Car Price Predction

Develop a model that predicts the probability of a used car in addition to predicting the price Based on the factors of external as well as internal factors of the car

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

In [2]:
```
cars_data=pd.read_csv("Car details v3.csv")
```

In [3]:
```
cars_data
```

Out[3]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	n
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	

8128 rows × 13 columns

In [4]:
```
cars_data.nunique()
```

Out[4]:
```
name             2058
year               29
selling_price     677
km_driven         921
fuel                4
seller_type         3
transmission        2
owner               5
mileage           393
engine            121
max_power         322
torque            441
seats               9
dtype: int64
```

In [5]:
```
cars_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [6]:
```
cars_data.describe()
```

Out[6]:

	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

In [7]:
```
cars_data.isna().sum()
```

Out[7]:
```
name              0
year              0
selling_price     0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         221
engine          221
max_power       215
torque          222
seats           221
dtype: int64
```

In [8]:
```
cars_data.duplicated().sum()
```

Out[8]:
```
1202
```

In [9]:
```
cars_data.drop_duplicates(inplace=True)
```

In [10]:
```
cars_data
```

Out[10]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	n
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	
8121	Maruti Wagon R VXI BS IV with ABS	2013	260000	50000	Petrol	Individual	Manual	Second Owner	
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	

6926 rows × 13 columns

In [11]:
```
cars_data[['mileage(kmpl)', 'column2']] = cars_data['mileage'].str.split(' ', expan
```

In [12]:
```
cars_data.drop("column2",inplace=True,axis=1)
```

In [13]:
```
cars_data
```

Out[13]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	n
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	
8121	Maruti Wagon R VXI BS IV with ABS	2013	260000	50000	Petrol	Individual	Manual	Second Owner	
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	

6926 rows × 14 columns

In [14]:
```
cars_data[['engine(CC)', 'column2']] = cars_data['engine'].str.split(' ', expand=Tr
```

In [15]:
```
cars_data.drop("column2",inplace=True,axis=1)
```

In [16]:
```
cars_data
```

Out[16]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	m
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	
8121	Maruti Wagon R VXI BS IV with ABS	2013	260000	50000	Petrol	Individual	Manual	Second Owner	
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	

6926 rows × 15 columns

In [17]:
```
cars_data["max_power"].isna().sum()
```

Out[17]: 205

In [18]:
```
cars_data["max_power"].value_counts()
```

Out[18]:
```
max_power
74 bhp        324
88.5 bhp      177
46.3 bhp      158
67 bhp        152
67.1 bhp      141
              ...
265 bhp         1
55.23 bhp       1
156 bhp         1
181.04 bhp      1
135.1 bhp       1
Name: count, Length: 322, dtype: int64
```

In [19]:
```
cars_data["max_power"].fillna("74 bhp",inplace=True)
```

```
C:\Users\Lenovo_Ideapad_slim3\AppData\Local\Temp\ipykernel_10328\3144470996.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  cars_data["max_power"].fillna("74 bhp",inplace=True)
```

In [20]:
```
cars_data["max_power"].isna().sum()
```

Out[20]: 0

In [21]:
```
cars_data[['max_power(bhp)', 'column2']] = cars_data['max_power'].str.split(' ', ex
```

In [22]:
```
cars_data.drop("column2",inplace=True,axis=1)
```

In [23]:
```
cars_data
```

Out[23]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	m
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	
8121	Maruti Wagon R VXI BS IV with ABS	2013	260000	50000	Petrol	Individual	Manual	Second Owner	
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	

6926 rows × 16 columns

In [24]:
```
cars_data.drop(["mileage","engine","max_power"],axis=1,inplace=True)
```

In [25]:
```
data=cars_data.drop("selling_price",axis=1)
```

In [26]:
```
cars_data=pd.merge(data,cars_data)
```

In [27]:
```
cars_data
```

Out[27]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	se
0	Maruti Swift Dzire VDI	2014	145500	Diesel	Individual	Manual	First Owner	190Nm@ 2000rpm	
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	Diesel	Individual	Manual	Second Owner	250Nm@ 1500-2500rpm	
2	Honda City 2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	12.7@ 2,700(kgm@ rpm)	
3	Honda City 2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	12.7@ 2,700(kgm@ rpm)	
4	Hyundai i20 Sportz Diesel	2010	127000	Diesel	Individual	Manual	First Owner	22.4 kgm at 1750-2750rpm	
...	...	...	...	...	...	...	...	...	
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	Petrol	Individual	Manual	Second Owner	90Nm@ 3500rpm	
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	Diesel	Individual	Manual	Second Owner	219.7Nm@ 1500-2750rpm	
7409	Hyundai i20 Magna	2013	110000	Petrol	Individual	Manual	First Owner	113.7Nm@ 4000rpm	
7410	Hyundai Verna CRDi SX	2007	119000	Diesel	Individual	Manual	Fourth & Above Owner	24@ 1,900-2,750(kgm@ rpm)	
7411	Maruti Swift Dzire ZDi	2009	120000	Diesel	Individual	Manual	First Owner	190Nm@ 2000rpm	

7412 rows × 13 columns

In [28]:
```
cars_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7412 entries, 0 to 7411
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   name            7412 non-null   object 
 1   year            7412 non-null   int64  
 2   km_driven       7412 non-null   int64  
 3   fuel            7412 non-null   object 
 4   seller_type     7412 non-null   object 
 5   transmission    7412 non-null   object 
 6   owner           7412 non-null   object 
 7   torque          7195 non-null   object 
 8   seats           7196 non-null   float64
 9   mileage(kmpl)   7196 non-null   object 
 10  engine(CC)      7196 non-null   object 
 11  max_power(bhp)  7412 non-null   object 
 12  selling_price   7412 non-null   int64  
dtypes: float64(1), int64(3), object(9)
memory usage: 752.9+ KB
```

In [29]:
```
cars_data["mileage(kmpl)"]=cars_data["mileage(kmpl)"].astype(float)
```

In [30]:
```
cars_data["engine(CC)"]=cars_data["engine(CC)"].astype(float)
```

In [31]:
```
cars_data["max_power(bhp)"].isna().sum()
```

Out[31]:
```
0
```

In [32]:
```
cars_data['max_power(bhp)'] = pd.to_numeric(cars_data['max_power(bhp)'])
```

In [33]:
```
cars_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7412 entries, 0 to 7411
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   name            7412 non-null   object 
 1   year            7412 non-null   int64  
 2   km_driven       7412 non-null   int64  
 3   fuel            7412 non-null   object 
 4   seller_type     7412 non-null   object 
 5   transmission    7412 non-null   object 
 6   owner           7412 non-null   object 
 7   torque          7195 non-null   object 
 8   seats           7196 non-null   float64
 9   mileage(kmpl)   7196 non-null   float64
 10  engine(CC)      7196 non-null   float64
 11  max_power(bhp)  7411 non-null   float64
 12  selling_price   7412 non-null   int64  
dtypes: float64(4), int64(3), object(6)
memory usage: 752.9+ KB
```

In [34]:
```
cars_data.isna().sum()
```

Out[34]:
```
name                0
year                0
km_driven           0
fuel                0
seller_type         0
transmission        0
owner               0
torque            217
seats             216
mileage(kmpl)     216
engine(CC)        216
max_power(bhp)      1
selling_price       0
dtype: int64
```

In [35]:
```
cars_data.loc[cars_data["max_power(bhp)"].isna()]
```

Out[35]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mil
4581	Maruti Omni CNG	2000	100000	CNG	Individual	Manual	Second Owner	NaN	8.0	

In [36]:
```
cars_data["max_power(bhp)"].fillna(cars_data["max_power(bhp)"].median(),inplace=Tru
```

```
C:\Users\Lenovo_Ideapad_slim3\AppData\Local\Temp\ipykernel_10328\3549038377.py:1: Fu
tureWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cars_data["max_power(bhp)"].fillna(cars_data["max_power(bhp)"].median(),inplace=Tr
ue)
```

In [37]:
```
cars_data.loc[cars_data["max_power(bhp)"].isna()]
```

Out[37]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [38]:
```
cars_data["engine(CC)"].fillna(cars_data["engine(CC)"].median(),inplace=True)
```

```
C:\Users\Lenovo_Ideapad_slim3\AppData\Local\Temp\ipykernel_10328\3135933936.py:1: Fu
tureWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cars_data["engine(CC)"].fillna(cars_data["engine(CC)"].median(),inplace=True)
```

In [39]:
```
cars_data["mileage(kmpl)"].fillna(cars_data["mileage(kmpl)"].median(),inplace=True)
```

```
C:\Users\Lenovo_Ideapad_slim3\AppData\Local\Temp\ipykernel_10328\2365437972.py:1: Fu
tureWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cars_data["mileage(kmpl)"].fillna(cars_data["mileage(kmpl)"].median(),inplace=Tru
e)
```

In [40]:
```
cars_data["seats"].value_counts()
```

Out[40]:
```
seats
5.0     5658
7.0     1020
8.0      232
4.0      130
9.0       76
6.0       59
10.0      18
2.0        2
14.0       1
Name: count, dtype: int64
```

In [41]:
```
cars_data["seats"].fillna(5.0,inplace=True)
```

```
C:\Users\Lenovo_Ideapad_slim3\AppData\Local\Temp\ipykernel_10328\1096849628.py:1: Fu
tureWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cars_data["seats"].fillna(5.0,inplace=True)
```

In [42]:
```
cars_data["torque"].value_counts()
```

Out[42]:
```
torque
190Nm@ 2000rpm                 556
200Nm@ 1750rpm                 426
90Nm@ 3500rpm                  371
62Nm@ 3000rpm                  180
114Nm@ 4000rpm                 160
                              ...
11.4@ 4,000(kgm@ rpm)            1
25.5@ 1,500-3,000(kgm@ rpm)      1
202Nm@ 3600-5200rpm              1
128Nm@ 3100rpm                   1
96  Nm at 3000  rpm              1
Name: count, Length: 441, dtype: int64
```

In [43]:
```
cars_data["torque"].fillna("190Nm@ 2000rpm ",inplace=True)
```

