TASK

# Exploratory Data Analysis on the Automobile Data Set

Visit our website

# Introduction

For this EDA we will work with an automobile data set to which a number of questions will be answered, and visualisations will be displayed to explain the data.

All required packages are imported

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

df = pd.read_csv('automobile.csv')
```

### Automobile data set

Using the head() function, the first five rows of the data set is displayed.

In [2]:
```
df.head()
```

Out[2]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepower |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111 |
| 1 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111 |
| 2 | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | 154 |
| 3 | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.40 | 10.0 | 102 |
| 4 | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.40 | 8.0 | 115 |

5 rows × 26 columns

## DATA CLEANING

On inspection, the data contains '?' which needs to be replaced with 'NAN'

In [3]:
```
df_auto = df.replace('?', np.NAN)
df_auto.head()
```

Out[3]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepower |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | NaN | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111 |
| 1 | 3 | NaN | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111 |
| 2 | 1 | NaN | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | 154 |
| 3 | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.40 | 10.0 | 102 |
| 4 | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.40 | 8.0 | 115 |

5 rows × 26 columns

# MISSING DATA

## Missing data in list form

We will now check for missing data in the data set by using the .isnull() and .sum() functions.

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

```
Out[4]: 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
```

## Missing data in table form

The missing data can also be displayed in table form as follows:

```
In [5]: df_auto.isnull()
```

Out[5]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepower | p |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | False | True | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 1 | False | True | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 2 | False | True | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 3 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 4 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 200 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 201 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 202 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 203 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |
| 204 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | F |

205 rows × 26 columns

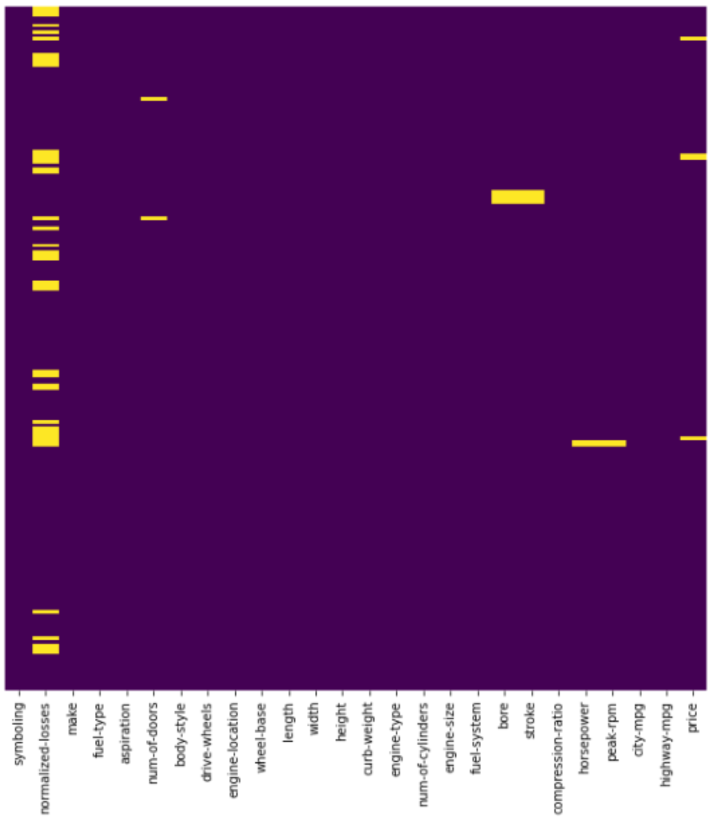
If a value is equal to 'True' in the table above, it means it is missing.

## Missing data in graphical form

Using seaborn visualization, we can see the missing data represented as yellow lines.

```
In [6]: plt.figure(figsize=(10, 10))

        sns.heatmap(df_auto.isnull(),yticklabels=False, cbar=False, cmap= 'viridis')
Out[6]: <AxesSubplot:>
```




## Statistics of the data set

The following table shows the statistics of the data set. We can use the mean values to replace the missing data in the data set.

