Name : J SAI KUMAR

ASSIGNMENTS - ST

Assignment 1 was on SQL.

Assignment - 2

2.1. Write a Python program to create a person class. Include attributes like name, country and date of birth. Implement a method to determine the person's age

In [1]:
```
#importing datatime module/package to retrive date
import datetime

#creating a class called Person
class Person:
    def __init__(self,name,country,dob):
        self.name = name
        self.country = country
        self.dob = dob

    def age_calculation(self):
        today = datetime.date.today()
        age = today.year - self.dob.year - ((today.month,today.day) < (self.dob.month
        return age
```

In [2]:
```
sai = Person("Sai","India",datetime.date(2002,9,29))
print(sai.name, "has" , sai.age_calculation(), "years")
```

```
Sai has 21 years
```

In [5]:
```
Ashok = Person("Ashok","India",datetime.date(1998,5,9))
print(Ashok.name, "has" , Ashok.age_calculation(), "years")
```

```
Ashok has 25 years
```

In [ ]:
```

```

2.2. Write a Python program to create a class that represents a shape. Include methods to calculate its area and perimeter. Implement subclasses for different shapes like circle, triangle, and square.

In [7]:
```python
#creating a class called Shape whih is a main class
class Shape:
    def __init__(self):
        pass

    def area(self):
        pass

    def perimeter(self):
        pass
```

In [8]:
```python
#creating subclasses for all the shapes

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14159 * self.radius



#traingle subclass
class Triangle(Shape):
    def __init__(self, side1, side2, side3):
        self.side1 = side1
        self.side2 = side2
        self.side3 = side3

    def area(self):
        s = (self.side1 + self.side2 + self.side3) / 2
        return (s * (s - self.side1) * (s - self.side2) * (s - self.side3)) ** 0.5

    def perimeter(self):
        return self.side1 + self.side2 + self.side3


#square subcllass
class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2

    def perimeter(self):
        return 4 * self.side
```

In [12]:
```
circle = Circle(5)
print("Circle Area:", circle.area())
print("Circle Perimeter:", circle.perimeter())
print()

triangle = Triangle(3, 4, 5)
print("Triangle Area:", triangle.area())
print("Triangle Perimeter:", triangle.perimeter())
print()

square = Square(4)
print("Square Area:", square.area())
print("Square Perimeter:", square.perimeter())
```

```
Circle Area: 78.53975
Circle Perimeter: 31.4159

Triangle Area: 6.0
Triangle Perimeter: 12

Square Area: 16
Square Perimeter: 16
```

In [ ]:
```
```

Assignemnt - 4

4.1. To Analyze the data set and identify the factors that effect the cost of the car.

In [13]:
```
#loading the data
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
```

There are Six steps of Data Analysis Process which includes :

1. Define the problem 2. Collect the Data 3. Data Cleaning 4. Analyzing the Data 5. Data Visualization 6. Presenting Data

1. The defination of problem : To Analyze the dataset and identify the factors that effect the cost of the car

2. Collection of Data :

In [14]:
```
#loading the data
df = pd.read_excel("CarPrice_Assignment.xlsx")
```

In [15]:
```
df
```

Out[15]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocat
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	fr
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	fr
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	fr
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	fr
4	5	2	audi 100ls	gas	std	four	sedan	4wd	fr
...	...	...	...	...	...	...	...	...	
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	fr
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	fr
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	fr
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	fr
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	fr

205 rows × 26 columns

In [16]:
```
#showing the data upto 5 first rows
df.head()
```

Out[16]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocatior
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	fron
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	fron
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	fron
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	fron
4	5	2	audi 100ls	gas	std	four	sedan	4wd	fron

5 rows × 26 columns

In [17]:
```
#showing the last 5 rows of the data
df.tail()
```

Out[17]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front

5 rows × 26 columns

In [18]:
```
df.columns  #columns/attributes of the data
```

Out[18]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [19]:
```
df.shape

#represent the number of records and columns respectively
```

Out[19]:
```
(205, 26)
```

In [20]:
```
df.info()
```

```
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

3. Data Cleaning/ Data Preprocessing :

In [21]:
```
#returns the minimum stats of the data which has numberical/quantitative records

df.describe()
```

Out[21]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000

In [22]:
```
df.describe(include = "object")
#returns the minimum stats of the attributes in the data which has qualitative datatyp
```

Out[22]:

	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	enginetype	cyli
count	205	205	205	205	205	205	205	205	
unique	147	2	2	2	5	3	2	7	
top	toyota corona	gas	std	four	sedan	fwd	front	ohc	
freq	6	185	168	115	96	120	202	148	

In [23]:
```
df.describe(include = "all")
#returns the minimum stats of the all type of attributes in the data
```

Out[23]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginel
count	205.000000	205.000000	205	205	205	205	205	205	
unique	NaN	NaN	147	2	2	2	5	3	
top	NaN	NaN	toyota corona	gas	std	four	sedan	fwd	
freq	NaN	NaN	6	185	168	115	96	120	
mean	103.000000	0.834146	NaN	NaN	NaN	NaN	NaN	NaN	
std	59.322565	1.245307	NaN	NaN	NaN	NaN	NaN	NaN	
min	1.000000	-2.000000	NaN	NaN	NaN	NaN	NaN	NaN	
25%	52.000000	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	
50%	103.000000	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	
75%	154.000000	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	
max	205.000000	3.000000	NaN	NaN	NaN	NaN	NaN	NaN	

11 rows × 26 columns

In [24]:
```
df.isnull().sum()
```

Out[24]:
```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