```
C:\Users\Lenovo_Ideapad_slim3\AppData\Local\Temp\ipykernel_10328\481521205.py:1: Fut
ureWarning: A value is trying to be set on a copy of a DataFrame or Series through c
hained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cars_data["torque"].fillna("190Nm@ 2000rpm ",inplace=True)
```

In [44]:
```
cars_data.isna().sum()
```

Out[44]:
```
name              0
year              0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
torque            0
seats             0
mileage(kmpl)     0
engine(CC)        0
max_power(bhp)    0
selling_price     0
dtype: int64
```

In [92]:
```
num_data=cars_data.select_dtypes(exclude="object")
```

In [94]:
```
cars_data
```

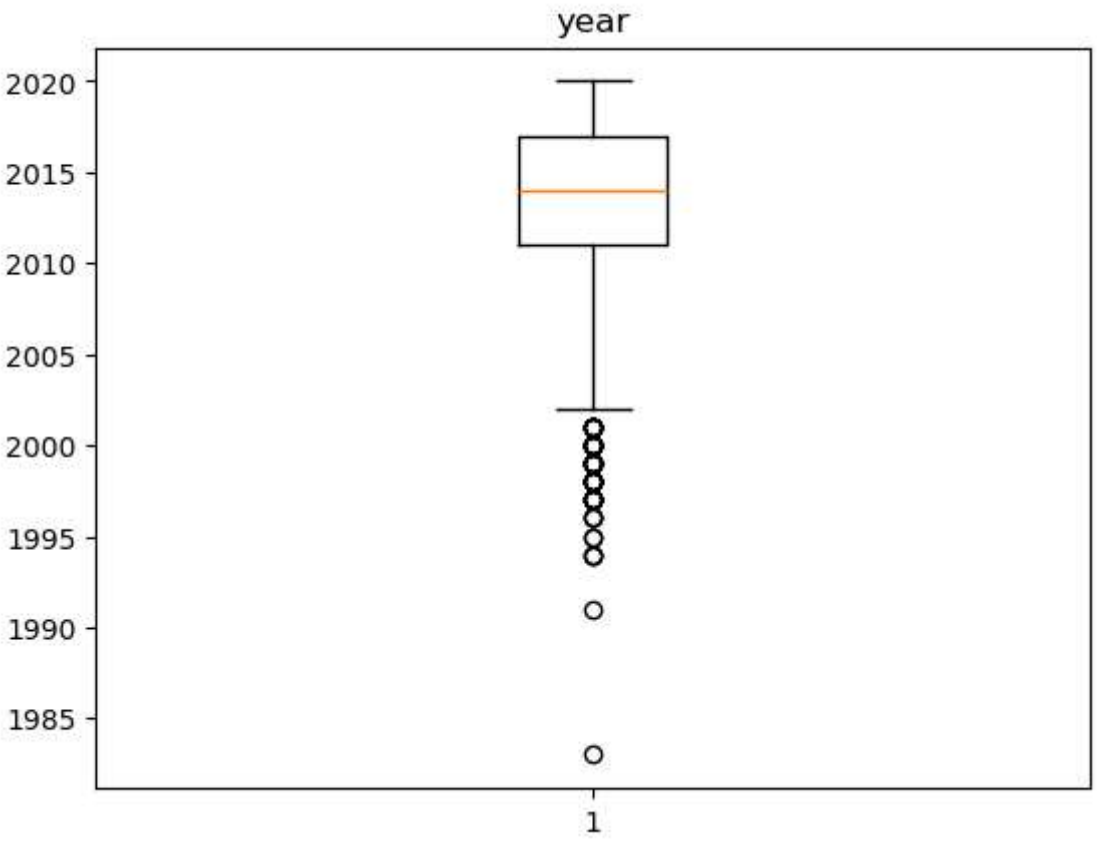
Out[94]:

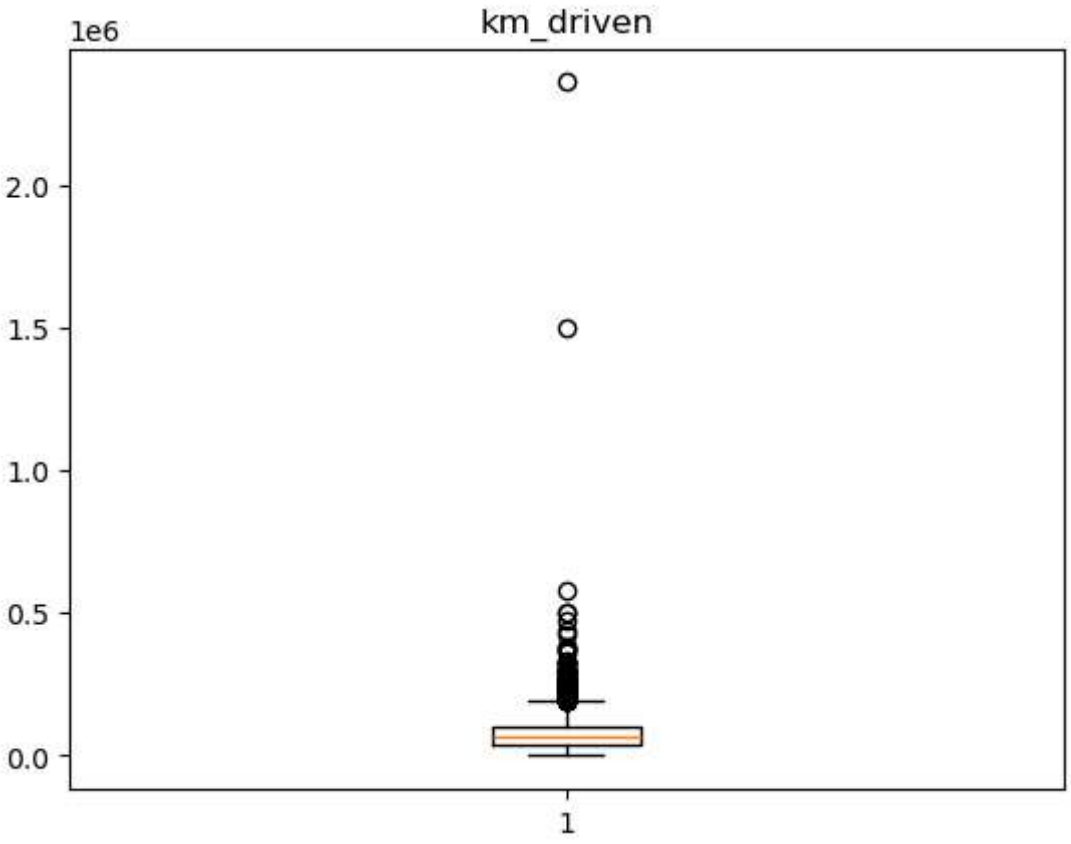
	name	year	km_driven	fuel	seller_type	transmission	owner	torque	se
0	Maruti Swift Dzire VDI	2014	145500	Diesel	Individual	Manual	First Owner	190Nm@ 2000rpm	
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	Diesel	Individual	Manual	Second Owner	250Nm@ 1500-2500rpm	
2	Honda City 2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	12.7@ 2,700(kgm@ rpm)	
3	Honda City 2017-2020 EXi	2006	140000	Petrol	Individual	Manual	Third Owner	12.7@ 2,700(kgm@ rpm)	
4	Hyundai i20 Sportz Diesel	2010	127000	Diesel	Individual	Manual	First Owner	22.4 kgm at 1750-2750rpm	
...	...	...	...	...	...	...	...	...	
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	Petrol	Individual	Manual	Second Owner	90Nm@ 3500rpm	
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	Diesel	Individual	Manual	Second Owner	219.7Nm@ 1500-2750rpm	
7409	Hyundai i20 Magna	2013	110000	Petrol	Individual	Manual	First Owner	113.7Nm@ 4000rpm	
7410	Hyundai Verna CRDi SX	2007	119000	Diesel	Individual	Manual	Fourth & Above Owner	24@ 1,900-2,750(kgm@ rpm)	
7411	Maruti Swift Dzire ZDi	2009	120000	Diesel	Individual	Manual	First Owner	190Nm@ 2000rpm	

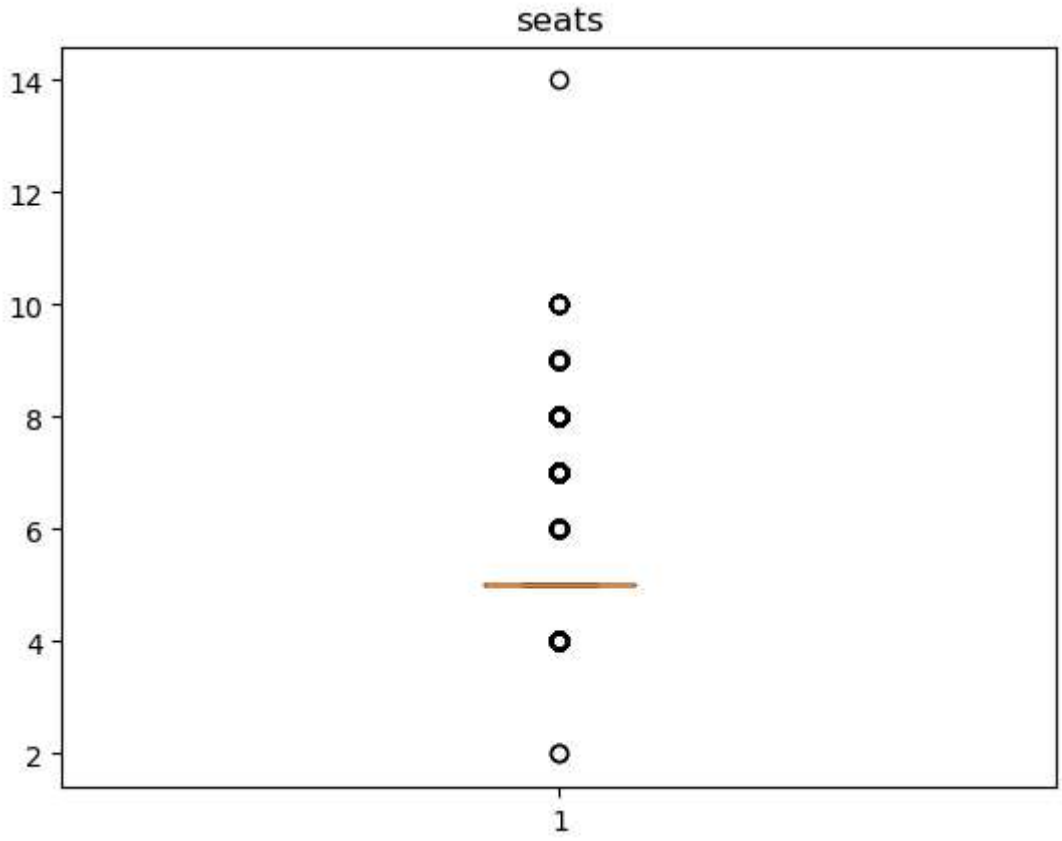
7412 rows × 13 columns

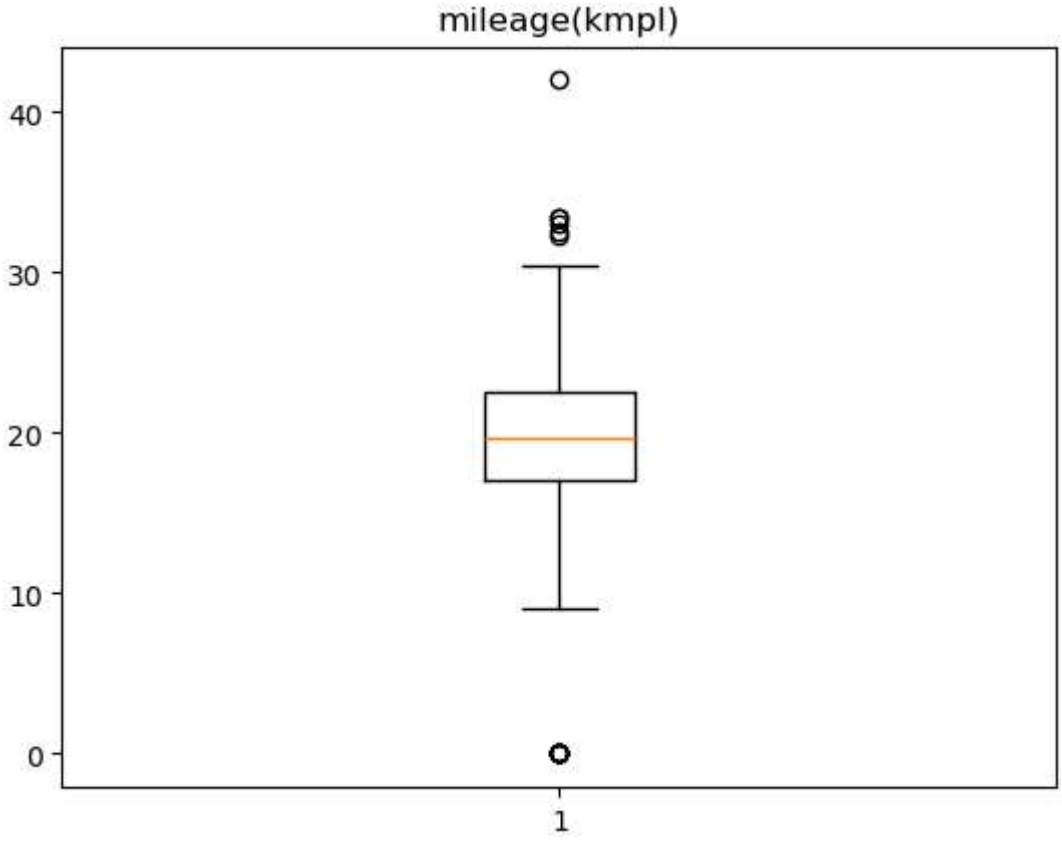
In [47]:
```
for i in num_data:
    plt.boxplot(num_data[i])
```

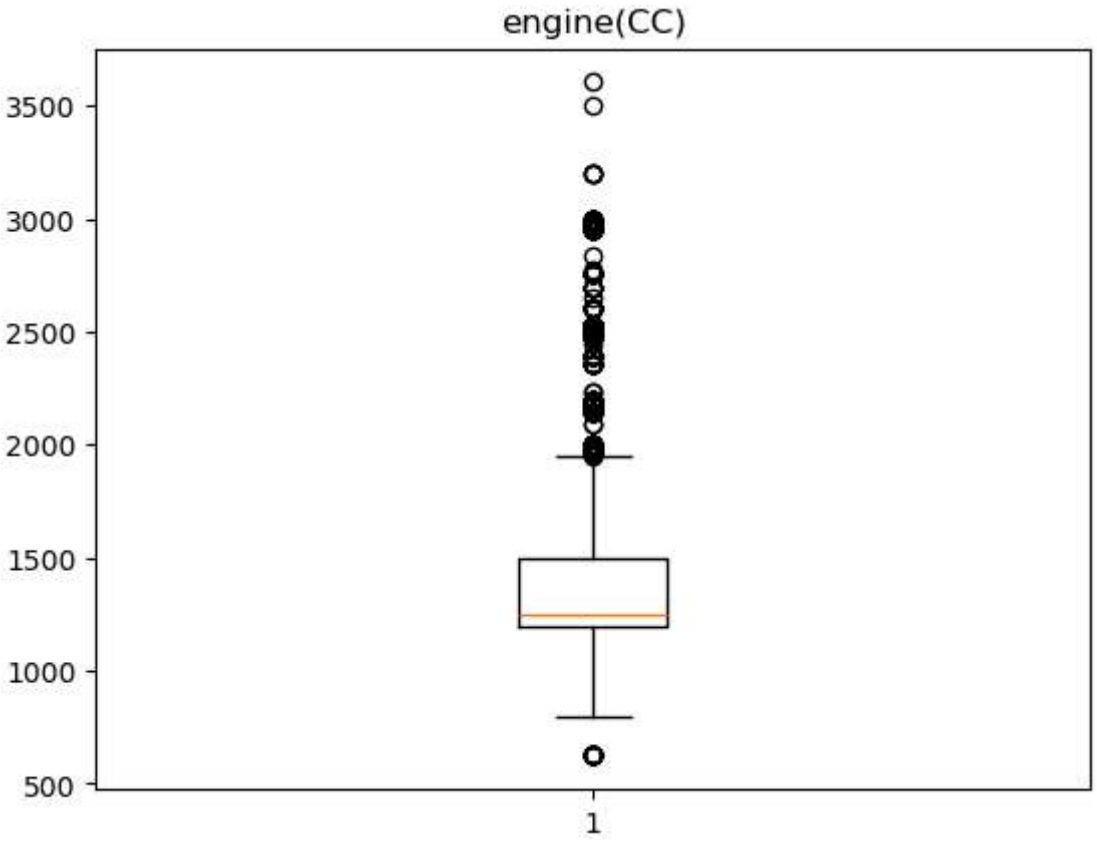
```
    plt.title(i)
    plt.show()
```

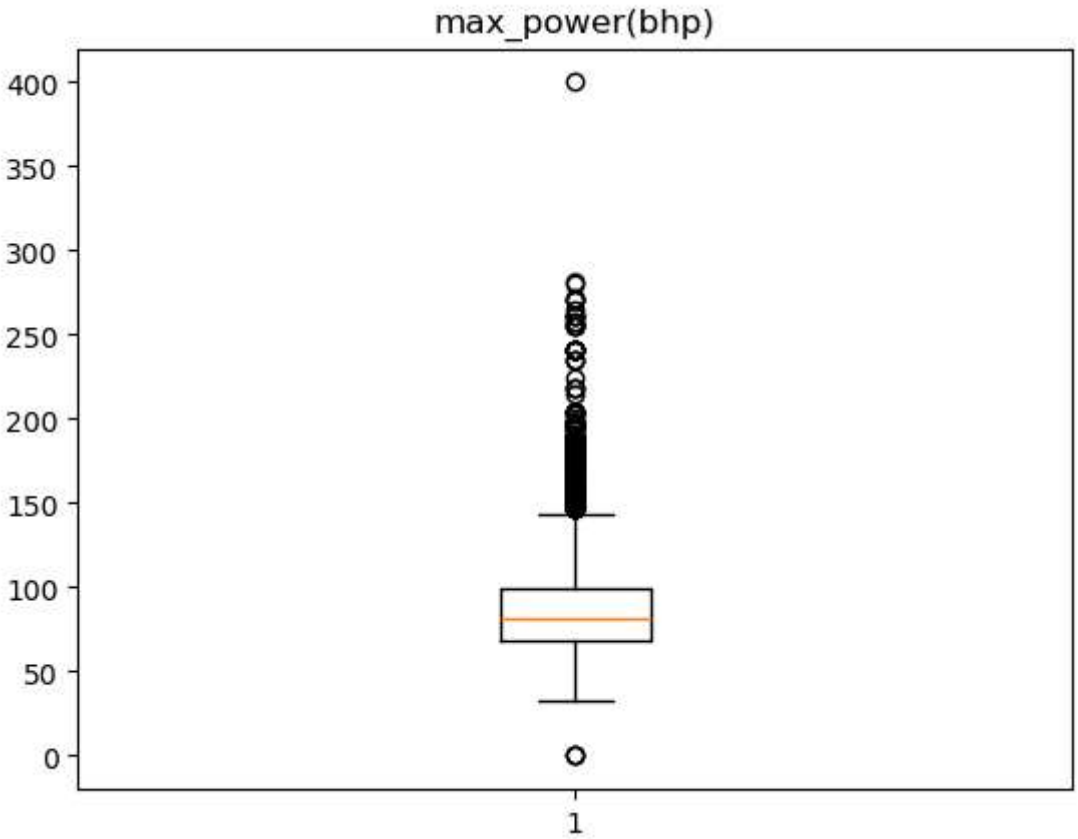


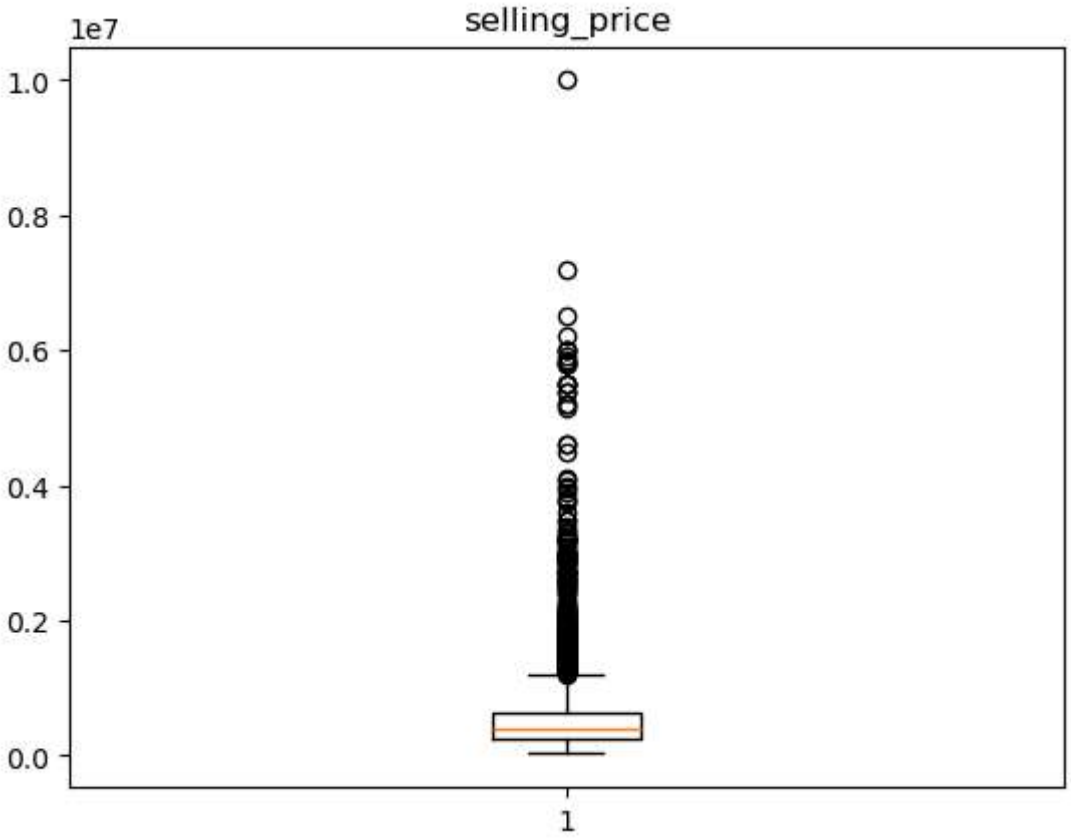



seats
14
12
10
8
6
4
2
1


mileage(kmpl)
40
30
20
10
0
1


engine(CC)
3500
3000
2500
2000
1500
1000
500
1


max_power(bhp)
400
350
300
250
200
150
100
50
0
1



In [48]:
```
outliers=["year","km_driven","seats","mileage(kmpl)","engine(CC)","max_power(bhp)"]
```

In [49]:
```
for i in cars_data:
    print(cars_data[i].value_counts())
    print(50*"*")
```

```
name
Maruti Swift Dzire VDI                          168
Maruti Alto 800 LXI                              92
Maruti Alto LXi                                  77
Maruti Swift VDI BSIV                            68
Maruti Swift VDI                                 66
                                                ...
Skoda Rapid 1.6 MPI AT Ambition BSIV              1
Ford Fiesta 1.6 ZXi ABS                           1
Honda Jazz 1.5 E i DTEC                           1
Mahindra Bolero 2011-2019 DI BSIII                1
Toyota Innova 2.5 GX (Diesel) 8 Seater BS IV      1
Name: count, Length: 2058, dtype: int64
**************************************************
year
2017    882
2016    741
2015    735
2012    660
2018    650
2014    645
2013    594
2011    584
2010    400
2019    365
2009    254
2008    211
2007    184
2006    133
2005     95
2020     65
2004     62
2003     49
2002     26
2000     20
1999     17
1997     11
1998     10
2001      9
1996      3
1994      3
1995      2
1983      1
1991      1
Name: count, dtype: int64
**************************************************
km_driven
120000    565
70000     502
80000     457
60000     436
50000     383
         ...
87237       1
62960       1
26634       1
```

```
163720      1
191000      1
Name: count, Length: 921, dtype: int64
**************************************************
fuel
Diesel    4025
Petrol    3291
CNG         58
LPG         38
Name: count, dtype: int64
**************************************************
seller_type
Individual          6698
Dealer               687
Trustmark Dealer      27
Name: count, dtype: int64
**************************************************
transmission
Manual       6814
Automatic     598
Name: count, dtype: int64
**************************************************
owner
First Owner             4562
Second Owner            2126
Third Owner              548
Fourth & Above Owner     171
Test Drive Car             5
Name: count, dtype: int64
**************************************************
torque
190Nm@ 2000rpm                556
200Nm@ 1750rpm                426
90Nm@ 3500rpm                 371
190Nm@ 2000rpm                217
62Nm@ 3000rpm                 180
                             ...
11.4@ 4,000(kgm@ rpm)           1
25.5@ 1,500-3,000(kgm@ rpm)     1
202Nm@ 3600-5200rpm             1
128Nm@ 3100rpm                  1
96  Nm at 3000  rpm             1
Name: count, Length: 442, dtype: int64
**************************************************
seats
5.0     5874
7.0     1020
8.0      232
4.0      130
9.0       76
6.0       59
10.0      18
2.0        2
14.0       1
Name: count, dtype: int64
**************************************************
```

```
mileage(kmpl)
19.70    404
18.90    230
18.60    162
21.10    161
17.00    130
        ...
15.63      1
11.88      1
12.12      1
14.66      1
16.51      1
Name: count, Length: 381, dtype: int64
**************************************************
engine(CC)
1248.0    1235
1197.0     746
796.0      471
998.0      418
1498.0     358
          ...
1489.0       1
1422.0       1
2496.0       1
3604.0       1
1950.0       1
Name: count, Length: 121, dtype: int64
**************************************************
max_power(bhp)
74.00     601
88.50     207
46.30     178
67.00     172
47.30     155
         ...
86.79       1
80.80       1
138.08      1
179.50      1
135.10      1
Name: count, Length: 319, dtype: int64
**************************************************
selling_price
300000    226
350000    212
600000    181
400000    177
550000    175
         ...
801000      1
317000      1
493000      1
236000      1
386000      1
Name: count, Length: 677, dtype: int64
**************************************************
```

In [50]:
```
#handling outliers
Q1=num_data["year"].quantile(0.25)
Q3=num_data["year"].quantile(0.75)
IQR=Q3-Q1
UB=Q3+(1.5*IQR)
LB=Q1-(1.5*IQR)
print(UB)
print(LB)
```