```
In [7]: df.describe()
Out[7]:
```

| | symboling | wheel-base | length | width | height | curb-weight | engine-size | compression-ratio | city-mpg | highway-mpg |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 205.000000 | 205.000000 | 205.000000 | 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 | 10.142537 | 25.219512 | 30.751220 |
| std | 1.245307 | 6.021776 | 12.337289 | 2.145204 | 2.443522 | 520.680204 | 41.642693 | 3.972040 | 6.542142 | 6.886443 |
| min | -2.000000 | 86.600000 | 141.100000 | 60.300000 | 47.800000 | 1488.000000 | 61.000000 | 7.000000 | 13.000000 | 16.000000 |
| 25% | 0.000000 | 94.500000 | 166.300000 | 64.100000 | 52.000000 | 2145.000000 | 97.000000 | 8.600000 | 19.000000 | 25.000000 |
| 50% | 1.000000 | 97.000000 | 173.200000 | 65.500000 | 54.100000 | 2414.000000 | 120.000000 | 9.000000 | 24.000000 | 30.000000 |
| 75% | 2.000000 | 102.400000 | 183.100000 | 66.900000 | 55.500000 | 2935.000000 | 141.000000 | 9.400000 | 30.000000 | 34.000000 |
| max | 3.000000 | 120.900000 | 208.100000 | 72.300000 | 59.800000 | 4066.000000 | 326.000000 | 23.000000 | 49.000000 | 54.000000 |

**normalized-losses**

```
In [8]: df_n = df[df['normalized-losses'] != '?']
        mean = df_n['normalized-losses'].astype(int).mean()
        df['normalized-losses'] = df['normalized-losses'].replace('?', mean).astype(int)
```

**num-of-doors**

```
In [9]: df['num-of-doors'] = df['num-of-doors'].replace('?', 'four')
```

**bore**

```
In [10]: df_n = df[df['bore'] != '?']
         mean = df_n['bore'].astype(float).mean()
         df['bore'] = df['bore'].replace('?', mean).astype(float)
```

**stroke**

```
In [11]: df_n = df[df['stroke'] != '?']
         mean = df_n['stroke'].astype(float).mean()
         df['stroke'] = df['stroke'].replace('?', mean).astype(float)
```

**horsepower**

```
In [12]: df_n = df[df['horsepower'] != '?']
         mean = df_n['horsepower'].astype(float).mean()
         df['horsepower'] = df['horsepower'].replace('?', mean).astype(float)
```

**peak-rpm**

```
In [13]: df_n = df[df['peak-rpm'] != '?']
         mean = df_n['peak-rpm'].astype(float).mean()
         df['peak-rpm'] = df['peak-rpm'].replace('?', mean).astype(float)
```

**price**

```
In [14]: df_n = df[df['price'] != '?']
         mean = df_n['price'].astype(float).mean()
         df['price'] = df['price'].replace('?', mean).astype(float)
         df.head()
```

Out[14]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepower |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111.0 |
| **1** | 3 | 122 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111.0 |
| **2** | 1 | 122 | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | 154.0 |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.40 | 10.0 | 102.0 |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.40 | 8.0 | 115.0 |

5 rows × 26 columns

## DATA STORIES AND VISUALIZATIONS

In this section of the EDA, we will extract stories and assumptions based on visualizations of the data set

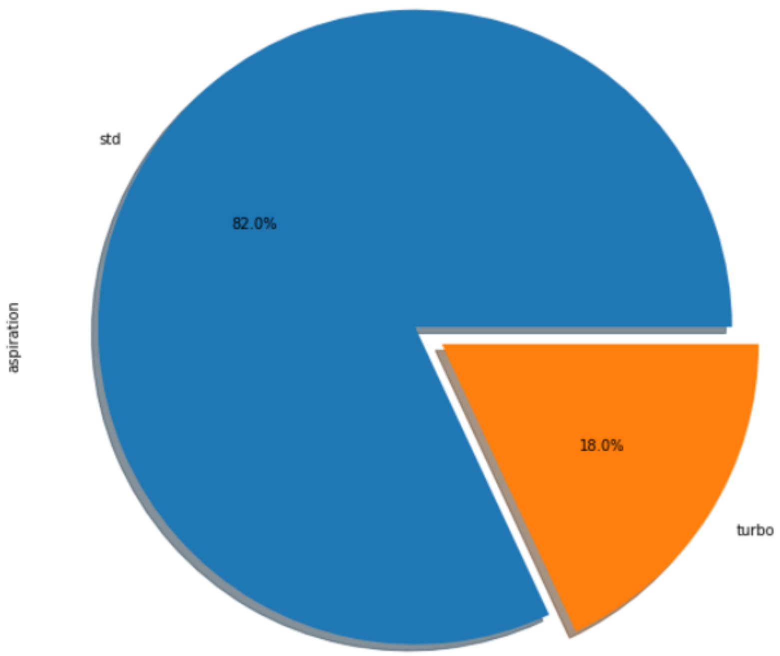
**Aspiration analysis**

Here we will check how many automobiles are normally aspirated versus how many have turbos. The functions below will print the numbers as std and turbo, and a pie chart.

