Data Understanding - Automobile Data

July 2, 2020

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

%matplotlib inline
```

[2]:
```python
automobile = pd.read_csv('Automobile price data _Raw_.csv')
```

[3]:
```python
automobile.head()
```

[3]:
```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0          3                 ?  alfa-romero       gas        std          two
1          3                 ?  alfa-romero       gas        std          two
2          1                 ?  alfa-romero       gas        std          two
3          2               164         audi       gas        std         four
4          2               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  …  engine-size  \
0  convertible          rwd           front        88.6  …          130
1  convertible          rwd           front        88.6  …          130
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
3         mpfi  3.19    3.40              10.0        102      5500       24
4         mpfi  3.19    3.40               8.0        115      5500       18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
```

```
4          22  17450

[5 rows x 26 columns]
```

[4]:
```python
automobile.shape
```

[4]:
```
(205, 26)
```

[5]:
```python
automobile.isnull().sum()
```

[5]:
```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

[6]:
```python
# Finding the missing values
automobile.isna().any()
```

[6]:
```
symboling            False
normalized-losses    False
make                 False
fuel-type            False
aspiration           False
num-of-doors         False
```

```
body-style           False
drive-wheels         False
engine-location      False
wheel-base           False
length               False
width                False
height               False
curb-weight          False
engine-type          False
num-of-cylinders     False
engine-size          False
fuel-system          False
bore                 False
stroke               False
compression-ratio    False
horsepower           False
peak-rpm             False
city-mpg             False
highway-mpg          False
price                False
dtype: bool
```

[7]:
```
automobile.describe()
```

[7]:
```
        symboling  wheel-base      length       width      height  \
count  205.000000  205.000000  205.000000  205.000000  205.000000
mean     0.834146   98.756585  174.049268   65.907805   53.724878
std      1.245307    6.021776   12.337289    2.145204    2.443522
min     -2.000000   86.600000  141.100000   60.300000   47.800000
25%      0.000000   94.500000  166.300000   64.100000   52.000000
50%      1.000000   97.000000  173.200000   65.500000   54.100000
75%      2.000000  102.400000  183.100000   66.900000   55.500000
max      3.000000  120.900000  208.100000   72.300000   59.800000

       curb-weight  engine-size  compression-ratio    city-mpg  highway-mpg
count   205.000000   205.000000         205.000000  205.000000   205.000000
mean   2555.565854   126.907317          10.142537   25.219512    30.751220
std     520.680204    41.642693           3.972040    6.542142     6.886443
min    1488.000000    61.000000           7.000000   13.000000    16.000000
25%    2145.000000    97.000000           8.600000   19.000000    25.000000
50%    2414.000000   120.000000           9.000000   24.000000    30.000000
75%    2935.000000   141.000000           9.400000   30.000000    34.000000
max    4066.000000   326.000000          23.000000   49.000000    54.000000
```

0.1 Automobile Data Description

1) symboling: ratings -3, -2, -1, 0, 1, 2, 3. degree to which the auto is more risky than its price indicates
2) normalized-losses: continuous from 65 to 256.
3) make: alfa-romero, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota, volkswagen, volvo
4) fuel-type: diesel, gas
5) aspiration: std, turbo.
6) num-of-doors: four, two.
7) body-style: hardtop, wagon, sedan, hatchback, convertible.
8) drive-wheels: 4wd, fwd, rwd.
9) engine-location: front, rear.
10) wheel-base: continuous from 86.6 120.9.
11) length: continuous from 141.1 to 208.1.
12) width: continuous from 60.3 to 72.3.
13) height: continuous from 47.8 to 59.8.
14) curb-weight: continuous from 1488 to 4066.
15) engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
16) num-of-cylinders: eight, five, four, six, three, twelve, two.
17) engine-size: continuous from 61 to 326.
18) fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
19) bore: continuous from 2.54 to 3.94.
20) stroke: continuous from 2.07 to 4.17.
21) compression-ratio: continuous from 7 to 23.
22) horsepower: continuous from 48 to 288.
23) peak-rpm: continuous from 4150 to 6600.
24) city-mpg: continuous from 13 to 49.
25) highway-mpg: continuous from 16 to 54.
26) price: continuous from 5118 to 45400.

[8]:
```
automobile.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    object
 19  stroke             205 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object
 22  peak-rpm           205 non-null    object
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              205 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

[9]:
```
automobile['normalized-losses'].value_counts()
#We are able to see 41 values have '?'
```

[9]:
```
?      41
161    11
91      8
150     7
134     6
128     6
104     6
168     5
74      5
95      5
```

```
85      5
103     5
94      5
65      5
102     5
93      4
106     4
118     4
122     4
148     4
154     3
83      3
115     3
125     3
101     3
137     3
153     2
164     2
192     2
108     2
113     2
188     2
119     2
197     2
194     2
110     2
81      2
145     2
158     2
89      2
87      2
129     2
107     1
186     1
78      1
142     1
121     1
98      1
256     1
90      1
77      1
231     1
Name: normalized-losses, dtype: int64
```

[10]:

```
automobile['num-of-doors'].value_counts()
# Here also in the 'num-of-doors' column we can see two '?' values.
```

[10]:
```
four    114
two      89
?         2
Name: num-of-doors, dtype: int64
```

[11]:
```
automobile['bore'].value_counts()
# In the bore column also, we have 4 values as '?'.
```

[11]:
```
3.62    23
3.19    20
3.15    15
2.97    12
3.03    12
3.46     9
3.78     8
3.43     8
3.31     8
2.91     7
3.27     7
3.05     6
3.58     6
3.54     6
3.39     6
3.01     5
3.70     5
3.35     4
?        4
3.17     3
3.74     3
3.59     3
3.80     2
3.50     2
3.47     2
3.94     2
3.24     2
3.13     2
3.33     2
3.63     2
3.34     1
3.76     1
2.54     1
2.68     1
3.61     1
3.08     1
2.99     1
3.60     1
2.92     1
```

```
Name: bore, dtype: int64
```

[12]:
```
automobile['stroke'].value_counts()
#In the stroke column, we have 4 values with '?' values.
```

[12]:
```
3.40    20
3.15    14
3.23    14
3.03    14
3.39    13
2.64    11
3.29     9
3.35     9
3.46     8
3.50     6
3.41     6
3.19     6
3.27     6
3.07     6
3.11     6
3.58     6
3.52     5
3.64     5
3.54     4
3.86     4
3.47     4
?        4
3.90     3
2.90     3
3.08     2
2.19     2
3.10     2
2.80     2
2.68     2
4.17     2
3.16     1
2.36     1
3.12     1
2.76     1
2.87     1
3.21     1
2.07     1
Name: stroke, dtype: int64
```

[13]:
```
automobile['horsepower'].value_counts()
#Also, 2 values with '?', in horsepower column
```

[13]:
```
68     19
70     11
69     10
116     9
110     8
95      7
88      6
160     6
62      6
101     6
114     6
102     5
82      5
84      5
76      5
97      5
145     5
123     4
92      4
86      4
111     4
90      3
152     3
85      3
207     3
73      3
182     3
121     3
176     2
161     2
155     2
56      2
100     2
156     2
184     2
112     2
162     2
52      2
?       2
94      2
135     1
120     1
262     1
60      1
115     1
64      1
55      1
```

```
58      1
142     1
72      1
48      1
134     1
288     1
78      1
106     1
154     1
175     1
200     1
140     1
143     1
Name: horsepower, dtype: int64
```

[14]:
```
automobile['peak-rpm'].value_counts()
#In peak-rpm also, we see two '?' values.
```

[14]:
```
5500    37
4800    36
5000    27
5200    23
5400    13
6000     9
5250     7
4500     7
5800     7
4200     5
4150     5
4350     4
4750     4
5100     3
4400     3
5900     3
4250     3
6600     2
?        2
4650     1
4900     1
5300     1
5750     1
5600     1
Name: peak-rpm, dtype: int64
```

[15]:
```
automobile['price'].value_counts()
```

[15]:
```
?        4
13499    2
18150    2
6229     2
9279     2
        ..
9639     1
22625    1
15645    1
9959     1
5389     1
Name: price, Length: 187, dtype: int64
```

[16]:
```
# We will replace the '?' values with NaN.
automobile = automobile.replace('?',np.NAN)
automobile.isnull().sum()
```

[16]:
```
symboling             0
normalized-losses    41
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

[18]:
```
# Change the datatypes for normalized-losses, bore, stroke, horsepower,␣
↪peak-rpm and price column
```

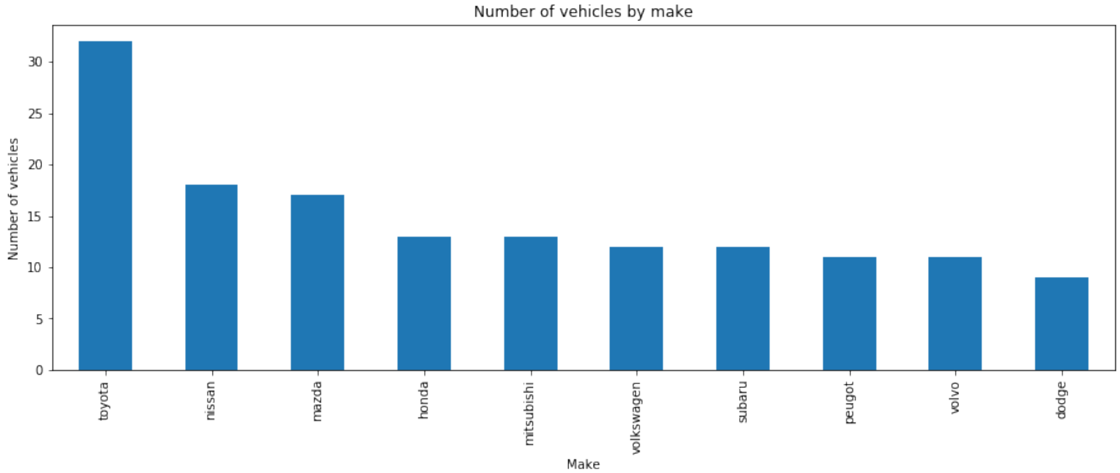
```
automobile[["normalized-losses","bore","stroke","horsepower","peak-rpm","price"]]=automobile[[
↪astype("float")
automobile.dtypes
```

[18]:
```
symboling              int64
normalized-losses    float64
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower           float64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

Vehicle make frequency diagram

[19]:
```
automobile.make.value_counts().nlargest(10).plot(kind='bar', figsize=(15,5))
plt.title("Number of vehicles by make")
plt.ylabel("Number of vehicles")
plt.xlabel("Make")
```

[19]:
```
Text(0.5, 0, 'Make')
```

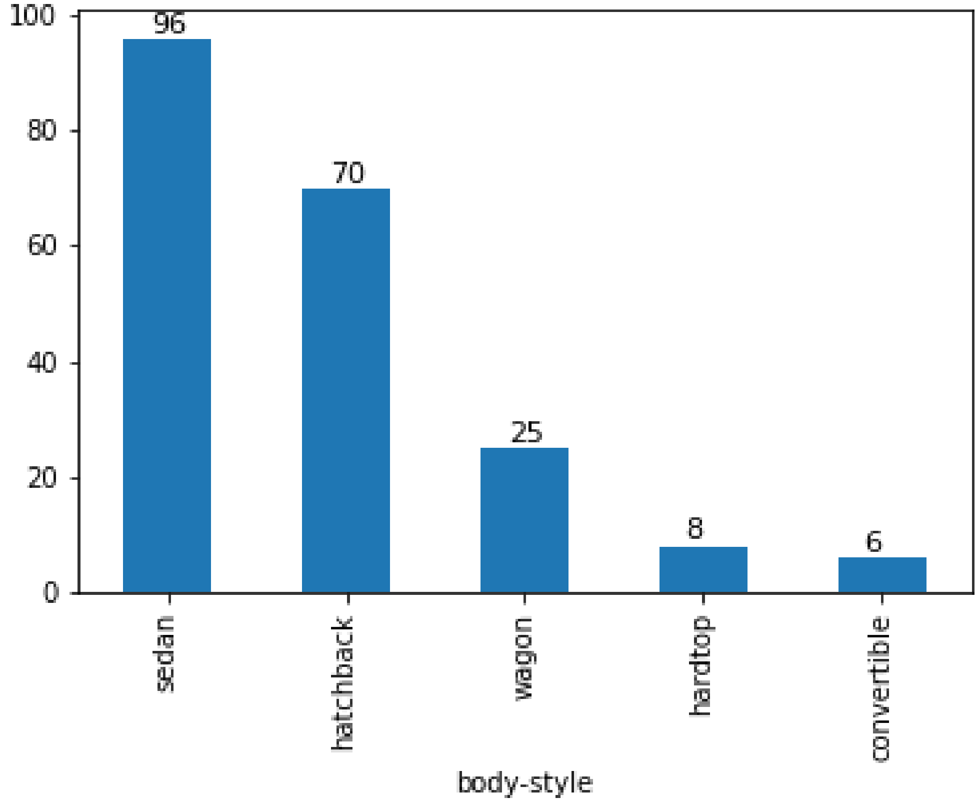


Insights:

1) Toyota is the make of the car which has most number of vehicles with more than 40% than the 2nd highest Nissan

Frequency of each car style

[20]:
```
cars_type = automobile.groupby(['body-style']).count()['make']
ax=cars_type.sort_values(ascending=False).plot.bar()
for p in ax.patches:
    ax.annotate(format(p.get_height()), (p.get_x()+0.15, p.
 ↪get_height()+1),fontsize=11)
```

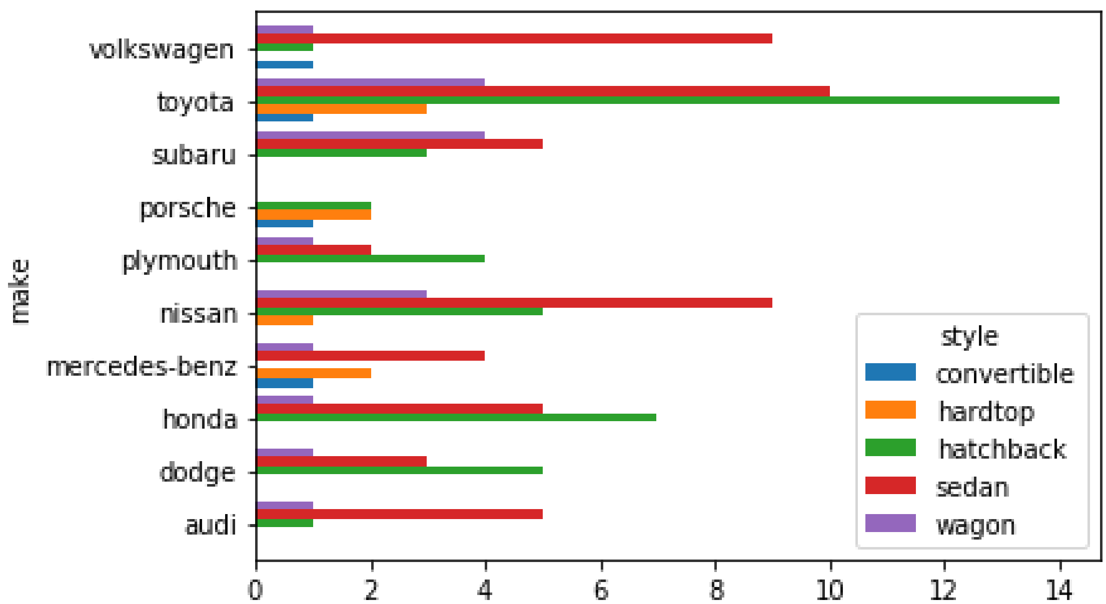


Insights:

1) Sedan is most popular car type. With almost 96 sedan cars made.

Let's see which car type the manufactures produce.

[21]:
```
a = automobile.groupby(['make','body-style']).count().reset_index()
a = a[['make','body-style','symboling']]
a.columns = ['make','style','count']
a = a.pivot('make','style','count')
a.dropna(thresh=3).plot.barh(width=0.85)
plt.show()
```

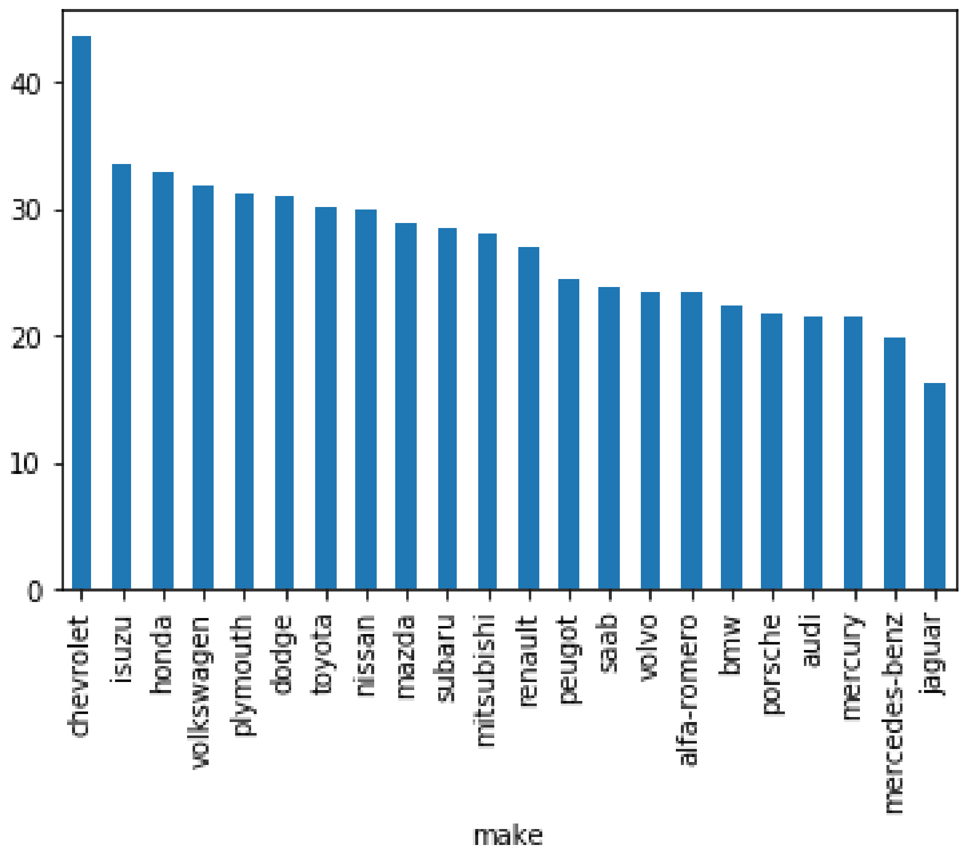


Insights:

1) Toyota makes hatchback car on large scale.
2) Also, Volkswagen and Nissan makes sedan cars a lot.

Mileage

[22]:
```
mileage = automobile.groupby(['make']).mean()
mileage['avg-mpg'] = ((mileage['city-mpg']+mileage['highway-mpg'])/2)
mileage['avg-mpg'].sort_values(ascending=False).plot.bar()
plt.show()
```



Insights:

1) Here we have calculated the avg-mlg according to the manufacturer by taking the average mileage of city and highway mpg. From the graph, we can see Chervolet giving the highest mileage and Jaguar the lowest mileage.

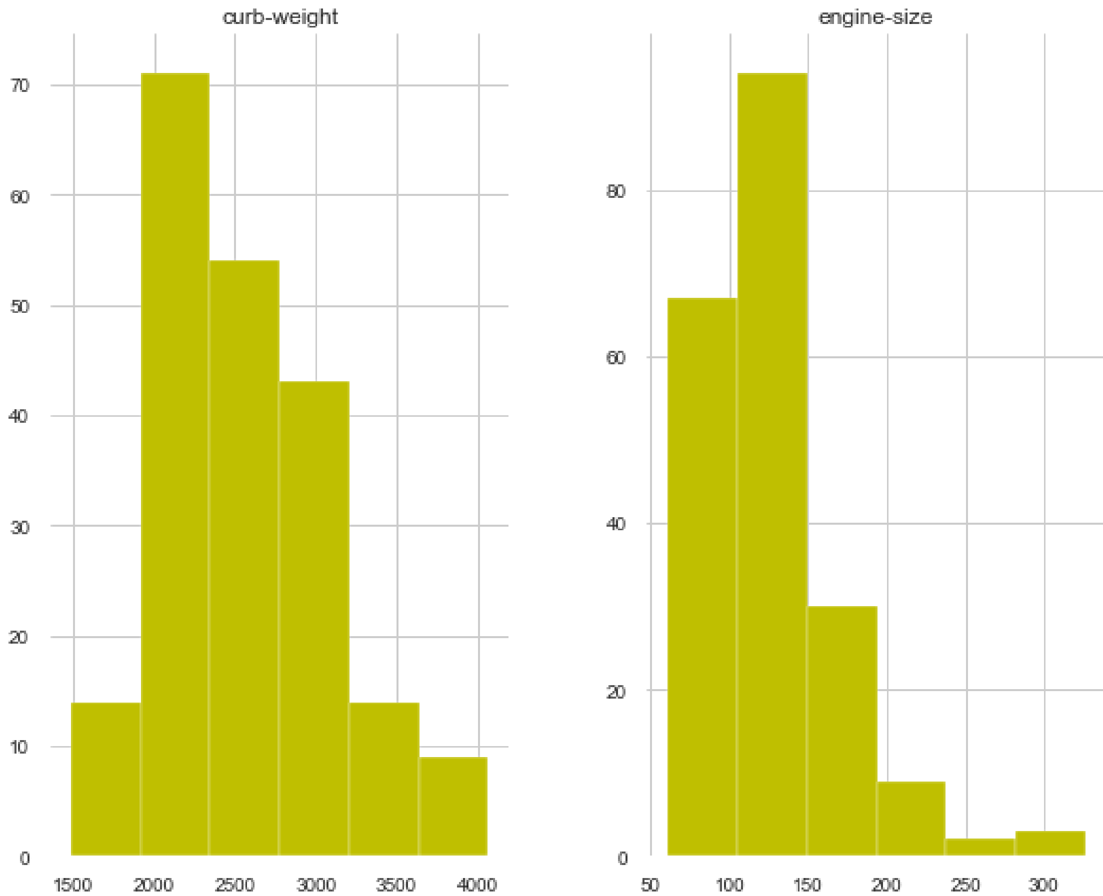
[22]:
```
automobile.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  164 non-null    float64
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       203 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
```

```
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               201 non-null    float64
 19  stroke             201 non-null    float64
 20  compression-ratio  205 non-null    float64
 21  horsepower         203 non-null    float64
 22  peak-rpm           203 non-null    float64
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              201 non-null    float64
dtypes: float64(11), int64(5), object(10)
memory usage: 41.8+ KB
```

[23]:
```
#plt.figure(figsize=(10,8))
automobile[['engine-size','curb-weight']].hist(figsize=(10,8),bins=6,color='Y')
plt.figure(figsize=(10,8))
plt.tight_layout()
plt.show()
```

```
/Users/anand/anaconda3/lib/python3.7/site-
packages/pandas/plotting/_matplotlib/hist.py:404: MatplotlibDeprecationWarning:
Support for uppercase single-letter colors is deprecated since Matplotlib 3.1
and will be removed in 3.3; please use lowercase instead.
  ax.hist(data[col].dropna().values, bins=bins, **kwds)
```



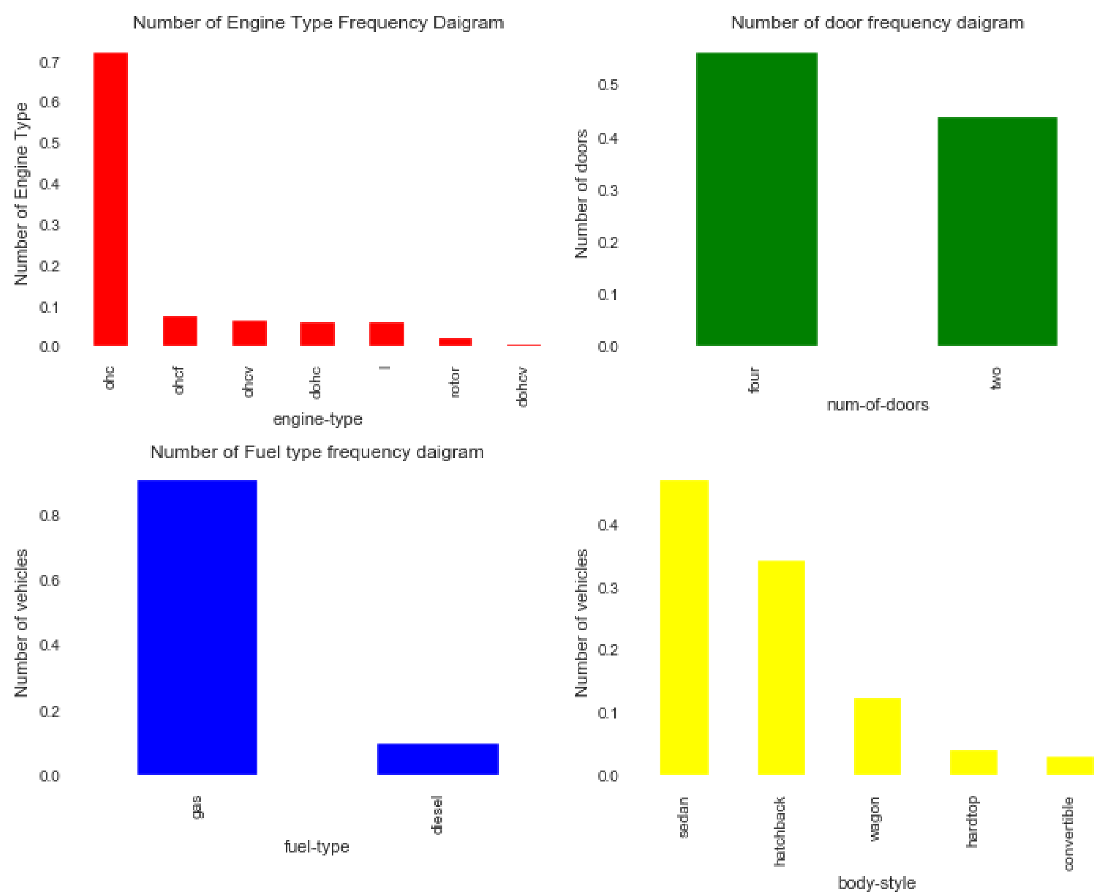

```
<Figure size 720x576 with 0 Axes>
```

Insights

1) Most of the car has a Curb Weight is in range 1900 to 3100.
2) The Engine size is in range 60 to 190.

[24]:
```
plt.figure(1)
plt.subplot(221)
automobile['engine-type'].value_counts(normalize=True).
↪plot(figsize=(10,8),kind='bar',color='red')
plt.title('Number of Engine Type Frequency Daigram')
plt.ylabel('Number of Engine Type')
plt.xlabel('engine-type')

plt.subplot(222)
automobile['num-of-doors'].value_counts(normalize=True).
↪plot(figsize=(10,8),kind='bar',color='green')
```

```
plt.title('Number of door frequency daigram')
plt.ylabel('Number of doors')
plt.xlabel('num-of-doors')

plt.subplot(223)
automobile['fuel-type'].value_counts(normalize=True).
↪plot(figsize=(10,8),kind='bar',color='blue')
plt.title('Number of Fuel type frequency daigram')
plt.ylabel('Number of vehicles')
plt.xlabel('fuel-type')


plt.subplot(224)
automobile['body-style'].value_counts(normalize=True).
↪plot(figsize=(10,8),kind='bar',color='yellow')
plt.ylabel('Number of vehicles')
plt.xlabel('body-style')

plt.tight_layout()
plt.show()
```



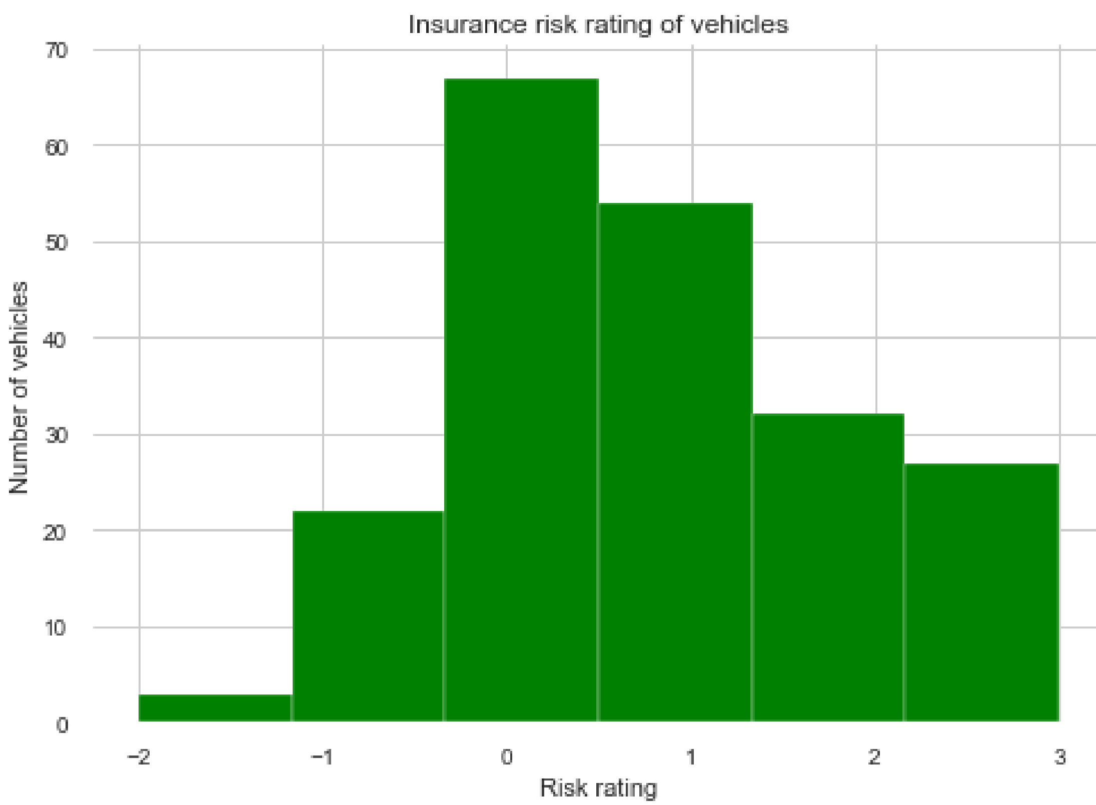
Insights

1) More than 70 percent of the vehicles have dhc engine type
2) Almost 57 percent cars have four doors
3) Gas is preferred by 85 percent of the vehicle owner.
4) Most produced vehicle is of sedan around 48 percent.

Insurance risk ratings Histogram

[28]:
```
automobile.symboling.hist(bins=6,color='green');
plt.title("Insurance risk rating of vehicles")
plt.ylabel("Number of vehicles")
plt.xlabel("Risk rating")
```

[28]: `Text(0.5, 0, 'Risk rating')`

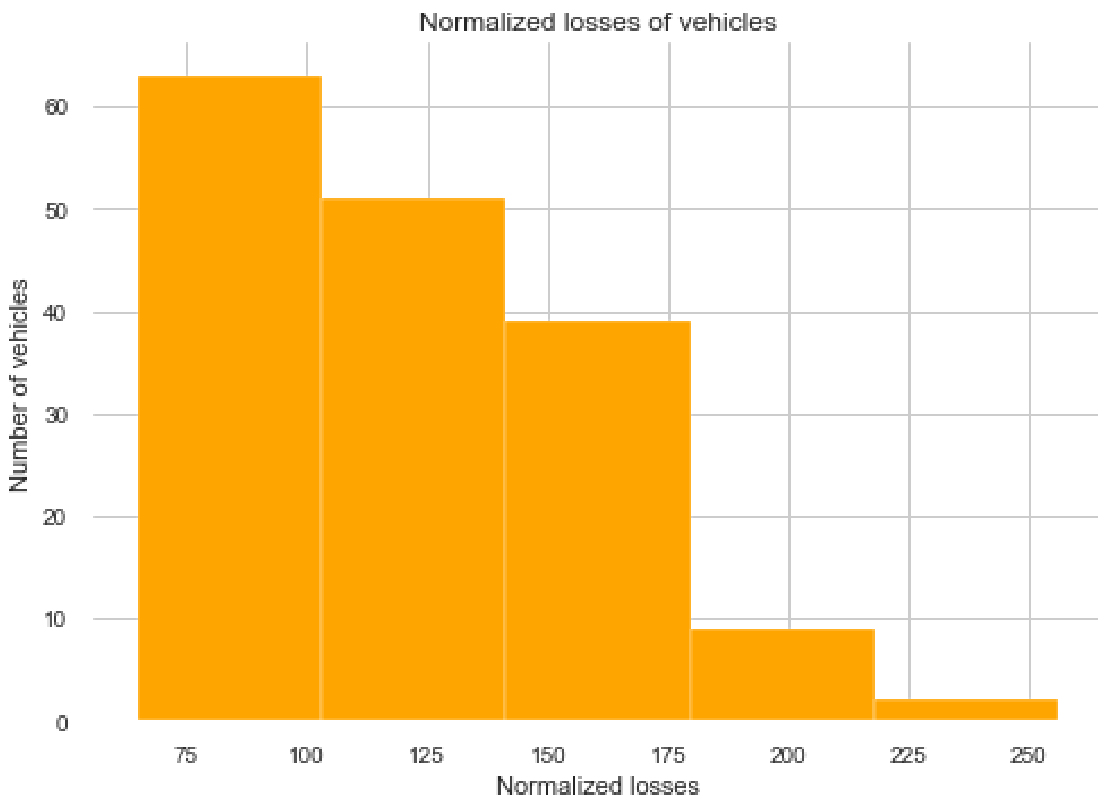


Symboling or the insurance risk rating have the ratings between -3 and 3 however for our dataset it starts from -2. There are more cars in the range of 0 and 1.

Normalized losses histogram

[29]:
```
automobile['normalized-losses'].hist(bins=5,color='orange');
plt.title('Normalized losses of vehicles')
plt.ylabel('Number of vehicles')
plt.xlabel('Normalized losses')
```

[29]:
```
Text(0.5, 0, 'Normalized losses')
```

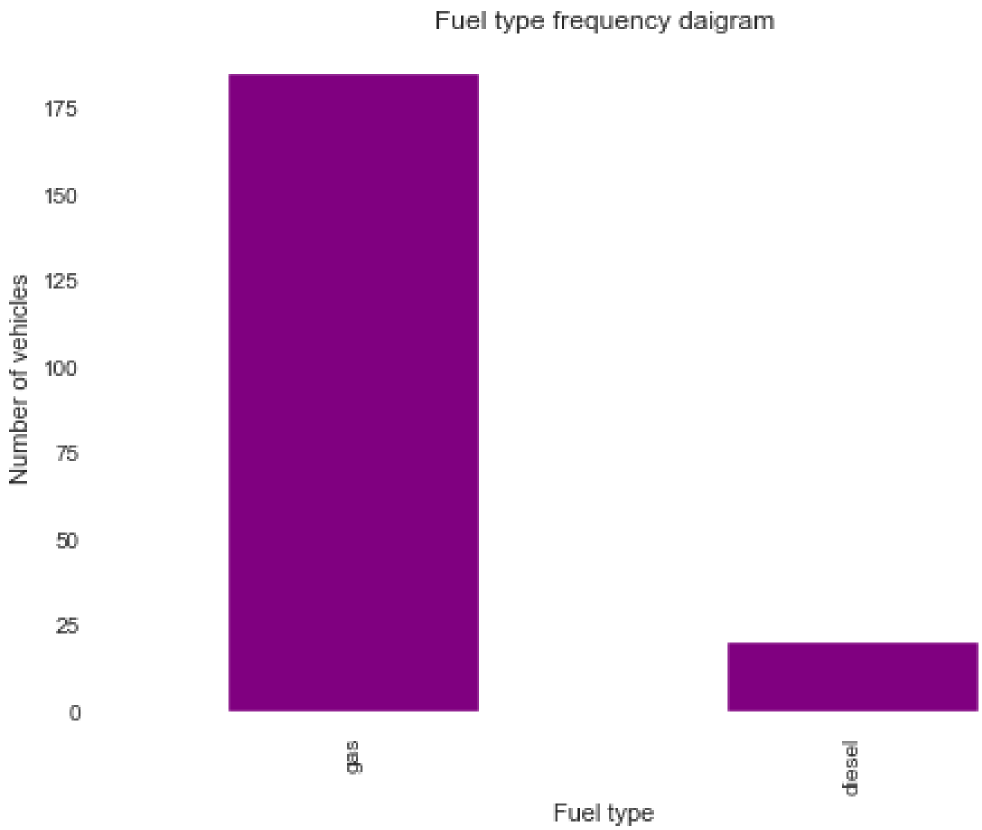


Normalized losses which is the average loss payment per insured vehicle year is has more number of cars in the range between 65 and 150.

Fuel type bar chart

[30]:
```
automobile['fuel-type'].value_counts().plot(kind='bar',color='purple')
plt.title('Fuel type frequency daigram')
plt.ylabel('Number of vehicles')
plt.xlabel('Fuel type')
```

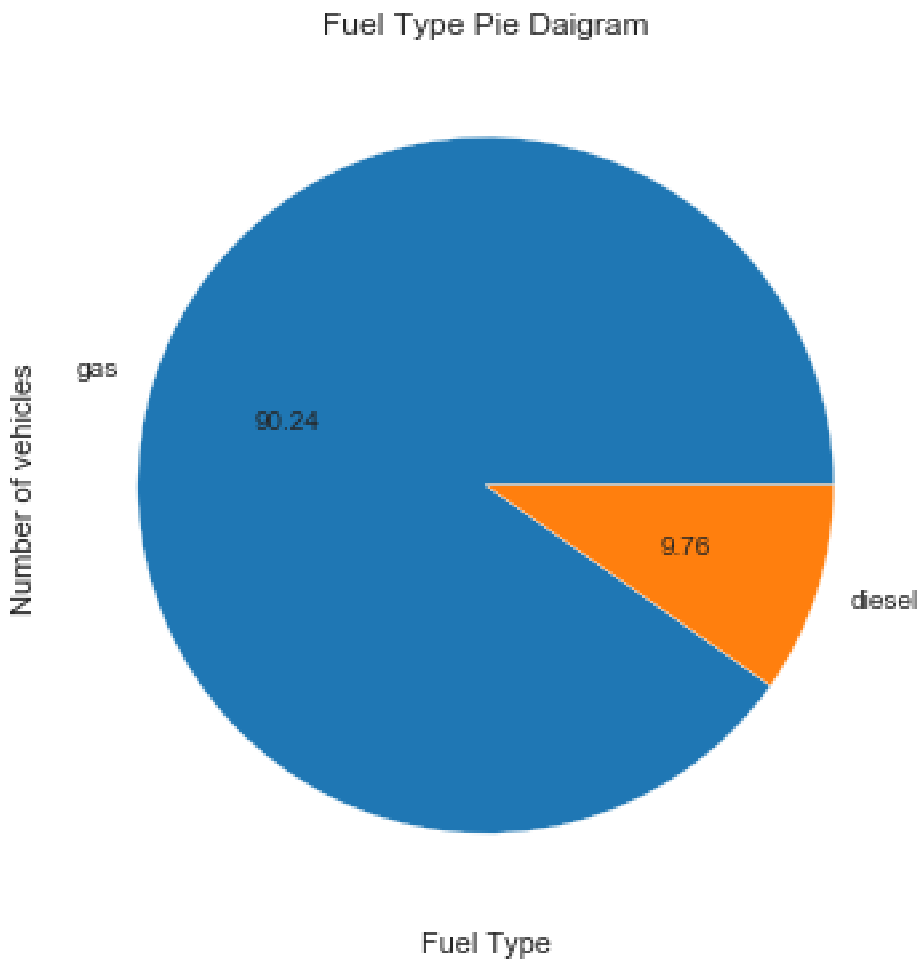
[30]:
```
Text(0.5, 0, 'Fuel type')
```



From the above graph, we can see most of the vehicles uses gas as fuel.

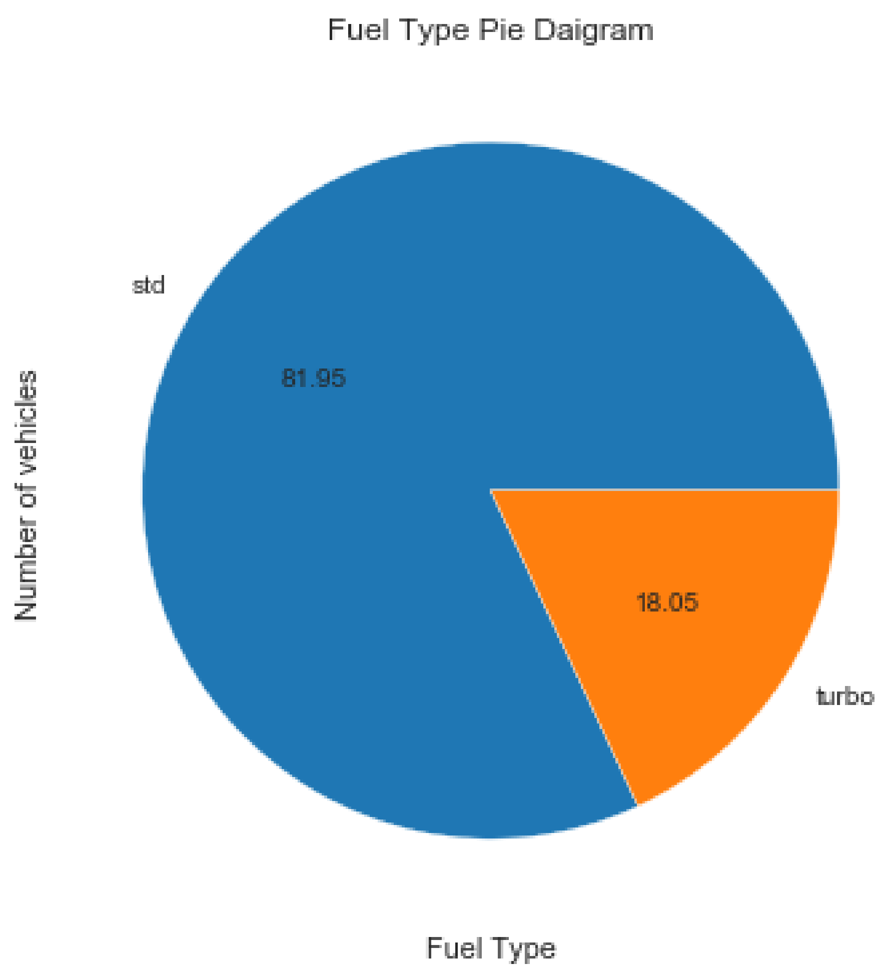
[33]:
```
# Fuel Type Pie chart
automobile['fuel-type'].value_counts().plot.pie(figsize=(6,6), autopct='%.2f')
plt.title('Fuel Type Pie Daigram')
plt.ylabel('Number of vehicles')
plt.xlabel('Fuel Type')
```

[33]:
```
Text(0.5, 0, 'Fuel Type')
```



[34]:
```
automobile['aspiration'].value_counts().plot.pie(figsize=(6,6), autopct='%.2f')
plt.title('Fuel Type Pie Daigram')
plt.ylabel('Number of vehicles')
plt.xlabel('Fuel Type')
```

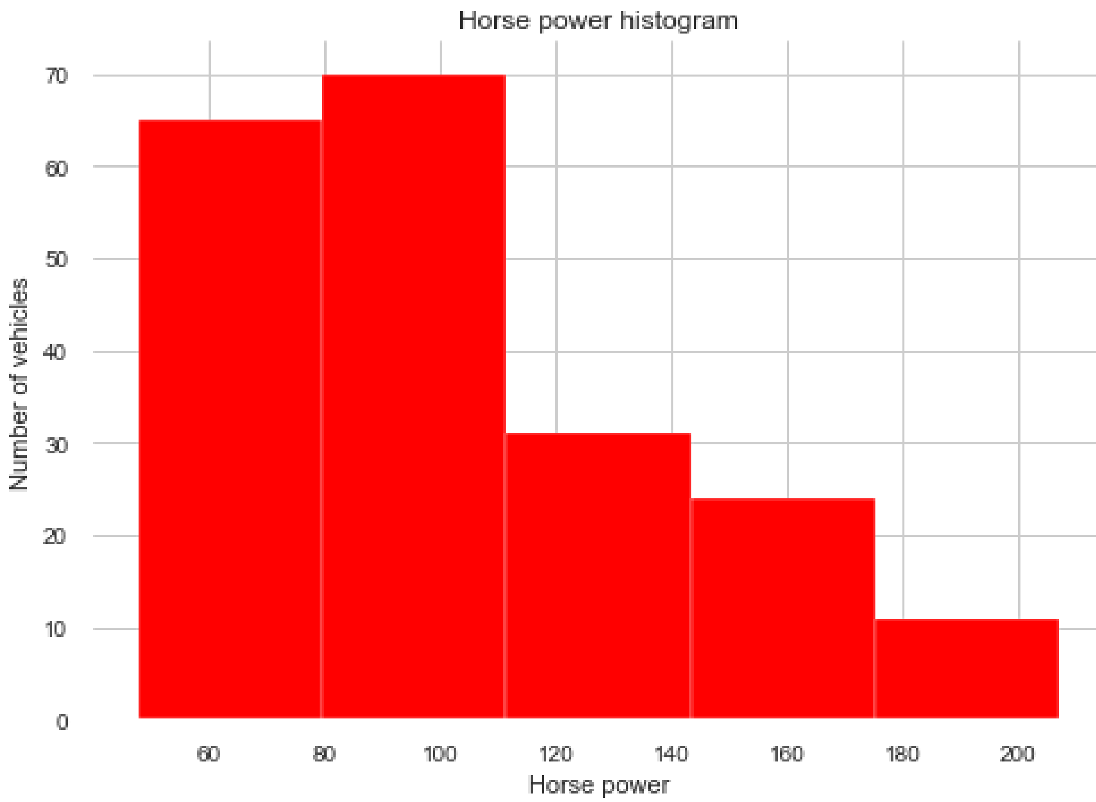
[34]: Text(0.5, 0, 'Fuel Type')



From the above pie charts we can see that, most of the cars use gas as fuel and are mostly standard aspiration.

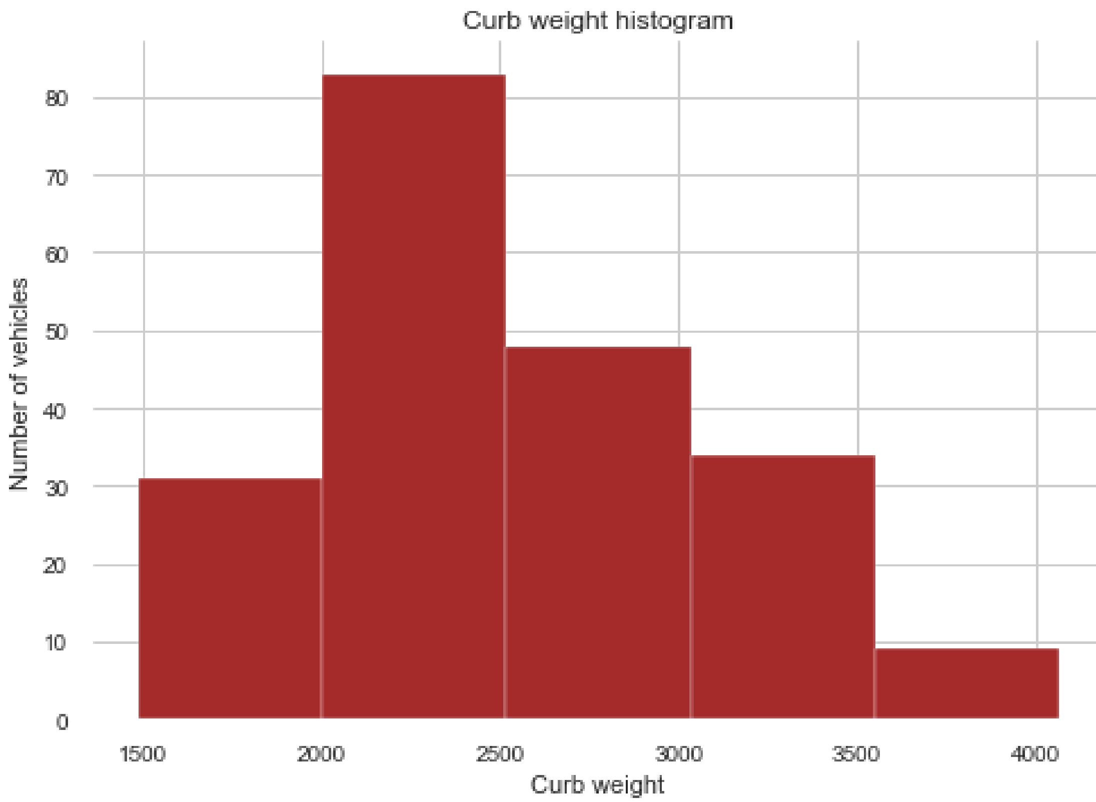
Horse power histogram

[35]:
```
automobile.horsepower[np.abs(automobile.horsepower-automobile.horsepower.
→mean())<=(3*automobile.horsepower.std())].hist(bins=5,color='red');
plt.title("Horse power histogram")
plt.ylabel('Number of vehicles')
plt.xlabel('Horse power');
```



Curb weight histogram

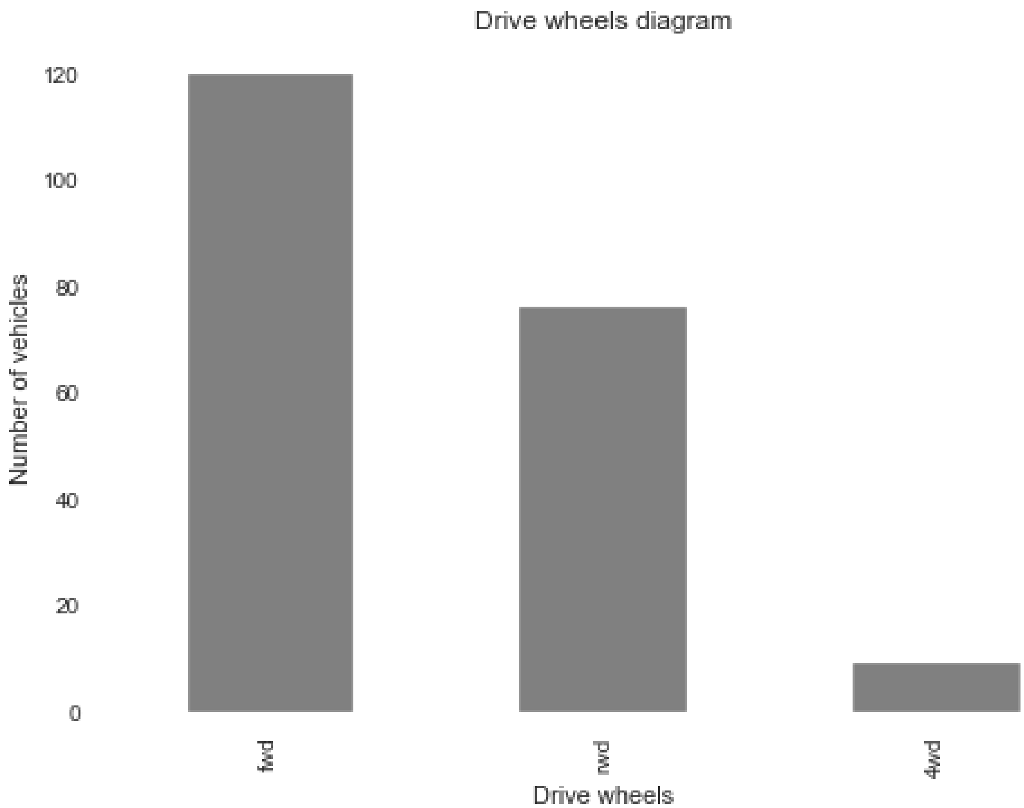
[36]:
```
automobile['curb-weight'].hist(bins=5,color='brown');
plt.title("Curb weight histogram")
plt.ylabel('Number of vehicles')
plt.xlabel('Curb weight');
```



Curb weight of the cars are distributed between 1500 and 4000 approximately

Drive wheels bar chart

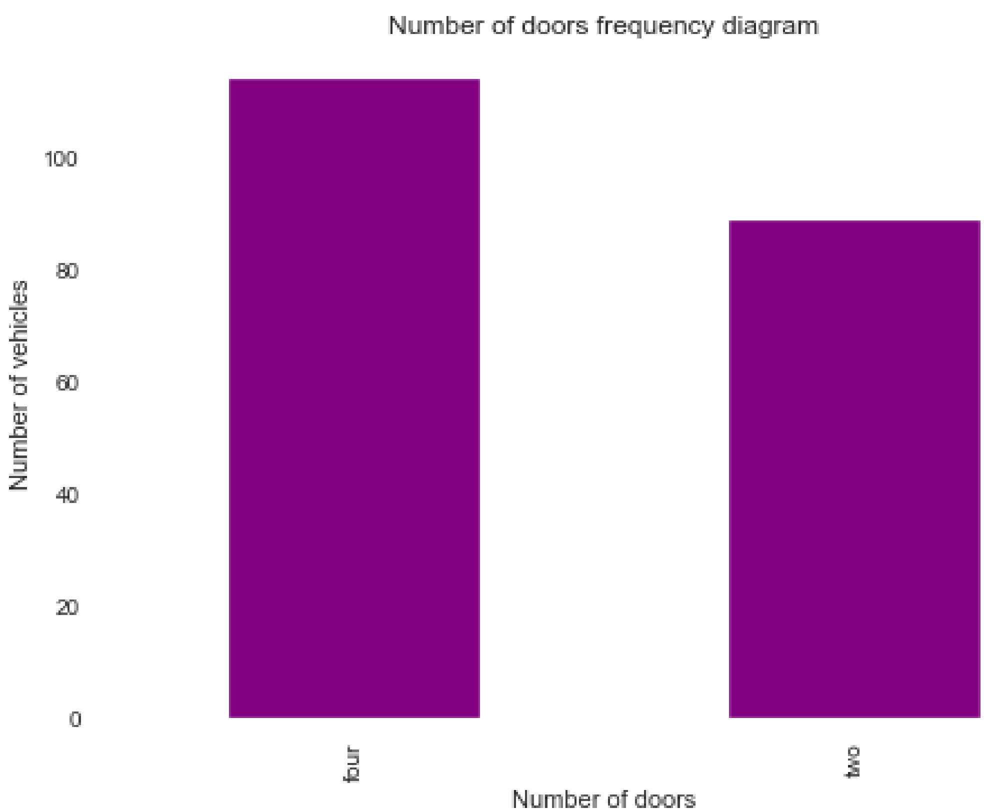
[37]:
```
automobile['drive-wheels'].value_counts().plot(kind='bar',color='grey')
plt.title("Drive wheels diagram")
plt.ylabel('Number of vehicles')
plt.xlabel('Drive wheels');
```



For drive wheels, front wheel drive has most number of cars followed by rear wheel and four wheel. There are very less number of cars for four wheel drive.

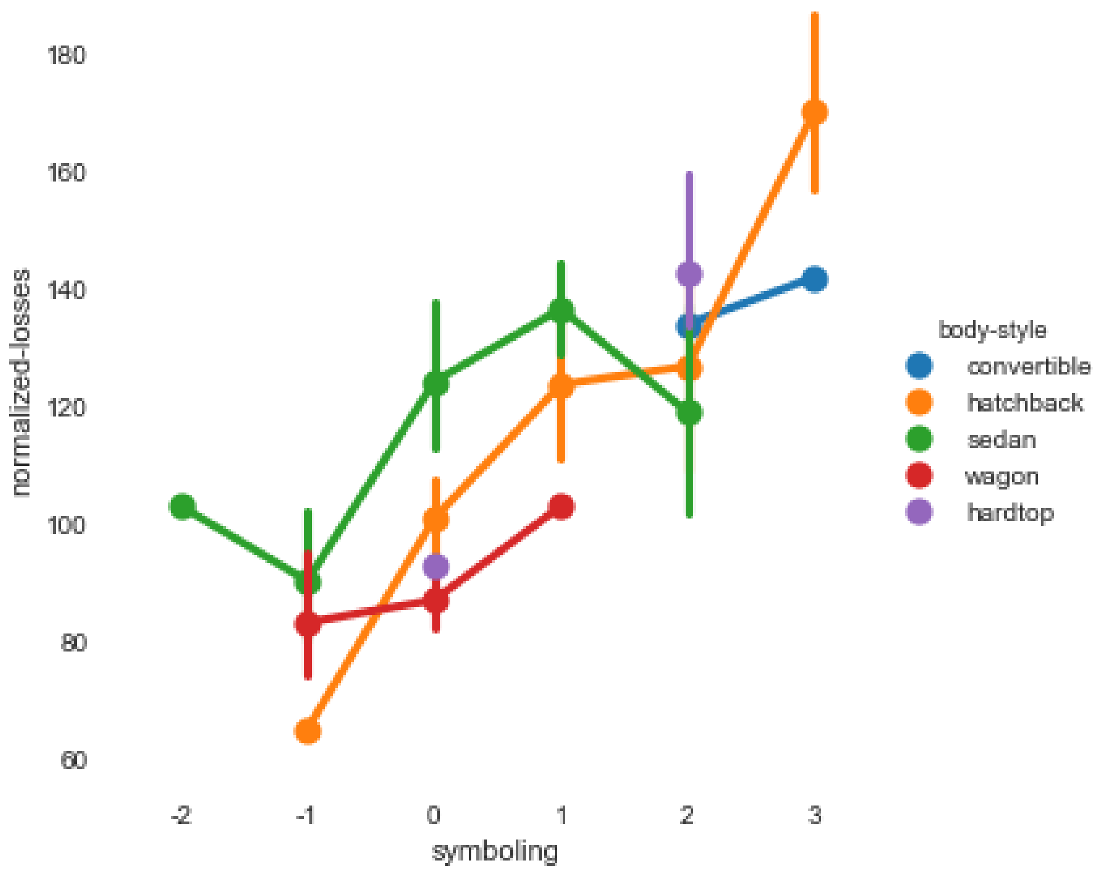
Number of doors bar chart

[38]:
```
automobile['num-of-doors'].value_counts().plot(kind='bar',color='purple')
plt.title("Number of doors frequency diagram")
plt.ylabel('Number of vehicles')
plt.xlabel('Number of doors');
```



[25]:
```
sns.catplot(data=automobile, y="normalized-losses", x="symboling" ,␣
 ↪hue="body-style" ,kind="point")
```

[25]:
```
<seaborn.axisgrid.FacetGrid at 0x7fe01b0a8590>
```

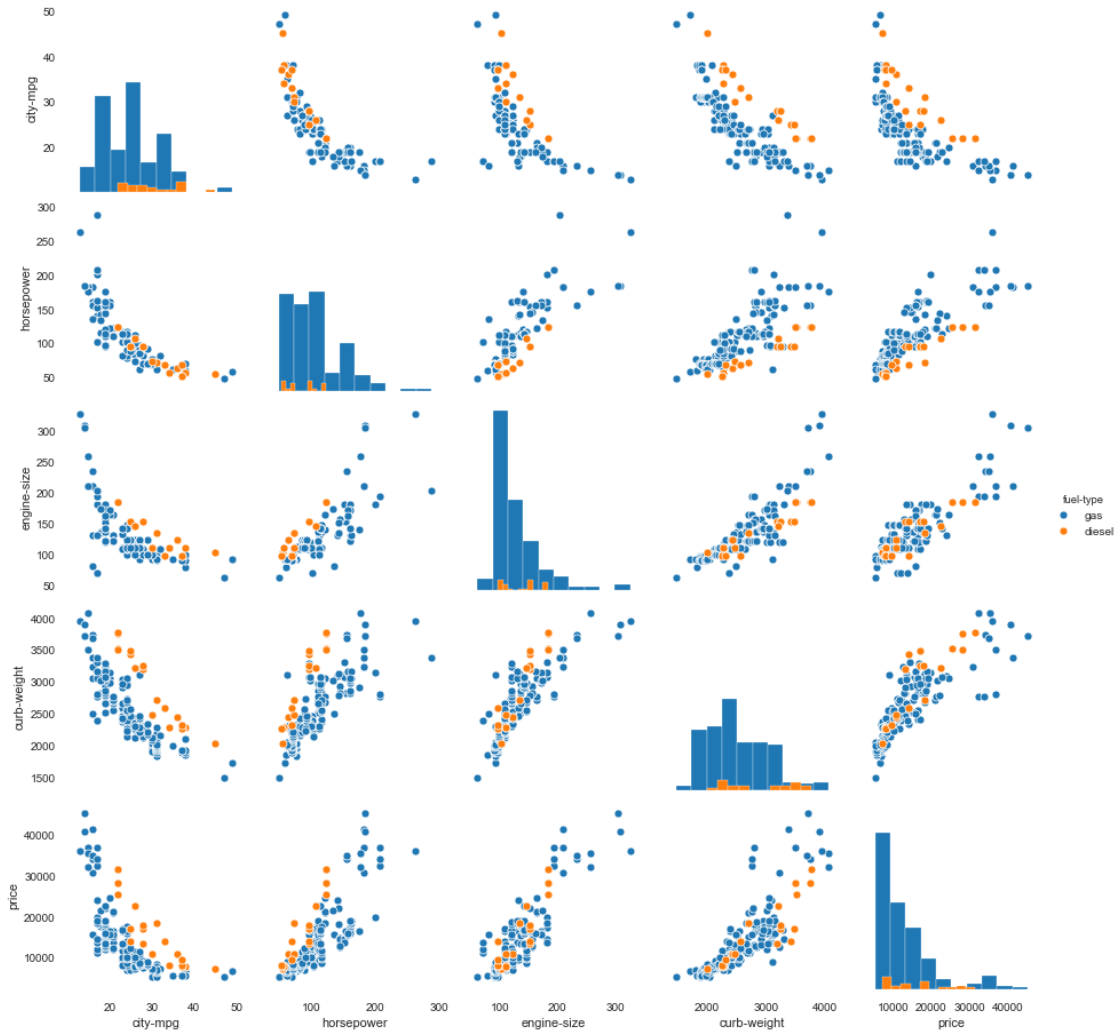


Losses findings:

Note :- here +3 means risky vehicle and -2 means safe vehicle

1) Increased in risk rating linearly increases in normalised losses in vehicle.
2) covertible car and hardtop car has mostly losses with risk rating above 0.
3) hatchback cars has highest losses at risk rating 3.
4) sedan and Wagon car has losses even in less risk (safe)rating

[26]:
```
g = sns.pairplot(automobile[["city-mpg", "horsepower", "engine-size",␣
 ↪"curb-weight","price", "fuel-type"]], hue="fuel-type", diag_kind="hist")
```



Insights

1) Vehicle Mileage decrease as increase in Horsepower , engine-size, Curb Weight
2) As horsepower increase the engine size increases.
3) Curbweight increases with the increase in Engine Size
4) engine size and curb-weight is positively co realted with price.
5) city-mpg is negatively corelated with price as increase horsepower reduces the mileage

[39]:
```
corr = automobile.corr()
sns.set_context("notebook", font_scale=1.0, rc={"lines.linewidth": 2.5})
plt.figure(figsize=(13,7))
a = sns.heatmap(corr, annot=True, fmt='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
```

```
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```

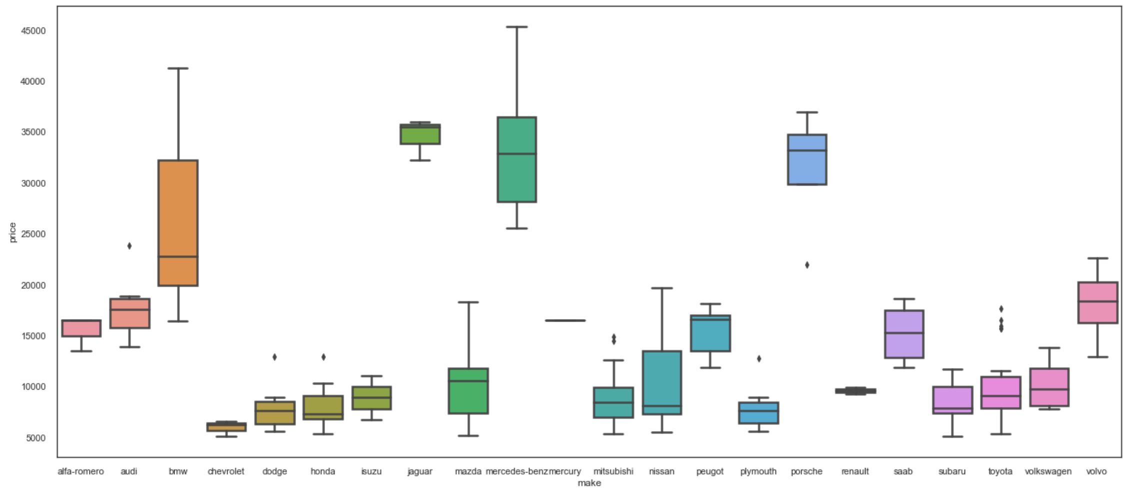
Insights

1) curb-weight and engine-size are positively correlated to the price.
2) Curb-weight is mostly correlated with engine-size,lenghth and width of the car.
3) Wheel-base is highly correlated with lenghth and width of the car.
4) Symboling and normalized car are correlated than the other fields

0.2 Bivariate Analysis

Boxplot of Price and make

[40]:
```
plt.rcParams['figure.figsize']=(23,10)
ax = sns.boxplot(x="make", y="price", data=automobile)
```

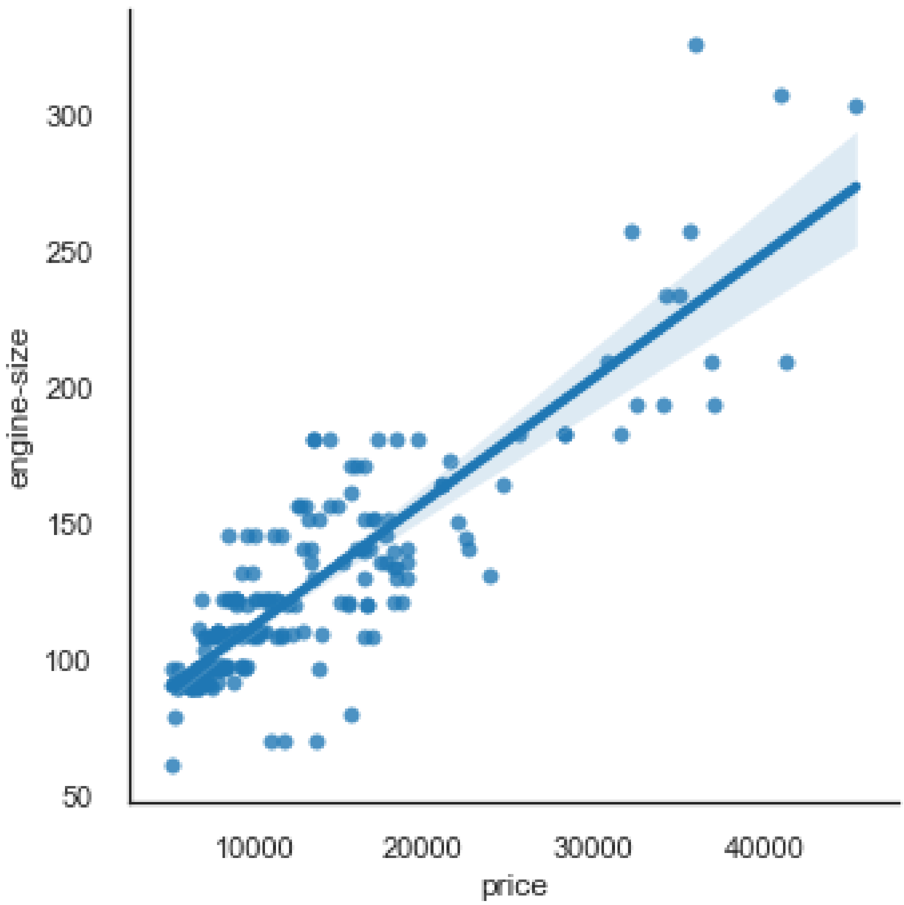


Findings:

1) The most expensive car is Mercedes-Benz and least expensive is Chervolet
2) The premium car brands is BMW, Jaguar, Mercedes and Porshe which costs more than 20000.
3) Less expensive cars or affordable cars costing less than 10000 are Chervolet, Dodge, Honda, Mitsubishi, Plymouth and Subaru.
4) Rest of the midarange cars are in between 10000 to 20000 which has highest number of cars.

Price vs Engine size

[41]:
```
g = sns.lmplot('price', 'engine-size', automobile)
```

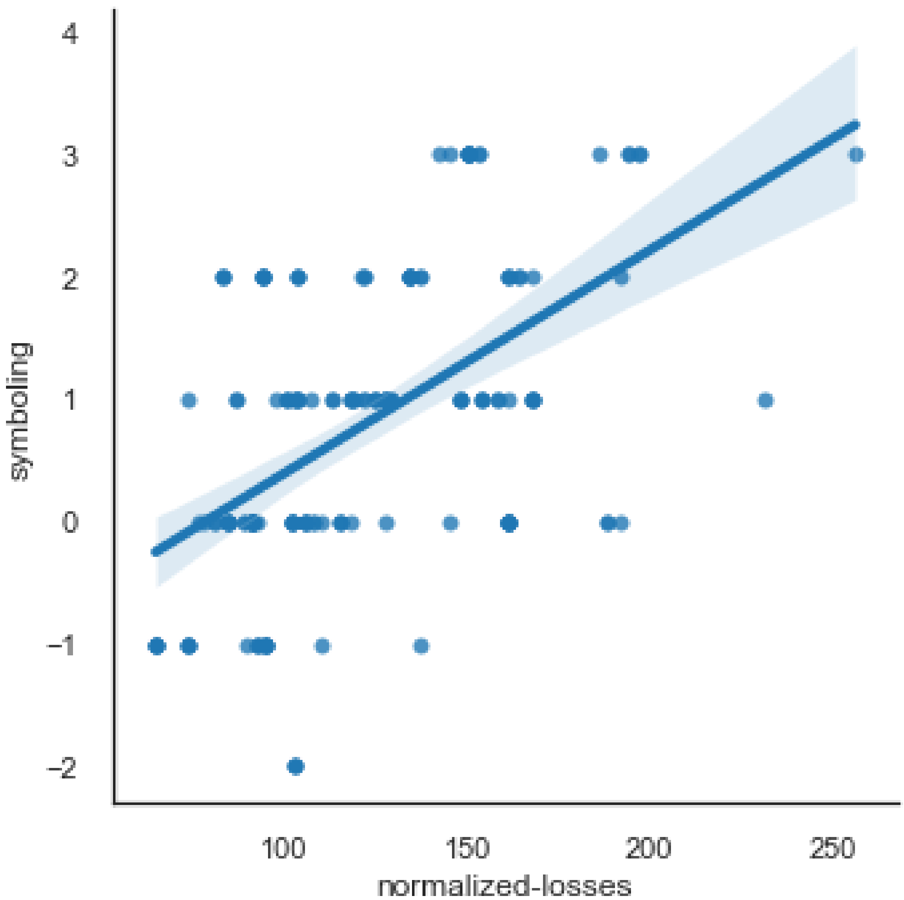


Findings:

The more the engine size more the price.

Normalized Losses VS Symboling

[42]:
```
g = sns.lmplot('normalized-losses', 'symboling', automobile);
```

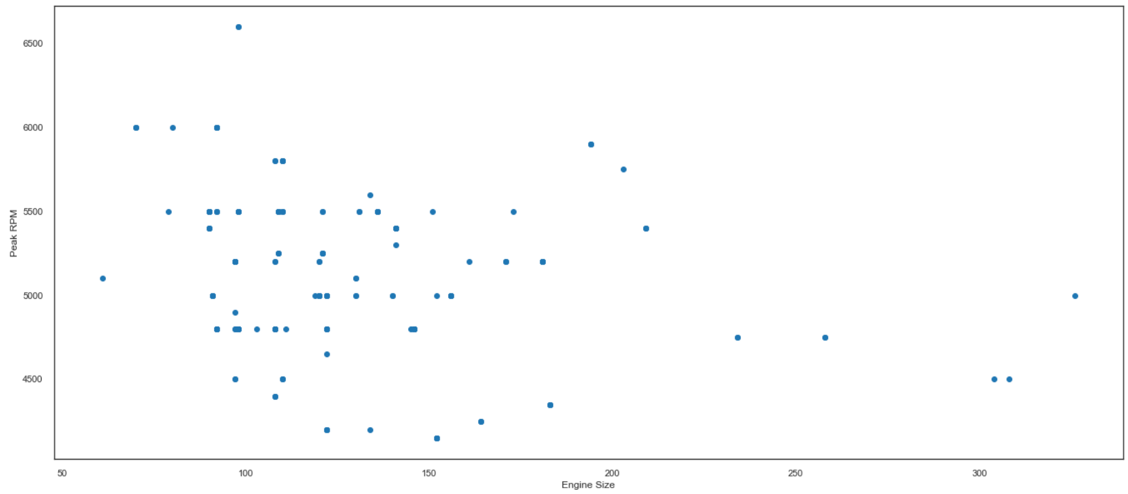


From the scatter plot we can clearly see that, lesser the rating of the car lesser the normalized loss. We can say that negative ratings are better for the car which has lesser losses.

Engine-Size VS Peak-RPM

[43]:
```
plt.scatter(automobile['engine-size'], automobile['peak-rpm'])
plt.xlabel('Engine Size')
plt.ylabel('Peak RPM')
```

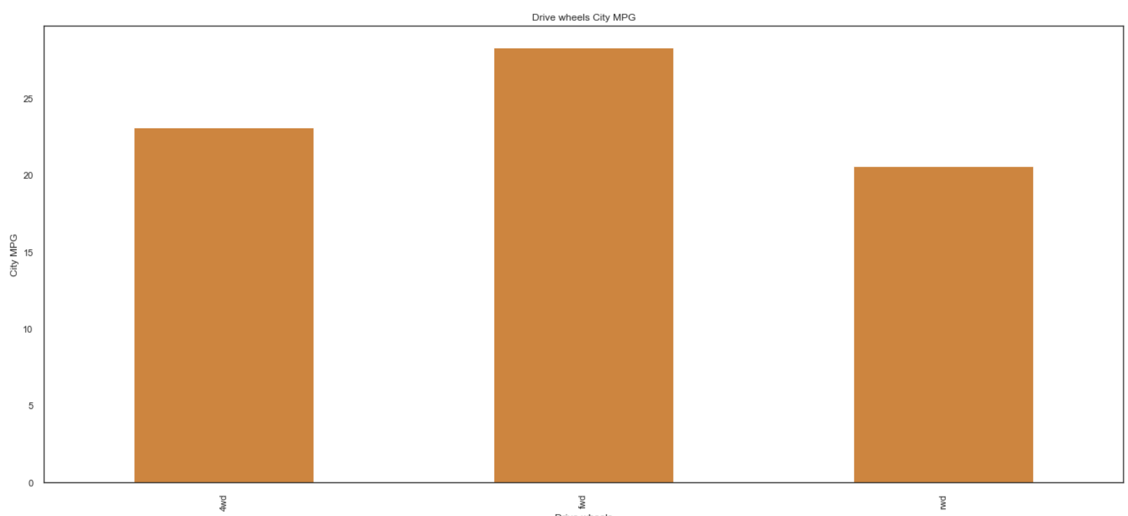
[43]: Text(0, 0.5, 'Peak RPM')



Greater the engine size lesser the Peak RPM.

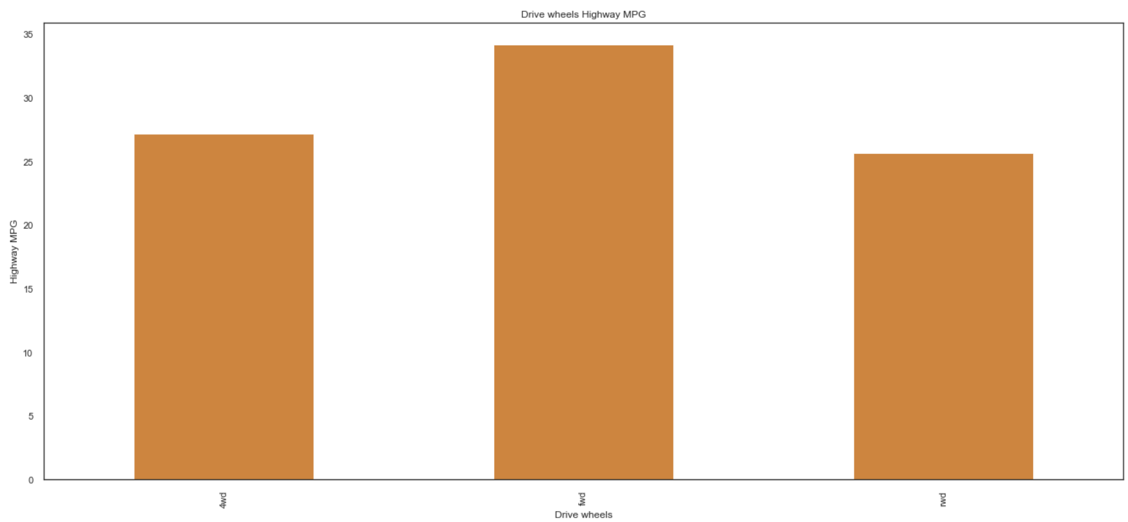
Drive wheels and City MPG bar chart

[53]:
```
automobile.groupby('drive-wheels')['city-mpg'].mean().plot(kind='bar', color =␣
↪'peru');
plt.title("Drive wheels City MPG")
plt.ylabel('City MPG')
plt.xlabel('Drive wheels');
```



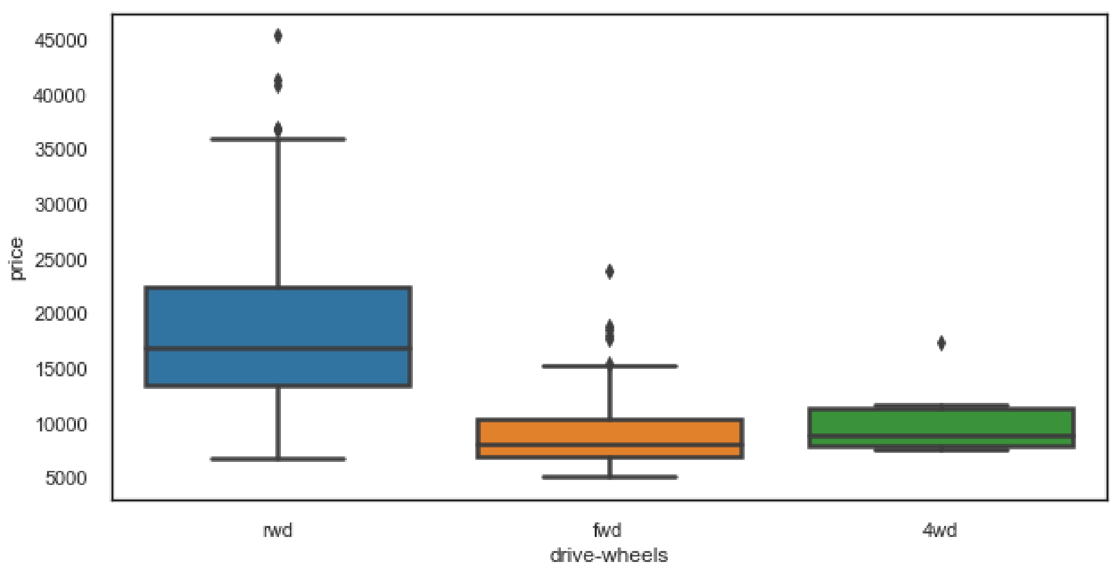
Drive wheels and Highway MPG bar chart

[54]:
```
automobile.groupby('drive-wheels')['highway-mpg'].mean().plot(kind='bar', color
↪= 'peru');
plt.title("Drive wheels Highway MPG")
plt.ylabel('Highway MPG')
plt.xlabel('Drive wheels');
```



Boxplot of Drive wheels and Price

[55]:
```
plt.rcParams['figure.figsize']=(10,5)
ax = sns.boxplot(x="drive-wheels", y="price", data=automobile)
```

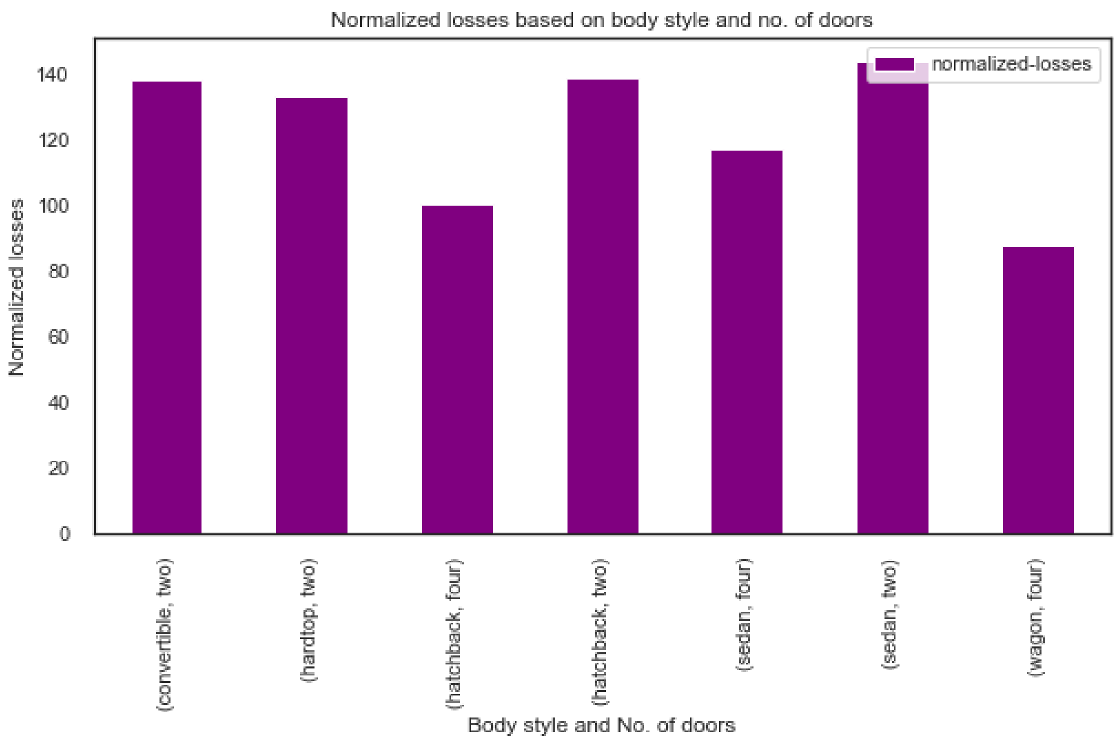


Findings: It's very evident that the Rear wheel drive cars are most expensive and front wheel is least expensive cars. Four wheel drive cars are little higher than the front wheel drive cars.

There is very less number of four wheel drive cars in our dataset so this picture might not be very accurate.

Normalized losses based on body style and no. of doors

[56]:
```
pd.pivot_table(automobile,index=['body-style','num-of-doors'],␣
↪values='normalized-losses').plot(kind='bar',color='purple')
plt.title("Normalized losses based on body style and no. of doors")
plt.ylabel('Normalized losses')
plt.xlabel('Body style and No. of doors');
```



Findings:

As we understand the normalized loss which is the average loss payment per insured vehicle is calculated with many features of the cars which includes body style and no. of doors.

Normalized losses are distributed across different body style but the two door cars has more number of losses than the four door cars.

Analysis of the data set provides

1) How the data set are distributed
2) Correlation between different fields and how they are related

3) Normalized loss of the manufacturer
4) Symboling : Cars are initially assigned a risk factor symbol associated with its price
5) Mileage : Mileage based on City and Highway driving for various make and attributes
6) Price : Factors affecting Price of the Automobile.
7) Importance of drive wheels and curb weight