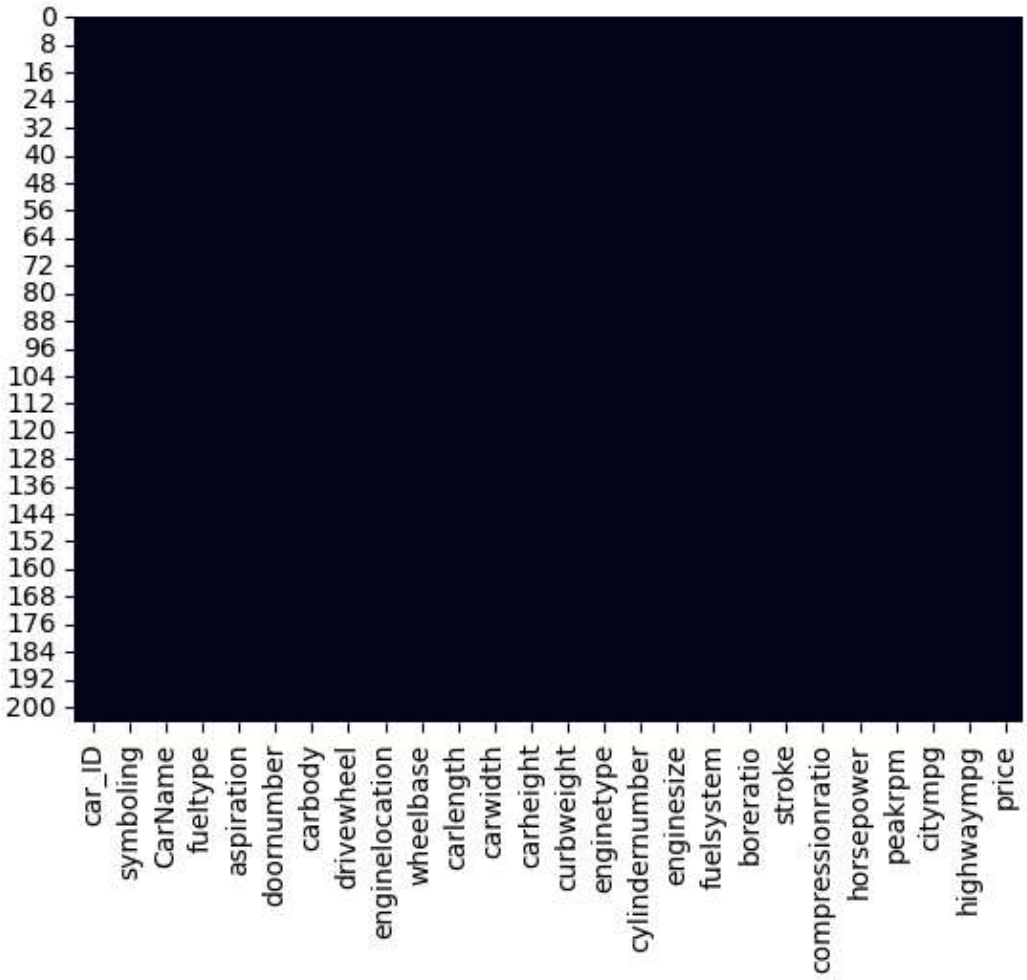
In [25]:
```
#lets represent the null values in a better manner

#get the count of null values
missing_values = df.isnull().sum()

#total misssing values
total = df.isnull().sum().sort_values(ascending = False)
#sort the values in descending order

#representing in percentage format for the null values
percent = ((df.isnull().sum() / df.shape[0])*100)

percent = percent.sort_values(ascending = False)
#concatenate the total missing values

missing_data = pd.concat([total,percent], axis = 1,
                         keys = ['Total Missing Values','Percentage of missing Values'

#adding the data types to better the represent

missing_data['Data(Dtypes)'] = df[missing_data.index].dtypes


#finally viewing the missing values

missing_data
```

Out[25]:

	Total Missing Values	Percentage of missing Values	Data(Dtypes)
car_ID	0	0.0	int64
symboling	0	0.0	int64
highwaympg	0	0.0	int64
citympg	0	0.0	int64
peakrpm	0	0.0	int64
horsepower	0	0.0	int64
compressionratio	0	0.0	float64
stroke	0	0.0	float64
boreratio	0	0.0	float64
fuelsystem	0	0.0	object
enginesize	0	0.0	int64
cylindernumber	0	0.0	object
enginetype	0	0.0	object
curbweight	0	0.0	int64
carheight	0	0.0	float64
carwidth	0	0.0	float64
carlength	0	0.0	float64
wheelbase	0	0.0	float64
enginelocation	0	0.0	object
drivewheel	0	0.0	object
carbody	0	0.0	object
doornumber	0	0.0	object
aspiration	0	0.0	object
fueltype	0	0.0	object
CarName	0	0.0	object
price	0	0.0	float64

In [26]:
```
import seaborn as sns
sns.heatmap(df.isnull(),cbar = False)


#This represents there are No Null values / missing values in the given dataset
```

Out[26]: <AxesSubplot:>



In [27]:
```
#checking for duplicate values

df.duplicated()
```

Out[27]:
```
0      False
1      False
2      False
3      False
4      False
       ...
200    False
201    False
202    False
203    False
204    False
Length: 205, dtype: bool
```

In [28]:
```
df.duplicated().value_counts()

#hence there are no duplicate values in the dataset
```

Out[28]:
```
False    205
dtype: int64
```

4. Analyzing the Data:

In [29]:
```
df.columns
```

Out[29]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [30]:
```
#lets analysize the fueltype attribute in thorough manner

df['fueltype'].values
```

Out[30]:
```
array(['gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'diesel', 'gas', 'gas', 'diesel', 'diesel', 'diesel', 'diesel',
       'diesel', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'diesel', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'diesel', 'gas', 'diesel', 'gas', 'diesel', 'gas',
       'diesel', 'gas', 'diesel', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'diesel', 'diesel', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'diesel',
       'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'diesel', 'gas',
       'diesel', 'gas', 'gas', 'diesel', 'gas', 'gas', 'gas', 'gas',
       'diesel', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas', 'gas',
       'gas', 'gas', 'diesel', 'gas'], dtype=object)
```