```
In [15]: df.aspiration.value_counts().plot(kind = 'pie', figsize=(10,10), autopct = '%1.1f%%',
                                            shadow=True, explode=[0.05, 0.05])

         print(df.aspiration.value_counts())

std      168
turbo     37
Name: aspiration, dtype: int64
```




**Results**

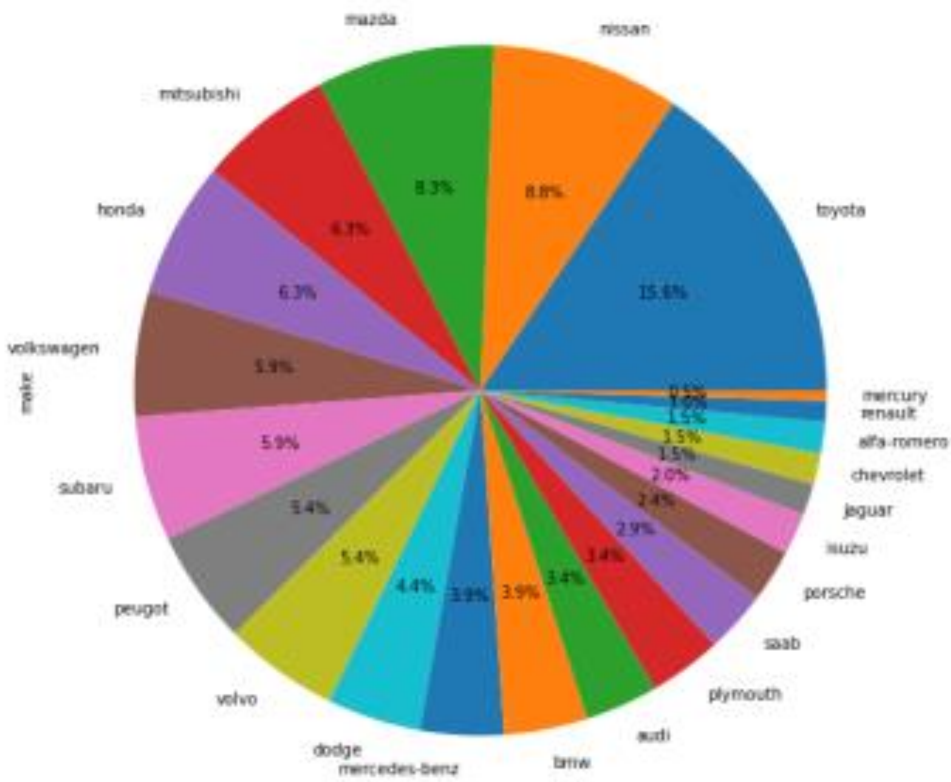
It is clear from the pie chart that the majority of the automobiles are normally aspirated with a percentage of 82%, compared to turbo automobiles with a percentage of 18%.

## Make analysis

Under make analysis we will check how many different makes each automobile manufacturer produces. The functions below will print the total per manufacturer and a pie chart.

```
In [16]: df.make.value_counts().plot(kind = 'pie', figsize=(10,10), autopct = '%1.1f%%')
         print(df.make.value_counts())
```

