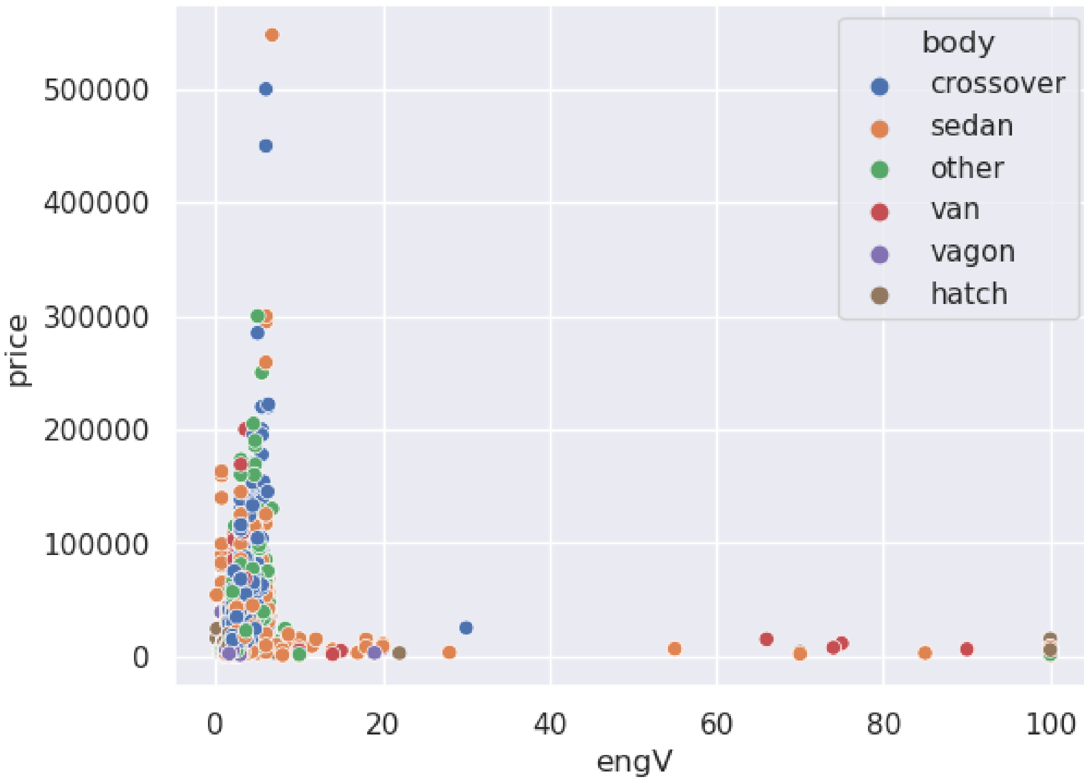
[63]: <AxesSubplot: xlabel='engV', ylabel='price'>



Except few outliers, it is clearly observed that the range of car price is between 0 to 150000 having the range of engine value between 0 to 6.

[64]:
```
sns.scatterplot(x='engV', y = 'price',data = carsales_ans ,hue ='body')
```

[64]:
```
<AxesSubplot: xlabel='engV', ylabel='price'>
```



4.6 Which engine type of cars users preferred maximum?

[66]:
```
carsales_ans.groupby('engType')['price'].count().sort_values(ascending=False)
```

[66]:
```
engType
Petrol    4259
Diesel    2821
Gas       1692
Other      443
Name: price, dtype: int64
```