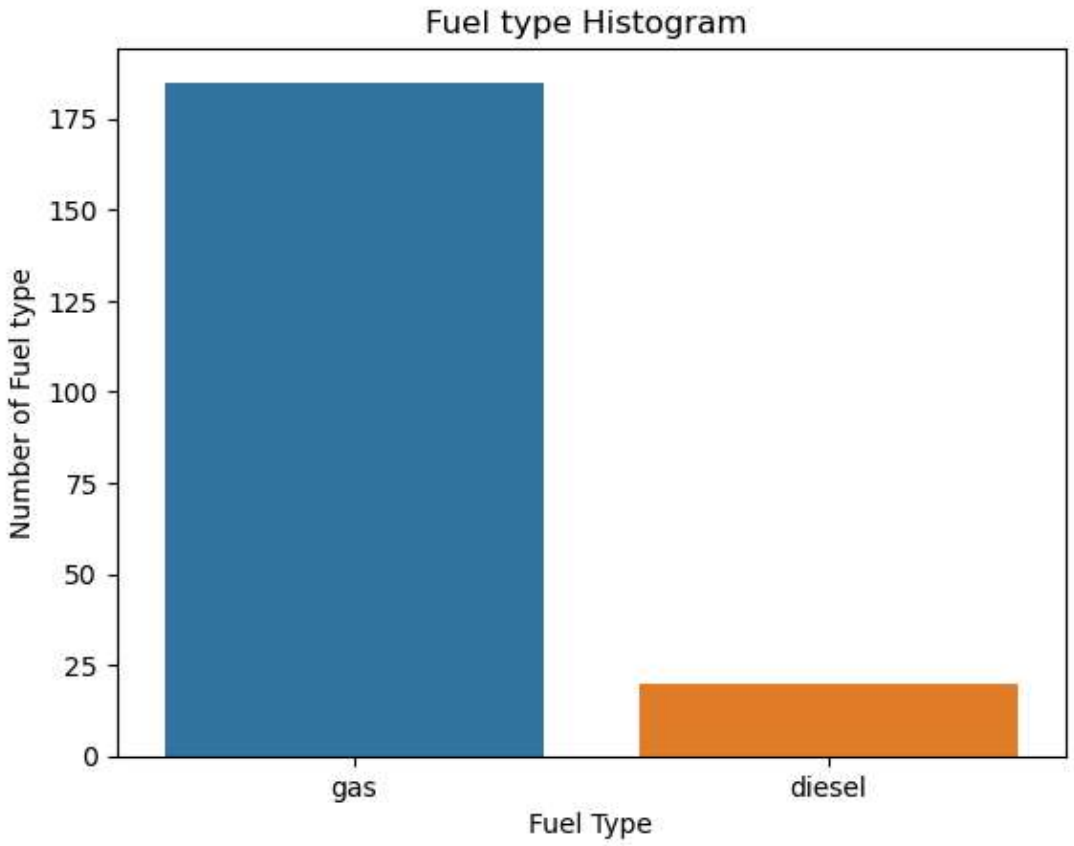
In [35]:
```
df['fueltype'].value_counts()
```

Out[35]:
```
gas       185
diesel     20
Name: fueltype, dtype: int64
```

5. Data Visualization

In [41]:
```
import seaborn as sns
sns.countplot(x = 'fueltype', data = df)
plt.title("Fuel type Histogram ")
plt.xlabel("Fuel Type")
plt.ylabel("Number of Fuel type")
```

Out[41]:
```
Text(0, 0.5, 'Number of Fuel type')
```



In [43]:
```
# Splitting company name from CarName column
CompanyName = df['CarName'].apply(lambda x : x.split(' ')[0])
df.insert(3,"Company car",CompanyName)
df.drop(['CarName'],axis=1,inplace=True)
df = df.rename(columns = {'Company car':'Company'})
df.head()
```

Out[43]:

	car_ID	symboling	Company	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	alfa-romero	gas	std	two	convertible	rwd	front
1	2	3	alfa-romero	gas	std	two	convertible	rwd	front
2	3	1	alfa-romero	gas	std	two	hatchback	rwd	front
3	4	2	audi	gas	std	four	sedan	fwd	front
4	5	2	audi	gas	std	four	sedan	4wd	front

5 rows × 26 columns

In [44]:

```
df_company = pd.DataFrame(df['Company'].unique(),columns = ["company"])
df_company
```

Out[44]:

	company
0	alfa-romero
1	audi
2	bmw
3	chevrolet
4	dodge
5	honda
6	isuzu
7	jaguar
8	maxda
9	mazda
10	buick
11	mercury
12	mitsubishi
13	Nissan
14	nissan
15	peugeot
16	plymouth
17	porsche
18	porcshce
19	renault
20	saab
21	subaru
22	toyota
23	toyouta
24	vokswagen
25	volkswagen
26	vw
27	volvo

As you see, some company have wrong name -"Maxda" and "Mazda" -"Nissan" and "nissan" -"porsche" and "porcshce" -"toyota" and "toyouta" -"vokswagen" and "volkswagen"

In [45]:
```
#change some wrong spelling word
for company in df['Company']:
    df["Company"]=df["Company"].str.replace("maxda", "mazda")
    df["Company"]=df["Company"].str.replace("Nissan", "nissan")
    df["Company"]=df["Company"].str.replace("porcshce", "porsche")
    df["Company"]=df["Company"].str.replace("toyouta", "toyota")
    df["Company"]=df["Company"].str.replace("vokswagen", "volkswagen")
    df["Company"]=df["Company"].str.replace("vw", "volkswagen")
df["Company"].unique()
```

Out[45]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

In [46]:
```
df.head()
```

Out[46]:

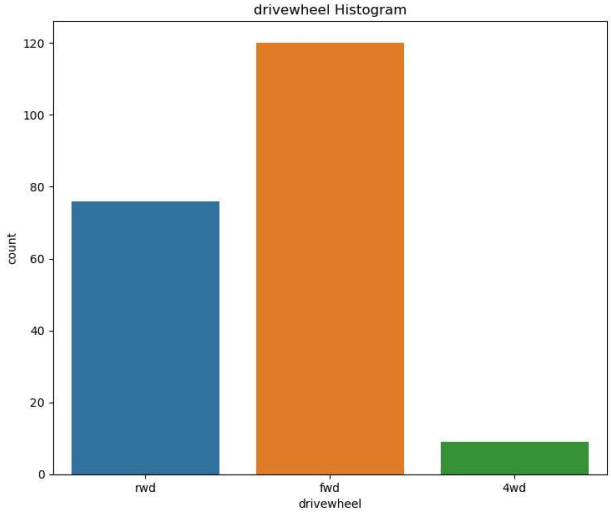
	car_ID	symboling	Company	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	alfa-romero	gas	std	two	convertible	rwd	front
1	2	3	alfa-romero	gas	std	two	convertible	rwd	front
2	3	1	alfa-romero	gas	std	two	hatchback	rwd	front
3	4	2	audi	gas	std	four	sedan	fwd	front
4	5	2	audi	gas	std	four	sedan	4wd	front

