Automobile Exploratory Data Analysis

May 18, 2024

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

[2]:
```python
# Loading the data from Automobile_price_data_raw csv file into data frame and
↪viewing top 10 rows.
data=pd.read_csv("E:\Python Projects\Automobile\Automobile_price_data _raw.csv")
data.head()
```

[2]:
```
   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     3.4              10.0        102      5500       24
4         mpfi  3.19     3.4               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]
```

[3]:
```
# Display Total Number of rows and columns
data.shape
```

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

[4]:
```
# Display Name of entitys or columns
data.columns
```

[4]:
```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

[5]:
```
# summary about the data type of entity,Number of rows and column, count of not␣
↪null value.
data.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
```

[6]:
```
#checking for count of null values in a column
data.isnull().sum()
```

[6]:
```
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
```

[7]:
```
# look at the descriptive statistics of the data
data.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
```

[8]:
```python
# look at the descriptive statistics of categorical data
data.describe(include="object")
```

[8]:
```
       normalized-losses    make fuel-type aspiration num-of-doors body-style  \
count                205     205       205        205          205        205
unique                52      22         2          2            3          5
top                    ?  toyota       gas        std         four      sedan
freq                  41      32       185        168          114         96

       drive-wheels engine-location engine-type num-of-cylinders fuel-system  \
count           205             205         205              205         205
unique            3               2           7                7           8
top             fwd           front         ohc             four        mpfi
freq            120             202         148              159          94

        bore stroke horsepower peak-rpm price
count    205    205        205      205   205
unique    39     37         60       24   187
top     3.62    3.4         68     5500     ?
freq      23     20         19       37     4
```

Thus we can we some columns having numeric data have object data type such as = normalized-losses,bore,stroke,horespower,peak-rpm,price. Lets analyse this columns and change their data types

[9]:
```python
# Look for all unique values in normalized-losses column
data["normalized-losses"].unique()
```

[9]:
```
array(['?', '164', '158', '192', '188', '121', '98', '81', '118', '148',
       '110', '145', '137', '101', '78', '106', '85', '107', '104', '113',
       '150', '129', '115', '93', '142', '161', '153', '125', '128',
       '122', '103', '168', '108', '194', '231', '119', '154', '74',
       '186', '83', '102', '89', '87', '77', '91', '134', '65', '197',
       '90', '94', '256', '95'], dtype=object)
```

[10]:
```
# Replace "?" with np.nan values and then Nan values.
data["normalized-losses"] = data["normalized-losses"].replace("?",np.NaN)
```

[11]:
```
data["normalized-losses"] = pd.
↪to_numeric(data["normalized-losses"],errors="coerce")
```

[12]:
```
normalized_osses_summary = {"mean" : data["normalized-losses"].mean(),
           "median" : data["normalized-losses"].median(),
           "mode" : data["normalized-losses"].mode()}

normalized_osses_summary
```

[12]:
```
{'mean': 122.0,
 'median': 115.0,
 'mode': 0    161.0
 Name: normalized-losses, dtype: float64}
```

As "?" have more number of counts we can not use median to fill the position. we can use the mean

[13]:
```
data["normalized-losses"] = data["normalized-losses"].
↪fillna(data["normalized-losses"].mean())
```

[14]:
```
# Look for all unique values in bore column
data["bore"].unique()
```

[14]:
```
array(['3.47', '2.68', '3.19', '3.13', '3.5', '3.31', '3.62', '2.91',
       '3.03', '2.97', '3.34', '3.6', '2.92', '3.15', '3.43', '3.63',
       '3.54', '3.08', '?', '3.39', '3.76', '3.58', '3.46', '3.8', '3.78',
       '3.17', '3.35', '3.59', '2.99', '3.33', '3.7', '3.61', '3.94',
       '3.74', '2.54', '3.05', '3.27', '3.24', '3.01'], dtype=object)
```

[15]:
```
data["bore"].replace("?",np.NaN,inplace=True)
```

[16]:
```
data["bore"] = pd.to_numeric(data["bore"],errors="coerce")
```

[17]:
```
bore_summary = {"mean" : data["bore"].mean(),
           "median" : data["bore"].median(),
           "mode" : data["bore"].mode()}

bore_summary
```

[17]:
```
{'mean': 3.3297512437810943,
 'median': 3.31,
 'mode': 0    3.62
 Name: bore, dtype: float64}
```

Replaces missing values with the median of the available values. It is more robust than mean imputation, especially for data with outliers or a non-normal distribution.

[18]:
```
data["bore"] = data["bore"].fillna(data["bore"].median())
data["bore"]
```

[18]:
```
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
```

[19]:
```
# Look for all unique values in stroke column
data["stroke"].unique()
```

[19]:
```
array(['2.68', '3.47', '3.4', '2.8', '3.19', '3.39', '3.03', '3.11',
       '3.23', '3.46', '3.9', '3.41', '3.07', '3.58', '4.17', '2.76',
       '3.15', '?', '3.16', '3.64', '3.1', '3.35', '3.12', '3.86', '3.29',
       '3.27', '3.52', '2.19', '3.21', '2.9', '2.07', '2.36', '2.64',
       '3.08', '3.5', '3.54', '2.87'], dtype=object)
```

[20]:
```
data["stroke"].replace("?",np.NaN,inplace=True)
data["stroke"] = pd.to_numeric(data["stroke"],errors="coerce")
```

[21]:
```
stroke_summary = {"mean" : data["stroke"].mean(),
           "median" : data["stroke"].median(),
           "mode" : data["stroke"].mode()}

stroke_summary
```

[21]:
```
{'mean': 3.255422885572139,
 'median': 3.29,
 'mode': 0    3.4
 Name: stroke, dtype: float64}
```

[22]:
```
data["stroke"] = data["stroke"].fillna(data["stroke"].median())
data["stroke"]
```

[22]:
```
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
```

[23]:
```
# Look for all unique values in horsepower column
data["horsepower"].unique()
```

[23]:
```
array(['111', '154', '102', '115', '110', '140', '160', '101', '121',
       '182', '48', '70', '68', '88', '145', '58', '76', '60', '86',
       '100', '78', '90', '176', '262', '135', '84', '64', '120', '72',
       '123', '155', '184', '175', '116', '69', '55', '97', '152', '200',
       '95', '142', '143', '207', '288', '?', '73', '82', '94', '62',
       '56', '112', '92', '161', '156', '52', '85', '114', '162', '134',
       '106'], dtype=object)
```

[24]:
```
data["horsepower"].replace("?",np.NaN,inplace=True)
data["horsepower"] = pd.to_numeric(data["horsepower"],errors="coerce")
```

[25]:
```
summary = {"mean" : data["horsepower"].mean(),
           "median" : data["horsepower"].median(),
           "mode" : data["horsepower"].mode()}

summary
```

[25]:
```
{'mean': 104.25615763546799,
 'median': 95.0,
 'mode': 0    68.0
 Name: horsepower, dtype: float64}
```

Replaces missing values with the median of the available values. It is more robust than mean imputation, especially for data with outliers or a non-normal distribution.

[26]:
```
data["horsepower"] = data["horsepower"].fillna(data["horsepower"].median())
data["horsepower"]
```

[26]:
```
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
```

[27]:
```
# Look for all unique values in price column
data["price"].unique()
```

[27]:
```
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)
```

[28]:
```
data["price"].replace("?",np.NaN,inplace=True)
data["price"] = pd.to_numeric(data["price"],errors="coerce")
```

[29]:
```
price_summary = {"mean" : data["price"].mean(),
           "median" : data["price"].median(),
           "mode" : data["price"].mode()}

price_summary
```

[29]:
```
{'mean': 13207.129353233831,
 'median': 10295.0,
 'mode': 0      5572.0
 1      6229.0
 2      6692.0
 3      7295.0
```

```
4      7609.0
5      7775.0
6      7898.0
7      7957.0
8      8495.0
9      8845.0
10     8921.0
11     9279.0
12    13499.0
13    16500.0
14    18150.0
Name: price, dtype: float64}
```

[30]:
```
data["price"] = data["price"].fillna(data["price"].median())
data["price"]
```

[30]:
```
0      13495.0
1      16500.0
2      16500.0
3      13950.0
4      17450.0
        …
200    16845.0
201    19045.0
202    21485.0
203    22470.0
204    22625.0
Name: price, Length: 205, dtype: float64
```

[31]:
```
# Look for all unique values in peak-rpm column
data["peak-rpm"].unique()
```

[31]:
```
array(['5000', '5500', '5800', '4250', '5400', '5100', '4800', '6000',
       '4750', '4650', '4200', '4350', '4500', '5200', '4150', '5600',
       '5900', '5750', '?', '5250', '4900', '4400', '6600', '5300'],
      dtype=object)
```

[32]:
```
data["peak-rpm"].replace("?",np.NaN,inplace=True)
data["peak-rpm"] = pd.to_numeric(data["peak-rpm"],errors="coerce")
```

[33]:
```
peak_rpm_summary = {"mean" : data["peak-rpm"].mean(),
           "median" : data["peak-rpm"].median(),
           "mode" : data["peak-rpm"].mode()}

peak_rpm_summary
```

[33]:
```
{'mean': 5125.369458128079,
 'median': 5200.0,
 'mode': 0    5500.0
 Name: peak-rpm, dtype: float64}
```

[34]:
```
data["peak-rpm"] = data["peak-rpm"].fillna(data["peak-rpm"].median())
data["peak-rpm"]
```

[34]:
```
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
```

[35]:
```
data.info()
```

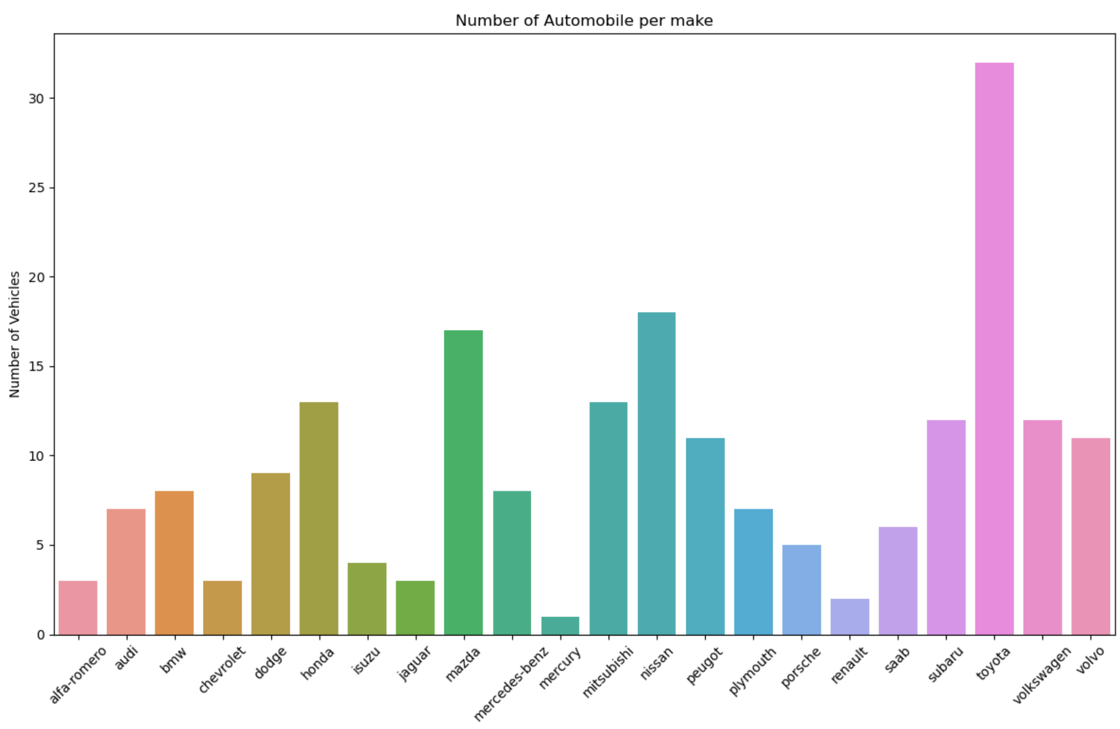
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    float64
 2   make               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    float64
 19  stroke             205 non-null    float64
```

```
 20   compression-ratio  205 non-null    float64
 21   horsepower         205 non-null    float64
 22   peak-rpm           205 non-null    float64
 23   city-mpg           205 non-null    int64
 24   highway-mpg        205 non-null    int64
 25   price              205 non-null    float64
dtypes: float64(11), int64(5), object(10)
memory usage: 41.8+ KB
```

[36]:
```
# saving the clean data to a new csv file
cleaned_data = data.to_csv("E:\Python␣
↪Projects\Automobile\cleaned_Automobile_price_data _raw.csv")
```

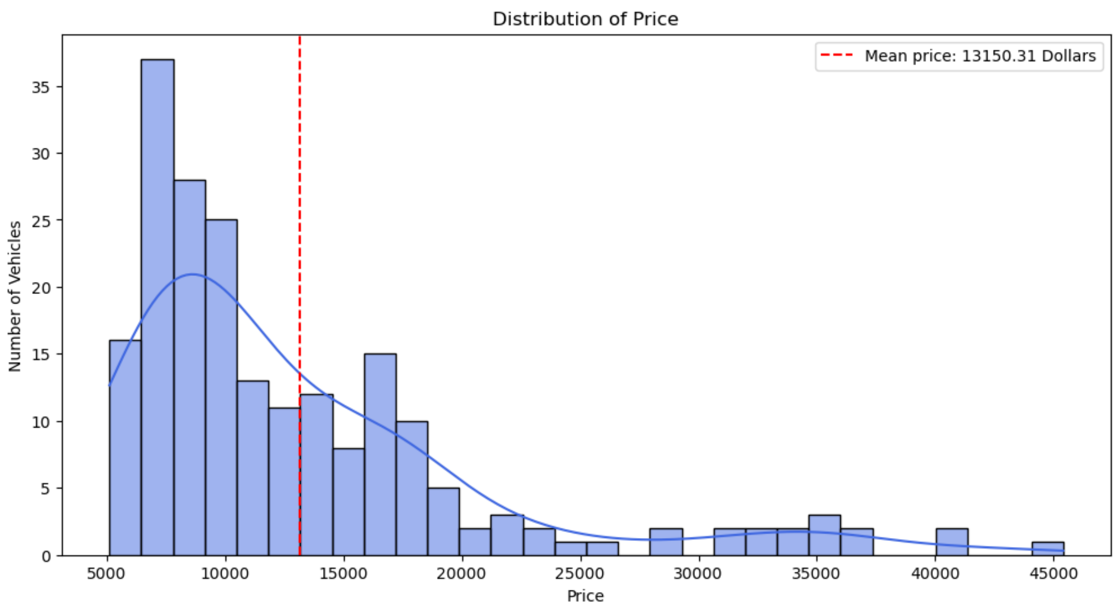
1 Perform Exploratory data analysis

[37]:
```
# Analyse the make of cars = count the number of cars make vise
plt.figure(figsize=(12,8))
sns.countplot(x=data["make"],data = data)
plt.title("Number of Automobile per make")
plt.xlabel("make")
plt.ylabel("Number of Vehicles")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

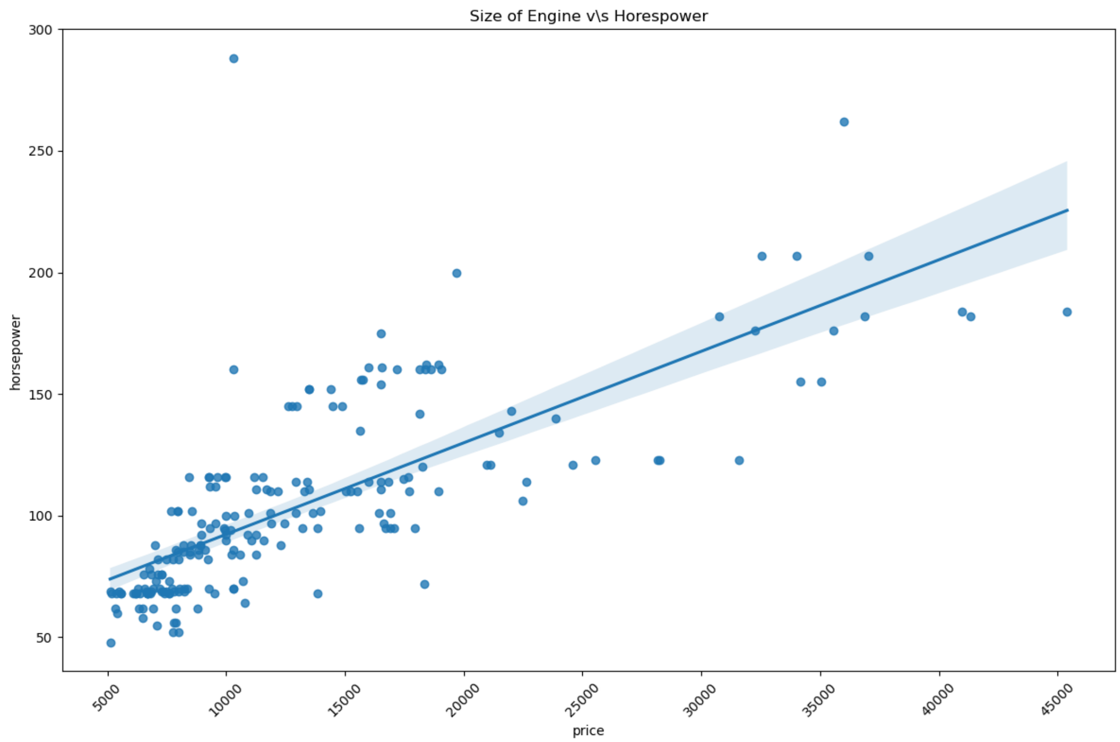


Toyota have most number of car models and Jaguar and Porsche have least.

[38]:
```
# analyzing the distribution of price
plt.figure(figsize=(12, 6))
sns.histplot(data['price'], bins=30, kde=True, color='royalblue')
plt.title('Distribution of Price')
plt.xlabel('Price')
plt.ylabel('Number of Vehicles')
plt.axvline(data['price'].mean(), color='red', linestyle='--', label=f'Mean␣
↪price: {data["price"].mean():.2f} Dollars')
plt.legend()
plt.show()
```

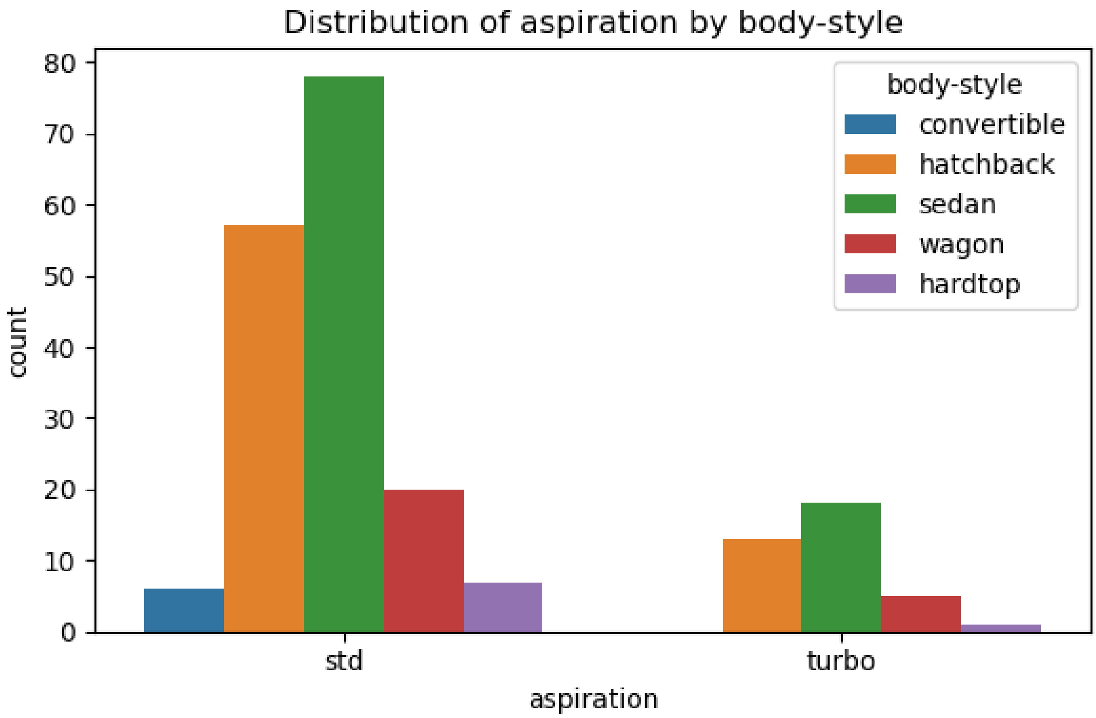


[39]:
```
# Relationship b/w engince-size and horsepower
plt.figure(figsize=(12,8))
sns.regplot(data,x=data["price"],y=data["horsepower"])
plt.title("Size of Engine v\s Horespower")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

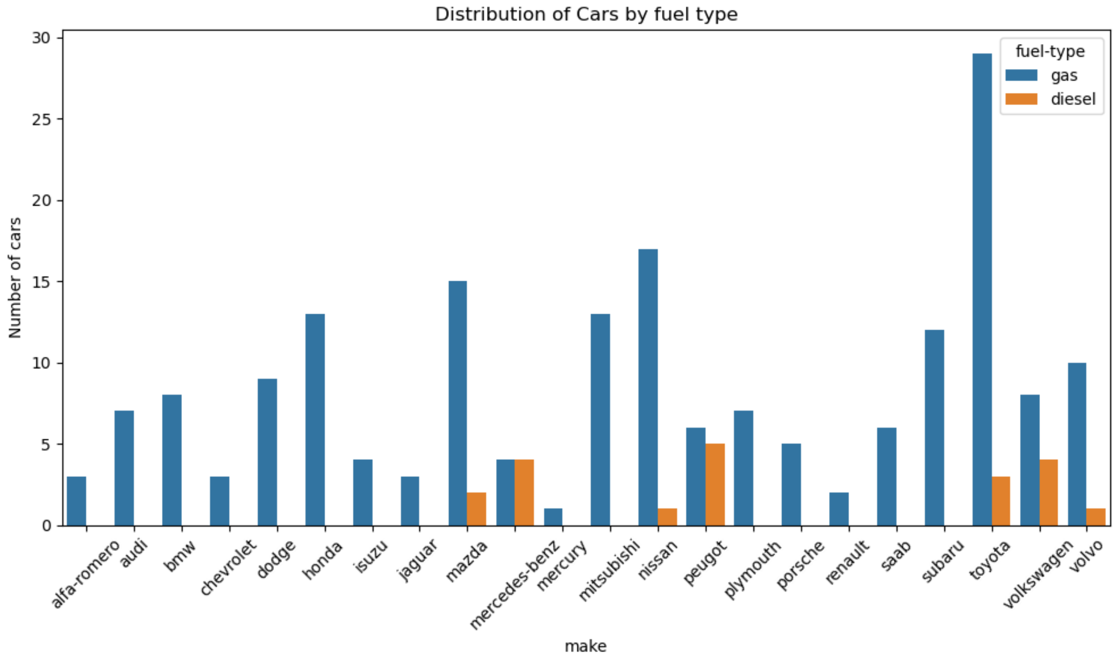


[40]:
```
# Distribution of Price by car body-style
plt.figure(figsize=(6,4))
sns.boxplot(x=data["body-style"],y=data["price"], data = data)
plt.title("Distribution of price by body style")
plt.tight_layout()
plt.show()
```

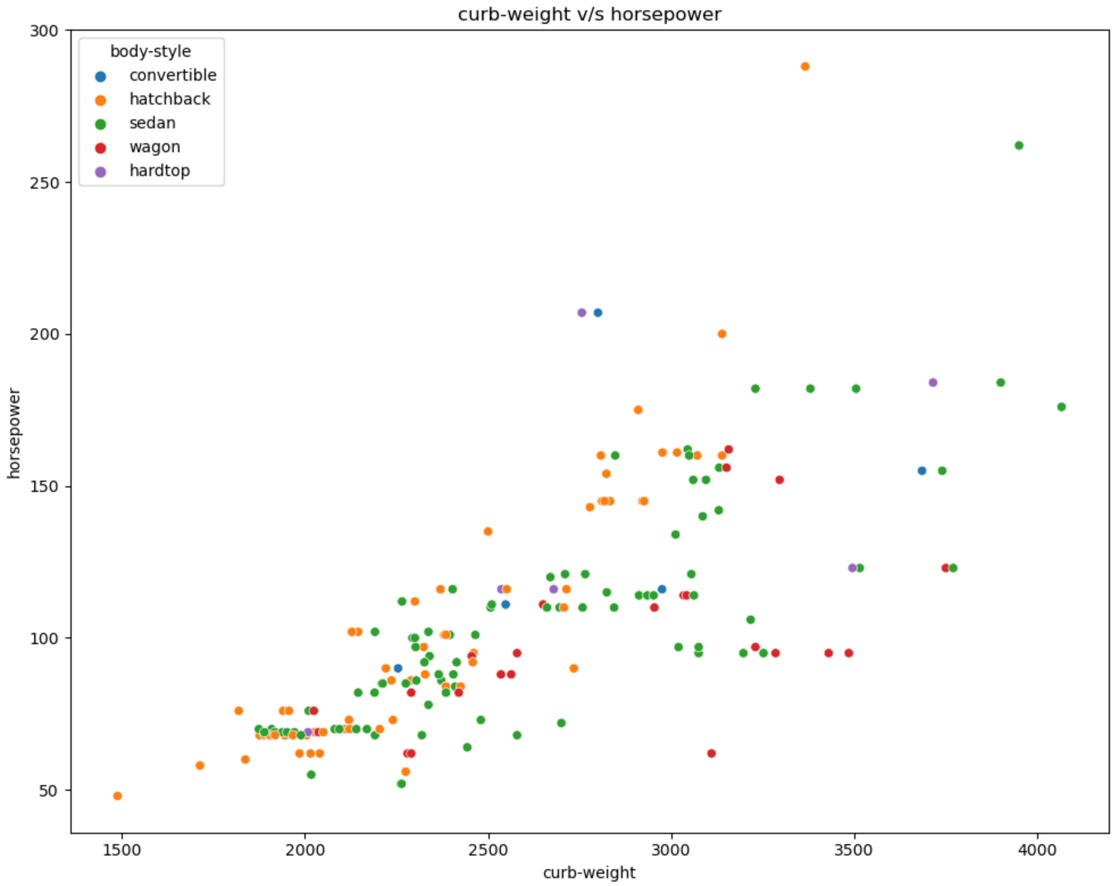
[41]:
```
# Distribution of aspiration by body-style
plt.figure(figsize=(6,4))
sns.countplot(data,x="aspiration",hue="body-style")
plt.title("Distribution of aspiration by body-style")
plt.tight_layout()
plt.show()
```



[42]:
```
# Distribution of Cars by fuel type
plt.figure(figsize=(10,6))
sns.countplot(data,x="make",hue="fuel-type")
plt.title("Distribution of Cars by fuel type")
plt.xticks(rotation=45)
plt.ylabel("Number of cars")
plt.tight_layout()
plt.show()
```

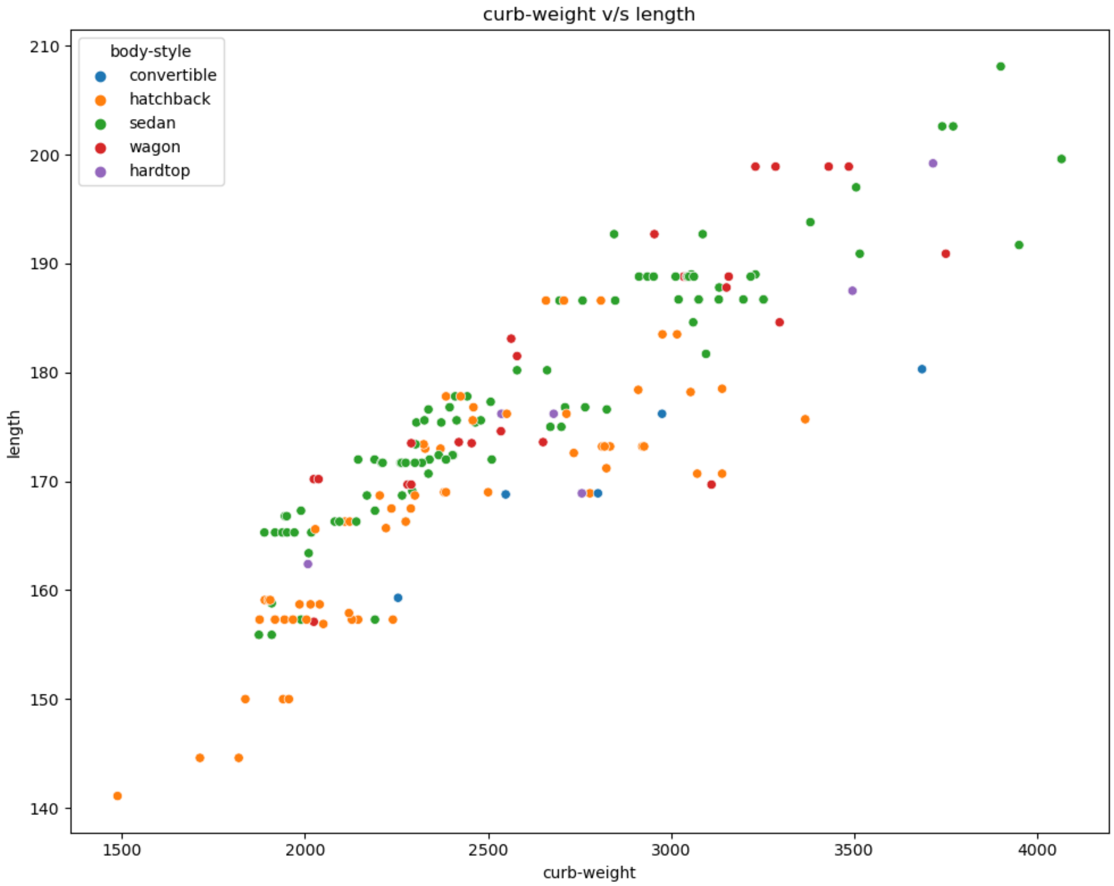


[43]:
```
# Distribution of curb-weight v/s horsepower
plt.figure(figsize=(10,8))
sns.
↪scatterplot(x=data["curb-weight"],y=data["horsepower"],hue=data["body-style"])
plt.title("curb-weight v/s horsepower")
plt.tight_layout()
plt.show()
```



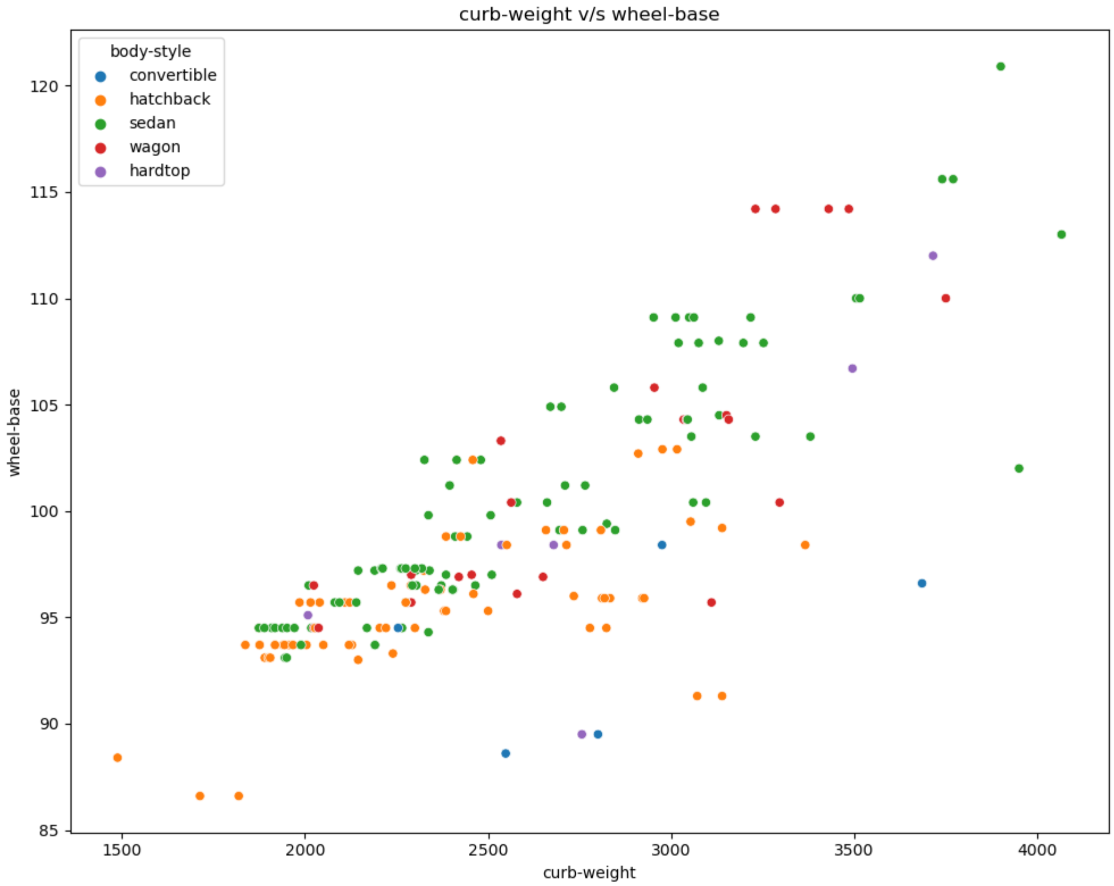
This chart focuses on cars and shows how their weight and Horsepower are related. When we look at the points on the chart, we see that as the weight of the car goes up, the Horsepower also tends to go up. This means there is a positive relationship between the Two. The relationship is quite strong, which means that the weight of the car can explain a lot of the variation in Horsepower.

[44]:
```
# Distribution of curb-weight v/s length
plt.figure(figsize=(10,8))
sns.scatterplot(x=data["curb-weight"],y=data["length"],hue=data["body-style"])
plt.title("curb-weight v/s length")
plt.tight_layout()
plt.show()
```



The heavier a car is, the longer it tends to be. If you want to Make a car longer, you might need to Make it heavier. The chart tells us about the connection between Curb-weight and Length. When Curb-weight goes up, Length also tends to go up on average.

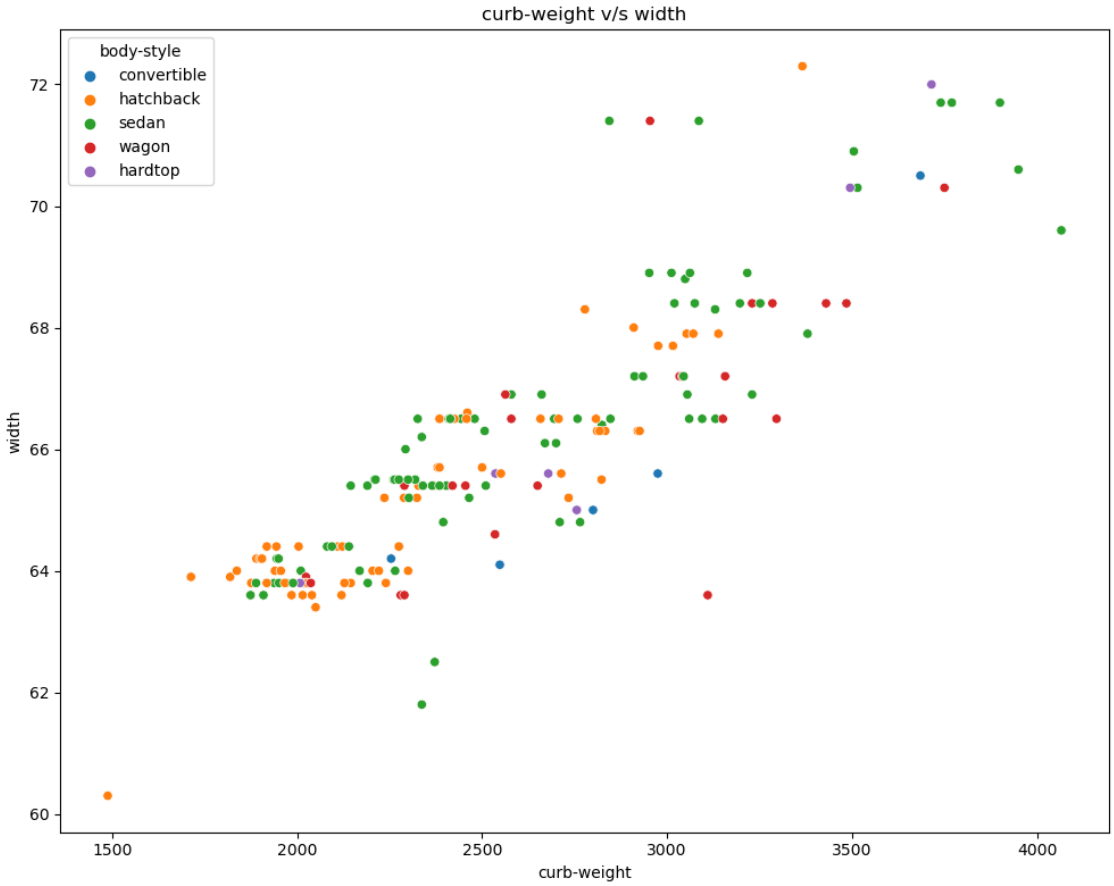
[45]:
```
# # Distribution of curb-weight v/s wheel-base
plt.figure(figsize=(10,8))
sns.
  ↪scatterplot(x=data["curb-weight"],y=data["wheel-base"],hue=data["body-style"])
plt.title("curb-weight v/s wheel-base")
plt.tight_layout()
plt.show()
```



When designing or choosing a Sedan, it's important to consider how the weight of the car might affect the distance between the wheels. A heavier car might need a longer Wheel-base for better stability and performance. We looked at cars with a Sedan body style and found that the weight of the car and the distance between the wheels, called the Wheel-base, are related. When the car's weight goes up, the Wheel-base tends to get bigger too.

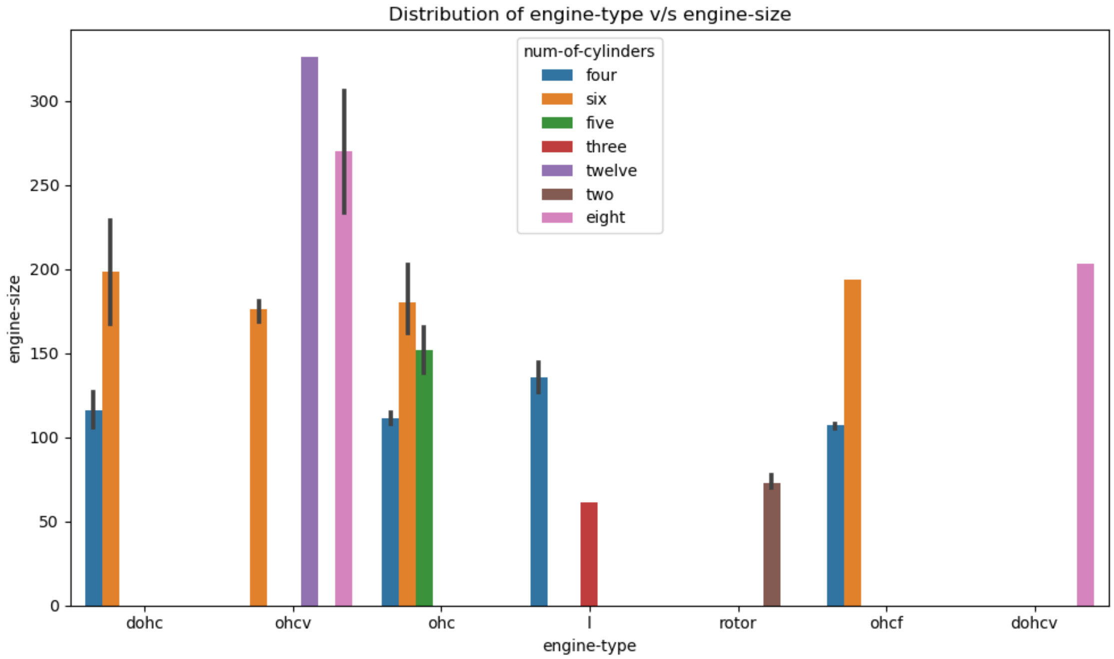
[46]:

```
# Distribution of curb-weight v/s width
plt.figure(figsize=(10,8))
sns.scatterplot(x=data["curb-weight"],y=data["width"],hue=data["body-style"])
plt.title("curb-weight v/s width")
plt.tight_layout()
plt.show()
```

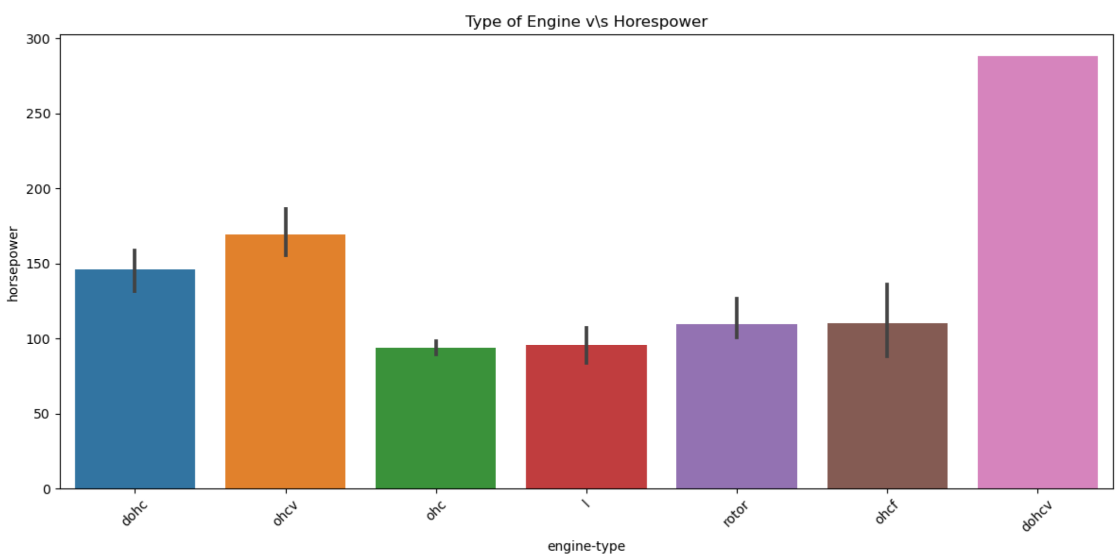


The heavier a car is, the longer it tends to be. If you want to make a car heavier, you might increase the width. The chart tells us about the connection between Curb-weight and Width. When Curb-weight goes up, Width also tends to go up on average. Sedan are more wider than any other car body style.

[47]:
```
#distribution of engine-type v/s engine-size
plt.figure(figsize=(10, 6))
sns.
↪barplot(x=data['engine-type'],y=data["engine-size"],hue=data["num-of-cylinders"])
plt.title('Distribution of engine-type v/s engine-size')
plt.xlabel('engine-type')
plt.ylabel('engine-size')
plt.tight_layout()
plt.show()
```

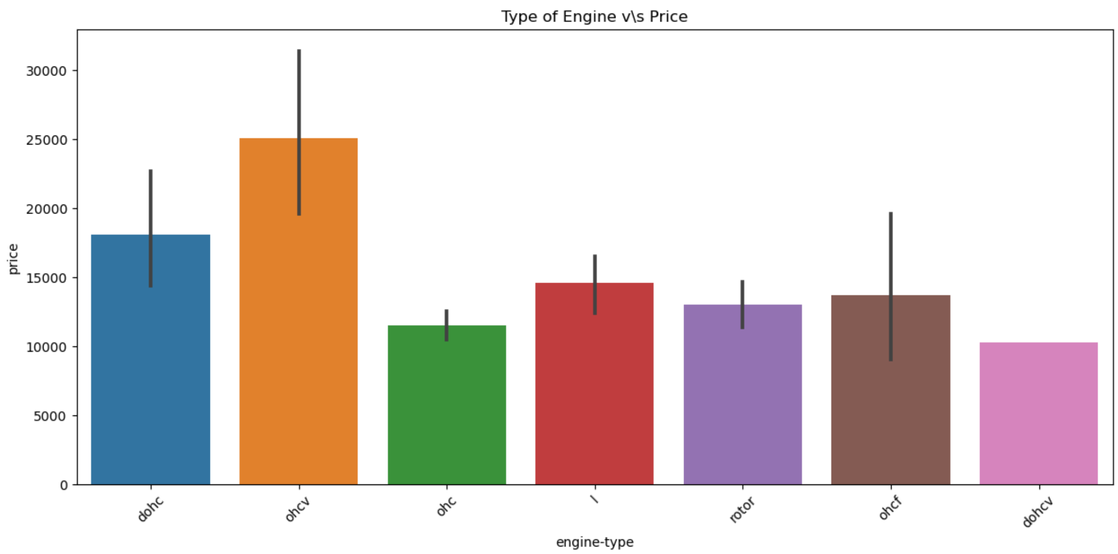


[48]:
```
# Distribution of Engine Type v/s Horsepower of car
plt.figure(figsize=(12,6))
sns.barplot(data,x=data["engine-type"],y=data["horsepower"])
plt.title("Type of Engine v\s Horespower")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



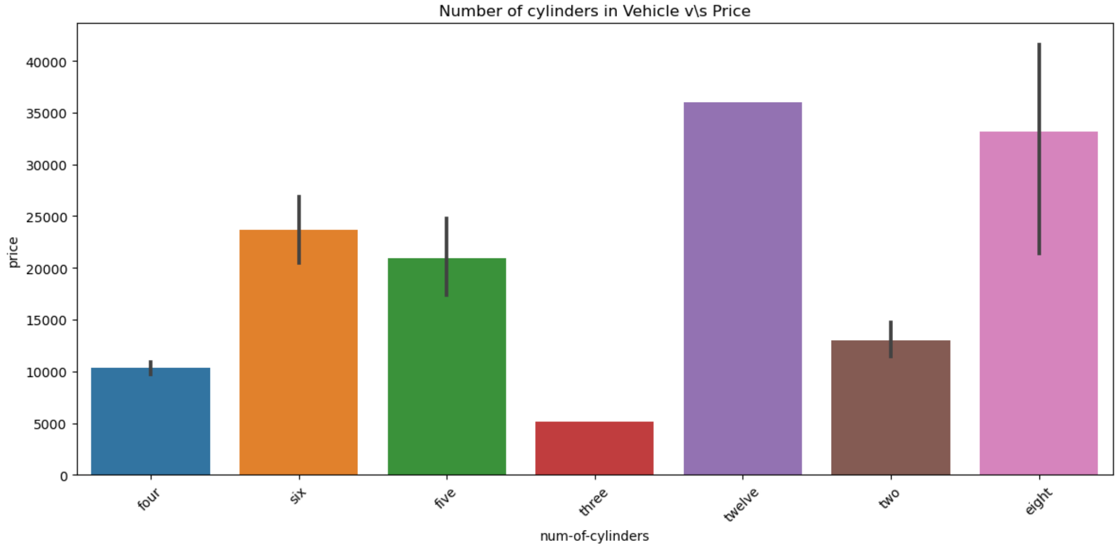
Above insight shows that Engine Type = dohcv generate more power than any other engine.

[49]:
```
# # Distribution of Engine Type v/s Price of car
plt.figure(figsize=(12,6))
sns.barplot(data,x=data["engine-type"],y=data["price"])
plt.title("Type of Engine v\s Price")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



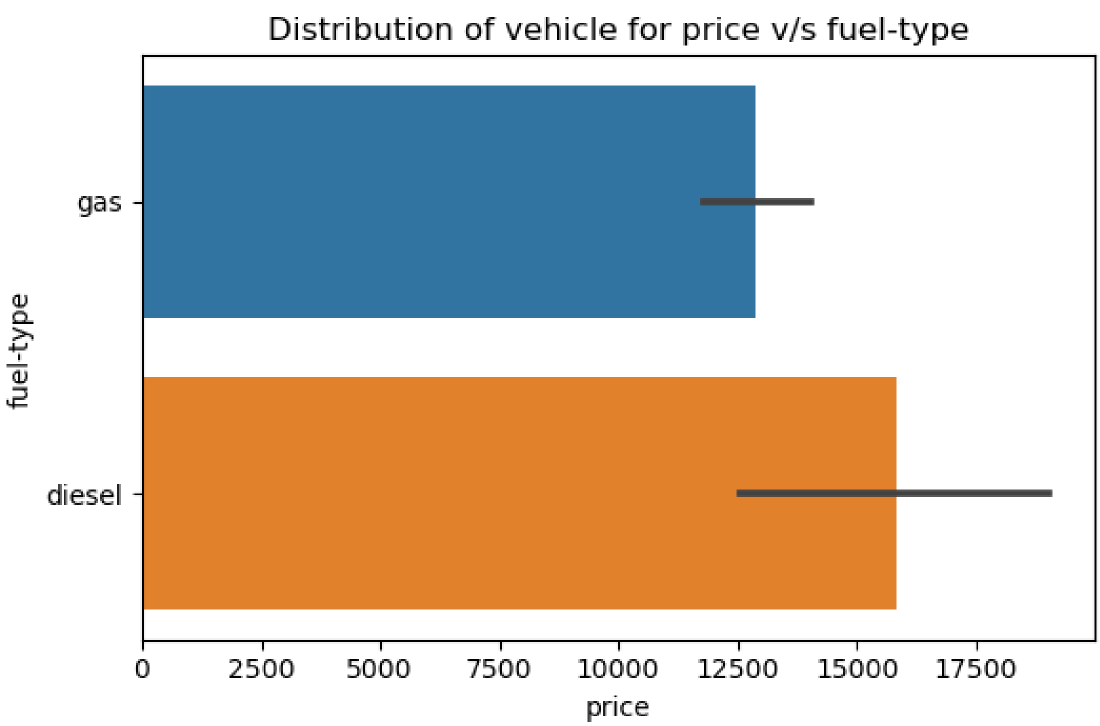
Above insight show that the price of ohcv engine is expensive.

[50]:
```
# Distribution of Number of cylinders in car v\s Price of the car
plt.figure(figsize=(12,6))
sns.barplot(data,x=data["num-of-cylinders"],y=data["price"])
plt.title("Number of cylinders in Vehicle v\s Price")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Above insight shows 12 cyclinder vehicles are expensive and 3 cyclinder vehicle are cheap.

[51]:
```
# Distribution of vehicle for price v/s fuel-type
plt.figure(figsize=(6,4))
sns.barplot(x=data["price"],y=data["fuel-type"])
plt.title("Distribution of vehicle for price v/s fuel-type")
plt.tight_layout()
plt.show()
```

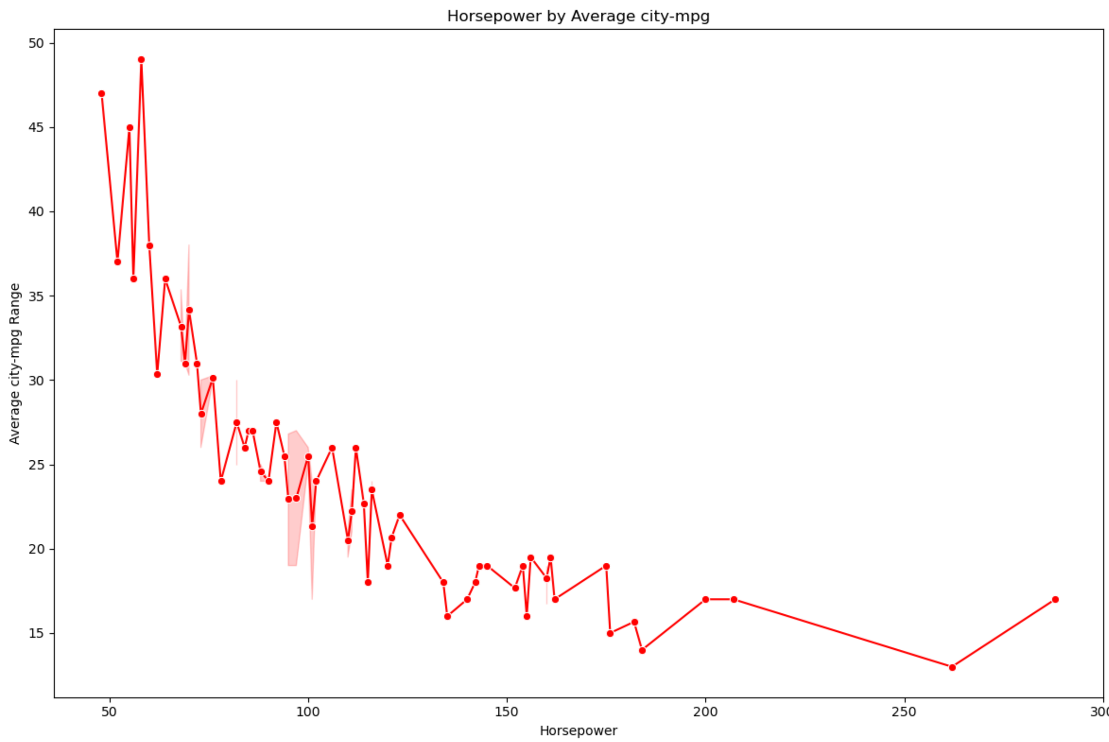


Above Insight show the price of diesel vehicles are more than the gas as a fuel-type.

[52]:
```
average_city_mpg = data.
 ↪groupby(["horsepower","bore","stroke","compression-ratio"])["city-mpg"].
 ↪mean().reset_index()
average_city_mpg
```

[52]:
```
    horsepower  bore  stroke  compression-ratio  city-mpg
0         48.0  2.91    3.03                9.5      47.0
1         52.0  3.01    3.40               23.0      37.0
2         55.0  2.99    3.47               21.9      45.0
3         56.0  3.27    3.35               22.5      36.0
4         58.0  2.91    3.41                9.6      49.0
..         …     …       …                 …         …
83       184.0  3.80    3.35                8.0      14.0
84       200.0  3.43    3.27                7.8      17.0
85       207.0  3.74    2.90                9.5      17.0
86       262.0  3.54    2.76               11.5      13.0
87       288.0  3.94    3.11               10.0      17.0

[88 rows x 5 columns]
```

[53]:
```
# Distribution of Horsepower by Average city-mpg
plt.figure(figsize=(12,8))
sns.
 ↪lineplot(x="horsepower",y="city-mpg",data=average_city_mpg,marker="o",color="red")
plt.title("Horsepower by Average city-mpg")
plt.xlabel("Horsepower")
plt.ylabel("Average city-mpg Range")
plt.tight_layout()
plt.show()
```



As the Horsepower of vehicle goes up the city milege of goes down.

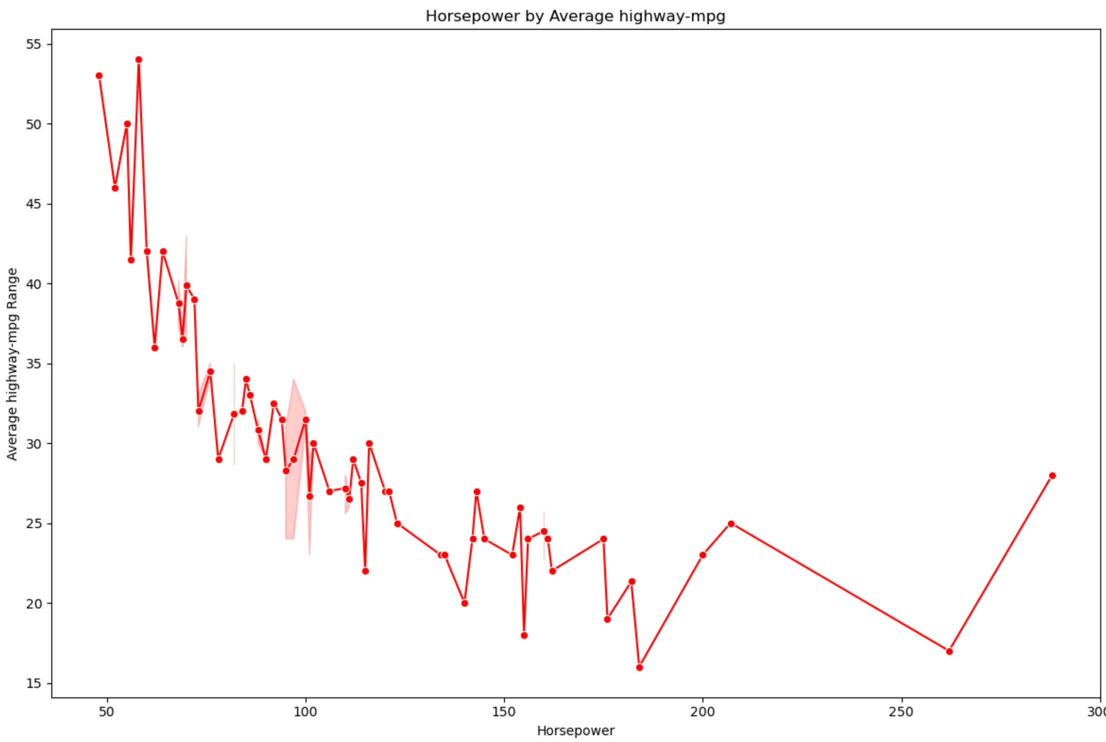
[54]:
```
average_highway_mpg = data.
  ↪groupby(["horsepower","bore","stroke","compression-ratio"])["highway-mpg"].
  ↪mean().reset_index()
average_highway_mpg
```

[54]:

	horsepower	bore	stroke	compression-ratio	highway-mpg
0	48.0	2.91	3.03	9.5	53.0
1	52.0	3.01	3.40	23.0	46.0
2	55.0	2.99	3.47	21.9	50.0
3	56.0	3.27	3.35	22.5	41.5
4	58.0	2.91	3.41	9.6	54.0
..	…	…	…	…	…
83	184.0	3.80	3.35	8.0	16.0
84	200.0	3.43	3.27	7.8	23.0
85	207.0	3.74	2.90	9.5	25.0
86	262.0	3.54	2.76	11.5	17.0
87	288.0	3.94	3.11	10.0	28.0

[88 rows x 5 columns]

[55]:
```
# Distribution of Horsepower by Average highway-mpg
plt.figure(figsize=(12,8))
sns.
 ↪lineplot(x="horsepower",y="highway-mpg",data=average_highway_mpg,marker="o",color="red")
plt.title("Horsepower by Average highway-mpg")
plt.xlabel("Horsepower")
plt.ylabel("Average highway-mpg Range")
plt.tight_layout()
plt.show()
```



As the horsepower of vehicle increases the highway mileage decreases

[56]:
```
average_horsepower = data.groupby("peak-rpm")["horsepower"].mean().reset_index()
average_horsepower
```

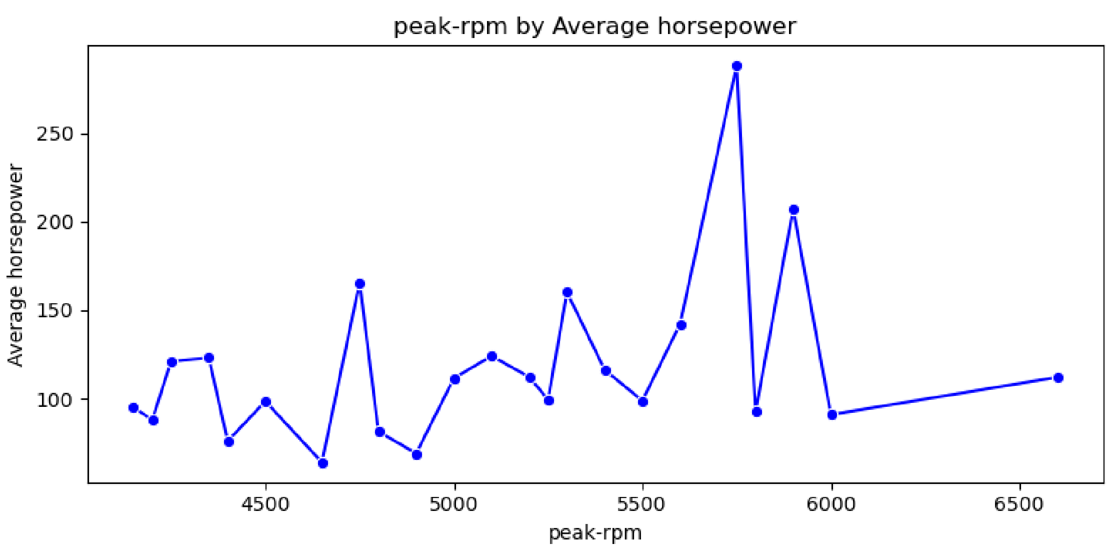
[56]:

	peak-rpm	horsepower
0	4150.0	95.000000
1	4200.0	88.000000
2	4250.0	121.000000
3	4350.0	123.000000
4	4400.0	76.000000
5	4500.0	98.428571
6	4650.0	64.000000
7	4750.0	165.500000

```
8     4800.0   81.361111
9     4900.0   69.000000
10    5000.0  111.444444
11    5100.0  124.000000
12    5200.0  112.120000
13    5250.0   99.285714
14    5300.0  160.000000
15    5400.0  116.153846
16    5500.0   98.756757
17    5600.0  142.000000
18    5750.0  288.000000
19    5800.0   92.428571
20    5900.0  207.000000
21    6000.0   90.888889
22    6600.0  112.000000
```

[57]:

```
# Distribution of Peak RPM of car by Horespower
plt.figure(figsize=(8,4))
sns.
 ↪lineplot(x="peak-rpm",y="horsepower",data=average_horsepower,marker="o",color="blue")
plt.title("peak-rpm by Average horsepower")
plt.xlabel("peak-rpm")
plt.ylabel("Average horsepower")
plt.tight_layout()
plt.show()
```



[58]:

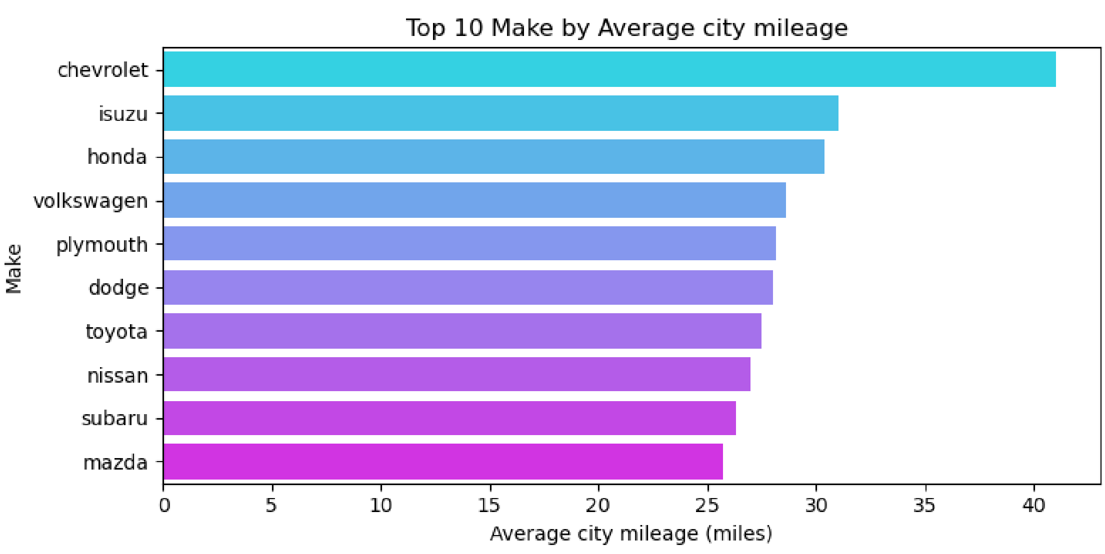
```
average_city_range_by_model = data.groupby(['make'])['city-mpg'].mean().
 ↪sort_values(ascending=False).reset_index()
```

```
# the top 10 models with the highest average city mileage range
top_range_make = average_city_range_by_model.head(10)
top_range_make
```

[58]:

	make	city-mpg
0	chevrolet	41.000000
1	isuzu	31.000000
2	honda	30.384615
3	volkswagen	28.583333
4	plymouth	28.142857
5	dodge	28.000000
6	toyota	27.500000
7	nissan	27.000000
8	subaru	26.333333
9	mazda	25.705882

[59]:
```
# Distribution of top 10 car make by average city mileage
plt.figure(figsize=(8, 4))
barplot = sns.barplot(x='city-mpg', y='make', data=top_range_make,␣
↪palette="cool")
plt.title('Top 10 Make by Average city mileage ')
plt.xlabel('Average city mileage (miles)')
plt.ylabel('Make')
plt.tight_layout()
plt.show()
```



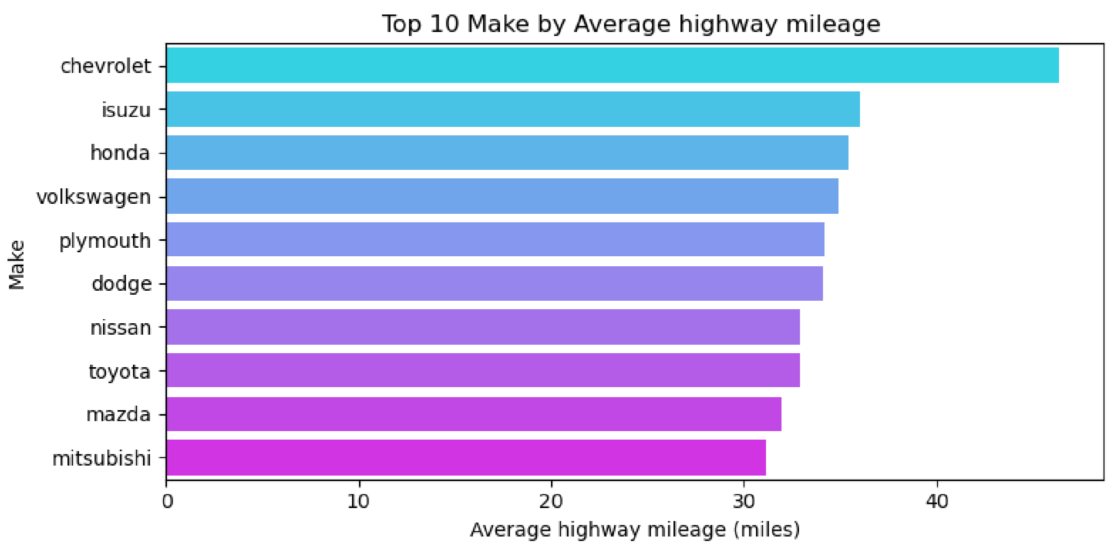
Chevrolet have more mileage and Subaru have low mileage in city.

[60]:
```
average_highway_range_by_make = data.groupby(['make'])['highway-mpg'].mean().
 ↪sort_values(ascending=False).reset_index()

# the top 10 models with the highest average city mileage range
top_h_range_make = average_highway_range_by_make.head(10)
top_h_range_make
```

[60]:

	make	highway-mpg
0	chevrolet	46.333333
1	isuzu	36.000000
2	honda	35.461538
3	volkswagen	34.916667
4	plymouth	34.142857
5	dodge	34.111111
6	nissan	32.944444
7	toyota	32.906250
8	mazda	31.941176
9	mitsubishi	31.153846

[61]:
```
# Distribution of top 10 car make by average highway mileage
plt.figure(figsize=(8, 4))
barplot = sns.barplot(x='highway-mpg', y='make', data=top_h_range_make,␣
 ↪palette="cool")
plt.title('Top 10 Make by Average highway mileage ')
plt.xlabel('Average highway mileage (miles)')
plt.ylabel('Make')
plt.tight_layout()
plt.show()
```



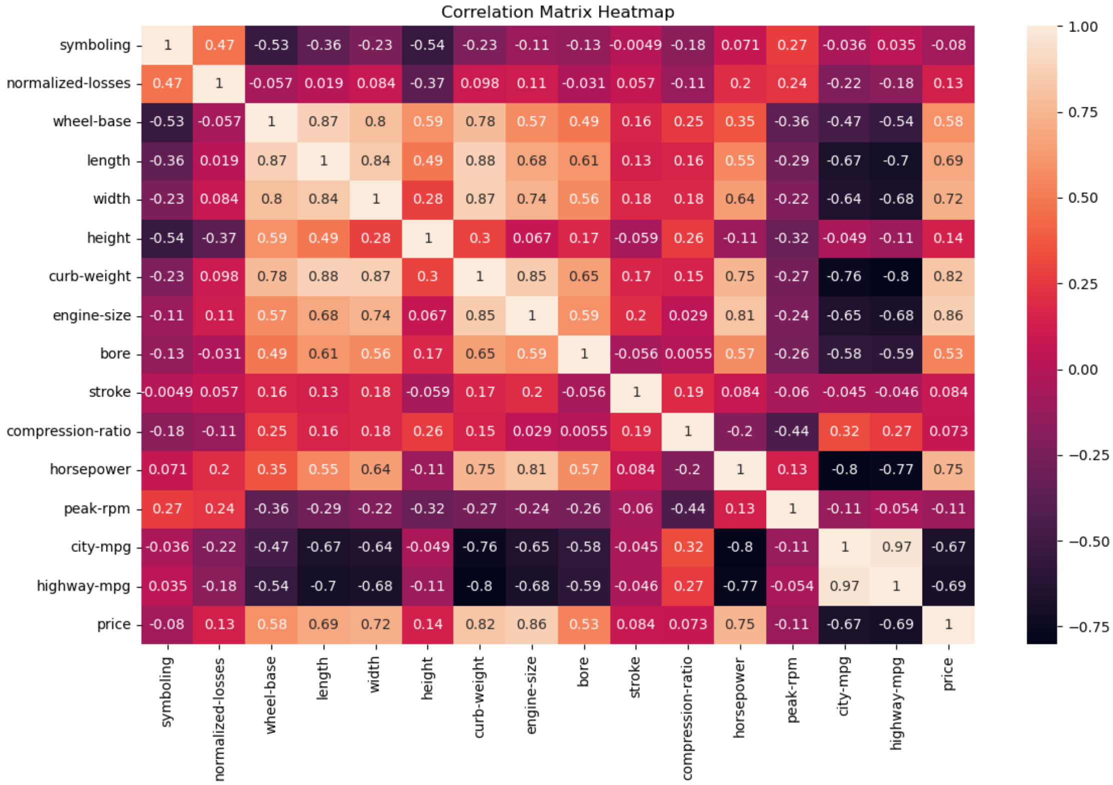
[62]:
```
# calculate the correlation matrix
n_data = data.select_dtypes(include=["Int64","Float64"])
correlation_matrix= n_data.corr()
correlation_matrix
```

[62]:
```
                   symboling  normalized-losses  wheel-base    length  \
symboling           1.000000           0.465190   -0.531954 -0.357612
normalized-losses   0.465190           1.000000   -0.056518  0.019209
wheel-base         -0.531954          -0.056518    1.000000  0.874587
length             -0.357612           0.019209    0.874587  1.000000
width              -0.232919           0.084195    0.795144  0.841118
height             -0.541038          -0.370706    0.589435  0.491029
curb-weight        -0.227691           0.097785    0.776386  0.877728
engine-size        -0.105790           0.110997    0.569329  0.683360
bore               -0.132563          -0.030528    0.489556  0.607016
stroke             -0.004928           0.056833    0.159684  0.128622
compression-ratio  -0.178515          -0.114525    0.249786  0.158414
horsepower          0.071064           0.203380    0.352876  0.553337
peak-rpm            0.273851           0.237719   -0.361338 -0.286362
city-mpg           -0.035823          -0.218749   -0.470414 -0.670909
highway-mpg         0.034606          -0.178221   -0.544082 -0.704662
price              -0.080149           0.133823    0.584847  0.686567

                      width    height  curb-weight  engine-size      bore  \
symboling         -0.232919 -0.541038    -0.227691    -0.105790 -0.132563
normalized-losses  0.084195 -0.370706     0.097785     0.110997 -0.030528
wheel-base         0.795144  0.589435     0.776386     0.569329  0.489556
length             0.841118  0.491029     0.877728     0.683360  0.607016
width              1.000000  0.279210     0.867032     0.735433  0.559262
height             0.279210  1.000000     0.295572     0.067149  0.173506
curb-weight        0.867032  0.295572     1.000000     0.850594  0.648848
engine-size        0.735433  0.067149     0.850594     1.000000  0.585636
bore               0.559262  0.173506     0.648848     0.585636  1.000000
stroke             0.182708 -0.058994     0.168164     0.200246 -0.056054
compression-ratio  0.181129  0.261214     0.151362     0.028971  0.005468
horsepower         0.641337 -0.109286     0.750927     0.810216  0.574258
peak-rpm          -0.219374 -0.321113    -0.266358    -0.244383 -0.256600
city-mpg          -0.642704 -0.048640    -0.757414    -0.653658 -0.582627
highway-mpg       -0.677218 -0.107358    -0.797465    -0.677470 -0.585352
price              0.724558  0.140439     0.819817     0.860343  0.532861

                     stroke  compression-ratio  horsepower  peak-rpm  \
symboling         -0.004928          -0.178515    0.071064  0.273851
normalized-losses  0.056833          -0.114525    0.203380  0.237719
wheel-base         0.159684           0.249786    0.352876 -0.361338
length             0.128622           0.158414    0.553337 -0.286362
width              0.182708           0.181129    0.641337 -0.219374
```

```
height            -0.058994           0.261214   -0.109286 -0.321113
curb-weight        0.168164           0.151362    0.750927 -0.266358
engine-size        0.200246           0.028971    0.810216 -0.244383
bore              -0.056054           0.005468    0.574258 -0.256600
stroke             1.000000           0.185679    0.083804 -0.059716
compression-ratio  0.185679           1.000000   -0.204851 -0.436441
horsepower         0.083804          -0.204851    1.000000  0.130565
peak-rpm          -0.059716          -0.436441    0.130565  1.000000
city-mpg          -0.044973           0.324701   -0.802170 -0.114230
highway-mpg       -0.046389           0.265201   -0.770780 -0.054195
price              0.083627           0.072890    0.749919 -0.107283

                   city-mpg  highway-mpg     price
symboling         -0.035823     0.034606 -0.080149
normalized-losses -0.218749    -0.178221  0.133823
wheel-base        -0.470414    -0.544082  0.584847
length            -0.670909    -0.704662  0.686567
width             -0.642704    -0.677218  0.724558
height            -0.048640    -0.107358  0.140439
curb-weight       -0.757414    -0.797465  0.819817
engine-size       -0.653658    -0.677470  0.860343
bore              -0.582627    -0.585352  0.532861
stroke            -0.044973    -0.046389  0.083627
compression-ratio  0.324701     0.265201  0.072890
horsepower        -0.802170    -0.770780  0.749919
peak-rpm          -0.114230    -0.054195 -0.107283
city-mpg           1.000000     0.971337 -0.668822
highway-mpg        0.971337     1.000000 -0.693037
price             -0.668822    -0.693037  1.000000
```

[63]:

```
#Plot the heatmap
plt.figure(figsize=(12,8))
sns.heatmap(correlation_matrix,annot=True)
plt.title("Correlation Matrix Heatmap")
plt.tight_layout()
plt.show()
```



2 Summary

Number of Vehicle in market with fuel-type gas is more than diesel. The price of diesel vehicles are more than the gas as a fuel-type. The Wheel-base,length, width, height of a vehicle is directly proportional to curb-weight of Vehicle. As the horesepower increases the Mileage Decreases in city and Hightway. The big size of engine the more power it will generate.As engine size 258 and 6 cyclinder generate more power. The price of ohcv engine is expensive as it generate more horsepower and function well. Chevrolet have more mileage and Subaru have low mileage in both City and Highway. Toyota have most number of Automobile Vehicles and Jaguar and Porsche have least number of Automobile Vehicles. Thus the Vehicle with Fuel Type - Diesel, N0.of cyclinders-12, Engine-type = dohcv are expensive in Market. Diesel vehicle give more mileage than gas and enhance the performance of the vehicle.