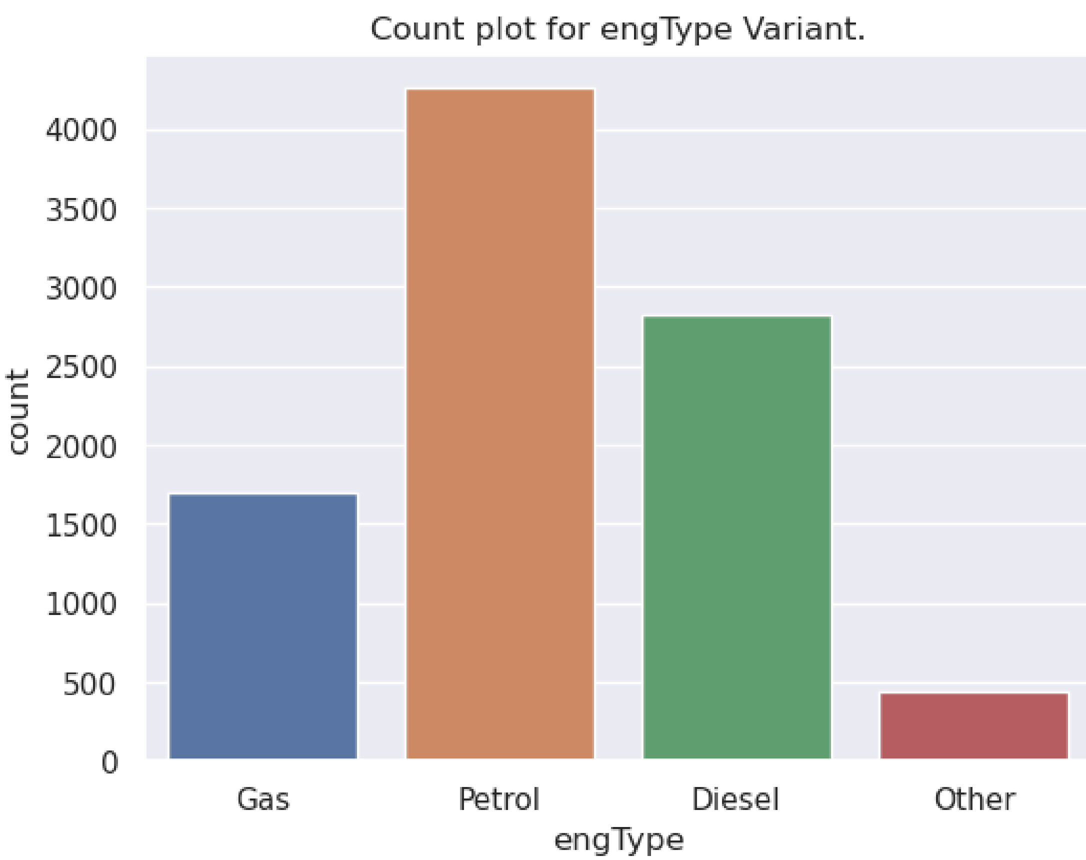
[71]:
```
carsales_ans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9215 entries, 0 to 9575
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   car           9215 non-null   object
 1   price         9215 non-null   float64
 2   body          9215 non-null   object
 3   mileage       9215 non-null   int64
```

```
 4   engV          9215 non-null   float64
 5   engType       9215 non-null   object
 6   registration  9215 non-null   object
 7   year          9215 non-null   int64
 8   model         9215 non-null   object
 9   drive         9215 non-null   object
dtypes: float64(2), int64(2), object(6)
memory usage: 1.0+ MB
```

[80]:
```python
# This will return body type of car which is selling maximum.
sns.countplot(x='engType',data=carsales_ans).set_title('Count plot for engType␣
↪Variant.')
```

[80]:
```
Text(0.5, 1.0, 'Count plot for engType Variant.')
```



Petrol cars are more preferred and followed by Diesel, Gas and others.

4.7 Establish coorelation between all the features using heatmap.

[76]:
```python
# creating a copy of data frame for performing sorrelation function
corr_car_sales_data = carsales_ans.corr()
```

```
corr_car_sales_data
```

```
/tmp/ipykernel_84/1481666568.py:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  corr_car_sales_data = carsales_ans.corr()
```