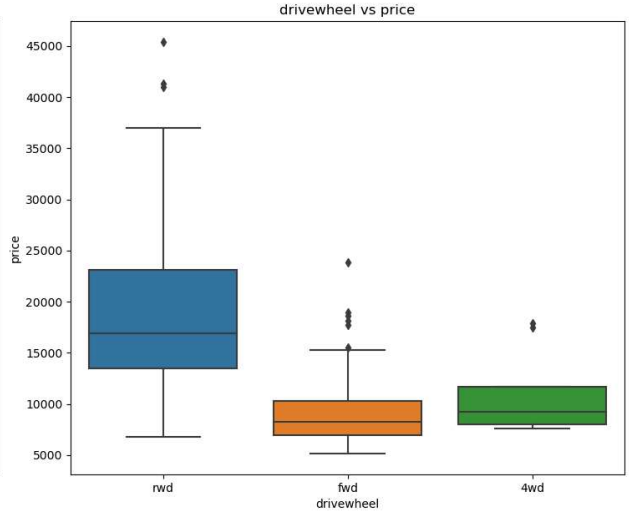
5 rows × 26 columns

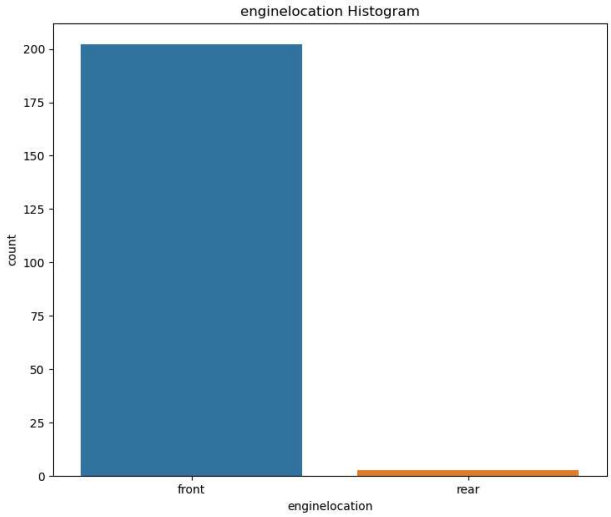
In [52]:
```
#lets see the fwd drive wheel and its range according to price as its lowest.
```

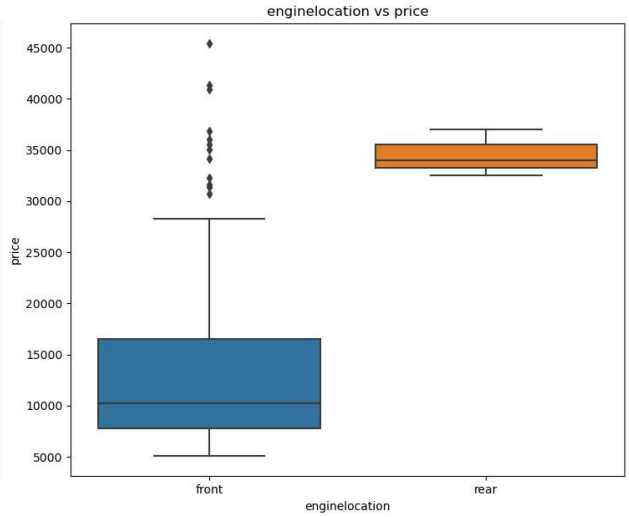
In [47]:

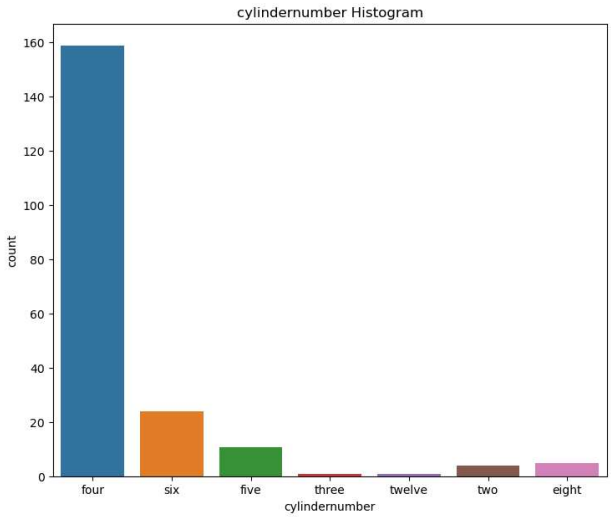
```
def fig_count(x, fig):
    plt.subplot(4,2,fig)
    plt.title(x + ' Histogram')
    sns.countplot(x = x,data = df )

    plt.subplot(4,2,fig +1)
    plt.title(x+ ' vs price')
    sns.boxplot(x = x, y = 'price',data = df)

plt.figure(figsize = (15,25))
list = ['drivewheel','enginelocation','cylindernumber','fuelsystem']
i = 1
for name in  list:
    fig_count(name,i)
    i = i +2
plt.tight_layout()
```

