In [1]:

```
# Import Libraries for the project
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import numpy as np
from collections import Counter
%matplotlib inline
sns.set()
from subprocess import check_output
import warnings
warnings.filterwarnings("ignore")
from scipy.stats import skew, kurtosis
```

In [2]:

```
#Load dataset and see the shape, from below, we have 9576 & 10 columns as regards to our data shape.
df = pd.read_excel(r"C:\Users\HENRY OKEOMA\Downloads\Car_Sales.xlsx")
df
```

Out[2]:

	car	price	body	mileage	engV	engType	registration	year	model	drive
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Class	rear
2	Mercedes-Benz	35000.0	other	135	5.5	Petrol	yes	2008	CL 550	rear
3	Mercedes-Benz	17800.0	van	162	1.8	Diesel	yes	2012	B 180	front
4	Mercedes-Benz	33000.0	vagon	91	NaN	Other	yes	2013	E-Class	NaN
...	...	...	...	...	...	...	...	...	...	...
9571	Hyundai	14500.0	crossover	140	2.0	Gas	yes	2011	Tucson	front
9572	Volkswagen	2200.0	vagon	150	1.6	Petrol	yes	1986	Passat B2	front
9573	Mercedes-Benz	18500.0	crossover	180	3.5	Petrol	yes	2008	ML 350	full
9574	Lexus	16999.0	sedan	150	3.5	Gas	yes	2008	ES 350	front
9575	Audi	22500.0	other	71	3.6	Petrol	yes	2007	Q7	full

9576 rows × 10 columns

In [3]:

```
# Check data for more understanding data types
df.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
```

we have some missing values in our dataset, hence the data will need further cleaning, now we shall check for duplicates.

In [4]:

```
# Check for duplicates
df.duplicated().sum()
```

Out[4]:

```
113
```

We have about 113 duplicated values in our dataset.

In [5]:

```
# Chech the Numerical variables of the dataset and convert them to integers
df.describe().astype(int)
```

Out[5]:

	price	mileage	engV	year
count	9576	9576	9142	9576
mean	15633	138	2	2006
std	24106	98	5	7
min	0	0	0	1953
25%	4999	70	1	2004
50%	9200	128	2	2008
75%	16700	194	2	2012
max	547800	999	99	2016

In [6]:

```
# Missing Value Counts to understand the extent of the missing values
df.isna().sum()
```

Out[6]:

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

In [7]:

```
# Total sum of the missing values
df.isna().sum().sum()
```

Out[7]:

```
945
```

In [25]:

```
pip install -U pandas-profiling
```

```
Collecting pandas-profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
     -------------------------------------- 324.4/324.4 kB 5.1 MB/s eta 0:00:00
Collecting ydata-profiling
  Downloading ydata_profiling-4.1.2-py2.py3-none-any.whl (345 kB)
     -------------------------------------- 345.9/345.9 kB 5.4 MB/s eta 0:00:00
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\henry okeoma\anaconda3\lib\site-pack
ages (from ydata-profiling->pandas-profiling) (6.0)
Requirement already satisfied: requests<2.29,>=2.24.0 in c:\users\henry okeoma\anaconda3\lib\site-
packages (from ydata-profiling->pandas-profiling) (2.28.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\henry okeoma\anaconda3\lib\site-pac
kages (from ydata-profiling->pandas-profiling) (2.11.3)
Requirement already satisfied: numpy<1.24,>=1.16.0 in c:\users\henry okeoma\anaconda3\lib\site-pac
kages (from ydata-profiling->pandas-profiling) (1.21.5)
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in c:\users\henry okeoma\anaconda3\lib\site
-packages (from ydata-profiling->pandas-profiling) (1.4.4)
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 c:\users\henry okeoma\anaconda3\lib\si
te-packages (from ydata-profiling->pandas-profiling) (0.13.2)
Requirement already satisfied: tqdm<4.65,>=4.48.2 in c:\users\henry okeoma\anaconda3\lib\site-pack
ages (from ydata-profiling->pandas-profiling) (4.64.1)
Collecting typeguard<2.14,>=2.13.2
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting phik<0.13,>=0.11.1
  Downloading phik-0.12.3-cp39-cp39-win_amd64.whl (663 kB)
     -------------------------------------- 663.5/663.5 kB 6.0 MB/s eta 0:00:00
Requirement already satisfied: scipy<1.10,>=1.4.1 in c:\users\henry okeoma\anaconda3\lib\site-pack
ages (from ydata-profiling->pandas-profiling) (1.9.1)
Requirement already satisfied: seaborn<0.13,>=0.10.1 in c:\users\henry okeoma\anaconda3\lib\site-p
ackages (from ydata-profiling->pandas-profiling) (0.11.2)
Collecting pydantic<1.11,>=1.8.1
  Downloading pydantic-1.10.7-cp39-cp39-win_amd64.whl (2.2 MB)
     ---------------------------------------- 2.2/2.2 MB 7.7 MB/s eta 0:00:00
Requirement already satisfied: matplotlib<3.7,>=3.2 in c:\users\henry okeoma\anaconda3\lib\site-pa
ckages (from ydata-profiling->pandas-profiling) (3.5.2)
Collecting visions[type_image_path]==0.7.5
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
     -------------------------------------- 102.7/102.7 kB 6.2 MB/s eta 0:00:00
Collecting htmlmin==0.1.12
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting imagehash==4.3.1
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
     -------------------------------------- 296.5/296.5 kB 6.1 MB/s eta 0:00:00
Requirement already satisfied: pillow in c:\users\henry okeoma\anaconda3\lib\site-packages (from i
magehash==4.3.1->ydata-profiling->pandas-profiling) (9.2.0)
Requirement already satisfied: PyWavelets in c:\users\henry okeoma\anaconda3\lib\site-packages (fr
om imagehash==4.3.1->ydata-profiling->pandas-profiling) (1.3.0)
Requirement already satisfied: networkx>=2.4 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling) (2.8.4)
Requirement already satisfied: attrs>=19.3.0 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling) (21.4.0)
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 7.2 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\henry okeoma\anaconda3\lib\site-packag
es (from jinja2<3.2,>=2.11.1->ydata-profiling->pandas-profiling) (2.0.1)
Requirement already satisfied: cycler>=0.10 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\henry okeoma\anaconda3\lib\site-packa
ges (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (1.4.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\henry okeoma\anaconda3\lib\site-packa
ges (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (4.25.0)
Requirement already satisfied: packaging>=20.0 in c:\users\henry okeoma\anaconda3\lib\site-package
s (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\henry okeoma\anaconda3\lib\site-packag
es (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\henry okeoma\anaconda3\lib\site-pa
ckages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from pandas!=1.4.0,<1.6,>1.1->ydata-profiling->pandas-profiling) (2022.1)
Requirement already satisfied: joblib>=0.14.1 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from phik<0.13,>=0.11.1->ydata-profiling->pandas-profiling) (1.1.0)
```

In [3]:

```
from pandas_profiling import ProfileReport
```

```
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\henry okeoma\anaconda3\lib\sit
e-packages (from pydantic<1.11,>=1.8.1->ydata-profiling->pandas-profiling) (4.3.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (3.3)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\henry okeoma\anaconda3\lib\sit
e-packages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\henry okeoma\anaconda3\lib\site-pack
ages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (2022.9.14)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\henry okeoma\anaconda3\lib\site-p
ackages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (1.26.11)
Requirement already satisfied: patsy>=0.5.2 in c:\users\henry okeoma\anaconda3\lib\site-packages
(from statsmodels<0.14,>=0.13.2->ydata-profiling->pandas-profiling) (0.5.2)
Requirement already satisfied: colorama in c:\users\henry okeoma\anaconda3\lib\site-packages (from
tqdm<4.65,>=4.48.2->ydata-profiling->pandas-profiling) (0.4.5)
Requirement already satisfied: six in c:\users\henry okeoma\anaconda3\lib\site-packages (from pats
y>=0.5.2->statsmodels<0.14,>=0.13.2->ydata-profiling->pandas-profiling) (1.16.0)
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py): started
  Building wheel for htmlmin (setup.py): finished with status 'done'
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27082 sha256=bc335f59
fa469b8e0723d443c569bfb852624a20236ef7d2cb5fad147135c7
  Stored in directory: c:\users\henry okeoma\appdata\local\pip\cache\wheels\1d\05\04\c6d7d3b6653
9e659ac6dfe81e2d6fd4c1a8316cc5a403300
Successfully built htmlmin
Installing collected packages: htmlmin, typeguard, tangled-up-in-unicode, pydantic, multimethod,
magehash, visions, phik, ydata-profiling, pandas-profiling
Successfully installed htmlmin-0.1.12 imagehash-4.3.1 multimethod-1.9.1 pandas-profiling-3.6.6 p
k-0.12.3 pydantic-1.10.7 tangled-up-in-unicode-0.2.0 typeguard-2.13.3 visions-0.7.5 ydata-profil
g-4.1.2
Note: you may need to restart the kernel to use updated packages.
```

In [4]:

```python
df_profile = ProfileReport(df, title='df of CarSales before cleaning')
df_profile
```

Summarize dataset: 100% 36/36 [00:04<00:00, 7.13it/s, Completed]

Generate report structure: 100% 1/1 [00:02<00:00, 2.66s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.07s/it]

Overview

Dataset statistics

Number of variables	10
Number of observations	9576
Missing cells	945
Missing cells (%)	1.0%
Duplicate rows	110
Duplicate rows (%)	1.1%
Total size in memory	748.2 KiB
Average record size in memory	80.0 B

Variable types

Categorical	4
Numeric	4
Boolean	1
Unsupported	1

Alerts

Dataset has 110 (1.1%) duplicate rows	**Duplicates**
car has a high cardinality: 87 distinct values	**High cardinality**

Out[4]:

In [10]:

```python
# DATA CLEANING.
# MAKE A COPY OF THE DATA FRAME
df1 = df.copy()
df1.head(3)
```

Out[10]:

	car	price	body	mileage	engV	engType	registration	year	model	drive
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Class	rear
2	Mercedes-Benz	35000.0	other	135	5.5	Petrol	yes	2008	CL 550	rear

In [11]:

```python
# We need to rename the features to more appropriate headers and easier to read
df.columns
```

Out[11]:

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

In [12]:

```python
df.columns = ['Car_Brands', 'Price', 'Body_type', 'Mileage', 'Engine_volume', 'Engine_type', 'Registration',
       'Year', 'Car_model', 'Drive_type']
df.head(2)
```

Out[12]:

	Car_Brands	Price	Body_type	Mileage	Engine_volume	Engine_type	Registration	Year	Car_model	Drive_type
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Class	rear

In [13]:

```python
# Now we need to handle the duplicates by dropping them (we recorded about 113 duplicates)
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

Out[13]:

```
0
```

In [14]:

```python
# Now we had some missing values on the EngV (now Engine_Volume) and the drive(now Drive_type).
# the former is a numerical variable while the later is categorical variable.
# We need to bring in the measures of central tendency to replace these values.
```

In [15]:

```python
# Find the Mode for the categorical valuable in Drive_type feature
df['Drive_type'].mode()
```

Out[15]:

```
0    front
Name: Drive_type, dtype: object
```

In [14]:

```
df.groupby('Drive_type').count()
```

Out[14]:

Drive_type	Car_Brands	Price	Body_type	Mileage	Engine_volume	Engine_type	Registration	Year	Car_model
front	5171	5171	5171	5171	4956	5171	5171	5171	5171
full	2422	2422	2422	2422	2366	2422	2422	2422	2422
rear	1360	1360	1360	1360	1305	1360	1360	1360	1360

In [16]:

```
# We need to fill the missing values on the Drive_type column with the mode = 'front'
df['Drive_type'] = df['Drive_type'].fillna('front')
```

In [18]:

```
df.info() ...# we can see that we have filed the Drive_type
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9463 entries, 0 to 9575
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Brands     9463 non-null   object 
 1   Price          9463 non-null   float64
 2   Body_type      9463 non-null   object 
 3   Mileage        9463 non-null   int64  
 4   Engine_volume  9029 non-null   float64
 5   Engine_type    9463 non-null   object 
 6   Registration   9463 non-null   object 
 7   Year           9463 non-null   int64  
 8   Car_model      9463 non-null   object 
 9   Drive_type     9463 non-null   object 
dtypes: float64(2), int64(2), object(6)
memory usage: 813.2+ KB
```

In [19]:

```
# We also need to fill the missing values of the Engine_volumes with the median.
#In this case we can also use the mean as they are the same. Using the median as a habit

df['Engine_volume'] = df.groupby(['Car_Brands','Body_type'])['Engine_volume'].transform(lambda x: x.fillna(x.med
```

In [20]:

```
# Check the final data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9463 entries, 0 to 9575
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Brands     9463 non-null   object 
 1   Price          9463 non-null   float64
 2   Body_type      9463 non-null   object 
 3   Mileage        9463 non-null   int64  
 4   Engine_volume  9453 non-null   float64
 5   Engine_type    9463 non-null   object 
 6   Registration   9463 non-null   object 
 7   Year           9463 non-null   int64  
 8   Car_model      9463 non-null   object 
 9   Drive_type     9463 non-null   object 
dtypes: float64(2), int64(2), object(6)
memory usage: 813.2+ KB
```

In [21]:

```
#
df.isna().sum() # we still have some missing values in our engine volumes, then we have to drop them
```

Out[21]:

```
Car_Brands       0
Price            0
Body_type        0
Mileage          0
Engine_volume   10
Engine_type      0
Registration     0
Year             0
Car_model        0
Drive_type       0
dtype: int64
```

In [22]:

```
# We now drop the values with the below
df.dropna(subset=['Engine_volume'],inplace=True)
df.isna().sum()
```

Out[22]:

```
Car_Brands       0
Price            0
Body_type        0
Mileage          0
Engine_volume    0
Engine_type      0
Registration     0
Year             0
Car_model        0
Drive_type       0
dtype: int64
```

In [23]:

```
# Dropping Entrie with Price <=0
df.Price[df.Price == 0].count()
```

Out[23]:

```
238
```

In [24]:

```
df = df.drop(df[df.Price <= 0].index)
df.Price[df.Price == 0].count()
```

Out[24]:

```
0
```

We have succeeded in cleaning our data, we have removed duplicates, we have replaced missing values, we have also dropped the zeros and our data is clean for analysis and visualisations, but before we do that, we can have a post profile after cleaning.

In [25]:

```
df_profile2 = ProfileReport(df, title='df_of_CarSales_After cleaning')
df_profile2
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[25]:

In [26]:

```
df.head(2)
```

Out[26]:

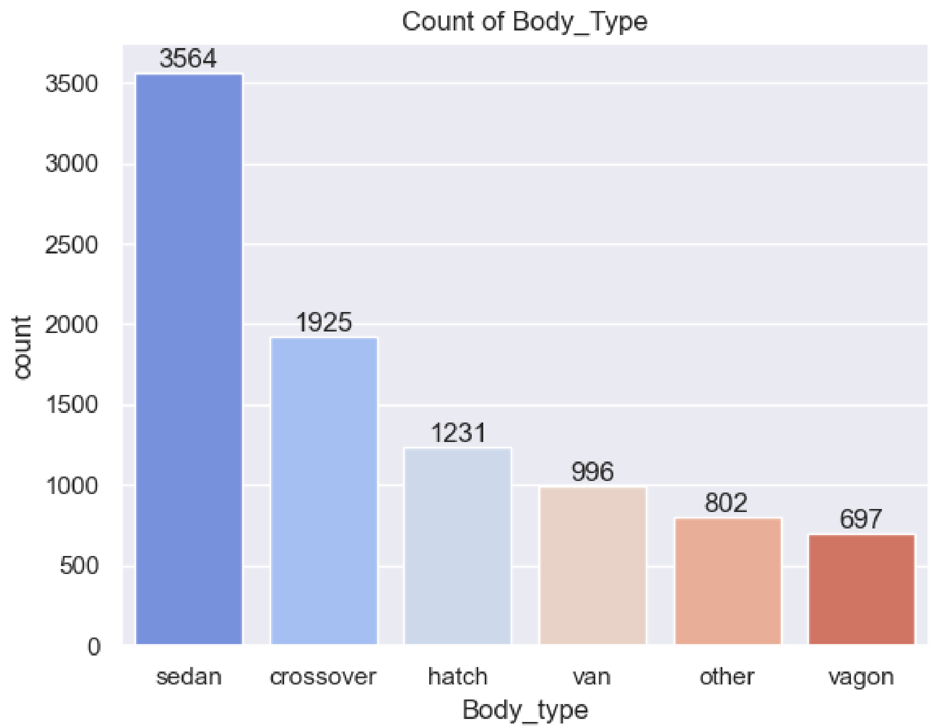
	Car_Brands	Price	Body_type	Mileage	Engine_volume	Engine_type	Registration	Year	Car_model	Drive_type
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Class	rear

In [ ]:

```
# Questions for  Analysis
#1. Which type of cars are sold maximum?
#2. What is the correlation between price and mileage?
#3. How many cars are registered?
#4. Does the registration status influence car price?
#5. What is the car price distribution based on Engine Value?
#6. Which car type has the highest pricing?
```

In [37]:

```
#1. Which type of cars are sold maximum?
ax = sns.countplot(x=df["Body_type"], order = df["Body_type"].value_counts(ascending=False).index, palette='coolw
values = df["Body_type"].value_counts(ascending=False).values
ax.bar_label(container = ax.containers[0], labels=values)
plt.title("Count of Body_Type");
```



Insight: We can see that the sedan body_type of cars sold the maximum from the dataset

In [39]:

```
2. # Relationship between price and mileage
plt.title('Relationship between price and mileage')
sns.scatterplot(y='Price', x='Mileage', data=df);
```

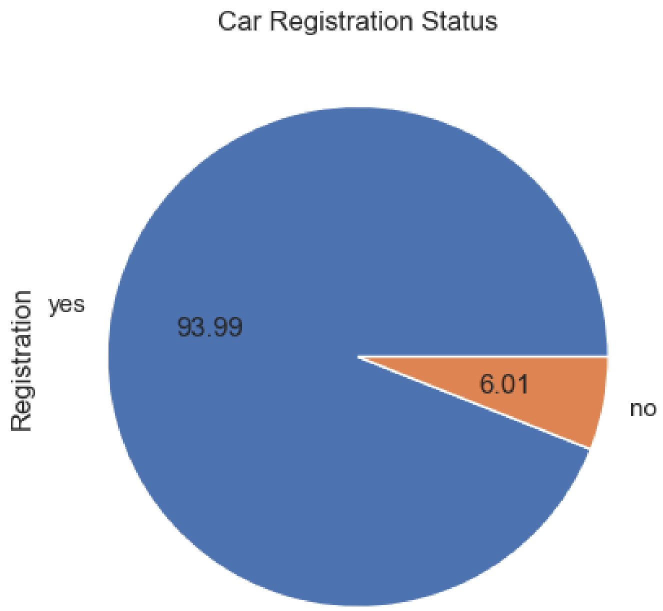

```
Insights:
```

```
There is a negative correlation between price and mileage. Car of lower mileage, have a higher price and vise
versa. Most of the Mileages are between 0 and  400 with price range of 0 - 150,000.

Also we can see some outliers in the plot
```

In [41]:

```
#3. How many cars are registered?
plt.figure(figsize =(10,5))
df['Registration'].value_counts(normalize=True).plot.pie(autopct="%.2f")
plt.title("Car Registration Status");
```



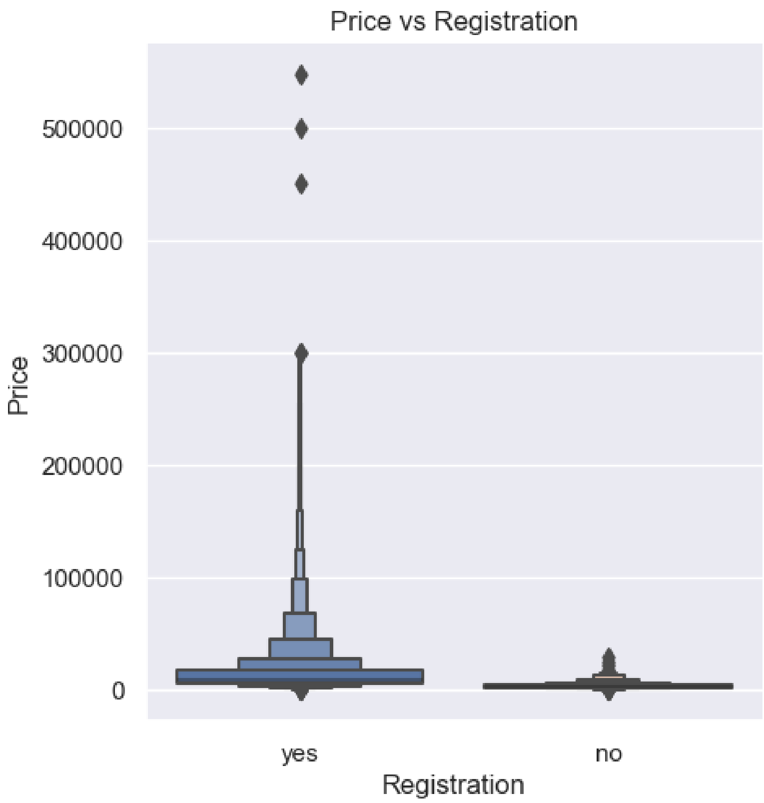
Insight:

Most of the cars are registered with a number of 93.99%, while just 6% is not registered

In [62]:

```
#4. Does the registration status influence car price?

sns.catplot(y='Price', x='Registration', data=df, kind='boxen')
plt.title('Price vs Registration')
plt.show()
```

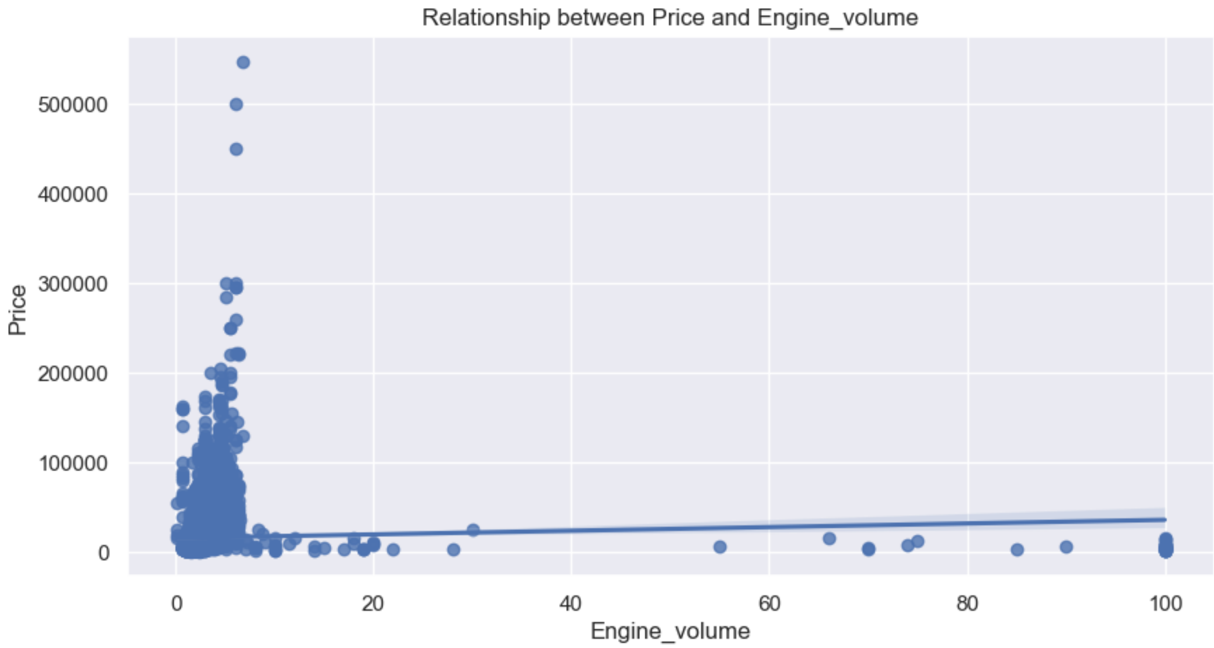



```
Insight:
The prices for non registered cars are very much lower than registered cars
```

In [64]:

```
# 5. What is the car price distribution based on Engine Value?

plt.figure(figsize=(10,5))
sns.regplot(x='Engine_volume',y='Price',data=df)
plt.title("Relationship between Price and Engine_volume");
```



Insight: Very weak positive correlation between the price/engine_volume. Most of the enine_volume are clustered around 1 & 5 engine volume with prices below 150,000

In [77]:

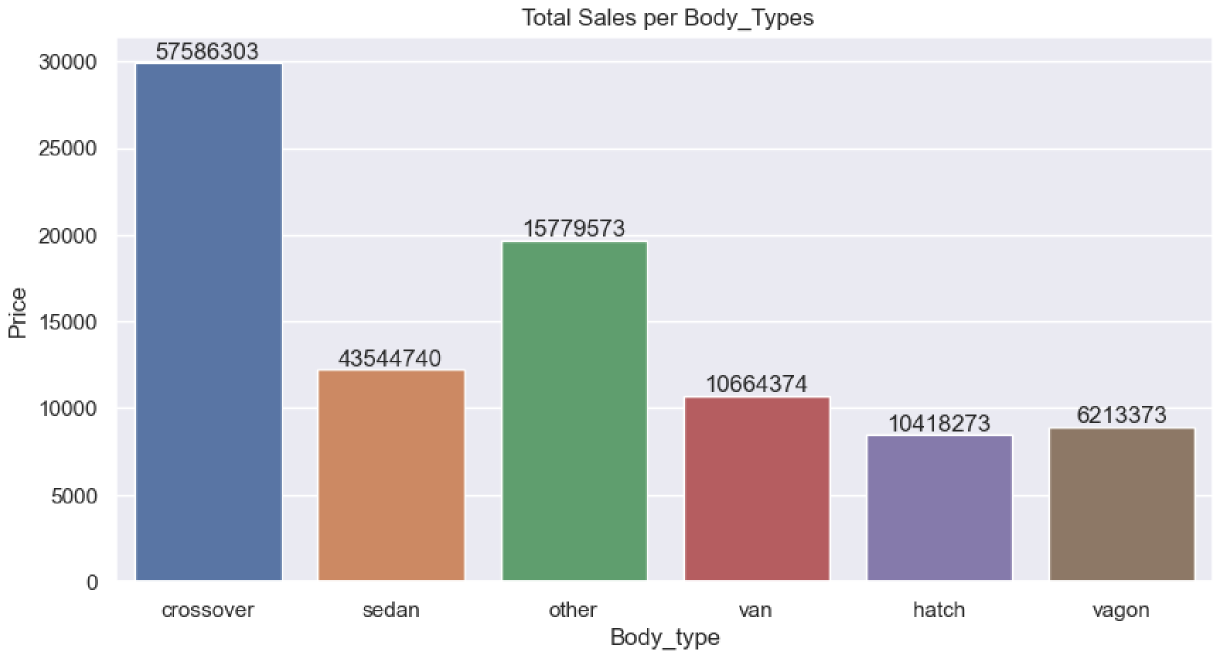
```
#6. Which car type has the highest pricing (Body_type)?
# We shall group the body_type and sum the prices of each
bodytype_p = df.groupby('Body_type')['Price'].sum().astype(int).sort_values(ascending=False)
bodytype_p
```

Out[77]:

```
Body_type
crossover    57586303
sedan        43544740
other        15779573
van          10664374
hatch        10418273
vagon         6213373
Name: Price, dtype: int32
```

In [84]:

```
# We shall use a simple barchart to visualise the Data
plt.figure(figsize=(10,5))
ax = sns.barplot(x='Body_type',y='Price', data=df, ci=None, order=bodytype_p.index)
values = bodytype_p.values
ax.bar_label(container = ax.containers[0], labels=values)
plt.title(" Total Sales per Body_Types")
plt.show();
```



Insight: The Crossover car types generated the highest income, although as we saw before, Sedan cars are sold the most.

Irrespective of the fact that Car type sedan was sold most (per count), crossover generated highest income (per price)

In [40]:

```
df.head(2)
```

Out[40]:

	Car_Brands	Price	Body_type	Mileage	Engine_volume	Engine_type	Registration	Year	Car_model	Drive_type
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Class	rear

In [30]:

```
# Car Brand with the highest price
C_brand_max = df.loc[df['Price'] == df['Price'].max(), ['Car_Brands','Price']]
C_brand_max
```

Out[30]:

	Car_Brands	Price
7621	Bentley	547800.0

```
Insight: Bentley is the Car with the highest Price
```

In [31]:

```
# Car Brand with the least price
C_brand_min = df.loc[df['Price'] == df['Price'].min(), ['Car_Brands','Price']]
C_brand_min
```

Out[31]:

	Car_Brands	Price
5010	GAZ	259.35

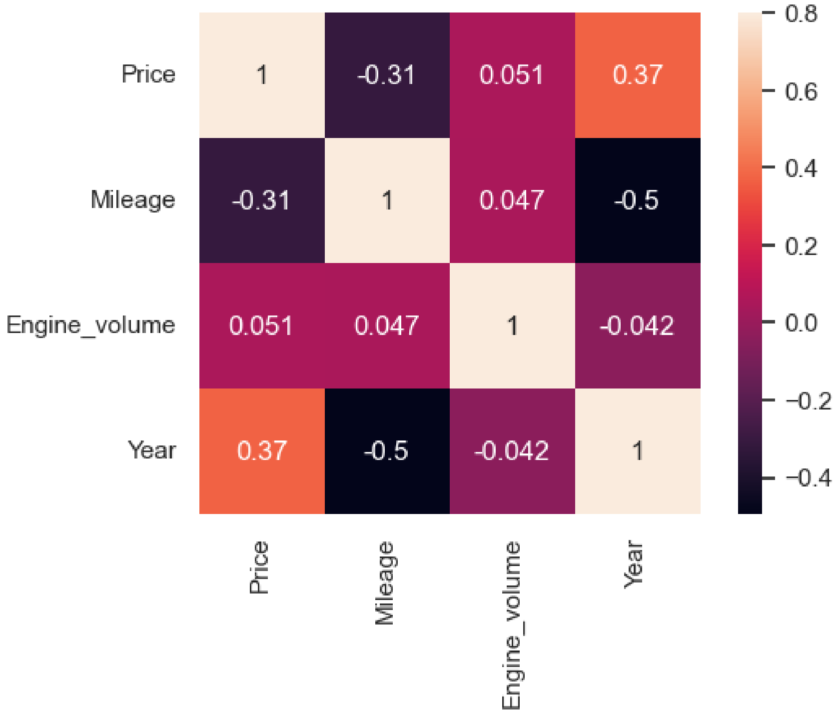
In [ ]:

```
Insight: Gaz is the car with the least price
```

In [54]:

```
# Correlation on the numeric variable

a = df.corr()
plt.figure(figsize=(6,4))
sns.heatmap(a, vmax=.8, square=True, annot=True)
plt.show()
```




```
From the above plot, we have a negative correlation between Price and Mileage, as one is increasing the other
is decreasing. Also we have a weak postive correlation between price and year.
```

```
Summary and Insights
1. Sedan car types are sold the maximum.
2. There is a negative correlation between price and mileage
3. Most of the cars are registered with a number of 93.99%, while just 6% is not registered.
4. The prices for non registered cars are very much lower than registered cars
5. Very weak positive correlation between the price/engine_volume. Most of the enine_volume are clustered
around 1 & 5 engine volume with prices below 150,000
6. The Crossover car types generated the highest income, although as we saw before, Sedan cars are sold the
most.Irrespective of the fact that Car type sedan was sold most (per count), crossover generated highest income
(per price)
```