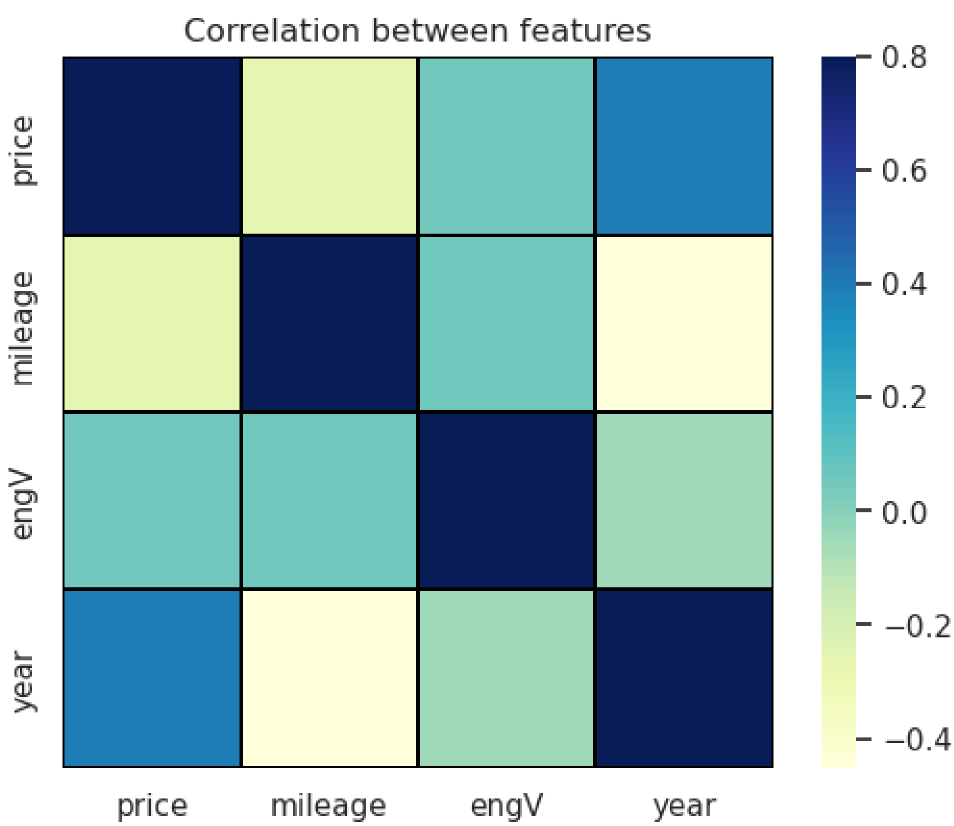
[76]:

	price	mileage	engV	year
price	1.000000	-0.256693	0.051242	0.391745
mileage	-0.256693	1.000000	0.055425	-0.451794
engV	0.051242	0.055425	1.000000	-0.047734
year	0.391745	-0.451794	-0.047734	1.000000

[77]:

```
sns.heatmap(corr_car_sales_data,vmax = 0.8 , linewidth = 0.01 ,square =␣
 ↪True,cmap = 'YlGnBu',linecolor = 'black')
plt.title('Correlation between features')
```

[77]: Text(0.5, 1.0, 'Correlation between features')



mileage and engV are negatively corelated with year.

mileage is also negatively corelated with year.

engV is positively coorelated with mileage and price.

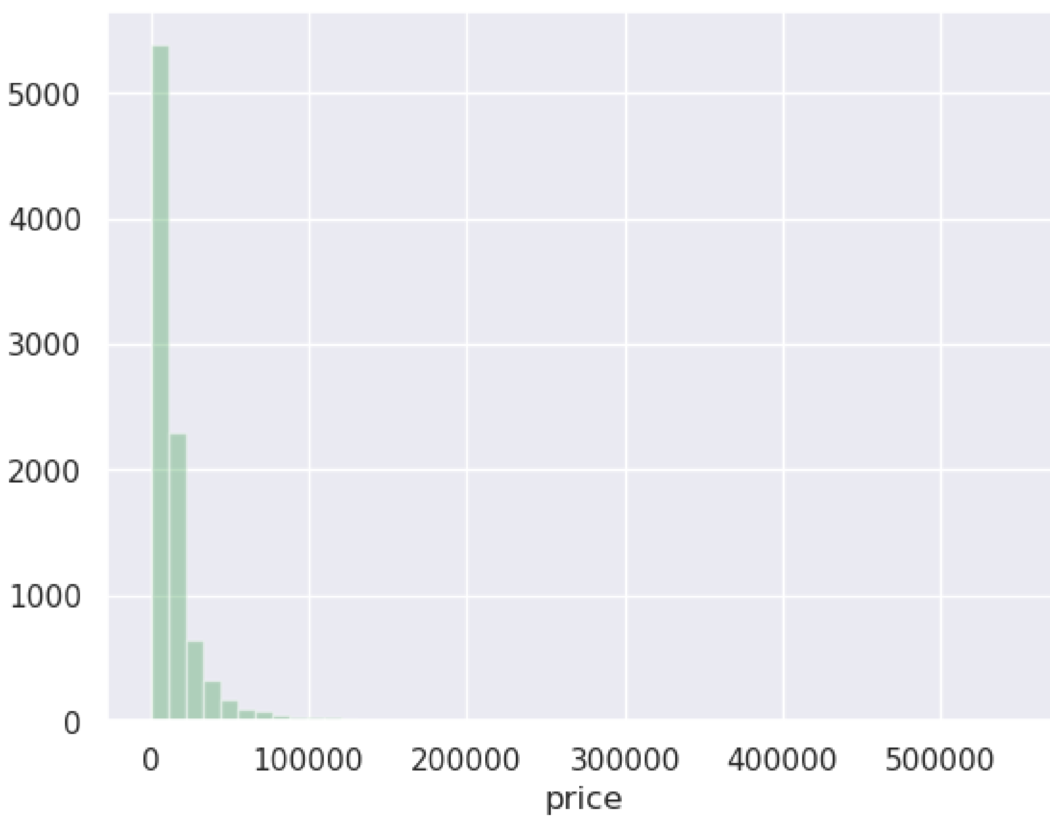
Positive corelation observed between year and price too.

4.8 Distribution of price.

[79]:
```
sns.distplot(carsales_ans['price'], color='g', kde=False)
```

```
/tmp/ipykernel_84/963584368.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(carsales_ans['price'], color='g', kde=False)
```

[79]: <AxesSubplot: xlabel='price'>

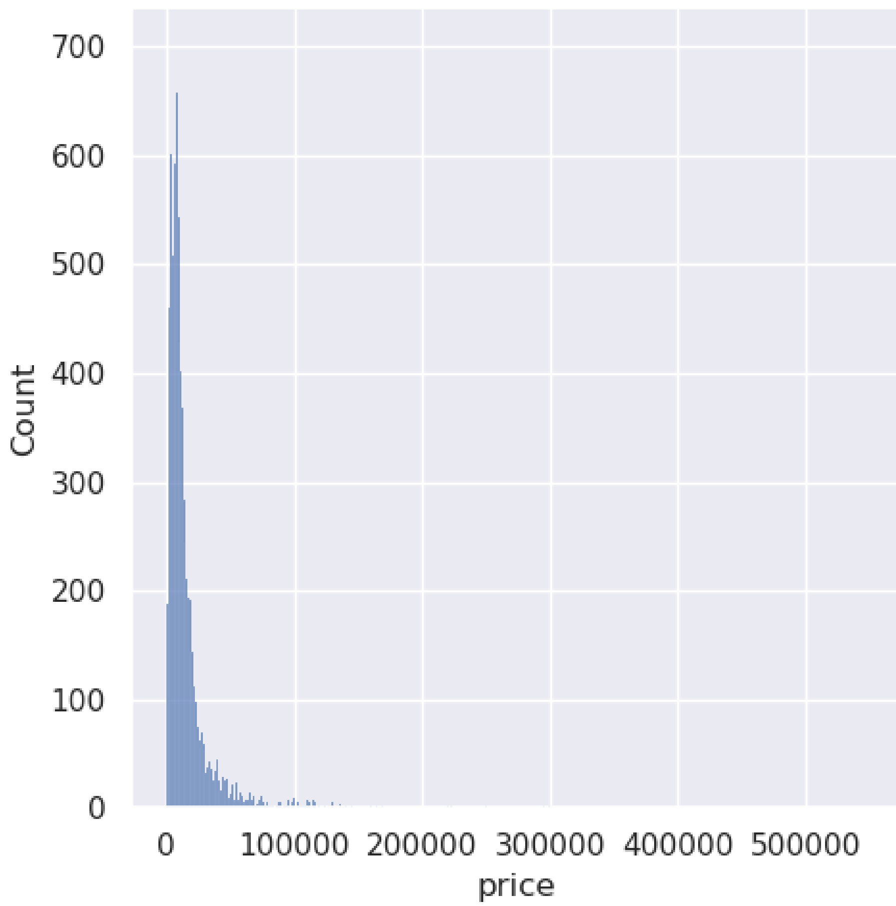


[78]:
```
sns.distplot(carsales_ans['price'],color = 'g')
plt.title("Price Distribution")
plt.show()
```

```
/tmp/ipykernel_84/2775759118.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(carsales_ans['price'],color = 'g')
```