```
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
```




### Results

- Japan has the highest rate of different models in the top three automobiles

- Toyota has almost double the amount of different models compared to its two closest competitors, Nissan and Mazda
- The low-cost car manufacturers have more models than the high-cost car manufacturers

### Price analysis

Here we will check the average price per car for each manufacturer. The functions below will print a bar chart showing the price per make.

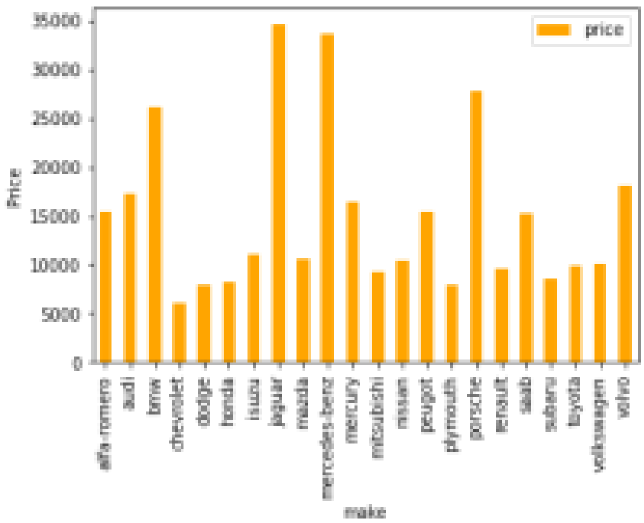
```
In [17]: group_by_make = df.groupby(by=['make'])

         car_data_avg = round(group_by_make.mean(), 0)

         price = pd.DataFrame({'price': car_data_avg['price']})

         price.plot(kind = 'bar', color = 'orange', ylabel = "Price")

Out[17]: <AxesSubplot:xlabel='make', ylabel='Price'>
```




### Results

- Jaguar, Mercedes-Benzes, BMW and Porsche produce the most expensive cars
- Toyota, Plymouth, Nissan and Chevrolet produce affordable cars
- The majority of the car companies sell cars for less than 20000

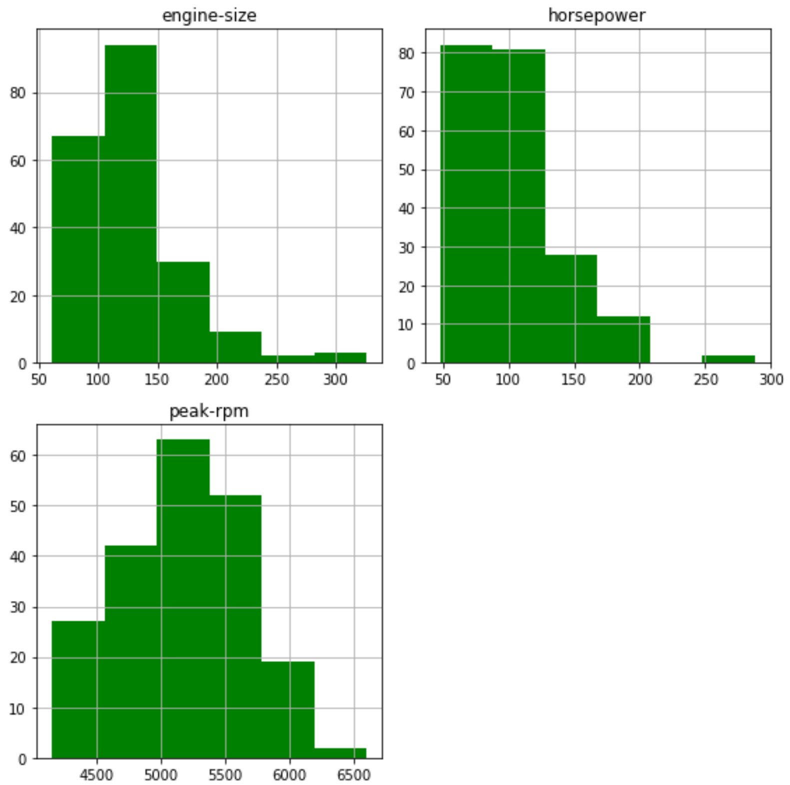
**Engine specification analysis**

In this section we will check the engine specifications in terms of 'engine size', 'horsepower' and 'peak-rpm'. The functions below will print three different histograms.

```
In [18]: df['engine-size'] = pd.to_numeric(df['engine-size'], errors='coerce')
         df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
         df['peak-rpm'] = pd.to_numeric(df['peak-rpm'], errors='coerce')

         df[['engine-size', 'horsepower', 'peak-rpm']].hist(figsize = (8, 8), bins = 6, color = 'Green')

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




**Results**

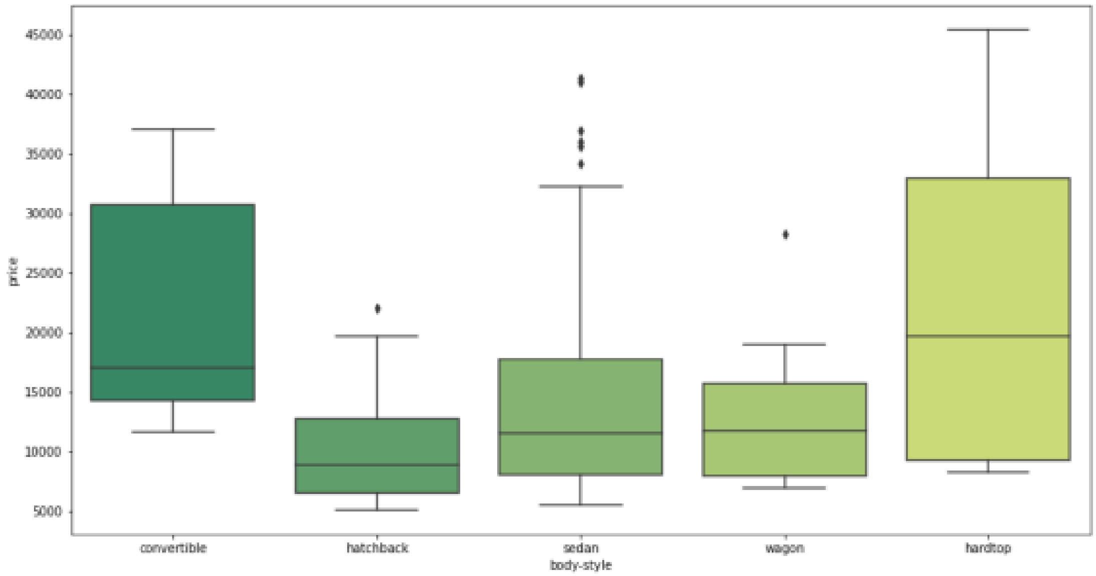
- The majority of the engine sizes are between 65 and 195
- The majority of the cars have horsepower 50 to 125
- The peak-rpm at 5000 to 5700

## Price vs Body style analysis

Here we will compare the price to body style of each automobile. The functions below will print a box plot for each body style.

```
In [19]: plt.figure(figsize=(15,8))
         sns.boxplot(x='body-style', y='price', data=df, palette='summer')

Out[19]: <AxesSubplot:xlabel='body-style', ylabel='price'>
```




## Results

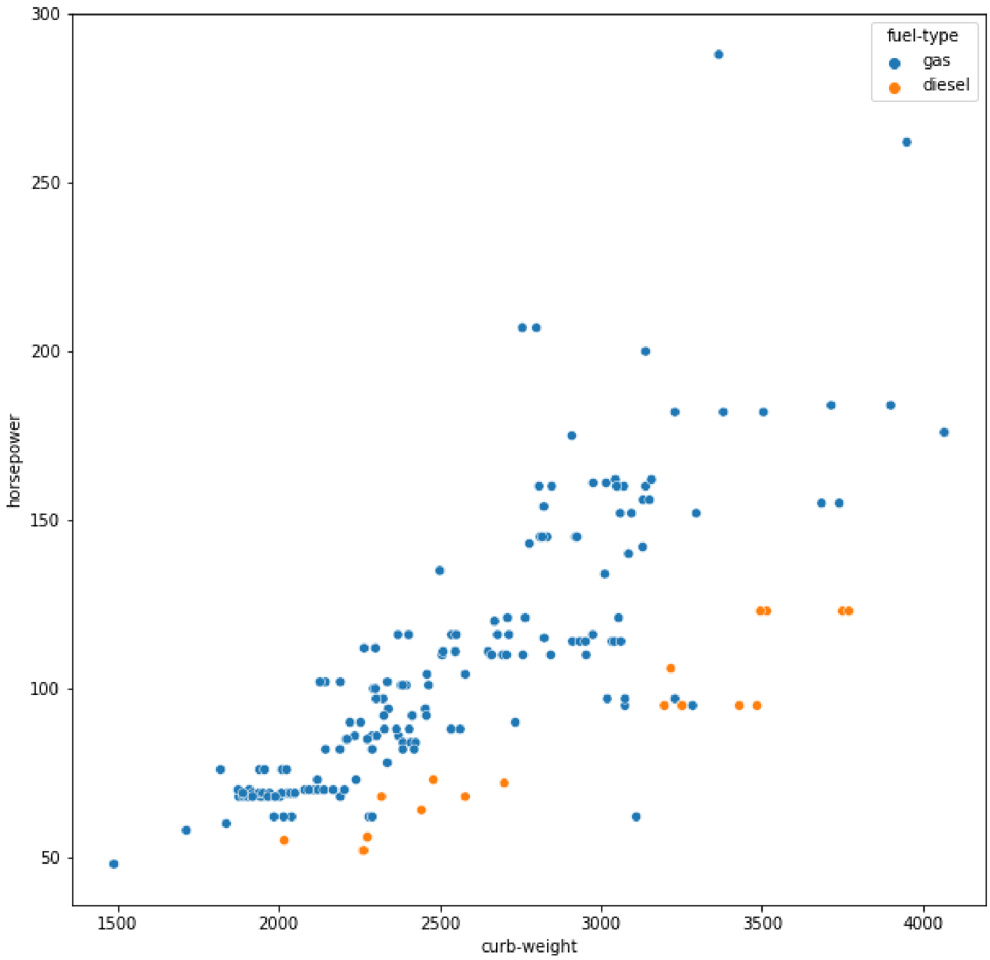
- Hardtop cars (at 20000) and convertibles (at 17500) are the most expensive models
- Hatchback models (at 8000) are the least expensive
- Sedan and wagons are more or less the same price (at 12500)

### Power to weight ratio

In this section we compare the power (horsepower) to weight (curb-weight which is the weight of the car with a full tank of gas) ratio. The comparison is also between gas and diesel cars, to see which type of fuel has more power. The function below is used to print a scatter graph to illustrate this.

```
In [20]: fig = plt.figure(figsize=(10, 10))
         sns.scatterplot(x='curb-weight', y='horsepower', hue='fuel-type', data=df)

Out[20]: <AxesSubplot:xlabel='curb-weight', ylabel='horsepower'>
```




### Results

As seen on the legend, the blue scatter plot represents gas (petrol) and the orange scatter plot represents diesel. It is clear that gas cars have higher power (horsepower) at high curb-weights; compared to diesel cars that have less power at the same curb-weight.

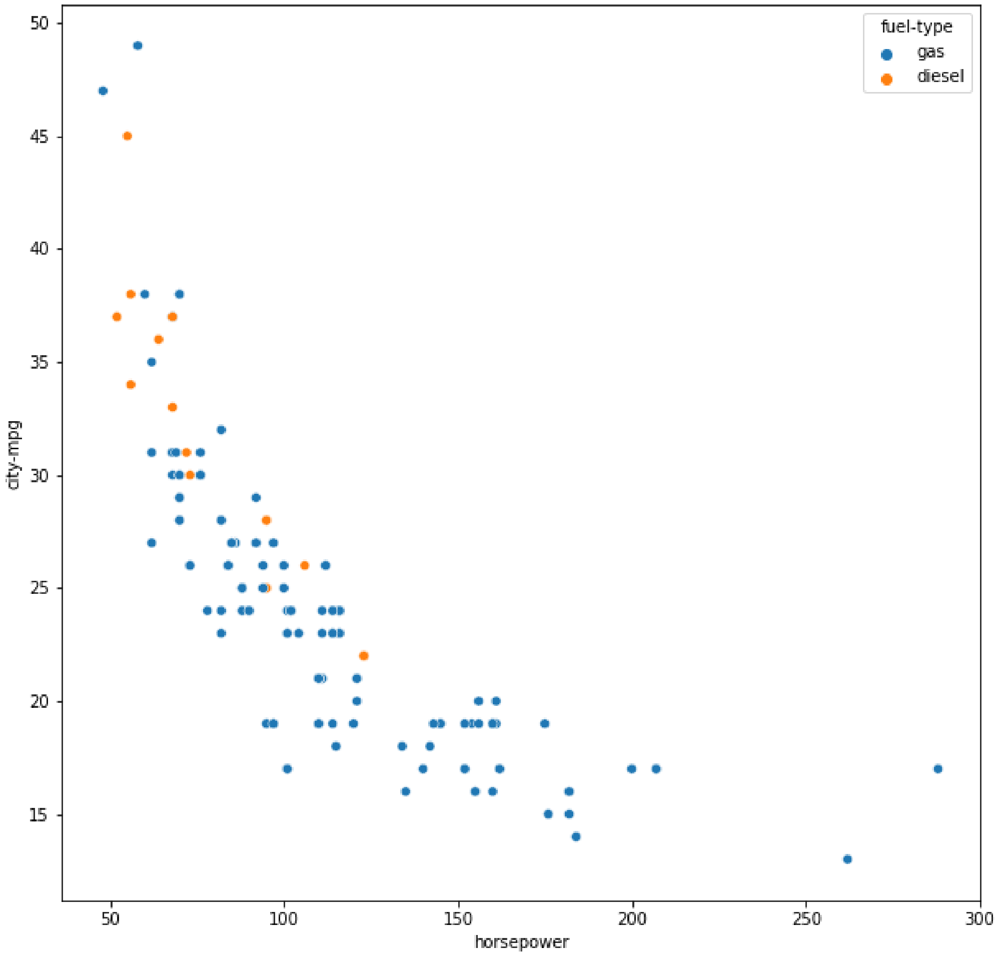
 

### Fuel economy

Here we will compare the fuel economy between gas and diesel cars. The function is used to illustrate this as a scatter graph.

```
In [21]: fig = plt.figure(figsize=(10, 10))
         sns.scatterplot(x='horsepower', y='city-mpg', hue='fuel-type', data=df)
```

```
Out[21]: <AxesSubplot:xlabel='horsepower', ylabel='city-mpg'>
```




### Results

This scatter plot diagram shows that the majority of diesel cars can reach higher miles per gallon (mpg) at lower power (horsepower) compared to gas/petrol cars. It needs to be noted though, that the diesel sample is smaller than the gas sample.

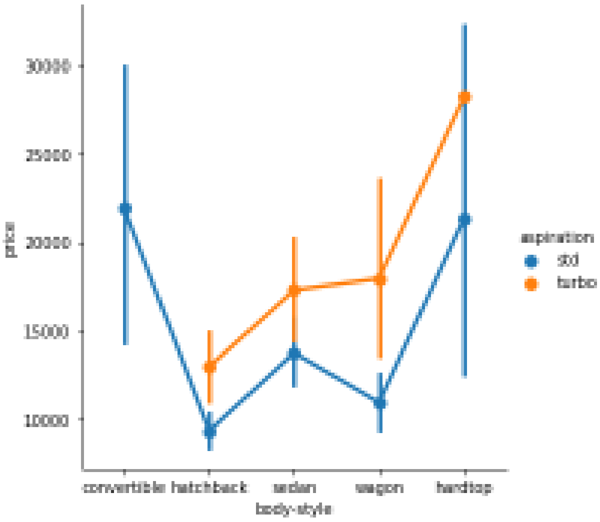
**Aspiration/ price analysis**

In this section we compare the price to body-style and also the aspiration type, which is compared between standard or turbo. The function below will plot a categorical graph to show the frequency between the two aspiration types.

```
In [22]: fig = plt.figure(figsize=(10, 10))
         sns.catplot(data = df, x = 'body-style', y = 'price', hue = 'aspiration', kind = 'point')

Out[22]: <seaborn.axisgrid.FacetGrid at 0x198f6b0dbd0>

         <Figure size 720x720 with 0 Axes>
```




**Results**

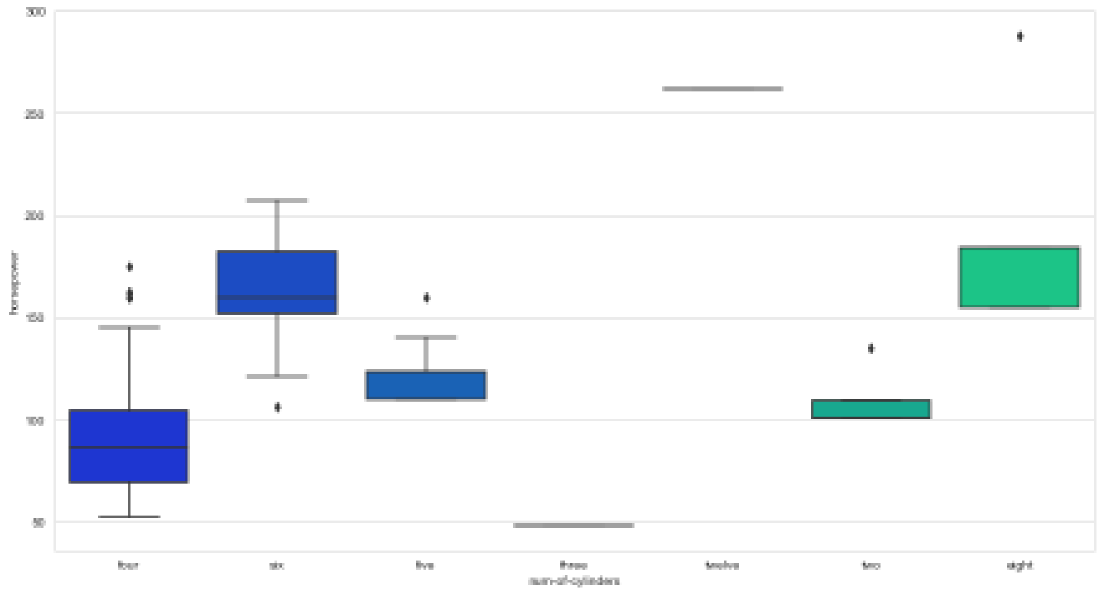
- Turbo cars are more expensive than normally aspirated cars
- Convertibles are the most expensive cars as normally aspirated cars
- Hardtops are the most expensive cars with turbos

**Engine size analysis**

Here we will compare the engine sizes. The power output (horsepower) is dependent on the number of cylinders of the different cars. The function below is used to plot a boxplot to illustrate this.

```
In [23]: sns.set_style('whitegrid')
         plt.figure(figsize=(15,8))
         sns.boxplot(data=df, x="num-of-cylinders", y="horsepower",palette="winter")

Out[23]: <AxesSubplot:xlabel='num-of-cylinders', ylabel='horsepower'>
```




**Results**

- Larger engines have more power (horsepower)
- The eight cylinder engine has the most power ranging from 155 to 300
- Cars with power above 200 have engine types six, eight and twelve cylinders

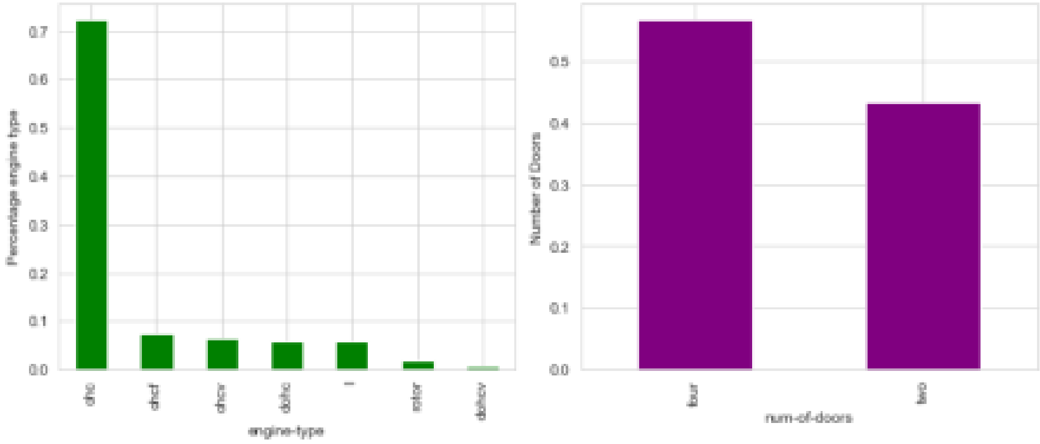
### Frequency analysis

In this section we will check the frequencies of the engine-types as well as number-of-doors. The functions below are used to plot bar charts to illustrate this.

In [24]:
```
plt.figure(1)
plt.subplot(221)
df['engine-type'].value_counts(normalize=True).plot(figsize=(10,8),kind='bar',color='green')
plt.ylabel('Percentage engine type')
plt.xlabel('engine-type');

plt.subplot(222)
df['num-of-doors'].value_counts(normalize=True).plot(figsize=(10,8),kind='bar',color='purple')
plt.ylabel('Number of Doors')
plt.xlabel('num-of-doors');

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




**THIS REPORT WAS WRITTEN BY : CHRISTOPHER KNIGHT**