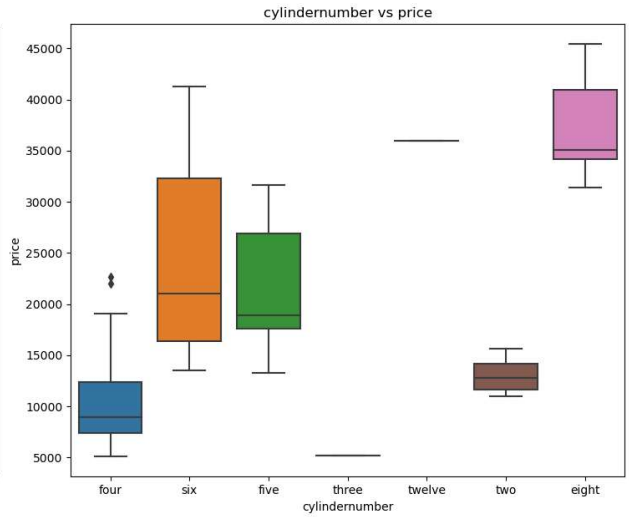

drivewheel Histogram
count
drivewheel
rwd
fwd
4wd

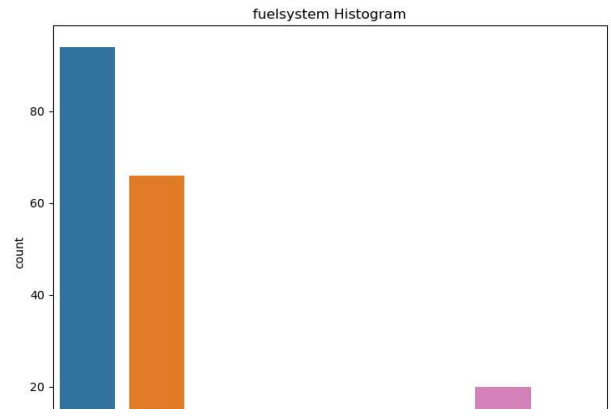

drivewheel vs price
price
drivewheel
rwd
fwd
4wd

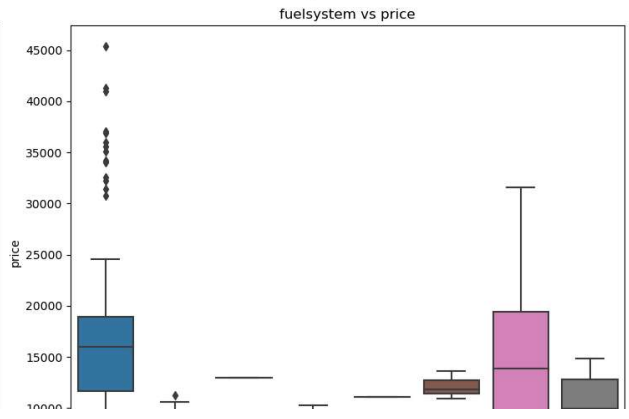

enginelocation Histogram
count
enginelocation
front
rear

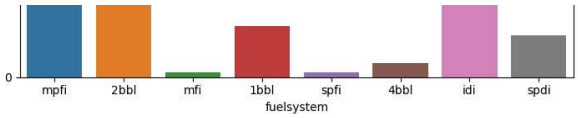

enginelocation vs price
price
enginelocation
front
rear

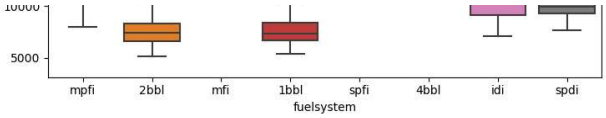

cylindernumber Histogram
count
cylindernumber
four
six
five
three
twelve
two
eight


cylindernumber vs price
price
cylindernumber
four
six
five
three
twelve
two
eight