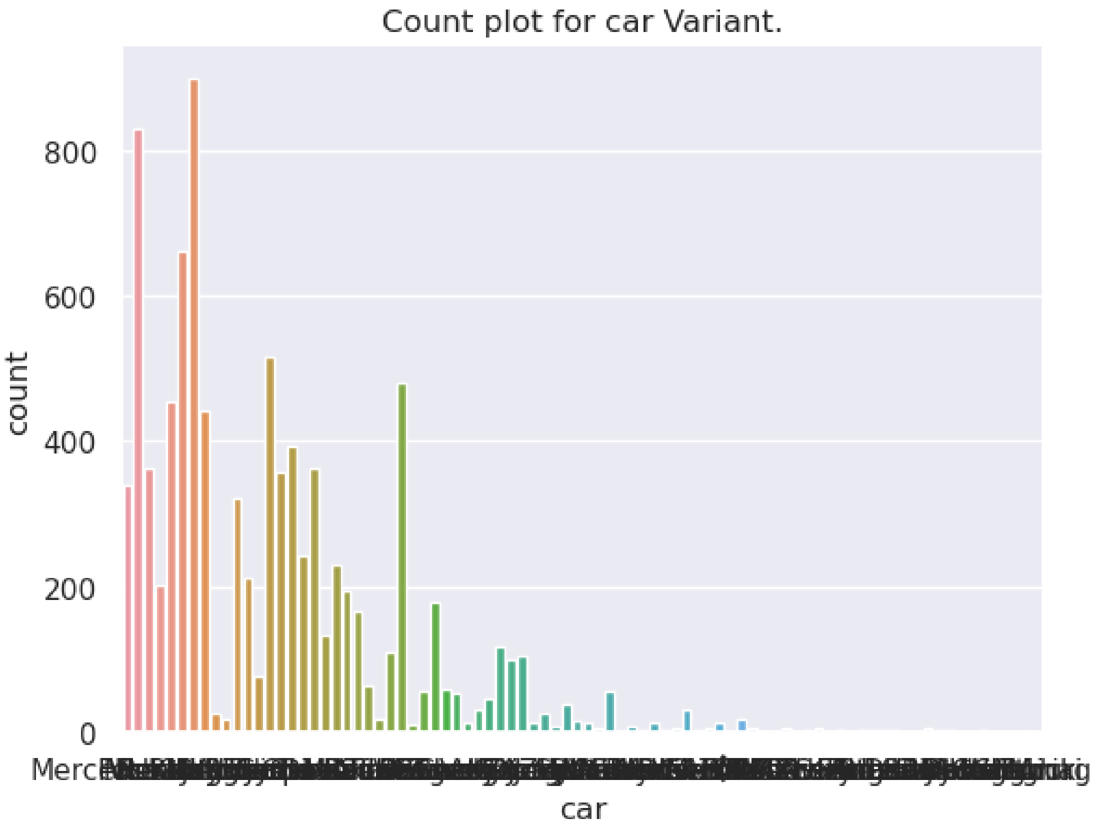
[76]:
```
sns.displot(data= carsales_ans['price'])
```

[76]: <seaborn.axisgrid.FacetGrid at 0x7fcf4c1b3a90>

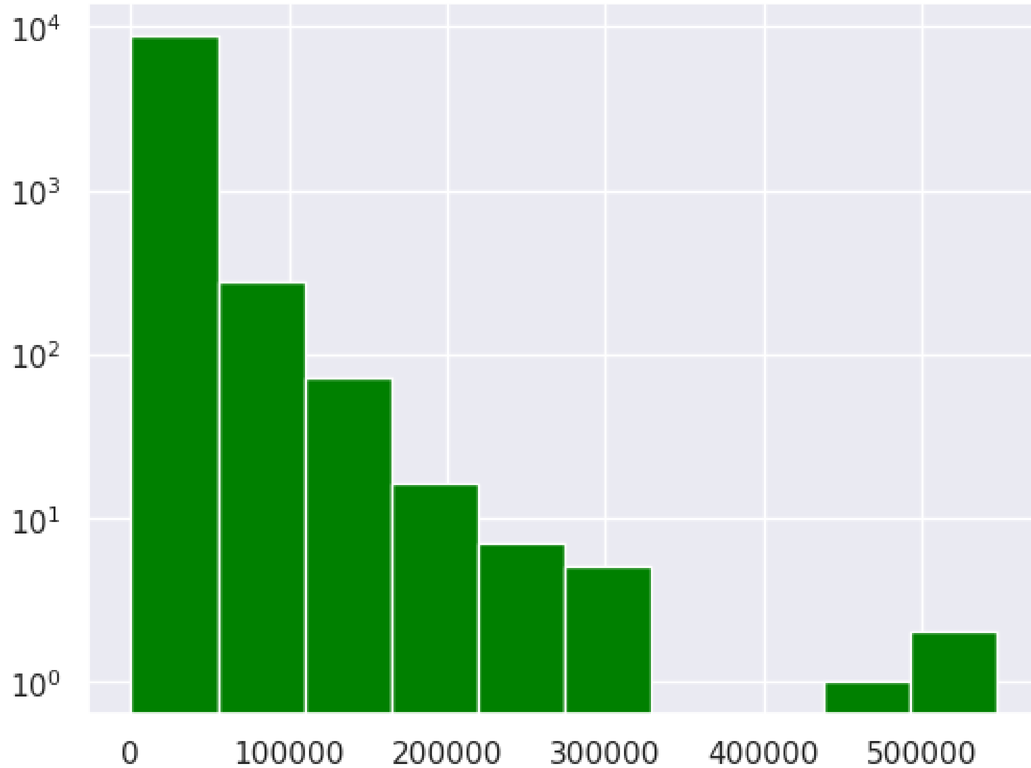



```
[84]: sns.countplot(x='car',data=carsales_ans).set_title('Count plot for car Variant.
      ↪')
```

```
[84]: Text(0.5, 1.0, 'Count plot for car Variant.')
```




```
[110]: plt.hist(carsales_ans["price"],log='car',color='green')
       plt.show()
```



4.4 Price distribution between registered and non_registered cars.

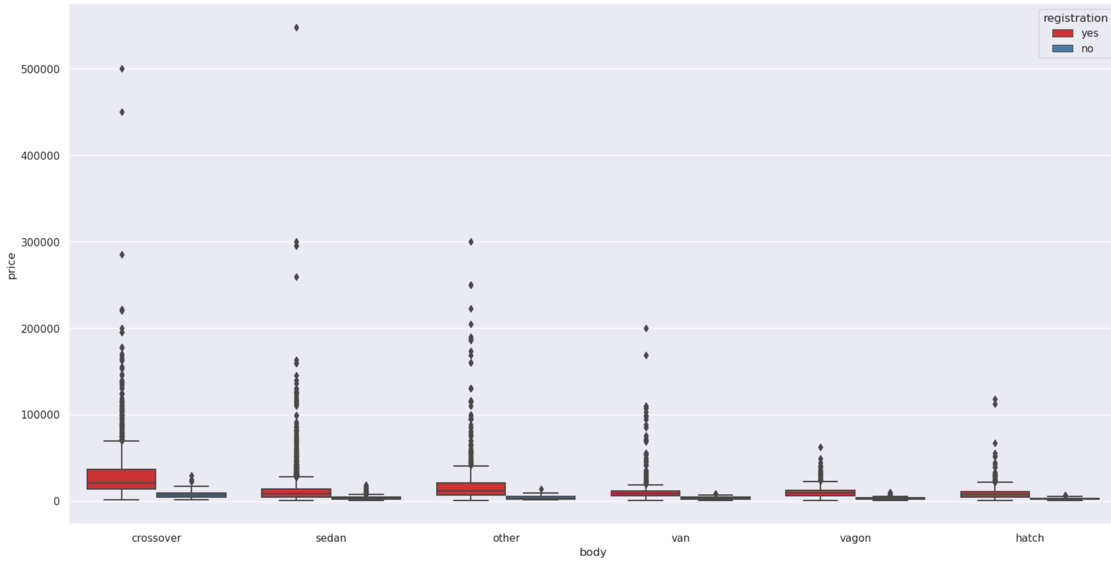
[ ]:
```
pd.pivot_table(
    carsales_ans,
    index = 'body',
    columns = 'registration',
    values = 'price',
    aggfunc = np.mean
)
```

[ ]:

registration	no	yes
body		
crossover	7951.310345	30597.282810
hatch	2563.750000	8723.459297
other	3936.687500	20329.349533
sedan	3938.627049	12826.420380
vagon	3124.428571	10279.830563
van	3488.457143	10970.113397

[67]:
```
plt.figure(figsize = (20,10))
sns.boxplot(data = carsales_ans, y='price', x = 'body',palette = 'Set1', hue =␣
↪'registration')
```

[67]: <AxesSubplot: xlabel='body', ylabel='price'>



Conclusion:

Sedan cars sold maximum.

Price is increasing as the engine value is increasing.

The price and mileage goes down as engine values decreasing.

Petrol cars are more preferred and followed by Diesel, Gas and others.

Majority of the cars are registered and the price of those cars are below 300000. Non-registered cars are cheaper in cost.

[ ]: