__Autombile Analysis___

Importing Basic Libraries Pandas,Numpy and Data file to start with

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

In [2]:
```
df=pd.read_csv(r"D:\Jenill\JT\RECR-BA\Ranjith Ape task\Automobile_data.csv")
df.head()
```

Out[2]:

	risk factor	losses	make	type of fuel	aspiration	total doors	body	wheels	location of engine	wheel base	...	size o engin
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	13
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	13
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	15
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	10
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	13

5 rows × 26 columns

EDA-Checking dataset for anamolies, disturbances and datatypes

In [3]:
```
df.shape
```

Out[3]:
```
(205, 26)
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
risk factor                 0
losses                      0
make                        0
type of fuel                0
aspiration                  0
total doors                 0
body                        0
wheels                      0
location of engine          0
wheel base                  0
length                      0
width                       0
height                      0
weight of curb              0
type of engine              0
total cylinders             0
size of engine              0
system of fuel              0
bore                        0
stroke                      0
ratio of comprehenssion     0
horsepower                  0
peak-rpm                    0
mpg in city                 0
mpg on highway              0
total price                 0
dtype: int64
```

In [5]:
```
df.describe()
```

Out[5]:

	risk factor	wheel base	length	width	height	weight of curb	size of engine	comp
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	

In [6]:
```
df.columns
```

Out[6]:
```
Index(['risk factor', 'losses', 'make', 'type of fuel', 'aspiration',
       'total doors', 'body', 'wheels', 'location of engine', 'wheel base',
       'length', 'width', 'height', 'weight of curb', 'type of engine',
       'total cylinders', 'size of engine', 'system of fuel', 'bore', 'stroke',
       'ratio of comprehenssion', 'horsepower', 'peak-rpm', 'mpg in city',
       'mpg on highway', 'total price'],
      dtype='object')
```

In [173…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 201 entries, 0 to 204
Data columns (total 26 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   risk_factor                 201 non-null    int64  
 1   losses                      201 non-null    float64
 2   make                        201 non-null    object 
 3   type_of_fuel                201 non-null    object 
 4   aspiration                  201 non-null    object 
 5   total_doors                 201 non-null    object 
 6   body                        201 non-null    object 
 7   wheels                      201 non-null    object 
 8   location_of_engine          201 non-null    object 
 9   wheel_base                  201 non-null    float64
 10  length                      201 non-null    float64
 11  width                       201 non-null    float64
 12  height                      201 non-null    float64
 13  weight_of_curb              201 non-null    int64  
 14  type_of_engine              201 non-null    object 
 15  total_cylinders             201 non-null    object 
 16  size_of_engine              201 non-null    int64  
 17  system_of_fuel              201 non-null    object 
 18  bore                        201 non-null    float64
 19  stroke                      201 non-null    float64
 20  ratio_of_comprehenssion     201 non-null    float64
 21  horsepower                  201 non-null    float64
 22  peak_rpm                    201 non-null    float64
 23  mpg_in_city                 201 non-null    int64  
 24  mpg_on_highway              201 non-null    int64  
 25  total_price                 201 non-null    float64
dtypes: float64(11), int64(5), object(10)
memory usage: 42.4+ KB
```

In [8]:

```
type(df.losses)
```

Out[8]:

```
pandas.core.series.Series
```

In [9]:

```
for i in df.columns:
    if df[i].dtypes=="O":
        a=df[i].value_counts()
        print(i)
        print(a)
```

```
losses
?      39
161    11
91      8
150     7
128     6
134     6
104     6
95      5
102     5
74      5
168     5
85      5
103     5
94      5
65      5
122     4
106     4
148     4
118     4
93      4
115     3
125     3
154     3
137     3
83      3
101     3
197     2
119     2
194     2
108     2
87      2
89      2
158     2
192     2
145     2
188     2
#       2
81      2
164     2
110     2
113     2
129     2
153     2
107     1
78      1
186     1
77      1
98      1
121     1
142     1
90      1
231     1
256     1
Name: losses, dtype: int64
make
toyota           32
nissan           18
mazda            17
mitsubishi       13
honda            13
volkswagen       12
subaru           12
peugot           11
```

```
volvo            11
dodge             9
mercedes-benz     8
bmw               8
audi              7
plymouth          7
saab              6
porsche           5
isuzu             4
jaguar            3
chevrolet         3
alfa-romero       3
renault           2
mercury           1
Name: make, dtype: int64
type of fuel
gas       185
diesel     20
Name: type of fuel, dtype: int64
aspiration
std      168
turbo     37
Name: aspiration, dtype: int64
total doors
four    114
two      89
?         2
Name: total doors, dtype: int64
body
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: body, dtype: int64
wheels
fwd    120
rwd     76
4wd      9
Name: wheels, dtype: int64
location of engine
front    202
rear       3
Name: location of engine, dtype: int64
type of engine
ohc      148
ohcf      15
ohcv      13
dohc      12
l         12
rotor      4
dohcv      1
Name: type of engine, dtype: int64
total cylinders
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: total cylinders, dtype: int64
system of fuel
mpfi    94
```

```
2bbl    66
idi     20
1bbl    11
spdi     9
4bbl     3
mfi      1
spfi     1
Name: system of fuel, dtype: int64
bore
3.62    23
3.19    20
3.15    15
3.03    12
2.97    12
3.46     9
3.31     8
3.78     8
3.43     8
3.27     7
2.91     7
3.39     6
3.54     6
3.05     6
3.58     6
3.7      5
3.01     5
3.35     4
?        4
3.17     3
3.59     3
3.74     3
3.47     2
3.94     2
3.24     2
3.13     2
3.63     2
3.5      2
3.8      2
3.33     2
2.54     1
3.08     1
3.61     1
3.34     1
3.6      1
2.92     1
3.76     1
2.68     1
2.99     1
Name: bore, dtype: int64
stroke
3.4     20
3.23    14
3.15    14
3.03    14
3.39    13
2.64    11
3.29     9
3.35     9
3.46     8
3.11     6
3.27     6
3.41     6
3.07     6
3.58     6
```

```
3.19     6
3.5      6
3.64     5
3.52     5
3.86     4
3.54     4
3.47     4
?        4
3.9      3
2.9      3
3.1      2
4.17     2
2.8      2
2.19     2
3.08     2
2.68     2
2.36     1
3.16     1
2.07     1
3.21     1
3.12     1
2.76     1
2.87     1
Name: stroke, dtype: int64
horsepower
68     19
70     11
69     10
116     9
110     8
95      7
88      6
62      6
101     6
160     6
114     6
84      5
97      5
102     5
145     5
82      5
76      5
111     4
92      4
123     4
86      4
90      3
73      3
85      3
207     3
182     3
121     3
152     3
112     2
56      2
161     2
156     2
94      2
52      2
?       2
162     2
155     2
184     2
100     2
```

```
176     2
55      1
262     1
134     1
115     1
140     1
48      1
58      1
60      1
78      1
135     1
200     1
64      1
120     1
72      1
154     1
288     1
143     1
142     1
175     1
106     1
Name: horsepower, dtype: int64
peak-rpm
5500    37
4800    36
5000    27
5200    23
5400    13
6000     9
5250     7
4500     7
5800     7
4200     5
4150     5
4750     4
4350     4
5100     3
4250     3
5900     3
4400     3
?        2
6600     2
4650     1
5600     1
5750     1
4900     1
5300     1
Name: peak-rpm, dtype: int64
total price
8921     2
18150    2
8845     2
8495     2
7775     2
        ..
45400    1
16503    1
5389     1
6189     1
22625    1
Name: total price, Length: 188, dtype: int64
```

Listed down the problems in the dataset

problems_list

"losses":"?",39

"losses":"#",2

"doors":"?",2

"bore":"?",4

"stroke":"?",4

"horsepower":"?",2

"peak-rpm":"?",2

Data rectification

In [10]:
```
df.losses.replace(to_replace="?", value="", inplace=True)
```

In [11]:
```
df.losses.replace(to_replace="#", value="", inplace=True)
```

In [12]:
```
df["total doors"].replace(to_replace="?", value="", inplace=True)
```

In [13]:
```
df.bore.replace(to_replace="?", value="", inplace=True)
```

In [14]:
```
df.stroke.replace(to_replace="?", value="", inplace=True)
```

In [15]:
```
df.horsepower.replace(to_replace="?", value="", inplace=True)
```

In [16]:
```
df["peak-rpm"].replace(to_replace="?", value="", inplace=True)
```

In [17]:
```
df.losses.value_counts()
```

Out[17]:

```
       41
161    11
91      8
150     7
134     6
128     6
104     6
85      5
94      5
65      5
102     5
74      5
168     5
103     5
95      5
106     4
93      4
118     4
148     4
122     4
83      3
125     3
154     3
115     3
137     3
101     3
119     2
87      2
89      2
192     2
197     2
158     2
81      2
188     2
194     2
153     2
129     2
108     2
110     2
164     2
145     2
113     2
256     1
107     1
90      1
231     1
142     1
121     1
78      1
98      1
186     1
77      1
Name: losses, dtype: int64
```

Changed data types weherever necessary

In [18]:

```
df.losses=pd.to_numeric(df.losses)
```

In [19]:

```
df.describe()
```

Out[19]:

	risk factor	losses	wheel base	length	width	height	weight of curb	s e
count	205.000000	164.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0
mean	0.834146	122.000000	98.756585	174.049268	65.907805	53.724878	2555.565854	126.9
std	1.245307	35.442168	6.021776	12.337289	2.145204	2.443522	520.680204	41.6
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.0
25%	0.000000	94.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.0
50%	1.000000	115.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.0
75%	2.000000	150.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.0
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.0

In [20]:
```
df.losses.fillna(df.losses.mean(),inplace=True)
```

In [21]:
```
df.losses.value_counts
```

Out[21]:
```
<bound method IndexOpsMixin.value_counts of 0      122.0
1      122.0
2      122.0
3      164.0
4      164.0
       ...
200     95.0
201     95.0
202     95.0
203     95.0
204     95.0
Name: losses, Length: 205, dtype: float64>
```

In [22]:
```
df["total doors"].replace(to_replace="",value="four",inplace=True)
```

In [23]:
```
df["total doors"].value_counts()
```

Out[23]:
```
four    116
two      89
Name: total doors, dtype: int64
```

In [24]:
```
df.bore=pd.to_numeric(df.bore)
```

In [25]:
```
df.bore.fillna(df.bore.mean(),inplace=True)
```

In [26]:
```
df.bore.value_counts
```

Out[26]:
```
<bound method IndexOpsMixin.value_counts of 0      3.47
1      3.47
2      2.68
3      3.19
4      3.19
       ...
200    3.78
201    3.78
202    3.58
203    3.01
204    3.78
Name: bore, Length: 205, dtype: float64>
```

In [27]:
```
df.stroke=pd.to_numeric(df.stroke)
```

In [28]:
```
df.stroke.fillna(df.stroke.mean(),inplace=True)
```

In [29]:
```
df.stroke.value_counts
```

Out[29]:
```
<bound method IndexOpsMixin.value_counts of 0      2.68
1      2.68
2      3.47
3      3.40
4      3.40
       ...
200    3.15
201    3.15
202    2.87
203    3.40
204    3.15
Name: stroke, Length: 205, dtype: float64>
```

In [30]:
```
df["peak-rpm"]=pd.to_numeric(df["peak-rpm"])
```

Some data has been filled with its "mode" value, some with "mean" values as per data distribution

In [31]:
```
df["peak-rpm"].fillna(df["peak-rpm"].mode(),inplace=True)
```

In [32]:
```
df["peak-rpm"].value_counts
```

Out[32]:
```
<bound method IndexOpsMixin.value_counts of 0      5000.0
1      5000.0
2      5000.0
3      5500.0
4      5500.0
        ...
200    5400.0
201    5300.0
202    5500.0
203    4800.0
204    5400.0
Name: peak-rpm, Length: 205, dtype: float64>
```

In [33]:
```
df.horsepower=pd.to_numeric(df.horsepower)
```

In [34]:
```
df.horsepower.fillna(df.horsepower.mean(),inplace=True)
```

In [35]:
```
df.horsepower.value_counts
```

Out[35]:
```
<bound method IndexOpsMixin.value_counts of 0      111.0
1      111.0
2      154.0
3      102.0
4      115.0
       ...
200    114.0
201    160.0
202    134.0
203    106.0
204    114.0
Name: horsepower, Length: 205, dtype: float64>
```

In [36]:
```
df.describe()
```

Out[36]:

	risk factor	losses	wheel base	length	width	height	weight of curb	s e
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0
mean	0.834146	122.000000	98.756585	174.049268	65.907805	53.724878	2555.565854	126.9
std	1.245307	31.681008	6.021776	12.337289	2.145204	2.443522	520.680204	41.6
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.0
25%	0.000000	101.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.0
50%	1.000000	122.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.0
75%	2.000000	137.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.0
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.0

In [37]:
```
for i in df.columns:
    if df[i].dtypes=="O":
        a=df[i].value_counts()
        print(i)
        print(a)
```

```
make
toyota           32
nissan           18
mazda            17
mitsubishi       13
honda            13
volkswagen       12
subaru           12
peugot           11
volvo            11
dodge             9
mercedes-benz     8
bmw               8
audi              7
plymouth          7
saab              6
porsche           5
isuzu             4
jaguar            3
chevrolet         3
alfa-romero       3
renault           2
mercury           1
Name: make, dtype: int64
type of fuel
gas       185
diesel     20
Name: type of fuel, dtype: int64
aspiration
std      168
turbo     37
Name: aspiration, dtype: int64
total doors
four    116
two      89
Name: total doors, dtype: int64
body
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: body, dtype: int64
wheels
fwd    120
rwd     76
4wd      9
Name: wheels, dtype: int64
location of engine
front    202
rear       3
Name: location of engine, dtype: int64
type of engine
ohc      148
ohcf      15
ohcv      13
dohc      12
l         12
rotor      4
dohcv      1
Name: type of engine, dtype: int64
total cylinders
four      159
six        24
```

```
five        11
eight        5
two          4
three        1
twelve       1
Name: total cylinders, dtype: int64
system of fuel
mpfi     94
2bbl     66
idi      20
1bbl     11
spdi      9
4bbl      3
mfi       1
spfi      1
Name: system of fuel, dtype: int64
total price
8921      2
18150     2
8845      2
8495      2
7775      2
         ..
45400     1
16503     1
5389      1
6189      1
22625     1
Name: total price, Length: 188, dtype: int64
```

In [38]:

```
df["total price"].unique()
```

Out[38]:

```
array(['13495', '16500', '13950', '17450', '15250', '17710', '18920',
       '23875', '?', '16430', '16925', '20970', '21105', '24565', '30760',
       '41315', '36880', '5151', '6295', '6575', '5572', '6377', '7957',
       '6229', '6692', '7609', '8558', '8921', '12964', '6479', '6855',
       '5399', '6529', '7129', '7295', '7895', '9095', '8845', '10295',
       '12945', '10345', '6785', '$', '11048', '32250', '35550', '36000',
       '5195', '6095', '6795', '6695', '7395', '10945', '11845', '13645',
       '15645', '8495', '10595', '10245', '10795', '11245', '18280',
       '18344', '25552', '28248', '28176', '31600', '34184', '35056',
       '40960', '45400', '16503', '5389', '6189', '6669', '7689', '9959',
       '8499', '12629', '14869', '14489', '6989', '8189', '9279', '5499',
       '7099', '6649', '6849', '7349', '7299', '7799', '7499', '7999',
       '8249', '8949', '9549', '13499', '14399', '17199', '19699',
       '18399', '11900', '13200', '12440', '13860', '15580', '16900',
       '16695', '17075', '16630', '17950', '18150', '12764', '22018',
       '32528', '34028', '37028', '9295', '9895', '11850', '12170',
       '15040', '15510', '18620', '5118', '7053', '7603', '7126', '7775',
       '9960', '9233', '11259', '7463', '10198', '8013', '11694', '5348',
       '6338', '6488', '6918', '7898', '8778', '6938', '7198', '7788',
       '7738', '8358', '9258', '8058', '8238', '9298', '9538', '8449',
       '9639', '9989', '11199', '11549', '17669', '8948', '10698', '9988',
       '10898', '11248', '16558', '15998', '15690', '15750', '7975',
       '7995', '8195', '9495', '9995', '11595', '9980', '13295', '13845',
       '12290', '12940', '13415', '15985', '16515', '18420', '18950',
       '16845', '19045', '21485', '22470', '22625'], dtype=object)
```

In [39]:

```
df.shape
```

Out[39]:

```
(205, 26)
```

In [40]:

```
df[df["total price"]=="?"]
```

Out[40]:

	risk factor	losses	make	type of fuel	aspiration	total doors	body	wheels	location of engine	wheel base	...	size eng
9	0	122.0	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	
129	1	122.0	porsche	gas	std	two	hatchback	rwd	front	98.4	...	

2 rows × 26 columns

In [41]:
```
df[df["total price"]=="$"]
```

Out[41]:

	risk factor	losses	make	type of fuel	aspiration	total doors	body	wheels	location of engine	wheel base	...	size of engine	sy of
44	1	122.0	isuzu	gas	std	two	sedan	fwd	front	94.5	...	90	
45	0	122.0	isuzu	gas	std	four	sedan	fwd	front	94.5	...	90	

2 rows × 26 columns

In [42]:
```
df["total price"].replace(to_replace="?",value="",inplace=True)
```

In [43]:
```
df["total price"].replace(to_replace="$",value="",inplace=True)
```

In [44]:
```
df["total price"]=pd.to_numeric(df["total price"])
```

In [45]:
```
df["total price"].fillna(df["total price"].mean(),inplace=True)
```

In [46]:
```
df.dropna(axis=0,inplace=True)
```

In [47]:
```
df.shape
```

Out[47]:
```
(203, 26)
```

In [48]:
```
df['losses']=df['losses'].round(2)
```

In [49]:
```
df['bore']=df['bore'].round(2)
```

In [50]:
```
df['stroke']=df['stroke'].round(2)
```

In [51]:
```
df['horsepower']=df['horsepower'].round(2)
```

In [52]:
```
df['total price']=df['total price'].round(2)
```

In [53]:
```
df.describe()
```

Out[53]:

	risk factor	losses	wheel base	length	width	height	weight of curb	s e
count	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.0
mean	0.832512	122.000000	98.782759	173.999015	65.901478	53.733498	2555.921182	126.8
std	1.247384	31.837458	6.045680	12.385511	2.154835	2.442864	523.205555	41.8
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.0
25%	0.000000	101.000000	94.500000	166.300000	64.050000	52.000000	2145.000000	97.0
50%	1.000000	122.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	119.0
75%	2.000000	137.000000	102.400000	183.300000	66.900000	55.500000	2943.500000	143.0
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.0

Libraries imported for Proper Visualization

In [54]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [55]:

```
df.describe()
```

Out[55]:

	risk factor	losses	wheel base	length	width	height	weight of curb	s e
count	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.000000	203.0
mean	0.832512	122.000000	98.782759	173.999015	65.901478	53.733498	2555.921182	126.8
std	1.247384	31.837458	6.045680	12.385511	2.154835	2.442864	523.205555	41.8
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.0
25%	0.000000	101.000000	94.500000	166.300000	64.050000	52.000000	2145.000000	97.0
50%	1.000000	122.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	119.0
75%	2.000000	137.000000	102.400000	183.300000	66.900000	55.500000	2943.500000	143.0
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.0

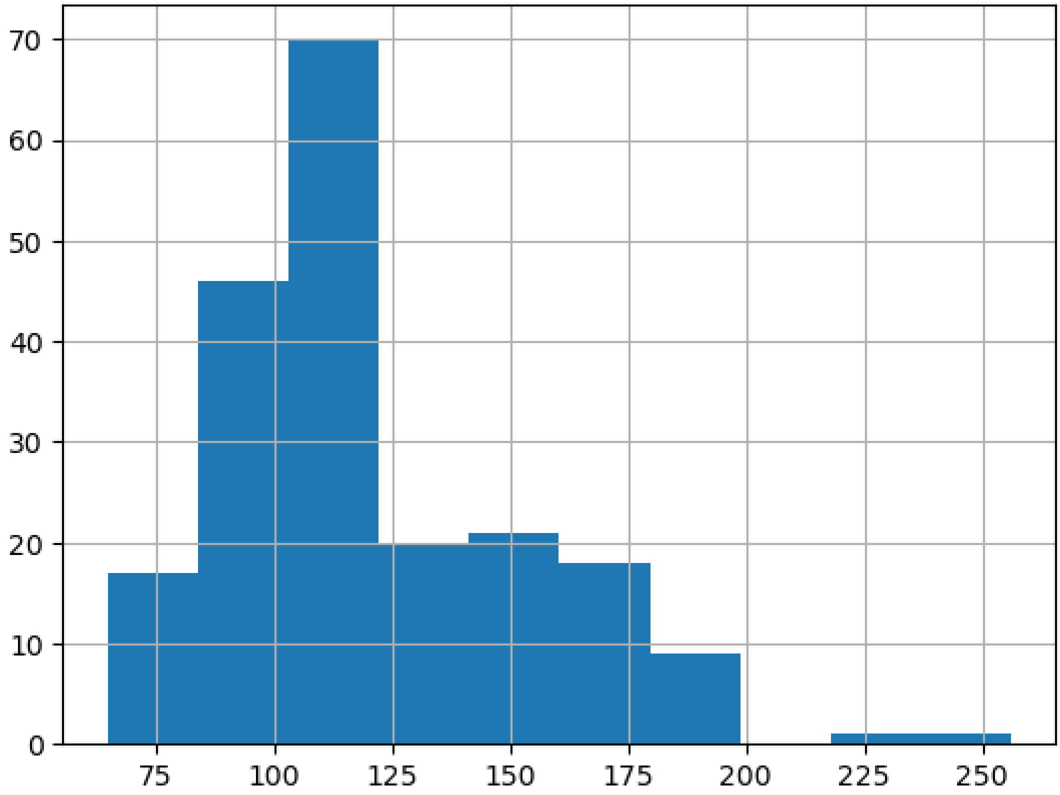
Histogram distribution of losses

In [56]:

```
df.losses.hist()
```

Out[56]:

<AxesSubplot:>



Outlier detection and removal

In [57]:
```
upper_limit = df.losses.mean() + 3*df.losses.std()
upper_limit
```

Out[57]: 217.5123754357839

In [58]:
```
df[df["losses"]>upper_limit]
```

Out[58]:

	risk factor	losses	make	type of fuel	aspiration	total doors	body	wheels	location of engine	wheel base	...
106	1	231.0	nissan	gas	std	two	hatchback	rwd	front	99.2	...
190	3	256.0	volkswagen	gas	std	two	hatchback	fwd	front	94.5	...

2 rows × 26 columns

In [59]:
```
df.drop(index=106,inplace=True)
```

In [60]:
```
df.drop(index=190,inplace=True)
```

In [61]:
```
df.columns=[each.replace(" ","_") for each in df.columns]
```

In [62]:
```
df.columns=[each.replace("-","_") for each in df.columns]
```

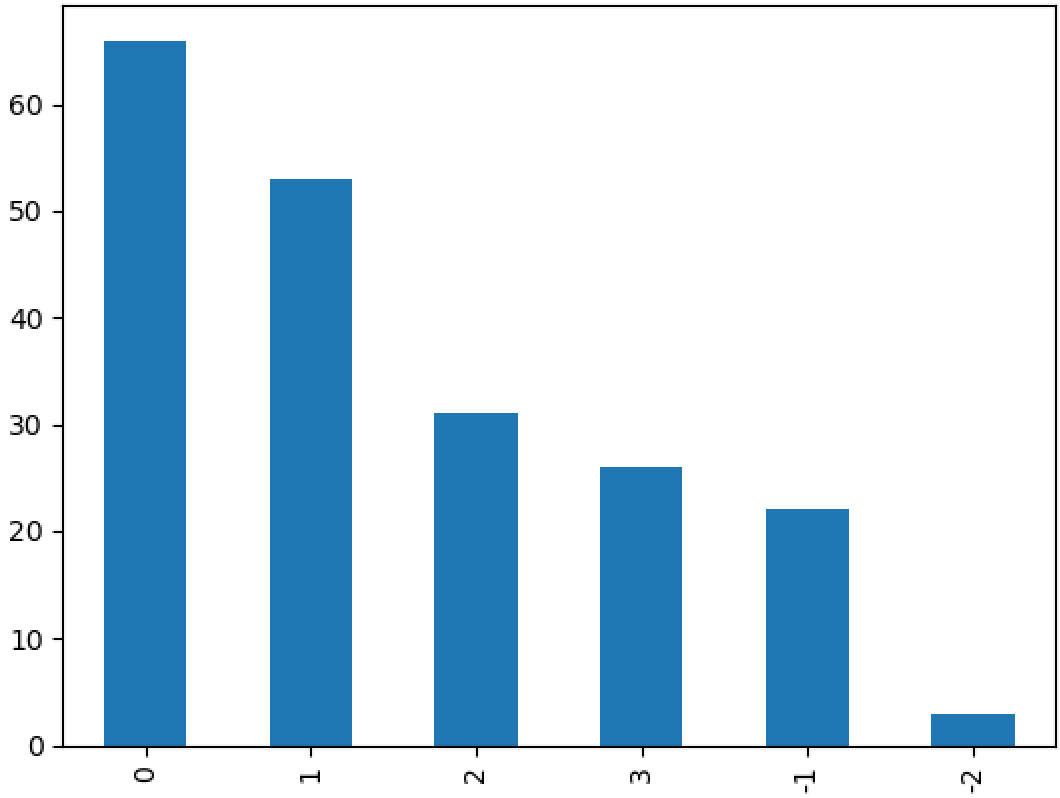
In [63]:
```
df.shape
```

Out[63]: (201, 26)

Risk factor- it shows the no. of samples are more having high risk as 1,2 and 3 compared to 0,-1 & -2

In [64]:
```
df.risk_factor.value_counts().plot(kind="bar")
```

Out[64]:
```
<AxesSubplot:>
```



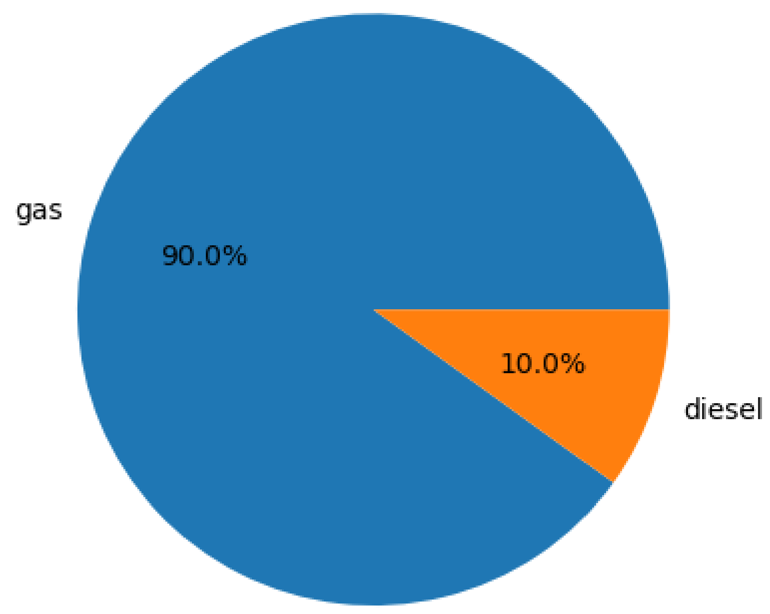
For Top 4 car makers

In [65]:
```
df.make.value_counts().nlargest(4)
```

Out[65]:
```
toyota        32
mazda         17
nissan        17
mitsubishi    13
Name: make, dtype: int64
```

Pie chart distribution of asked output i.e. most used fuel

In [68]:
```
y=np.array(df.type_of_fuel.value_counts().values)
x=df.type_of_fuel.value_counts().index
plt.pie(y,labels=x,autopct="%1.1f%%")
plt.show()
```



Maker having highest use of diesel

In [69]:
```
df.make[df.type_of_fuel=="diesel"].value_counts().head(1)
```

Out[69]:
```
peugot    5
Name: make, dtype: int64
```

Highest installed aspiration count

In [70]:
```
df.aspiration.value_counts().head(1)
```

Out[70]:
```
std    164
Name: aspiration, dtype: int64
```

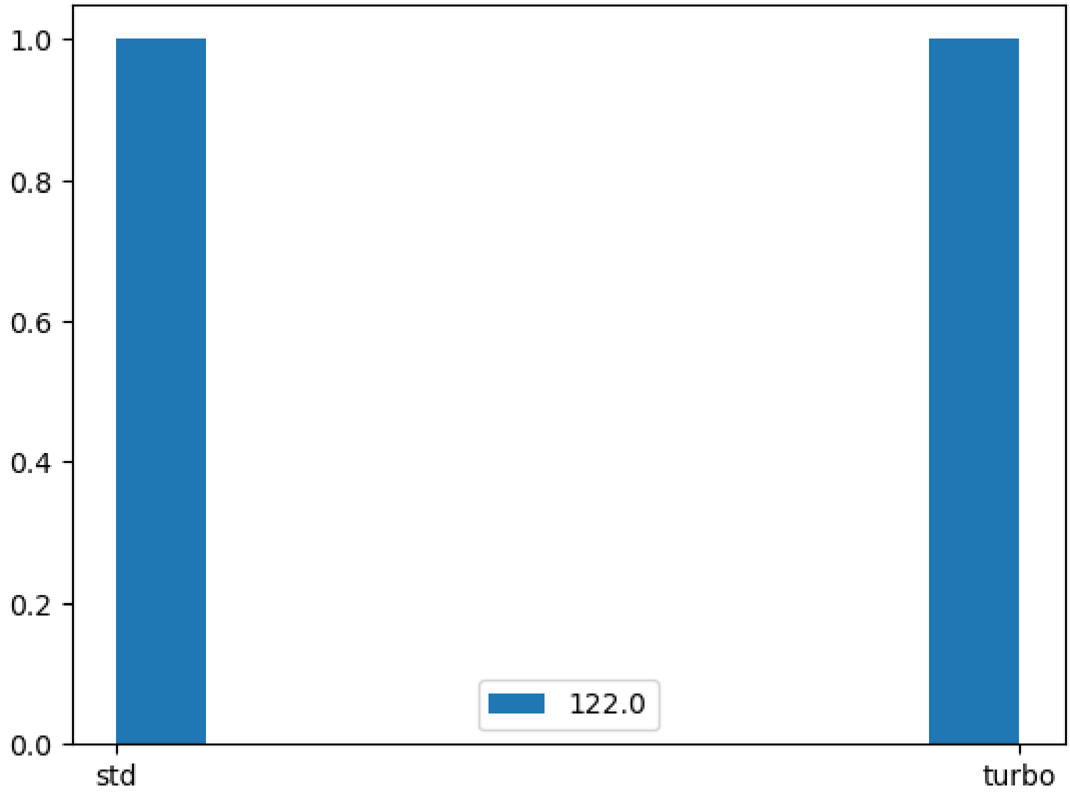
Named 2 makes having highest utilization of turbo

In [71]:
```
df.make[df.aspiration=="turbo"].value_counts().head(2)
```

Out[71]:
```
mitsubishi    6
peugot        6
Name: make, dtype: int64
```

Higher losses in std and tubro histogram comparison

In [73]:
```
#ax = plt.subplots(figsize =(10, 7))
plt.hist(df.aspiration.value_counts().index)
y=np.array(df["losses"].value_counts().index)
plt.legend(y)
plt.show()
```



Car makers who doesn't have 2 doors

In [74]:
```
df.make[df.total_doors=="four"].value_counts()
```

Out[74]:
```
toyota           18
volvo            11
peugot           11
subaru            9
nissan            9
volkswagen        8
mazda             8
bmw               5
audi              5
mercedes-benz     5
honda             5
dodge             5
plymouth          4
mitsubishi        4
saab              3
jaguar            2
isuzu             2
chevrolet         1
Name: make, dtype: int64
```

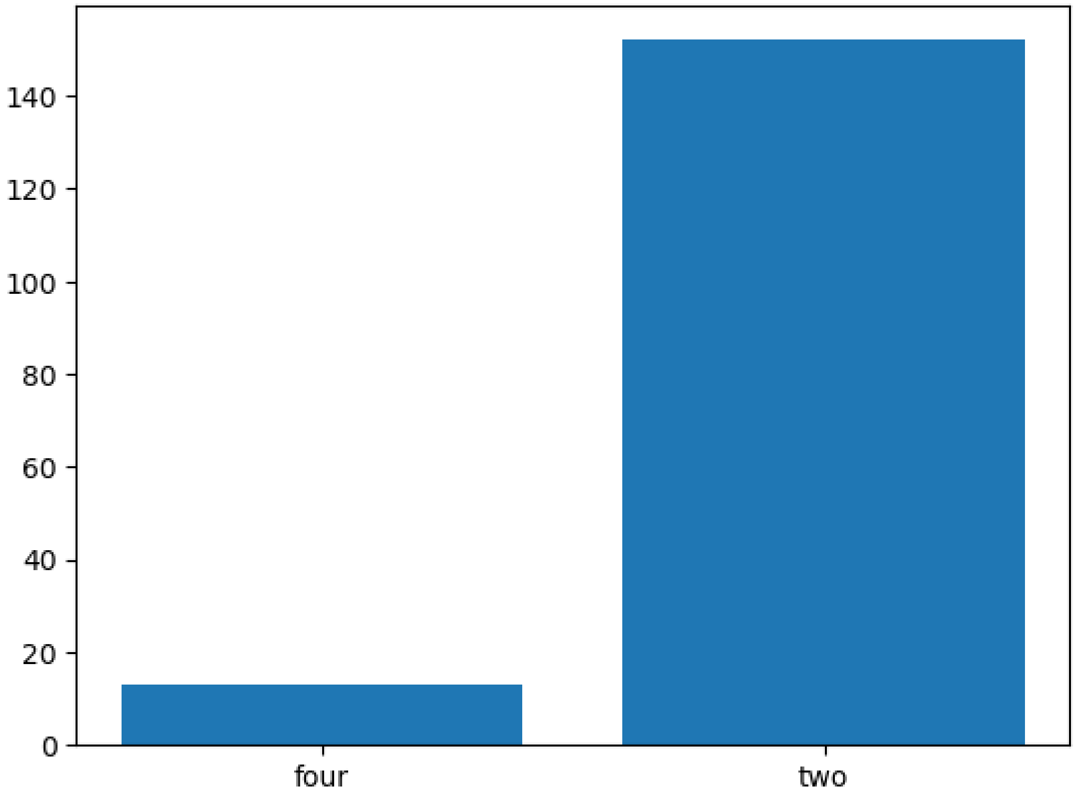
Car makers who doesn't have 4 doors

In [75]:
```
df.make[df.total_doors=="two"].value_counts()
```

Out[75]:
```
toyota           14
mitsubishi        9
mazda             9
honda             8
nissan            8
porsche           5
dodge             4
alfa-romero       3
subaru            3
saab              3
plymouth          3
mercedes-benz     3
bmw               3
volkswagen        3
audi              2
isuzu             2
chevrolet         2
mercury           1
jaguar            1
Name: make, dtype: int64
```

No of Doors and Losses

In [178…
```
grouped = df.groupby('total_doors')['risk_factor'].sum()
grouped
x=grouped.index
y=grouped.values
plt.bar(x,y)
plt.show()
```



Most frequently occuring cars by its Body type

In [81]:
```
df.body.unique()
```

Out[81]:
```
array(['convertible', 'hatchback', 'sedan', 'wagon', 'hardtop'],
      dtype=object)
```

In [82]:
```
grouped = df.groupby("body")
grouped.head(1)
```

Out[82]:

	risk_factor	losses	make	type_of_fuel	aspiration	total_doors	body	wheels	locati
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	
3	2	164.0	audi	gas	std	four	sedan	fwd	
7	1	122.0	audi	gas	std	four	wagon	fwd	
69	0	93.0	mercedes-benz	diesel	turbo	two	hardtop	rwd	

5 rows × 26 columns

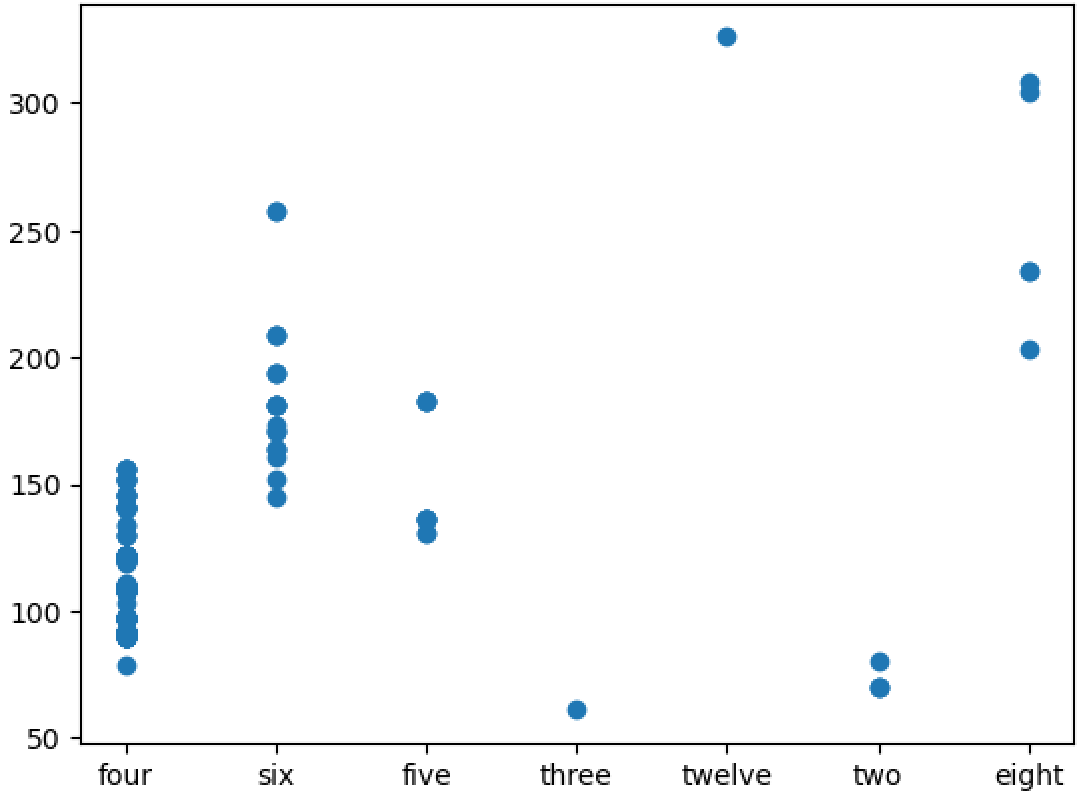
In [83]:
```
df.body.value_counts().head(2)
```

Out[83]:
```
sedan        96
hatchback    67
Name: body, dtype: int64
```

Size of engine and total cylinders relationships

In [84]:
```
print("shows Linear Relationship-->as the no.of cylinders increase the size of eng
plt.scatter(df.total_cylinders,df.size_of_engine)
plt.show()
```

```
shows Linear Relationship-->as the no.of cylinders increase the size of engine inc
rease
```



In [85]:
```
df.columns
```

Out[85]:
```
Index(['risk_factor', 'losses', 'make', 'type_of_fuel', 'aspiration',
       'total_doors', 'body', 'wheels', 'location_of_engine', 'wheel_base',
       'length', 'width', 'height', 'weight_of_curb', 'type_of_engine',
       'total_cylinders', 'size_of_engine', 'system_of_fuel', 'bore', 'stroke',
       'ratio_of_comprehenssion', 'horsepower', 'peak_rpm', 'mpg_in_city',
       'mpg_on_highway', 'total_price'],
      dtype='object')
```

Fuel type having highest ratio of compression

In [86]:
```
grouped = df.groupby('type_of_fuel')['ratio_of_comprehenssion'].sum()
grouped.nlargest(1)
```

Out[86]:
```
type_of_fuel
gas    1604.22
Name: ratio_of_comprehenssion, dtype: float64
```

Correlations of other features with Target variable i.e. total_price

In [87]:
```
df_corr=df.corr()
```

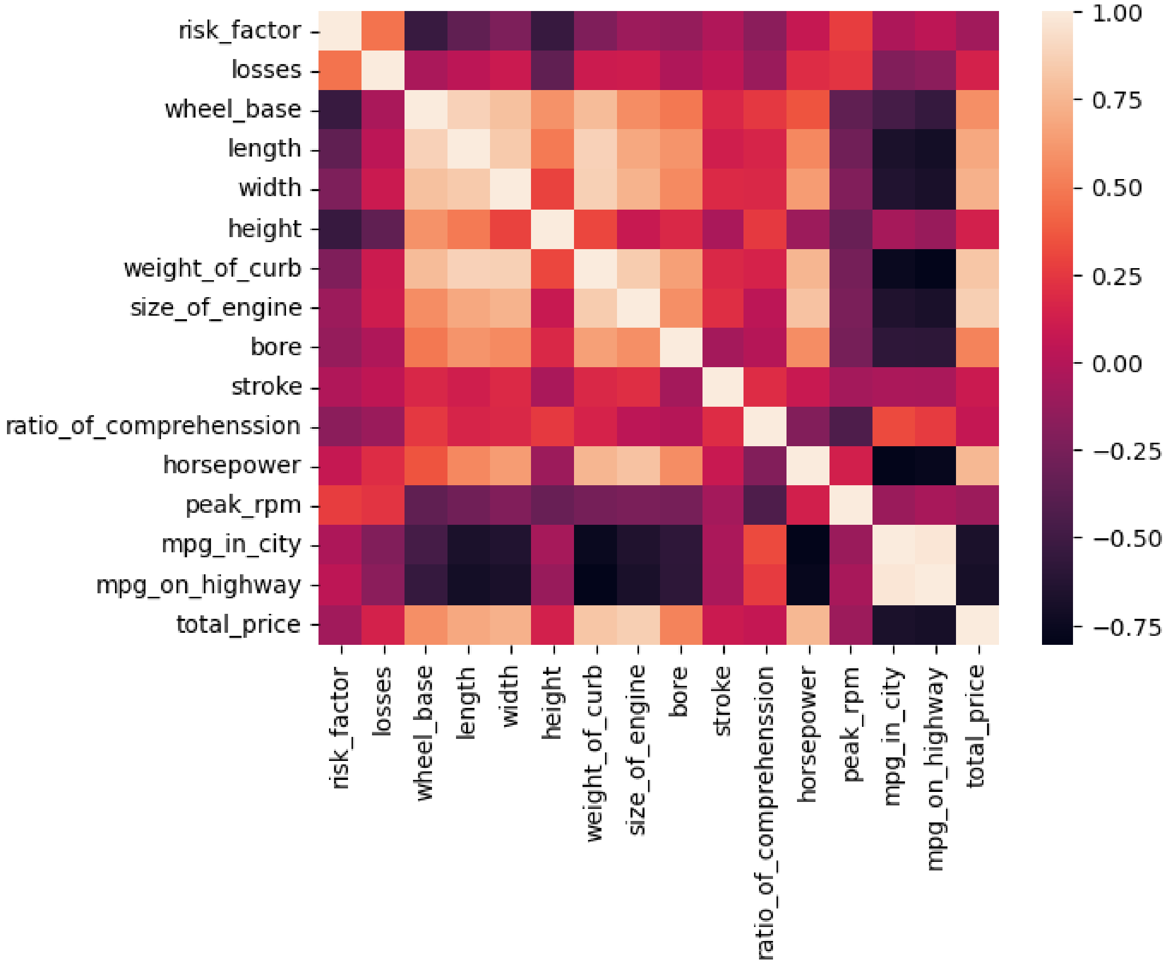
In [88]:
```
df_corr.total_price.sort_values(ascending=False)
```

Out[88]:
```
total_price                  1.000000
size_of_engine               0.863476
weight_of_curb               0.821151
horsepower                   0.758516
width                        0.730388
length                       0.685818
wheel_base                   0.582262
bore                         0.534732
losses                       0.142754
height                       0.138823
stroke                       0.094476
ratio_of_comprehenssion      0.069698
risk_factor                 -0.079865
peak_rpm                    -0.100177
mpg_in_city                 -0.669943
mpg_on_highway              -0.691240
Name: total_price, dtype: float64
```

Graphical representation of correlation heatmap

In [90]:
```
sns.heatmap(df.corr())
```

Out[90]: <AxesSubplot:>



splitting up the dependent variable and independent variable

In [91]:
```
x=df.drop(["total_price","location_of_engine"],axis=1)
y=df["total_price"]
```

In [92]:
```
x.shape,y.shape
```

Out[92]:
```
((201, 24), (201,))
```

Scaling data to address class imbalance-Feature engineering

In [93]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [94]:
```
mmscaler= MinMaxScaler()
```

In [95]:
```
x.describe()
```

Out[95]:

	risk_factor	losses	wheel_base	length	width	height	weight_of_curb	si
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	0.820896	120.791045	98.801990	174.017910	65.900995	53.765174	2554.686567	
std	1.244091	29.548369	6.068179	12.429355	2.156780	2.432628	523.659428	
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	
25%	0.000000	101.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	
50%	1.000000	122.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	
75%	2.000000	137.000000	102.400000	183.500000	66.900000	55.500000	2935.000000	
max	3.000000	197.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	

In [96]:
```
x.shape
```

Out[96]:
```
(201, 24)
```

In [97]:
```
x[['size_of_engine', 'weight_of_curb', 'horsepower',
       'width', 'mpg_on_highway', 'length', 'mpg_in_city', 'wheel_base',
       'bore', 'losses', 'height', 'peak_rpm', 'stroke',
       'ratio_of_comprehenssion']]=mmscaler.fit_transform(x[['size_of_engine', 'we
       'width', 'mpg_on_highway', 'length', 'mpg_in_city', 'wheel_base',
       'bore', 'losses', 'height', 'peak_rpm', 'stroke',
       'ratio_of_comprehenssion']])
```

In [ ]:
```

```

One hot encoding of the data, so that string type data can be converted to numeric

In [98]:
```
x=pd.get_dummies(x)
```

In [99]:
```
x.head()
```

Out[99]:

	risk_factor	losses	wheel_base	length	width	height	weight_of_curb	size_of_engine
0	3	0.431818	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377
1	3	0.431818	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377
2	1	0.431818	0.230321	0.449254	0.433333	0.383333	0.517843	0.343396
3	2	0.750000	0.384840	0.529851	0.491667	0.541667	0.329325	0.181132
4	2	0.750000	0.373178	0.529851	0.508333	0.541667	0.518231	0.283019

5 rows × 72 columns

Checking if there are intersting correlation happening

In [100…

```
x.corr().abs()
```

Out[100]:

	risk_factor	losses	wheel_base	length	width	height	weight_of_cur
risk_factor	1.000000	0.466866	0.532249	0.355862	0.230066	0.537029	0.22534
losses	0.466866	1.000000	0.046472	0.029374	0.094473	0.354358	0.10019
wheel_base	0.532249	0.046472	1.000000	0.878029	0.798412	0.594851	0.77845
length	0.355862	0.029374	0.878029	1.000000	0.841204	0.498810	0.87951
width	0.230066	0.094473	0.798412	0.841204	1.000000	0.289296	0.86687
...	...	...	...	...	...	...	...
system_of_fuel_idi	0.193099	0.096324	0.306872	0.214274	0.236245	0.283542	0.21898
system_of_fuel_mfi	0.124164	0.058078	0.033900	0.004665	0.013114	0.103890	0.03469
system_of_fuel_mpfi	0.000963	0.173729	0.362181	0.515960	0.466023	0.130019	0.52678
system_of_fuel_spdi	0.186325	0.078254	0.119298	0.079476	0.045945	0.286373	0.00208
system_of_fuel_spfi	0.067184	0.002900	0.032732	0.008087	0.023040	0.068922	0.02427

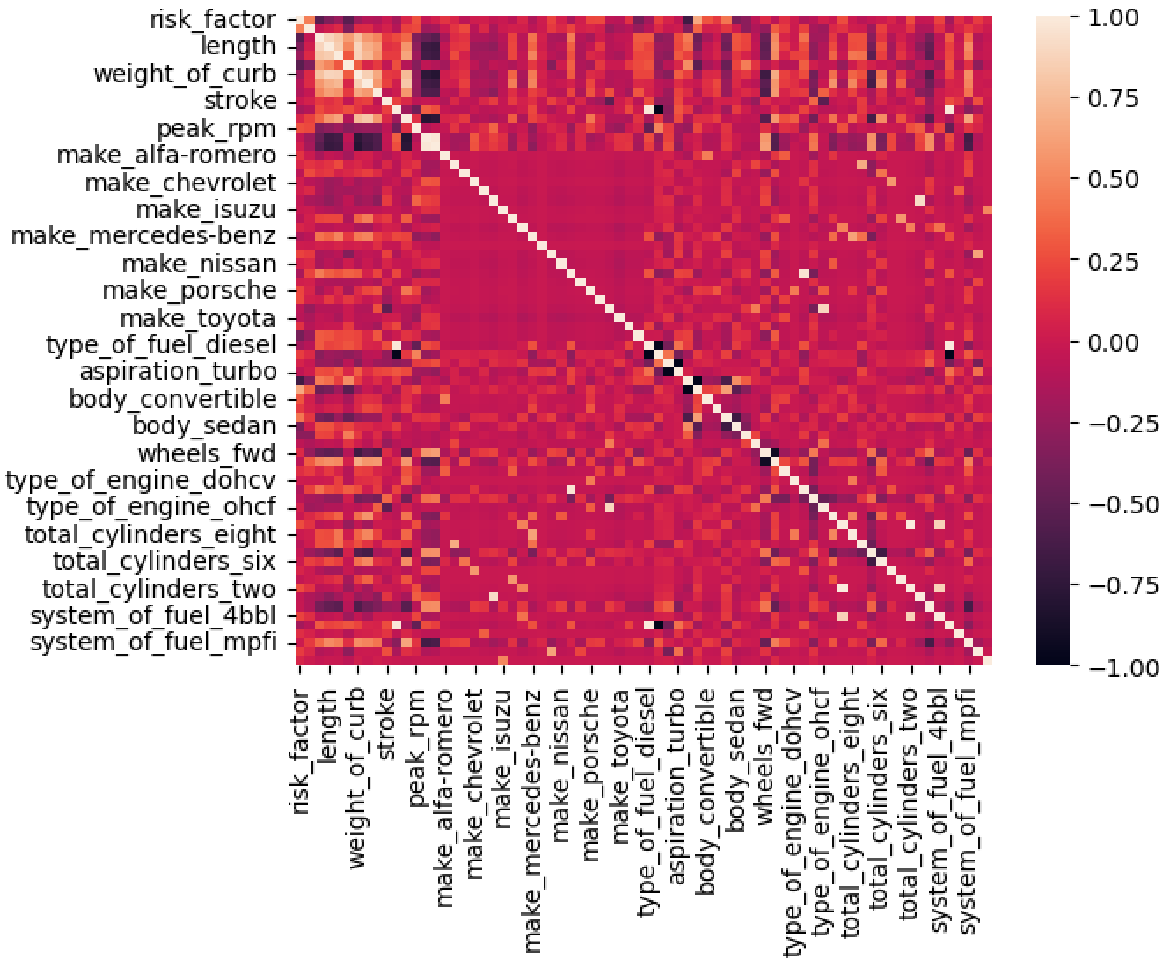
72 rows × 72 columns

In [101…

```
sns.heatmap(x.corr())
```

Out[101]:

```
<AxesSubplot:>
```



Splitting model into training and testing dataset

In [102…
```
from sklearn.model_selection import train_test_split
```

In [103…
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=10)
```

In [104…
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[104]:
```
((160, 72), (41, 72), (160,), (41,))
```

In [105…
```
x_train=x_train.iloc[:,1:]
```

In [106…
```
x_test=x_test.iloc[:,1:]
```

Applying Multiple Linear regression

In [107…
```
from sklearn.linear_model import LinearRegression
```

In [108…
```
linear_model=LinearRegression()
```

In [109…
```
linear_model.fit(x_train,y_train)
```

Out[109]:
```
LinearRegression()
```

In [110…
```
x_test.head()
```

Out[110]:

	losses	wheel_base	length	width	height	weight_of_curb	size_of_engine	bore
59	0.484848	0.355685	0.547761	0.516667	0.491667	0.347944	0.230189	0.607143
5	0.431818	0.384840	0.540299	0.500000	0.441667	0.395268	0.283019	0.464286
20	0.121212	0.230321	0.264179	0.275000	0.350000	0.163305	0.109434	0.350000
128	0.431818	0.084548	0.414925	0.391667	0.316667	0.508922	0.501887	0.857143
52	0.295455	0.189504	0.268657	0.325000	0.525000	0.161753	0.113208	0.350000

5 rows × 71 columns

In [111…

```
y_pred=linear_model.predict(x_test)
y_pred
```

Out[111]:

```
array([ 1.09800000e+04,  1.33520000e+04,  9.42471118e+15,  3.74360000e+04,
        7.24400000e+03,  9.42471118e+15,  8.70800000e+03,  1.27320000e+04,
        2.64000000e+04,  1.28760000e+04,  6.31600000e+03,  1.99760000e+04,
        1.08005023e+16,  5.09600000e+03,  1.25040000e+04,  1.19760000e+04,
        1.67760000e+04,  6.94000000e+03,  1.67720000e+04,  1.71160000e+04,
        6.57200000e+03,  2.75080000e+04,  1.09800000e+04,  1.08880000e+04,
        1.63640000e+04,  8.73600000e+03,  7.10000000e+03,  1.34480000e+04,
        3.53480000e+04,  9.72800000e+03,  6.43200000e+03,  9.32400000e+03,
        6.58000000e+03,  6.12800000e+03,  8.04400000e+03,  1.61240000e+04,
        1.55440000e+04,  1.12240000e+04, -5.58845650e+13,  1.67960000e+04,
        1.59440000e+04])
```

In [112…

```
from sklearn.metrics import r2_score,accuracy_score
```

Score from Regression-ML Alogirthm

In [113…

```
score=linear_model.score(x_train,y_train)
score
```

Out[113]:

```
0.9637330897920863
```

In [120…

```
score2=linear_model.score(x_test,y_test)
score2
```

Out[120]:

```
-1.5373213899407477e+23
```

In [114…

```
y_pred=linear_model.predict(x_test)
y_pred
```

Out[114]:

```
array([ 1.09800000e+04,  1.33520000e+04,  9.42471118e+15,  3.74360000e+04,
        7.24400000e+03,  9.42471118e+15,  8.70800000e+03,  1.27320000e+04,
        2.64000000e+04,  1.28760000e+04,  6.31600000e+03,  1.99760000e+04,
        1.08005023e+16,  5.09600000e+03,  1.25040000e+04,  1.19760000e+04,
        1.67760000e+04,  6.94000000e+03,  1.67720000e+04,  1.71160000e+04,
        6.57200000e+03,  2.75080000e+04,  1.09800000e+04,  1.08880000e+04,
        1.63640000e+04,  8.73600000e+03,  7.10000000e+03,  1.34480000e+04,
        3.53480000e+04,  9.72800000e+03,  6.43200000e+03,  9.32400000e+03,
        6.58000000e+03,  6.12800000e+03,  8.04400000e+03,  1.61240000e+04,
        1.55440000e+04,  1.12240000e+04, -5.58845650e+13,  1.67960000e+04,
        1.59440000e+04])
```

This score clearyl suggest of Overfitting model

Applying Regularization L1 & L2

In [121…
```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

In [122…
```
ridge_reg=Ridge(alpha=7,max_iter=155)
ridge_reg.fit(x_train,y_train)
```

Out[122]:
```
Ridge(alpha=7, max_iter=155)
```

In [123…
```
ridge_reg.score(x_train,y_train)
```

Out[123]:
```
0.8625123462705805
```

In [124…
```
ridge_reg.score(x_test,y_test)
```

Out[124]:
```
0.8499251968035425
```

After iterations with different parameters, scores with ridge has improved-Overfitting removed but accuracy is still less

Applying LASSO regularization

In [169…
```
lasso_reg=Lasso(alpha=7,max_iter=120)
lasso_reg.fit(x_train,y_train)
```

```
C:\Users\Hp\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.p
y:647: ConvergenceWarning: Objective did not converge. You might want to increase
the number of iterations, check the scale of the features or consider increasing r
egularisation. Duality gap: 1.183e+07, tolerance: 1.065e+06
  model = cd_fast.enet_coordinate_descent(
```

Out[169]:
```
Lasso(alpha=7, max_iter=120)
```

In [170…
```
lasso_reg.score(x_train,y_train)
```

Out[170]:
```
0.9539825792900626
```

In [171…
```
lasso_reg.score(x_test,y_test)
```

Out[171]:
```
0.9328324470172962
```

This accuracy shows the model improved with the help of LASSO

Applying other ML-Alogirthm as Decision Tree & Random Forest

In [128…
```
from sklearn.tree import DecisionTreeRegressor
```

In [146…
```
treemodel= DecisionTreeRegressor(min_impurity_decrease=0.05,random_state=11)
```

In [147…
```
treemodel.fit(x_train,y_train)
```

Out[147]:
```
DecisionTreeRegressor(min_impurity_decrease=0.05, random_state=11)
```

In [148…
```
y_pred2=treemodel.predict(x_test)
y_pred2
```

Out[148]:
```
array([10345. , 13950. ,  5499. , 33278. ,  6095. ,  5151. , 10295. ,
       11845. , 25552. , 22018. ,  7150.5, 16925. , 11900. ,  7603. ,
       10198. , 10198. , 17075. ,  6189. , 17950. , 16630. ,  6918. ,
       15580. , 10345. ,  8189. , 16695. ,  7995. ,  6918. , 13495. ,
       35550. ,  8948.5,  8238. , 10295. ,  5572. ,  6849. ,  7295. ,
       16515. , 15985. ,  8921. , 12764. , 15998. , 16515. ])
```

In [149…
```
r2=r2_score(y_pred2,y_test)
r2
```

Out[149]:
```
0.9158020049879919
```

In [150…
```
from sklearn.ensemble import RandomForestRegressor
```

In [165…
```
forestmodel= RandomForestRegressor(n_estimators=25,random_state=10)
```

In [166…
```
forestmodel.fit(x_train,y_train)
```

Out[166]:
```
RandomForestRegressor(n_estimators=25, random_state=10)
```

In [167…
```
y_pred=forestmodel.predict(x_test)
y_pred
```

Out[167]:
```
array([ 9877.82666667, 12038.16      ,  8168.2716    , 31142.1252    ,
        5964.48      ,  8291.0068    , 10727.6       , 12213.        ,
       29491.72      , 15058.64      ,  7324.52      , 14814.4       ,
       16566.2208    ,  7562.6       , 10214.21333333,  9796.30666667,
       16787.16      ,  6446.04      , 16547.        , 18215.4452    ,
        8030.36      , 14398.16      ,  9877.82666667,  9615.24      ,
       14670.64      ,  8042.96      ,  8104.92      , 12807.76      ,
       36163.6       ,  9610.05333333,  8177.32      ,  9980.56      ,
        5912.04      ,  6852.8452    ,  7363.34      , 15624.1652    ,
       15646.8104    , 10088.        , 12743.32      , 16374.76      ,
       15688.36      ])
```

In [168…
```
r2=r2_score(y_pred,y_test)
r2
```

Out[168]:
```
0.9221138840248214
```

observations from ML-Algorithms

1. Compared to Multiple Linear regression, accuracy in Ridge increases

2. Compared to Ridge, accuracy in Lasso increases

3. Compared to Multiple Linear regression, accuracy in Decision Tree increases

4. Compared to Decision Tree, accuracy in Random Forest increases

Overall insights from excercise

1. Diesel usage is very less compared to gas cars

2. Gas vehicles are very high ratio of compresssion

3. Four wheel has very low risk factor instead of two wheel

4. As the cylinder increases from 2 to 12, the size of engine also increases

5. Sedan and Hatchback vehicles are highly preferred

6. Price is highly impacted by size of engine &weight of curb

7. Cars having high prices decreases risk factor, mpg in city and highway