```
2026.0
2002.0
```

In [51]:
```
cars_data.loc[cars_data["year"]>UB]
```

Out[51]:

name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [52]:
```
cars_data.loc[cars_data["year"]<LB]
```

Out[52]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seat
8	Maruti 800 DX BSII	2001	5000	Petrol	Individual	Manual	Second Owner	59Nm@ 2500rpm	4.(
192	Maruti 800 Std	1999	40000	Petrol	Individual	Manual	Second Owner	59Nm@ 2500rpm	4.(
202	Daewoo Matiz SD	2000	60000	Petrol	Individual	Manual	First Owner	71Nm@ 3500rpm	5.(
341	Mahindra Willys CJ 3B 4X4	1983	10000	Diesel	Individual	Manual	Third Owner	190Nm@ 2000rpm	5.(
785	Hyundai Santro GLS I - Euro I	1999	110000	Petrol	Individual	Manual	Second Owner	190Nm@ 2000rpm	5.(
...	...	...	...	...	...	...	...	...	.
7259	Maruti Alto LX	2000	90000	Petrol	Individual	Manual	Second Owner	62Nm@ 3000rpm	5.(
7277	Hyundai Santro LS zipPlus	2000	50000	Petrol	Individual	Manual	Second Owner	190Nm@ 2000rpm	5.(
7308	Maruti 800 AC	1998	40000	Petrol	Individual	Manual	Second Owner	59Nm@ 2500rpm	4.(
7325	Maruti Gypsy King Soft Top	1997	186388	Petrol	Individual	Manual	Second Owner	103Nm@ 4500rpm	8.(
7402	Maruti 800 AC	1997	120000	Petrol	Individual	Manual	First Owner	59Nm@ 2500rpm	4.(

77 rows × 13 columns

In [53]:
```
cars_data.loc[cars_data["year"]<LB,"year"]=LB
```

In [54]:
```
cars_data.loc[cars_data["year"]<LB]
```

Out[54]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [55]:
```
#handling outliers
Q1=num_data["km_driven"].quantile(0.25)
Q3=num_data["km_driven"].quantile(0.75)
IQR=Q3-Q1
```

```
UB=Q3+(1.5*IQR)
LB=Q1-(1.5*IQR)
print(UB)
print(LB)
```

```
190000.0
-50000.0
```

In [56]:
```
cars_data.loc[cars_data["km_driven"]<LB]
```

Out[56]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [57]:
```
cars_data.loc[cars_data["km_driven"]>UB]
```

Out[57]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	s
297	Maruti Swift Dzire ZDi	2012	193000	Diesel	Individual	Manual	First Owner	190Nm@ 2000rpm	
312	Mahindra Bolero DI DX 7 Seater	2007	207890	Diesel	Individual	Manual	Second Owner	180 Nm at 1440-1500rpm	
418	Toyota Innova 2.5 G1 Diesel 8-seater	2005	240000	Diesel	Individual	Manual	Third Owner	20.4@ 1400-3400(kgm@ rpm)	
420	Hyundai Verna XXi ABS (Petrol)	2009	214000	Petrol	Individual	Manual	Second Owner	14.9@ 3,000(kgm@ rpm)	
421	Hyundai Verna CRDi SX	2009	214000	Diesel	Individual	Manual	Second Owner	24@ 1,900-2,750(kgm@ rpm)	
...	...	...	...	...	...	...	...	...	
7316	Hyundai Verna 1.6 SX	2012	200000	Diesel	Individual	Manual	Second Owner	259.8Nm@ 1900-2750rpm	
7356	Toyota Innova 2.5 V Diesel 7-seater	2010	200000	Diesel	Individual	Manual	Second Owner	20.4@ 1400-3400(kgm@ rpm)	
7362	Toyota Innova 2.5 E 7 STR	2009	250000	Diesel	Individual	Manual	First Owner	20.4@ 1400-3400(kgm@ rpm)	
7380	Ford Figo Diesel Titanium	2012	194000	Diesel	Individual	Manual	First Owner	160Nm@ 2000rpm	
7406	Hyundai Santro Xing GLS	2008	191000	Petrol	Individual	Manual	First Owner	96.1Nm@ 3000rpm	

171 rows × 13 columns

In [58]:
```
cars_data.loc[cars_data["km_driven"]>UB,"km_driven"]=UB
```

In [59]:
```
cars_data.loc[cars_data["km_driven"]>UB]
```

Out[59]:

name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [67]:
```
#handling outliers
Q1=num_data["mileage(kmpl)"].quantile(0.25)
Q3=num_data["mileage(kmpl)"].quantile(0.75)
IQR=Q3-Q1
UB=Q3+(1.5*IQR)
LB=Q1-(1.5*IQR)
print(UB)
print(LB)
```

```
30.849999999999998
8.690000000000001
```

In [68]:
```
cars_data.loc[cars_data["mileage(kmpl)"]>UB]
```

Out[68]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seat
39	Maruti Alto 800 CNG LXI Optional	2019	10000	CNG	Individual	Manual	Second Owner	60Nm@ 3500rpm	4.
183	Volvo XC90 T8 Excellence BSIV	2017	30000	Petrol	Individual	Automatic	First Owner	640Nm@ 1740rpm	4.
2277	Maruti Wagon R CNG LXI	2013	80000	CNG	Individual	Manual	Second Owner	78Nm@ 3500rpm	5.
3765	Maruti Alto 800 CNG LXI	2017	67000	CNG	Individual	Manual	First Owner	60Nm@ 3500rpm	4.
5379	Maruti Alto K10 LXI CNG	2019	20000	CNG	Individual	Manual	First Owner	78Nm@ 3500rpm	4.
5408	Maruti Alto 800 LXI CNG	2020	16000	CNG	Individual	Manual	First Owner	69Nm@ 3500rpm	5.

In [69]:
```
cars_data.loc[cars_data["mileage(kmpl)"]>UB,"mileage(kmpl)"]=UB
```

In [70]:
```
cars_data.loc[cars_data["mileage(kmpl)"]>UB]
```

Out[70]:

name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [71]:
```
cars_data.loc[cars_data["mileage(kmpl)"]<LB]
```

Out[71]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque
669	Tata Indica Vista Aura Safire Anniversary Edition	2009	28900	Petrol	Individual	Manual	Second Owner	9.8@ 3,000(kgm@ rpm)
797	Hyundai Santro Xing GL	2009	90000	Petrol	Individual	Manual	Second Owner	96.1Nm@ 3000rpm
1624	Hyundai Santro Xing GL	2008	128000	Petrol	Individual	Manual	First Owner	96.1Nm@ 3000rpm
1652	Mercedes-Benz M-Class ML 350 4Matic	2011	110000	Diesel	Individual	Automatic	Third Owner	510@ 1600-2400
2091	Land Rover Freelander 2 TD4 HSE	2013	64788	Diesel	Dealer	Automatic	First Owner	400 Nm /2000 rpm
2327	Hyundai Santro Xing (Non-AC)	2010	80000	Petrol	Individual	Manual	Second Owner	96.1Nm@ 3000rpm
2674	Hyundai Santro Xing (Non-AC)	2013	15000	Petrol	Individual	Manual	First Owner	96.1Nm@ 3000rpm
4882	Hyundai Santro Xing GL	2008	40000	Petrol	Individual	Manual	First Owner	96.1Nm@ 3000rpm
5442	Volkswagen Polo GT TSI BSIV	2014	28080	Petrol	Dealer	Automatic	First Owner	175nm@ 1500-4100rpm
5445	Volkswagen Polo GT TSI BSIV	2014	28100	Petrol	Dealer	Automatic	First Owner	175nm@ 1500-4100rpm
5499	Mahindra Bolero Pik-Up FB 1.7T	2020	5000	Diesel	Individual	Manual	First Owner	200Nm@ 1400-2200rpm
6049	Hyundai Santro Xing GL	2010	110000	Petrol	Individual	Manual	First Owner	96.1Nm@ 3000rpm
6106	Mahindra Bolero Pik-Up CBC 1.7T	2019	80000	Diesel	Individual	Manual	First Owner	200Nm@ 1400-2200rpm

	name	year	km_driven	fuel	seller_type	transmission	owner	torque
6292	Hyundai Santro Xing GL	2011	40000	Petrol	Individual	Manual	Fourth & Above Owner	96.1Nm@ 3000rpm
6778	Mercedes-Benz GLC 220d 4MATIC	2017	60000	Diesel	Dealer	Automatic	First Owner	400nm@ 2800rpm

In [72]:
```
cars_data.loc[cars_data["mileage(kmpl)"]<LB,"mileage(kmpl)"]=LB
```

In [73]:
```
cars_data.loc[cars_data["mileage(kmpl)"]<LB]
```

Out[73]:

name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [74]:
```
#handling outliers
Q1=num_data["max_power(bhp)"].quantile(0.25)
Q3=num_data["max_power(bhp)"].quantile(0.75)
IQR=Q3-Q1
UB=Q3+(1.5*IQR)
LB=Q1-(1.5*IQR)
print(UB)
print(LB)
```

```
145.39999999999998
21.56000000000001
```

In [75]:
```
cars_data.loc[cars_data["max_power(bhp)"]>UB]
```

Out[75]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	se
45	Jeep Compass 1.4 Limited Plus BSIV	2019	5000	Petrol	Individual	Automatic	First Owner	250Nm@ 1750-2500rpm	
51	Toyota Fortuner 4x4 MT	2014	77000	Diesel	Dealer	Manual	First Owner	343Nm@ 1400-3400rpm	
55	Mitsubishi Pajero Sport 4X4	2013	151000	Diesel	Dealer	Manual	First Owner	400Nm@ 2000-2500rpm	
59	Toyota Innova Crysta 2.8 ZX AT BSIV	2016	127700	Diesel	Dealer	Automatic	Second Owner	360Nm@ 1200-3400rpm	
61	Audi A6 2.0 TDI Technology	2013	33900	Diesel	Dealer	Automatic	Second Owner	380Nm@ 1750-2500rpm	
...	...	...	...	...	...	...	...	...	
7335	Honda CR-V 2.4L 4WD AT	2012	50171	Petrol	Dealer	Automatic	First Owner	218Nm@ 4200rpm	
7340	Honda Accord 2.4 A/T	2013	50000	Petrol	Dealer	Automatic	First Owner	222Nm@ 4300rpm	
7341	Audi A3 40 TFSI Premium	2017	8000	Petrol	Dealer	Automatic	First Owner	250Nm@ 1250-5000rpm	
7368	Chevrolet Captiva LT	2008	100000	Diesel	Individual	Manual	Second Owner	320Nm@ 2000rpm	
7370	Chevrolet Cruze LT	2011	40000	Diesel	Individual	Manual	First Owner	327Nm@ 2600rpm	

377 rows × 13 columns

In [76]:
```
cars_data.loc[cars_data["max_power(bhp)"]>UB,"max_power(bhp)"]=UB
```

In [77]:
```
cars_data.loc[cars_data["max_power(bhp)"]>UB]
```

Out[77]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [78]:
```
cars_data.loc[cars_data["max_power(bhp)"]<LB]
```

Out[78]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats
597	Maruti Alto K10 LXI	2011	97500	Petrol	Individual	Manual	First Owner	190Nm@ 2000rpm	5.0
1408	Maruti Swift Dzire VDI Optional	2017	41232	Diesel	Dealer	Manual	First Owner	190Nm@ 2000rpm	5.0
2523	Tata Indica Vista Quadrajet LS	2012	70000	Diesel	Individual	Manual	First Owner	190Nm@ 2000rpm	5.0

In [79]:
```
cars_data.loc[cars_data["max_power(bhp)"]<LB,"max_power(bhp)"]=LB
```

In [80]:
```
cars_data.loc[cars_data["max_power(bhp)"]<LB]
```

Out[80]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [81]:
```
#handling outliers
Q1=num_data["engine(CC)"].quantile(0.25)
Q3=num_data["engine(CC)"].quantile(0.75)
IQR=Q3-Q1
UB=Q3+(1.5*IQR)
LB=Q1-(1.5*IQR)
print(UB)
print(LB)
```

```
1949.5
745.5
```

In [82]:
```
cars_data.loc[cars_data["engine(CC)"]>UB]
```

Out[82]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque	s
43	Tata Safari DICOR 2.2 LX 4x2	2011	60000	Diesel	Individual	Manual	Second Owner	320Nm@ 1700-2700rpm	
51	Toyota Fortuner 4x4 MT	2014	77000	Diesel	Dealer	Manual	First Owner	343Nm@ 1400-3400rpm	
52	Toyota Innova 2.5 G (Diesel) 7 Seater BS IV	2013	99000	Diesel	Dealer	Manual	First Owner	200Nm@ 1400-3400rpm	
53	Mercedes-Benz B Class B180	2014	27800	Diesel	Dealer	Automatic	Second Owner	200Nm@ 1250-4000rpm	
55	Mitsubishi Pajero Sport 4X4	2013	151000	Diesel	Dealer	Manual	First Owner	400Nm@ 2000-2500rpm	
...	...	...	...	...	...	...	...	...	
7365	Tata New Safari DICOR 2.2 EX 4x2	2010	100000	Diesel	Individual	Manual	First Owner	320Nm@ 1700-2700rpm	
7368	Chevrolet Captiva LT	2008	100000	Diesel	Individual	Manual	Second Owner	320Nm@ 2000rpm	
7370	Chevrolet Cruze LT	2011	40000	Diesel	Individual	Manual	First Owner	327Nm@ 2600rpm	
7384	Mahindra XUV500 W8 2WD	2013	120000	Diesel	Individual	Manual	First Owner	330Nm@ 1600-2800rpm	
7400	Toyota Innova 2.5 GX (Diesel) 8 Seater BS IV	2010	129000	Diesel	Individual	Manual	First Owner	20.4@ 1400-3400(kgm@ rpm)	

1232 rows × 13 columns

In [83]:
```
cars_data.loc[cars_data["engine(CC)"]>UB,"engine(CC)"]=UB
```

In [84]:
```
cars_data.loc[cars_data["engine(CC)"]>UB]
```

Out[84]:

name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [85]:
```
cars_data.loc[cars_data["engine(CC)"]<LB]
```

Out[85]:

	name	year	km_driven	fuel	seller_type	transmission	owner	torque
389	Tata Nano STD	2012	18000	Petrol	Dealer	Manual	First Owner	51Nm@ 4000rpm
716	Tata Nano Cx	2011	10000	Petrol	Individual	Manual	Third Owner	48Nm@ 3000rpm
1176	Tata Nano CX	2013	30000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
1177	Tata Nano CX	2013	30000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
1227	Tata Nano Cx BSIV	2010	50000	Petrol	Individual	Manual	Third Owner	48@ 3,000+/-500(NM@ rpm)
1391	Tata Nano XTA	2015	20000	Petrol	Individual	Automatic	First Owner	51Nm@ 4000rpm
2226	Tata Nano CX	2013	30000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
2227	Tata Nano CX	2013	30000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
2386	Tata Nano LX	2015	60000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
2419	Tata Nano Lx BSIV	2010	26432	Petrol	Dealer	Manual	First Owner	51Nm@ 4000+/-500rpm
2720	Tata Nano Cx	2009	15000	Petrol	Individual	Manual	First Owner	48Nm@ 3000rpm
2802	Tata Nano Twist XT	2015	25000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
3576	Tata Nano XE	2017	70000	Petrol	Individual	Manual	Third Owner	51Nm@ 4000rpm

	name	year	km_driven	fuel	seller_type	transmission	owner	torque
3796	Tata Nano Cx BSIII	2012	40000	Petrol	Individual	Manual	Second Owner	51Nm@ 4000+/-500rpm
4116	Tata Nano XTA	2018	55000	Petrol	Dealer	Automatic	First Owner	51Nm@ 4000rpm
4240	Tata Nano XE	2015	60000	Petrol	Individual	Manual	First Owner	51Nm@ 4000rpm
4535	Tata Nano XTA	2016	22000	Petrol	Individual	Automatic	First Owner	51Nm@ 4000rpm
4835	Tata Nano Cx BSIII	2012	45000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
5286	Tata Nano Twist XE	2015	15000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
5476	Tata Nano Lx	2011	40000	Petrol	Individual	Manual	First Owner	4.8kgm@ 3000rpm
6047	Tata Nano Twist XT	2014	50000	Petrol	Individual	Manual	Second Owner	51Nm@ 4000+/-500rpm
6198	Tata Nano Lx BSIV	2010	30400	Petrol	Individual	Manual	Fourth & Above Owner	48@ 3,000+/-500(NM@ rpm)
6285	Tata Nano Lx BSIV	2012	90000	Petrol	Individual	Manual	First Owner	51Nm@ 4000+/-500rpm
6996	Tata Nano Lx BSIV	2012	50000	Petrol	Individual	Manual	Second Owner	51Nm@ 4000+/-500rpm
7004	Tata Nano Cx	2011	80000	Petrol	Individual	Manual	First Owner	48Nm@ 3000rpm

	name	year	km_driven	fuel	seller_type	transmission	owner	torque
7167	Tata Nano LX SE	2012	28000	Petrol	Individual	Manual	Second Owner	51Nm@ 4000+/-500rpm
7374	Tata Nano Cx	2011	25000	Petrol	Individual	Manual	Second Owner	48Nm@ 3000rpm

In [86]:
```
cars_data.loc[cars_data["engine(CC)"]<LB,"engine(CC)"]=LB
```

In [87]:
```
cars_data.loc[cars_data["engine(CC)"]<LB]
```

Out[87]:

name	year	km_driven	fuel	seller_type	transmission	owner	torque	seats	mileage(k

In [92]:
```
cars_data["transmission"]=cars_data["transmission"].map({"Manual":1,"Automatic":0})
```

In [93]:
```
cars_data["transmission"].value_counts()
```

Out[93]:
```
transmission
1    6814
0     598
Name: count, dtype: int64
```

In [94]:
```
cars_data=pd.get_dummies(cars_data,columns=["fuel"])
```

In [95]:
```
cars_data[["fuel_CNG","fuel_Diesel","fuel_LPG","fuel_Petrol"]]=cars_data[["fuel_CNG
```

In [96]:
```
cars_data
```

Out[96]:

	name	year	km_driven	seller_type	transmission	owner	torque	seats	m
0	Maruti Swift Dzire VDI	2014	145500	Individual	1	First Owner	190Nm@ 2000rpm	5.0	
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	Individual	1	Second Owner	250Nm@ 1500-2500rpm	5.0	
2	Honda City 2017-2020 EXi	2006	140000	Individual	1	Third Owner	12.7@ 2,700(kgm@ rpm)	5.0	
3	Honda City 2017-2020 EXi	2006	140000	Individual	1	Third Owner	12.7@ 2,700(kgm@ rpm)	5.0	
4	Hyundai i20 Sportz Diesel	2010	127000	Individual	1	First Owner	22.4 kgm at 1750-2750rpm	5.0	
...	...	...	...	...	...	...	...	...	
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	Individual	1	Second Owner	90Nm@ 3500rpm	5.0	
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	Individual	1	Second Owner	219.7Nm@ 1500-2750rpm	5.0	
7409	Hyundai i20 Magna	2013	110000	Individual	1	First Owner	113.7Nm@ 4000rpm	5.0	
7410	Hyundai Verna CRDi SX	2007	119000	Individual	1	Fourth & Above Owner	24@ 1,900-2,750(kgm@ rpm)	5.0	
7411	Maruti Swift Dzire ZDi	2009	120000	Individual	1	First Owner	190Nm@ 2000rpm	5.0	

7412 rows × 16 columns

In [97]:
```
cars_data
```

Out[97]:

	name	year	km_driven	seller_type	transmission	owner	torque	seats	m
0	Maruti Swift Dzire VDI	2014	145500	Individual	1	First Owner	190Nm@ 2000rpm	5.0	
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	Individual	1	Second Owner	250Nm@ 1500-2500rpm	5.0	
2	Honda City 2017-2020 EXi	2006	140000	Individual	1	Third Owner	12.7@ 2,700(kgm@ rpm)	5.0	
3	Honda City 2017-2020 EXi	2006	140000	Individual	1	Third Owner	12.7@ 2,700(kgm@ rpm)	5.0	
4	Hyundai i20 Sportz Diesel	2010	127000	Individual	1	First Owner	22.4 kgm at 1750-2750rpm	5.0	
...	...	...	...	...	...	...	...	...	
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	Individual	1	Second Owner	90Nm@ 3500rpm	5.0	
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	Individual	1	Second Owner	219.7Nm@ 1500-2750rpm	5.0	
7409	Hyundai i20 Magna	2013	110000	Individual	1	First Owner	113.7Nm@ 4000rpm	5.0	
7410	Hyundai Verna CRDi SX	2007	119000	Individual	1	Fourth & Above Owner	24@ 1,900-2,750(kgm@ rpm)	5.0	
7411	Maruti Swift Dzire ZDi	2009	120000	Individual	1	First Owner	190Nm@ 2000rpm	5.0	

7412 rows × 16 columns

In [98]:
```
cars_data=pd.get_dummies(cars_data,columns=["seller_type"])
```

In [99]:
```
cars_data[["seller_type_Dealer","seller_type_Individual","seller_type_Trustmark Dea
```

In [100…
```
cars_data
```

Out[100…

	name	year	km_driven	transmission	owner	torque	seats	mileage(kmpl)
0	Maruti Swift Dzire VDI	2014	145500	1	First Owner	190Nm@ 2000rpm	5.0	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	1	Second Owner	250Nm@ 1500-2500rpm	5.0	21.14
2	Honda City 2017-2020 EXi	2006	140000	1	Third Owner	12.7@ 2,700(kgm@ rpm)	5.0	17.70
3	Honda City 2017-2020 EXi	2006	140000	1	Third Owner	12.7@ 2,700(kgm@ rpm)	5.0	17.70
4	Hyundai i20 Sportz Diesel	2010	127000	1	First Owner	22.4 kgm at 1750-2750rpm	5.0	23.00
…	…	…	…	…	…	…	…	…
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	1	Second Owner	90Nm@ 3500rpm	5.0	18.90
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	1	Second Owner	219.7Nm@ 1500-2750rpm	5.0	22.54
7409	Hyundai i20 Magna	2013	110000	1	First Owner	113.7Nm@ 4000rpm	5.0	18.50
7410	Hyundai Verna CRDi SX	2007	119000	1	Fourth & Above Owner	24@ 1,900-2,750(kgm@ rpm)	5.0	16.80
7411	Maruti Swift Dzire ZDi	2009	120000	1	First Owner	190Nm@ 2000rpm	5.0	19.30

7412 rows × 18 columns

In [101…
```
cars_data=pd.get_dummies(cars_data,columns=["owner"])
```

In [102…
```
cars_data[["owner_First Owner","owner_Fourth & Above Owner","owner_Second Owner","o
```

In [103…
```
cars_data
```

Out[103…

	name	year	km_driven	transmission	torque	seats	mileage(kmpl)	engine(
0	Maruti Swift Dzire VDI	2014	145500	1	190Nm@ 2000rpm	5.0	23.40	12
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	1	250Nm@ 1500-2500rpm	5.0	21.14	14
2	Honda City 2017-2020 EXi	2006	140000	1	12.7@ 2,700(kgm@ rpm)	5.0	17.70	14
3	Honda City 2017-2020 EXi	2006	140000	1	12.7@ 2,700(kgm@ rpm)	5.0	17.70	14
4	Hyundai i20 Sportz Diesel	2010	127000	1	22.4 kgm at 1750-2750rpm	5.0	23.00	13
...	...	...	...	...	...	...	...	
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	1	90Nm@ 3500rpm	5.0	18.90	9
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	1	219.7Nm@ 1500-2750rpm	5.0	22.54	13
7409	Hyundai i20 Magna	2013	110000	1	113.7Nm@ 4000rpm	5.0	18.50	11
7410	Hyundai Verna CRDi SX	2007	119000	1	24@ 1,900-2,750(kgm@ rpm)	5.0	16.80	14
7411	Maruti Swift Dzire ZDi	2009	120000	1	190Nm@ 2000rpm	5.0	19.30	12

7412 rows × 22 columns

In [104…
```
cars_data.drop(["fuel_CNG","seller_type_Trustmark Dealer","owner_First Owner"],axis
```

In [105…
```
cars_data.value_counts().sum()
```

Out[105… 7412

In [106…
```
cars_data
```

Out[106…

	name	year	km_driven	transmission	torque	seats	mileage(kmpl)	engine(
0	Maruti Swift Dzire VDI	2014	145500	1	190Nm@ 2000rpm	5.0	23.40	12
1	Skoda Rapid 1.5 TDI Ambition	2014	120000	1	250Nm@ 1500-2500rpm	5.0	21.14	14
2	Honda City 2017-2020 EXi	2006	140000	1	12.7@ 2,700(kgm@ rpm)	5.0	17.70	14
3	Honda City 2017-2020 EXi	2006	140000	1	12.7@ 2,700(kgm@ rpm)	5.0	17.70	14
4	Hyundai i20 Sportz Diesel	2010	127000	1	22.4 kgm at 1750-2750rpm	5.0	23.00	13
...	...	...	...	...	...	...	...	
7407	Maruti Wagon R VXI BS IV with ABS	2013	50000	1	90Nm@ 3500rpm	5.0	18.90	9
7408	Hyundai i20 Magna 1.4 CRDi	2014	80000	1	219.7Nm@ 1500-2750rpm	5.0	22.54	13
7409	Hyundai i20 Magna	2013	110000	1	113.7Nm@ 4000rpm	5.0	18.50	11
7410	Hyundai Verna CRDi SX	2007	119000	1	24@ 1,900-2,750(kgm@ rpm)	5.0	16.80	14
7411	Maruti Swift Dzire ZDi	2009	120000	1	190Nm@ 2000rpm	5.0	19.30	12

7412 rows × 19 columns

In [107…
```
cars=cars_data.drop("name",axis=1)
```

In [108…
```
cars_df=cars.drop("torque",axis=1)
```

In [109…
```
cars_df
```

Out[109…

	year	km_driven	transmission	seats	mileage(kmpl)	engine(CC)	max_power(bhp)
0	2014	145500	1	5.0	23.40	1248.0	74.00
1	2014	120000	1	5.0	21.14	1498.0	103.52
2	2006	140000	1	5.0	17.70	1497.0	78.00
3	2006	140000	1	5.0	17.70	1497.0	78.00
4	2010	127000	1	5.0	23.00	1396.0	90.00
…	…	…	…	…	…	…	…
7407	2013	50000	1	5.0	18.90	998.0	67.10
7408	2014	80000	1	5.0	22.54	1396.0	88.73
7409	2013	110000	1	5.0	18.50	1197.0	82.85
7410	2007	119000	1	5.0	16.80	1493.0	110.00
7411	2009	120000	1	5.0	19.30	1248.0	73.90

7412 rows × 17 columns

In [110…
```
cars_df.corr()
```

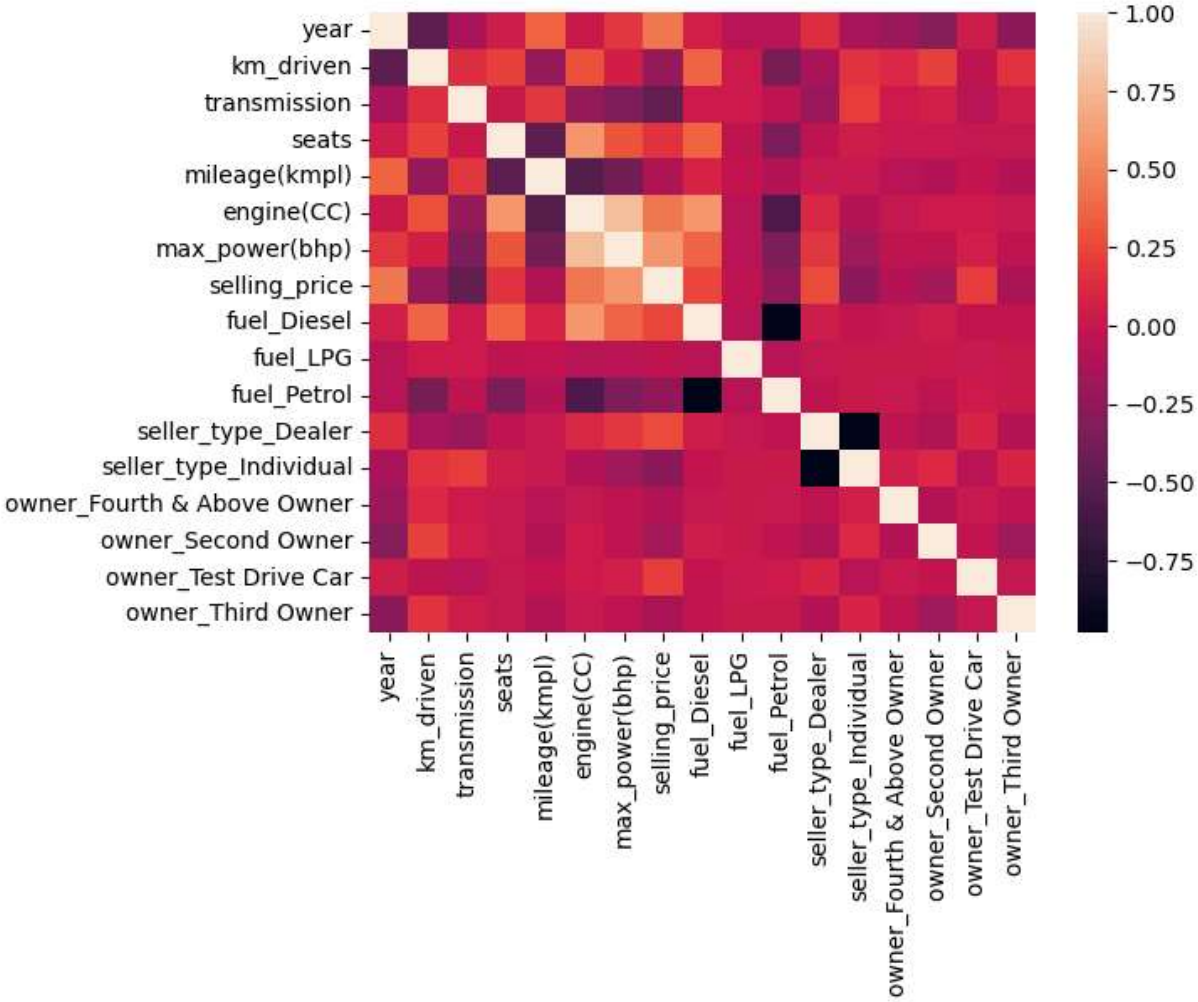
Out[110…

	year	km_driven	transmission	seats	mileage(kmpl)	engine
year	1.000000	-0.491126	-0.141002	0.040739	0.351974	0.01
km_driven	-0.491126	1.000000	0.147025	0.227081	-0.226650	0.28
transmission	-0.141002	0.147025	1.000000	0.017018	0.177826	-0.21
seats	0.040739	0.227081	0.017018	1.000000	-0.472994	0.58
mileage(kmpl)	0.351974	-0.226650	0.177826	-0.472994	1.000000	-0.53
engine(CC)	0.012544	0.289517	-0.216039	0.583609	-0.536935	1.00
max_power(bhp)	0.178667	0.065085	-0.351034	0.308126	-0.387527	0.77
selling_price	0.439045	-0.217106	-0.461970	0.166751	-0.118252	0.44
fuel_Diesel	0.061161	0.359095	0.020622	0.352886	0.080548	0.58
fuel_LPG	-0.064399	0.023961	0.021266	-0.030833	-0.017087	-0.06
fuel_Petrol	-0.056323	-0.361894	-0.028398	-0.343170	-0.094442	-0.56
seller_type_Dealer	0.140861	-0.146448	-0.209389	-0.029897	-0.003708	0.09
seller_type_Individual	-0.147784	0.154167	0.218938	0.033737	0.005862	-0.09
owner_Fourth & Above Owner	-0.208398	0.112121	0.029026	0.002203	-0.081258	-0.00
owner_Second Owner	-0.299049	0.233386	0.058627	0.003015	-0.102291	0.02
owner_Test Drive Car	0.036673	-0.034635	-0.068623	-0.011159	-0.017855	0.02
owner_Third Owner	-0.271993	0.169863	0.040161	-0.009856	-0.104961	0.00

In [111…

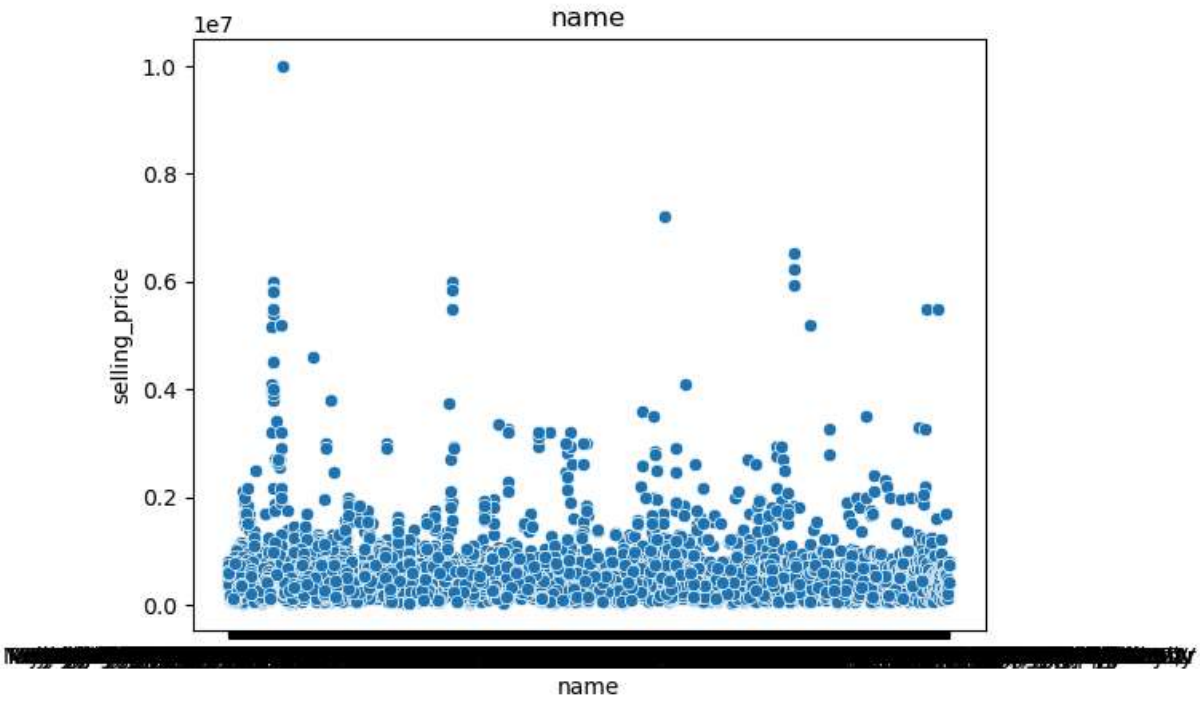
```
sns.heatmap(cars_df.corr())
```

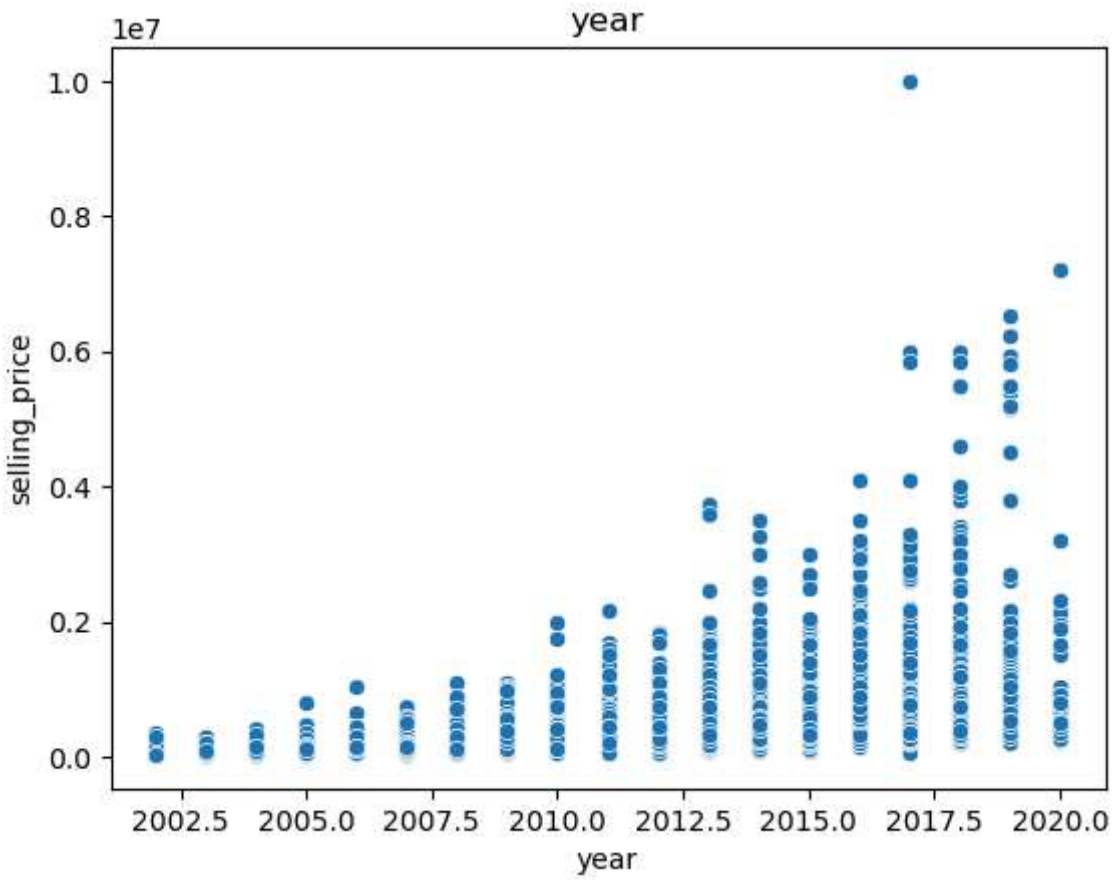
Out[111… <Axes: >

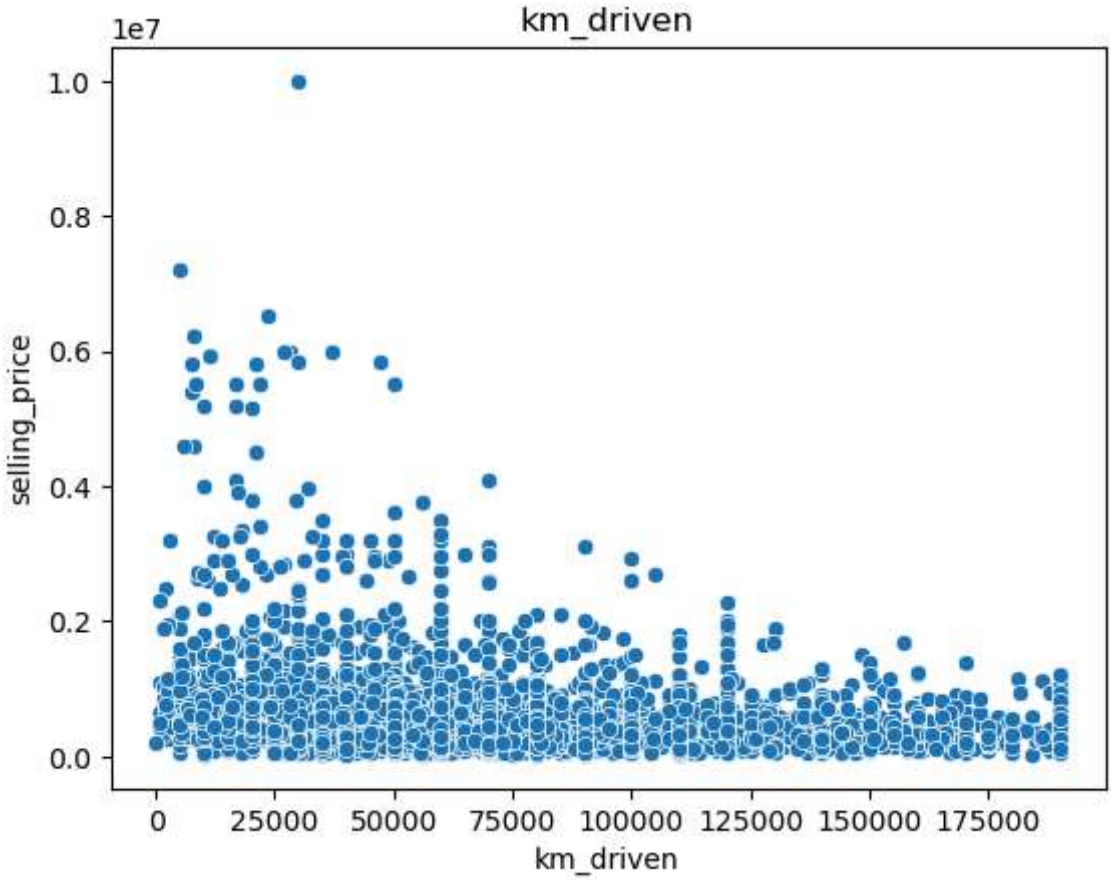


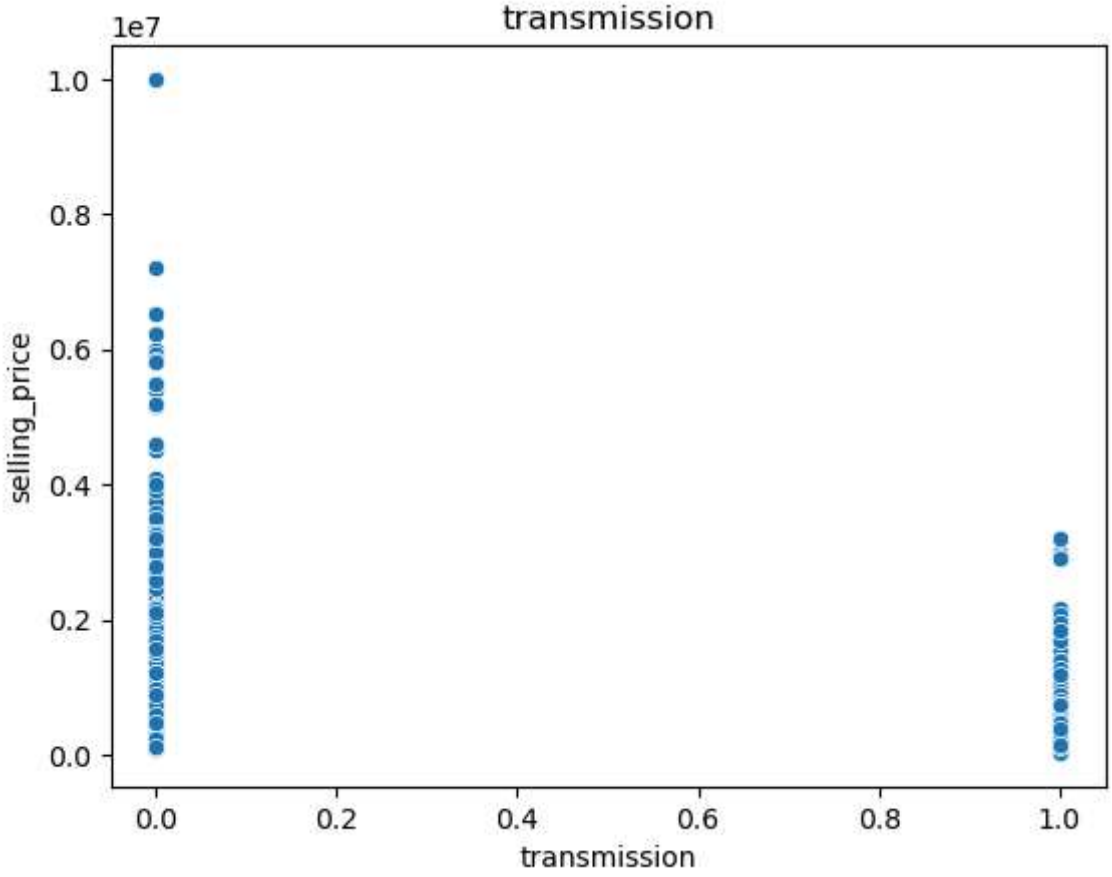
In [112…

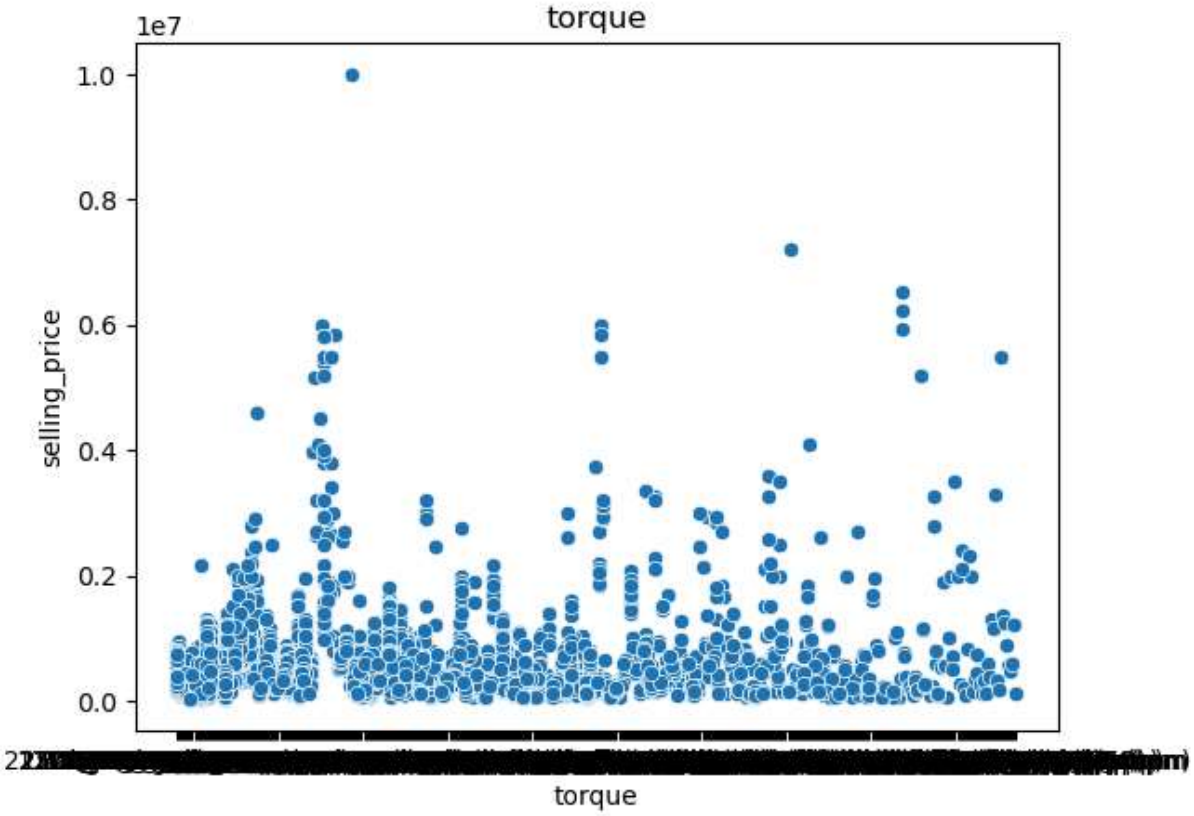
```
for i in cars_data:
    sns.scatterplot(cars_data,x=cars_data[i],y="selling_price")
    plt.title(i)
    plt.show()
```

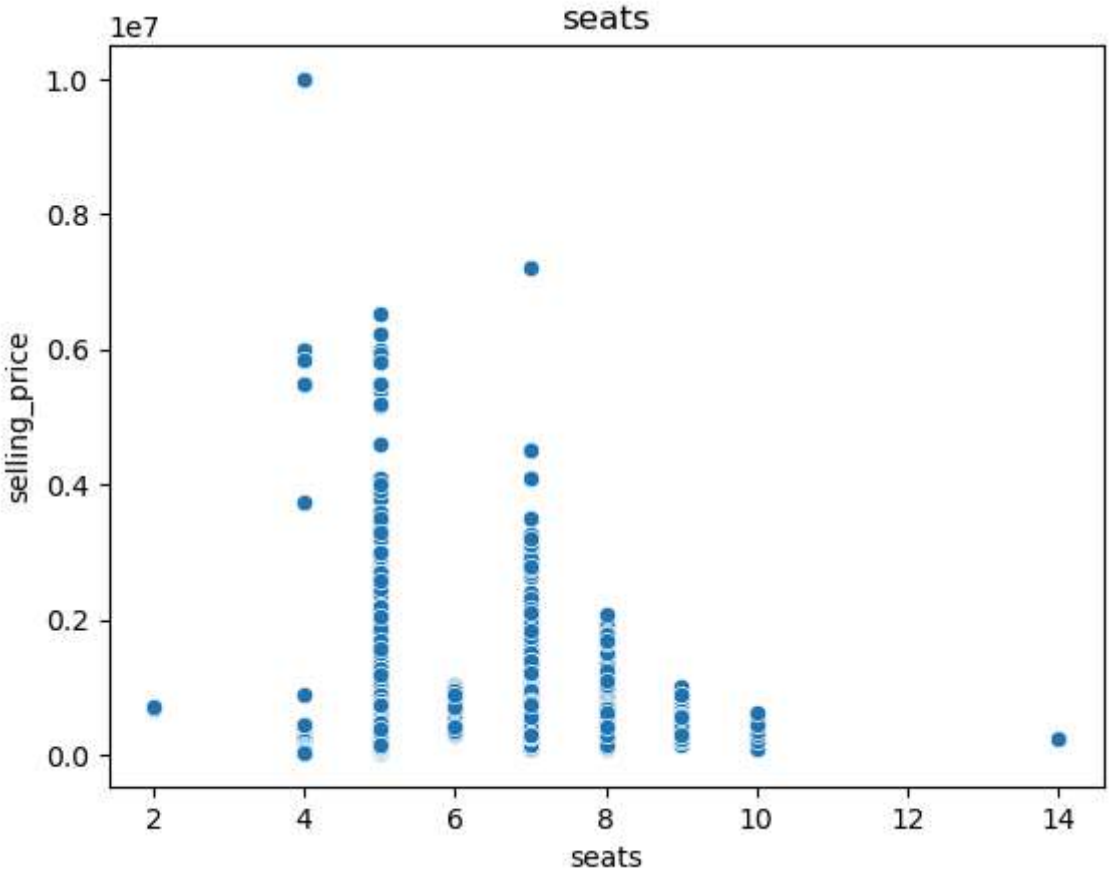

name
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
name

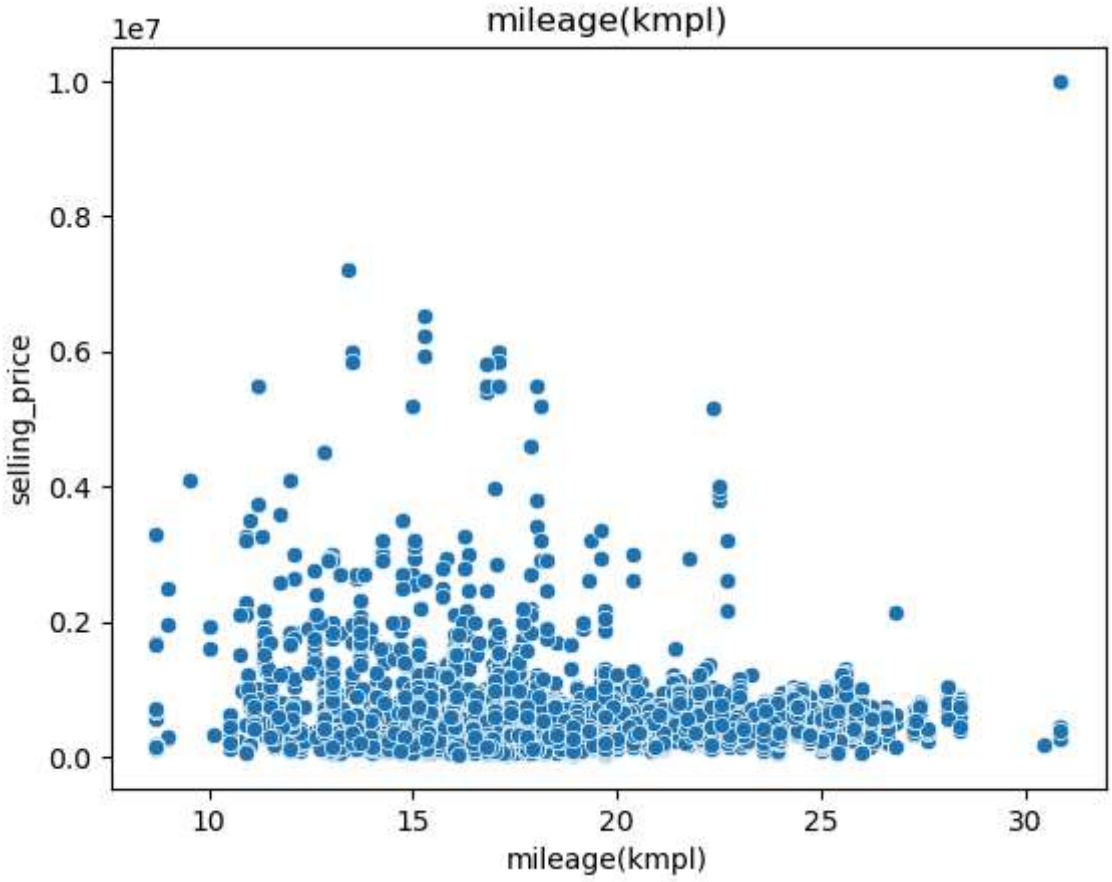

year
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
2002.5
2005.0
2007.5
2010.0
2012.5
2015.0
2017.5
2020.0
year

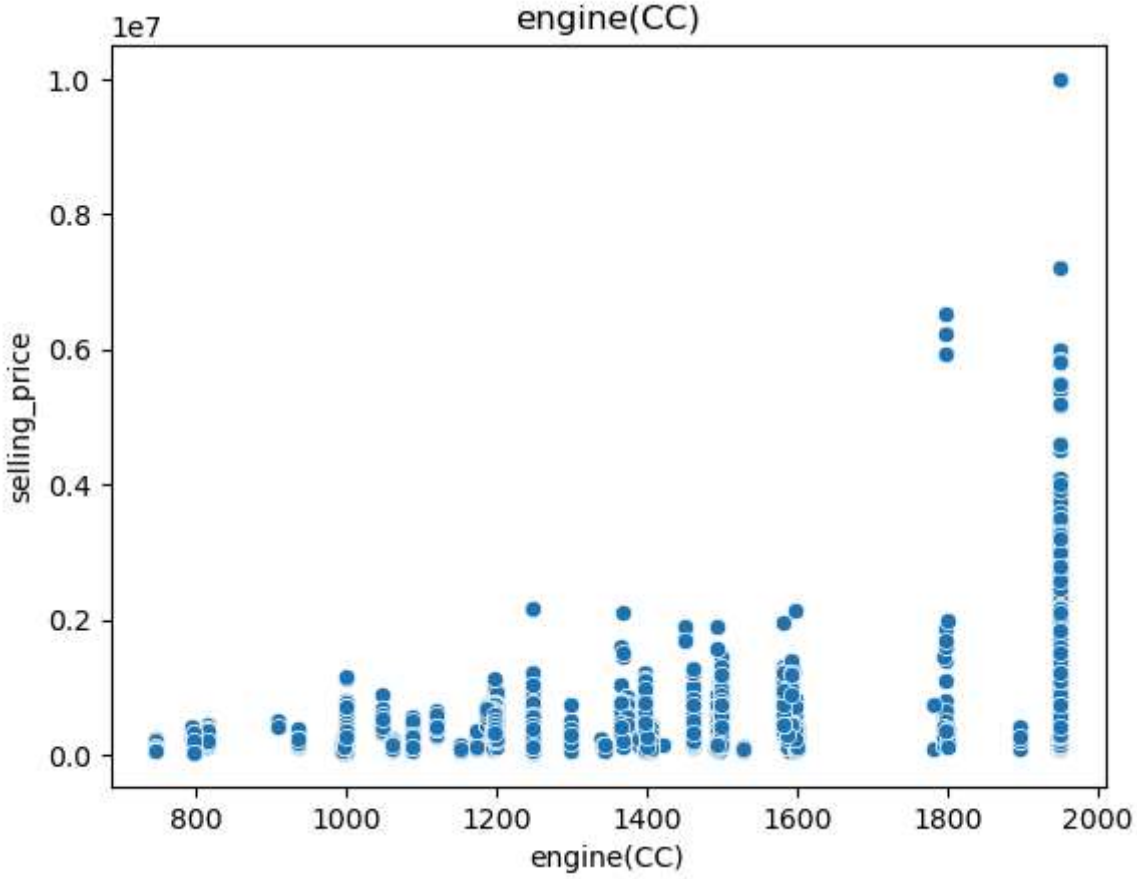

km_driven
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0
25000
50000
75000
100000
125000
150000
175000
km_driven

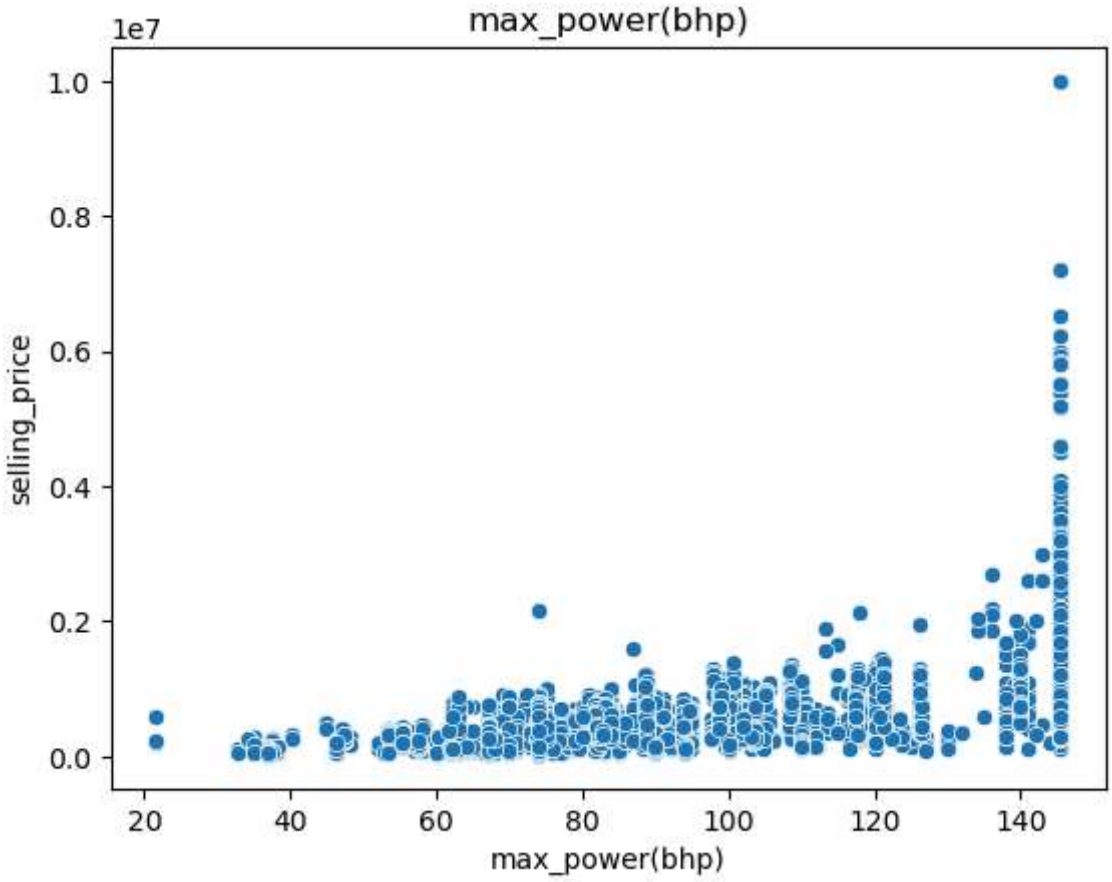

transmission
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
transmission

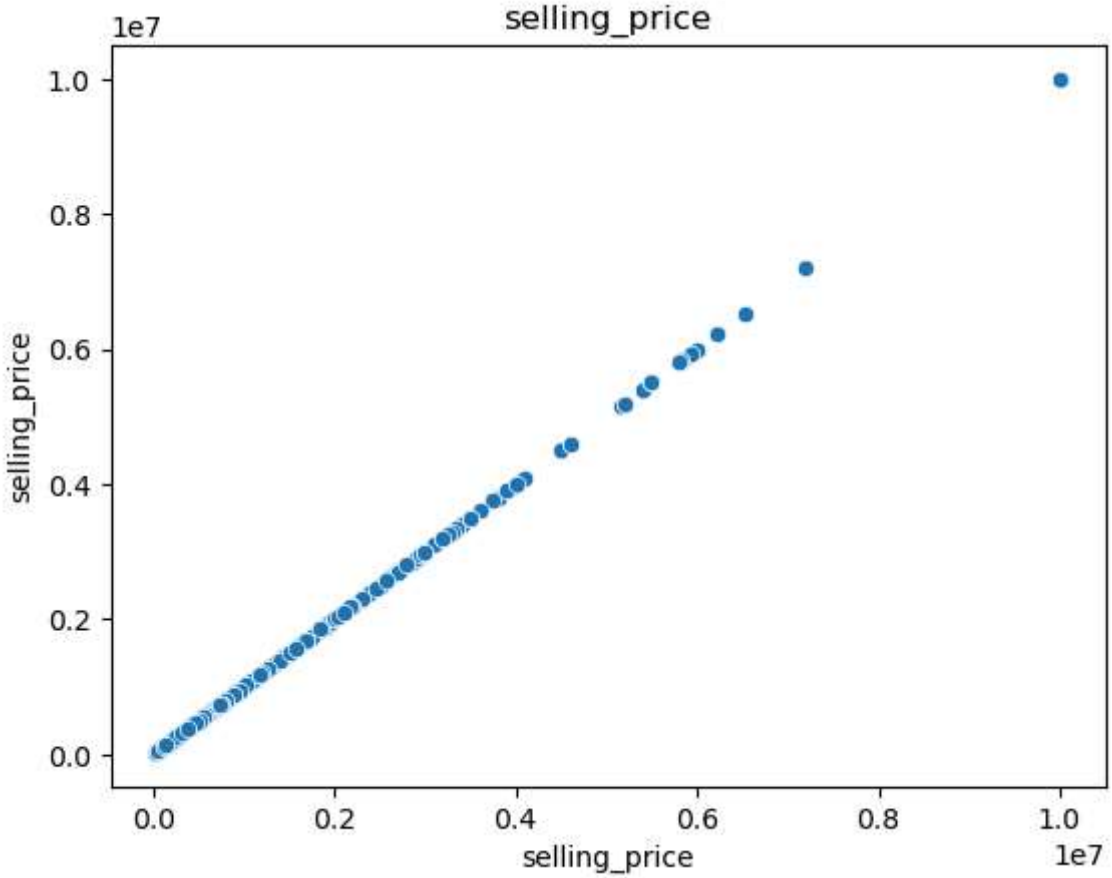

torque
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
torque

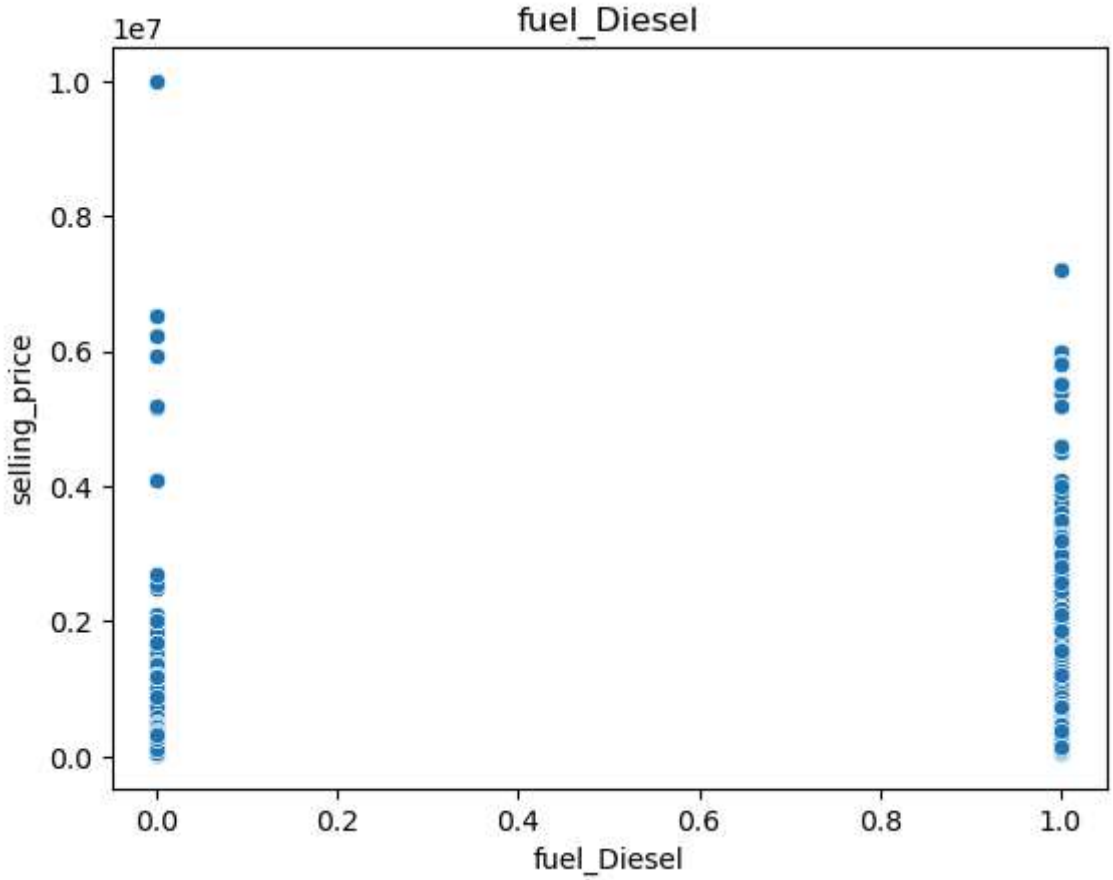

seats
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
2
4
6
8
10
12
14
seats

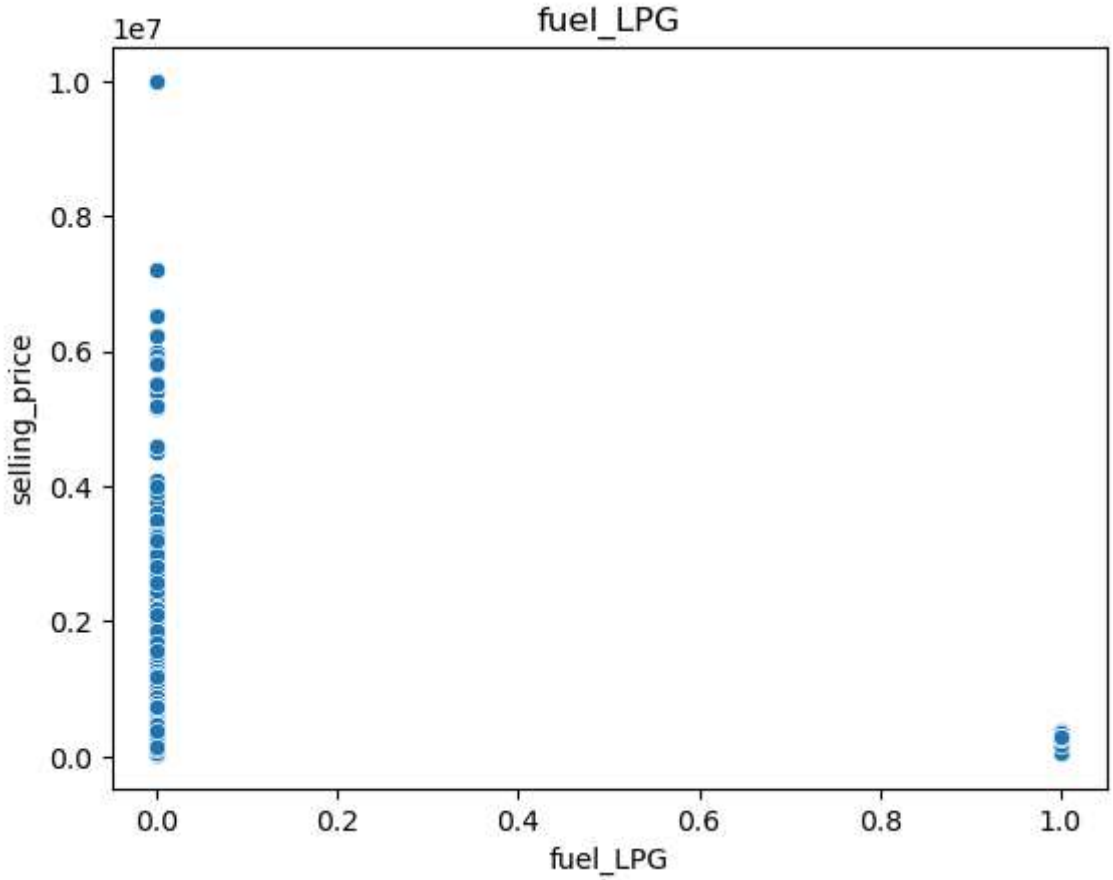

mileage(kmpl)
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
10
15
20
25
30
mileage(kmpl)

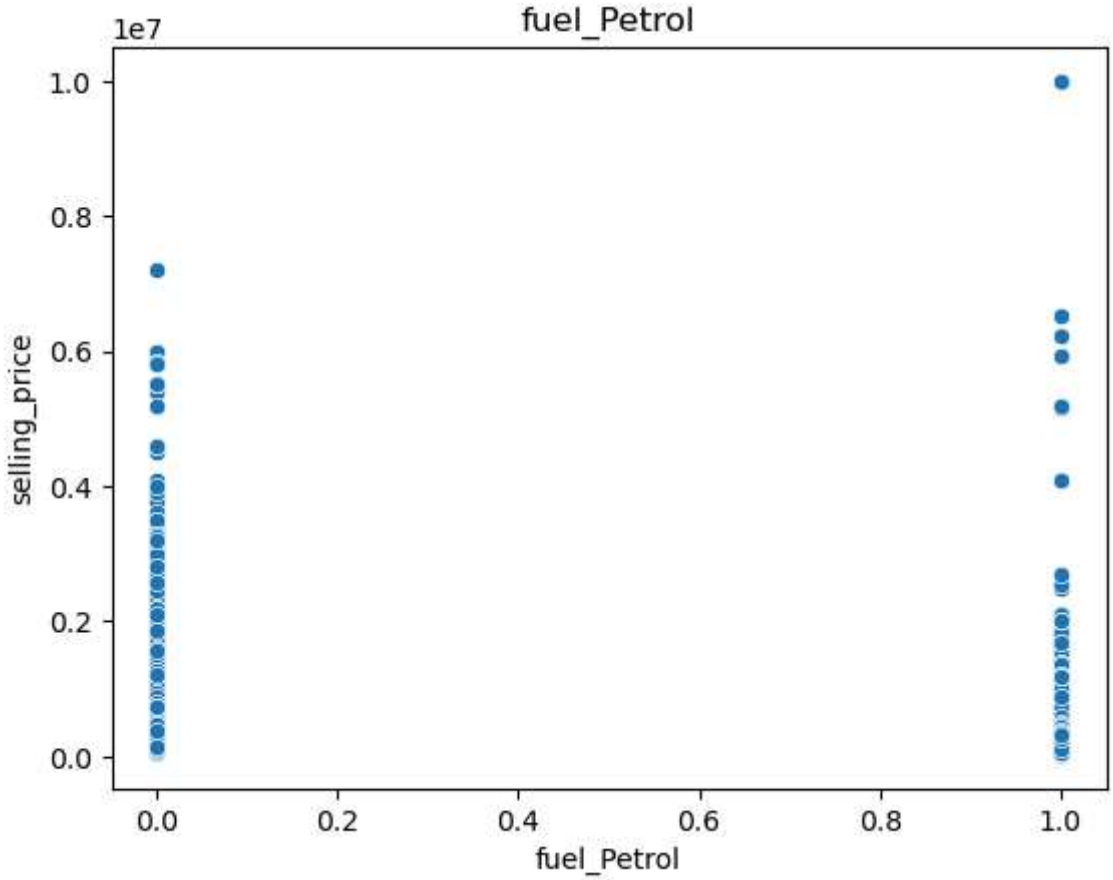

engine(CC)
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
800
1000
1200
1400
1600
1800
2000
engine(CC)

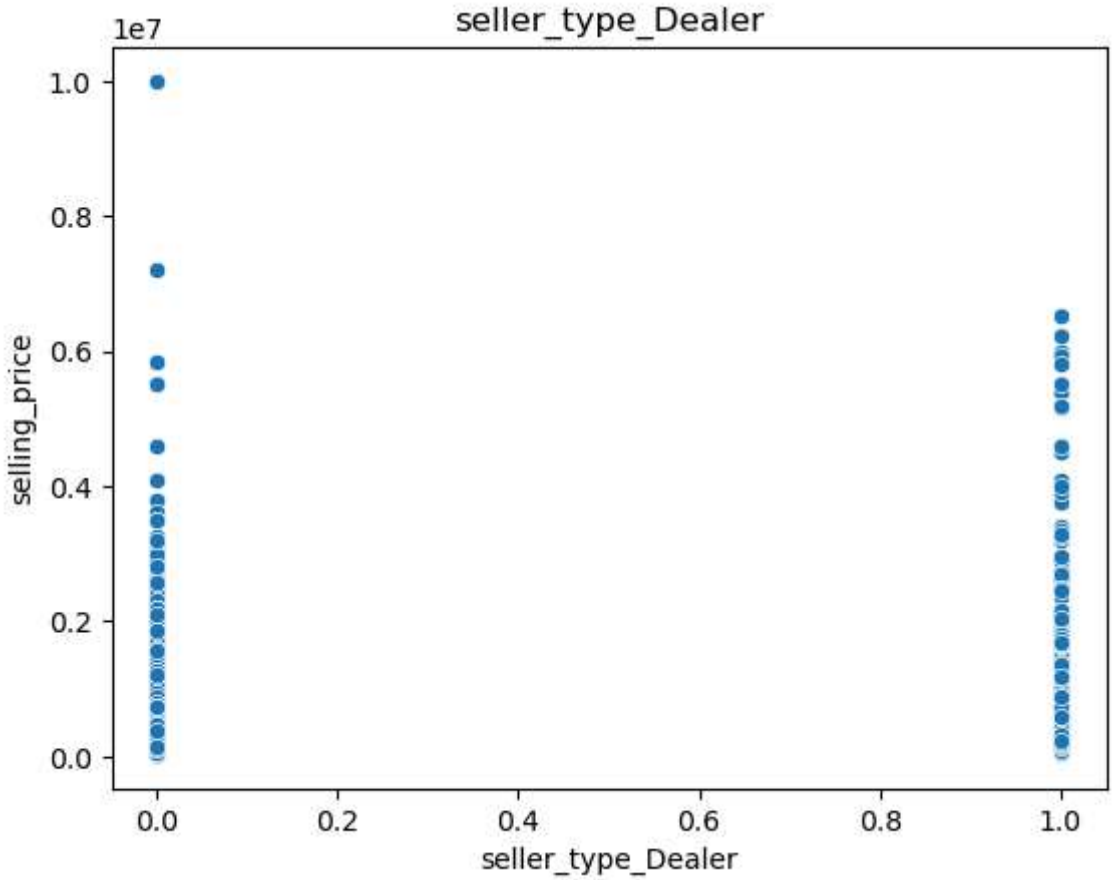

max_power(bhp)
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
20
40
60
80
100
120
140
max_power(bhp)

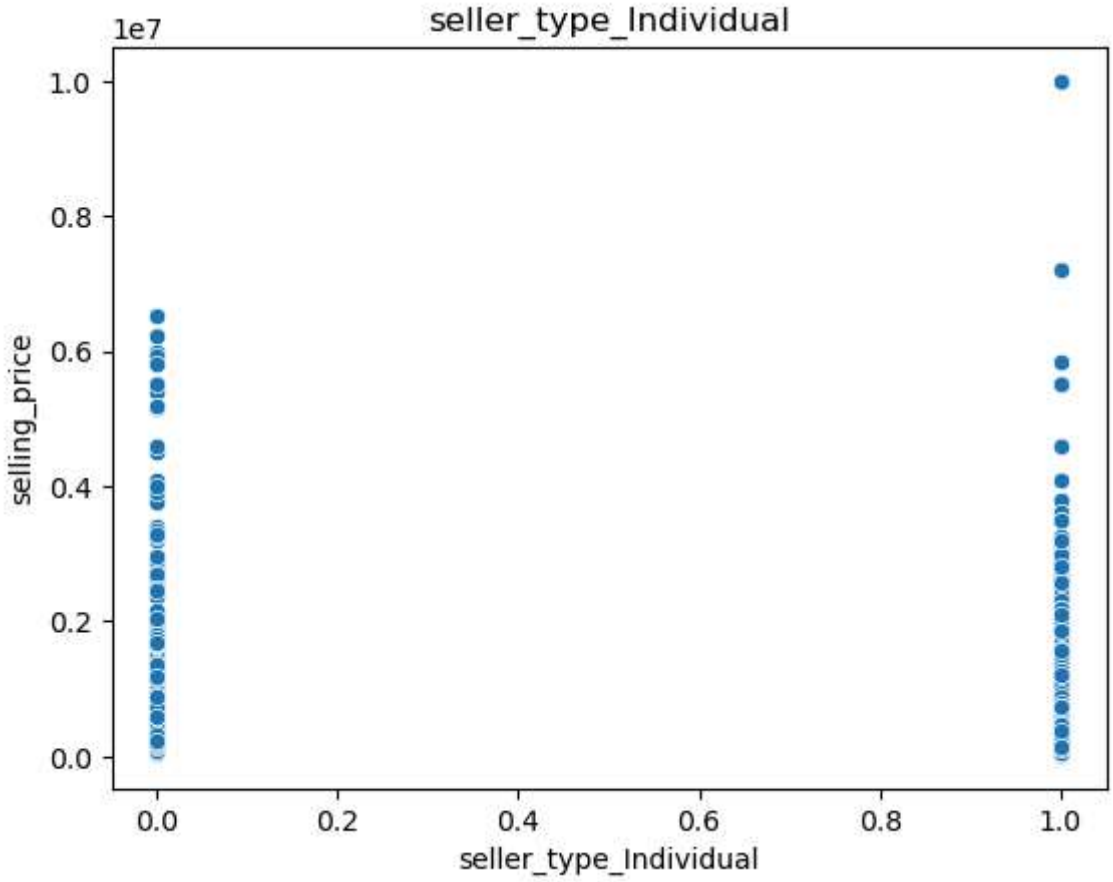

selling_price
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
selling_price
1e7

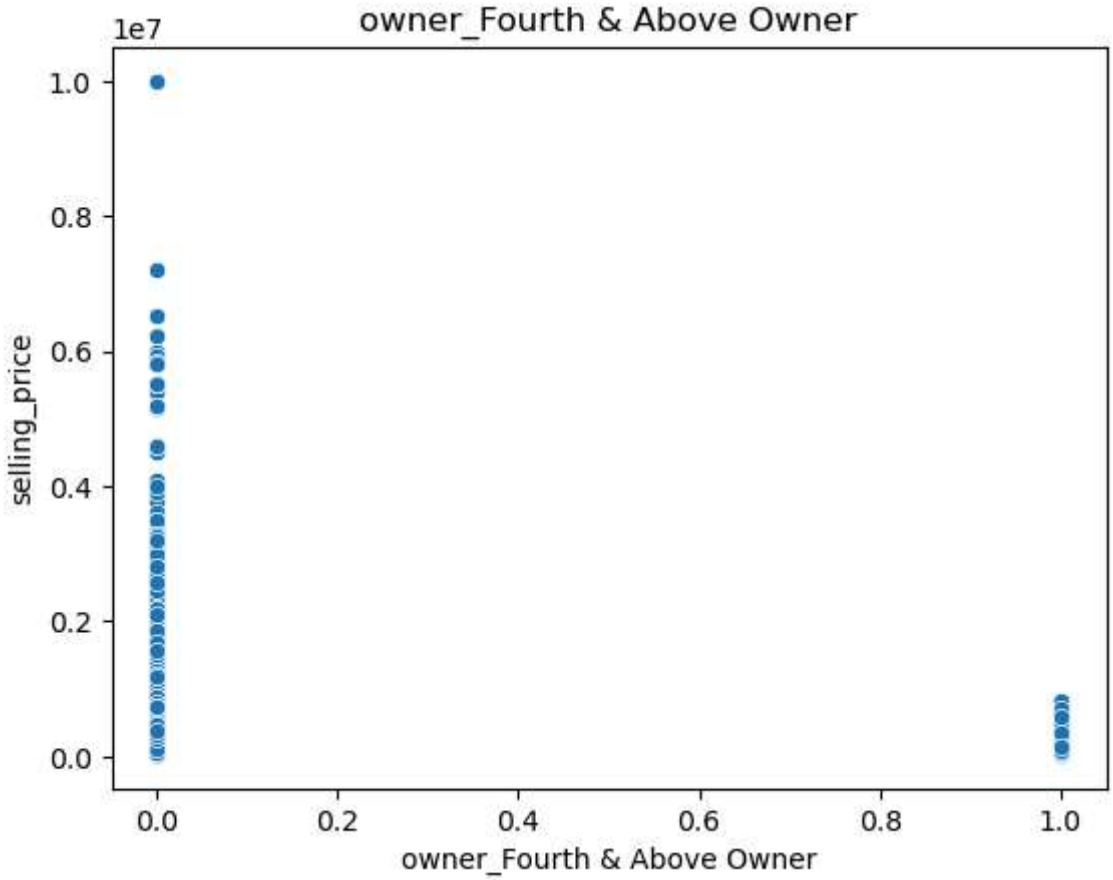

fuel_Diesel
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
fuel_Diesel

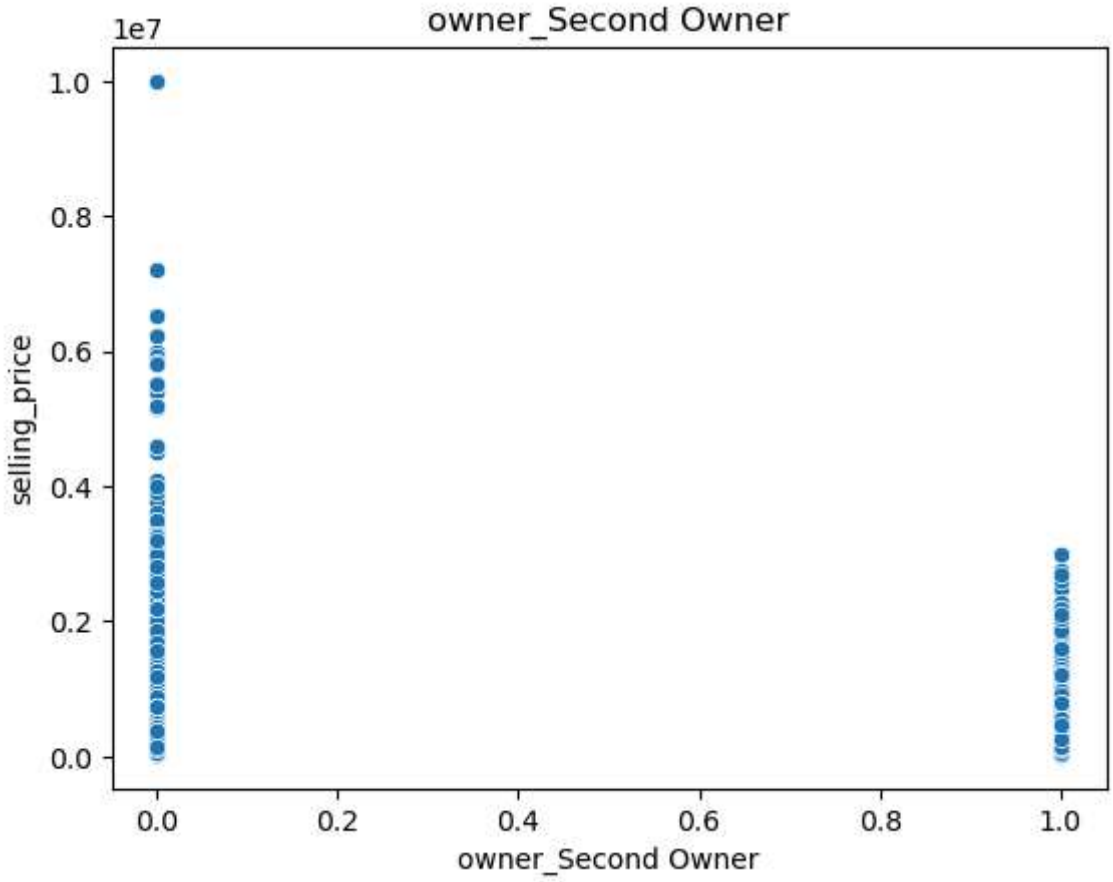

fuel_LPG
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
fuel_LPG

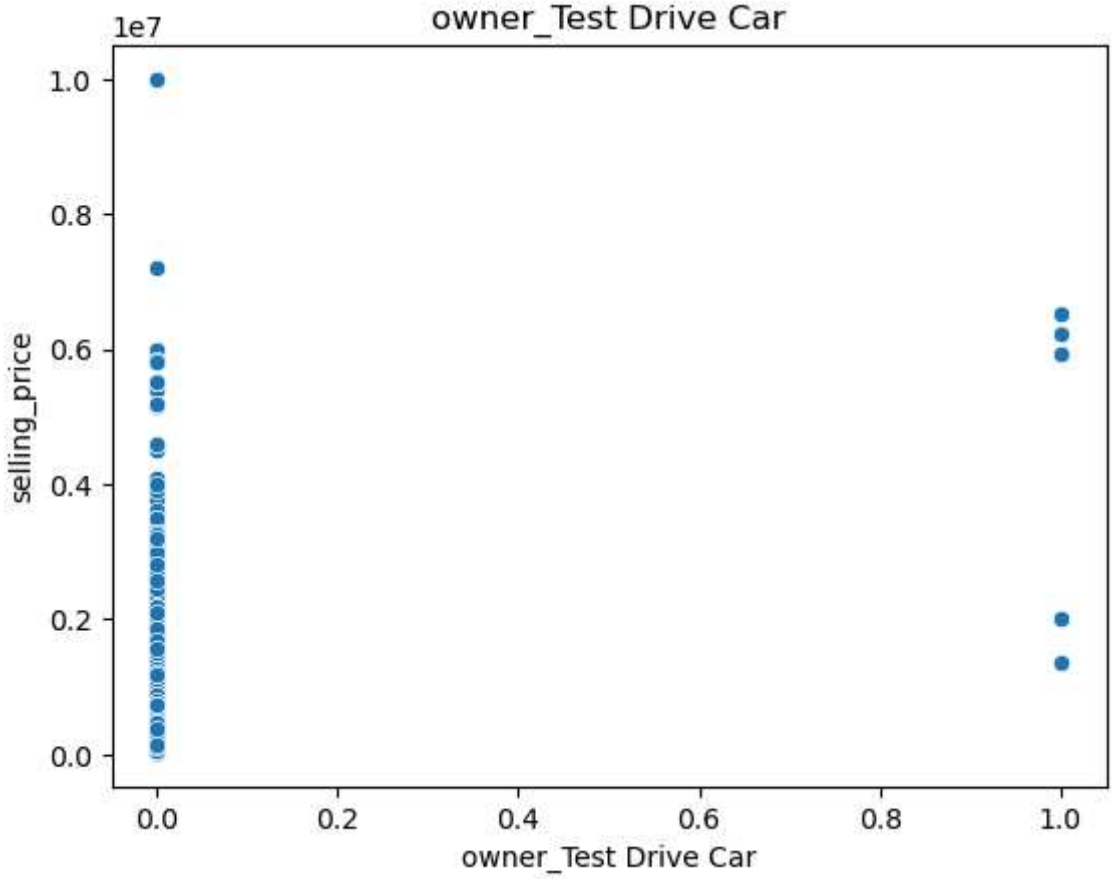

fuel_Petrol
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
fuel_Petrol

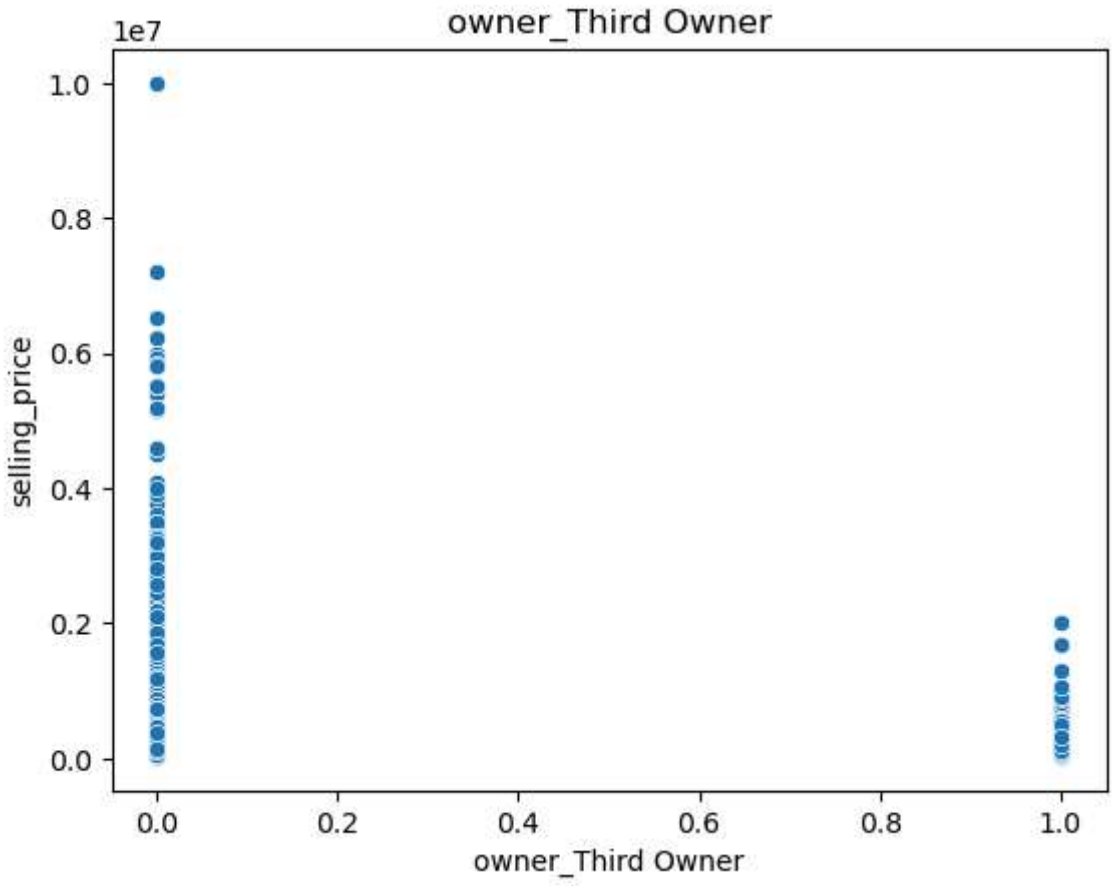

seller_type_Dealer
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
seller_type_Dealer


seller_type_Individual
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
seller_type_Individual


owner_Fourth & Above Owner
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
owner_Fourth & Above Owner


owner_Second Owner
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
owner_Second Owner


owner_Test Drive Car
1e7
1.0
0.8
0.6
0.4
0.2
0.0
selling_price
0.0
0.2
0.4
0.6
0.8
1.0
owner_Test Drive Car



In [113…