fuelsystem Histogram
count


fuelsystem vs price
price



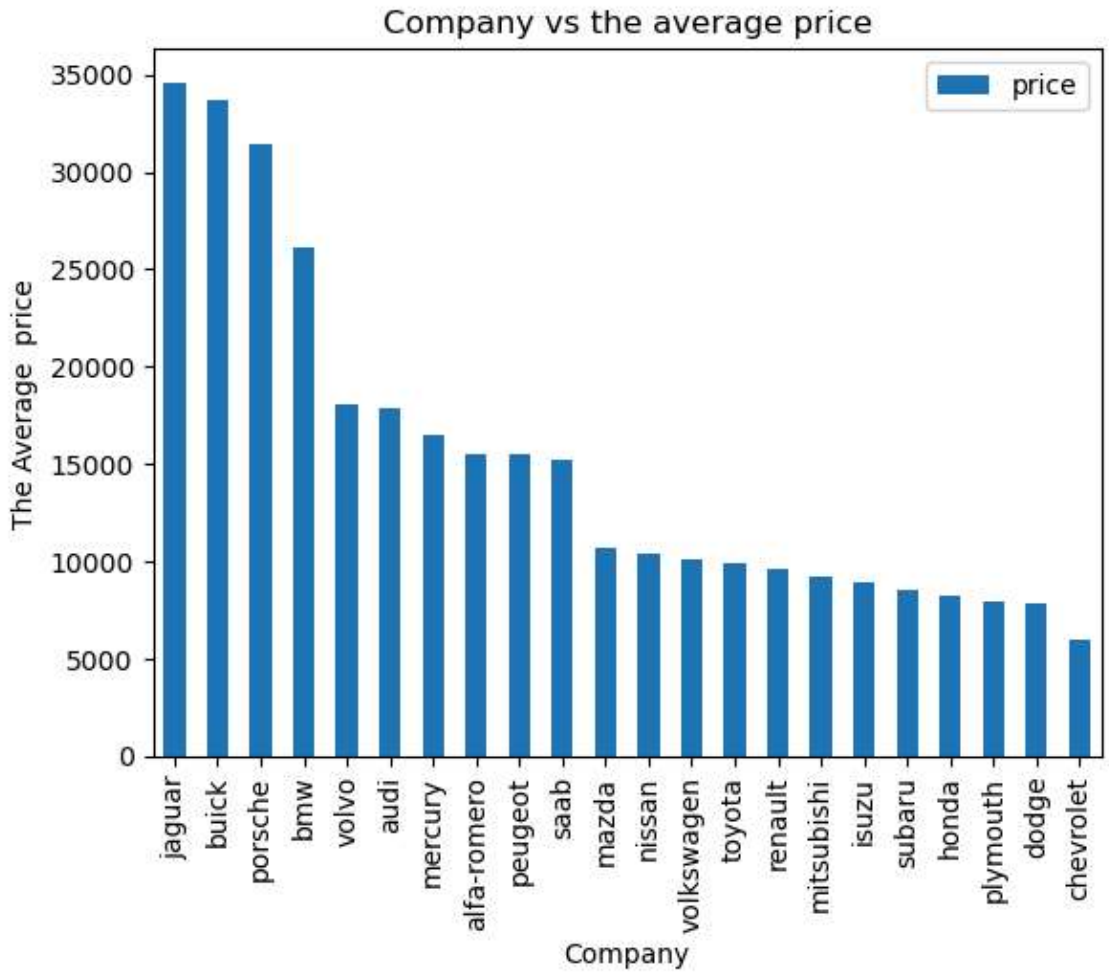


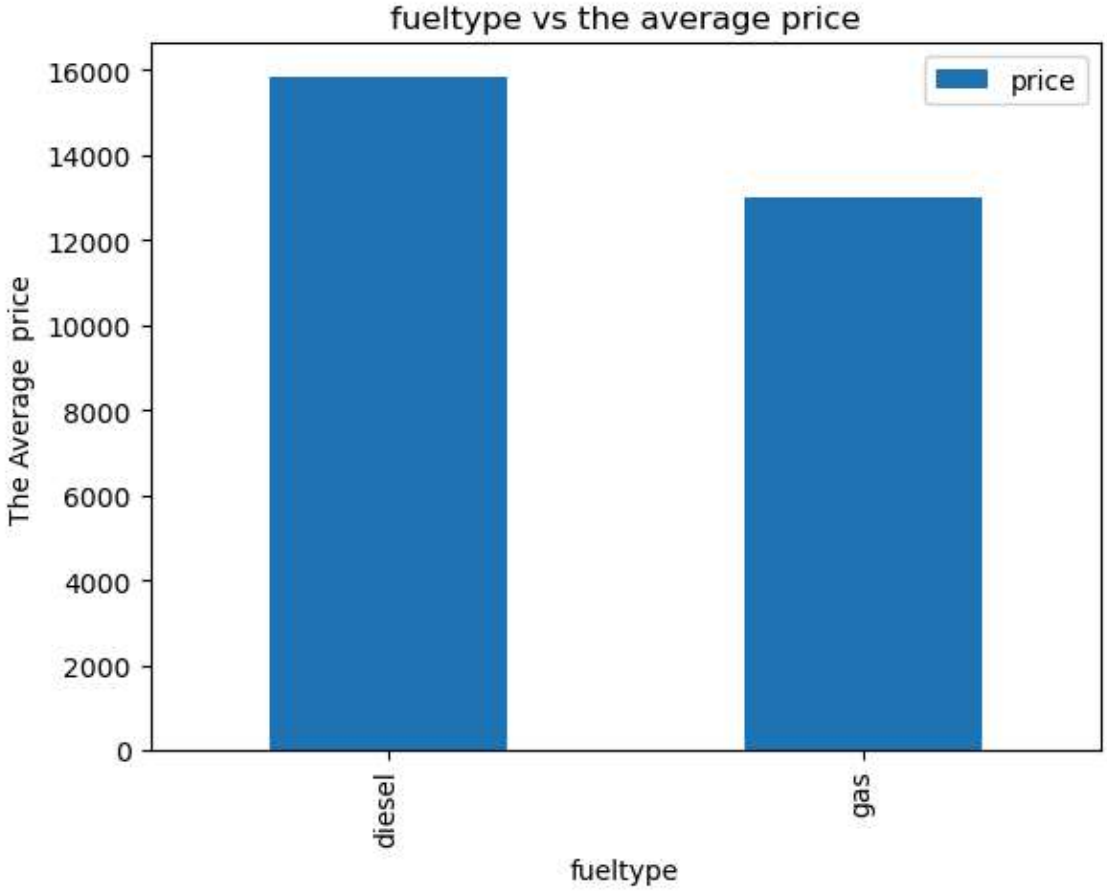
In [51]:
```
#les plot the average price as per the company
#average price vs the fuel type
```

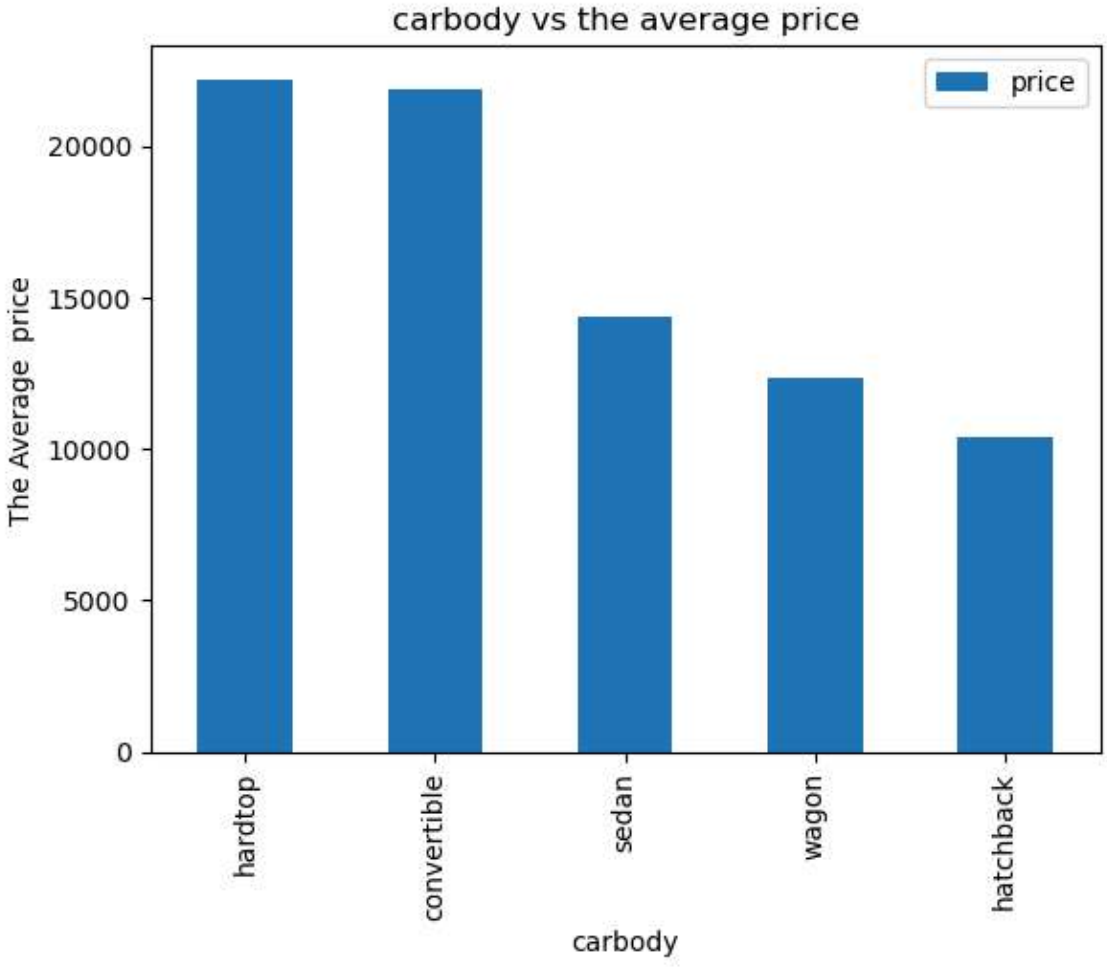
In [48]:

```
plt.figure(figsize=(20,6))
def plot_count(x):
    df2 = pd.DataFrame(df.groupby([x])['price'].mean().sort_values(ascending = False)
    df2.plot.bar()
    plt.title(x+' vs the average price')
    plt.ylabel("The Average  price")
    plt.show()
list = ['Company','fueltype','carbody']
for name in list:
    plot_count(name)

plt.tight_layout()
```

```
<Figure size 2000x600 with 0 Axes>
```



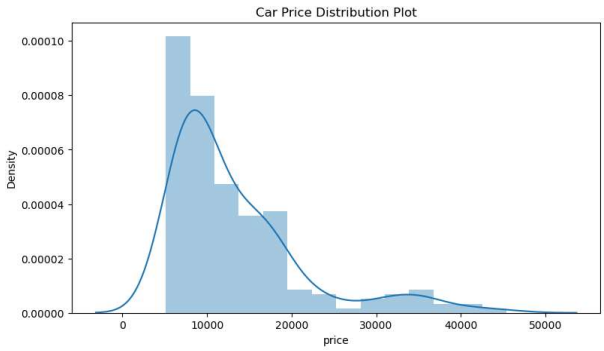

fueltype vs the average price
price
16000
14000
12000
10000
8000
6000
4000
2000
0
The Average price
diesel
gas
fueltype

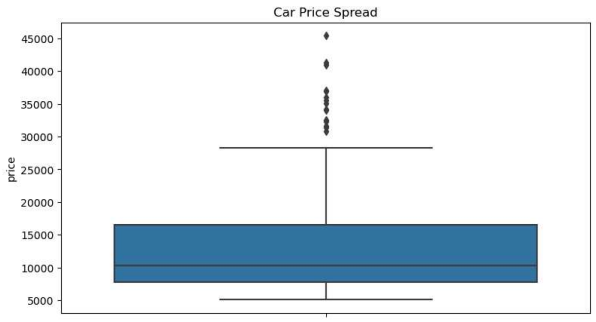

carbody vs the average price
price
20000
15000
10000
5000
0
The Average price
hardtop
convertible
sedan
wagon
hatchback
carbody

```
<Figure size 640x480 with 0 Axes>
```

In [49]:
```
#Box plot to see the distribution of price column


plt.figure(figsize=(20,5))

plt.subplot(1,2,1)
plt.title('Car Price Distribution Plot')
sns.distplot(df.price)

plt.subplot(1,2,2)
plt.title('Car Price Spread')
sns.boxplot(y=df.price)

plt.show()
```

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





In [50]:
```
df['price'].describe()
```

Out[50]:
```
count      205.000000
mean     13276.710571
std       7988.852332
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      16503.000000
max      45400.000000
Name: price, dtype: float64
```

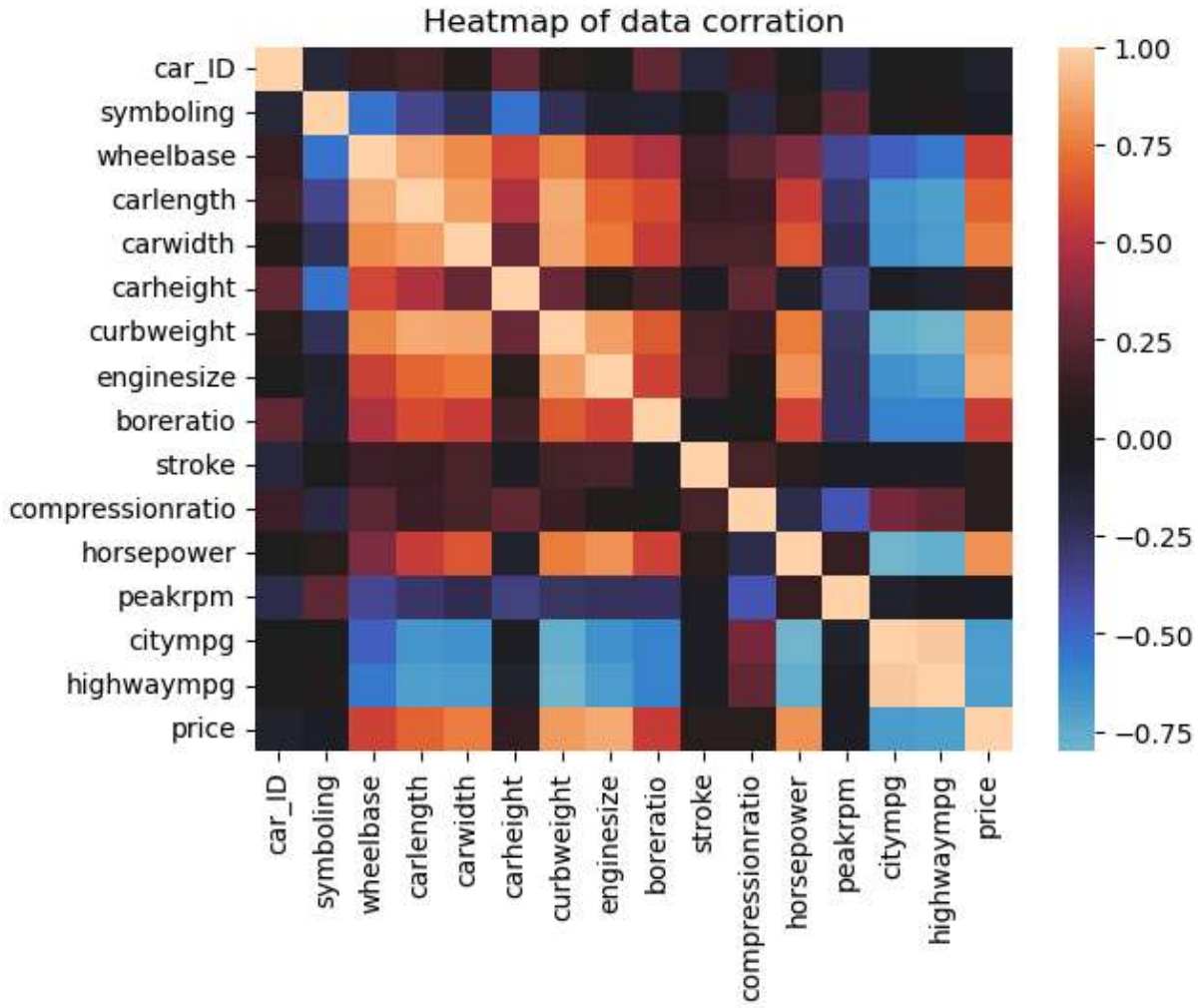
This represent the minimum,maximum prices of the cars

In [54]:
```
# display the corration
df_corr = df.corr()
df_corr
```

Out[54]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesiz
car_ID	1.000000	-0.151621	0.129729	0.170636	0.052387	0.255960	0.071962	-0.03393
symboling	-0.151621	1.000000	-0.531954	-0.357612	-0.232919	-0.541038	-0.227691	-0.10579
wheelbase	0.129729	-0.531954	1.000000	0.874587	0.795144	0.589435	0.776386	0.56932
carlength	0.170636	-0.357612	0.874587	1.000000	0.841118	0.491029	0.877728	0.68336
carwidth	0.052387	-0.232919	0.795144	0.841118	1.000000	0.279210	0.867032	0.73543
carheight	0.255960	-0.541038	0.589435	0.491029	0.279210	1.000000	0.295572	0.06714
curbweight	0.071962	-0.227691	0.776386	0.877728	0.867032	0.295572	1.000000	0.85059
enginesize	-0.033930	-0.105790	0.569329	0.683360	0.735433	0.067149	0.850594	1.00000
boreratio	0.260064	-0.130051	0.488750	0.606454	0.559150	0.171071	0.648480	0.58377
stroke	-0.160824	-0.008735	0.160959	0.129533	0.182942	-0.055307	0.168790	0.20312
compressionratio	0.150276	-0.178515	0.249786	0.158414	0.181129	0.261214	0.151362	0.02897
horsepower	-0.015006	0.070873	0.353294	0.552623	0.640732	-0.108802	0.750739	0.80976
peakrpm	-0.203789	0.273606	-0.360469	-0.287242	-0.220012	-0.320411	-0.266243	-0.24466
citympg	0.015940	-0.035823	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.65365
highwaympg	0.011255	0.034606	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.67747
price	-0.109093	-0.079978	0.577816	0.682920	0.759325	0.119336	0.835305	0.87414

In [89]:
```
#the heat map to uderstand the variation of data overall
plt.title("Heatmap of data corration")
sns.heatmap(df_corr,center=0)
plt.show()
```



6. REPORTING DATA-DRIVEN INSIGHTS

FINAL INSIGHTS :

1. The GAS fueltype are going in market as customers are willing to buy the Gas fueltype car
2. The price starts from 5118.000000 and have an range till 45400.000000 and the average car price is 13276.710571
3. The price of the Car totally depends on the Car Brand, Fueltype , Engine, Horsepower and Milege
4. Jaguar company cars are the most expensive cars in the dataset
5. Car prices of diesel fueltype are higher than gas fueltype cars

4.2. Develop a model which would predict the car prices by doing appropriate data analysis and data preparation.

In [58]:
```
df.head()
```

Out[58]:

	car_ID	symboling	Company	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	alfa-romero	gas	std	two	convertible	rwd	front
1	2	3	alfa-romero	gas	std	two	convertible	rwd	front
2	3	1	alfa-romero	gas	std	two	hatchback	rwd	front
3	4	2	audi	gas	std	four	sedan	fwd	front
4	5	2	audi	gas	std	four	sedan	4wd	front

5 rows × 26 columns

In [59]:
```
df.describe()
```

Out[59]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000

In [60]:
```
df.shape
```

Out[60]:
```
(205, 26)
```

In [62]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   Company           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [63]:
```
df.isnull().sum()
```

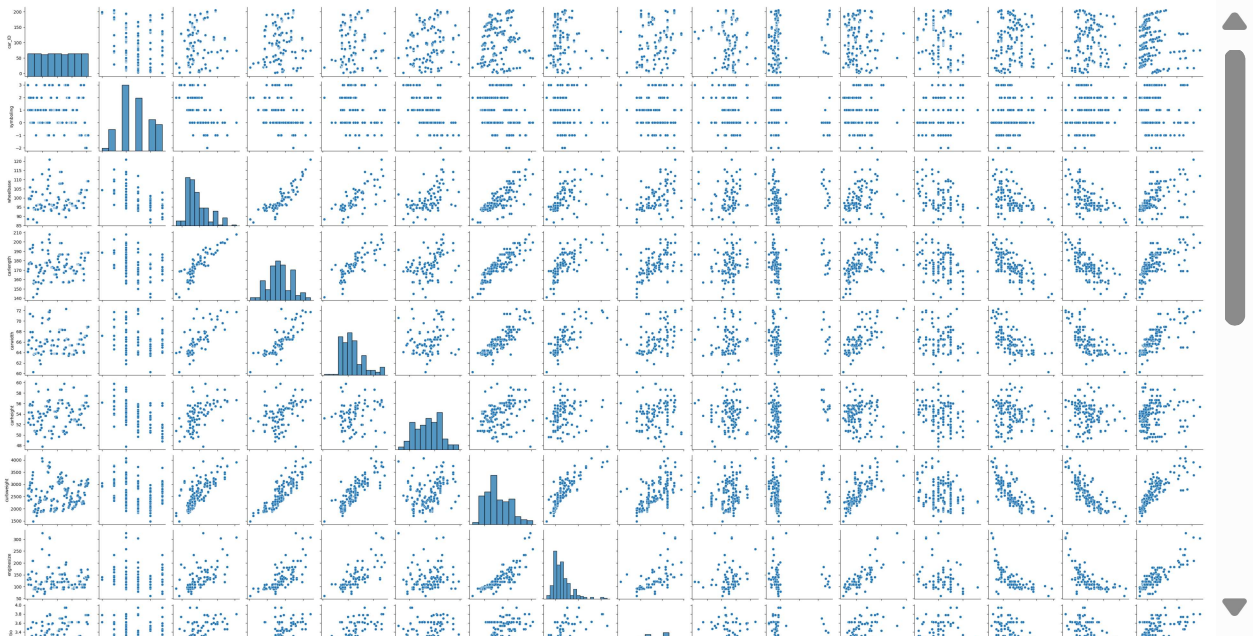
Out[63]:
```
car_ID              0
symboling           0
Company             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

In [64]:
```
print(df.dtypes)
```

```
car_ID              int64
symboling           int64
Company            object
fueltype           object
aspiration         object
doornumber         object
carbody            object
drivewheel         object
enginelocation     object
wheelbase         float64
carlength         float64
carwidth          float64
carheight         float64
curbweight          int64
enginetype         object
cylindernumber     object
enginesize          int64
fuelsystem         object
boreratio         float64
stroke            float64
compressionratio  float64
horsepower          int64
peakrpm             int64
citympg             int64
highwaympg          int64
price             float64
dtype: object
```

In [74]:
```
#importing libraries
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [75]:
```
#datavisualizaton part
sns.pairplot(df)
plt.show()
```



In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
corr_matrix = df.corr()
sns.heatmap(corr_matrix,annot= True)
plt.show()
```

In [77]:
```
#train test split method from sklearn

X = df.drop('price',axis = 1)
y = df['price']
X_train,X_test, y_train, y_test = train_test_split(X,y,test_size= 0.2,random_state = 
```

In [ ]:
```
#standardizeation features for train test

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
model = LinearRegression()
model.fit(X_train_scaled,y_train)
```

In [ ]:
```
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test,y_pred)
r2 = r2_score(y_test,y_pred)
print("MSE is " ,mse)
print("R-Squared : ",r2)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: