# Car Price Prediction - Assignment Solution

Inorder to train a linear regression in predicting the price of cars the problem is divided into 4 sections:

- Data understanding and exploration
- Data cleaning
- Data preparation
- Model building and evaluation

## 1. Data Understanding and Exploration

Importing the required libraries .

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Reading the dataset
cars = pd.read_csv("CarPrice_Assignment.csv")
```

In [4]:
```
# Displaying all the columns in the dataset using pandas set_option
pd.set_option('display.max_columns',None)
# Let's take a look at the first few rows
cars.head()
```

Out[4]:

| | car_ID | symboling | CarName | fueltype | aspiration | doornumber | carbody | drivewheel | enginelocation | wheelbase | c |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 3 | alfa-romero giulia | gas | std | two | convertible | rwd | front | 88.6 | |
| **1** | 2 | 3 | alfa-romero stelvio | gas | std | two | convertible | rwd | front | 88.6 | |
| **2** | 3 | 1 | alfa-romero Quadrifoglio | gas | std | two | hatchback | rwd | front | 94.5 | |
| **3** | 4 | 2 | audi 100 ls | gas | std | four | sedan | fwd | front | 99.8 | |
| **4** | 5 | 2 | audi 100ls | gas | std | four | sedan | 4wd | front | 99.4 | |

In [5]:
```
# info to find summary of the dataset there 205 rows with no null values and 26 columns.
print(cars.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
None
```

**Understanding the Data Dictionary**

The data dictionary contains the meaning of various attributes or columns; some non-obvious/less clear ones are:

In [6]:
```
# Symboling is a attribute in dataset that shows the insurance risk rating, here -2 (least risky)
# a categorical variable
# Most cars are 0,1,2
cars['symboling'].astype('category').value_counts()
```

Out[6]:
```
0     67
1     54
2     32
3     27
-1    22
-2     3
Name: symboling, dtype: int64
```

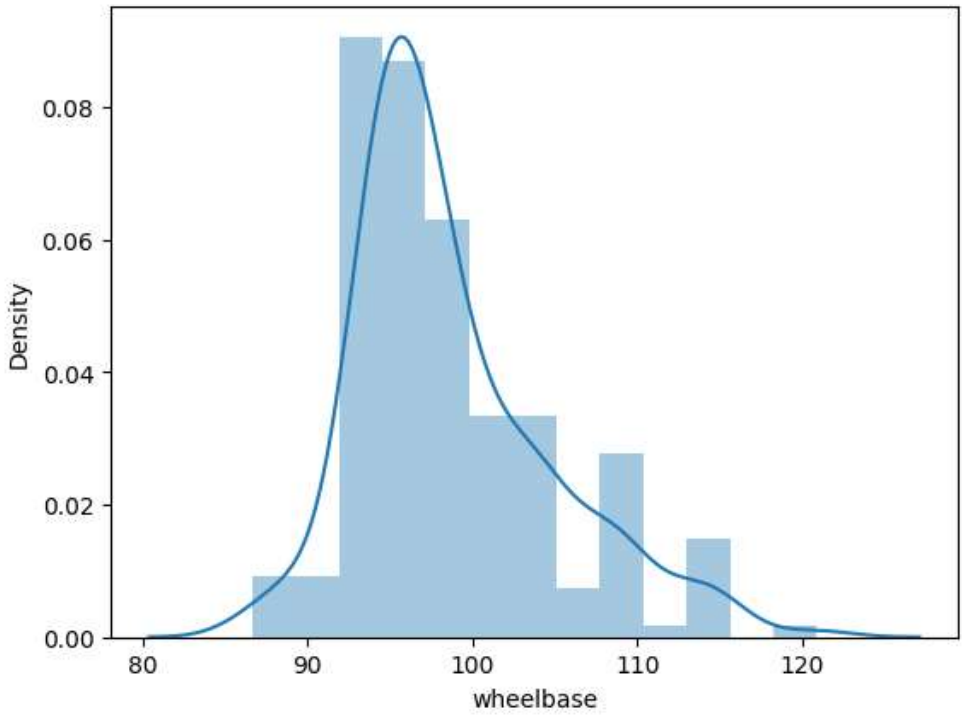
In [7]:
```
# Aspiration: An (internal combustion) engine property showing whether the oxygen intake is stand
# pressure) or through turbocharging (pressurised oxygen intake)

cars['aspiration'].astype('category').value_counts()
```

Out[7]:
```
std      168
turbo     37
Name: aspiration, dtype: int64
```

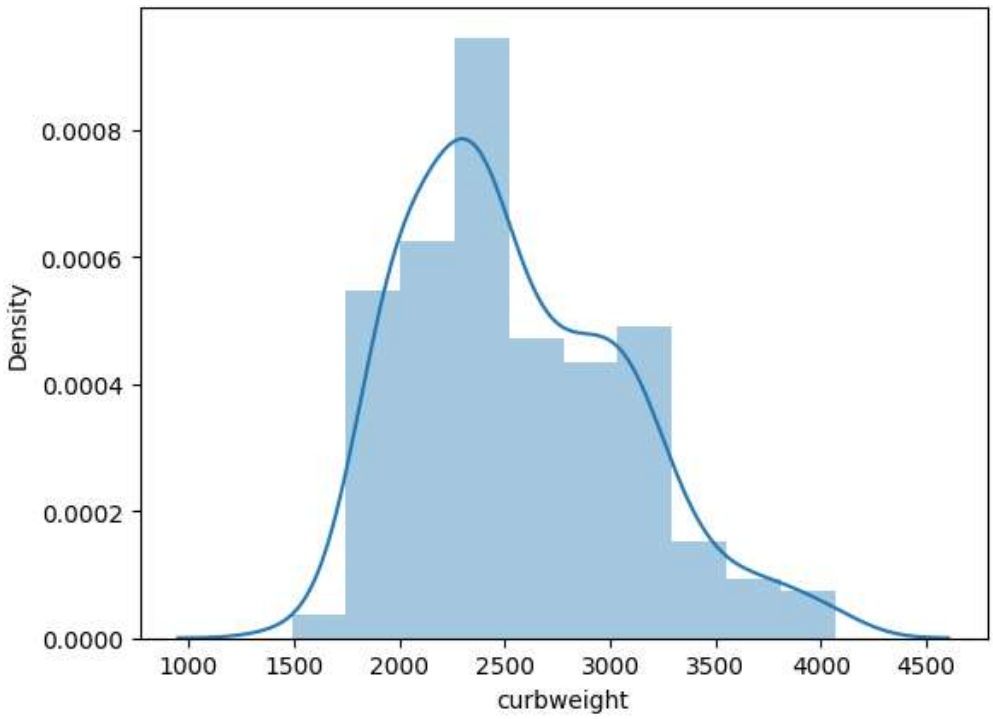
In [8]:
```
# Drivewheel: frontwheel, rarewheel or four-wheel drive
cars['drivewheel'].astype('category').value_counts()
```

Out[8]:
```
fwd    120
rwd     76
4wd      9
Name: drivewheel, dtype: int64
```

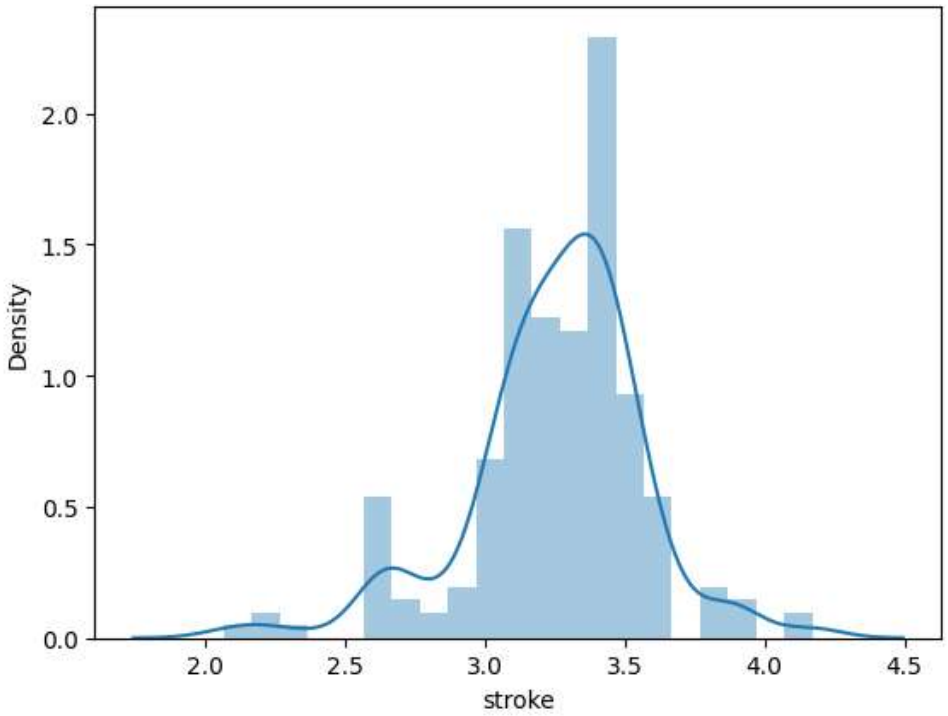
In [9]:
```
# Wheelbase: distance between centre of front and rarewheels
sns.distplot(cars['wheelbase'])
plt.show()
```



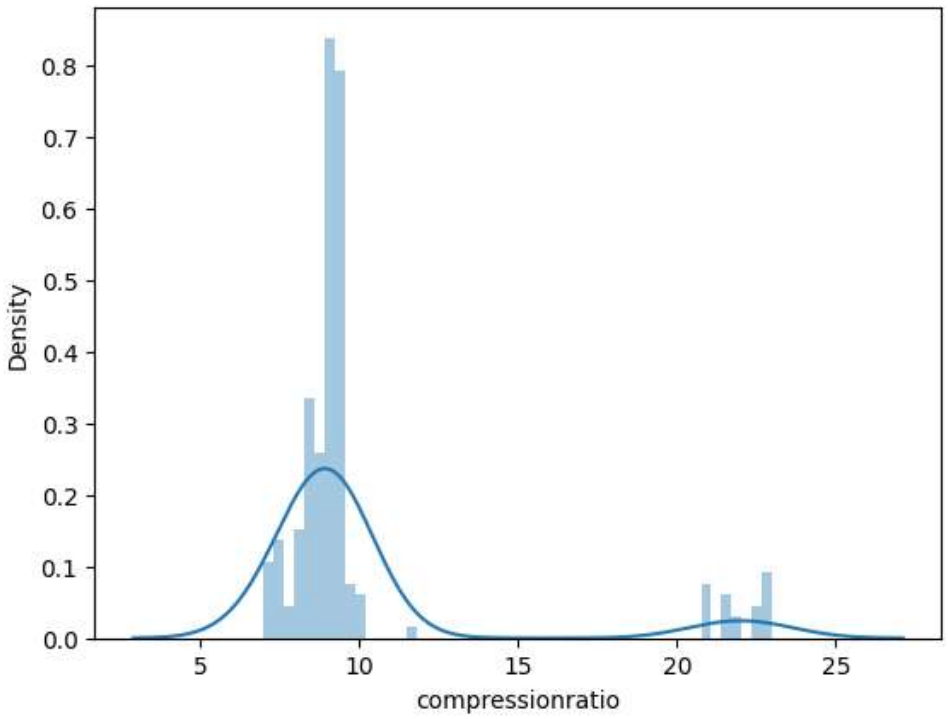
In [10]:
```
# Curbweight: weight of car without occupants or baggage
sns.distplot(cars['curbweight'])
plt.show()
```



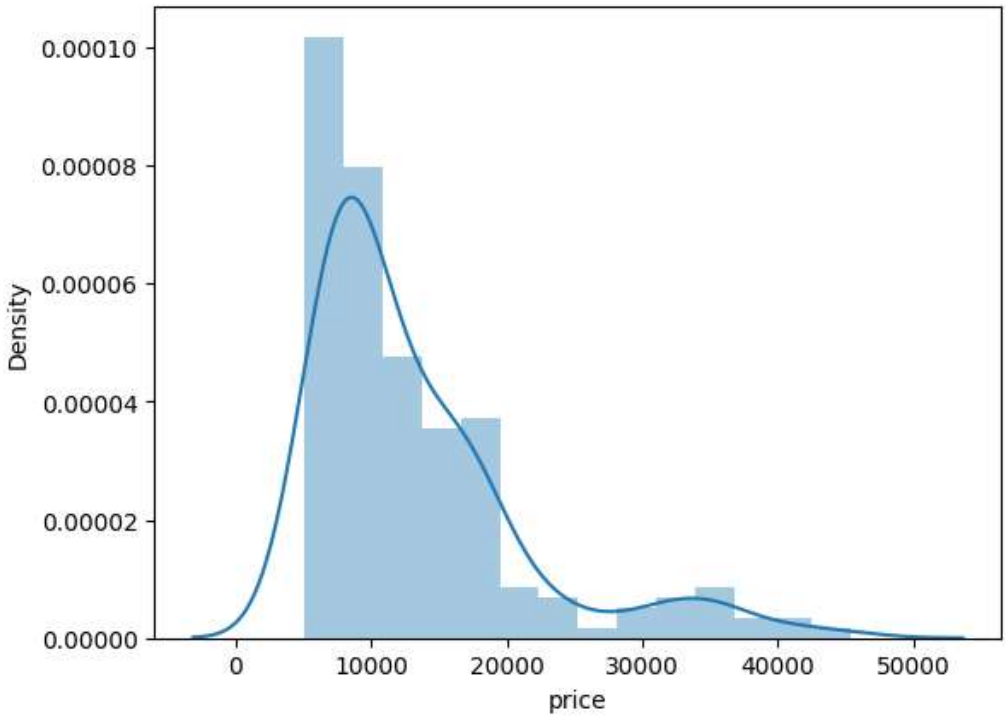

In [11]:
```
# Stroke: volume of the engine (the distance traveled by the piston in each cycle)
sns.distplot(cars['stroke'])
plt.show()
```



In [12]:
```
# Compression ration: ration of volume of compression chamber at largest capacity to least capaci
sns.distplot(cars['compressionratio'])
plt.show()
```




In [13]:
```
# Target variable: price of car
sns.distplot(cars['price'])
plt.show()
```




**Data Exploration**

To perform linear regression, the (numeric) target variable should be linearly related to *at least one another numeric variable*.

Here we subset the list of all (independent) numeric variables, and then make a **pairwise plot.**

In [14]:
```
# All numeric (float and int) variables in the dataset
cars_numeric = cars.select_dtypes(include=['float64', 'int64'])
cars_numeric.head()
```

Out[14]:

| | car_ID | symboling | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | boreratio | stroke | compression |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 3 | 88.6 | 168.8 | 64.1 | 48.8 | 2548 | 130 | 3.47 | 2.68 | |
| **1** | 2 | 3 | 88.6 | 168.8 | 64.1 | 48.8 | 2548 | 130 | 3.47 | 2.68 | |
| **2** | 3 | 1 | 94.5 | 171.2 | 65.5 | 52.4 | 2823 | 152 | 2.68 | 3.47 | |
| **3** | 4 | 2 | 99.8 | 176.6 | 66.2 | 54.3 | 2337 | 109 | 3.19 | 3.40 | |
| **4** | 5 | 2 | 99.4 | 176.6 | 66.4 | 54.3 | 2824 | 136 | 3.19 | 3.40 | |

Here, as you might notice, `car_ID` isn't of any use to building a linear regression model. Hence, we drop it.

In [15]:
```
# Dropping car_ID from cars dataset
cars = cars.drop(['car_ID'], axis=1)
cars.head()
```

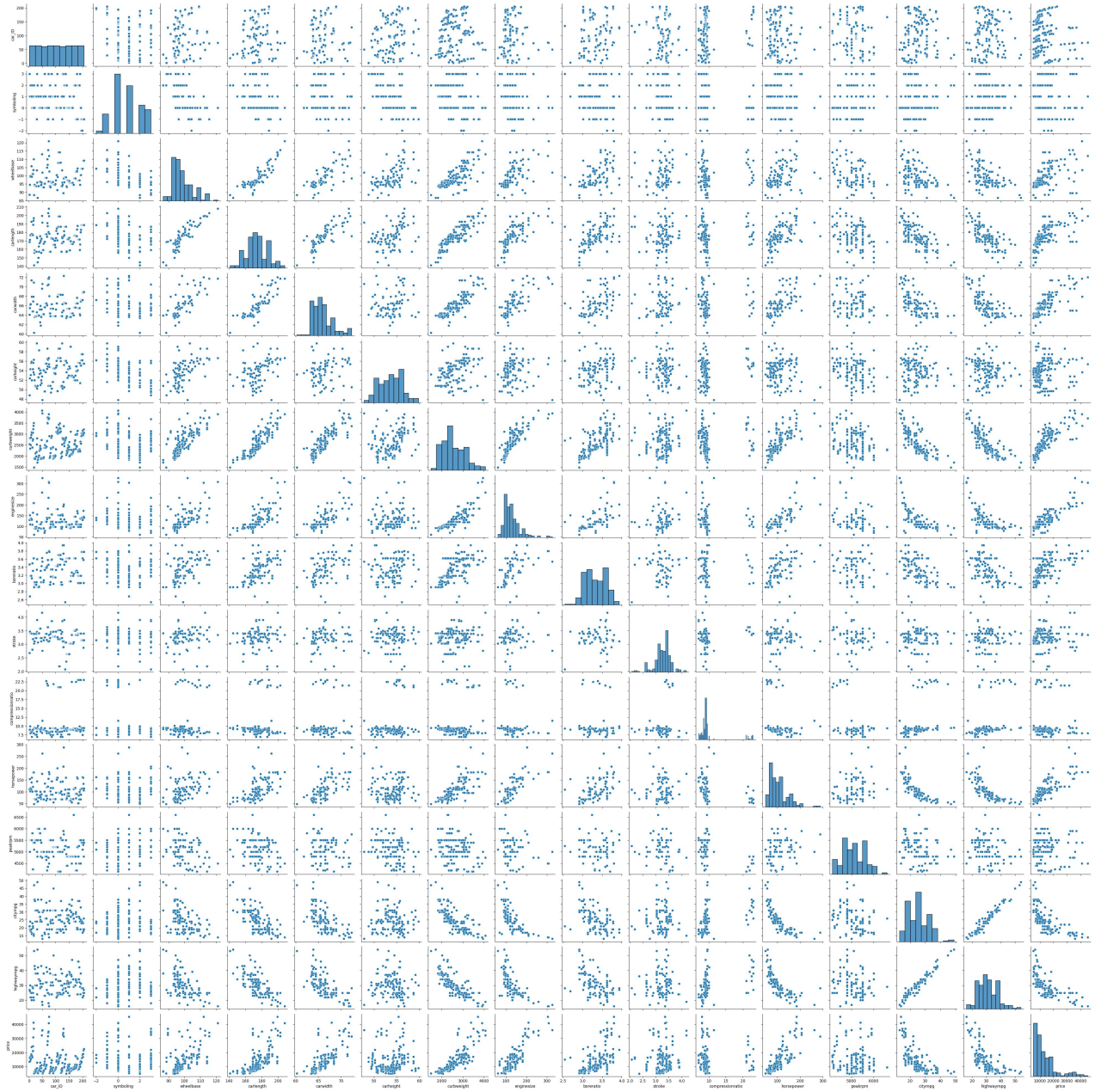
Out[15]:

| | symboling | CarName | fueltype | aspiration | doornumber | carbody | drivewheel | enginelocation | wheelbase | carlength |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | alfa-romero giulia | gas | std | two | convertible | rwd | front | 88.6 | 168.8 |
| **1** | 3 | alfa-romero stelvio | gas | std | two | convertible | rwd | front | 88.6 | 168.8 |
| **2** | 1 | alfa-romero Quadrifoglio | gas | std | two | hatchback | rwd | front | 94.5 | 171.2 |
| **3** | 2 | audi 100 ls | gas | std | four | sedan | fwd | front | 99.8 | 176.6 |
| **4** | 2 | audi 100ls | gas | std | four | sedan | 4wd | front | 99.4 | 176.6 |

Let's now make a pairwise scatter plot and observe linear relationships.

In [16]:
```
# Pairwise scatter plot

plt.figure(figsize=(20, 10))
sns.pairplot(cars_numeric)
plt.show()
```

```
<Figure size 2000x1000 with 0 Axes>
```



This is quite hard to read, and we can rather plot correlations between variables. Also, a heatmap is pretty useful to visualise multiple correlations in one plot.

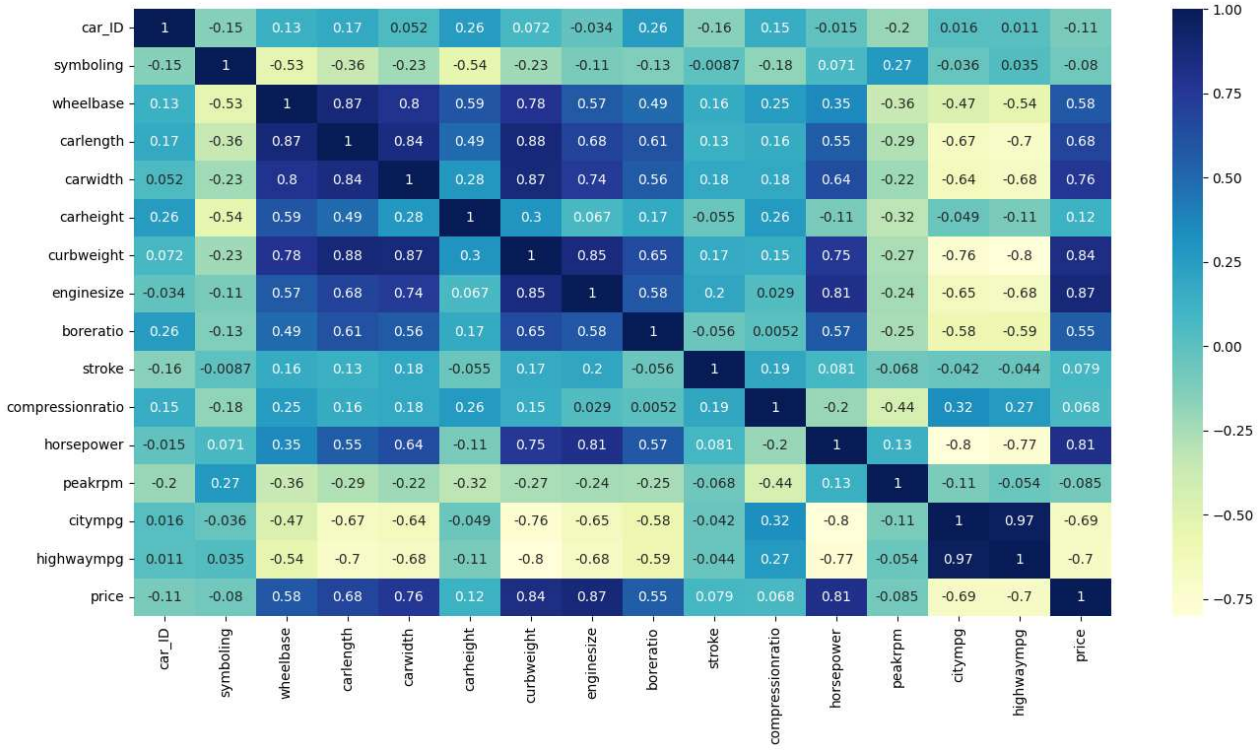
In [17]:
```
# Correlation matrix
cor = cars_numeric.corr()
cor
```

Out[17]:

| | car_ID | symboling | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | boreratio |
|---|---|---|---|---|---|---|---|---|---|
| **car_ID** | 1.000000 | -0.151621 | 0.129729 | 0.170636 | 0.052387 | 0.255960 | 0.071962 | -0.033930 | 0.260064 |
| **symboling** | -0.151621 | 1.000000 | -0.531954 | -0.357612 | -0.232919 | -0.541038 | -0.227691 | -0.105790 | -0.130051 |
| **wheelbase** | 0.129729 | -0.531954 | 1.000000 | 0.874587 | 0.795144 | 0.589435 | 0.776386 | 0.569329 | 0.488750 |
| **carlength** | 0.170636 | -0.357612 | 0.874587 | 1.000000 | 0.841118 | 0.491029 | 0.877728 | 0.683360 | 0.606454 |
| **carwidth** | 0.052387 | -0.232919 | 0.795144 | 0.841118 | 1.000000 | 0.279210 | 0.867032 | 0.735433 | 0.559150 |
| **carheight** | 0.255960 | -0.541038 | 0.589435 | 0.491029 | 0.279210 | 1.000000 | 0.295572 | 0.067149 | 0.171071 |
| **curbweight** | 0.071962 | -0.227691 | 0.776386 | 0.877728 | 0.867032 | 0.295572 | 1.000000 | 0.850594 | 0.648480 |
| **enginesize** | -0.033930 | -0.105790 | 0.569329 | 0.683360 | 0.735433 | 0.067149 | 0.850594 | 1.000000 | 0.583774 |
| **boreratio** | 0.260064 | -0.130051 | 0.488750 | 0.606454 | 0.559150 | 0.171071 | 0.648480 | 0.583774 | 1.000000 |
| **stroke** | -0.160824 | -0.008735 | 0.160959 | 0.129533 | 0.182942 | -0.055307 | 0.168790 | 0.203129 | -0.055909 |
| **compressionratio** | 0.150276 | -0.178515 | 0.249786 | 0.158414 | 0.181129 | 0.261214 | 0.151362 | 0.028971 | 0.005197 |
| **horsepower** | -0.015006 | 0.070873 | 0.353294 | 0.552623 | 0.640732 | -0.108802 | 0.750739 | 0.809769 | 0.573677 |
| **peakrpm** | -0.203789 | 0.273606 | -0.360469 | -0.287242 | -0.220012 | -0.320411 | -0.266243 | -0.244660 | -0.254976 |
| **citympg** | 0.015940 | -0.035823 | -0.470414 | -0.670909 | -0.642704 | -0.048640 | -0.757414 | -0.653658 | -0.584532 |
| **highwaympg** | 0.011255 | 0.034606 | -0.544082 | -0.704662 | -0.677218 | -0.107358 | -0.797465 | -0.677470 | -0.587012 |
| **price** | -0.109093 | -0.079978 | 0.577816 | 0.682920 | 0.759325 | 0.119336 | 0.835305 | 0.874145 | 0.553173 |

Plotting the correlations on a heatmap for better visualisation

In [18]:
```
# Figure size
plt.figure(figsize=(16,8))

# Heatmap
sns.heatmap(cor, cmap="YlGnBu", annot=True)
plt.show()
```




The above heatmap gives some useful insights:

**Positive Correlation of price with independent variables:**

- Price is highly (positively) correlated with wheelbase, carlength, carwidth, curbweight, enginesize, horsepower (all of these variables represent the size/weight/engine power of the car)

**Negative Correlation of price with independent variables:**

- Price is negatively correlated to `citympg` and `highwaympg` (-0.70 approximately). This suggest that cars having high mileage may fall in the 'economy' cars category, and are priced lower (think Maruti Alto/Swift type of cars, which are designed to be affordable by the middle class, who value mileage more than horsepower/size of car etc.)

**Correlation among independent variables:(Multicollinearity)**

- Many independent variables are highly correlated (look at the top-left part of matrix): wheelbase, carlength, carwidth, curbweight, enginesize etc. are all measures of 'size/weight' of the car, and are positively correlated

Thus, while building the model, we have to pay attention to multicollinearity.

# 2. Data Cleaning

Now the 2nd step in Model Building `Data Cleaning` is performed.

We've seen that there are no missing values in the dataset and the variables are in the correct format.

In [19]:
```
# Variable formats checking again.
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   symboling         205 non-null    int64  
 1   CarName           205 non-null    object 
 2   fueltype          205 non-null    object 
 3   aspiration        205 non-null    object 
 4   doornumber        205 non-null    object 
 5   carbody           205 non-null    object 
 6   drivewheel        205 non-null    object 
 7   enginelocation    205 non-null    object 
 8   wheelbase         205 non-null    float64
 9   carlength         205 non-null    float64
 10  carwidth          205 non-null    float64
 11  carheight         205 non-null    float64
 12  curbweight        205 non-null    int64  
 13  enginetype        205 non-null    object 
 14  cylindernumber    205 non-null    object 
 15  enginesize        205 non-null    int64  
 16  fuelsystem        205 non-null    object 
 17  boreratio         205 non-null    float64
 18  stroke            205 non-null    float64
 19  compressionratio  205 non-null    float64
 20  horsepower        205 non-null    int64  
 21  peakrpm           205 non-null    int64  
 22  citympg           205 non-null    int64  
 23  highwaympg        205 non-null    int64  
 24  price             205 non-null    float64
dtypes: float64(8), int64(7), object(10)
memory usage: 40.2+ KB
```

Next, we need to extract the company name from the column `CarName`.

In [20]:
```
# CarName: first few entries
cars['CarName'][:30]
```

Out[20]:
```
0            alfa-romero giulia
1           alfa-romero stelvio
2      alfa-romero Quadrifoglio
3                    audi 100 ls
4                     audi 100ls
5                       audi fox
6                     audi 100ls
7                      audi 5000
8                      audi 4000
9            audi 5000s (diesel)
10                       bmw 320i
11                       bmw 320i
12                         bmw x1
13                         bmw x3
14                         bmw z4
15                         bmw x4
16                         bmw x5
17                         bmw x3
18              chevrolet impala
19         chevrolet monte carlo
20           chevrolet vega 2300
21                 dodge rampage
22           dodge challenger se
23                    dodge d200
24             dodge monaco (sw)
25            dodge colt hardtop
26               dodge colt (sw)
27          dodge coronet custom
28             dodge dart custom
29    dodge coronet custom (sw)
Name: CarName, dtype: object
```

In the above insight we can observe that car company name is mentioned first then followed by its model and sub-models. E.g. alfa-romero, audi, chevrolet, dodge, bmx etc.. are the car names.

Thus, we need to simply extract the string before a space.

In [21]:
```
# Extracting carname from the varible and storing in new variable carnames

# Using split of Strings: str.split() by space
carnames = cars['CarName'].apply(lambda x: x.split(" ")[0])
carnames = pd.DataFrame(carnames) # changing to pandas DataFrame to add it to cars dataset
carnames[:30] # checking for 1st 30 values
```

Out[21]:

| | CarName |
|---|---|
| 0 | alfa-romero |
| 1 | alfa-romero |
| 2 | alfa-romero |
| 3 | audi |
| 4 | audi |
| 5 | audi |
| 6 | audi |
| 7 | audi |
| 8 | audi |
| 9 | audi |
| 10 | bmw |
| 11 | bmw |
| 12 | bmw |
| 13 | bmw |
| 14 | bmw |
| 15 | bmw |
| 16 | bmw |
| 17 | bmw |
| 18 | chevrolet |
| 19 | chevrolet |
| 20 | chevrolet |
| 21 | dodge |
| 22 | dodge |
| 23 | dodge |
| 24 | dodge |
| 25 | dodge |
| 26 | dodge |
| 27 | dodge |
| 28 | dodge |
| 29 | dodge |

Creating a new column car_company & concatinating in dataset to store the company name.

In [22]:
```
# Create a new column named car_company
cars['car_company'] = carnames
```

In [23]:
```
# Look at all values since this column will be used as a categorical variable
cars['car_company'].astype('category').value_counts()
```

Out[23]:
```
toyota         31
nissan         17
mazda          15
honda          13
mitsubishi     13
subaru         12
peugeot        11
volvo          11
dodge           9
volkswagen      9
buick           8
bmw             8
plymouth        7
audi            7
saab            6
isuzu           4
porsche         4
chevrolet       3
jaguar          3
alfa-romero     3
vw              2
renault         2
maxda           2
porcshce        1
toyouta         1
vokswagen       1
mercury         1
Nissan          1
Name: car_company, dtype: int64
```

In above output notice that **some car-company names are misspelled -** vw and vokswagen should be volkswagen, porcshce should be porsche, toyouta should be toyota, Nissan should be nissan, maxda should be mazda etc.

This is a data quality issue, that we need to solve.

In [24]:
```
# Replacing the misspelled car_company names

# volkswagen
cars.loc[(cars['car_company'] == "vw") | (cars['car_company'] == "vokswagen"), 'car_company'] = '

# porsche
cars.loc[cars['car_company'] == "porcshce", 'car_company'] = 'porsche'

# toyota
cars.loc[cars['car_company'] == "toyouta", 'car_company'] = 'toyota'

# nissan
cars.loc[cars['car_company'] == "Nissan", 'car_company'] = 'nissan'

# mazda
cars.loc[cars['car_company'] == "maxda", 'car_company'] = 'mazda'
```

In [25]:
```
cars['car_company'].astype('category').value_counts()
```

Out[25]:
```
toyota         32
nissan         18
mazda          17
mitsubishi     13
honda          13
volkswagen     12
subaru         12
peugeot        11
volvo          11
dodge           9
buick           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: car_company, dtype: int64
```

The `car_company` variable is from Data Quality issues.But now it's of no use in model buiding so removing it.

In [26]:
```
# Drop carname variable
cars = cars.drop('CarName', axis=1)
```

In [27]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   symboling         205 non-null    int64  
 1   fueltype          205 non-null    object 
 2   aspiration        205 non-null    object 
 3   doornumber        205 non-null    object 
 4   carbody           205 non-null    object 
 5   drivewheel        205 non-null    object 
 6   enginelocation    205 non-null    object 
 7   wheelbase         205 non-null    float64
 8   carlength         205 non-null    float64
 9   carwidth          205 non-null    float64
 10  carheight         205 non-null    float64
 11  curbweight        205 non-null    int64  
 12  enginetype        205 non-null    object 
 13  cylindernumber    205 non-null    object 
 14  enginesize        205 non-null    int64  
 15  fuelsystem        205 non-null    object 
 16  boreratio         205 non-null    float64
 17  stroke            205 non-null    float64
 18  compressionratio  205 non-null    float64
 19  horsepower        205 non-null    int64  
 20  peakrpm           205 non-null    int64  
 21  citympg           205 non-null    int64  
 22  highwaympg        205 non-null    int64  
 23  price             205 non-null    float64
 24  car_company       205 non-null    object 
dtypes: float64(8), int64(7), object(10)
memory usage: 40.2+ KB
```

In [28]:
```
# Checking for outliers using describe
cars.describe()
# From below insight we can observe that dataset is free from extreme outliers.
```

Out[28]:

| | symboling | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | boreratio | stroke | co |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 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 | 3.329756 | 3.255415 | |
| **std** | 1.245307 | 6.021776 | 12.337289 | 2.145204 | 2.443522 | 520.680204 | 41.642693 | 0.270844 | 0.313597 | |
| **min** | -2.000000 | 86.600000 | 141.100000 | 60.300000 | 47.800000 | 1488.000000 | 61.000000 | 2.540000 | 2.070000 | |
| **25%** | 0.000000 | 94.500000 | 166.300000 | 64.100000 | 52.000000 | 2145.000000 | 97.000000 | 3.150000 | 3.110000 | |
| **50%** | 1.000000 | 97.000000 | 173.200000 | 65.500000 | 54.100000 | 2414.000000 | 120.000000 | 3.310000 | 3.290000 | |
| **75%** | 2.000000 | 102.400000 | 183.100000 | 66.900000 | 55.500000 | 2935.000000 | 141.000000 | 3.580000 | 3.410000 | |
| **max** | 3.000000 | 120.900000 | 208.100000 | 72.300000 | 59.800000 | 4066.000000 | 326.000000 | 3.940000 | 4.170000 | |

# 3. Data Preparation

**Data Preparation**

The 3rd step in model building i.e, Data Preparation. Let's now prepare the data and build the model.

In [29]:
```
# Checking the dataset again.
cars.head()
```

Out[29]:

| | symboling | fueltype | aspiration | doornumber | carbody | drivewheel | enginelocation | wheelbase | carlength | carwidth |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | gas | std | two | convertible | rwd | front | 88.6 | 168.8 | 64.1 |
| **1** | 3 | gas | std | two | convertible | rwd | front | 88.6 | 168.8 | 64.1 |
| **2** | 1 | gas | std | two | hatchback | rwd | front | 94.5 | 171.2 | 65.5 |
| **3** | 2 | gas | std | four | sedan | fwd | front | 99.8 | 176.6 | 66.2 |
| **4** | 2 | gas | std | four | sedan | 4wd | front | 99.4 | 176.6 | 66.4 |

Notice that two variables - `doornumber` and `cylindernumber` are numeric types with the numbers written as words. Let's map these to actual numbers.

In [30]:
```
# Checking the different levels of 'cylindernumber'
cars['cylindernumber'].astype('category').value_counts()
```

Out[30]:
```
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: cylindernumber, dtype: int64
```

In [31]:
```
# Checking the different levels of 'doornumber'
cars['doornumber'].astype('category').value_counts()
```

Out[31]:
```
four    115
two      90
Name: doornumber, dtype: int64
```

In [32]:
```
# A function to map the categorical levels to actual numbers. Using the categorical levels above
def num_map(x):
    return x.map({'two': 2, "three": 3, "four": 4, "five": 5, "six": 6, "eight": 8, "twelve": 12}

# Applying the function to the two columns
cars[['cylindernumber', 'doornumber']] = cars[['cylindernumber', 'doornumber']].apply(num_map)
# We can also use lambda function directly without need of explicit function.
```

Now there are some categorical variables for which we cannot directly convert to numerical types as done above. So, we make use of get_dummies of pandas to suitable dummies for different levels of the variables. Let's now create dummy variables for the categorical variables.

In [33]:
```
# First we subset all the categorical variables and store in a variable.
cars_categorical = cars.select_dtypes(include=['object'])
cars_categorical.head()
```

Out[33]:

| | fueltype | aspiration | carbody | drivewheel | enginelocation | enginetype | fuelsystem | car_company |
|---|---|---|---|---|---|---|---|---|
| **0** | gas | std | convertible | rwd | front | dohc | mpfi | alfa-romero |
| **1** | gas | std | convertible | rwd | front | dohc | mpfi | alfa-romero |
| **2** | gas | std | hatchback | rwd | front | ohcv | mpfi | alfa-romero |
| **3** | gas | std | sedan | fwd | front | ohc | mpfi | audi |
| **4** | gas | std | sedan | 4wd | front | ohc | mpfi | audi |

In [34]:
```
# Convert them into dummies and dropping the first dummy to get k-1 dummies where 'k' is total du
cars_dummies = pd.get_dummies(cars_categorical, drop_first=True)
cars_dummies.head()
```

Out[34]:

| | fueltype_gas | aspiration_turbo | carbody_hardtop | carbody_hatchback | carbody_sedan | carbody_wagon | drivewheel_fwd |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **3** | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| **4** | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

In [35]:
```
# Dropping categorical variable columns present in the dataset and inserting new dummies.
cars = cars.drop(list(cars_categorical.columns), axis=1)
```

In [36]:
```
# Concatenating dummy variables with original cars dataframe.
cars = pd.concat([cars, cars_dummies], axis=1)
```

In [37]:
```
# Checking the first few rows
cars.head()
```

Out[37]:

| | symboling | doornumber | wheelbase | carlength | carwidth | carheight | curbweight | cylindernumber | enginesize | boreratio |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 2 | 88.6 | 168.8 | 64.1 | 48.8 | 2548 | 4 | 130 | 3.47 |
| **1** | 3 | 2 | 88.6 | 168.8 | 64.1 | 48.8 | 2548 | 4 | 130 | 3.47 |
| **2** | 1 | 2 | 94.5 | 171.2 | 65.5 | 52.4 | 2823 | 6 | 152 | 2.68 |
| **3** | 2 | 4 | 99.8 | 176.6 | 66.2 | 54.3 | 2337 | 4 | 109 | 3.19 |
| **4** | 2 | 4 | 99.4 | 176.6 | 66.4 | 54.3 | 2824 | 5 | 136 | 3.19 |

## 3. Model Building and Evaluation

The 3rd and most important step in Machine Learning is Model Building and Evaluating it.

Let's start building the model. The first step to model building is the usual test-train split.

In [38]:
```
# Split the dataframe into train and test sets
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(cars, train_size=0.7, test_size=0.3, random_state=100)
```

## Scaling

Now we have done the test-train split, we need to scale the variables for better interpretability. But we only need to scale the numeric columns and not the dummy variables. Let's take a look at the list of numeric variables we had created in the beginning. Also, the scaling has to be done only on the train dataset as you don't want the model to learn anything from the test data.

In [39]:
```
# These are the numeric variables that we done at first and now we need to scale them
cars_numeric.columns
```

Out[39]:
```
Index(['car_ID', 'symboling', 'wheelbase', 'carlength', 'carwidth',
       'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

Now scaling all these columns using StandardScaler. We can also use another scaler named Min_Max_Scaler that scales values between 0 and 1.Also, notice that we had converted two other variables, i.e, 'doornumber' and 'cylindernumber' to numeric types. So you would need to include them in the varlist as well

In [40]:
```
# Importing the StandardScaler()
from sklearn.preprocessing import StandardScaler

# Creating a scaling object
scaler = StandardScaler()

# Creating a list of the variables that we need to scale
varlist = ['symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight','curbweight', 'enginesi
           'compressionratio','horsepower', 'peakrpm', 'citympg', 'highwaympg', 'doornumber', 'cy

# Scaling these variables using 'fit_transform'
df_train[varlist] = scaler.fit_transform(df_train[varlist])
```

In [41]:
```
# Looking at the train dataframe
df_train.head()
```

Out[41]:

| | symboling | doornumber | wheelbase | carlength | carwidth | carheight | curbweight | cylindernumber | enginesize | borer |
|---|---|---|---|---|---|---|---|---|---|---|
| **122** | 0.170159 | 0.887412 | -0.811836 | -0.487238 | -0.924500 | -1.134628 | -0.642128 | -0.351431 | -0.660242 | -1.297 |
| **125** | 1.848278 | -1.126872 | -0.677177 | -0.359789 | 1.114978 | -1.382026 | 0.439415 | -0.351431 | 0.637806 | 2.432 |
| **166** | 0.170159 | -1.126872 | -0.677177 | -0.375720 | -0.833856 | -0.392434 | -0.441296 | -0.351431 | -0.660242 | -0.259 |
| **1** | 1.848278 | -1.126872 | -1.670284 | -0.367754 | -0.788535 | -1.959288 | 0.015642 | -0.351431 | 0.123485 | 0.625 |
| **199** | -1.507960 | 0.887412 | 0.972390 | 1.225364 | 0.616439 | 1.627983 | 1.137720 | -0.351431 | 0.123485 | 1.201 |

As we know, the variables have been appropriately scaled.

In [42]:
```
# Split the train dataset into X_train(excluding 'price') and y_train('price')

y_train = df_train.pop('price')
X_train = df_train
```

## Building the first model with all the features

In [43]:
```
# Creating object of Linear Regression
lm = LinearRegression()

# Fit a line
lm.fit(X_train, y_train)
```

Out[43]:
```
LinearRegression

LinearRegression()
```

In [44]:
```
# Print the coefficients and intercept
print(lm.coef_)
print(lm.intercept_)
```

```
[-8.85082799e-03  2.05396206e-02  2.15459688e-01 -1.20891864e-01
  2.14312437e-01 -1.69734529e-01  2.71975134e-01 -2.88819050e-01
  9.98176607e-01 -3.16364315e-01 -1.09027004e-01 -4.45741225e-01
 -1.28216803e-01  1.89046429e-01  6.10383234e-02  7.95225308e-02
 -7.13939879e-01  3.92563218e-01 -5.68397039e-01 -6.33741325e-01
 -5.33209547e-01 -4.44213621e-01 -6.30834556e-02  4.57246372e-02
  9.85062979e-01  9.02650092e-01  9.94703704e-01  2.82949424e-01
  6.10461201e-01  8.12155616e-03  1.12048455e+00  1.19956250e-01
 -3.05423110e-01  7.13939879e-01 -4.44089210e-16 -4.00301122e-02
 -5.85575952e-02  5.55111512e-16 -1.28333697e-01  1.01695043e+00
  1.12409662e-01 -5.91117054e-01 -7.18611643e-01 -5.01548738e-01
 -3.06608953e-01 -2.48183458e-01 -1.57753086e-01 -1.11022302e-16
 -8.09182737e-01 -2.48597762e-01 -1.33327298e+00 -7.13453493e-01
  7.94779273e-01 -3.31355674e-01  7.05291444e-01 -3.74601778e-01
 -1.77619987e-01 -1.76633338e-01  1.86531496e-02]
0.9701688083156828
```

### Model Building Using RFE

- Now, we have close to 60 features in the X_train set.
- But for building the model we need to have less features that will give more accurate predictions.
- But it is not possible to manually eliminate these features. So, now building a model using recursive feature elimination(RFE) to select features.
- We'll first start off with an arbitrary number of features (15 seems to be a good number to begin with), and then use the `statsmodels` library to build models using the reduced features (because `SKLearn` doesn't have `Adjusted R- squared` that `statsmodels` has).

In [45]:
```
# Import RFE
from sklearn.feature_selection import RFE

# RFE with 15 features
lm = LinearRegression()
rfe1 = RFE(lm, n_features_to_select=15, step=1)

# Fit with 15 features
rfe1.fit(X_train, y_train)

# Print the boolean results
print(rfe1.support_,'\n\n')

# Displaying column-wise ranking given by RFE
print(rfe1.ranking_,'\n\n')

# Displaying the columns of ranking 1
print(X_train.columns[rfe1.support_])
```

```
[False False False False  True False False False  True False False  True
 False False False False  True False False False False False False False
  True False  True False  True False  True False False  True False False
 False False False  True False False False False False False  True False
  True False  True False False  True False  True False False False]


[41 39 28 31  1 26 17 32  1 18 33  1 23 24 37 30  1 16 14 12 13 15 35 40
  1 22  1 20  1 42  1 36 10  1 43 27 19 44  9  1 34  8  6  4 25 29  1 45
  1  2  1  7 11  1 21  1  3  5 38]


Index(['carwidth', 'enginesize', 'compressionratio', 'fueltype_gas',
       'enginelocation_rear', 'enginetype_l', 'enginetype_ohcf',
       'enginetype_rotor', 'fuelsystem_idi', 'car_company_bmw',
       'car_company_mazda', 'car_company_mitsubishi', 'car_company_peugeot',
       'car_company_renault', 'car_company_subaru'],
      dtype='object')
```

**Model Building and Evaluation**

Now buiding the model by `statsmodels` using the features given by RFE

In [46]:
```
# Import statsmodels
import statsmodels.api as sm

# Subset/Including only the features selected by rfe1
col1 = X_train.columns[rfe1.support_]

# Subsetting training data for 15 selected columns
X_train_rfe1 = X_train[col1]

# Add a constant to the model because in statsmodels it fits the line by default through origin.
X_train_rfe1 = sm.add_constant(X_train_rfe1)
X_train_rfe1.head()
```

Out[46]:

| | const | carwidth | enginesize | compressionratio | fueltype_gas | enginelocation_rear | enginetype_l | enginetype_ohcf | er |
|---|---|---|---|---|---|---|---|---|---|
| **122** | 1.0 | -0.924500 | -0.660242 | -0.172569 | 1 | 0 | 0 | 0 | |
| **125** | 1.0 | 1.114978 | 0.637806 | -0.146125 | 1 | 0 | 0 | 0 | |
| **166** | 1.0 | -0.833856 | -0.660242 | -0.172569 | 1 | 0 | 0 | 0 | |
| **1** | 1.0 | -0.788535 | 0.123485 | -0.278345 | 1 | 0 | 0 | 0 | |
| **199** | 1.0 | 0.616439 | 0.123485 | -0.675002 | 1 | 0 | 0 | 0 | |

In [47]:
```
# Fitting the model with 15 variables
lm1 = sm.OLS(y_train, X_train_rfe1).fit()
print(lm1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.920
Model:                            OLS   Adj. R-squared:                  0.912
Method:                 Least Squares   F-statistic:                     114.1
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           4.59e-64
Time:                        22:09:53   Log-Likelihood:                -22.314
No. Observations:                 143   AIC:                             72.63
Df Residuals:                     129   BIC:                             114.1
Df Model:                          13
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        0.4776      0.162      2.940      0.004       0.156       0.799
carwidth                     0.4304      0.046      9.403      0.000       0.340       0.521
enginesize                   0.4790      0.045     10.594      0.000       0.390       0.568
compressionratio            -0.4505      0.162     -2.781      0.006      -0.771      -0.130
fueltype_gas                -0.6269      0.208     -3.016      0.003      -1.038      -0.216
enginelocation_rear          1.4894      0.211      7.071      0.000       1.073       1.906
enginetype_l                 0.8983      0.307      2.921      0.004       0.290       1.507
enginetype_ohcf              0.6436      0.109      5.905      0.000       0.428       0.859
enginetype_rotor             0.9536      0.188      5.076      0.000       0.582       1.325
fuelsystem_idi               1.1045      0.369      2.995      0.003       0.375       1.834
car_company_bmw              1.0678      0.131      8.141      0.000       0.808       1.327
car_company_mazda           -0.2439      0.106     -2.299      0.023      -0.454      -0.034
car_company_mitsubishi      -0.3956      0.112     -3.517      0.001      -0.618      -0.173
car_company_peugeot         -1.4098      0.338     -4.171      0.000      -2.079      -0.741
car_company_renault         -0.6778      0.214     -3.173      0.002      -1.100      -0.255
car_company_subaru          -0.8458      0.122     -6.915      0.000      -1.088      -0.604
==============================================================================
Omnibus:                        8.408   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.015   Jarque-Bera (JB):                9.825
Skew:                           0.399   Prob(JB):                      0.00735
Kurtosis:                       4.005   Cond. No.                     3.60e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.23e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

- The model seems to be doing a good job.
- Because the p-values are less than 0.05 and adjusted R-squared is 0.9(approx) means 90% of variability of dependent variable is explained by the independent varibles
- Now looking at the VIF values.

In [48]:
```
# Checking for the VIF values of the feature/independent variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [49]:
```
# Creating a dataframe that will contain the names of all the feature variables and their respect
vif = pd.DataFrame()
vif['Features'] = X_train_rfe1.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe1.values, i) for i in range(X_train_rfe1.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

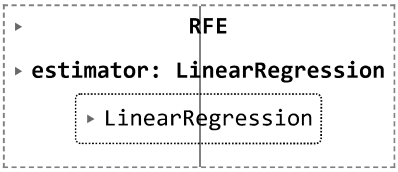
Out[49]:

| | Features | VIF |
|---|---|---|
| 4 | fueltype_gas | inf |
| 5 | enginelocation_rear | inf |
| 7 | enginetype_ohcf | inf |
| 9 | fuelsystem_idi | inf |
| 15 | car_company_subaru | inf |
| 3 | compressionratio | 42.32 |
| 13 | car_company_peugeot | 9.73 |
| 6 | enginetype_l | 8.99 |
| 1 | carwidth | 3.38 |
| 2 | enginesize | 3.30 |
| 8 | enginetype_rotor | 1.55 |
| 11 | car_company_mazda | 1.50 |
| 12 | car_company_mitsubishi | 1.20 |
| 10 | car_company_bmw | 1.12 |
| 14 | car_company_renault | 1.01 |
| 0 | const | 0.00 |

- In above results of VIF some features have a VIF value `inf` that means R-squared is 1 which indicates that the variable is highly/strongly correlated with other independent variables in the dataset.
- These variables aren't of use.
- But manually elimination is time consuming and makes the code unnecessarily long. Now buiding a model with 10 features by using RFE.

In [50]:
```
# RFE with 10 features
from sklearn.feature_selection import RFE

# RFE with 10 features
lm = LinearRegression()
rfe2 = RFE(lm, n_features_to_select=10, step=1)

# Fit with 10 features
rfe2.fit(X_train, y_train)
```

Out[50]:



In [51]:
```
# Subset the features selected by rfe2
col2 = X_train.columns[rfe2.support_]

# Subsetting training data for 10 selected columns
X_train_rfe2 = X_train[col2]

# Add a constant to the model
X_train_rfe2 = sm.add_constant(X_train_rfe2)

# Fitting the model with 10 variables
lm2 = sm.OLS(y_train, X_train_rfe2).fit()
print(lm2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.907
Model:                            OLS   Adj. R-squared:                  0.901
Method:                 Least Squares   F-statistic:                     144.3
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           3.98e-64
Time:                        22:09:53   Log-Likelihood:                -33.027
No. Observations:                 143   AIC:                             86.05
Df Residuals:                     133   BIC:                             115.7
Df Model:                           9
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                       -0.0506      0.030     -1.675      0.096      -0.110       0.009
carwidth                     0.4611      0.047      9.801      0.000       0.368       0.554
enginesize                   0.4806      0.047     10.124      0.000       0.387       0.575
enginelocation_rear          1.4538      0.223      6.519      0.000       1.013       1.895
enginetype_l                 0.9450      0.326      2.902      0.004       0.301       1.589
enginetype_ohcf              0.6553      0.115      5.678      0.000       0.427       0.884
enginetype_rotor             0.6927      0.172      4.029      0.000       0.353       1.033
car_company_bmw              1.1247      0.138      8.162      0.000       0.852       1.397
car_company_peugeot         -1.2582      0.354     -3.550      0.001      -1.959      -0.557
car_company_renault         -0.6256      0.226     -2.770      0.006      -1.072      -0.179
car_company_subaru          -0.7985      0.129     -6.192      0.000      -1.054      -0.543
==============================================================================
Omnibus:                        5.615   Durbin-Watson:                   1.942
Prob(Omnibus):                  0.060   Jarque-Bera (JB):                5.456
Skew:                           0.349   Prob(JB):                       0.0654
Kurtosis:                       3.655   Cond. No.                     2.00e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.3e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Note that the adjusted R-squared value hasn't dropped much. It has gone from `0.912` to `0.901`. So 10 variables seems to be a good number to start with.

In [52]:
```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [53]:
```
# Create a dataframe that will contain the names of all the feature variables and their respective
vif = pd.DataFrame()
vif['Features'] = X_train_rfe2.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe2.values, i) for i in range(X_train_rfe2.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[53]:

| | Features | VIF |
|---|---|---|
| 3 | enginelocation_rear | inf |
| 5 | enginetype_ohcf | inf |
| 10 | car_company_subaru | inf |
| 8 | car_company_peugeot | 9.49 |
| 4 | enginetype_l | 8.95 |
| 2 | enginesize | 3.23 |
| 1 | carwidth | 3.17 |
| 0 | const | 1.31 |
| 6 | enginetype_rotor | 1.15 |
| 7 | car_company_bmw | 1.09 |
| 9 | car_company_renault | 1.01 |

- There are still some variables that need to be dropped because they have VIF value `inf` .
- But we drop only one feature at a time, then again checking for VIF of remaining feature.
- Let's start by dropping `car_company_subaru` .

In [54]:
```
X_train_rfe2.drop('car_company_subaru', axis = 1, inplace = True)
```

In [55]:
```
# Refitting with 9 variables

X_train_rfe2 = sm.add_constant(X_train_rfe2)

# Fitting the model with 9 variables
lm2 = sm.OLS(y_train, X_train_rfe2).fit()
print(lm2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.907
Model:                            OLS   Adj. R-squared:                  0.901
Method:                 Least Squares   F-statistic:                     144.3
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           3.98e-64
Time:                        22:09:54   Log-Likelihood:                -33.027
No. Observations:                 143   AIC:                             86.05
Df Residuals:                     133   BIC:                             115.7
Df Model:                           9
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                       -0.0506      0.030     -1.675      0.096      -0.110       0.009
carwidth                     0.4611      0.047      9.801      0.000       0.368       0.554
enginesize                   0.4806      0.047     10.124      0.000       0.387       0.575
enginelocation_rear          2.2524      0.346      6.518      0.000       1.569       2.936
enginetype_l                 0.9450      0.326      2.902      0.004       0.301       1.589
enginetype_ohcf             -0.1433      0.101     -1.422      0.157      -0.343       0.056
enginetype_rotor             0.6927      0.172      4.029      0.000       0.353       1.033
car_company_bmw              1.1247      0.138      8.162      0.000       0.852       1.397
car_company_peugeot         -1.2582      0.354     -3.550      0.001      -1.959      -0.557
car_company_renault         -0.6256      0.226     -2.770      0.006      -1.072      -0.179
==============================================================================
Omnibus:                        5.615   Durbin-Watson:                   1.942
Prob(Omnibus):                  0.060   Jarque-Bera (JB):                5.456
Skew:                           0.349   Prob(JB):                       0.0654
Kurtosis:                       3.655   Cond. No.                         23.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [56]:
```
# Creating a dataframe that will contain the names of all the feature variables and their respect

vif = pd.DataFrame()
vif['Features'] = X_train_rfe2.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe2.values, i) for i in range(X_train_rfe2.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[56]:

| | Features | VIF |
|---|---|---|
| 8 | car_company_peugeot | 9.49 |
| 4 | enginetype_l | 8.95 |
| 2 | enginesize | 3.23 |
| 1 | carwidth | 3.17 |
| 0 | const | 1.31 |
| 3 | enginelocation_rear | 1.19 |
| 6 | enginetype_rotor | 1.15 |
| 5 | enginetype_ohcf | 1.12 |
| 7 | car_company_bmw | 1.09 |
| 9 | car_company_renault | 1.01 |

- The infinite VIFs have now dropped to a suitable value by excluding `inf`.
- A good feature will have a VIF value less than 5.
- But first we consider p-value if features having p-value > 0.05 we need to remove them.
- So, `enginetype_ohfc` has become insignificant. We need to drop it.

In [57]:
```
X_train_rfe2.drop('enginetype_ohcf', axis = 1, inplace = True)
```

In [58]:
```
# Refitting with 8 variables
X_train_rfe2 = sm.add_constant(X_train_rfe2)

# Fitting the model with 8 variables
lm2 = sm.OLS(y_train, X_train_rfe2).fit()
print(lm2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.906
Model:                            OLS   Adj. R-squared:                  0.900
Method:                 Least Squares   F-statistic:                     160.8
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           8.22e-65
Time:                        22:09:54   Log-Likelihood:                -34.105
No. Observations:                 143   AIC:                             86.21
Df Residuals:                     134   BIC:                             112.9
Df Model:                           8
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                   -0.0635      0.029     -2.195      0.030      -0.121      -0.006
carwidth                 0.4596      0.047      9.735      0.000       0.366       0.553
enginesize               0.4870      0.047     10.264      0.000       0.393       0.581
enginelocation_rear      2.1107      0.332      6.355      0.000       1.454       2.768
enginetype_l             0.9641      0.327      2.952      0.004       0.318       1.610
enginetype_rotor         0.7137      0.172      4.151      0.000       0.374       1.054
car_company_bmw          1.1312      0.138      8.183      0.000       0.858       1.405
car_company_peugeot     -1.2647      0.356     -3.555      0.001      -1.968      -0.561
car_company_renault     -0.6133      0.227     -2.707      0.008      -1.061      -0.165
==============================================================================
Omnibus:                        5.533   Durbin-Watson:                   1.944
Prob(Omnibus):                  0.063   Jarque-Bera (JB):                5.168
Skew:                           0.374   Prob(JB):                       0.0755
Kurtosis:                       3.555   Cond. No.                         23.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [59]:
```
# Create a dataframe that will contain the names of all the feature variables and their respective

vif = pd.DataFrame()
vif['Features'] = X_train_rfe2.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe2.values, i) for i in range(X_train_rfe2.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[59]:

| | Features | VIF |
|---|---|---|
| 7 | car_company_peugeot | 9.49 |
| 4 | enginetype_l | 8.94 |
| 2 | enginesize | 3.20 |
| 1 | carwidth | 3.17 |
| 0 | const | 1.19 |
| 5 | enginetype_rotor | 1.14 |
| 3 | enginelocation_rear | 1.09 |
| 6 | car_company_bmw | 1.09 |
| 8 | car_company_renault | 1.01 |

- The variables seem significant, but we still have few high VIFs(>5).
- Let's drop them one by one and see if the Adjusted R-squared score is getting affected.

In [60]:
```
X_train_rfe2.drop('car_company_peugeot', axis = 1, inplace = True)
```

In [61]:
```
# Refitting with 7 variables
X_train_rfe2 = sm.add_constant(X_train_rfe2)

# Fitting the model with 7 variables
lm2 = sm.OLS(y_train, X_train_rfe2).fit()
print(lm2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.897
Model:                            OLS   Adj. R-squared:                  0.891
Method:                 Least Squares   F-statistic:                     167.5
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           2.49e-63
Time:                        22:09:54   Log-Likelihood:                -40.550
No. Observations:                 143   AIC:                             97.10
Df Residuals:                     135   BIC:                             120.8
Df Model:                           7
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                   -0.0663      0.030     -2.198      0.030      -0.126      -0.007
carwidth                 0.4115      0.047      8.729      0.000       0.318       0.505
enginesize               0.5101      0.049     10.415      0.000       0.413       0.607
enginelocation_rear      2.0560      0.346      5.946      0.000       1.372       2.740
enginetype_l            -0.1238      0.119     -1.040      0.300      -0.359       0.112
enginetype_rotor         0.7431      0.179      4.152      0.000       0.389       1.097
car_company_bmw          1.1269      0.144      7.822      0.000       0.842       1.412
car_company_renault     -0.5991      0.236     -2.538      0.012      -1.066      -0.132
==============================================================================
Omnibus:                       10.615   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.005   Jarque-Bera (JB):               11.115
Skew:                           0.570   Prob(JB):                      0.00386
Kurtosis:                       3.752   Cond. No.                         16.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [62]:
```
# Create a dataframe that will contain the names of all the feature variables and their respective
vif = pd.DataFrame()
vif['Features'] = X_train_rfe2.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe2.values, i) for i in range(X_train_rfe2.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[62]:

| | Features | VIF |
|---|---|---|
| 2 | enginesize | 3.14 |
| 1 | carwidth | 2.91 |
| 0 | const | 1.19 |
| 5 | enginetype_rotor | 1.14 |
| 3 | enginelocation_rear | 1.09 |
| 4 | enginetype_l | 1.09 |
| 6 | car_company_bmw | 1.09 |
| 7 | car_company_renault | 1.00 |

- VIF of features are in satisfying conditions(<5).
- But the `enginetype_l` variable has a p-value of `0.3`.
- Let's drop it and see if it affects the model much.

In [63]:
```
# Refitting with 6 variables
X_train_rfe2.drop('enginetype_l', axis = 1, inplace = True)

X_train_rfe2 = sm.add_constant(X_train_rfe2)

# Fitting the model with 6 variables
lm2 = sm.OLS(y_train, X_train_rfe2).fit()
print(lm2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.896
Model:                            OLS   Adj. R-squared:                  0.891
Method:                 Least Squares   F-statistic:                     195.2
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           2.92e-64
Time:                        22:09:54   Log-Likelihood:                -41.121
No. Observations:                 143   AIC:                             96.24
Df Residuals:                     136   BIC:                             117.0
Df Model:                           6
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                   -0.0748      0.029     -2.578      0.011      -0.132      -0.017
carwidth                 0.3978      0.045      8.785      0.000       0.308       0.487
enginesize               0.5204      0.048     10.846      0.000       0.426       0.615
enginelocation_rear      2.0419      0.346      5.908      0.000       1.358       2.725
enginetype_rotor         0.7640      0.178      4.295      0.000       0.412       1.116
car_company_bmw          1.1294      0.144      7.838      0.000       0.844       1.414
car_company_renault     -0.5879      0.236     -2.492      0.014      -1.054      -0.121
==============================================================================
Omnibus:                        7.920   Durbin-Watson:                   1.970
Prob(Omnibus):                  0.019   Jarque-Bera (JB):                7.687
Skew:                           0.497   Prob(JB):                       0.0214
Kurtosis:                       3.549   Cond. No.                         16.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [64]:
```
# Create a dataframe that will contain the names of all the feature variables and their respectiv
vif = pd.DataFrame()
vif['Features'] = X_train_rfe2.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe2.values, i) for i in range(X_train_rfe2.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[64]:

| | Features | VIF |
|---|---|---|
| 2 | enginesize | 3.01 |
| 1 | carwidth | 2.68 |
| 4 | enginetype_rotor | 1.12 |
| 0 | const | 1.10 |
| 5 | car_company_bmw | 1.09 |
| 3 | enginelocation_rear | 1.08 |
| 6 | car_company_renault | 1.00 |

- All the VIF values(<5) and p-values(0.05) seem to be in a good range.
- Also the `Adjusted R-squared` value has dropped from `0.91` with **15 variables** to just `0.89` using **6 variables**.
- This model is explaining most of the variance without being too complex.

- Now the model is in good format.

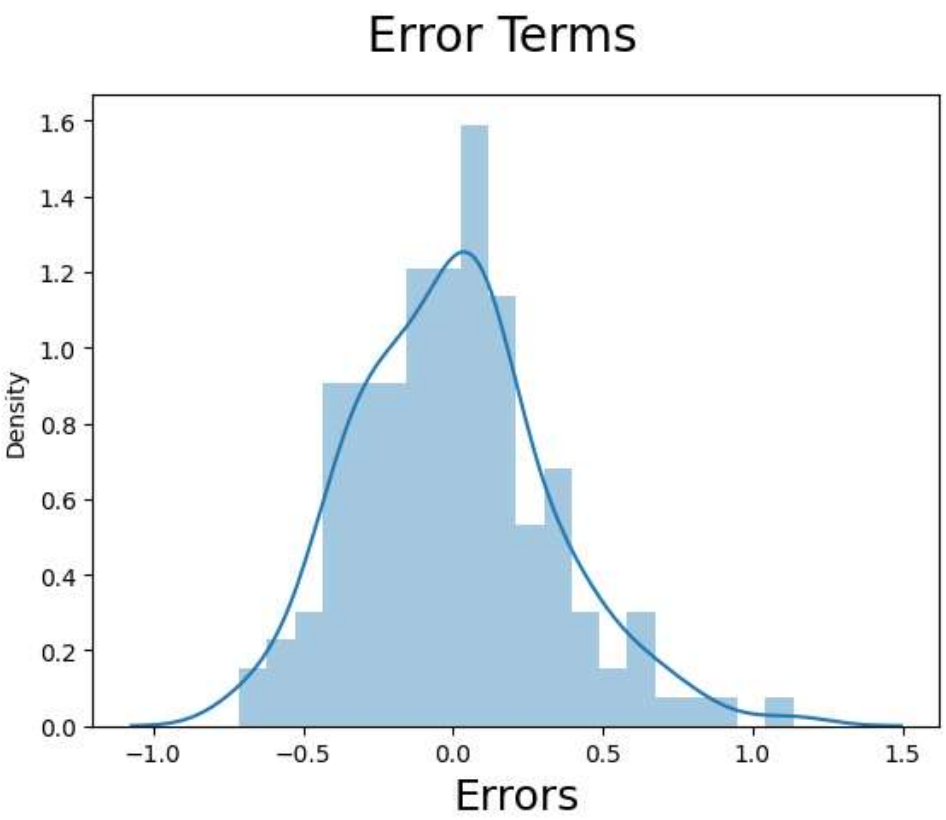
## Residual Analysis

Before we make predictions on the test set, we need to analyse the residuals.

In [65]:
```
y_train_price = lm2.predict(X_train_rfe2)
```

In [66]:
```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                         # X-label
```

Out[66]:
```
Text(0.5, 0, 'Errors')
```




- The error terms are fairly normally distributed and the mean of the error terms is zero(approx).
- The assumptions of the linear model is satisfied here.
- Let's make the predictions on the test set.

## Making Predictions

We would first need to scale the test set as well using only transform but not fit_transform so that the model will not learn about the data.

In [67]:
```
df_test[varlist] = scaler.transform(df_test[varlist])
```

In [68]:
```
# Splitting the 'df_test' set into X_test and y_test
y_test = df_test.pop('price')
X_test = df_test
```

In [69]:
```
# Checking the list 'col2' which had the 10 variables RFE had selected
col2
```

Out[69]:
```
Index(['carwidth', 'enginesize', 'enginelocation_rear', 'enginetype_l',
       'enginetype_ohcf', 'enginetype_rotor', 'car_company_bmw',
       'car_company_peugeot', 'car_company_renault', 'car_company_subaru'],
      dtype='object')
```

In [70]:
```
# Subsetting these columns and create a new dataframe 'X_test_rfe2'
X_test_rfe2 = X_test[col2]
```

In [71]:
```
# Dropping the variables that we had manually eliminated as well previously while maintaining a g
X_test_rfe2 = X_test_rfe2.drop(['enginetype_ohcf', 'car_company_peugeot', 'enginetype_l', 'car_co
```

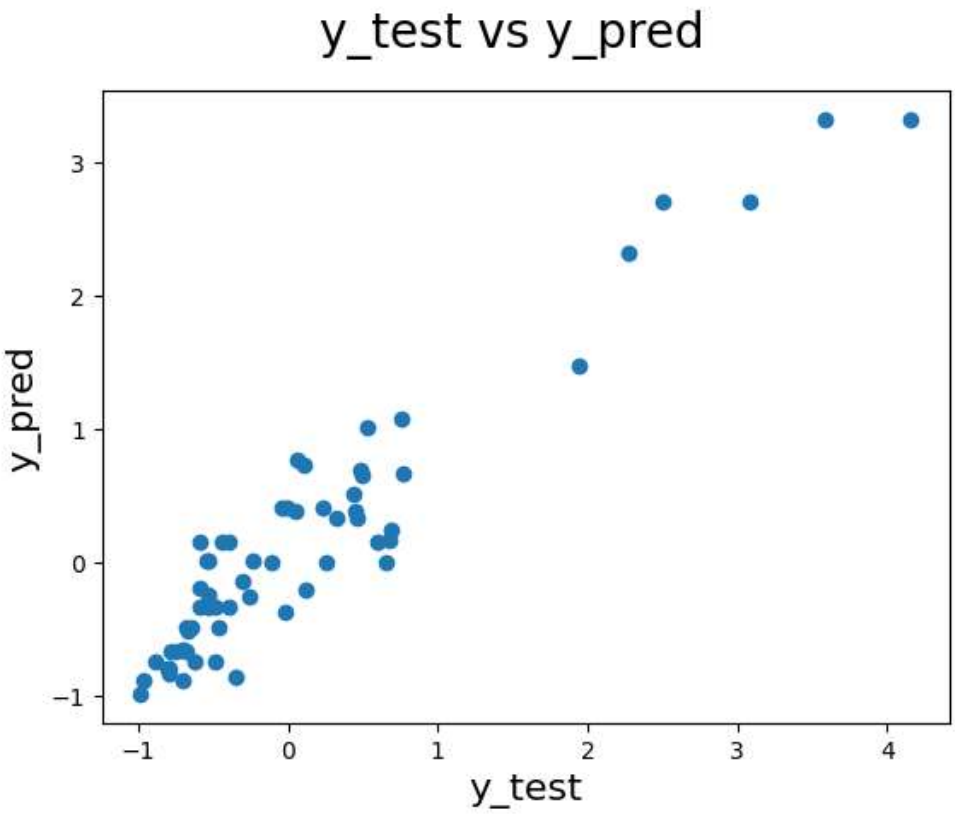
In [72]:
```
# Addind a constant to the test set created
X_test_rfe2 = sm.add_constant(X_test_rfe2)
X_test_rfe2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 62 entries, 160 to 128
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   const                62 non-null     float64
 1   carwidth             62 non-null     float64
 2   enginesize           62 non-null     float64
 3   enginelocation_rear  62 non-null     uint8  
 4   enginetype_rotor     62 non-null     uint8  
 5   car_company_bmw      62 non-null     uint8  
 6   car_company_renault  62 non-null     uint8  
dtypes: float64(3), uint8(4)
memory usage: 2.2 KB
```

In [73]:
```
# Making predictions
y_pred = lm2.predict(X_test_rfe2)
```

In [74]:
```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(y_test, y_pred)
fig.suptitle('y_test vs y_pred', fontsize = 20)              # Plot heading
plt.xlabel('y_test', fontsize = 16)                          # X-label
plt.ylabel('y_pred', fontsize = 16)
```

Out[74]:
```
Text(0, 0.5, 'y_pred')
```




- From the above plot, we can find that the model is doing well on the test set as well.
- Let's check the R-squared and more importantly, the adjusted R-squared value for the test set.

In [75]:
```
# r2_score for 6 variables between the actual y_test and predicted y_pred.
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

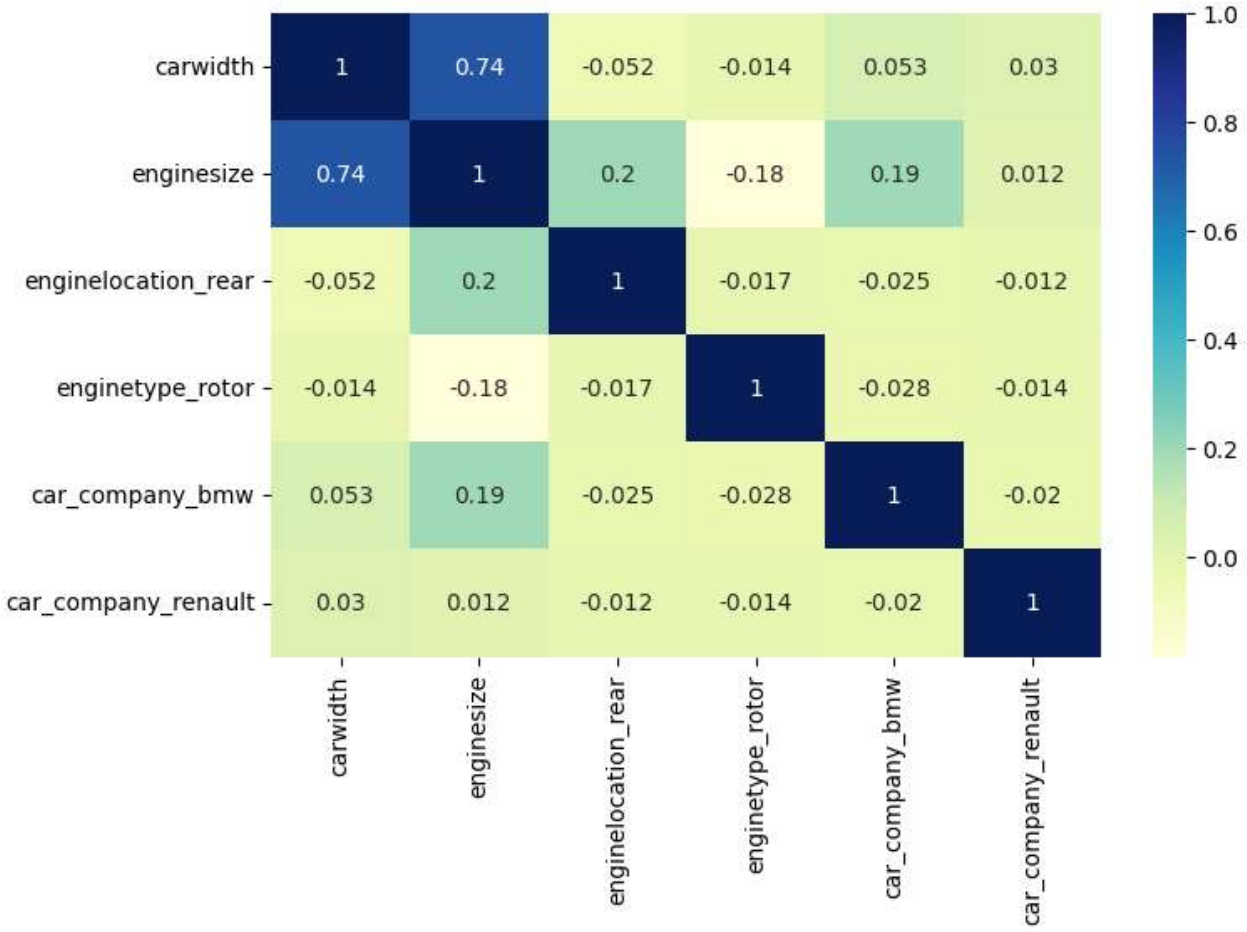
Out[75]:
```
0.8997211435182687
```

- Thus, for the model with 6 variables, the r-squared on training and test data is about 89.6% and 89.9% respectively.
- The adjusted r-squared on the train set is about is about 89.1%.
- So the predictions by the model are good.

**Checking the correlations between the final predictor variables**

In [76]:
```
col2 = col2.drop(['enginetype_ohcf', 'car_company_peugeot', 'enginetype_l', 'car_company_subaru']
```

In [77]:
```
# Figure size
plt.figure(figsize=(8,5))

# Heatmap
sns.heatmap(cars[col2].corr(), cmap="YlGnBu", annot=True)
plt.show()
```




- Thus, the model consists of the 6 variables mentioned above.
- But again we can see that there is high positive correlation between the features i.e., `carwidth` , `enginesize`
- So previously enginesize VIF is about `3.01` so remove and rebuild the model with 5 features

In [84]:
```
# Dropping the enginesize feature and again building teh model with 5 features.
X_train_rfe2.drop('enginesize', axis = 1, inplace = True)

# Adding a constant
X_train_rfe3 = sm.add_constant(X_train_rfe2)

# Fitting the model with 5 variables
lm2 = sm.OLS(y_train, X_train_rfe3).fit()
print(lm2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.806
Model:                            OLS   Adj. R-squared:                  0.799
Method:                 Least Squares   F-statistic:                     113.8
Date:                Mon, 13 Feb 2023   Prob (F-statistic):           5.36e-47
Time:                        22:28:06   Log-Likelihood:                -85.682
No. Observations:                 143   AIC:                             183.4
Df Residuals:                     137   BIC:                             201.1
Df Model:                           5
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                   -0.0818      0.039     -2.073      0.040      -0.160      -0.004
carwidth                 0.7856      0.038     20.778      0.000       0.711       0.860
enginelocation_rear      3.0765      0.452      6.806      0.000       2.183       3.970
enginetype_rotor         0.1269      0.228      0.556      0.579      -0.325       0.579
car_company_bmw          1.5581      0.189      8.264      0.000       1.185       1.931
car_company_renault     -0.6159      0.321     -1.919      0.057      -1.251       0.019
==============================================================================
Omnibus:                       29.821   Durbin-Watson:                   2.085
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               58.703
Skew:                           0.922   Prob(JB):                     1.79e-13
Kurtosis:                       5.541   Cond. No.                         12.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [85]:
```
# Dropping the const variable in the train set
# X_train_rfe2.drop('const',axis=1,inplace=True)
# Create a dataframe that will contain the names of all the feature variables and their respective
vif = pd.DataFrame()
vif['Features'] = X_train_rfe3.columns
vif['VIF'] = [variance_inflation_factor(X_train_rfe3.values, i) for i in range(X_train_rfe3.shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[85]:

| | Features | VIF |
|---|---|---|
| 0 | const | 1.10 |
| 1 | carwidth | 1.01 |
| 4 | car_company_bmw | 1.01 |
| 2 | enginelocation_rear | 1.00 |
| 3 | enginetype_rotor | 1.00 |
| 5 | car_company_renault | 1.00 |

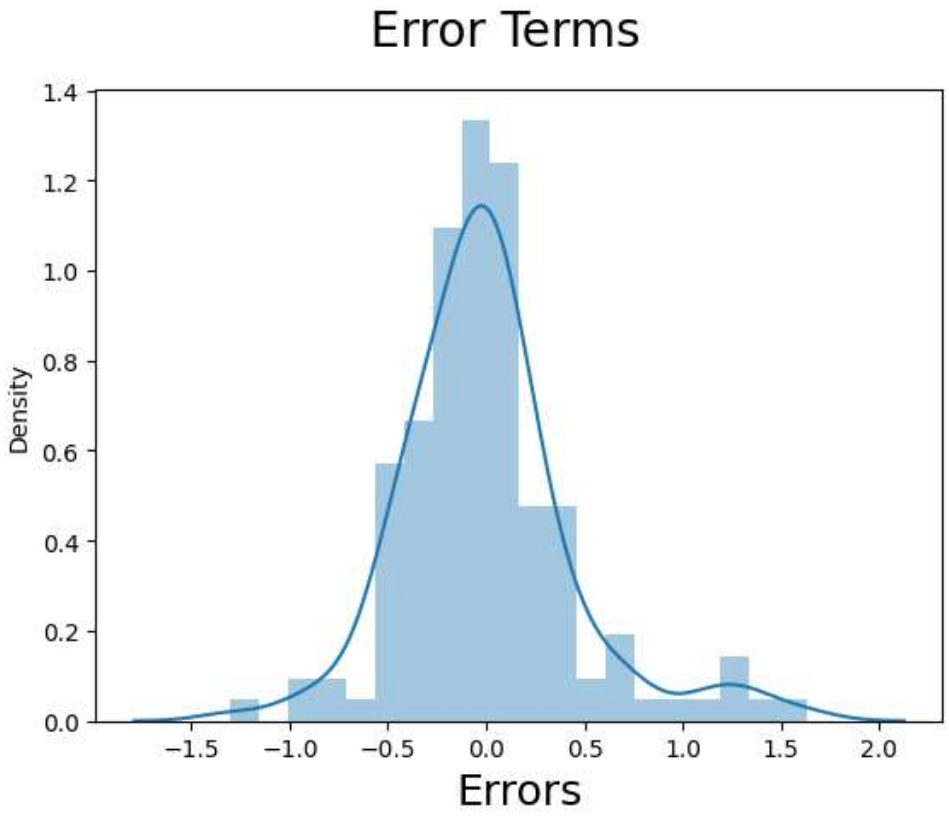
## Residual Analysis

Before we make predictions on the test set, we need to analyse the residuals using the 5 features.

In [87]:
```
y_train_price = lm2.predict(X_train_rfe3)
```

In [88]:
```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                         # X-label
```

Out[88]:
```
Text(0.5, 0, 'Errors')
```




- The error terms are fairly normally distributed and the mean of the error terms is zero(approx).
- The assumptions of the linear model is satisfied here.
- Let's make the predictions on the test set.

## Making Predictions

We would first need to scale the test set as well using only transform but not fit_transform so that the model will not learn about the data.

In [92]:
```
# Checking the list 'col2' which has 5 features.
col2
```

Out[92]:
```
Index(['carwidth', 'enginelocation_rear', 'enginetype_rotor',
       'car_company_bmw', 'car_company_renault'],
      dtype='object')
```

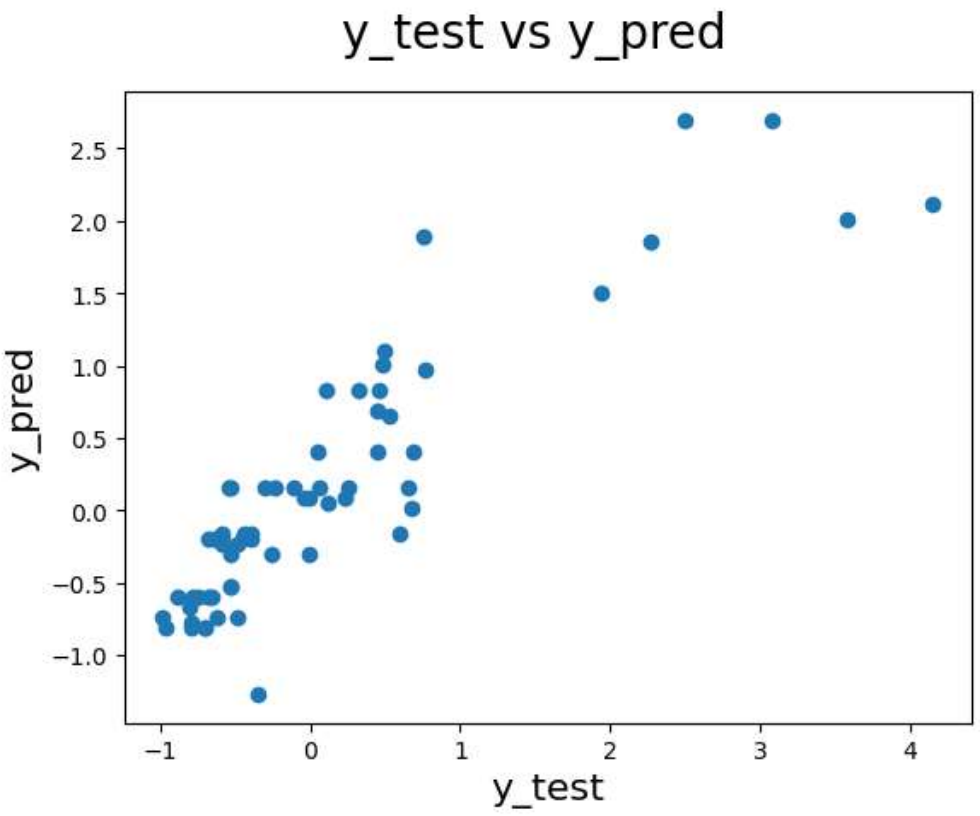
In [95]:
```
X_test_rfe3 = X_test[col2]
# Addind a constant to the test set created
X_test_rfe3 = sm.add_constant(X_test_rfe3)
X_test_rfe3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 62 entries, 160 to 128
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   const                62 non-null     float64
 1   carwidth             62 non-null     float64
 2   enginelocation_rear  62 non-null     uint8  
 3   enginetype_rotor     62 non-null     uint8  
 4   car_company_bmw      62 non-null     uint8  
 5   car_company_renault  62 non-null     uint8  
dtypes: float64(2), uint8(4)
memory usage: 1.7 KB
```

In [96]:
```
# Making predictions
y_pred = lm2.predict(X_test_rfe3)
```

In [97]:
```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(y_test, y_pred)
fig.suptitle('y_test vs y_pred', fontsize = 20)              # Plot heading
plt.xlabel('y_test', fontsize = 16)                          # X-label
plt.ylabel('y_pred', fontsize = 16)
```

Out[97]: Text(0, 0.5, 'y_pred')



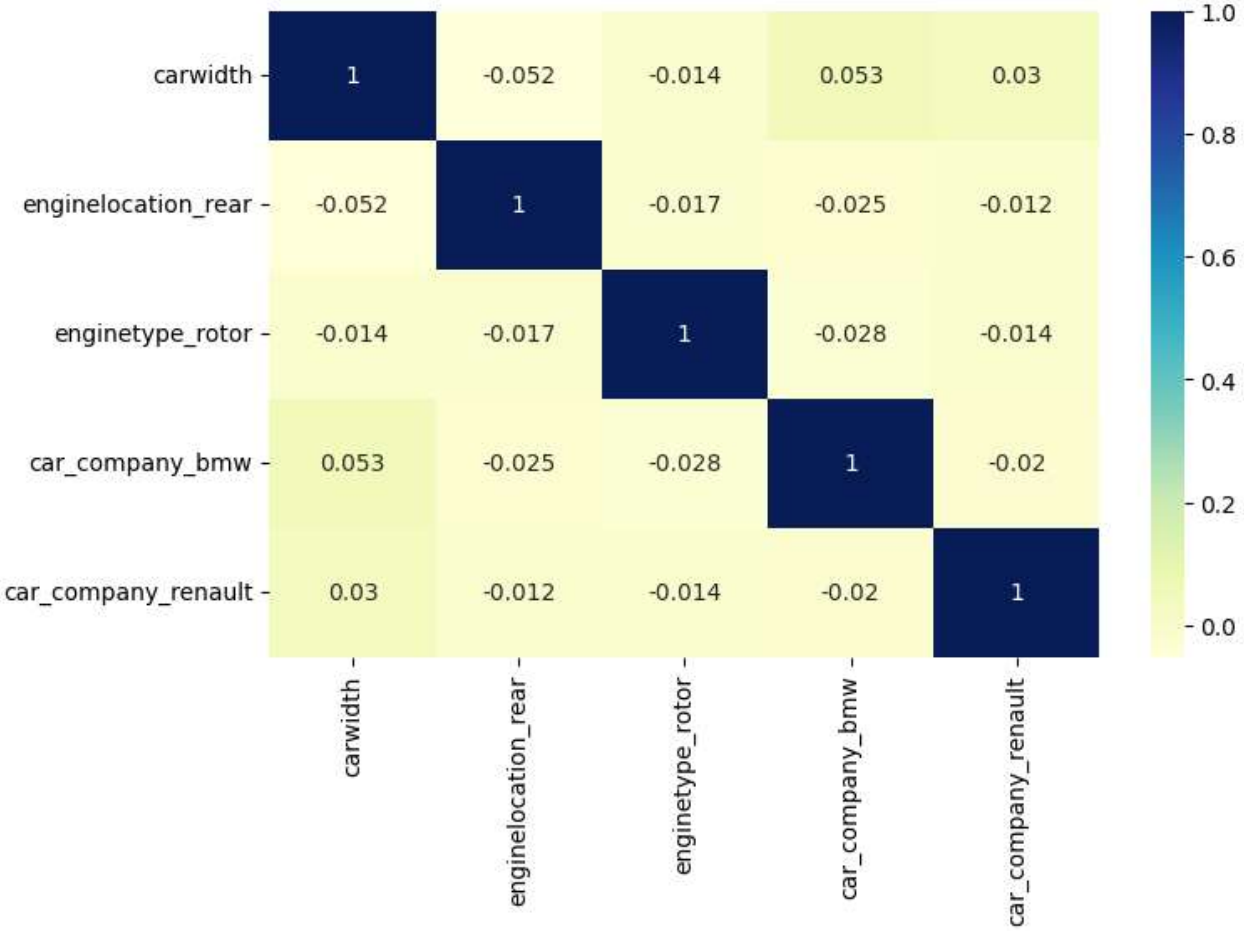
- From the above plot, we can find that the model is doing well on the test set as well.
- Let's check the R-squared and more importantly, the adjusted R-squared value for the test set.

In [98]:
```
# r2_score for 6 variables between the actual y_test and predicted y_pred.
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[98]:
```
0.7817514683111357
```

- Thus, for the model with 5 variables, the r-squared on training and test data is about 80% and 78% respectively.
- The adjusted r-squared on the train set is about is about 79.9%%.
- So the predictions by the model are good.

In [99]:
```
# Figure size
plt.figure(figsize=(8,5))

# Heatmap
sns.heatmap(cars[col2].corr(), cmap="YlGnBu", annot=True)
plt.show()
```




- So, this is the final model that has with 5 variables as shown above.
- Now, the correlation between the independent is decreased.