```
cars_df
```

Out[113…

	year	km_driven	transmission	seats	mileage(kmpl)	engine(CC)	max_power(bhp)
0	2014	145500	1	5.0	23.40	1248.0	74.00
1	2014	120000	1	5.0	21.14	1498.0	103.52
2	2006	140000	1	5.0	17.70	1497.0	78.00
3	2006	140000	1	5.0	17.70	1497.0	78.00
4	2010	127000	1	5.0	23.00	1396.0	90.00
…	…	…	…	…	…	…	…
7407	2013	50000	1	5.0	18.90	998.0	67.10
7408	2014	80000	1	5.0	22.54	1396.0	88.73
7409	2013	110000	1	5.0	18.50	1197.0	82.85
7410	2007	119000	1	5.0	16.80	1493.0	110.00
7411	2009	120000	1	5.0	19.30	1248.0	73.90

7412 rows × 17 columns

In [114…
```
from scipy.stats import skew
for i in cars_df:
    print(i,"--->",skew(cars_df[i]))
```

```
year ---> -0.7461174623555729
km_driven ---> 0.6784934306557338
transmission ---> -3.0793520946026725
seats ---> 1.9894636002537804
mileage(kmpl) ---> -0.02841761678291445
engine(CC) ---> 0.41984853633131897
max_power(bhp) ---> 0.7211438728578453
selling_price ---> 5.6620513032462
fuel_Diesel ---> -0.17279458878736642
fuel_LPG ---> 13.858491433776791
fuel_Petrol ---> 0.22537868673879413
seller_type_Dealer ---> 2.80910903007586
seller_type_Individual ---> -2.736339044366389
owner_Fourth & Above Owner ---> 6.353632406937305
owner_Second Owner ---> 0.9426313363533518
owner_Test Drive Car ---> 38.46297797931313
owner_Third Owner ---> 3.2565908977168614
```

In [115…
```
cars_df["seats"]=np.log(cars_df["seats"])
```

In [116…
```
skew(cars_df["seats"])
```

Out[116… 1.5955771452180667

In [117…
```
cars_df["selling_price"]=np.log(cars_df["selling_price"])
```

In [118…
```
skew(cars_df["selling_price"])
```

Out[118… -0.16025239145848685

In [119…
```
x=cars_df.drop("selling_price",axis=1)
y=cars_df["selling_price"]
```

In [120…
```
from sklearn.preprocessing import StandardScaler
```

In [121…
```
SS=StandardScaler()
```

In [122…
```
for features in x:
    x[features]=SS.fit_transform(x[[features]])
```

In [123…
```
from sklearn.model_selection import train_test_split
```

In [124…
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=3)
```

In [125…
```
from sklearn.linear_model import LinearRegression
```

In [126…
```
lr=LinearRegression()
```

In [127…
```
model=lr.fit(x_train,y_train)
```

In [128…
```
y_predict=model.predict(x_test)
y_predict
```

Out[128…
```
array([12.96052547, 13.11780698, 12.97053987, ..., 13.31099445,
       10.96576274, 12.71306938])
```

In [129…
```
residual=y_test-y_predict
residual
```

Out[129…
```
1119   -0.194837
513     0.282188
2683    0.366935
6045    0.065745
1409   -0.219459
          ...
2103   -0.335188
945     0.204966
7355   -0.387084
5064   -0.369128
3066    0.186150
Name: selling_price, Length: 2224, dtype: float64
```

In [130…
```
model.intercept_
```

Out[130… 12.860938202573392

In [131…
```
model.coef_
```

Out[131…
```
array([ 0.42873913, -0.03066301, -0.08065135,  0.04924791,  0.0255519 ,
        0.09685216,  0.25432804,  0.10445275,  0.01071395,  0.0094515 ,
        0.00125968, -0.02950426, -0.01406452, -0.03354937,  0.01650622,
       -0.03000076])
```

In [132…
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [133…
```
y_pred=model.predict(x_train)

mse=mean_squared_error(y_train,y_pred)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_train,y_pred)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_train,y_pred)
print("r2_score",r_squared_value)
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_train.shape[0]-1)) / (x_train.shape[0] - x_trai
print("adj_r2",adj_r2)
```

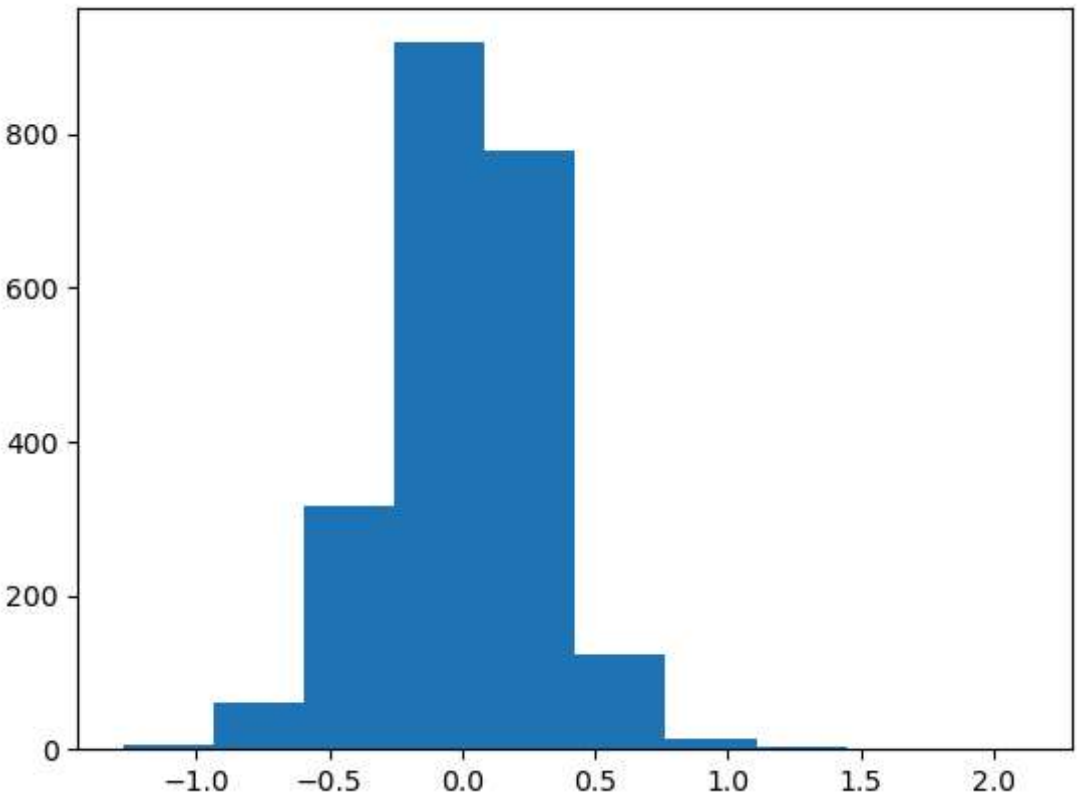
```
mean_squared_error 0.09608462982187943
**************************************************
rmse 0.3099752083987999
**************************************************
mean_absolute_error 0.23664104801643782
**************************************************
r2_score 0.8322251215402573
**************************************************
adj_r2 0.8317059960219135
```

In [134…
```
y_pred=model.predict(x_test)

mse=mean_squared_error(y_test,y_pred)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_test,y_pred)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_test,y_pred)
print("r2_score",r_squared_value)
```

```
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_test.shape[0]-1)) / (x_test.shape[0] - x_test.s
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.09308639156388164
**************************************************
rmse 0.30510062530889975
**************************************************
mean_absolute_error 0.23242717203455002
**************************************************
r2_score 0.8361574486133551
**************************************************
adj_r2 0.8349696457940591
```

In [135…
```
#normalized residual
plt.hist(residual)
plt.show()
```



In [136…
```
model.score(x_train,y_train)
```

Out[136… 0.8322251215402573

In [137…
```
model.score(x_test,y_test)
```

Out[137… 0.8361574486133551

In [138…
```
from sklearn.tree import DecisionTreeRegressor, plot_tree
```

In [139…
```
DTC=DecisionTreeRegressor(random_state=11)
```

In [140…
```
DTC.fit(x_train, y_train)
```

Out[140…
```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=11)
```

In [141…
```
y_pred_train=DTC.predict(x_train)

mse=mean_squared_error(y_train,y_pred_train)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_train,y_pred_train)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_train,y_pred_train)
print("r2_score",r_squared_value)
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_train.shape[0]-1)) / (x_train.shape[0] - x_trai
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.002150051744759462
**************************************************
rmse 0.04636865045221245
**************************************************
mean_absolute_error 0.01308662711197939
**************************************************
r2_score 0.9962457609419125
**************************************************
adj_r2 0.9962341446539742
```

In [142…
```
y_pred=DTC.predict(x_test)

mse=mean_squared_error(y_test,y_pred)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_test,y_pred)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_test,y_pred)
print("r2_score",r_squared_value)
print(50*"*")
```

```
adj_r2=1-(((1-r_squared_value) * (x_test.shape[0]-1)) / (x_test.shape[0] - x_test.s
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.0960045504265881
**************************************************
rmse 0.30984601082890856
**************************************************
mean_absolute_error 0.21449366399824402
**************************************************
r2_score 0.8310211597811764
**************************************************
adj_r2 0.8297961206133009
```

In [143…
```
DTC.score(x_train,y_train)
```

Out[143…
```
0.9962457609419125
```

In [144…
```
DTC.score(x_test,y_test)
```

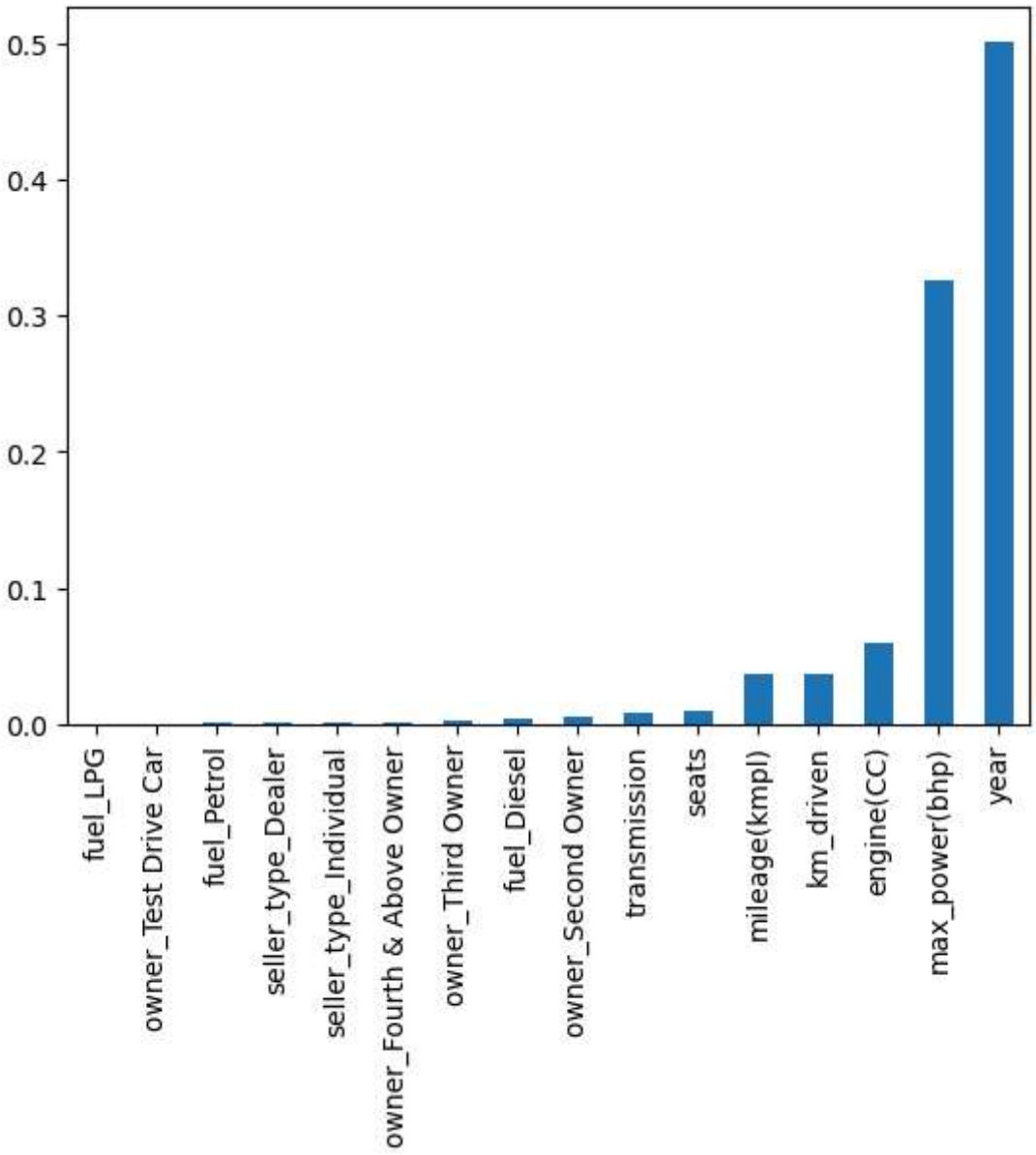
Out[144…
```
0.8310211597811764
```

In [145…
```
array=DTC.feature_importances_
array
```

Out[145…
```
array([5.01359386e-01, 3.74025633e-02, 8.94503272e-03, 9.84604634e-03,
       3.73191701e-02, 6.03568982e-02, 3.25661670e-01, 4.61139065e-03,
       3.24945707e-04, 1.17620317e-03, 1.23847165e-03, 1.24688725e-03,
       1.54349583e-03, 6.23279960e-03, 5.07523452e-04, 2.22751597e-03])
```

In [146…
```
s1=pd.Series(array,index=x.columns)
s1.sort_values().plot(kind="bar")
```

Out[146…
```
<Axes: >
```



In [147…
```
from sklearn.model_selection import GridSearchCV
```

In [294…
```
hyperparameter={"criterion":["squared_error"],
                "max_depth":np.arange(3,9),
                "min_samples_split":np.arange(2,20),
               "min_samples_leaf":np.arange(2,15)}
```

In [296…
```
gscv=GridSearchCV(DTC,hyperparameter,cv=5)
```

In [298…
```
gscv.fit(x_train,y_train)
```

Out[298…
GridSearchCV
estimator: DecisionTreeRegressor
DecisionTreeRegressor

In [300…
```
gscv.best_estimator_
```

Out[300…
```
DecisionTreeRegressor(max_depth=8, min_samples_leaf=7, min_samples_split=16,
                      random_state=11)
```

In [302…
```
DT_REG = gscv.best_estimator_
DT_REG.fit(x_train, y_train)
```

Out[302…
```
DecisionTreeRegressor(max_depth=8, min_samples_leaf=7, min_samples_split=16,
                      random_state=11)
```

In [304…
```
y_pred_train=DT_REG.predict(x_train)

mse=mean_squared_error(y_train,y_pred_train)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_train,y_pred_train)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_train,y_pred_train)
print("r2_score",r_squared_value)
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_train.shape[0]-1)) / (x_train.shape[0] - x_trai
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.057553353249635175
**************************************************
rmse 0.23990279958690597
**************************************************
mean_absolute_error 0.17769891283170988
**************************************************
r2_score 0.8995051876214919
**************************************************
adj_r2 0.8991942386758226
```

In [306…
```
y_pred=DT_REG.predict(x_test)

mse=mean_squared_error(y_test,y_pred)
print("mean_squared_error",mse)
```

```
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_test,y_pred)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_test,y_pred)
print("r2_score",r_squared_value)
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_test.shape[0]-1)) / (x_test.shape[0] - x_test.s
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.07076923181002191
**************************************************
rmse 0.26602487066066194
**************************************************
mean_absolute_error 0.18741380622772555
**************************************************
r2_score 0.875438167656658
**************************************************
adj_r2 0.8745351367017449
```

In [308…
```
DT_REG.score(x_train,y_train)
```

Out[308…
```
0.8995051876214919
```

In [310…
```
DT_REG.score(x_test,y_test)
```

Out[310…
```
0.875438167656658
```

In [312…
```
from sklearn.ensemble import RandomForestRegressor
```

In [314…
```
RF=RandomForestRegressor()
```

In [316…
```
RF.fit(x_train,y_train)
```

Out[316…
```
RandomForestRegressor
RandomForestRegressor()
```

In [317…
```
y_pred_train=RF.predict(x_train)

mse=mean_squared_error(y_train,y_pred_train)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")
```

```
mae=mean_absolute_error(y_train,y_pred_train)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_train,y_pred_train)
print("r2_score",r_squared_value)
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_train.shape[0]-1)) / (x_train.shape[0] - x_trai
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.00940512882785531
**************************************************
rmse 0.09698004345150249
**************************************************
mean_absolute_error 0.06876844102061556
**************************************************
r2_score 0.9835775571085945
**************************************************
adj_r2 0.9835267431294294
```

In [318…

```
y_pred=RF.predict(x_test)

mse=mean_squared_error(y_test,y_pred)
print("mean_squared_error",mse)
print(50*"*")

rmse=np.sqrt(mse)
print("rmse",rmse)
print(50*"*")

mae=mean_absolute_error(y_test,y_pred)
print("mean_absolute_error",mae)
print(50*"*")

r_squared_value=r2_score(y_test,y_pred)
print("r2_score",r_squared_value)
print(50*"*")

adj_r2=1-(((1-r_squared_value) * (x_test.shape[0]-1)) / (x_test.shape[0] - x_test.s
print("adj_r2",adj_r2)
```

```
mean_squared_error 0.055596546225683725
**************************************************
rmse 0.2357891987044439
**************************************************
mean_absolute_error 0.1666716823589742
**************************************************
r2_score 0.9021438061045652
**************************************************
adj_r2 0.9014343819530805
```

In [320…

```
RF.score(x_train,y_train)
```

Out[320…

```
0.9835775571085945
```

In [324…
```
RF.score(x_test,y_test)
```

Out[324…
```
0.9021438061045652
```

In [ ]:

In [ ]:

In [ ]: